**ĐẠI HỌC QUỐC GIA HÀ NỘI**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

****

**TIỂU LUẬN KẾT THÚC MÔN HỌC**

**MẬT MÃ VÀ AN TOÀN THÔNG TIN**

**Đề tài : Thuật toán và chương trình đếm số điểm trên đường cong Elliptic với số nguyên tố p lớn hơn 160 bit.**

GIẢNG VIÊN: LÊ PHÊ ĐÔ

20021392

20020027

SINH VIÊN: LƯU ĐẠT TUẤN MINH

NGUYỄN ĐẮC QUÁN

Hà Nội – 11/2022

**MỤC LỤC**

[DANH MỤC HÌNH VẼ 3](#_bookmark0)

[MỞ ĐẦU 4](#_bookmark1)

[CHƯƠNG 1: TỔNG QUAN VỀ ĐƯỜNG CONG ELLIPTIC( *Fp*) 5](#_bookmark2)

* 1. [Định nghĩa, tính chất của đường cong 5](#_bookmark3)
     1. [Định nghĩa 5](#_bookmark4)
     2. [Tính chất 5](#_bookmark4)

[CHƯƠNG 2: THUẬT TOÁN ĐẾM SỐ ĐIỂM TRÊN ĐƯỜNG CONG ELLIPTIC 6](#_bookmark5)

* 1. [Thuật toán liên quan 6](#_bookmark6)
  2. [Phương pháp cơ bản (Naive approach) 6](#_bookmark6)
  3. Thuật toán Shanks-Mestre 8
  4. Thuật toán Schoof 8
  5. Thuật toán SEA (Schoof-Elkies-Atkins) 9

[CHƯƠNG 3: CÁC CHƯƠNG TRÌNH TÍNH TOÁN 10](#_bookmark11)

* 1. [Pari/GP 10](#_bookmark12)
  2. [Magma 12](#_bookmark13)
  3. SageMath 13

[CHƯƠNG 4: XÂY DỰNG CHƯƠNG TRÌNH 10](#_bookmark11)

* 1. [Nhiệm vụ của chương trình 14](#_bookmark15)
     1. [Tại sao dùng Pari/GP? 15](#_bookmark16)
     2. [Gọi thư viện Pari/GP trong Python 15](#_bookmark16)
     3. [Chạy chương trình trên các ngôn ngữ khác 15](#_bookmark17)
  2. [Kết quả của chương trình (Nền tảng .NET) 16](#_bookmark18)
     1. [Giao diện 17](#_bookmark19)
     2. Các bước thực hiện chương trình 17
     3. Kết quả thực nghiệm [22](#_bookmark21)

[KẾT LUẬN 26](#_bookmark23)

[TÀI LIỆU THAM KHẢO 26](#_bookmark24)

## DANH MỤC HÌNH VẼ

[Trường đồ thị của đường cong Elliptic 5](#_bookmark23)

[Ví dụ về sea data package 13](#_bookmark23)

# MỞ ĐẦU

Mật mã học ngày càng đóng một vai trò quan trọng hơn trong xã hội của chúng ta: thẻ thông minh, thanh toán qua Internet, ngân hàng trực tuyến. . . . Tất cả các ứng dụng này cần phải được bảo vệ thông tin.

Đường cong Elliptic được ứng dụng nhiều trong mật mã học và công nghệ thông tin. Hệ mật đường cong Elliptic được giới thiệu bởi Victor Miller (Miller, 1986) và Neal Koblitz (Koblitz, 1987) vào năm 1985. Theo tiến độ thời gian và nghiên cứu và phát triển chuyên sâu, hệ mật trên đường cong Elliptic hiện đang được triển khai rộng rãi. Hệ mật cung cấp khóa có kích thước nhỏ hơn, tiết kiệm băng thông và triển khai nhanh hơn so với RSA(Rivest-Shamir-Adleman). Tính chất thú vị nhất của các đường cong Elliptic là cấu trúc nhóm của các điểm được tạo bởi đường cong, trong đó các điểm trên các đường cong Elliptic tạo thành một nhóm. Tính bảo mật của hệ mật đường cong Elliptic dựa trên độ khó của việc giải bài toán logarit rời rạc trên đường cong Elliptic.

Bài toán logarit rời rạc trên đường cong Elliptic tương tự như bài toán logarit rời rạc đại số thông thường, *l = gx*, cho trước *l* và *g*, không thể tính được *x*. Bài toán logarit rời rạc trên đường cong Elliptic đề cập đến việc giải *n* trong quan hệ *P = nG.* Cho điểm P và điểm G, rất khó giải được số nguyên *n*. Để triển khai bài toán logarit rời rạc trong hệ mật trên đường cong Elliptic, nhiệm vụ quan trọng nhất là tính số điểm trên đường cong. Một số thuật toán đếm điểm đã được phát minh để thực hiện nhiệm vụ này.

Bài tiểu luận tập trung vào thuật toán và chương trình đếm số điểm trên đường cong Elliptic. Nội dung nghiên cứu được chia làm 4 phần:

##### Tổng quan về đường cong Elliptic

* **Thuật toán đếm số điểm trên đường cong Elliptic**

##### Các chương trình tính toán

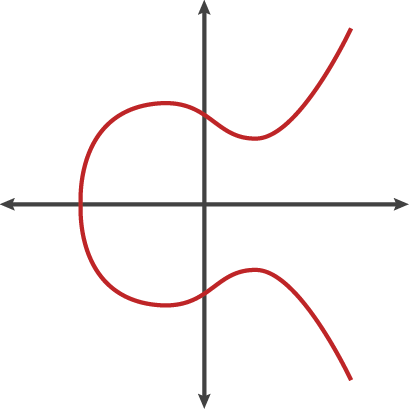
* **Xây dựng chương trình sử dụng thuật toán**

## CHƯƠNG 1: TỔNG QUAN VỀ ĐƯỜNG CONG ELLIPTIC E(*F p*)

### Định nghĩa, tính chất của đường cong

#### Định nghĩa

Thông thường, một đường cong Elliptic được định nghĩa bằng phương trình có dạng như sau: *y*2=*x*3+*ax* +*b* [1] và có điều kiện là: 4 *a*3+27 *b*2 ≠ 0



Hình 1: Trường đồ thị của đường cong Elliptic

#### Tính chất

Nếu hai điểm P1 (x1, y1) và P2 (x2, y2) với x1 ≠ x2 nằm trên đường cùng một đường cong Elliptic E, thì đường thẳng qua hai điểm P1 và P2 sẽ cắt một điểm duy nhất P3 ( x3, y3) có thể xác định thông qua P1 và P2 nằm trên đường cong E.

Tiếp tuyến của đường cong tại điểm bất kỳ P(x, y) trên đường cong E cũng cắt đường cong Elliptic E tại một điểm duy nhất nằm trên đường E, điểm này cũng có thể xác định được thông qua P.

## CHƯƠNG 2: THUẬT TOÁN ĐẾM SỐ ĐIỂM TRÊN ĐƯỜNG CONG ELLIPTIC

## 2.0 Các phép biến đổi điểm và thuật toán liên quan đến đường cong Elliptic

## Để chuẩn bị các kiến thức nền nhằm xây dựng thuật toán, chúng em sẽ đưa ra một số định lý, thuật toán thông dụng sau đây [2]

## Định lý Hasse: số điểm trên đường cong Elipstic trên Z/Zp sẽ thỏa mãn

## |#E(Fp) − (p + 1)| ≤ 2 √ p

## Ký hiệu legrendre: Nếu p là số nguyên tố và a ko chia hết cho p thì:

## Tồn tại b: b2 = a (mod p) <=> a(p-1)/2 = 1(mod p)

## Thuật toán Toneli- Shanks về tìm căn bậc 2 module p trong thời gian O(log(p)\*M(p)) với M(p) là độ phức tạp để nhân 2 số < p

## Định lý thặng dư trung hoa để xây dựng nghiệm của hệ phương

## trình đồng dư với các module đôi 1 nguyên tố cùng nhau

## Định lý lagrange về nhóm: nếu (G, \*) là 1 nhóm thì với mọi a thuộc G:

## a|G| = id với id là phần tử đơn vị của G

## 2.1 Phương đếm pháp cơ bản (Naive approach)

Ta sẽ duyệt qua toàn bộ x từ 0 đến p – 1, với mỗi x ta sẽ dùng ký hiệu legendre để kiểm tra xem tồn tại y để (x, y) là 1 điểm thuộc E hay không, độ phức tạp là ~O(p). Vốn dĩ với các phần tử thặng dư bậc 2 mod p trên đường cong Elliptic, sẽ tồn tại 2 điểm (x1, y1) và (x1, p - y1) nên ta thêm 1 đơn vị vào p. Với các phần tử không là thặng dư bậc 2, sẽ không có điểm nào và phải bớt 1 đơn vị khỏi p. Và với mỗi x trên Fp mà phần tử đó chia hết cho p thì chỉ tồn tại duy nhất 1 điểm trên đường cong (x, 0).

public BigInteger L(BigInteger n, BigInteger k) //Legendre symbol

{

var a = BigInteger.ModPow(n, (k - 1) / 2, k);

if (a == k - 1) return -1;

return a;

}

public BigInteger Order(BigInteger a, BigInteger b, BigInteger p) //time complexity O(p)

{

int count = 1;

for (var x = 0; x < p; x++)

{

var y = (BigInteger.Pow(x, 3) + a \* x + b) % p;

count += L(y, p);

}

count += p;

return count;

}

#### 2. 2 Phương pháp Baby step – giant step

#### Ý tưởng của phương pháp này là áp dụng định lý lagrange, nếu ta có 1 điểm P(x,y) thuộc E thì #E \* P = id (tức điểm vô cùng).

#### Từ đó, ta sẽ cố gắng random 1 điểm P thuộc E và tìm số N nào đó thuộc khoảng

#### [0, 2 sqrt(p) + 2] thỏa mãn (p – sqrt(p) + N) \* P = id bằng phương pháp babystep-giantstep với

#### độ phức tạp khoảng O(p^¼) từ đó N có khả năng trở thành đáp án ta cần tìm

#### Pseudo code:

#### Int order(Point P, EllipticCurve E) { //trả về cấp của P trên E

#### Int m = sqrt(4 \* sqrt(p)), count = 0, maybe\_ord;

#### HashTable baby\_step

#### for(i từ 1 đến m) {

#### If(-i\*P == id) return i // nếu cấp của P <= m thì hàm sẽ trả về sớm

#### baby\_step[-i\*P] = i;

#### }

#### // đến đây ta đảm bảo cấp của P phải > m nên P, 2P, ... mP đôi 1 phân biệt

#### for(i từ 1 đến m) {

#### Int giant\_step = p – sqrt(p) + 1 + m \* i

#### If (giant\_step \* P thuộc baby\_step) { // tức giant\_step\*P = -j\*P với j < m

#### if(count == 0) count = 1, maybe\_ord = giant\_step + baby\_step[giant\_step\*P]

#### else {

#### ord = giant\_step + baby\_step[giant\_step\*P] - ord; //đảm bảo < 4sqrt(p)

#### List primes = prime\_factor(ord) // chạy trong sqrt(ord)

#### while(tồn tại p thuộc primes để (ord/p)\*P == id) ord /= p

#### return ord // nếu có 2 số thuộc khoảng thì trả về tại đây

#### }

#### }

#### }

#### return maybe\_ord; // có duy nhất 1 thỏa mãn nên số đó chính là #E

#### }

#### Int cardinatary(EllipticCurve E) {

#### Range hasse\_bound = (p – 2sqrt(p) +1, p + 2sqrt(p) + 1)

#### Int result = 1

#### While(hasse\_bound.not\_contain(result)) {

#### result = lcm(result, ord(random\_point(E), E))

#### // do #E chia hết cho mọi cấp và tỷ lệ chọn được cấp to là rất cao

#### }

#### Return result

#### }

#### 

#### 

#### 2.3 Thuật toán Schoof

#### Để tính được #E mod q, ta sẽ sử dụng 1 tính chất của đa thức division polynomial với division polynomial thứ k (ký hiệu Psi\_k)

#### Do với mọi điểm (x,y) ta đều có

#### (x^p^2, y^p^2) + p(x, y) = t(x^p, y^p) trong đó t = p + 1 - #E (\*)

#### Ý tưởng của thuật toán schoof là ta sẽ sử dụng định lý thặng dư trung hoa để tìm được #E mod q với 1 q là 1 số nguyên tố đủ nhỏ và để tính được #E mod q ta sẽ xét phương trình đồng dư đa thức (\*) trên nhóm con E[q] được định nghĩa bởi

#### E[q] = { P thuộc E | q\*E = id} (để ý đa thức đặc trưng của nhóm con này là Psi\_q)

#### Từ đó thay vì tìm t thỏa mãn (\*) trong Zp[x,y]/(x^3+ax+b-y^2) ta sẽ giảm được về bài toán tìm t thuộc (0, ... q-1) để phương trình (\*) thỏa mãn với Zp[x,y]/(x^3+ax+b-y^2, Psi\_q)

#### Để ý là khi q lẻ thì Psi\_q là 1 đa thức theo x và có công thức truy hồi tường minh và bậc của Psi\_q khi đó cùng lắm là (q^2-1)/3 và mọi đa thức khi áp dụng với phép cộng điểm và nhân điểm luôn là 1 đa thức theo x hoặc y nhân 1 đa thức theo x

#### Psuedo code:

#### Int cardinatary(EllipticCurve E) {

#### // dãy 1 vài số nguyên tố lẻ đầu tiên mà tích > 4 sqrt(p)

#### List small\_primes = get\_odd\_prime(4sqrt(p))

#### // dãy các số số dư #E khi chia cho q với q thuộc small\_primes

#### List remainder

#### // các division polynomial thứ 1 đến max(small\_primes)

#### PolyList div\_poly = calculate\_div\_poly(max(small\_primes))

#### for(q nguyên tố thuộc small\_primes) {

#### Int p1 = p % q // |p1| < q/2

#### // lhs là vế trái của phương trình (\*) sau khi lấy mod div\_poly[q]

#### // các phép cộng, nhân điểm ở trên đều theo nghĩa đa thức

#### Poly lhs = (((x^p^2, y^p^2) + p1\*(x,y)) % div\_poly[q]).x // hoành độ

#### if( lhs != 0) {

#### for(i từ 1 đến (q – 1)/2) { // brute force t mod q

#### Poly rhs = (j \* (x, y) % div\_poly[q]).x

#### if(rhs.x^q % div\_poly[q] == {x}) {

#### if((rhs.y - {y})/ {y}) % div\_poly[q] == 0) remainder[q] = -i

#### else remainder[q] = i

#### }

#### }

#### }

#### else if(pow\_mod(p, (q-1)/2, q)==1) { //rightside = 0 thì phải xử lý riêng

#### Int w = mod\_sqrt(q, p) // sử dụng thuật toán Toneli-Shanks

#### Poly rhs = w(x, y) % poly[q]

#### if( gcd(x^p – rhs.x, poly[q]) == 1) remainder[q] = 0

#### else if(gcd(y^p – rhs.y, poly[q]) == 1) remainder[q] = 2\*w

#### else remainder[q] = -2\*w

#### }

#### else {

#### remainder[q] = 0

#### }

#### }

#### return chinese\_remainder(small\_primes, remainder);

#### }

#### 

#### Độ phức tạp thuật toán:

#### 

#### Với mỗi số nguyên tố q < O(log p), ta sẽ thực hiện q phép thử, mỗi phép thử bắt ta phải tính q(x, y) và (x^p^2, y^p^2) mod div\_poly[q], để ý bậc của div\_poly[q] cùng lắm là O(q^2) và dùng thuật toán nhân lũy thừa (double and add) ta cần ít nhất O(log(p)) lần nhân đa thức và lấy mod,

#### => độ phức tạp sẽ là O(log(p)^3 \* M(log(p)^2, p)) với M(n, p) là độ phức tạp của phép nhân đa thức bậc n trên Zp

#### => nếu sử dụng phương pháp nhân truyền thống có thể lên tới O(log^8(p)) nhưng ta có thể tối ưu hóa bằng phép nhân FFT để giảm độ phức tạp xuống O(log^5(p))

#### 2.4 Thuật toán SEA (Schoof–Elkies–Atkin)

#### Thuật toán SEA là 1 biến thể cải tiến của thuật Schoof, cụ thể thay vì chọn O(log) số nguyên tố đầu tiên, ta xét các số nguyên tố mà phương trình (\*) có 2 nghiệm trong Fq hay không (nếu có ta gọi q là 1 số nguyên tố Elkies), khi đó sẽ xét 1 đa thức bậc thấp hơn (cụ thể là O(q) thay vì O(q^2) so với division polynomial) dẫn đến độ phức tạp giảm xuống O(log^3\*M(log, p) ~ O(log^4(p))

#### 

Psuedo code:

Bool is\_elkise\_prime(int q, EllipticCurve E) {

Int j = j\_invariant(E) // = 1728(4a^3)/(4a^3 + 9b^2)

Return gcd(x^p –x, division\_poly(E, q, j) == 1 // là division polynomial bậc q với y = j

}

Int cardinatary(EllipticCurve E) {

... phần đầu giống thuật toán schoof, ta vẫn sẽ khởi tạo mảng gồm các số nguyên tố nhỏ và division polynomial

if(is\_elkies\_prime(q, E)) {

Poly F = kernel\_isogeny(E, q) // F là ước của div\_poly[q] deg(F) và O(q)

... tương tự như schoof nhưng thay vì lấy mod div\_poly[q] ta lấy mod F

}

else {

Continue // ta sẽ bỏ qua các số nguyên tố như vậy vì nếu không mất O(q^2)

// thuật toán sẽ hoạt động tốt nếu nhiều số nguyên tố là Elkies prime

// thực nghiệm cho thấy tỷ lệ có 1 số nguyên tố như vậy là khá cao

}

.... lấy thặng dư trung hoa như thuật toán Schoof

}

#### CHƯƠNG 3: CÁC CHƯƠNG TRÌNH TÍNH TOÁN

### Pari/GP

PARI/GP là một hệ thống đại số máy tính với mục đích chính là hỗ trợ tính toán lý thuyết số. Ba điểm mạnh của nó là tốc độ, khả năng sử dụng trực tiếp các kiểu dữ liệu quen thuộc trong toán học và mô-đun lý thuyết đại số mở rộng.

Hệ thống PARI/GP bao gồm các thành phần tiêu chuẩn sau:

* PARI là một thư viện C, cho phép tính toán nhanh và có thể được gọi từ một ứng dụng ngôn ngữ bậc cao (ví dụ: được viết bằng C, C++, Pascal, Fortran, Perl hoặc Python).
* gp là một giao diện dòng lệnh tương tác dễ sử dụng cho phép truy cập vào các chức năng PARI. Nó hoạt động như một máy tính lập trình phức tạp chứa hầu hết các lệnh điều khiển của một ngôn ngữ chuẩn như C. GP là tên của ngôn ngữ kịch bản của gp có thể được sử dụng để lập trình gp.

Cũng có sẵn gp2c, trình biên dịch GP-to-C, biên dịch các tập lệnh GP sang ngôn ngữ C và tải rõ ràng các chức năng kết quả vào gp. Ưu điểm của điều này là các tập lệnh được biên dịch bằng gp2c thường sẽ chạy nhanh hơn từ ba đến bốn lần. gp2c hiểu gần như tất cả GP [3].

PARI/GP thực hiện các phép tính chính xác tùy ý (ví dụ: phần trị có thể dài hàng triệu chữ số hoặc hàng tỷ chữ số trên máy 64-bit). Nó có thể tính toán thừa số, thực hiện tính toán đường cong Elliptic và các phép tính lý thuyết đại số. Nó cũng cho phép tính toán với ma trận, đa thức, chuỗi lũy thừa, đại số và nhiều hàm đặc biệt.

PARI/GP đi kèm với khả năng vẽ đồ họa tích hợp riêng. PARI/GP có một số khả năng thao tác tượng trưng, ​​ví dụ, xử lý hàm phân thức và đa thức đa biến. Nó cũng có một số khả năng tích hợp và lấy tích phân.

PARI/GP có thể được biên dịch với GMP (GNU Multiple Precision Arithmetic Library) cung cấp khả năng tính toán nhanh hơn so với nhân hệ điều hành đo độ chính xác tùy ý gốc của PARI/GP.

Syntax để tính số điểm trên đường cong Elliptic trong Pari/GP

ellinit([a, b]\*Mod(t, p)).no

E = ellinit([a, b]\*Mod(t, p)); ellcard(E);

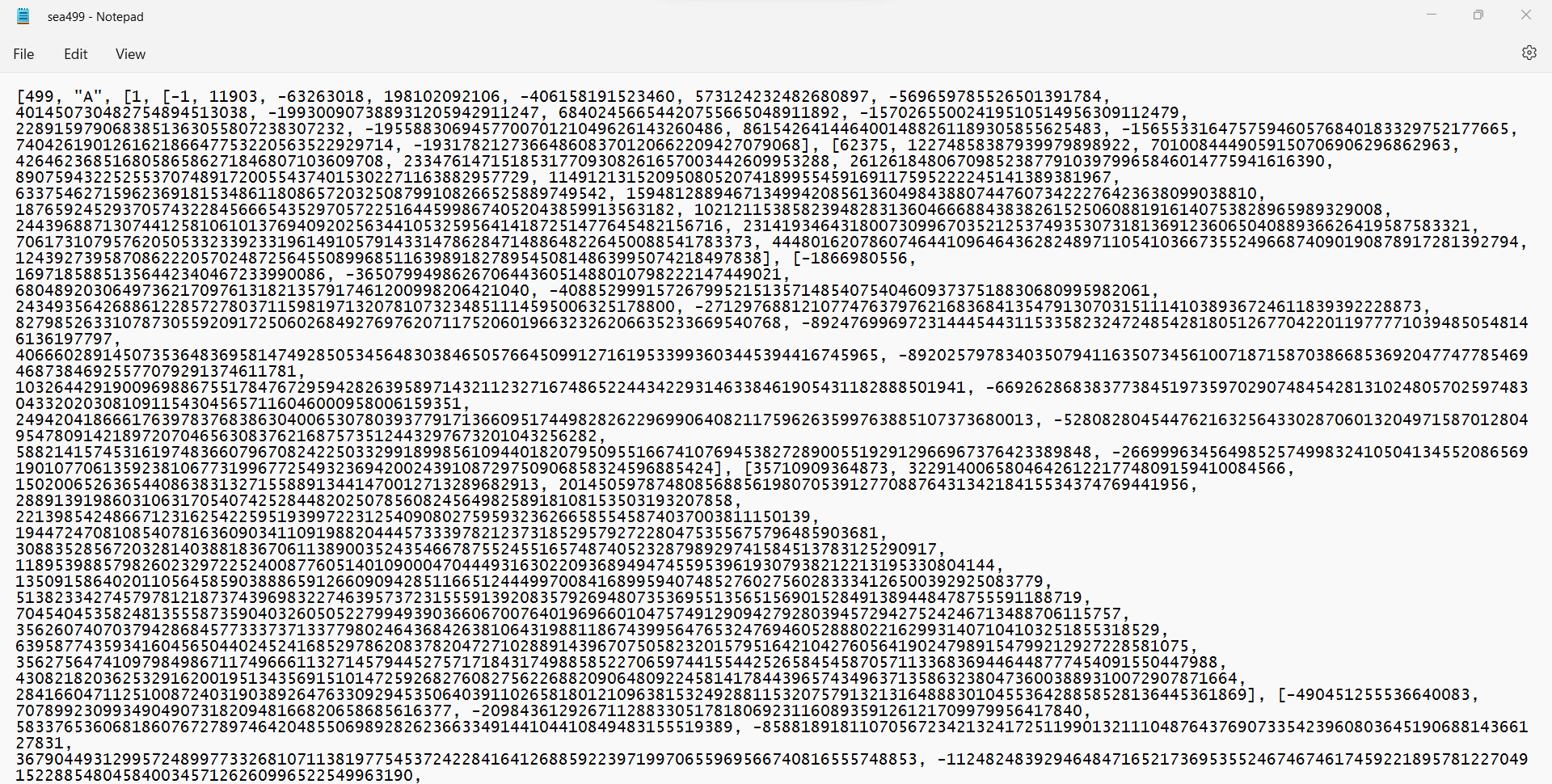
//t là bậc của p (lấy t = 1)

PARI/GP sử dụng thuật toán SEA để đếm số điểm trên đường cong Elliptic. Trong đó, thư viện ngôn ngữ C PARI dùng hàm ellcard. [4]

GEN ellcard (GEN E, {p}): given an Elliptic curve E defined over a finite field Fq, return the order of the group E(𝔽q); for other fields of definition K, p must define a finite residue field, (p prime for K = Qp or Q; p a maximal ideal for K a number field), return the order of the (nonsingular) reduction of E.

Doc: Let E be an ell structure as output by ellinit function, attached to an Elliptic curve E/K. If K = 𝔽q is finite, return the order of the group E(𝔽q).For other fields of definition and p defining a finite residue field 𝔽q, return the order of the reduction of E: the argument p is best left omitted if K = ℚ\_ℓ (else we must have p = ℓ) and must be a prime number (K = ℚ) or prime ideal (K a general number field) with residue field 𝔽q otherwise. The equation need not be minimal or even integral at p; of course, a minimal model will be more efficient. The function considers the group of nonsingular points of the reduction of a minimal model of the curve at p, so also makes sense when the curve has bad reduction.

When the characteristic of the finite field is large, the availability of the seadata package will speed the computation.



File sea499 with upto 262-bit Atkin prime

##### Magma

Magma chứa phương pháp thực hiện hiệu quả thuật toán SEA, với thuật toán mở rộng của Lercier cho các trường cơ sở đặc số 2, để tính số điểm trên đường cong Elliptic trên một trường hữu hạn. Ngoài ra còn có các phép nâng chính tắc hệ số p-adic nhanh hơn và phép chỉnh hình p-adic với đặc số nhỏ. Các tính toán được thực hiện trong trường nhỏ nhất mà đường cong được xác định và kết quả được nâng lên trường ban đầu.

Magma là mã nguồn đóng, không miễn phí và khó mở rộng bởi người dùng. Đồng nghĩa là Magma không thể bị thay đổi ngoại trừ bởi các lập trình viên cốt lõi, vì bản thân Magma có hơn hai triệu dòng mã C được biên dịch, kết hợp với khoảng nửa triệu dòng mã Magma được tích hợp (mà bất kỳ ai cũng có thể đọc và sửa đổi).

Syntax để tính số điểm trên đường cong Elliptic trong Magma

Order(EllipticCurve([GF(p) | a, b]));

##### SageMath

SageMath (trước đây là Sage hoặc SAGE, "System for Algebra and Geometry Experimentation") là một hệ thống đại số máy tính (CAS) với các tính năng bao gồm nhiều khía cạnh của toán học, bao gồm đại số, tổ hợp, lý thuyết đồ thị, giải tích số, lý thuyết số, giải tích và thống kê. SageMath sử dụng cú pháp tương tự Python.

Là một chương trình tính toán mã nguồn mở và miễn phí, SageMath được phát triển bởi chuyên gia và sinh viên, hỗ trợ bởi tình nguyện viên và trợ cấp. Bản thiết kế Sage tận dụng một số ý tưởng từ Magma, như phân cấp cha, thành phần, danh mục. Bản chất Sage là sự tích hợp bởi nhiều thư viện hay package đi kèm viết bằng C như FLINT, GAP hay NTL… và đẩy nhanh tốc độ tính toán bằng việc dùng ngôn ngữ Cython để chuyển đổi Python sang C.

Sage chứa sea.gp, triển khai nhanh thuật toán SEA để đếm số điểm trên đường cong Elliptic. Hiện tại, Sage chọn sử dụng thuật toán SEA khi Sage cũng có extension để so sánh kết quả với magma: magma.eval(“Order(EllipticCurve([GF(p) | a, b]))”)

Syntax để tính số điểm trên đường cong Elliptic trong SageMath

EllipticCurve(Zmod(p), [a, b]).cardinality()

Cú pháp đầy đủ của hàm cardinality là

**def** cardinality**(**self**,** algorithm**=None,** extension\_degree**=1)**

Trong đó, tham số algorithm là mặc định, có kiểu là string. Các giá trị tham số algorithm có thể truyền vào bao gồm:

* + - * ‘pari’: sử dụng hàm ellcard từ thư viện PARI C
      * ‘bsgs’: sử dụng thuật toán baby-step giant-step được triển khai trong Sage, với phiên bản Cremona-Sutherland của thủ thuật Mestre.
      * ‘exhaustive’: sử dụng phương pháp đếm cơ bản
      * ‘subfield’: rút gọn từ trường ban đầu thành trường thứ cấp, với điều kiện là j-invariant nằm trong một trường con.
      * ‘all’: kết hợp ‘pari’ và ‘bsgs’, trả về kết quả nếu phù hợp hoặc nếu không phù hợp thì báo lỗi ‘AssertionError’.

Tham số extension\_degree có kiểu là integer d, giá trị mặc định là 1. Tham số này quyết định cấp của p, tức mở rộng trường GF{p} thành GF{p^d}.

**CHƯƠNG 4:** [**XÂY**](https://docs.google.com/document/d/1NCg1O1_UqFLXPkUBzsgZVtoBzRWYq4bl/edit#heading=h.3whwml4) **DỰNG CHƯƠNG TRÌNH**



##### Nhiệm vụ chương trình

Chương trình nhắm tới việc hiển thị số điểm trên đường cong Elliptic sử dụng thuật toán đã nêu trên. Do chương trình thực hiện dùng ngôn ngữ C# Winform trên môi trường lập trình Visual Studio .NET 2022 là ngôn ngữ bậc cao, ta sẽ hướng đến việc tích hợp các chương trình bên ngoài sử dụng ngôn ngữ bậc thấp có sẵn để đẩy nhanh thời gian tính toán.

##### Tại sao dùng Pari/gp?

SageMath vốn chỉ có thể được dùng trên CLI của Sage, do Python gốc không tồn tại các hàm trong Sage. Do đó, Pari/gp là lựa chọn hợp lý, do kích thước nhỏ, tính tiện gọn và có thể được gọi từ C hoặc Python.

##### Gọi thư viện Pari/gp trong Python

Sau khi tải Pari/gp [2], ta dùng thư viện ctypes để gọi các hàm từ thư viện liên kết động. Do chương trình chạy trên hệ điều hành Windows, thư viện cần gọi ở đây là libpari.dll (trên Linux, hậu tố sẽ là .so).

import ctypes

# load the library

pari=ctypes.cdll.LoadLibrary("libpari.dll")

Sau đó, ta sẽ đặt kiểu dữ liệu tham số truyền vào và trả về cho các hàm trong pari để pari phân tích cú pháp. Đầu ra sẽ dùng hàm pari.pari\_printf(format, result) để trả về kết quả được định dạng theo format.

pari.Fp\_ellcard.restype = POINTER(c\_long)

pari.Fp\_ellcard.argtypes = [POINTER(c\_char\_p), POINTER(c\_char\_p), POINTER(c\_char\_p)]

pari.Fp\_ellcard\_SEA.restype = POINTER(c\_long)

pari.Fp\_ellcard\_SEA.argtypes = [POINTER(c\_char\_p), POINTER(c\_char\_p), POINTER(c\_char\_p), c\_long]

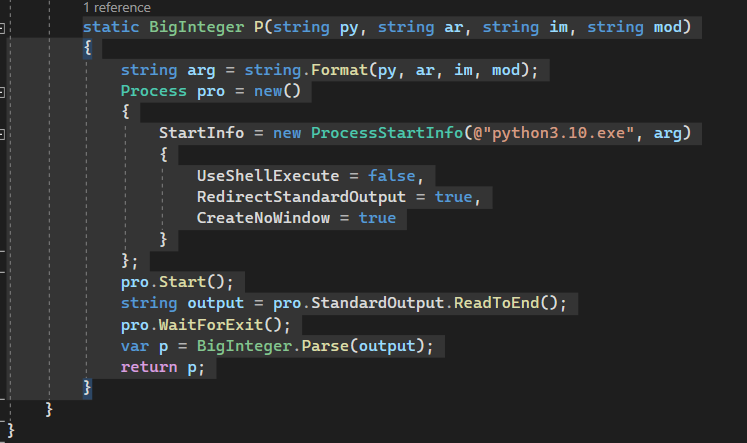
pari.strtoi.restype = POINTER(c\_char\_p)

Cuối cùng, ta khởi tạo thư viện để các hàm Pari hoạt động. Ta sẽ đặt kích thước maxsize và maxprime lớn nhất có thể.

# initialize the library

pari.pari\_init(2\*\*32-1, 2\*\*32-1)

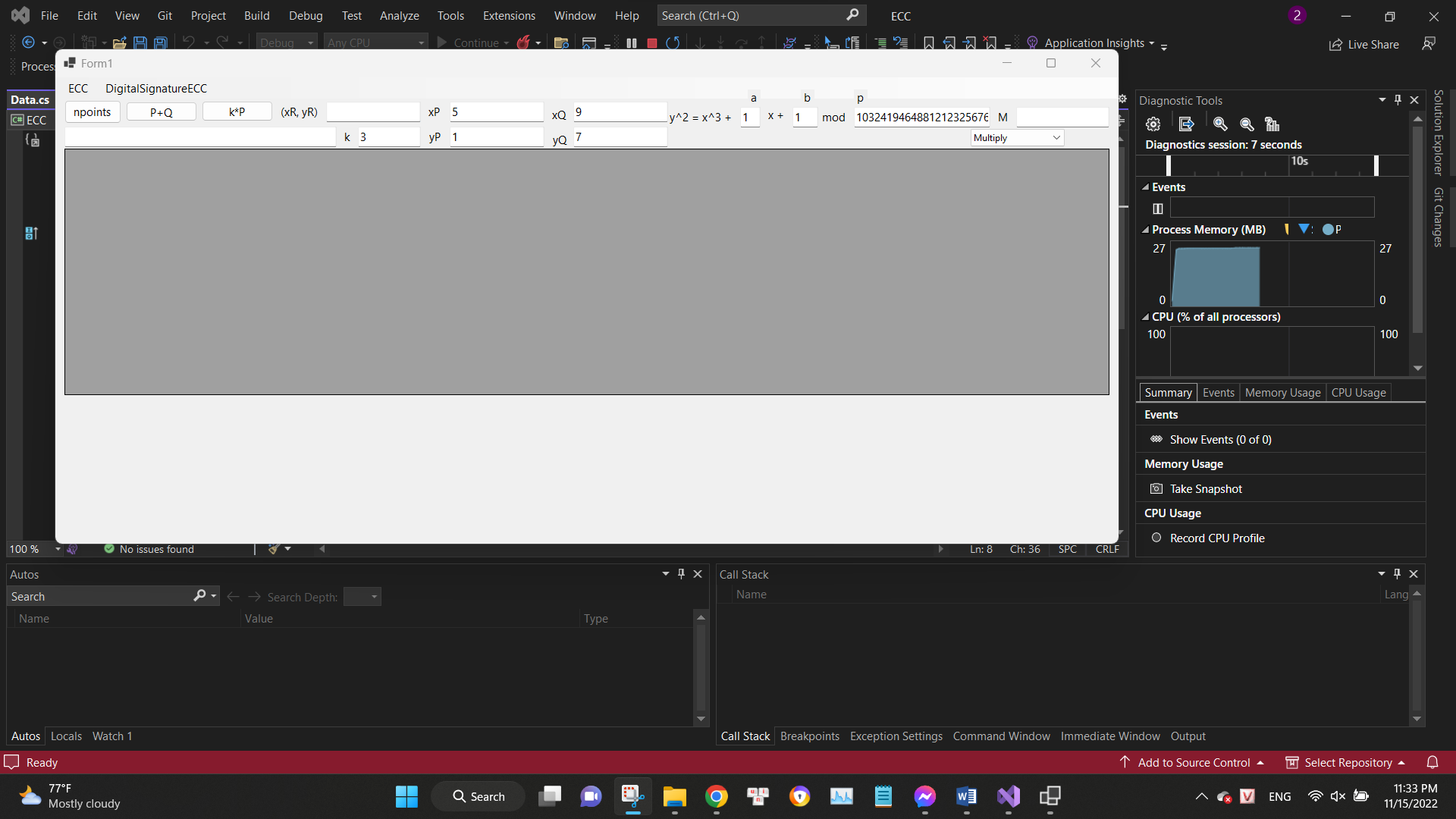
##### Chạy chương trình trên các ngôn ngữ khác

Trong nhiều trường hợp, ta có thể chạy code của ngôn ngữ khác thông qua việc chạy dòng lệnh bên ngoài. Các ngôn ngữ C# hay Java đều có lớp Process để hỗ trợ nhiệm vụ này. Hàm tạo có thể truyền tham số trực tiếp bao gồm tên tệp thực thi, với chức năng phiên dịch ngôn ngữ kịch bản tương ứng, và tham số dưới dạng string gồm các đầu vào như khi chạy trên giao diện dòng lệnh (command prompt). Một cách khác đó là có thể dùng thuộc tính StartInfo của Process rồi đặt FileName và Arguments. 

##### Kết quả của chương trình (Nền tảng .NET)

* + 1. **Giao diện**

Do ta xây dựng chương trình bằng ứng dụng Winforms- thư viện lớp GUI nên việc xây dựng giao diện là rất dễ dàng.



* + 1. **Các bước thực hiện chương trình**

Sau khi điền các giá trị a, b, p vào trong các textBox, ta sẽ click vào button npoints để đếm số điểm trên đường cong Elliptic. Chương trình sẽ bắt đầu phân tích cú pháp các giá trị trong textBox để thực hiện tính toán.

Truyền tham số vào Process:

string directory = AppDomain.CurrentDomain.BaseDirectory.Substring(0, AppDomain.CurrentDomain.BaseDirectory.Length - 25);

string py = directory + @"Ellinit.py {0} {1} {2}";

N = P(py, a.ToString(), b.ToString(), p.ToString());

Process sau khi chạy chương trình python sẽ trả về kết quả, phân tích cú pháp kết quả đó và gán vào trong textBox để hiển thị kết quả.

NumOfPoints.Text = N.ToString();

* + 1. **Kết quả thực nghiệm**
* Test case 1:

a = 1; b = 1; p = 1032419464881212325676479338986998839248491144169;

Kết quả: 1032419464881212325676478371189042478956918423696

* Test case 2:

a = 1; b = 1; p = 17

Kết quả: 18 //Lưu ý: #E(Fp) (mod p) sẽ chỉ tính #E(Fp) bằng thuật toán SEA với p đủ lớn, bởi vì |#E(Fp) − (p + 1)| ≤ 2 √ p. (Thực tế chỉ cần p>17 là đủ lớn). Do đó ta cần dùng hàm Fp\_ellcard trong trường hợp p bé và hàm Fp\_ellcard\_SEA cho p đủ lớn. Ta sẽ lấy điều kiện để tương đồng với hệ thống Pari/gp.

* Test case 3:

a = 250; b = 137; p = 1770497638664186404498962392225382062349387735577;

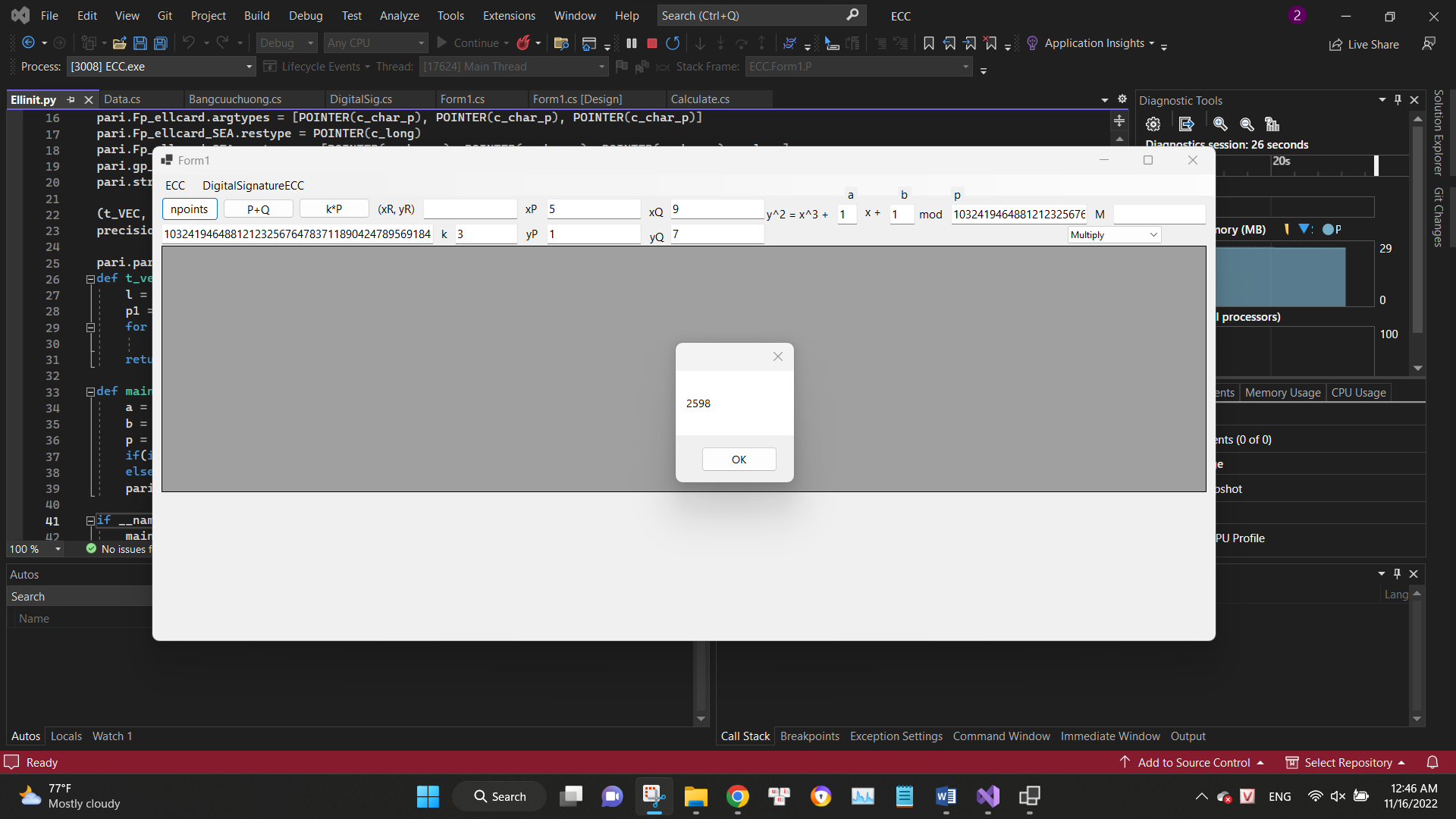
Kết quả: 1770497638664186404498960904014534934474489969784

* Test case 4:

a = 70569; b = 1981; p = 4282958018274153614864926283860056191329127422421;

Kết quả: 4282958018274153614864927816681782585028922562845

* Thời gian chạy trung bình là *~*2500ms (Ta dùng Stopwatch để tính thời gian chạy và MessageBox để hiển thị thời gian chạy).



**KẾT LUẬN**

Lý thuyết về đường cong Elliptic được tìm hiểu bởi lịch sử lâu dài của nó và sự đa dạng của các phương pháp đã được sử dụng trong nghiên cứu của nó. Qua tiểu luận, ta hướng đến việc sử dụng thuật toán và các chương trình đếm điểm, đặc biệt sử dụng thuật toán đếm điểm nâng cao SEA để đếm số điểm trên đường cong Elliptic trên trường hữu hạn Fp, trong đó p là số nguyên tố rất lớn.

Ứng dụng của thuật toán đếm điểm là rất quan trọng trong việc xây dựng hệ mật trên đường cong Elliptic. Do đó, đường cong Elliptic đã thu hút các nhà toán học bởi khía cạnh toán học của nó. Trong tương lai, với sự phát triển của máy tính và các hệ thống lý thuyết đại số, thời gian tính toán số điểm trên đường cong Elliptic sẽ ngày càng được rút ngắn, giúp việc xây dựng hệ mật ngày càng nhanh chóng và thực tiễn hơn.

# TÀI LIỆU THAM KHẢO

1. 1. Đỗ Xuân Bình, “Chữ ký số, chứng chỉ số và cơ sở hạ tầng khóa công khai

– Các vấn đề kỹ thuật và ứng dụng”, Nhà xuất bản bưu điện, 12 – 2007

1. 2. Suresh Sundriyal, “Counting points on Elliptic curves over Zp”, 2008

– https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=1356&context=theses

1. 3. <https://en.wikipedia.org/wiki/PARI/GP>
2. [https://fossies.org/linux/pari/src/functions/Elliptic\_curves/ellcard](https://fossies.org/linux/pari/src/functions/elliptic_curves/ellcard)