UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

ROLUL ALGORITMILOR GEOMETRICI ÎN CRIPTOGRAFIE

Absolvent Cuciureanu Dragoș-Adrian

Coordonator științific Prof. Dr. Stupariu Mihai-Sorin

București, iunie 2023

Cuprins

Rezumat

În această lucrare de licență vom aborda tematica ”Rolul algoritmilor geometrici în criptografie”, focusul fiind pus pe curbe eliptice peste corpuri finite.

Subiectul curbelor eliptice în criptografie înglobează o mare cantitate de teorie matematică și criptografică. Scopul acestei lucrări este de a oferi un rezumat concis al conceptelor fundamentale necesare aplicațiilor criptografice.

Această teză va fi împărțită în două părți: prima va consta într-o sinteză sumară a teoriei, iar cea din urmă va fi o documentație a aplicației suport ce are ca scop prezentarea a diferiți algoritmi și parcurgerea treptată a acestora bazat pe inputul utilizatorului.

Abstract

In this Bachelor Thesis we will address the topic ”The role of geometric algorithms in cryptography”, the focus being on elliptic curves over finite fields.

The subject of elliptic curves in cryptography encompasses a large amount of mathematical and cryptography theory. The purpose of this thesis is to provide a concise summary of the fundamental concepts required for cryptography applications.

This study will be divided into two parts: the first will consist of a brief synthesis of the theory, and the latter will be a documentation of the supporting application that aims to present different algorithms and present them step by step based on user's input.

1 Introducere

1.1 Rolul curbe eliptice în criptografie

Curbele eliptice au o serie de avantaje în comparație cu alte metode criptografice, cum ar fi dimensiuni reduse ale cheilor, calcule mai rapide și securitatea mai bună. Securitatea este redată de problema logaritmului discret pe curbe eliptice. Rezolvarea problemei logaritmului discret pe curbe eliptice este considerată dificilă din punct de vedere computațional, ceea ce este esențial pentru securitatea algoritmilor criptografici bazați pe curbe eliptice, cum ar fi schimburile de chei (exemplu: schimbul de chei Diffie-Hellman pe curbe eliptice), schemele de criptare și decriptare (exemplu: criptosistemul ElGamal) și semnăturile digitale (exemplu: ECDSA).

Vom parcurge în capitolele următoare concepte fundamentale, caracteristici și algoritmi asociați cu curbele eliptice, servind drept bază pentru înțelegerea rolului lor în protocoalele criptografice.

1.2 Scopul lucrării

Lucrarea are ca scop contribuția la asimilarea teoriei criptografiei pe curbe eliptice și a relevanței sale în domeniul criptografiei moderne și înțelegerea tehnică prin algoritmii și operațiile implementate ce simulează treptat procesul de lucru.

1.3 Structura lucrării

Teza va începe prin a oferi o imagine de ansamblu a curbelor eliptice, explorând proprietățile lor matematice. Această introducere va servi drept fundație pentru înțelegerea conceptelor ulterioare cu privire la beneficiile și dezavantajele curbelor eliptice în criptografie. Apoi, vom examina diferiți algoritmi și protocoale criptografice care folosesc curbele eliptice, incluzând o prezentare aprofundată a mecanismelor de schimb de chei, scheme de criptare și semnături digitale.

Capitolul 2 face o introducere în teoria curbelor eliptice, atât peste mulțimea $\mathbb{R}$, cât și peste corpuri finite $\mathbb{F}_p$ și vom prezenta algoritmii de adunare și înmulțire pe acestea.

Capitolul 3 face o introducere în criptografie și protocoale criptografice ce vor fi adaptate pe curbe eliptice.

Capitolul 4 prezintă adaptarea de la protocoalele criptografice normale la protocoalele criptografice pe curbe eliptice peste corpuri finite.

Capitolul 5 descrie implementările algoritmilor realizați în aplicația suport. Aplicația poate fi considerată un calculator pentru diferiți algoritmi peste curbe eliptice, dar reprezintă și un instrument educativ, prezentând fiecare algoritm atât teoric, cât și practic prin calcule explicite bazate pe inputul utilizatorului.

În capitolul 6 sunt prezentate concluziile acestei lucrări și posibile implementări viitoare.

2 Preliminarii și concepte de bază în curbe eliptice

2.1 Curbe eliptice

Următorul paragraf a fost preluat și adaptat din [17].

Definiţia 2.1. Fie un corp comutativ pe care îl vom nota cu $\mathbb{K}$ și două elemente a, b aparținând corpului $\mathbb{K}$. Polinomul $f(X) = X^3 + aX + b$ cu coeficienți în $\mathbb{K}$ definește o curbă peste corpul $\mathbb{K}$:

$$E(\mathbb{K}) = \{(X, Y) \in \mathbb{K}^2 : Y^2 = f(X)\}$$

$\mathbb{K}$ poate fi unul dintre următoarele corpuri: corpul numerelor reale $\mathbb{R}$, corpul $\mathbb{F}_p$ (cu p număr prim) sau corpul $\mathbb{F}_{p^k}$ (cu p număr prim și $k \geq 1$).

Fie x_1, x_2, x_3 rădăcinile polinomului $f(X) = X^3 + aX + b$, atunci discriminantul său este:

$$\begin{aligned}\Delta_E =& \big[(x_1 - x_2)(x_2 - x_3)(x_3 - x_1)\big]^2 \\ =& -16(4a^3 + 27b^2)\end{aligned}$$

Cum în viitor vom lua Δ_E în contextul $\Delta_E \neq 0$, putem considera:

$$\begin{aligned}\Delta_E =& -4a^3 - 27b^2 \text{ sau} \\ =& 4a^3 + 27b^2\end{aligned}$$

Pentru a avea o curbă eliptică, toate rădăcinile trebuie sa fie distincte una față de celelalte.

Observația 2.2. Polinomul f are rădăcini distincte una față de celelalte, dacă și numai dacă $\Delta_E \neq 0$.

Definiţia 2.3. Fie $F(X, Y) = X^3 + aX + b - Y^2$ și punctul $P(x_0, y_0) \in E(\mathbb{K})$ un punct de pe curbă. Punctul se numește singular dacă:

$$\frac{\partial F}{\partial x}(x_0, y_0) = \frac{\partial F}{\partial y}(x_0, y_0) = 0$$

Definiţia 2.4. O curbă $E(\mathbb{K})$ cu $\Delta_E \neq 0$ nu are puncte singulare.

Exemplul 2.5. Exemple de curbe care au $\Delta_E = 0$ (au puncte singulare):

$E1 : Y^2 = X^3$ vârful de coordonate $(0, 0)$ este punct de inflexiune

$E2 : Y^2 = (X + 4)^2(X - 8) = X^3 - 48X - 128$ punctul de coordonate $(-4, 0)$ este izolat

Acestea sunt exemplele ilustrate (reprezentările au fost realizate cu [6]):

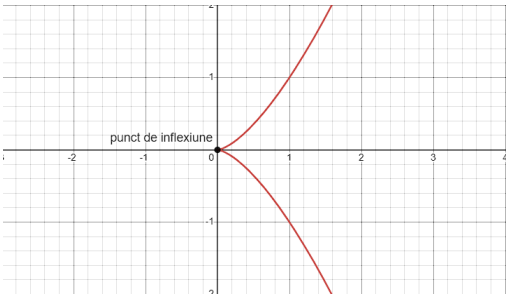


Figura 1: Punctul de inflexiune al $E_1 : Y^2 = X^3$

Figura 2: Punctul izolat al $E_2 : X^3 - 48X - 128$

Definiţia 2.6. Definim ca fiind punctul de la ”infinit” punctul $O(\infty, \infty)$.

Acest punct nu există în planul XOY. Făcând o referință la definiția din [32], care precizează că două drepte paralele se întâlnesc la ”infinit”, atunci considerăm că toate verticalele paralele cu OY se intersectează în punctul $O(\infty, \infty)$.

Definiţia 2.7. Fie un corp comutativ $\mathbb{K}$, cu $a, b \in \mathbb{K}$ două elemente aparținând de corpul $\mathbb{K}$ și $f(X) = X^3 + aX + b$ un polinom cu coeficienți în $\mathbb{K}$. Acest polinom definește o curbă eliptică peste corpul $\mathbb{K}$:

$$E(\mathbb{K}) = \{(X, Y) \in \mathbb{K}^2 : Y^2 = f(X)\} \cup \{O(\infty, \infty)\}$$
$$\text{cu } \Delta_E \neq 0$$

Pe scurt, putem spune că o curbă eliptică este totalitatea soluțiilor ecuației cu $\Delta_E \neq 0$ de forma:

$$E : Y^2 = X^3 + aX + b$$

Ecuațiile de această natură se numesc ecuații *Weierstrass*.

Exemplul 2.8. Exemple de curbe eliptice:

$$E_1 : Y^2 = X^3 + X + 8$$
$$E_2 : Y^2 = X^3 - 4X - 1$$

Acestea sunt exemplele ilustrate (reprezentările au fost realizate cu [6]):

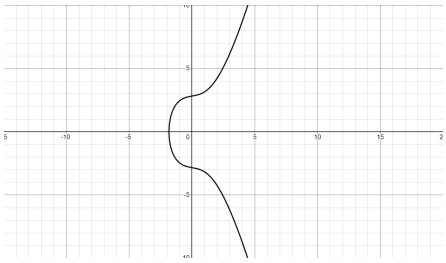


Figura 3: $E_1 : Y^2 = X^3 + X + 8$

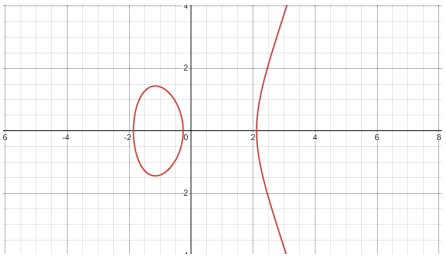


Figura 4: $E_2 : Y^2 = X^3 - 4X - 1$

2.2 Adunarea punctelor pe curbe eliptice

Adunarea pe o curbă eliptică este operația de a ”combina” două puncte de pe curbă pentru a produce un al treilea punct. Operația de adăugare este definită geometric și joacă un rol fundamental în criptografia cu curbe eliptice.

Fie două puncte de pe curba eliptică E: P și Q, de coordonate (x_1, y_1), respectiv de coordonate (x_2, y_2). Trasăm dreapta ce trece prin cele 2 puncte. Cum curba eliptică este determinată de un polinom de gradul 3, dreapta desenată va interesecta curba eliptică E în exact 3 puncte (nu este nevoie ca acestea să fie distincte). Vom nota al treilea punct din intersecție cu $R'(x_3, y_3)$, realizăm reflecția sa față de axa OX (curba eliptică E este simetrică față de axa OX) adică, din punct de vedere numeric înmulțim coordonata a doua a punctului R cu -1, acest punct va fi notat cu R și va avea coordonatele $(x_3, -y_3)$.

Punctul R este rezultatul adunării între punctele P si Q:

$$P + Q = R$$

Exemplul 2.9. Fie curba eliptică E de ecuație:

$$E : Y^2 = X^3 - 94X + 361 \tag{1}$$

Și punctele $P(4, 7)$ și $Q(8, 11)$ de pe E. Calculăm ecuația dreptei L ce intersectează punctele cu ajutorul pantei:

$$\begin{aligned} m &= \frac{y_2 - y_1}{x_2 - x_1} \\ &= \frac{11 - 7}{8 - 4} = 1 \\ L &: Y - y_1 = m(X - x_1) \\ L &: Y - 7 = 1(X - 4) \\ L &: Y = X + 3 \end{aligned} \tag{2}$$

Pentru a determina R' calculăm toate punctele de intersecție dintre dreapta L si curba eliptică E, substituind ecuația lui L (2) în ecuația lui E (1):

$$\begin{aligned} (X+3)^2 &= X^3 - 94X + 361 \\ X^2 + 6X + 9 &= X^3 - 94X + 361 \\ X^3 - X^2 - 100X + 352 &= 0 \end{aligned} \tag{3}$$

Cum ecuația rezultată la (3) este de gradul 3, va avea exact 3 rădăcini. Cum punctele $P(4,7)$ și $Q(8,11)$ sunt si pe dreaptă si pe curbă, înseamnă că 4 și 8 sunt rădăcini, deci ne rămâne de găsit doar a 3-a rădăcină (descompunem în factori):

$$X^3 - X^2 - 100X + 352 = (X-8)(X-4)(X+11) = 0$$

Din descompunerea în factori determinăm că a treia rădăcină este -11, aceasta fiind și componenta R'_x, acum calculăm componeta R'_y din ecuația dreptei de la punctul (2):

$$\begin{aligned} Y &= X + 3 \\ &= (-11) + 3 = -8 \end{aligned}$$

Astfel obținem punctul $R'(-11,-8)$, tot ce rămăne de făcut pentru a obține rezultatul căutat este să reflectăm componenta R'_y și obținem $R(-11,8)$, prin urmare:

$$P + Q = (-11, 8)$$

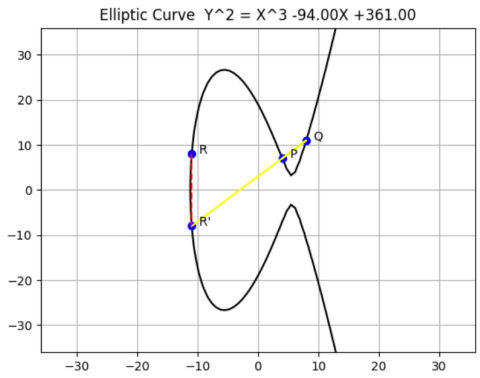


Figura 5: Reprezentare grafică

Point	X	Y
P	4.0	7.0
Q	8.0	11.0
R	-11.0	8.0

Figura 6: Tabel cu rezultate

Acesta este cazul general de adunare a două puncte de pe o curbă eliptică. Însă există și câteva spețe particulare, fie punctul P, de coordonate (x,y) și opusul său $P' = -P$, de coordonate $(x,-y)$ și punctul $O(\infty,\infty)$:

(i) $P + P$ adunarea unui punct cu el însuși

(ii) $P + P'$ adunarea unui punct cu opusul său

(iii) $P + O$ adunarea unui punct cu infinit

Începem cu primul caz particular, pentru a realiza $P + P$ dreapta L va fi tangenta la E, astfel dreapta intersectează curba eliptică tot în 3 puncte, doar că 2 dintre acestea sunt P (putem să facem o paralelă cum un polinom ca $(x-1)^2$, care are 2 rădăcini, chiar dacă acestea sunt identice). Al treilea punct va fi R' și apoi calculăm R la fel ca mai sus.

Exemplul 2.10. Fie curba eliptică E de ecuație:

$$E : Y^2 = X^3 + 3X + 29 \tag{4}$$

Și punctul $P(5, 13)$ de pe E, calculăm $P + P$:
Determinăm panta lui E prin diferențiere în (1):

$$\begin{aligned} 2Y\frac{dY}{dX} &= 3X^2 + 3 \\ \frac{dY}{dX} &= \frac{3X^2+3}{2Y} \end{aligned}$$

Calculăm panta lui E în P substituind P în ecuație și obținem panta $m = \dfrac{78}{26} = 3$. Astfel tangenta la E în P este:

$$\begin{aligned} L &: Y - y_1 = m(X - x_1) \\ L &: Y - 13 = 3(X - 5) \\ L &: Y = 3X - 2 \end{aligned} \tag{5}$$

Analog ca la adunarea generală, determinăm R' calculând toate punctele de intersecție dintre dreapta L si curba eliptică E, substituind ecuația lui L (5) în ecuația lui E (4):

$$\begin{aligned} (3X-2)^2 &= X^3 + 3X + 29 \\ 9X^2 - 12X + 4 &= X^3 + 3X + 29 \\ X^3 - 9X^2 + 15X + 25 &= 0 \\ (X-5)^2(X+1) &= 0 \end{aligned} \tag{6}$$

Observăm din descompunere ca rădăcinile sunt: -1, 5 și 5 (de remarcat că rădăcina 5, mai exact componenta P_x apare de două ori). Cea de-a treia rădăcină este componenta R'_x, pe care o

introducem în ecuația de la (5) și obținem $R'(-1,-5)$, respectiv $R(-1,5)$:

$$P + P = (-1, 5)$$

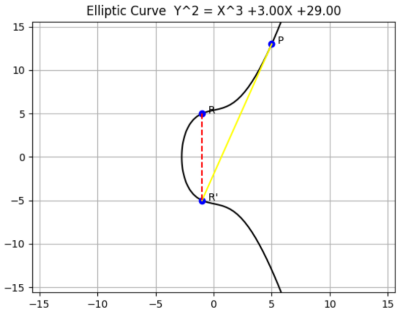


Figura 7: Reprezentare grafică

Point	*X*	*Y*
P	5.0	13.0
Q	5.0	13.0
R	−1.0	5.0

Figura 8: Tabel cu rezultate

Trecem la al doilea caz: $P + P'$ (adunarea unui punct cu opusul său).

Observăm că apar probleme când încercăm să adunăm la punctul $P(x, y)$, reflexia (opusul) sa față de axa OX, mai exact punctul $P' = -P = (x, -y)$. Încercăm să reprezentăm geometric adunarea ducând dreapta L prin punctele P și P', astfel creând o verticală paralelă cu OY cu $X = x$. Neexistând un al treilea punct al intersecției, considerăm punctul $O(\infty, \infty)$ definit anterior ca fiind rezultatul adunării.

$$P + P' = O$$

Exemplul 2.11. Considerăm aceeași curbă eliptică E de la (1) și punctele $P(4, 7)$ și $P'(4, -7)$ existente pe E și calculăm $P + P'$ ca fiind:

$$P(4, 7) + P'(4, -7) = O(\infty, \infty)$$

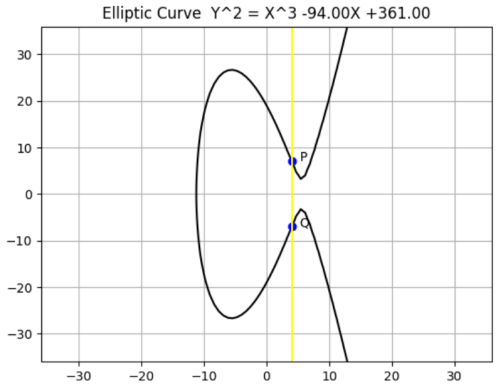


Figura 9: Reprezentare grafică

Point	*X*	*Y*
P	4.0	7.0
Q	4.0	-7.0
R	*inf*	*inf*

Figura 10: Tabel cu rezultate

Cazul al treilea: $P + O$ (adunarea unui punct cu infinit).

Odată ce am definit punctul de la ”infinit” O, trebuie să definim cum se adună un punct oarecare $P(x, y)$ de pe o curbă elitptică E cu punctul O. Și această speță este mai ușor de prezentat cu ajutorul reprezentării geometrice. Cum punctul O există pe fiecare verticală, putem construi dreapta L ca fiind dreapta ce trece prin punctele P și O. Dreapta L și curba eliptică E au trei puncte de intersecție, mai precis: P, O și P'. Deci, conform algoritmului de adunare definit anterior, rezultă că $R' = P'$ și $R = P$, astfel:

$$P + O = P$$

Deci, punctul O este element neutru pentru adunarea pe curbe eliptice.

Exemplul 2.12. Considerăm aceeași curbă eliptică E de la (1) și punctul $P(4, 7)$ existent pe E și calculăm $P + O$ ca fiind:

$$P(4, 7) + O(\infty, \infty) = P(4, 7)$$

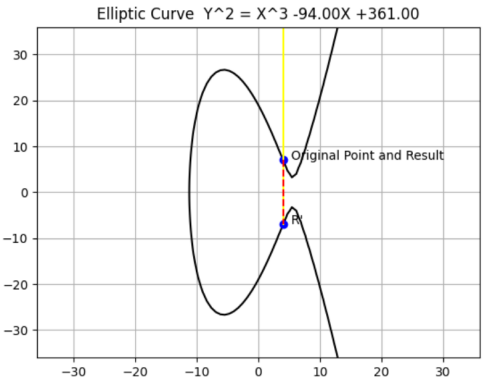


Figura 11: Reprezentare grafică

Point	*X*	*Y*
P	4.0	7.0
Q	*inf*	*inf*
R	4.0	7.0

Figura 12: Tabel cu rezultate

Pe lângă acestea mai există încă o speță particulară, care este reuniunea cazurilor 1 și 2 anterioare. Mai exact când adunăm punctul $P(x, y)$ cu el însuși și rezultă $O(\infty, \infty)$:

$$P + P = O$$

Dar tocmai am arătat că:

$$P + P' = O$$

Deci, P de coordonate (x, y) este egal cu opusul său P', de coordonate $(x, -y)$, ceea ce înseamnă că $y = -y$, singura soluție validă fiind $y = 0$. Pe scurt acest caz se poate realiza doar când adunăm un punct cu el însuși și tangenta este o verticală paralelă cu OY.

Exemplul 2.13. Fie curba eliptică E în $\mathbb{R}$:

$$E : Y^2 = X^3 - 6X + 9$$

Și punctele $P(-3, 0)$ și $P'(-3, 0)$:

$$\begin{aligned} P(-3,0) &= P'(-3,0) \\ P(-3,0) + P(-3,0) &= P(-3,0) + P'(-3,0) \\ &= 0(\infty, \infty) \end{aligned}$$

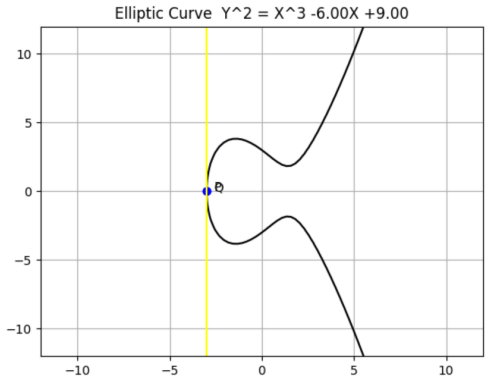


Figura 13: Reprezentare grafică

Point	*X*	*Y*
P	-3.0	0.0
Q	-3.0	0.0
R	inf	inf

Figura 14: Tabel cu rezultate

Definiţia 2.14. Fie o curbă eliptică E în $\mathbb{R}$ și punctele P și Q, de coordonate (x_1, y_1), respectiv de coordonate (x_2, y_2), notăm reflexia punctului față de axa OX ca fiind $Q' = -Q$, de coordonate $(x_2, -y_2)$. Definim scăderea ca fiind:

$$P - Q = P + (-Q) = P + Q'$$

Definiţia 2.15. Fie o curbă eliptică E în $\mathbb{R}$ și punctul $P(x, y)$ de pe E și scalarul n, definim înmulțirea ca fiind adunarea repetată a punctului P cu el însuși de n ori:

$$nP = \underbrace{P + P + P + ... + P}_{\text{de n ori}}$$

Exemplul 2.16. Considerăm aceeași curbă eliptică E în $\mathbb{R}$ de la (4), punctul $P(5, 13)$ existent pe E și scalarul $n = 3$:

$$\begin{aligned} 3P &= P + P + P \\ &= (5, 13) + (5, 13) + (5, 13) \\ &= (-1, 5) + (5, 13) \\ &= (-2.2222, -3.3704) \end{aligned}$$

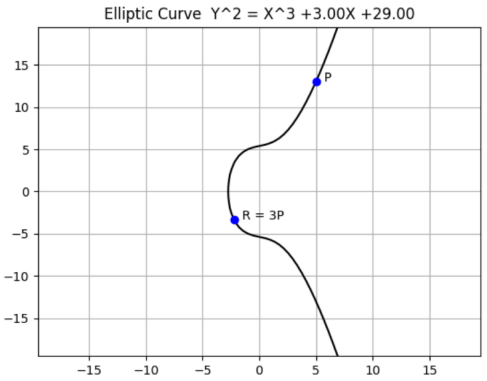


Figura 15: Reprezentare grafică

Point	X	Y
P	5.0	13.0
R	-2.2222	-3.3704

Figura 16: Tabel cu rezultate

Următoarele teoreme au fost preluate și adaptate din [18] și folosite în 5.3.1.

Teorema 2.17. Fie o curbă eliptică E în $\mathbb{R}$, și fie $\mathbb{E}$ mulțimea punctelor de pe E, atunci ($\mathbb{E}$, $+$) formează un grup abelian:

1. Comutativitate: $P + Q = Q + P$ pentru orice $P, Q \in \mathbb{E}$
2. Asociativitate: $(P + Q) + R = P + (Q + R)$ pentru orice $P, Q, R \in \mathbb{E}$
3. Existența elementului neutru: $P + O = O + P = P$ pentru orice $P \in \mathbb{E}$
4. Existența elementului invers: $P + (-P) = O$ pentru orice $P \in \mathbb{E}$

Teorema 2.18. Fie o curbă eliptică E:

$$E : Y^2 = X^3 + aX + b$$

și punctele P și Q, de coordonate (x_1, y_1), respectiv de coordonate (x_2, y_2) de pe E, algoritmul de adunare în pseudocod este:

(a) dacă $P = O$, atunci $P + Q = Q$

(b) altfel, dacă $Q = O$, atunci $P + Q = P$

(c) altfel, dacă $x_1 = x_2$ și $y_1 = -y_2$, atunci $P + Q = O$

(d) altfel, calculăm panta m ca fiind:

$$m = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & \text{dacă } P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1} & \text{dacă } P = Q \end{cases}$$

și fie:

$$x_3 = m^2 - x_1 - x_2$$
$$y_3 = m(x_1 - x_3) - y_1$$

rezultă:

$$P + Q = R = (x_3, y_3)$$

2.3 Curbe eliptice peste corpuri finite

În subcapitolele anterioare am prezentat curbele eliptice și operațiile ce se pot realiza pe acestea. Abordarea a fost predominantă geometric, spre exemplu, la adunarea a două puncte P și Q de pe o curbă eliptică E, am trasat o dreaptă ce trece prin cele două puncte și am determinat care este al treilea punct în care intersectează dreapta curba eliptică și am realizat reflexia față de axa OX. Aceasta poate fi observată în subcapitolul anterior unde am arătat cum se realizează operații pe curbe eliptice.

Conform [23], în criptografie se folosesc curbe eliptice peste corpuri finite $\mathbb{F}_p$ și nu curbe eliptice în $\mathbb{R}$, deoarece acestea oferă securitate mai bună și eficiență din punct de vedere computațional. Aceste proprietăți fac din criptografia cu curbe eliptice pe corpuri finite o alegere populară în sistemele criptografice moderne. Deci, trebuie să examinăm curbele eliptice ale căror puncte au coordonate într-un corp finit $\mathbb{F}_p$ pentru a aplica teoria curbelor eliptice la criptografie.

Definiţia 2.19. Fie $\mathbb{F}_p$ un corp comutativ, unde p este număr prim, cu $p \geq 2$. Fie $a, b \in \mathbb{F}_p$ două elemente ce aparțin corpului $\mathbb{F}_p$ și $f(X) = X^3 + aX + b$ un polinom cu coeficienți în $\mathbb{F}_p$. Acest polinom definește o curbă eliptică peste corpul $\mathbb{F}_p$:

$$E(\mathbb{F}_p) = \{(X, Y) \in \mathbb{F}_p^2 : Y^2 = f(X)\} \cup \{O\}$$
$$\text{cu } \Delta_E = 4a^3 + 27b^2 \neq 0$$

Datele din exemplu au fost preluate din [17].

Exemplul 2.20. Fie curbă eliptică E peste $\mathbb{F}_5$:

$$E : Y^2 = X^3 + 1 \text{ (mod } 5) \tag{7}$$

O metodă de a determina punctele curbei eliptice $E(\mathbb{F}_5)$ este de a da toate valorile lui X, mai precis $0, 1, 2, 3, 4$:

$$\begin{aligned} f(0) &= 0 + 1 = 1 \\ f(1) &= 1 + 1 = 2 \\ f(2) &= 8 + 1 = 3 + 1 = 4 \\ f(3) &= 27 + 1 = 2 + 1 = 3 \\ f(4) &= 64 + 1 = 4 + 1 = 5 = 0 \end{aligned}$$

Și să verificăm în ecuația curbei care rezultate sunt resturi pătratice:

$$\begin{aligned} 0^2 &\equiv 0 \text{ (mod } 5) \\ 1^2 &\equiv 4^2 \equiv 1 \text{ (mod } 5) \\ 2^2 &\equiv 3^2 \equiv 4 \text{ (mod } 5) \end{aligned}$$

Observăm că doar 0, 1 și 4 sunt resturi pătratice, deci spre exemplu pentru $X = 0$, $Y^2 = f(X) = 1$, ceea ce înseamnă că Y poate fi ± 1 modulo 5, adică 1 sau 4, deci rezultă punctele $(0, 1)$ și $(0, 4)$ în $E(\mathbb{F}_5)$.

Analog determinăm restul punctelor:

$$E(\mathbb{F}_5) = \big\{(0, 1), (0, 4), (2, 2), (2, 3), (4, 0), O\big\}$$

Astfel, cardinalul mulțimii $E(\mathbb{F}_5)$ este 6.

2.4 Adunarea punctelor pe curbe eliptice peste corpuri finite

Fie o curbă eliptică E peste $\mathbb{F}_p$:

$$E : Y^2 = X^3 + aX + b \text{ (mod } p)$$

și punctele P și Q, de coordonate (x_1, y_1), respectiv de coordonate (x_2, y_2) de pe $E(\mathbb{F}_5)$ și punctul $R(x_3, y_3)$ rezultatul adunării dintre P și Q. Algoritmul de adunare a punctelor P și Q este același

ca la teorema 2.18, doar că operațiile vor fi modulo p, nu în $\mathbb{R}$. Observăm că operațiile folosite sunt adunarea, scăderea, înmulțirea și împărțirea asupra coordonatelor punctelor și coeficienților lui E. Și cum operațiile folosite sunt închise cu $\mathbb{F}_p$, înseamnă că și rezultatul va fi tot în $\mathbb{F}_p$.

Următoarea teoremă este preluată și adaptată din [18] și folosită în 5.3.2.

Teorema 2.21. Fie o curbă eliptică E peste $\mathbb{F}_p$:

$$E : Y^2 = X^3 + aX + b \pmod{p}$$

și punctele $P(x_1, y_1)$ și $Q(x_2, y_2)$ de pe $E(\mathbb{F}_p)$.

(a) Algoritmul de adunare a punctelor pe o curbă eliptică (teorema 2.18) aplicat pe punctele P și Q rezultă într-un punct în $E(\mathbb{F}_p)$. Notăm acest punct cu $P + Q$.

(b) Această lege de adunare pe $E(\mathbb{F}_p)$ satisface toate proprietățile enumerate în teorema 2.17. Cu alte cuvinte, $E(\mathbb{F}_p)$ cu legea de adunare formează un grup finit.

Exemplul 2.22. Considerăm aceeași curbă eliptică E în $\mathbb{F}_5$ de la (7) și fie punctele $P(0, 1)$ și $Q(2, 3)$ de pe $E(\mathbb{F}_5)$. Folosind algoritmul de adunare a punctelor pe o curbă eliptică (teorema 2.18) realizăm $P + Q$, începând cu calculul pantei dreptei:

$$\begin{aligned} m &= \frac{y_2 - y_1}{x_2 - x_1} \\ &= \frac{2 - 0}{3 - 1} \\ &= \frac{2}{2} = 1 \end{aligned}$$

În continuare calculăm ν:

$$\begin{aligned} \nu &= y_1 - mx_1 \\ &= 1 - (1 \cdot 0) = 1 \end{aligned}$$

Ne rămâne de calculat doar rezultatul adunării:

$$\begin{aligned} x_3 &= m^2 - x_1 - x_2 \\ &= 1^2 - 0 - 2 \\ &= -1 = 4 \\ y_3 &= -(mx_3 + \nu) \\ &= -(1 \cdot 4 + 1) \\ &= -5 = 0 \end{aligned}$$

Cum calculăm în $\mathbb{F}_5$, $-1 \equiv 4 \pmod 5$ și $-5 \equiv 0 \pmod 5$. Deci, rezultatul final este:

$$P(0,1) + Q(2,3) = (4,0)$$

Analog, conform teoremei de adunare, putem aduna și $P(0,1) + P(0,1)$:

$$\begin{aligned} m &= \frac{3x_1^2 + a}{2y_1} \\ &= \frac{0^2 + 0}{2 \cdot 1} = 0 \\ \nu &= y_1 - mx_1 \\ &= 1 - (0 \cdot 0) = 1 \end{aligned}$$

Și, în final:

$$\begin{aligned} x_3 &= m^2 - x_1 - x_2 \\ &= 0^2 - 0 - 0 = 0 \\ y_3 &= -(mx_3 + \nu) \\ &= -(0 \cdot 0 + 1) \\ &= -1 = 4 \end{aligned}$$

Deci, rezultatul final este:

$$P(0,1) + P(0,1) = (0,4)$$

Putem face analog toate adunările din $E(\mathbb{F}_5)$ și ar rezulta:

$+$	O	$(0,1)$	$(0,4)$	$(2,2)$	$(2,3)$	$(4,0)$
O	O	$(0,1)$	$(0,4)$	$(2,2)$	$(2,3)$	$(4,0)$
$(0,1)$	$(0,1)$	$(0,4)$	O	$(2,3)$	$(4,0)$	$(2,2)$
$(0,4)$	$(0,4)$	O	$(0,1)$	$(0,4)$	$(2,2)$	$(2,3)$
$(2,2)$	$(2,2)$	$(2,3)$	$(4,0)$	$(0,4)$	O	$(0,1)$
$(2,3)$	$(2,3)$	$(4,0)$	$(2,2)$	O	$(0,1)$	$(0,4)$
$(4,0)$	$(4,0)$	$(2,2)$	$(2,3)$	$(0,1)$	$(0,4)$	O

Tabelul 1: Tabelul de adunare pentru $E : Y^2 = X^3 + 1 \pmod 5$

2.5 Ordinul unei curbe eliptice peste corpuri finite

Următorul paragraf este preluat și adaptat din [18].

Mulțimea punctelor din $E(\mathbb{F}_p)$ este finită, numărul maxim de posibilități fiind p pentru X și p pentru Y, adică p^2 aranjamente. Dar cum ecuația curbei eliptice peste un corp finit este:

$$E : Y^2 = X^3 + aX + b \text{ (mod } p)$$

Înseamnă că pentru fiecare Y există maxim 2 valori X care pot exista. Luând în considerare și punctul O, ajungem la $\#E(\mathbb{F}_p)$ (cardinalul sau ordinul grupului $E(\mathbb{F}_p)$) are cel mult $2p + 1$ puncte cu tot cu punctul O (punctul la infinit). Acest estimat este doar o margine superioară și este mult mai mare decât valoarea în practică.

Când introducem o valoare pentru X, există trei posibilități pentru valoarea cantității:

$$X^3 + aX + b$$

În primul caz, poate fi un rest pătratic modulo p, caz în care are două pătrate și obținem două puncte în $E(\mathbb{F}_p)$. Acest lucru se întâmplă în aproximativ 50% din cazuri. În al doilea rând, poate fi un modulo p nereziduu, caz în care aruncăm X. Aceasta se întâmplă și în aproximativ 50% din situații. În al treilea rând, ar putea fi egal cu 0, caz în care obținem un punct în $E(\mathbb{F}_p)$, dar acest caz se întâmplă foarte rar. Astfel am putea aștepta ca numărul de puncte din $E(\mathbb{F}_p)$ să fie aproximativ:

$$\#E(\mathbb{F}_p) \approx 50\% \cdot 2p + 1 = p + 1$$

Următoarea teoremă și observație sunt preluate din [18].

Teorema 2.23. (Hasse) Fie E o curbă eliptică peste $\mathbb{F}_p$, atunci

$$\#E(\mathbb{F}_p) = p + 1 - t_p \text{ cu } |t_p| \leq 2\sqrt{p}$$

Observația 2.24. Din teorema lui Hasse rezultă următoarea inegalitate

$$p + 1 - 2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p}$$

Definiţia 2.25. Variabila t_p, care verifică ecuația:

$$t_p = p + 1 - \#E(\mathbb{F}_p)$$

din teorema 2.23 se numește urma lui Frobenius pentru $E/\mathbb{F}_p$.

Următoarea observație este inspirată din [18].

Observația 2.26. Pentru a se determina ordinul se poate realiza ”brut force”, adică a se înlocui fiecare valoare a lui X din intervalul $\left[0, 1, \cdots p-1\right]$ și verificat rezultatul ecuației curbei (adică Y^2) față de un tabel precalculat cu toate pătratele modulo p. Acest proces ar dura $O(p)$ (timp liniar față de numărul prim p), deci este foarte ineficient. Algoritmul pași de copil pași de uriaș realizat de Shanks determină ordinul pe baza coliziunilor între puncte și ordinul acestora. Algoritmul din urmă are o complexitate de $O(\sqrt{p})$ și este prezentat într-un subcapitol viitor. Schoof a găsit un algoritm pentru a calcula $\#E(\mathbb{F}_p)$ în $O((\log p)^6)$ (un algoritm de timp polinomial). Algoritmul lui Schoof a fost îmbunătățit de Elkies și Atkin și adus la o complexitate de $O((\log p)^4)$, iar acum este cunoscut ca algoritmul SEA (după inițialele celor trei, mai multe detalii se pot găsi în [25]).

Acest capitol a avut ca scop sumarizarea conceptelor de bază a curbelor eliptice pentru aplicarea acestora în criptografie. Există multe articole și cărți ce explică teoria și concepte mai avansate, câteva exemple fiind: [5], [21], [22], [29], [30].

3 Preliminarii și concepte de bază în Criptografie

3.1 Problema logaritmului discret (DLP)

Problema logaritmului discret, sau mai bine cunoscută după numele din engleză: Discrete Logarithm Problem (DLP) apare în multe probleme matematice, însă, pentru curbele eliptice apare cel mai des peste corpuri finite $\mathbb{F}_p$ ($\mathbb{F}_p$ fiind un corp comutativ, iar p fiind un număr prim). Prima carte publicată ce abordează construirea de chei publice se bazează pe problema logaritmului discret peste corpuri finite $\mathbb{F}_p$ și a fost realizată de Diffie și Hellman [7] (acești autori consacrați sunt și cei după care a fost numit algoritmul de schimb de chei Diffie-Hellman).

Reamintim "mica" teoremă a lui Fermat și teorema rădăcinii primitive. Cele trei exemple legate de mica teorema lui Fermat și teorema rădăcinii primitive și teoremele au fost preluate din [18].

Teorema 3.1. (Mica teoremă a lui Fermat) Fie p un număr prim și a un număr întreg, atunci

$$a^{p-1} \equiv \begin{cases} 1 \pmod{p} & \text{dacă } p \nmid a \\ 0 \pmod{p} & \text{dacă } p \mid a \end{cases}$$

Exemplul 3.2. Fie numărul prim $p = 15485863$, deci teorema 3.1 ne spune că

$$2^{15485862} \equiv 1 \pmod{15485863}$$

Astfel, fără a face niciun calcul putem să determinăm că numărul $2^{15485862} - 1$, ce are aproximativ 4500000 de cifre (pentru determinarea numărului de cifre am folosit acest calculator), este multiplu al numărului 15485863.

Observația 3.3. (a) Din exemplul anterior observăm că oricât de mare ar fi un număr prim p, putem să determinăm multiplii săi, aceștia fiind de natura:

$$a^p - 1 \text{ cu } p \nmid a$$

(b) Această observație este preluată din [18]. Folosind mica teoremă a lui Fermat și algoritmul de ridicat rapid la putere, putem să determinăm eficient inversele modulo-ului p, adică

$$a^{p-2} \equiv a^{-1} \pmod{p}$$

Putem demonstra a doua observație înmulțind de ambele părți cu a și obținem mica teoremă a lui Fermat $a^{p-1} \equiv 1 \pmod{p}$.

Exemplul 3.4. Fie numărul prim $p = 17449$ și numărul 7814. Putem calcula inversul lui 7814 modulo 17449 folosind observația anterioară:

$$7814^{17447} \equiv 7814^{-1} \equiv 1284 \text{ (mod } 17449)$$

Teorema 3.5. (Teorema rădăcinii primitive) Fie p un număr prim, există un element $g \in \mathbb{F}_p^*$, astfel încât puterile lui g generează fiecare element al lui $\mathbb{F}_p^*$.

$$\mathbb{F}_p^* = \left\{1, g, g^2, ..., g^{p-2}\right\}$$

Elementele cu această proprietate se denumesc rădăcinini primitive ale lui $\mathbb{F}_p$ sau generatoare (baze) ale $\mathbb{F}_p$. Ele sunt elemente ale mulțimii finite $\mathbb{F}_p^*$, având ordinul $p - 1$.

Exemplul 3.6. Corpul finit $\mathbb{F}_{11}$ îl are pe 2 ca generator, deoarece în $\mathbb{F}_{11}$

$$\begin{aligned}
&2^0 \equiv 1 \;\; 2^1 \equiv 2 \;\; 2^2 \equiv 4 \;\; 2^3 \equiv 8 \\
&2^4 \equiv 5 \;\; 2^5 \equiv 10 \;\; 2^6 \equiv 9 \;\; 2^7 \equiv 7 \\
&2^8 \equiv 3 \;\; 2^9 \equiv 6
\end{aligned}$$

Deoarece toate elementele nenule ale mulțimii $\mathbb{F}_{11}$ au fost generate din puteri ale lui 2. Însă corpul finit $\mathbb{F}_{17}$ nu îl are pe 2 ca generator, deoarece în $\mathbb{F}_{17}$

$$\begin{aligned}
&2^0 \equiv 1 \;\; 2^1 \equiv 2 \;\; 2^2 \equiv 4 \;\; 2^3 \equiv 8 \\
&2^4 \equiv 16 \;\; 2^5 \equiv 15 \;\; 2^6 \equiv 13 \;\; 2^7 \equiv 9 \\
&2^8 \equiv 1
\end{aligned}$$

Observăm că $2^0 \equiv 2^8 \equiv 1$, deci nu putem crea toate elementele nenule ale mulțimii $\mathbb{F}_{17}$.

Demonstrațiile acestor teoreme (teorema lui Fermat și teorema rădăcinii primitive) sunt mai complexe și sunt prezentate și detaliate în amănunt în [31].

Definiţia 3.7. (Problema logaritmului discret) Fie g o rădăcină primă a $\mathbb{F}_p$ și $h \in \mathbb{F}_p^*$. Problema logaritmului discret este problema identificării exponentului x astfel încât

$$g^x \equiv h \text{ (mod } p)$$

Numărul x se numește logaritmul discret al lui h cu baza în g și se notează $log_g(h)$.

Acest paragraf a fost preluat și adaptat din [18].

Observația 3.8. (a) Fie g o rădăcină primă a $\mathbb{F}_p$, cu p număr prim și $h \in \mathbb{F}_p^*$, problema logaritmului discret presupune identificarea numărului x pentru ca $g^x \equiv h \pmod{p}$. Conform micii teoreme a lui Fermat (teorema 3.1) avem $g^{p-1} \equiv 1 \pmod{p}$, ceea ce înseamnă ca dacă înmulțim g^x cu g ridicat la un multiplu de $(p-1)$ rezutatul va fi tot h. Mai exact, dacă x este soluția căutată, atunci și $x + k(p-1)$ cu $k \in \mathbb{Z}^*$ este o soluție. Adică

$$\begin{aligned} g^{x+k(p-1)} &= g^x \cdot g^{k\cdot(p-1)} \\ &= g^x \cdot (g^{(p-1)})^k \end{aligned}$$

$$g^x \cdot (g^{(p-1)})^k \equiv h \cdot 1^k \equiv h \pmod{p} \text{ pentru orice } k \in \mathbb{Z}^*$$

De aceea precizăm că ne referim la soluția logaritmului discret ca fiind x-ul din intervalul $[0, ...p-2]$.

(b) Este rezonabil să ne referim la log_g ca fiind un „logaritm", deoarece traduce înmulțirea în adunare în același mod ca și funcția normală de logaritm.

Definiţia 3.9. Fie G un grup a cărei lege de compoziție o notăm ca fiind $\star$. Problema logaritmului discret pentru grupul G este problema identificării, pentru orice două elemente h și g cu $h, g \in G$, numărul întreg x astfel încât

$$\underbrace{g \star g \star g \star ... \star g}_{\text{de x ori}} = h$$

3.2 Schimbul de chei Diffie–Hellman

În istoria comunicațiilor secrete, crearea criptografiei cu cheie publică de către Diffie și Hellman în 1976 și urmat de criptosistemul cu chei publice RSA (după inițialele autorilor) realizat de Rivest, Shamir și Adleman în 1978, sunt momente definitorii. Semnificația criptosistemelor cu chei publice și a protocoalelor de semnătură digitală pe care le suportă este greu de estimat în era curentă a computerelor și a internetului.

Unul din obiectivele principale al criptografiei (cu cheia publică) este de a permite ca două persoane să facă schimb de informații confidențiale, chiar dacă nu s-au întâlnit niciodată și pot comunica doar prin intermediul unui canal care este monitorizat de un adversar. Adică de a putea comunica pe un canal nesigur, fără necesitatea întâlnirii anterioare.

Vom defini câteva notații pe care le vom folosi de acum înainte în cadrul acestei teze. Persoanele ce doresc să comunice între ele le vom denumi Alice (va fi notată cu A) și Bob (va fi notat cu B). iar adversarul, cel ce dorește să determine mesajul trimis o vom numi Eve (va fi notată cu E).

Codurile și cifrurile până de curând s-au bazat pe premisa că părțile care încercau să comunice (Bob și Alice) au împărtășit o cheie secretă pe care inamicul lor, adică Eve, nu o știe. Bob și-ar

cripta mesajul folosind cheia secretă și Alice l-ar decripta folosind aceeași cheie secretă, iar Eve nu putea să realizeze decriptarea mesajului, deorece nu are acces la cheia secretă. Totuși există un mare dezavantaj la criptosistemele cu cheie privată, și acela este că Alice și Bob trebuie să stabilească cheia secretă a priori conversației. Acesta este și avantajul principal al criptografiei cu cheie publică, Alice și Bob pot comunica pe un canal nesigur, chiar dacă nu au avut contact direct înainte. Aceste criptosisteme se bazează pe probleme grele din matematică, precum problema logaritmului discret prezentată anterior, pentru a asigura securitatea comunicării. Mai precis, se bazează pe probleme ce sunt foarte complicat de rezolvat fără nicio informație suplimentară, dar care se pot rezolva ușor cu acestea.

Diffie și Hellman au folosit dificultatea problemei logaritmului discret peste $\mathbb{F}_p^*$ pentru a realiza schimbul de chei între cei ce comunică.

Schimbul de chei Diffie-Hellman este un protocol criptografic care permite celor două părți participante să stabilească o cheie secretă partajată pe un canal de comunicare nesigur. A fost introdus de Whitfield Diffie și Martin Hellman și este utilizat pe scară largă pentru stabilirea securizată a cheilor în diferite sisteme criptografice. Vom determina importanța existenței acestui algoritm în capitolele viitoare în cadrul criptografiei bazate pe curbe eliptice.

Acesta este algoritmul ce se aplică pentru schimbul de chei Diffie-Hellman:

(1) Configurare: cele două părți de comunicare, Alice și Bob, convin asupra anumitor parametri:

- un număr prim foarte mare pe care o să-l notăm cu p
- un număr întreg nenul g modulo p

  notă: numărul g ar trebui să fie ales astfel încăt ordinul său în $\mathbb{F}_p^*$ să fie un număr prim mare

  notă: cum comunicarea se realizează pe un canal neprivat, Eve are acces la p și g

(2) Alegerea numerelor secrete:

- Alice alege un număr întreg secret a pe care nu-l împărtașește cu nimeni și calculează

$$A \equiv g^a \pmod{p}$$

- Bob alege un număr întreg secret b pe care nu-l împărtașește cu nimeni și calculează

$$B \equiv g^b \pmod{p}$$

(3) Schimbul de chei publice: Alice îi trimite A lui Bob și Bob îi trimite B lui Alice pe canalul neprivat:

- Alice își folosește numărul secret și calculează

$$A' \equiv B^a \pmod{p}$$

- Bob își folosește numărul secret și calculează

$$B' \equiv A^b \pmod{p}$$

notă: cum comunicarea se realizează pe un canal neprivat, Eve are acces la A și B

(4) Cheia secretă comună rezultată: valorile pe care le calculează sunt egale și reprezintă cheia comună. Vom arăta acest fapt și în exemplul următor:

$$A' \equiv B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \equiv B' \pmod{p}$$

Schimbul de chei Diffie-Hellman este sumarizat în figura 17 (preluată din [18]).

Public Parameter Creation	
A trusted party chooses and publishes a (large) prime p and an integer g having large prime order in $\mathbb{F}_p^*$.	
Private Computations	
Alice	**Bob**
Choose a secret integer a.	Choose a secret integer b.
Compute $A \equiv g^a \pmod{p}$.	Compute $B \equiv g^b \pmod{p}$.
Public Exchange of Values	
Alice sends A to Bob $\longrightarrow$	A
B $\longleftarrow$	Bob sends B to Alice
Further Private Computations	
Alice	**Bob**
Compute the number $B^a \pmod{p}$.	Compute the number $A^b \pmod{p}$.
The shared secret value is $B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \pmod{p}$.	

Figura 17: Schimbul de chei Diffie-Hellman

Exemplul 3.10. Vom urma pașii din algoritmul prezentat anterior.

Pasul 1, Alice și Bob stabilesc numărul $p = 839$ și rădăcina primitivă $g = 113$.

Pasul 2, Alice alege numărul întreg secret $a = 577$ și calculează

$$A = 434 \equiv 113^{577} \pmod{839}$$

Analog, Bob alege numărul întreg secret $b = 349$ și calculează

$$B = 767 \equiv 113^{349} \pmod{839}$$

Pasul 3, Alice îi trimite 434 (A) lui Bob și Bob îi trimite 767 (B) lui Alice pe canalul nesecurizat.
Alice își folosește numărul secret și calculează

$$A' = 627 \equiv 767^{577} \pmod{839}$$

Bob își folosește numărul secret și calculează

$$B' = 627 \equiv 434^{349} \pmod{839}$$

Pasul 4, Alice și Bob au obținut cheia secretă comună

$$627 \equiv 113^{577 \cdot 349} \equiv A^b \equiv B^a \pmod{839}$$

Eve are acces la următoarele date: $839(f)$, $113(g)$, $390(A)$ și $691(B)$. Ceea ce înseamnă ca dacă Eve dorește să determine cheia secretă comună a lui Alice și Bob, aceasta trebuie să rezolve una din cele două congruențe pentru a determina numărul secret a al lui Alice sau b al lui Bob:

$$113^a \equiv 434 \pmod{839}$$
$$113^b \equiv 767 \pmod{839}$$

Observăm că ecuațiile anterioare reprezintă problema logaritmului discret (în Definiția 3.7). Pentru a găsi rezolvarea, Eve trebuie să dea ”brute force”, încercând toate puterile lui 113 modulo 839 pentru a găsi una din cele două congruențe. În practică, se recomanda ca p să aibă mai mult de 1000 de cifre în compoziția sa și g să aibă ordin un număr prim aproximativ jumătate din p, pentru a îngreuna rezolvarea problemei logaritmului discret prin luarea tuturor valorilor.

Astfel, problema logaritmului discret oferă securitate schimbului de chei Diffie-Hellman.

Următorul paragraf a fost preluat și adaptat din [18].

Dacă Eve nu poate rezolva problema logaritmului discret, ar părea că Alice și Bob sunt în siguranță, dar acest lucru nu este neapărat adevărat. Rezolvarea problemei logaritmului discret este o modalitate de a determina valoarea comună a lui Alice și Bob, dar aceasta nu este problema specifică pe care Eve vrea să o rezolve, aceasta încercâd să determine valoarea $g^{ab} \pmod{p}$. Securitatea cheii comune transmise între Alice și Bob este dată de dificultatea cu care se rezolvă următoarea problemă, ce este potențial mai ușoară.

Definiţia 3.11. (Problema Diffie-Hellman) Fie p un număr prim și g un număr întreg. Problema Diffie-Hellman, sau Diffie–Hellman Problem (DHP) în engleză, este problema determinării valorii $g^{ab} \pmod{p}$ cunoscând $g^a \pmod{p}$ și $g^b \pmod{p}$.

Observația 3.12. Presupunem că există un algoritm eficient care să rezolve Problema Diffie-Hellman,

momentan nu se știe dacă se poate folosi acest algoritm pentru a rezolva eficient problema logaritmului discret.

3.3 Criptosistemul cu cheie publică ElGamal

Chiar dacă din punct de vedere istoric, RSA a fost primul criptosistem cu cheie publică, dezvoltarea naturală a unui criptosistem în urma lucrării realizate de Diffie și Hellman (Lucrarea [7]) este un sistem descris de Taher ElGamal în 1985 în [15]. Algoritmul ElGamal pentru criptarea de chei publice se bazează pe problema logaritmului discret și se aseamănă cu schimbul de chei Diffie-Hellman prezentat în subcapitolul anterior. În continuare vom detalia criptosistemul cu cheie publică El Gamal ce se bazează pe problema logaritmului discret pe $\mathbb{F}_p^*$.

Acesta este algoritmul dacă Bob dorește să-i trimită un mesaj lui Alice:

(1) Configurare: se stabilesc parametrii publici:

- un număr prim foarte mare pe care o să-l notăm cu p
- un număr întreg nenul g modulo p

  notă: numărul g ar trebui să fie ales astfel încât ordinul său în $\mathbb{F}_p^*$ să fie un număr prim mare

  notă: cum comunicarea se realizează pe un canal neprivat, Eve are acces la p și g

(2) Crearea cheii secrete:

- Alice alege o cheie secretă a din intervalul $[1, 2, ..., p-1]$ pe care nu-l împărtașește cu nimeni
- Alice calculează cheia publică A

$$A \equiv g^a \pmod{p}$$

- Alice își publică cheia publică

(3) Criptarea:

- Bob alege mesajul m pe care dorește să-l trmită
- Bob alege o cheie aleatoare temporală k
- Bob folosește cheia publică a lui Alice pentru a calcula

$$c_1 \equiv g^k \pmod{p}$$

Respectiv cea de-a doua componentă a mesajului criptat

$$c_2 \equiv mA^k \pmod{p}$$

- Bob îi trimite mesajul criptat (c_1, c_2) lui Alice

  notă: cum comunicarea se realizează pe un canal neprivat, Eve are acces la (c_1, c_2)

(4) Decriptarea:

- Alice calculează

$$(c_1^a)^{-1} \cdot c_2 \pmod{p}$$

- rezultatul obținut este m

Algoritmul ElGamal pentru creare de chei, criptare și decriptare este sumarizat în figura 18 (preluată din [18]).

Public Parameter Creation	
A trusted party chooses and publishes a large prime p and an element g modulo p of large (prime) order.	
Alice	**Bob**
Key Creation	
Chooses private key $1 \leq a \leq p-1$. Computes $A = g^a \pmod{p}$. Publishes the public key A.	
Encryption	
	Chooses plaintext m. Chooses random ephemeral key k. Uses Alice's public key A to compute $c_1 = g^k \pmod{p}$ and $c_2 = mA^k \pmod{p}$. Sends ciphertext (c_1, c_2) to Alice.
Decryption	
Compute $(c_1^a)^{-1} \cdot c_2 \pmod{p}$. This quantity is equal to m.	

Figura 18: Algoritmul ElGamal pentru creare de chei, criptare și decriptare

Arătăm faptul că rezultatul ecuației $(c_1^a)^{-1} \cdot c_2 \pmod{p}$ obținut de Alice chiar este mesajul m în exemplul următor.

Exemplul 3.13. Vom urma pașii din algoritmul prezentat anterior.

Pasul 1, se stabilesc $p = 719$ și rădăcina primitivă $g = 11$.

Pasul 2, Alice alege numărul întreg secret $a = 232$ și calculează

$$A = 673 \equiv 11^{232} \pmod{719}$$

Alice publică cheia publică A.

Pasul 3, Bob alege mesajul $m = 269$, cheia aleatoare temporală $k = 586$ și criptează mesajul

$$\begin{aligned} c_1 &= 566 \equiv 11^{586} \pmod{719} \\ c_2 &\equiv 269 \cdot 673^{586} \pmod{719} \\ &\equiv 269 \cdot 214 \equiv 46 \pmod{719} \end{aligned}$$

Bob îi trimite lui Alice mesajul criptat $(c_1, c_2) = (566, 46)$.

Pasul 4, Alice decriptează mesajul primit de la Bob

$$\begin{aligned} (c_1^a)^{-1} \cdot c_2 &\equiv (566^{232})^{-1} \cdot 46 \pmod{719} \\ &\equiv 214^{-1} \cdot 46 \pmod{719} \\ &\equiv 84 \cdot 46 \equiv 269 \pmod{719} \end{aligned}$$

Eve are acces la următoarele date: numărul prim 719 (p), rădăcina primitivă 11 (g), cheia pu-blică 673 (A) și mesajul criptat format din 566 (c_1), respectiv 46 (c_2). Dacă Eve dorește să afle mesajul decriptat m pe care Bob i l-a trimis lui Alice, atunci Eve trebuie să rezolve $A \equiv g^a \pmod{p}$. Observăm că Eve trebuie să rezolve problema logaritmului discret ca să determine valoarea mesajului transmis.

Următoarea observație, propoziție și demonstrație au fost preluate și adaptate din [18].

Observația 3.14. În criptosistemul de chei publice ElGamal, mesajul inițial m pe care Bob dorește să-l trimită lui Alice este un număr din intervalul $[2, 3, ..., p-1]$. Numerele din componența mesajului criptat c_1 și c_2 sunt tot din intervalul $[2, 3, ..., p-1]$ și de o lungime asemănătoare cu m. Astfel, pentru fiecare mesaj pe care Bob dorește să-l trimită lui Alice lungimea textului criptat va fi aproximativ de două ori mai mare. De aceea, spunem că ElGamal are o expansiune a mesajului inițial de $2\ la\ 1$.

Legat de securitatea criptosistemului ElGamal, am vrea ca acesta să fie cel puțin la fel de greu de atacat de către Eve ca Problema Diffie-Hellman (Problema 3.11). Adică, mai exact am vrea să demonstrăm că dacă Eve poate sparge criposistemul ElGamal, aceasta poate rezolva Problema Diffie-Hellman.

Propoziţia 3.15. *Fie un număr prim p și o rădăcină g pe care le folosim pentru criptarea ElGamal. Să presupunem că Eve are acces la un oracol care decriptează ElGamal texte cifrate criptate arbitrare folosind chei publice arbitrare ElGamal. Atunci ea poate folosi oracolul pentru a rezolva Problema Diffie–Hellman (Problema 3.11).*

Definiţia 3.16. Un atac în care Eve are acces la un oracol care decriptează texte cifrate arbitrare sunt cunoscute ca un ”chosen ciphertext attack”.

4 Criptografia pe curbe eliptice (ECC)

În comparație cu alte metode criptografice, curbele eliptice au o serie de avantaje, cum ar fi dimensiuni reduse ale cheilor, calcule mai rapide și securitatea mai bună. Aceste beneficii fac din criptografia pe curbe eliptice o alegere excelentă pentru operații criptografice eficiente și sigure în diverse aplicații, cum ar fi comunicarea securizată, criptarea și semnăturile digitale.

4.1 Problema logaritmului discret pe curbe eliptice (ECDLP)

Problema logaritmului discret pe curbe eliptice, sau Elliptic Curve Discrete Logarithm Problem (ECDLP) în engleză, este o problemă matematică ce formează baza securității pentru criptografia pe curbe eliptice. Rezolvarea problemei logaritmului discret pe curbe eliptice este considerată dificilă din punct de vedere computațional, ceea ce este esențial pentru securitatea algoritmilor criptografici bazați pe curbe eliptice, cum ar fi schimburile de chei (de exemplu, schimbul de chei Diffie-Hellman pe curbe eliptice), schemele de criptare (de exemplu, criptosistemul ElGamal) și semnăturile digitale (de exemplu, ECDSA).

Am prezentat bazele problemei logaritmului discret în capitolul 2 (Definițiile 3.7 și 3.9). Pentru a crea un criptosistem bazat pe problema logaritmului discret peste corpul finit $\mathbb{F}_p^*$ adaptăm problema logaritmului discret inițială astfel, Alice publică două numere g și h, cu $g, h \in \mathbb{Z}^*$ și exponentul secret x pentru care

$$g^x \equiv h \pmod{p}$$

Sau dacă explicităm ecuația

$$\underbrace{g \cdot g \cdot g \cdot ... \cdot g}_{\text{de x ori}} \equiv h \pmod{p}$$

Aceasta este ecuația pe care Eve trebuie să o rezolve pentru a compromite sistemul.

Adaptăm problema logaritmului discret pentru a putea fi folosită pe curbe eliptice. Fie curba eliptică E peste corpul finit $\mathbb{F}_p$ și grupul de puncte $E(\mathbb{F}_p)$ al acestei curbe. Alice publică două puncte $P(x_1, y_1)$ și $Q(x_2, y_2)$ cu $P, Q \in E(\mathbb{F}_p)$ și numărul întreg n pentru care

$$Q = \underbrace{P + P + P + ... + P}_{\text{n adunări de puncte pe curba eliptică E}} = nP$$

În această ecuație, Eve trebuie să afle de câte ori P trebuie adunat cu el însuși pentru a obține Q.

Definiţia 4.1. (Problema logaritmului discret peste curbe eliptice) Fie E o curbă eliptică peste corpul finit $\mathbb{F}_p$ și fie punctele P și Q, de coordonate (x_1, y_1), respectiv de coordonate (x_2, y_2), cu

$P, Q \in E(\mathbb{F}_p)$. Problema logaritmului discret peste curbe eliptice, sau Elliptic Curve Discrete Logarithm Problem (ECDLP) în engleză, este problema identificării numărului întreg n astfel încât

$$Q = nP$$

Analog cu problemei logaritmului discret peste $\mathbb{F}_p^*$, notăm n ca fiind

$$n = log_P(Q)$$

Numărul întreg n se numește logaritmul discret eliptic al lui Q în raport cu P.

Acest paragraf a fost preluat și adaptat din [18].

Observația 4.2. (a) Definiția pentru $log_P(Q)$ nu este tocmai precisă. Este posibil să existe tupluri de puncte $P, Q \in E(\mathbb{F}_p)$ pentru care Q să nu fie un multiplu de P. În acest caz funcția $log_P(Q)$ nu este definită (asta ar fi însemnat ca toate punctele să fie de ordin p). Totuși, din punct de vedere criptografic, inițial, Alice alege punctul public P și numărul secret n și publică punctul $Q = nP$. În acest caz $log_P(Q)$ există mereu.

(b) Există mai multe valori ale numărului n pentru care $Q = nP$. Definim cel mai mic număr întreg pozitiv s ($s \in \mathbb{Z}_+^*$) pentru care $sP = O$ ca fiind ordinul punctului P. Fie n_0 un punct pentru care $Q = n_0P$, atunci

$$\begin{aligned}(n_0 + is)P &= n_0P + isP \\ &= n_0P + O \\ &= Q \text{ pentru orice } i \in \mathbb{Z}^*\end{aligned}$$

(c) Considerând observația anterioară, putem spune că $log_P(Q)$ este un element din $\mathbb{Z}/s\mathbb{Z}$, deoarece $log_P(Q)$ este un număr întreg modulo s. Avantajul pentru a defini valoarea în $\mathbb{Z}/s\mathbb{Z}$ este că logaritmul discret eliptic satisface egalitatea

$$log_P(Q_1 + Q_2) = log_P(Q_1) + log_P(Q_2) \text{ pentru orice } (Q_1, Q_2) \in E(\mathbb{F}_p)$$

Exemplul 4.3. Fie curba eliptică

$$E : Y^2 = X^3 + 2X + 9 \text{ peste } \mathbb{F}_{127}$$

Și punctele $P = (34, 30)$ și $Q = (99, 37)$, cu $P, Q \in E(\mathbb{F}_{127})$. Calculăm

$$Q = 12P \Rightarrow log_P(Q) = 12$$

Analog, punctele $R = (91, 115)$ și $S = (98, 33)$, cu $R, S \in E(\mathbb{F}_{127})$. Calculăm

$$R = 25P \Rightarrow log_P(R) = 25$$
$$S = 41P \Rightarrow log_P(S) = 41$$

Menționăm că este o posibilitate să existe numere care să nu fie multipli ai lui P. Această speță se întâmplă când ordinul punctului P este diferit de ordinul curbei eliptice (adică punctul P nu este generator). În exemplul nostru ordinul lui P este 72, iar ordinul curbei eliptice este 144, deci există puncte în $E(\mathbb{F}_{127})$ care să nu fie multipli ai lui P.

4.2 Securitatea oferită de ECDLP

Securitatea oferită de problema logaritmului discret pe curbe eliptice este un aspect esențial al criptografiei pe curbe eliptice, fără de care acest domeniu nu ar putea exista. Problema logaritmului discret pe curbe eliptice poate fi considerat fundația pentru securitatea multor algoritmi criptografici bazați pe curbe eliptice, cum ar fi schimburile de chei (de exemplu, schimbul de chei Diffie-Hellman pe curbe eliptice), schemele de criptare (de exemplu, criptosistemul ElGamal) și semnăturile digitale (de exemplu, ECDSA).

Cel mai rapid algoritm cunoscut momentan pentru rezolvarea problemei logaritmului discret pe curbe eliptice durează aproximativ $\sqrt{o}$ pași, unde o este ordinul curbei eliptice. Dar din teorema lui Hasse (teorema 2.23) $o \approx p + 1$.

Pentru a prezenta securitatea acestei probleme vom lua un exemplu. Considerăm că avem o curbă eliptică modulo un număr pe 256 de biți, atunci conform teorei lui Hasse și ordinul va fi tot aproximativ un număr pe 256 de biți. Extragem radicalul din acest număr și va rezulta un număr pe aproximativ 128 de biți, care va fi $\sqrt{o} \approx 10^{37}$. Un CPU de un GHz are o putere de procesare de un miliard de operații pe secundă, ceea ce înseamnă că ar dura $10^{37}/10^9 = 10^{28}$ secunde pentru a sparge ECDLP. Convertim secundele în ani și obținem $10^{28}/(60 \cdot 60 \cdot 24 \cdot 365) = 10^{28}/(3.1 \cdot 10^7) \approx 3.2 \cdot 10^{21}$ ani. Pentru a pune în perspectivă acest număr universul are $13.7 \cdot 10^9$ ani, deci putem trage concluzia că problema logaritmului discret pe curbe eliptice este foarte sigură.

Există mai mulți algoritmi ce pot rezolva problema logaritmului discret pe curbe eliptice în $O(\sqrt{o})$, eficiența acestora depinzând de natura curbei eliptice. Unul dintre aceștia este algoritmul pași de copil pași de gigant realizat de Shanks, sau Baby-step giant-step în engleză. Acest algoritm este de tipul "meet in the middle" și poate fi folosit atât pentru a rezolva ECDLP cât și ajustat pentru a determina cardinalul $E(\mathbb{F}_p)$ (algoritmul pentru determinarea cardinalului poate fi îmbunătățit până la o complexitate de $O(\sqrt[4]{o})$). Pentru a determina $Q = nP$ și fie $m = \lceil \sqrt{o} \rceil + 1$ (adică parte întreagă),

acest algoritm încearcă să indentifice o coliziune între următoarele două liste:

$$\text{Lista1}: \quad P, 2P, 3P, ..., (m-1)P$$
$$\text{Lista2}: \quad mP + Q, 2mP + Q, 3mP + Q, ..., (m-1)^2P + Q$$

Notăm indicele primei liste cu j și pe celei de-a doua liste cu k, forma lui n va fi $n = km + j$. O coliziune ar însemna:

$$Q = nP = (km + j)P$$
$$Q = kmP + jP$$

Observăm că arată aproape exact ca listele, doar că adunăm kmP pe partea greșită de egal, astfel valoarea punctului va fi diferită în coordonata y, dar coordonatele x vor fi la fel și vom putea verifica după acestea coliziunea. După acest algoritm $n = j + km$, pentru care $Q = nP$, problema este că s-ar putea să nu fie cel mai mic număr pentru care se verifică egalitatea, de aceea pentru a afla cel mai mic număr putem calcula n modulo ordinul punctului P în curba eliptică.

Definiţia 4.4. Dacă cel mai bun atac ce se poate realiza asupra unui algoritm pentru a-l compromite necesită 2^n pași, atunci spunem că acest algoritm deține un nivel de securitate de n biți.

Tabelul 2 (preluat din [16]) prezintă nivelul de securitate al criptosistemelor normale, comparativ cu cele eliptice.

Criptosistem	Nivel de securitate (biți)			
	80	128	192	256
DH	1024 biți	3072 biți	7680 biți	15360 biți
ElGamal	1024 biți	3072 biți	7680 biți	15360 biți
DSA	1024 biți	3072 biți	7680 biți	15360 biți
RSA	1024 biți	3072 biți	7680 biți	15360 biți
ECDH	160 biți	256 biți	385 biți	512 biți
EC ElGamal	160 biți	256 biți	385 biți	512 biți
ECDSA	160 biți	256 biți	385 biți	512 biți

Tabelul 2: Nivelul de securitate a diferite criptosisteme

Observația 4.5. (a) Observăm că criptografia pe curbe elitpice oferă o securitate mult mai bună pentru dimensiuni mai mici ale chielor, astfel oferind performanță din punct de vedere al spațiului și al timpul în care se realizează dferite operații pe acestea. Datorită acestor motive, criptografia pe curbe elitpice este preferată în diferite domenii cum ar fi Blockchain și dispozitive mobile.

(b) Criptosistemele eliptice cu chei pe 256 de biți oferă o securitate la fel de bună precum criptosistemele clasice cu chei pe 15360 biți bazate pe problema logaritmului discret.

(c) În practică se folosesc criptosistemele eliptice cu chei pe cel puțin 256 de biți, ceea ce am prezentat anterior că denotă o securitate performantă.

Două dintre cele mai populare și folosite curbe eiptice sunt secp256k1 și secp521r1. Acestea sunt folosite predominant pentru schimbul eliptic de chei Diffie Hellman (ECDH) și pentru algoritmul eliptic de semnare digitală (ECDSA).

Secp256k1 este o curbă eliptică utilizată frecvent în criptosisteme, în special în contextul domeniului Blockchain. A fost creată de către Standards for Efficient Cryptography Group (SECG), iar motivul pentru care aceasta este cunoscută se datorează faptului că este curba eliptică folosită de către Bitcoin. Conform [2], secp256k1 este adesea cu peste 30% mai rapidă decât alte curbe, cât timp implementarea este optimizată. Numărul 256 din denumire face referință la numărul de biți folosiți, mai multe detalii tehnice se pot găsi aici: [19].

Secp521r1, cunoscută și sub numele de NIST P-521, este una dintre curbele eliptice definite de către National Institute of Standards and Technology (NIST) pentru utilizarea în criptografia cu curbe eliptice peste corpuri finite. Denumirea secp521r1 provine de la numărul prim, care este pe 521 de biți. Astfel, datoriă ordinului mare al curbei eliptice, securitatea acesteia împotriva a diferite atacuri criptografice este foarte ridicată. Mai multe detalii tehnice se pot găsi aici: [20].

4.3 Algoritmul dublare și adunare

Algoritmul dublare și adunare, sau Double and Add în engleză, este o metodă standard folosită pentru a efectua înmulțirea unui punct de pe o curbă eliptică peste un corp finit cu un scalar. Adică dorim să calculăm $Q = nP$ în $E(\mathbb{F}_p)$ cât mai eficient.

Acesta este algoritmul dublare și adunare:

(1) Configurare: se stabilesc următoarele date:

- o curbă eliptică E peste $\mathbb{F}_p$
- un punct $P \in \mathbb{F}_p$
- un scalar pe care o să-l notăm n, cu $n \in \mathbb{N}^*$

(2) Transformarea scalarului în forma sa binară:

- Orice scalar are o forma binară unică

$$n = n_0 + n_1 \cdot 2 + n_2 \cdot 2^2 + ... + n_k \cdot 2^k$$
$$cu\ n_0, n_1, ..., n_r \in \{0, 1\}$$

(3) Inițializăm variabilele locale:

- inițializăm valoarea $Q = P$, ce va fi $P \cdot 2^{\text{iterația curentă}}$
- inițializăm valoarea $R = O$, ce va fi rezultatul final

(4) Iterăm prin toți biții lui n, începând de la cel mai nesemnificativ:

- dacă $n \bmod 2 = 1$, $R = R + Q$
- dublăm valoarea lui Q, $Q = 2Q$
- dacă mai există biți ai lui n, trecem la următorul bit al lui n

(5) Rezultatul final va fi R, care va fi $R = nP$.

Algoritmul dublare și adunare este sumarizat în pseudocod în figura 19 (preluată din [18]).

Input. Point $P \in E(\mathbb{F}_p)$ and integer $n \geq 1$.
1. Set $Q = P$ and $R = \mathcal{O}$.
2. Loop while $n > 0$.
  3. If $n \equiv 1 \pmod 2$, set $R = R + Q$.
  4. Set $Q = 2Q$ and $n = \lfloor n/2 \rfloor$.
  5. If $n > 0$, continue with loop at Step **2**.
6. Return the point R, which equals nP.

Figura 19: Algoritmul dublare și adunare

Exemplul 4.6. Fie numărul prim $p = 2399$ și curba eliptică E

$$E : Y^2 = X^3 + X + 11 \pmod{2399}$$

și punctul $P = (196, 41) \in E(\mathbb{F}_{2399})$ și scalarul $n = 1187$.

Algoritmul dublare și adunare este prezentat pas cu pas în figura 20.

Step i	n	$Q = 2^i P$	R
0	1187	$(196, 41)$	(inf, inf)
1	593	$(780, 848)$	$(196, 41)$
2	296	$(674, 624)$	$(594, 1274)$
3	148	$(1260, 2140)$	$(594, 1274)$
4	74	$(1398, 913)$	$(594, 1274)$
5	37	$(1195, 1089)$	$(594, 1274)$
6	18	$(1234, 2344)$	$(923, 1837)$
7	9	$(1550, 567)$	$(923, 1837)$
8	4	$(509, 689)$	$(2385, 933)$
9	2	$(1474, 1715)$	$(2385, 933)$
10	1	$(312, 793)$	$(2385, 933)$
11	0	$(854, 1176)$	$(1062, 368)$

Figura 20: $1187 \cdot (196, 41)$ pe $Y^2 = X^3 + X + 11 \pmod{2399}$

Algoritmul necesită 11 dublaje și 5 adunări, iar rezultatul final este $1187P = (1062, 368)$.

În figura 21 (preluată din [26]) se poate vedea diferența de pași între adunare succesivă și dublare și adunare.

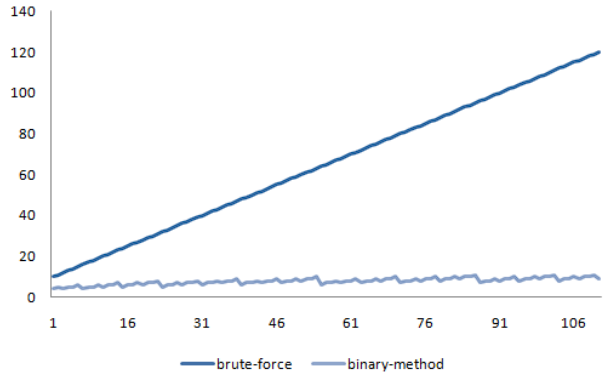


Figura 21: Adunare succesivă vs. dublare și adunare

Daca scalarul n este mult mai mare decât numarul prim p al curbei eliptice, putem prima dată să determinăm oridinul o al punctului P și apoi să înmulțim punctul doar cu n modulo o.

4.4 Schimbul de chei Diffie–Hellman pe curbe eliptice (ECDH)

În acest capitol vom prezenta prima integrare a curbelor eliptice în criptografie. Începem cu schimbul de chei Diffie-Hellman pe curbe eliptice sau Elliptic Curve Diffie-Hellman (ECDH) în engleză, care este un protocol de schimb de chei bazat pe criptografia cu curbe eliptice. ECDH permite ca două părți să convină în siguranță asupra unei chei secrete partajate pe un canal de comunicare nesigur. Realizăm schimbul de chei Diffie-Hellman pe curbe cliptice analog cu cel normal, dar schimbăm problema logaritmului discret peste corpul finit $\mathbb{F}_p$ cu problema logaritmului discret pe curba eliptică $E(\mathbb{F}_p)$.

Vom trece prin aceleași etape ca la schimbul de chei Diffie-Hellman normal.

Acesta este algoritmul ce se aplică pentru schimbul de chei Diffie-Hellman pe curbe eliptice:

(1) Configurare: cele două părți ce comunicare, Alice și Bob, convin asupra anumitor parametri:

- un număr prim foarte mare pe care o să-l notăm cu p
- o curbă eliptică E peste $\mathbb{F}_p$
- un punct $P \in \mathbb{F}_p$

  notă: cum comunicarea se realizează pe un canal neprivat, Eve are acces la p, E și P

(2) Alegerea numerelor secrete:

- Alice alege un număr întreg secret n_A pe care nu-l împărtășește cu nimeni și calculează

$$Q_A = n_A P$$

- Bob alege un număr întreg secret n_B pe care nu-l împărtășește cu nimeni și calculează

$$Q_B = n_B P$$

(3) Schimbul de chei publice: Alice îi trimite Q_A lui Bob și Bob îi trimite Q_B lui Alice pe canalul neprivat:

- Alice își folosește numărul secret și calculează

$$Q'_A = n_A Q_B$$

- Bob își folosește numărul secret și calculează

$$Q'_B = n_B Q_A$$

notă: cum comunicarea se realizează pe un canal neprivat, Eve are acces la Q_A și Q_B

(4) Cheia secretă comună rezultată: valorile pe care le calculează sunt egale și reprezintă cheia comună:

$$Q'_A = n_A Q_B = n_A(n_B P) = n_B(n_A P) = n_B Q_A = Q'_B$$

Schimbul de chei Diffie-Hellman pe curbe eliptice este sumarizat în figura 22 (preluată din [18]).

Public Parameter Creation	
A trusted party chooses and publishes a (large) prime p, an elliptic curve E over $\mathbb{F}_p$, and a point P in $E(\mathbb{F}_p)$.	
Private Computations	
Alice	**Bob**
Chooses a secret integer n_A.	Chooses a secret integer n_B.
Computes the point $Q_A = n_A P$.	Computes the point $Q_B = n_B P$.
Public Exchange of Values	
Alice sends Q_A to Bob $\longrightarrow$	Q_A
Q_B $\longleftarrow$	Bob sends Q_B to Alice
Further Private Computations	
Alice	**Bob**
Computes the point $n_A Q_B$.	Computes the point $n_B Q_A$.
The shared secret value is $n_A Q_B = n_A(n_B P) = n_B(n_A P) = n_B Q_A$.	

Figura 22: Schimbul de chei Diffie-Hellman pe curbe eliptice

Și prezentat în diagrama din figura 23 (preluată și modificată din [4]).

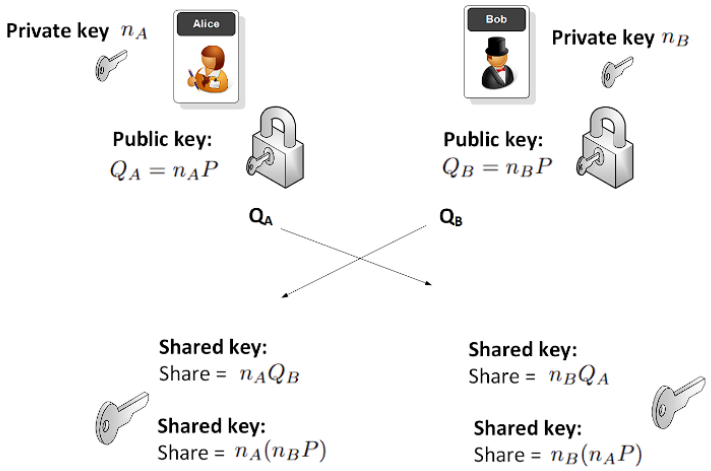


Figura 23: Diagrama schimbului de chei Diffie-Hellman pe curbe eliptice

Schimbul de chei eliptic Diffie-Hellman oferă mai multe avantaje în ceea ce privește eficiența și securitatea în comparație cu alte schimburi de chei. Acesta oferă securitate puternică cu chei de lungimi mai scurte, făcându-l eficient din punct de vedere computațional. Protocolul este utilizat pe scară largă în diferite sisteme și protocoale criptografice, inclusiv în protocoale de comunicații sigure, cum ar fi TLS (Transport Layer Security) pentru stabilirea unei conexiuni sigure pe internet.

Exemplul 4.7. Vom urma pașii din algoritmul prezentat anterior.

Pasul 1, Alice și Bob stabilesc numărul $p = 4219$ și curba eliptică E și punctul P, cu $P \in \mathbb{F}_p$

$$E : Y^2 = X^3 + 268X + 1344 \text{ (mod } 4219)$$
$$P = (940, 256) \in E(\mathbb{F}_{4219})$$

Pasul 2, Alice alege numărul întreg secret $n_A = 852$ și calculează

$$Q_A = 852P = (2267, 3379) \in E(\mathbb{F}_{4219})$$

Analog, Bob alege numărul întreg secret $n_b = 1407$ și calculează

$$Q_B = 1407P = (3129, 2271) \in E(\mathbb{F}_{4219})$$

Pasul 3, Alice îi trimite $(2267, 3379)$ (Q_A) lui Bob și Bob îi trimite $(3129, 2271)$ (Q_B) lui Alice pe canalul nesecurizat.

Alice își folosește numărul secret și calculează

$$Q'_A = 852Q_B = (677, 2358) \in E(\mathbb{F}_{4219})$$

Bob își folosește numărul secret și calculează

$$Q'_B = 1407Q_A = (677, 2358) \in E(\mathbb{F}_{4219})$$

Pasul 4, Alice și Bob au obținut cheia secretă comună

$$(677, 2358) = n_A n_B P = Q'_A = Q'_B$$

Punctul secret comun este $(3347, 1242)$, dar după cum va fi explicat în observația 4.9, suntem interesați doar de componenta x a punctului, adică cheia secretă comună va fi considerată numărul 677.

Eve are acces la următoarele date: numărul prim p, curba eliptică E, punctul de pe curbă P și valorile Q_A și Q_B. Ceea ce înseamnă că dacă Eve dorește să determine cheia secretă comună a lui Alice și Bob, aceasta trebuie sa rezolve una din cele două probleme ale logaritmului discret pe curbe eliptice

$$Q_A = n_A P$$
$$Q_B = n_B P$$

Ca la problema logaritmului discret, Eve poate rezolva analogul problemei Diffie-Hellman (Definiția 3.11) adaptată pentru curbe eliptice.

Definiţia 4.8. (Problema Diffie-Hellman pe curbe eliptice) Fie $E(\mathbb{F}_p)$ o curbă eliptică peste un corp finit și fie punctul $P \in E(\mathbb{F}_p)$. Problema Diffie-Hellman pe curbe eliptice este problema determinării valorii $n_1 n_2 P$ cunoscând valorile $n_1 P$ și $n_2 P$.

Următoarul paragraf a fost preluat și adaptat din [18]

Observația 4.9. Schimbul de chei Diffie–Hellman pe curbe eliptice necesită ca Alice și Bob să schimbe puncte pe o curbă eliptică. Un punct $Q \in E(\mathbb{F}_p)$ este format din două coordonate $Q = (x_Q, y_Q)$, unde x_Q și y_Q sunt elemente ale corpului finit $(\mathbb{F}_p)$, așa că pare că Alice trebuie să-i trimită lui Bob două numere în $(\mathbb{F}_p)$. Cu toate acestea, cele două numere modulo p nu conțin la fel de multe informații ca două numere arbitrare, deoarece sunt legate prin formula

$$E : y_Q^2 = x_Q^3 + ax_Q + b \pmod p$$

A se nota că Eve cunoaște A și B, deci dacă poate ghici valoarea corectă a lui x_Q, atunci există doar două valori posibile pentru y_Q, iar în practică se calculează ușor cele două valori ale lui y_Q.

Prin urmare, există puține motive pentru ca Alice să trimită ambele coordonate ale Q_A lui Bob, deoarece coordonata y conține puține informații suplimentare. În schimb, ea îi trimite lui Bob doar

coordonatele x a punctului Q_A. Bob calculează apoi și folosește unul dintre cele două coordonate y posibile. Dacă se întâmplă să aleagă y „corect", atunci el folosește Q_A, iar dacă alege y „incorect" (care este negativul lui y corect), atunci el folosește -Q_A. În orice caz, Bob ajunge să calculeze unul dintre valorile

$$\pm n_B Q_A = \pm(n_A n_B)P$$

În mod similar, Alice ajunge să calculeze unul dintre $\pm(n_A n_B)P$. Apoi Alice și Bob utilizează coordonatele x ca valoare secretă comună, deoarece acea coordonată x este la fel indiferent de ce y folosesc.

4.5 Criptosistemul ElGamal pe curbe eliptice

Criptosistemul ElGamal pe curbe eliptice este o schemă de criptare cu cheie publică bazată pe criptografia pe curbe eliptice. Este o extensie a criposistemului ElGamal (prezentat în capitolul 3.3), adaptat pentru a lucra pe curbe eliptice în locul corpurilor finite tradiționale.

Acesta este algoritmul dacă Bob dorește să-i trimită un mesaj lui Alice:

(1) Configurare: se stabilesc parametrii publici:

- un număr prim foarte mare pe care o să-l notăm cu p
- o curbă eliptică E peste $\mathbb{F}_p$
- un punct P, cu $P \in E(\mathbb{F}_p)$

  notă: cum comunicarea se realizează pe un canal neprivat, Eve are acces la numărul prim p, curba eliptică E peste $\mathbb{F}_p$ și la punctul P

(2) Crearea cheii publice:

- Alice alege o cheie secretă n_A pe care nu o împărtășește cu nimeni
- Alice calculează cheia publică Q_A

$$Q_A = n_A P$$

- Alice își publică cheia publică

(3) Criptarea:

- Bob alege mesajul M, cu $M \in E(\mathbb{F}_p)$, pe care dorește să-l trimită
- Bob alege o cheie aleatoare temporală k

- Bob folosește cheia publică a lui Alice pentru a calcula

$$C_1 = kP \in E(\mathbb{F}_p)$$

Respectiv cea de-a doua componentă a mesajului criptat

$$C_2 = M + kQ_A \in E(\mathbb{F}_p)$$

- Bob îi trimite mesajul criptat (C_1, C_2) lui Alice

notă: cum comunicarea se realizează pe un canal neprivat, Eve are acces la (C_1, C_2)

(4) Decriptarea:

- Alice calculează

$$C_2 - n_A C_1 \in E(\mathbb{F}_p)$$

- rezultatul obținut este M

Algoritmul eliptic ElGamal pentru creare de chei, criptare și decriptare este sumarizat în figura 24 (preluată din [18]).

Public Parameter Creation	
A trusted party chooses and publishes a (large) prime p, an elliptic curve E over $\mathbb{F}_p$, and a point P in $E(\mathbb{F}_p)$.	
Alice	**Bob**
Key Creation	
Chooses a private key n_A. Computes $Q_A = n_A P$ in $E(\mathbb{F}_p)$. Publishes the public key Q_A.	
Encryption	
	Chooses plaintext $M \in E(\mathbb{F}_p)$. Chooses an ephemeral key k. Uses Alice's public key Q_A to compute $C_1 = kP \in E(\mathbb{F}_p)$. and $C_2 = M + kQ_A \in E(\mathbb{F}_p)$. Sends ciphtertext (C_1, C_2) to Alice.
Decryption	
Computes $C_2 - n_A C_1 \in E(\mathbb{F}_p)$. This quantity is equal to M.	

Figura 24: Algoritmul eliptic ElGamal pentru creare de chei, criptare și decriptare

Și prezentat în diagrama din figura 25 (preluată și modificată din [3]).

Exemplul 4.10. Vom urma pașii din algoritmul prezentat anterior.

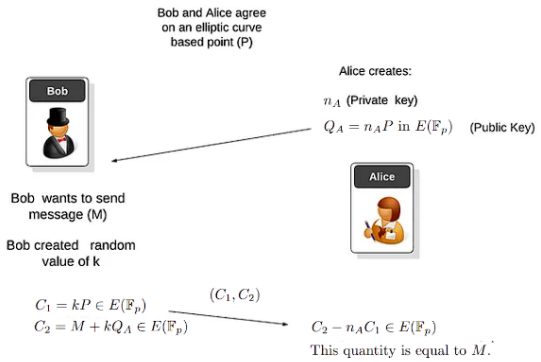


Figura 25: Diagrama criptosistemului eliptic ElGamal

Pasul 1, configurare: Alice și Bob stabilesc numărul $p = 31$ și curba eliptică E și punctul P

$$E : Y^2 = X^3 + X + 13 \text{ (mod } 31)$$
$$P = (9, 10) \in E(\mathbb{F}_{31})$$

Pasul 2, crearea cheii publice: Alice alege numărul întreg secret $n_A = 5$ și calculează

$$Q_A = n_A P = 5(9, 10) = (25, 16) \in E(\mathbb{F}_{31})$$

Alice își publică cheia publică $Q_A = (25, 16)$

Pasul 3, criptarea: Bob alege ca mesajul să fie $M = (20, 2)$ și cheia temporală $k = 7$, Bob folosește cheia publică a lui Alice pentru a calcula

$$\begin{aligned} C_1 &= kP = 7(9, 10) = (6, 24) \\ C_2 &= M + kQ_A = (20, 2) + 7(25, 16) \\ &= (20, 2) + (9, 10) = (22, 22) \end{aligned}$$

Bob îi trimite mesajul criptat $(C_1, C_2) = ((6, 24), (22, 22))$ lui Alice

Pasul 4, decriptarea: Alice decriptează mesajul primit de la Bob

$$\begin{aligned} C_2 - n_A C_1 &= (22, 22) - 5(6, 24) \\ &= (22, 22) - (9, 10) \\ &= (22, 22) + (9, 21) = (20, 2) \end{aligned}$$

Rezultatul $(20, 2)$, fiind punctul M ales de Bob.

Securitatea criptosistemului ElGamal pe curbe eliptice se bazează pe rezolvarea complexă a problemei logaritmului discret pe curbe eliptice, necesitând prea mult timp pentru a fi soluționată.

Confidențialitatea mesajului este păstrată, deoarece un adversar care nu cunoaște cheia privată sau cheia privată temporală nu ar putea să calculeze secretul partajat sau să recupereze mesajul.

De asemenea, de marcat este faptul că criptosistemul ElGamal oferă securitate semantică, ceea ce înseamnă că un atacator nu poate afla nicio informație despre mesajul trimis din textul cifrat fără cunoașterea cheii private. Cu toate acestea, este important faptul că ElGamal nu oferă integritatea mesajului sau autentificare. Pentru a realiza aceste proprietăți, mecanisme suplimentare precum semnăturile digitale sau codurile de autentificare a mesajelor, sau message authentication codes (MAC) în engleză, pot fi utilizate împreună cu criptarea ElGamal pe curbe eliptice.

Există totuși două probleme majore legate de acest criptosistem.

Observația 4.11. (a) Prima problemă este că nu există un mod natural de a atașa un mesaj textual la un punct din $E(\mathbb{F}_p)$, acest fapt duce la dificultăți când trebuie partajate texte. Conform teoremei lui Hasse (teorema 2.23) există aproximativ p puncte diferite în $E(\mathbb{F}_p)$, deci numai p diferite mesaje text ce pot fi criptate. Au fost propuse diferite metode pentru a rezolva această problemă, una dintre acestea fiind implementată în criptosistemul eliptic Menezes-Vanstone, în care textul este mascat de către mesajul M ales și nu scufundat.

(b) A doua problemă este rata de expansiune de 4 la 1 pentru criptosistemul eliptic ElGamal, comparativ cu rata de 2 la 1 criptosistemul ElGamal peste corpuri finite. Acest lucru se datorează trimiterii mesajului criptat (C_1, C_2), unde fiecare element este un punct de pe curba eliptică, deci are două componente. Putem încerca abordarea de la schimbul de chei eliptic Diffie-Hellman, unde trimiteam doar componenta x a punctului, însă în acest caz rezultatele finale sunt diferite ($C_2 - n_A C_1$ are un rezultat diferit de $C_2 + n_A C_1$). Totuși, pentru fiecare coordonată x sunt maxim 2 valori y ce pot exista, deci la componenta x a punctelor C_1, C_2, putem atașa un bit $\in \{0, 1\}$ care să denote ce coordonată y utilizăm în calcule, bitul fiind 0 dacă este mai mică decât jumătatea moduloului folosit de curba eliptică și 1 altfel (cum punctele sunt opuse, înseamnă că adunarea coordonatelor y ale acestora o să fie fix modulo-ul, ceea ce înseamnă că punctele opuse nu pot avea același bit atașat).

4.6 Algoritmul de semnare digitală pe curbe eliptice (ECDSA)

Algoritmul de semnare digitală pe curbe eliptice, sau Elliptic Curve Digital Signature Algorithm (ECDSA) în engleză, este un algoritm de semnătură digitală bazat pe criptografia cu curbe eliptice. Acesta a fost oficial implementat în [1] și permite crearea de semnături digitale care pot fi verificate de orice persoană care știe valoarea cheii publice. ECDSA este utilizat pe scară largă pentru autentificarea mesajelor, integritatea și non-repudierea în diferite criptosisteme.

Acesta este algoritmul dacă Alice semnează un document, iar Bob dorește să verifice semnătura:

(1) Configurare: se stabilesc parametrii publici:

- un număr prim foarte mare pe care o să-l notăm cu p
- o curbă eliptică E peste $\mathbb{F}_p$
- un punct G, cu $G \in E(\mathbb{F}_p)$

  notă: ordinul punctului G ar trebui să fie un număr prim mare, notat q

(2) Crearea cheii de verificare:

- Alice alege o cheie de semnare secretă s pe care nu o împărtășește cu nimeni cu proprietatea că

$$1 < s < q - 1$$

- Alice calculează cheia de verificare publică V

$$V = sG \in E(\mathbb{F}_p)$$

- Alice își publică cheia de verificare publică V

(3) Semnare:

- Alice alege documentul ce va fi semnat d mod q
- Alice alege o cheie temporală e mod q
- Alice calculează punctul R, cu $R \in E(\mathbb{F}_p)$

$$R = eG$$

- Alice procesează semnăturile s_1 și s_2, unde s_1 este componenta x a punctului R

$$s_1 \equiv R.x \pmod{q}$$
$$s_2 \equiv e^{-1}(d + ss_1) \pmod{q}$$

- Alice publică semnătura (s_1, s_2)

(4) Verificare:

- Bob calculează numerele v_1

$$v_1 \equiv ds_2^{-1} \pmod{q}$$

  Și respectiv v_2

$$v_2 \equiv d + s1s_2^{-1} \pmod{q}$$

- Bob calculează punctul R', cu $R' \in E(\mathbb{F}_p)$

$$R' = v_1 G + v_2 V$$

- Bob verifică semnătura după egalitatea componentelor x ale rezultatelor

$$s_1 \equiv R'.x \pmod{q}$$

Algoritmul de semnare digitală pe curbe eliptice este sumarizat în figura 26 (preluată din [18]).

Public Parameter Creation	
A trusted party chooses a finite field $\mathbb{F}_p$, an elliptic curve $E/\mathbb{F}_p$, and a point $G \in E(\mathbb{F}_p)$ of large prime order q.	
Alice	**Bob**
Key Creation	
Choose secret signing key $1 < s < q-1$. Compute $V = sG \in E(\mathbb{F}_p)$. Publish the verification key V.	
Signing	
Choose document d mod q. Choose ephemeral key e mod q. Compute $eG \in E(\mathbb{F}_p)$ and then, $s_1 = x(eG) \bmod q$ and $s_2 \equiv (d + ss_1)e^{-1} \pmod{q}$. Publish the signature (s_1, s_2).	
Verification	
	Compute $v_1 \equiv ds_2^{-1} \pmod{q}$ and $v_2 \equiv s_1 s_2^{-1} \pmod{q}$. Compute $v_1 G + v_2 V \in E(\mathbb{F}_p)$ and verify that $x(v_1 G + v_2 V) \bmod q = s_1$.

Figura 26: Algoritmul de semnare digitală pe curbe eliptice

Și prezentat în diagrama din figura 27 (preluată din [24]).

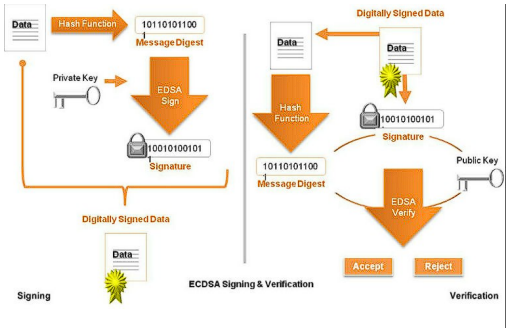


Figura 27: Diagrama algoritmului de semnare digitală pe curbe eliptice

Pentru a se determina valoarea documentului d dintr-un mesaj de tip text m se poate folosi orice funcție hash, vom folosi SHA-256 în exemplele următoare.

Exemplul 4.12. Vom urma pașii din algoritmul prezentat anterior.

Pasul 1, configurare: Alice și Bob stabilesc numărul $p = 3797$ și curba eliptică E și punctul G

$$\begin{aligned} E &: Y^2 = X^3 + 412X + 2356 \text{ (mod } 3797) \\ G &= (2460, 99) \in E(\mathbb{F}_{3797}) \\ q &= \text{ ord}(G) = 1249 \end{aligned}$$

Pasul 2, crearea cheii de verificare: Alice alege cheia secretă se semnare $s = 1024$ și calculează

$$V = sG = 1024(2460, 99) = (1753, 1084) \in E(\mathbb{F}_{3797})$$

Alice își publică cheia de verificare $V = (1753, 1084)$

Pasul 3, semnarea: Alice alege ca mesajul $m =$ "The Book of Five Rings" să fie semnat și determină documentul d ca fiind

$$\begin{aligned} hash(m) &= \text{SHA-256}(m) \\ d &\equiv base_{10}(hash(m)) \text{ (mod } 1249) \\ &\equiv 267 \end{aligned}$$

Datorită faptului că rezultatul este foarte mare nu încape pe ecran, așa că am trecut direct rezultatul, etapele ecuației se pot observa în 5.7.

Alice alege cheia temporală $e = 361$ și calculează punctul R

$$\begin{aligned} R &= eG \\ &= 361 \cdot (2460, 99) \\ &= (3030, 2497) \in E(\mathbb{F}_{3797}) \end{aligned}$$

Și semnăturile s_1 și s_2

$$\begin{aligned} s_1 &\equiv R_x \text{ (mod } 1249) \\ &\equiv 532 \\ s_2 &\equiv e^{-1}(d + ss_1) \text{ (mod } 1249) \\ &\equiv 361^{-1} \cdot (267 + 1024 \cdot 542) \text{ (mod } 1249) \\ &\equiv 932 \end{aligned}$$

Alice publică semnătura $(s_1, s_2) = (532, 932)$.

Pasul 4, verificarea: Bob determină v_1 și v_2

$$\begin{aligned} v_1 &\equiv ds_2^{-1} \pmod{1249} \\ &\equiv 267 \cdot 932^{-1} \pmod{1249} \\ &\equiv 141 \\ v_2 &\equiv s_1 s_2^{-1} \pmod{1249} \\ &\equiv 532 \cdot 932^{-1} \pmod{1249} \\ &\equiv 1137 \end{aligned}$$

Apoi, Bob calculează punctul R'

$$\begin{aligned} R' &= v_1 G + v_2 V \\ &= 141 \cdot (2460, 99) + 1137 \cdot (1753, 1084) \\ &= (329, 2357) + (3292, 645) \\ &= (3030, 2497) \in E(\mathbb{F}_{3797}) \end{aligned}$$

Bob verifică semnătura după egalitatea componentelor x ale rezultatelor

$$\begin{aligned} s_1 &\equiv R'.x \pmod{q} \\ 532 &\equiv 3030 \pmod{1249} \\ 532 &\equiv 532 \end{aligned}$$

Pentru că numerele sunt egale semnătura este validă.

Securitatea algoritmul de semnare digitală pe curbe eliptice se bazează pe dificultatea de rezolvare a problemei logaritmului discret pe curbe eliptice. Cheia privată este utilizată în procesul de semnare, iar cheia publică permite verificarea semnăturii. Schema este concepută astfel încât numai deținătorul cheii private poate produce semnături valide, în timp ce oricine are acces la cheia publică poate verifica semnăturile.

Algoritmul de semnare digitală pe curbe eliptice oferă mai multe avantaje față de alte scheme de semnătură digitală, inclusiv dimensiuni mai scurte ale cheilor și calcule mai eficiente în comparație cu schemele tradiționale bazate pe corpuri finite. Acesta este utilizat pe scară largă în diverse sisteme și protocoale criptografice, inclusiv protocoale de comunicații securizate precum TLS (Transport Layer Security) și certificate digitale pentru verificarea identității în infrastructurile cu chei publice, sau public key infrastructures (PKI) în engleză.

5 Aplicația suport

Aplicația suport pe care am dezvoltat-o poate fi considerată un calculator pentru diferiți algoritmi și protocoale criptografice pe curbe eliptice, dar este și un instrument educativ, prezentând sau simulând fiecare algoritm pas cu pas atât prin teorie, cât și prin calcule explicite bazate pe inputul utilizatorului. Aplicația a fost realizată în engleză, iar toate capturile de ecran din acest capitol au fost realizate integral în aceasta.

5.1 Tehnologii utilizate

În această secțiune vom face o scurtă introducere și detaliere a tehnologiilor folosite în aplicația suport. Câteva dintre acestea sunt: Python, Flask, NumPy, SimPy, Matplotlib, Bootstrap, LaTeX-MathML și MathJax.

Python este un limbaj de programare foarte popular, versatil, ”high-level”, care poate fi utilizat în aproape toate domeniile informaticii, precum dezvoltare web, analiză de date, inteligență artificială, automatizare și multe altele. Python este un limbaj interpretat, ceea ce înseamnă că este executat linie cu linie, fără a fi nevoie de compilare explicită, iar variabilele acestuia sunt dinamice, ceea ce înseamnă că tipurile de variabile sunt determinate în timpul execuției și nu este nevoie să fie instanțiate ca un anumit tip. Am ales acest limbaj pentru a-mi realiza aplicația suport datorită experiențelor anterioare cu acesta în perioada facultății și dorința de a încerca un framework pe care nu l-am mai folosit înainte, dar este folosit la scară largă în industrie: Flask.

Flask este un framework popular și ușor de folosit pentru realizarea de aplicații web utilizând Python. Acesta mai este denumit și ca un microframework, deoarece oferă doar caracteristicile esențiale necesare dezvoltării web. Caracteristicile principale pe care le-am folosit în aplicație suport sunt: rutarea și maparea URL-urilor (Flask folosește un mecanism de rutare pentru a mapa adresele URL la funcțiile corespunzătoare, astfel fiind foarte ușor de folosit, înțeles și scalat) și gestionarea cererilor de tip HTTP (Flask oferă o modalitate accesibilă de a gestiona cererile HTTP și de a accesa datele solicitate, datele de formular și anteturile, fapt pe care se bazează scheletul principal al aplicației dezvoltate). Documentația oficială se regăsește aici [9].

Python are un ecosistem amplu de librării externe, câteva dintre acestea pe care le-am folosit sunt: NumPy, SimPy, Matplotlib, Bootstrap, LaTeXMathML și MathJax.

NumPy (Numeric Python) este o bibliotecă fundamentală în Python pentru calculul numeric și științific pe care am folosit-o extensiv în aplicație. Matplotlib este o bibliotecă Python populară pentru crearea de grafice și vizualizări de date. Aceasta oferă funcționalități standard pentru generarea de grafice în diferite formate și pentru a le putea vizualiza. Aceste două librării au fost folosite împreună pentru a realiza graficele prezentate în aplicație. Când se realiza un request ce necesita un anumit grafic îl cream cu ajutorul a NumPy și Matplotlib, îl salvam în folderul ”static/images”

și returnăm ruta și un boolean, cu ajutorul căruia știam din html că pot încărca din ruta oferită poza dorită. Documentațiile oficiale ale acestor librării se regăsesesc aici [13], [12].

SimPy este o bibliotecă care are diferite module, precum ntheory (Number Theory) ce pot ajuta cu operații pe curbe eliptice, cum ar fi determinarea ordinului unei curbe eliptice. Documentația oficială se regăsește aici [14].

Bootstrap este un framework pentru front-end popular pentru dezvoltarea site-urilor si aplicațiilor web. Acesta oferă la dispoziția dezvoltatorului componente standard ce pot fi folosite, fiind open-source. În aplicația dezvoltată am folosit bootstrap pentru realizarea navbarului (meniului) și modificarea stilului acestuia. Documentația navbarului se regăsește aici [8].

LaTeXMathML și MathJax sunt două biblioteci JavaScript care permit afișarea matematicii în formatul LaTeX în pagini web. Am folosit LaTeXMathML pentru scrierea în LaTeX inline (adică în interiorul unui paragraf) și MathJax pentru realizarea ecuațiilor, acestea fiind ariile la care excelează fiecare bibliotecă. Am scris în aplicație în LaTeX pentru a formatarea profesională a elementor matematice. Documentațiile acestor librării se regăsesesc aici [10], [11].

5.2 Structura proiectului

Structura proiectului este de tip modulară (adică este un proiect organizat pe foldere, în care fiecare folder are propriul său scop în aplicație, iar codul este împărțit în componente logice). Proiectul este împărțit în unități mici, autonome, ceea ce face mai ușor înțelegerea și lucrul cu implementarea realizată. Fiecare modul are propriul set de fișiere și dependențe, astfel, codul fiind organizat și separat pe funcționalități, ceea ce face foarte ușoară scalarea acestuia. Un alt avantaj al acestei structuri este reutilizarea codului în alte componente, aceasta fiind foarte ușoară (trebuie configurate doar dependențele/importurile a componentelor ce se doresc a fi utilizate).

Structura proiectului este prezentată în figura 28.

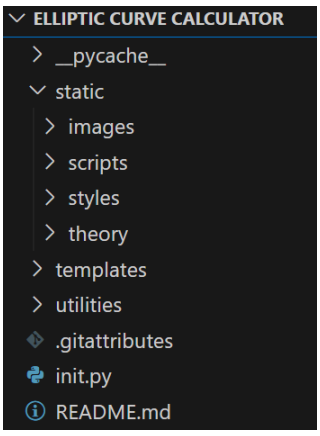


Figura 28: Structura proiectului

Directorul ”static” este împărțit în 4 foldere. Folderul ”images” conține grafurile create în scopul reprezentării geometrice ale operațiilor efectuate, aceasta este ruta ce se trimite către html pentru a se putea afișa imaginile. ”Styles” și ”scripts” conțin fișierele de css și javascript folosite în aplicație pentru a customiza randarea elementelor în pagină și aplicarea funcțiilor implementate. Iar folderul ”theory” conține Lucrarea de licență către care se face referință pentru mai multe detalii legate de algoritmii folosiți.

Folderul ”templates” conține toate htmlurile pe baza cărora se randează paginile aplicației.

Iar în folderul ”utilities” sunt implementate toate fișierele suport de python pentru funcționalitățile create. Acest folder poate fi considerat din multe puncte de vedere ”creierul aplicației”.

Fișierul ”init.py” poate fi considerat ”mainul aplicației”, fiind fișierul principal în care se dirijează demersul aplicației. Modul general în care funcționează acesta este: atunci când un utilizator dă click pe un element din navbar se deschide dându-se un ”GET” către ”init.py” și se randează doar htmlul ce conține un formular cu datele ce trebuie completate de către utilizator. Atunci când acestea sunt completate și se dă ”submit” se apelează ”POST” către ”init.py”, se recuperează toate datele din form și pe baza acestora se calculează algoritmul cu asistența fișierelor suport și se returnează hmtlul paginii plus toate variabilele cu ajutorul cărora se randează pașii algoritmului folosit.

În imaginea 29 este prezentat navbarul aplicației împreună cu un dropdown deschis cu ajutorul cărora se poate naviga în paginile existente.

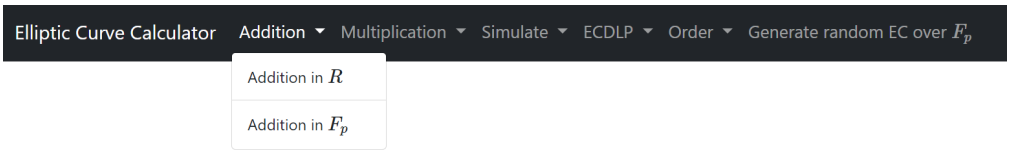


Figura 29: Meniul aplicației

Formul unde se completează datele pentru algoritm:

```
<form id = "myForm" action = "/addition_r" method = "POST">
    <label> Coefficient $a$:</label>
    <input type = "number" name = "coefficient_curve_a" required/>
    <br>
    <label> Coefficient $b$: </label>
    <input type = "number" name = "coefficient_curve_b" required/>
    <br><br>

    <label> Point $P_x$ coordinate: </label>
    <input type = "text" name = "coordinate_p_x"/>
    <br>
    <label> Point $P_y$ coordinate: </label>
    <input type = "text" name = "coordinate_p_y"/>
    <br><br>
```

```
    <label> Point $Q_x$ coordinate: </label>
    <input type = "text" name = "coordinate_q_x"/>
    <br>
    <label> Point $Q_y$ coordinate: </label>
    <input type = "text" name = "coordinate_q_y"/>
    <br><br>

    <input type = "submit" value = OK />
    <input type = "button" onclick = "reset()" value = "Reset">
    <br>
</form>
```

Fragmentul de cod 1: Completare form

Imaginea 30 prezintă cum arată formul în aplicație.

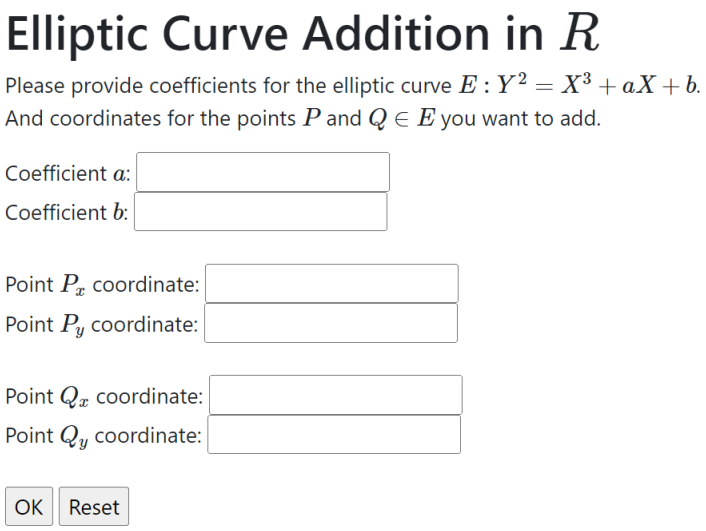


Figura 30: Formul pentru adunare în $\mathbb{R}$

Modul în care se realizează rutarea:

```
@app.route('/addition_r', methods=['GET', 'POST'])
def addition_r():
```

Fragmentul de cod 2: Crearea rutelor

Verificarea tipului de request:

```
if request.method == 'POST':
```

Fragmentul de cod 3: Verificarea requestului

Preluarea datelor în python din Form:

```
#coefficients of the elliptic curve
a = float(request.form['coefficient_curve_a'])
b = float(request.form['coefficient_curve_b'])

#point P
px = request.form['coordinate_p_x']
py = request.form['coordinate_p_y']

# determine if P == O('inf', 'inf')
if px != 'inf':
    px = float(px)
    py = float(py)

#point Q
qx = request.form['coordinate_q_x']
qy = request.form['coordinate_q_y']

# determine if Q == O('inf', 'inf')
if qx != 'inf':
    qx = float(qx)
    qy = float(qy)
```

Fragmentul de cod 4: Preluarea datelor

Returnarea către html a variabilelor:

```
return render_template('addition_r.html', get_plot = True, plot_url = '
   static/images/elliptic_curve_addition_r.png', data = data, case = case,
   slope = slope, coefficients = coefficients)
```

Fragmentul de cod 5: Returnarea variabilelor

Și folosirea acestora în html utilizând LaTeXMathML și MathJax pentru randarea în LaTeX, următorul exemplu fiind pasul al doilea al schimbului de chei eliptic Diffie-Hellman:

```
<h4>Step 2: Secret key computation</h4>
Alice chosses a secret number $n_A = {{secret_numbers[0]}}$, and calculates
    her public key
    <p><span class="math display">
        \begin{equation}
        \begin{split}
            Q_A &amp; = n_AP \\
            &amp; = {{secret_numbers[0]}} \cdot {{p}} \\
            &amp; = {{public_keys[0]}} \in E(F_{ {{ec[2]}} })
        \end{split}
        \end{equation}
    </span></p>
```

Fragmentul de cod 6: Utilizarea variabilelor în html

5.3 Adunare a două puncte pe o curbă eliptică în $\mathbb{R}$ și $\mathbb{F}_p$

5.3.1 Adunare a două puncte pe o curbă eliptică în $\mathbb{R}$

Am realizat adunarea a două puncte pe o curbă eliptică în $\mathbb{R}$ conform algoritmului prezentat la (2.18). Acesta fiind codul folosit pentru adunare în $\mathbb{R}$:

```
#function for adding 2 points on the elliptic curve
def add_points_r(a, p, q):
    px, py = p[0], p[1]
    qx, qy = q[0], q[1]

    #case 1: P or Q == O('inf', 'inf')
    if px == 'inf':
        return (qx, qy, 1, None)
    if qx == 'inf':
        return (px, py, 1, None)

    #case 2: P == -Q
    if py == -qy:
        return ('inf', 'inf', 2, None)

    #case 3: P == Q
    if px == qx and py == qy:
        #slope
        s = (3 * px ** 2 + a) / (2 * py)
        #new point
        rx = s ** 2 - 2 * px
        ry = s * (px - rx) - py

        return (round(rx, 4), round(ry, 4), 3, s)

    #case 4: P != Q
    #slope
    s = (qy - py) / (qx - px)
    #new point
    rx = s ** 2 - px - qx
    ry = s * (px - rx) - py

    return (round(rx, 4), round(ry, 4), 4, s)
```

Fragmentul de cod 7: Adunarea în $\mathbb{R}$

Observăm că pe lângă rezultat mai returnăm încă două numere, primul număr reprezintă cazul în care ne aflăm, iar ce de-al doilea număr este panta dreptei, necontând în primele două cazuri, are valoarea 0. Acestea sunt folosite pentru a determina în ce caz ne aflăm din front-end pentru a ști ce explicație să randăm și cu ce valori. Rezultatul se calculează cu 4 zecimale.

Vom lua exemplul folosit anterior (în subcapitolul 2.2) și vom dezvolta.

Exemplul 5.1. Fie curba eliptică E de ecuație:

$$E : Y^2 = X^3 - 94X + 361 \tag{8}$$

Și punctele $P(4, 7)$, $P' = -P = (4, -7)$ și $Q(8, 11)$ de pe E și punctul la infinit $O(\infty, \infty)$.

(1) $P + O$ adunarea unui punct cu infinit

Rezultatul este P, fără a fi nevoie de calcule suplimentare.

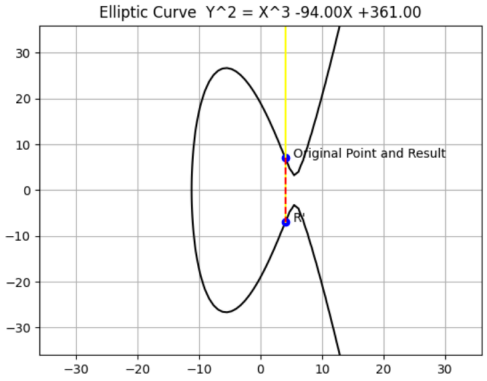


Figura 31: Reprezentare grafică

Point	*X*	*Y*
P	4.0	7.0
Q	*inf*	*inf*
R	4.0	7.0

Figura 32: Tabel cu rezultate

(2) $P + P'$ adunarea unui punct cu opusul său

Rezultatul este O, fără a fi nevoie de calcule suplimentare.

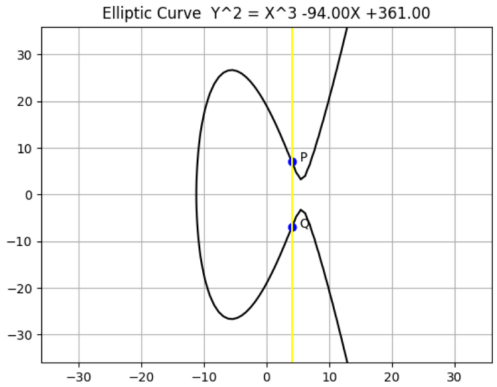


Figura 33: Reprezentare grafică

Point	*X*	*Y*
P	4.0	7.0
Q	4.0	-7.0
R	*inf*	*inf*

Figura 34: Tabel cu rezultate

(3) $P + P$ adunarea unui punct cu el însuși

Doar pentru acest exemplu, fie curba eliptică E de ecuație:

$$E : Y^2 = X^3 + 3X + 29 \tag{9}$$

Și punctul $P(5, 13)$ de pe E.

Calculăm panta dreptei, iar apoi punctul ce reprezintă rezultatul operației (în aplicație operațiile sunt pe mijlocul paginii, dar pentru spațiu au fost decupate). Acestea au fost prezentate în aplicație, apoi s-au folosit datele oferite de utilizator pentru a se ajunge la rezultat.

We are in the third case where $P = Q$, meaning $x_P = x_Q$ and $y_P = y_Q$.
First, we calculate the slope m of the line that is tangent to the line of the elliptic curve E at P:

$$\begin{aligned} m &= \frac{3x_P + a}{2y_P} \\ &= \frac{3 \cdot 5.0 + 3.0}{2 \cdot 13.0} \\ &= 3.0 \end{aligned}$$

Then we determine the coordinates of the result R:

$$\begin{aligned} x_R &= m^2 - x_P^2 - x_Q^2 \\ &= (3.0)^2 - (5.0)^2 - (5.0)^2 \\ &= -1.0 \end{aligned}$$

$$\begin{aligned} y_R &= m(x_P - x_R) - y_P \\ &= 3.0(5.0 - -1.0) - 13.0 \\ &= 5.0 \end{aligned}$$

So the result of $P + Q$ is $R(-1.0, 5.0)$.

Figura 35: Panta și rezultatul de la exemplul 3

Și rezultă

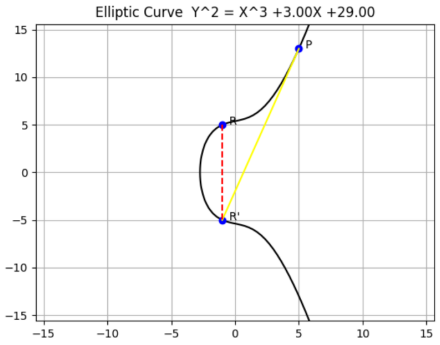


Figura 36: Reprezentare grafică

Point	*X*	*Y*
P	5.0	13.0
Q	5.0	13.0
R	-1.0	5.0

Figura 37: Tabel cu rezultate

(4) $P + Q$ adunarea a două puncte diferite

Calculăm panta dreptei, iar apoi punctul ce reprezintă rezultatul operației (în aplicație operațiile sunt pe mijlocul paginii, dar pentru spațiu au fost decupate). Acestea au fost prezentate în aplicație, apoi s-au folosit datele oferite de utilizator pentru a se ajunde la rezultat.

We are in the fourth case where $P \neq Q$, meaning $x_P \neq x_Q$ or $y_P \neq y_Q$.
First we calculate the slope m of the line that passes through the points P and Q:

$$\begin{aligned} m &= \frac{y_Q - y_P}{x_Q - x_P} \\ &= \frac{11.0 - 7.0}{8.0 - 4.0} \\ &= 1.0 \end{aligned}$$

Then we determine the coordinates of the result R.

$$\begin{aligned} x_R &= m^2 - x_P - x_Q \\ &= (1.0)^2 - 4.0 - 8.0 \\ &= -11.0 \end{aligned}$$

$$\begin{aligned} y_R &= m(x_P - x_R) - y_P \\ &= 1.0(4.0 - -11.0) - 7.0 \\ &= 8.0 \end{aligned}$$

So the result of $P(4.0, 7.0) + Q(8.0, 11.0)$ is $R(-11.0, 8.0)$.

Figura 38: Panta și rezultatul de la exemplul 4

Și rezultă

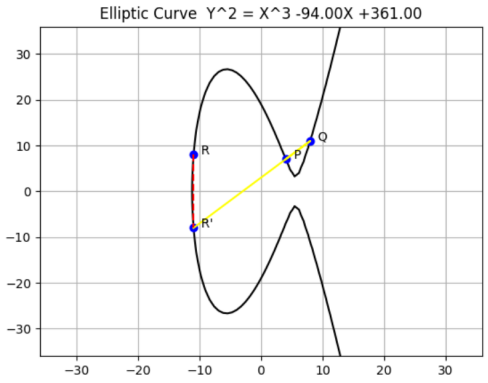


Figura 39: Reprezentare grafică

$Point$	X	Y
P	4.0	7.0
Q	8.0	11.0
R	-11.0	8.0

Figura 40: Tabel cu rezultate

Reprezentările grafice au fost realizate cu ajutorul a NumPy și Matplotlib. Am folosit NumPy pentru a putea crea curbele eliptice (funcția lor) în $\mathbb{R}$ și Matplotlib pentru plotarea afectivă a imaginii și salvarea acesteia în fișierul predestinat.

```
#function for drawing the image for a finite result and saving it
def draw_image_addition_r_fin(a, b, px, py, qx, qy, rx, ry, rpx, rpy):
    #we determine the scale of the graph based on the max abs point or 20
    points = [abs(px), abs(py), abs(qx), abs(qy), abs(rx), abs(ry), abs(rpx
    ), abs(rpy), 10]
    scale = max(points) * 1.2
```

```
    #divide the list of points into xs and ys for the lines we will draw
    #line1 - line that intersect P and Q
    #line2 - line for the reverse R'
    line1_x = [px, qx, rpx]
    line1_y = [py, qy, rpy]
    line2_x = [rx, rpx]
    line2_y = [ry, rpy]

    #drawing the curve
    y, x = np.ogrid[-scale:scale:100j, -scale:scale:100j]
    plt.contour(x.ravel(), y.ravel(), y**2 - x**3 - x * a - b, [0], colors=
    'black')

    #draw the initial points
    plt.plot(px, py, 'bo')
    plt.text(px, py, f'  P')

    #determine if the points are different
    if (px, py) != (qx, qy):
        plt.plot(qx, qy, 'bo')
        plt.text(qx, qy, f'  Q')

    #draw the sum of the points
    plt.plot(rpx, rpy, 'bo')
    plt.text(rpx, rpy, f'  R\'')
    plt.plot(rx, ry, 'bo')
    plt.text(rx, ry, f'  R')

    #draw the lines
    plt.plot(line1_x, line1_y, color='yellow')
    plt.plot(line2_x, line2_y, color='red', linestyle='dashed')

    plt.grid()

    sign_a = ''
    if a >= 0:
        sign_a = '+'

    sign_b = ''
    if b >= 0:
        sign_b = '+'

    plt.title(f'Elliptic Curve  Y^2 = X^3{sign_a}{a:.2f}X{sign_b}{b:.2f}')
    plt.savefig('static/images/elliptic_curve_addition_r.png')
```

Fragmentul de cod 8: Plotarea unei adunări cu rezultat finit

Inițial determinăm scara pe care o dorim să o aibă imaginea ca fiind valoarea maximă a punctelor în modul sau valoarea 10, apoi înmulțim cu 1.2 pentru a nu tăia anumite elemente precum puncte dacă sunt aproape de marginea superioară sau inferioară. Folosim funcția $np.ogrid$ pentru a da valori variabilelor și $plt.contour$ și ecuația curbei eliptice $y^2 - x^3 - x \cdot a - b = 0$, putem reprezenta grafic curba.

5.3.2 Adunare a două puncte pe o curbă eliptică în $\mathbb{F}_p$

Am realizat adunarea a două puncte pe o curbă eliptică peste un corp finit $\mathbb{F}_p$ conform algoritmului prezentat la (2.21). Procesul este foarte asemănător cu cel de la adunarea în $\mathbb{R}$, dar acum toate calculele vor fi modulo p. Observăm că doar cazurile 3 și 4 definite anterior sunt impactate.

```
#case 3: P == Q
if px == qx and py == qy:
    term1 = int(3 * px ** 2 + a)
    term2 = pow(int(2 * py), -1, prime)

    #slope
    s = (term1 * term2) % prime

    #new point
    rx = (s ** 2 - 2 * px) % prime
    ry = (s * (px - rx) - py) % prime

    return (rx, ry, 3, s, term1, term2)

#case 4: P != Q
term1 = int(qy - py)
term2 = pow(int(qx - px), -1, prime)

#slope
s = (term1 * term2) % prime

#new point
rx = (s ** 2 - px - qx) % prime
ry = (s * (px - rx) - py) % prime

return (rx, ry, 4, int(s), term1, term2)
```

Fragmentul de cod 9: Adunarea în $\mathbb{F}_p$

Putem observa că am păstrat aproape identic codul, doar că ne-am folosit de formula următoare pentru a rezolva adunarea.

$$\frac{x}{y} \pmod p = x \cdot y^{-1} \pmod p$$

Vom lua exemplul folosit anterior (în subcapitolul 2.4) și vom dezvolta.

Exemplul 5.2. Fie curbă eliptică E peste $\mathbb{F}_{97}$:

$$E : Y^2 = X^3 + 2X + 3 \pmod{97}$$

Și punctele $P(17, 10)$ și $Q(95, 31)$, cu $P, Q \in E(\mathbb{F}_{97})$. Pentru a aduna P și Q, mai întâi trebuie să calculăm panta dreptei, iar apoi să calculăm coordonatele rezultatului. Acestea au fost prezentate în aplicație, apoi s-au folosit datele oferite de utilizator pentru a se ajunge la rezultat.

We remind that if a fraction is not an integer in F_p then:

$$\frac{x}{y} \pmod{\text{p}} = x \cdot y^{-1} \pmod{\text{p}}$$

First we calculate the slope m of the line that passes through the points P and Q:

$$\begin{aligned} m &= \frac{y_Q - y_P}{x_Q - x_P} \pmod{\text{p}} \\ &= \frac{31 - 10}{95 - 17} \pmod{97} \\ &= (31 - 10) \cdot (95 - 17)^{-1} \pmod{97} \\ &= 21 \cdot 51 \pmod{97} \\ &= 4 \end{aligned}$$

Then we determine the coordinates of the result R:

$$\begin{aligned} x_R &= m^2 - x_P - x_Q \pmod{\text{p}} \\ &= (4)^2 - 17 - 95 \pmod{97} \\ &= 1 \end{aligned}$$

$$\begin{aligned} y_R &= m(x_P - x_R) - y_P \pmod{\text{p}} \\ &= 4(17 - 1) - 10 \pmod{97} \\ &= 54 \end{aligned}$$

So the result of $P(17, 10) + Q(95, 31)$ is $R(1, 54)$.

Figura 41: Panta și rezultatul

Și rezultă

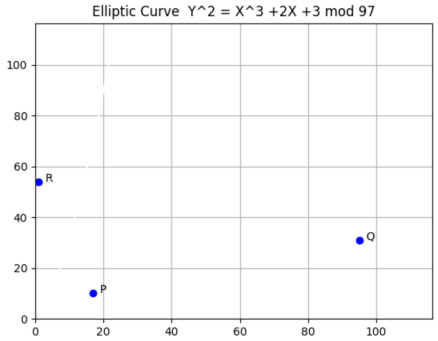


Figura 42: Reprezentare grafică

Point	*X*	*Y*
P	17	10
Q	95	31
R	1	54

Figura 43: Tabel cu rezultate

5.4 Înmulțirea unui punct cu un scalar pe o curbă eliptică în $\mathbb{R}$ și $\mathbb{F}_p$

5.4.1 Înmulțirea unui punct cu un scalar pe o curbă eliptică în $\mathbb{R}$

Pentru înmulțirea unui punct cu un scalar pe o curbă eliptică în $\mathbb{R}$ am implementat algoritmul dublare și adunare, acesta este la fel ca cel de la subcapitolul viitor (subcapitolul 5.4.2), dar calculele se realizează în $\mathbb{R}$ și nu în $\mathbb{F}_p$.

Exemplul 5.3. Fie curba eliptică E

$$E : Y^2 = X^3 + X + 13$$

punctul $P = (4, 9) \in E$ și scalarul $n = 81$.

Algoritmul dublare și adunare este prezentat pas cu pas în figura 44.

Step i	n	$Q = 2^i P$	R
0	81	$(4, 9)$	(inf, inf)
1	40	$(-0.5895, 3.4937)$	$(4, 9)$
2	20	$(1.2644, -4.0356)$	$(4, 9)$
3	10	$(-2.0131, 1.682)$	$(4, 9)$
4	5	$(19.3247, -85.1412)$	$(4, 9)$
5	2	$(4.7146, -11.0685)$	$(14.413, 54.9682)$
6	1	$(-0.0813, -3.5947)$	$(14.413, 54.9682)$
7	0	$(0.1827, 3.6322)$	$(1.9932, -4.7871)$

Figura 44: $81 \cdot (4, 9)$ pe $Y^2 = X^3 + X + 13$

Algoritmul necesită 7 dublaje și 3 adunări (acestea sunt determinate în funcție) și rezultă (calculele au fost făcute cu precizie la a patra zecimală).

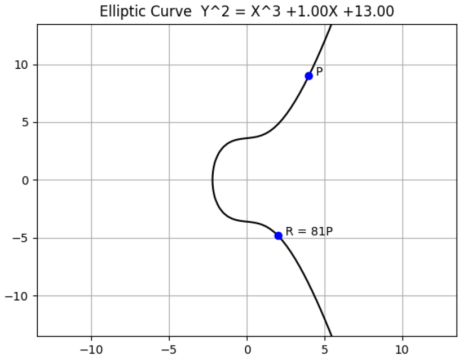


Figura 45: Reprezentare grafică

Point	X	Y
P	4.0	9.0
R	1.9932	−4.7871

Figura 46: Tabel cu rezultate

5.4.2 Înmulțirea unui punct cu un scalar pe o curbă eliptică în $\mathbb{F}_p$

Pentru înmulțirea unui punct cu un scalar pe o curbă eliptică în $\mathbb{F}_p$ am implementat algoritmul dublare și adunare prezentat anterior (în subcapitolul 4.3).

```
#function for multipling a points by a positive number on the elliptic
    curve
def multiply_points_f_double_and_add(a, p, n, prime):
    #the list that will contain all the steps for the double and add
    algorithm
    multiplication_steps = []

    if p[0] != 'inf':
        q = (int(p[0]), int(p[1]))
    else:
        q = (p[0], p[1])
    r = ('inf', 'inf')
    step = 0
    additions = 0

    multiplication_steps.append((step, n, q, r))

    #double and add algorithm
    while n > 0:
        step += 1

        if n % 2 == 1:
            rx, ry, _, _, _, _ = add_points_f(a, r, q, prime)
            if rx != 'inf':
                r = (int(rx), int(ry))
            else:
                r = (rx, ry)

            additions += 1

        qx, qy, _, _, _, _ = add_points_f(a, q, q, prime)
        if qx != 'inf':
            q = (int(qx), int(qy))
        else:
            q = (qx, qy)

        n //= 2

        multiplication_steps.append((step, n, q, r))

    multiplications = len(multiplication_steps) - 1

    return ((p, r), multiplication_steps, additions, multiplications)
```

Fragmentul de cod 10: Algoritmul dublare și adunare în $\mathbb{F}_p$

Funcția returnează punctul inițial P, rezultatul $R = nP$ și numărul de adunări și înmulțiri pe care le-a efectuat algoritmul.

Folosim datele din exemplul din subcapitolul 4.3.

Exemplul 5.4. Fie numărul prim $p = 2399$ și curba eliptică E

$$E : Y^2 = X^3 + X + 11 \pmod{2399}$$

și punctul $P = (196, 41) \in E(\mathbb{F}_{2399})$ și scalarul $n = 1187$.

Algoritmul dublare și adunare este prezentat pas cu pas în figura 47.

Step i	n	$Q = 2^i P$	R
0	1187	$(196, 41)$	(∞, ∞)
1	593	$(780, 848)$	$(196, 41)$
2	296	$(674, 624)$	$(594, 1274)$
3	148	$(1260, 2140)$	$(594, 1274)$
4	74	$(1398, 913)$	$(594, 1274)$
5	37	$(1195, 1089)$	$(594, 1274)$
6	18	$(1234, 2344)$	$(923, 1837)$
7	9	$(1550, 567)$	$(923, 1837)$
8	4	$(509, 689)$	$(2385, 933)$
9	2	$(1474, 1715)$	$(2385, 933)$
10	1	$(312, 793)$	$(2385, 933)$
11	0	$(854, 1176)$	$(1062, 368)$

Figura 47: $1187 \cdot (196, 41)$ pe $Y^2 = X^3 + X + 11 \pmod{2399}$

Algoritmul necesită 11 dublaje și 5 adunări, iar rezultatul final este $1187P = (1062, 368)$.

5.5 Simulare ECDH

5.5.1 Simulare ECDH

Pentru a simula schimbul de chei Diffie-Hellman am implementat procesul descris anterior în subcapitolul 4.4.

```
if request.method == 'POST':
    #clear the buffer
    plt.clf()

    #coefficients of the elliptic curve
    a = float(request.form['coefficient_curve_a'])
    b = float(request.form['coefficient_curve_b'])

    #prime number in Fp
    prime = int(request.form['prime_p'])

    #details of the elliptic curve
```

```
    ec = (int(a), int(b), prime)

    #point P
    px = request.form['coordinate_p_x']
    py = request.form['coordinate_p_y']

    # determine if P == O('inf', 'inf')
    if px != 'inf':
        px = float(px)
        py = float(py)

    p = (int(px), int(py))

    #Alice's secret number
    na = int(request.form['secret_n_a'])

    #Bob's secret number
    nb = int(request.form['secret_n_b'])

    secret_numbers = (na, nb)

    #step 2: secret keys
    public_key_a = multiply_points_f_double_and_add(a, p, na, prime)[0][1]
    public_key_b = multiply_points_f_double_and_add(a, p, nb, prime)[0][1]

    public_keys = (public_key_a, public_key_b)

    #step 3: shared keys
    shared_key_a = multiply_points_f_double_and_add(a, public_key_b, na,
prime)[0][1]
    shared_key_b = multiply_points_f_double_and_add(a, public_key_a, nb,
prime)[0][1]

    shared_keys = (shared_key_a, shared_key_b)

    return render_template('normal_ecdh.html', get_plot = True, plot_url =
'static/theory/ECDH.png', ec = ec, p = p, secret_numbers =
secret_numbers, public_keys = public_keys, shared_keys = shared_keys)
```

Fragmentul de cod 11: Schimbul de chei Diffie-Hellman

Folosim datele din exemplul din subcapitolul 4.4 și trecem în paralel prin toate etapele algoritmului cu implementarea din aplicație.

Exemplul 5.5. Pasul 1, Alice și Bob stabilesc numărul $p = 4219$ și curba eliptică E și punctul P, cu $P \in \mathbb{F}_p$

$$E : Y^2 = X^3 + 268X + 1344 \text{ (mod } 4219)$$
$$P = (940, 256) \in E(\mathbb{F}_{4219})$$

Step 1: Configuration

Alice and Bob decide and and publish a prime number $p = 4219$, the elliptic curve E over F_{4219} and the point $P \in F_{4219}$:

$$E : Y^2 = X^3 + 268X + 1344 \pmod{4219}$$
$$P = (940, 256) \in E(F_{4219})$$

Figura 48: Pasul 1 al schimbului de chei eliptic Diffie-Hellman

Pasul 2, Alice alege numărul întreg secret $n_A = 852$ și calculează

$$Q_A = 852P = (2267, 3379) \in E(\mathbb{F}_{4219})$$

Analog, Bob alege numărul întreg secret $n_b = 1407$ și calculează

$$Q_B = 1407P = (3129, 2271) \in E(\mathbb{F}_{4219})$$

Step 2: Secret key computation

Alice chosses a secret number $n_A = 852$, and calculates her public key

$$\begin{aligned} Q_A &= n_A P \\ &= 852 \cdot (940, 256) \\ &= (2267, 3379) \in E(F_{4219}) \end{aligned}$$

Bob chosses a secret number $n_B = 1407$, and calculates his public key

$$\begin{aligned} Q_B &= n_B P \\ &= 1407 \cdot (940, 256) \\ &= (3129, 2271) \in E(F_{4219}) \end{aligned}$$

Figura 49: Pasul 2 al schimbului de chei eliptic Diffie-Hellman

Pasul 3, Alice îi trimite $(2267, 3379)$ (Q_A) lui Bob și Bob îi trimite $(3129, 2271)$ (Q_B) lui Alice pe canalul nesecurizat.

Alice își folosește numărul secret și calculează

$$Q'_A = 852Q_B = (677, 2358) \in E(\mathbb{F}_{4219})$$

Bob își folosește numărul secret și calculează

$$Q'_B = 1407Q_A = (677, 2358) \in E(\mathbb{F}_{4219})$$

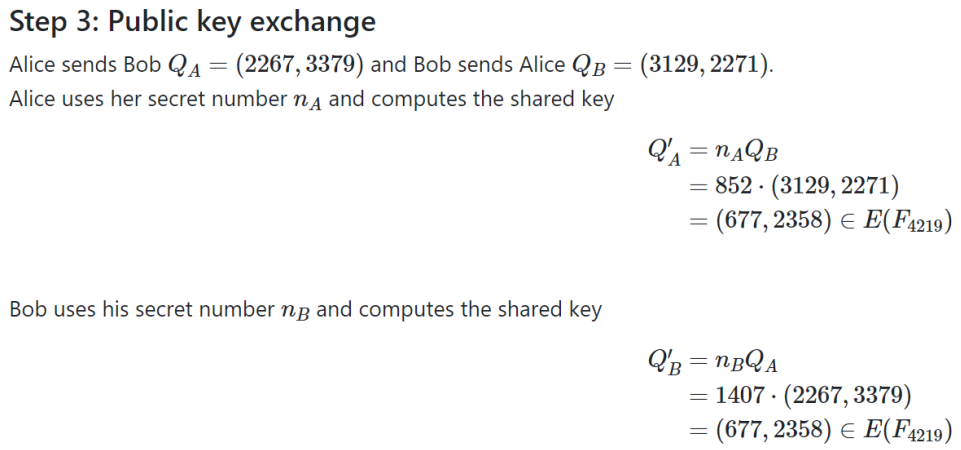


Figura 50: Pasul 3 al schimbului de chei eliptic Diffie-Hellman

Pasul 4, Alice și Bob au obținut cheia secretă comună

$$(677, 2358) = n_A n_B P = Q'_A = Q'_B$$

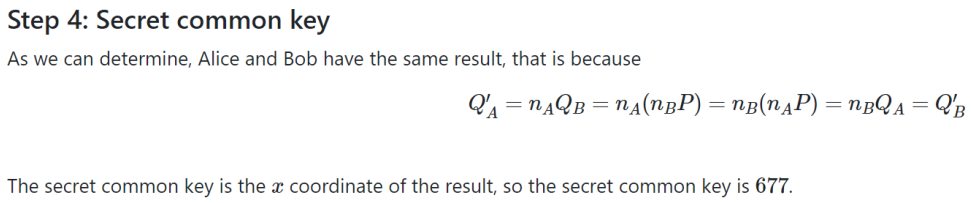


Figura 51: Pasul 4 al schimbului de chei eliptic Diffie-Hellman

5.5.2 Simulare ECDH îmbunătățit

Acest algoritm este analog cu cel normal până la al treilea pas unde se publică doar componenta x a punctelor (acest proces a fost descris în subcapitolul 4.4). Partea ce adaugă complexitate este determinarea componentei y, pentru un anumit punct, pentru care am dezvoltat următoarele funcții ajutătoare suplimentare.

```
#determines the y^2 based on the equation and the x
def result_equation_ec(a, b, prime, x):
    result = (x**3 + a * x + b) % prime

    return int(result)


#determines the y based on the x
def determine_y(prime, x):
```

```
    power = int((prime + 1) / 4)
    result = pow(int(x), power, prime)

    return (int(result), power)
```

Fragmentul de cod 12: Funcții suport pentru ECDH îmbunătățit

Acesta este codul implementat folosit pentru simularea schimbului de chei Diffie-Hellman îmbunătățit.

```
if request.method == 'POST':
    #clear the buffer
    plt.clf()

    #coefficients of the elliptic curve
    a = float(request.form['coefficient_curve_a'])
    b = float(request.form['coefficient_curve_b'])

    #prime number in Fp
    prime = int(request.form['prime_p'])

    #details of the elliptic curve
    ec = (int(a), int(b), prime)

    #point P
    px = request.form['coordinate_p_x']
    py = request.form['coordinate_p_y']

    # determine if P == O('inf', 'inf')
    if px != 'inf':
        px = float(px)
        py = float(py)

    p = (int(px), int(py))

    #Alice's secret number
    na = int(request.form['secret_n_a'])

    #Bob's secret number
    nb = int(request.form['secret_n_b'])

    secret_numbers = (na, nb)

    #step 2: secret keys
    public_key_a = multiply_points_f_double_and_add(a, p, na, prime)[0][1]
    public_key_b = multiply_points_f_double_and_add(a, p, nb, prime)[0][1]

    #step 3:
    #determine square ys
    square_ya = result_equation_ec(a, b, prime, public_key_a[0])
    square_yb = result_equation_ec(a, b, prime, public_key_b[0])

    square_ys = (square_ya, square_yb)
```

```
    #determine ys
    ya, power_ya = determine_y(prime, square_ya)
    yb, power_yb = determine_y(prime, square_yb)

    public_keys = ((public_key_a[0], ya) , (public_key_b[0], yb))
    ys = (ya, yb)
    power_ys = (power_ya, power_yb)

    # shared keys
    shared_key_a = multiply_points_f_double_and_add(a, public_keys[1], na,
prime)[0][1]
    shared_key_b = multiply_points_f_double_and_add(a, public_keys[0], nb,
prime)[0][1]

    shared_keys = (shared_key_a, shared_key_b)

    return render_template('improved_ecdh.html', get_plot = True, plot_url
= 'static/theory/ECDH.png', ec = ec, p = p, secret_numbers =
secret_numbers, public_keys = public_keys, square_ys = square_ys, ys =
ys, power_ys = power_ys, shared_keys = shared_keys)
```

Fragmentul de cod 13: Schimbul de chei eliptic Diffie-Hellman îmbunătățit

Adițional față de algoritmul normal, începând de la pasul 3, determinăm y^2 din ecuația curbei eliptice cu coordonata x primită și apoi din acest pătrat determină coordonata y (aceasta este posibil să nu fie cea inițială, dar cum am demonstrat în subcapitolul 4.4, acest fapt nu impactează rezultatul final).

Folosim curba eliptică din exemplul din subcapitolul 4.4 și trecem în paralel prin toate etapele algoritmului cu implementarea din aplicație.

Exemplul 5.6. Pasul 1, Alice și Bob stabilesc numărul $p = 4219$ și curba eliptică E și punctul P, cu $P \in \mathbb{F}_p$

$$E : Y^2 = X^3 + 268X + 1344 \pmod{4219}$$
$$P = (940, 256) \in E(\mathbb{F}_{4219})$$

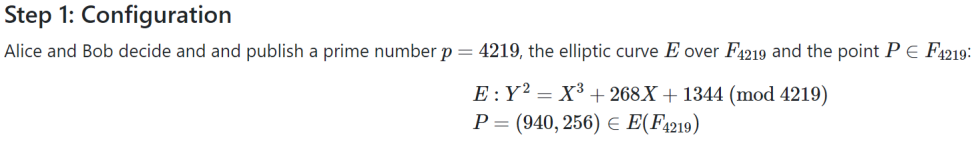


Figura 52: Pasul 1 al schimbului de chei eliptic Diffie-Hellman îmbunătățit

Pasul 2, Alice alege numărul întreg secret $n_A = 1327$ și calculează

$$Q_A = 1327P = (1933, 2897) \in E(\mathbb{F}_{4219})$$

Analog, Bob alege numărul întreg secret $n_b = 763$ și calculează

$$Q_B = 763P = (172, 3005) \in E(\mathbb{F}_{4219})$$

Step 2: Secret key computation

Alice chosses a secret number $n_A = 1327$, and calculates her public key

$$\begin{aligned} Q_A &= n_A P \\ &= 1327 \cdot (940, 256) \\ &= (1933, 2897) \in E(F_{4219}) \end{aligned}$$

Bob chosses a secret number $n_B = 763$, and calculates his public key

$$\begin{aligned} Q_B &= n_B P \\ &= 763 \cdot (940, 256) \\ &= (172, 3005) \in E(F_{4219}) \end{aligned}$$

Figura 53: Pasul 2 al schimbului de chei eliptic Diffie-Hellman îmbunătățit

Pasul 3, Alice îi trimite 1933 (x_A) lui Bob și Bob îi trimite 172 (x_B) lui Alice pe canalul nesecurizat.

Alice substituie $x_B = 172$ în ecuația curbei eliptice E pentru a determina y_B^2

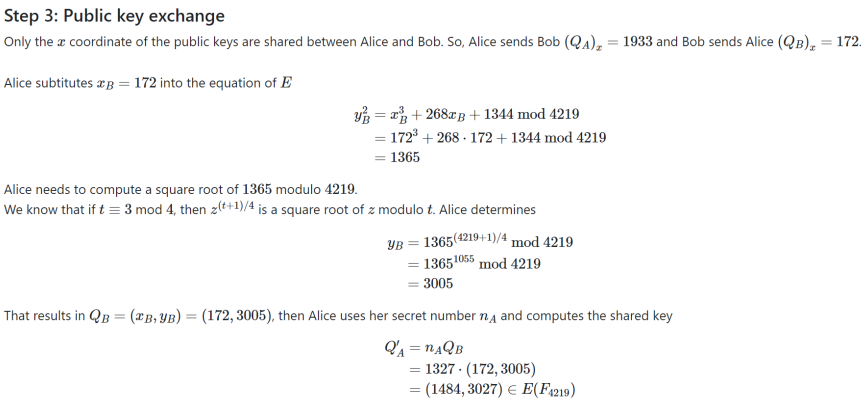
$$\begin{aligned} y_B^2 &\equiv x_B^3 + 268 x_B + 1344 \pmod{4219} \\ &\equiv 172^3 + 268 \cdot 172 + 1344 \pmod{4219} \\ &\equiv 1365 \end{aligned}$$

Alice determină y_B

$$y_B = 1365^{(4219+1)/4} \equiv 3005 \pmod{4219}$$

Și determină Q'_A

$$\begin{aligned} Q'_A &= n_A Q_B \\ &= 1327 \cdot (172, 3005) \\ &= (1484, 3027) \in E(\mathbb{F}_{4219}) \end{aligned}$$



Step 3: Public key exchange

Only the x coordinate of the public keys are shared between Alice and Bob. So, Alice sends Bob $(Q_A)_x = 1933$ and Bob sends Alice $(Q_B)_x = 172$.

Alice subtitutes $x_B = 172$ into the equation of E

$$\begin{aligned} y_B^2 &= x_B^3 + 268x_B + 1344 \bmod 4219 \\ &= 172^3 + 268 \cdot 172 + 1344 \bmod 4219 \\ &= 1365 \end{aligned}$$

Alice needs to compute a square root of 1365 modulo 4219.
We know that if $t \equiv 3 \bmod 4$, then $z^{(t+1)/4}$ is a square root of z modulo t. Alice determines

$$\begin{aligned} y_B &= 1365^{(4219+1)/4} \bmod 4219 \\ &= 1365^{1055} \bmod 4219 \\ &= 3005 \end{aligned}$$

That results in $Q_B = (x_B, y_B) = (172, 3005)$, then Alice uses her secret number n_A and computes the shared key

$$\begin{aligned} Q'_A &= n_A Q_B \\ &= 1327 \cdot (172, 3005) \\ &= (1484, 3027) \in E(F_{4219}) \end{aligned}$$

Figura 54: Pasul 3 al schimbului de chei eliptic Diffie-Hellman îmbunătățit realizat de Alice

Bob substituie $x_A = 1933$ în ecuația curbei eliptice E pentru a determina y_A^2

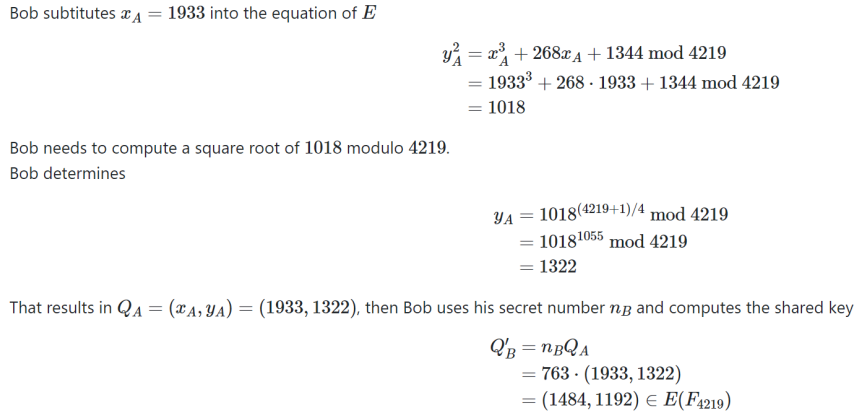
$$\begin{aligned} y_A^2 &\equiv x_A^3 + 268x_A + 1344 \pmod{4219} \\ &\equiv 1933^3 + 268 \cdot 1933 + 1344 \pmod{4219} \\ &\equiv 1018 \end{aligned}$$

Bob determină y_A

$$y_A = 1018^{(4219+1)/4} \equiv 1322 \pmod{4219}$$

Și determină Q'_B

$$\begin{aligned} Q'_B &= n_B Q_A \\ &= 763 \cdot (1933, 1322) \\ &= (1484, 1192) \in E(\mathbb{F}_{4219}) \end{aligned}$$



Bob subtitutes $x_A = 1933$ into the equation of E

$$\begin{aligned} y_A^2 &= x_A^3 + 268x_A + 1344 \bmod 4219 \\ &= 1933^3 + 268 \cdot 1933 + 1344 \bmod 4219 \\ &= 1018 \end{aligned}$$

Bob needs to compute a square root of 1018 modulo 4219.
Bob determines

$$\begin{aligned} y_A &= 1018^{(4219+1)/4} \bmod 4219 \\ &= 1018^{1055} \bmod 4219 \\ &= 1322 \end{aligned}$$

That results in $Q_A = (x_A, y_A) = (1933, 1322)$, then Bob uses his secret number n_B and computes the shared key

$$\begin{aligned} Q'_B &= n_B Q_A \\ &= 763 \cdot (1933, 1322) \\ &= (1484, 1192) \in E(F_{4219}) \end{aligned}$$

Figura 55: Pasul 3 al schimbului de chei eliptic Diffie-Hellman îmbunătățit realizat de Bob

Pasul 4, Alice și Bob au obținut cheia secretă comună, aceasta fiind coordonata $x = 1484$ a valorilor Q'_A și Q'_B, chiar dacă valoarea coordonatei y este diferită.

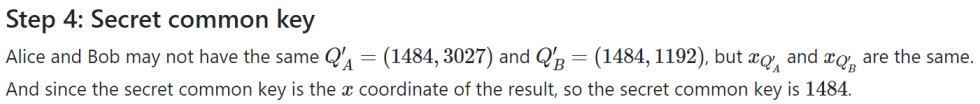


Figura 56: Pasul 4 al schimbului de chei eliptic Diffie-Hellman

5.6 Simulare criptosistem eliptic ElGamal

5.6.1 Simulare criptosistem eliptic ElGamal

Pentru a simula criptosistemul eliptic ElGamal (creare de chei, criptare și decriptare) am implementat procesul descris anterior în subcapitolul 4.5.

```
if request.method == 'POST':
    #clear the buffer
    plt.clf()

    #coefficients of the elliptic curve
    a = float(request.form['coefficient_curve_a'])
    b = float(request.form['coefficient_curve_b'])

    #prime number in Fp
    prime = int(request.form['prime_p'])

    #details of the elliptic curve
    ec = (int(a), int(b), prime)

    #point P
    px = request.form['coordinate_p_x']
    py = request.form['coordinate_p_y']

    p = (int(px), int(py))

    #Alice's secret number
    na = int(request.form['secret_n_a'])

    #message point M
    mx = request.form['coordinate_m_x']
    my = request.form['coordinate_m_y']

    m = (int(mx), int(my))

    #Bob's temporal number
    k = int(request.form['temporal_key_k'])

    #step 2: public key creation
    qa = multiply_points_f_double_and_add(a, p, na, prime)[0][1]
```

```
    #step 3: encryption
    c1 = multiply_points_f_double_and_add(a, p, k, prime)[0][1]

    k_qa = multiply_points_f_double_and_add(a, qa, k, prime)[0][1]
    c2 = add_points_f(a, m, k_qa, prime)[:2]

    #step 4: decryption
    na_c1 = multiply_points_f_double_and_add(a, c1, na, prime)[0][1]
    reverse_na_c1 = (na_c1[0], prime - na_c1[1])

    final_m = add_points_f(a, c2, reverse_na_c1, prime)[:2]

    return render_template('normal_eceg.html', get_plot = True, plot_url =
'static/theory/ECEG.png', ec = ec, p = p, na = na, m = m, k = k, qa = qa
, c1 = c1, k_qa = k_qa, c2 = c2, na_c1 = na_c1, reverse_na_c1 =
reverse_na_c1, final_m = final_m)
```

Fragmentul de cod 14: Criptosistemul eliptic ElGamal

Folosim datele din exemplul din subcapitolul 4.5 și trecem în paralel prin toate etapele algoritmului cu implementarea din aplicație.

Exemplul 5.7. Pasul 1, configurare: Alice și Bob stabilesc numărul $p = 31$ și curba eliptică E și punctul P

$$E : Y^2 = X^3 + X + 13 \pmod{31}$$
$$P = (9, 10) \in E(\mathbb{F}_{31})$$

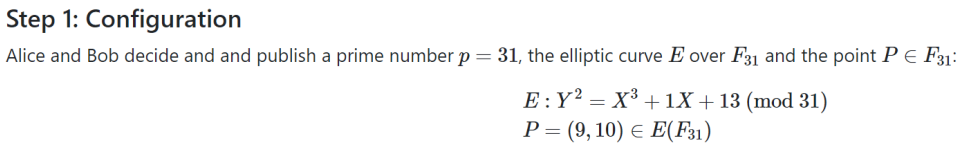


Figura 57: Pasul 1 al criptosistemului eliptic ElGamal

Pasul 2, crearea cheii publice: Alice alege numărul întreg secret $n_A = 5$ și calculează

$$Q_A = n_A P = 5(9, 10) = (25, 16)$$

Alice își publică cheia publică $Q_A = (25, 16)$.

Step 2: Secret key computation

Alice chosses a secret number $n_A = 5$, and calculates her public key

$$\begin{aligned} Q_A &= n_A P \\ &= 5 \cdot (9, 10) \\ &= (25, 16) \in E(F_{31}) \end{aligned}$$

Alice publishes her public key $Q_A = (25, 16)$.

Figura 58: Pasul 2 al criptosistemului eliptic ElGamal

Pasul 3, criptarea: Bob alege ca mesajul să fie $M = (20, 2)$ și cheia temporală $k = 7$, Bob folosește cheia publică a lui Alice pentru a calcula

$$\begin{aligned} C_1 &= kP = 7(9, 10) = (6, 24) \\ C_2 &= M + kQ_A = (20, 2) + 7(25, 16) \\ &= (20, 2) + (9, 10) = (22, 22) \end{aligned}$$

Bob îi trimite mesajul criptat $(C_1, C_2) = ((6, 24), (22, 22))$ lui Alice.

Step 3: Encryption

Bob chooses the message $M = (20, 2)$ and temporal key $k = 7$

$$\begin{aligned} C_1 &= kP \\ &= 7 \cdot (9, 10) \\ &= (6, 24) \in E(F_{31}) \end{aligned}$$

$$\begin{aligned} C_2 &= M + kQ_A \\ &= (20, 2) + 7 \cdot (25, 16) \\ &= (20, 2) + (9, 10) \\ &= (22, 22) \in E(F_{31}) \end{aligned}$$

Bob sends Alice the encrypted message $(C_1, C_2) = ((6, 24), (22, 22))$.

Figura 59: Pasul 3 al criptosistemului eliptic ElGamal

Pasul 4, decriptarea: Alice decriptează mesajul primit de la Bob

$$\begin{aligned} C_2 - n_A C_1 &= (22, 22) - 5(6, 24) \\ &= (22, 22) - (9, 10) \\ &= (22, 22) + (9, 21) = (20, 2) \end{aligned}$$

Rezultatul $(20, 2)$, fiind punctul M ales de Bob.

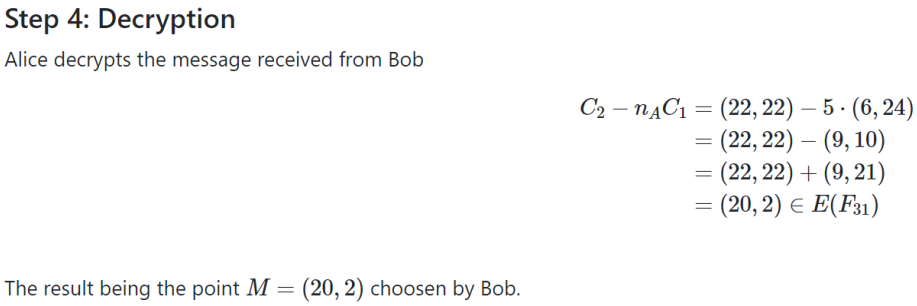


Figura 60: Pasul 4 al criptosistemului eliptic ElGamal

5.6.2 Simulare criptosistem eliptic ElGamal îmbunătățit

Acest algoritm este analog cu cel normal până la al treilea pas unde Bob îi trimite lui Alice doar componenta x a punctelor criptate C_1 și C_2 (acest proces a fost descris în subcapitolul 4.5). Partea ce adaugă complexitate este determinarea componentei y, pentru un anumit punct.

Acesta este codul implementat folosit pentru simularea criptosistemului eliptic ElGamal (creare de chei, criptare și decriptare) îmbunătățit.

```
if request.method == 'POST':
    #clear the buffer
    plt.clf()

    #coefficients of the elliptic curve
    a = float(request.form['coefficient_curve_a'])
    b = float(request.form['coefficient_curve_b'])

    #prime number in Fp
    prime = int(request.form['prime_p'])

    #details of the elliptic curve
    ec = (int(a), int(b), prime)

    #point P
    px = request.form['coordinate_p_x']
    py = request.form['coordinate_p_y']

    p = (int(px), int(py))

    #Alice's secret number
    na = int(request.form['secret_n_a'])

    #message point M
    mx = request.form['coordinate_m_x']
    my = request.form['coordinate_m_y']

    m = (int(mx), int(my))

    #Bob's temporal number
```

```
    k = int(request.form['temporal_key_k'])

    #step 2: public key creation
    qa = multiply_points_f_double_and_add(a, p, na, prime)[0][1]

    #step 3: encryption
    c1 = multiply_points_f_double_and_add(a, p, k, prime)[0][1]

    k_qa = multiply_points_f_double_and_add(a, qa, k, prime)[0][1]
    c2 = add_points_f(a, m, k_qa, prime)[:2]

    bit_c1 = 0
    if c1[1] > prime / 2:
        bit_c1 = 1

    bit_c2 = 0
    if c2[1] > prime / 2:
        bit_c2 = 1

    bits = (bit_c1, bit_c2)

    #step 4: decryption

    #determine square ys
    square_yc1 = result_equation_ec(a, b, prime, c1[0])
    square_yc2 = result_equation_ec(a, b, prime, c2[0])

    square_ys = (square_yc1, square_yc2)

    #determine ys
    yc1, power_yc1 = determine_y(prime, square_yc1)
    yc2, power_yc2 = determine_y(prime, square_yc2)

    ys = (yc1, yc2)
    power_ys = (power_yc1, power_yc2)

    #determine real ys
    real_yc1 = yc1
    if (bit_c1 == 0 and yc1 > prime / 2) or (bit_c1 == 1 and yc1 < prime /
2):
        real_yc1 = prime - yc1

    real_yc2 = yc2
    if (bit_c2 == 0 and yc2 > prime / 2) or (bit_c2 == 1 and yc2 < prime /
2):
        real_yc2 = prime - yc2

    real_ys = (real_yc1, real_yc2)

    #decrypt process
    na_c1 = multiply_points_f_double_and_add(a, c1, na, prime)[0][1]
    reverse_na_c1 = (na_c1[0], prime - na_c1[1])

    final_m = add_points_f(a, c2, reverse_na_c1, prime)[:2]
```

```
    return render_template('improved_eceg.html', get_plot = True, plot_url
= 'static/theory/ECEG.png', ec = ec, p = p, na = na, m = m, k = k, qa =
qa, c1 = c1, k_qa = k_qa, c2 = c2, na_c1 = na_c1, bits = bits, ys = ys,
reverse_na_c1 = reverse_na_c1, final_m = final_m, square_ys = square_ys,
 power_ys = power_ys, real_ys = real_ys)
```

Fragmentul de cod 15: Criptosistemul eliptic ElGamal îmbunătățit

Adițional față de algoritmul normal, începând de la pasul 3, calculăm biții ce se atașează la componentele x ale punctelor C_1 și C_2 pentru a se trimite către Alice. Aceasta determină y^2 din ecuația curbei eliptice cu coordonata x primită și apoi din acest pătrat determină coordonata y (aceasta este posibil să nu fie cea inițială, pentru asta verificăm bitul atașat).

Folosim datele din exemplul din subcapitolul 5.6.1 și trecem în paralel prin toate etapele algoritmului cu implementarea din aplicație.

Exemplul 5.8. Pasul 1, configurare: Alice și Bob stabilesc numărul $p = 31$ și curba eliptică E și punctul P

$$E : Y^2 = X^3 + X + 13 \pmod{31}$$
$$P = (9, 10) \in E(\mathbb{F}_{31})$$

Step 1: Configuration

Alice and Bob decide and and publish a prime number $p = 31$, the elliptic curve E over F_{31} and the point $P \in F_{31}$:

$$E : Y^2 = X^3 + 1X + 13 \pmod{31}$$
$$P = (9, 10) \in E(F_{31})$$

Figura 61: Pasul 1 al criptosistemului eliptic ElGamal îmbunătățit

Pasul 2, crearea cheii publice: Alice alege numărul întreg secret $n_A = 5$ și calculează

$$Q_A = n_A P = 5(9, 10) = (25, 16)$$

Alice își publică cheia publică $Q_A = (25, 16)$.

Step 2: Secret key computation

Alice chosses a secret number $n_A = 5$, and calculates her public key

$$\begin{aligned} Q_A &= n_A P \\ &= 5 \cdot (9, 10) \\ &= (25, 16) \in E(F_{31}) \end{aligned}$$

Alice publishes her public key $Q_A = (25, 16)$.

Figura 62: Pasul 2 al criptosistemului eliptic ElGamal îmbunătățit

Pasul 3, criptarea: Bob alege ca mesajul să fie $M = (20, 2)$ și cheia temporală $k = 7$, Bob folosește cheia publică a lui Alice pentru a calcula

$$\begin{aligned} C_1 &= kP = 7(9, 10) = (6, 24) \\ C_2 &= M + kQ_A = (20, 2) + 7(25, 16) \\ &= (20, 2) + (9, 10) = (22, 22) \end{aligned}$$

Cum și $y_{C_1} = 24$ și $y_{C_2} = 22$ sunt mai mari decât $\frac{p}{2} = 15.5$ rezultă $bit_{C_1} = bit_{C_2} = 1$. Bob îi trimite mesajul criptat $((x_{C_1}, bit_{C_1}), (x_{C_2}, bit_{C_2})) = ((6, 1), (22, 1))$ lui Alice.

Step 3: Encryption

Bob chooses the message $M = (20, 2)$ and temporal key $k = 7$

$$\begin{aligned} C_1 &= kP \\ &= 7 \cdot (9, 10) \\ &= (6, 24) \in E(F_{31}) \end{aligned}$$

$$\begin{aligned} C_2 &= M + kQ_A \\ &= (20, 2) + 7 \cdot (25, 16) \\ &= (20, 2) + (9, 10) \\ &= (22, 22) \in E(F_{31}) \end{aligned}$$

Bob sends only the x coordinate of C_1 and C_2 to Alice and an extra bit based on the y coordinate value. If $0 \leq y < p/2$, the $bit = 0$, else the $bit = 1$.
Bob sends Alice the encrypted message $((x_{C_1}, bit_{C_1}), (x_{C_2}, bit_{C_2})) = ((6, 1), (22, 1))$.

Figura 63: Pasul 3 al criptosistemului eliptic ElGamal îmbunătățit

Pasul 4, decriptarea: Alice calculează C_1 și C_2
Alice substituie $x_{C_1} = 6$ în ecuația curbei eliptice E pentru a determina $y^2_{C_1}$

$$\begin{aligned} y^2_{C_1} &\equiv x^3_{C_1} + x_{C_1} + 13 \pmod{31} \\ &\equiv 6^3 + 1 \cdot 6 + 13 \pmod{31} \\ &\equiv 18 \end{aligned}$$

Alice determină y_{C_1}

$$y_{C_1} = 18^{(31+1)/4} \equiv 7 \pmod{31}$$

Cum $bit_{C_1} = 1$ și $y_{C_1} = 7$, adevăratul $y_{C_1} = p - 7 = 24$, deci $C_1 = (6, 24)$.

Step 4: Decryption

Alice first needs to determine the y coordinates of the points C_1 and C_2.

Alice subtitutes $x_{C_1} = 6$ into the equation of E

$$\begin{aligned} y_{C_1}^2 &= x_{C_1}^3 + 1x_{C_1} + 13 \bmod 31 \\ &= 6^3 + 1 \cdot 6 + 13 \bmod 31 \\ &= 18 \end{aligned}$$

Alice needs to compute a square root of 18 modulo 31.
We know that if $t \equiv 3 \bmod 4$, then $z^{(t+1)/4}$ is a square root of z modulo t. Alice determines

$$\begin{aligned} y_{C_1} &= 18^{(31+1)/4} \bmod 31 \\ &= 18^8 \bmod 31 \\ &= 7 \end{aligned}$$

Since $bit_{C_1} = 1$ and $y_{C_1} = 7$, the real $y_{C_1} = 24$. So $C_1 = (6, 24)$.

Figura 64: Pasul 4 al criptosistemului eliptic ElGamal îmbunătățit, determinare C_1

Alice substituie $x_{C_2} = 22$ în ecuația curbei eliptice E pentru a determina $y_{C_2}^2$

$$\begin{aligned} y_{C_2}^2 &\equiv x_{C_2}^3 + x_{C_2} + 13 \pmod{31} \\ &\equiv 22^3 + 1 \cdot 22 + 13 \pmod{31} \\ &\equiv 19 \end{aligned}$$

Alice determină y_{C_2}

$$y_{C_2} = 19^{(31+1)/4} \equiv 9 \pmod{31}$$

Cum $bit_{C_2} = 1$ și $y_{C_2} = 9$, adevăratul $y_{C_2} = p - 7 = 22$, deci $C_2 = (22, 22)$.

$$\begin{aligned} C_2 - n_A C_1 &= (22, 22) - 5(6, 24) \\ &= (22, 22) - (9, 10) \\ &= (22, 22) + (9, 21) = (20, 2) \end{aligned}$$

Rezultatul $(20, 2)$, fiind punctul M ales de Bob.

Alice subtitutes $x_{C_2} = 22$ into the equation of E

$$\begin{aligned} y_{C_2}^2 &= x_{C_2}^3 + 1x_{C_2} + 13 \bmod 31 \\ &= 22^3 + 1 \cdot 22 + 13 \bmod 31 \\ &= 19 \end{aligned}$$

Alice needs to compute a square root of 18 modulo 31. Alice determines

$$\begin{aligned} y_{C_2} &= 19^{(31+1)/4} \bmod 31 \\ &= 19^8 \bmod 31 \\ &= 9 \end{aligned}$$

Since $bit_{C_2} = 1$ and $y_{C_2} = 9$, the real $y_{C_2} = 22$. So $C_2 = (22, 22)$.

Alice decrypts the message, based on C_1 and C_2

$$\begin{aligned} C_2 - n_A C_1 &= (22, 22) - 5 \cdot (6, 24) \\ &= (22, 22) - (9, 10) \\ &= (22, 22) + (9, 21) \\ &= (20, 2) \in E(F_{31}) \end{aligned}$$

The result being the point $M = (20, 2)$ choosen by Bob.

Figura 65: Pasul 4 al criptosistemului eliptic ElGamal îmbunătățit, determinare C_2 și rezultat

5.7 Simulare ECDSA

Pentru a simula algoritmul de semnare digitală pe curbe eliptice am implementat procesul descris anterior în subcapitolul 4.6.

```
if request.method == 'POST':
    #clear the buffer
    plt.clf()

    #coefficients of the elliptic curve
    a = float(request.form['coefficient_curve_a'])
    b = float(request.form['coefficient_curve_b'])

    #prime number in Fp
    prime = int(request.form['prime_p'])

    #details of the elliptic curve
    ec = (int(a), int(b), prime)

    #point G
    gx = request.form['coordinate_g_x']
    gy = request.form['coordinate_g_y']

    g = (int(gx), int(gy))

    #order of point G
    q = order_point_baby_step_giant_step(a, prime, g)[0]

    #Alice's secret signing key
    s = int(request.form['secret_s'])
```

```
    #text message that will be signed
    m = request.form['message_m']

    #hash of the message via SHA-256
    hash_m = sha256(m.encode('utf-8')).hexdigest()

    #the document that will be signed
    int_d = int(hash_m, base=16)
    d = int_d % q

    #Bob's ephemeral number
    e = int(request.form['ephemeral_key_e'])

    #step 2: key creation

    #verification key
    v = multiply_points_f_double_and_add(a, g, s, prime)[0][1]

    #step 3: signing

    #point R
    r = multiply_points_f_double_and_add(a, g, e, prime)[0][1]

    #signatures
    s1 = r[0] % q
    s2 = (d + s * s1) * pow(e, -1, q) % q

    #step 4: verification

    #verifications
    v1 = d * pow(s2, -1, q) % q
    v2 = s1 * pow(s2, -1, q) % q

    #the verification point R'
    term1 = multiply_points_f_double_and_add(a, g, v1, prime)[0][1]
    term2 = multiply_points_f_double_and_add(a, v, v2, prime)[0][1]
    r_prime = add_points_f(a, term1, term2, prime)[:2]
    r_prime_x = r_prime[0] % q

    return render_template('normal_ecdsa.html', get_plot = True, plot_url =
    'static/theory/ECDSA.png', ec = ec, g = g, q = q, s = s, m = m, hash_m
    = hash_m, int_d = int_d, d = d, e = e, v = v, r = r, s1 = s1, s2 = s2,
    v1 = v1, v2 = v2, term1 = term1, term2 = term2, r_prime = r_prime,
    r_prime_x = r_prime_x)
```

Fragmentul de cod 16: Algoritmul de semnare digitală pe curbe eliptice

Folosim datele din exemplul din subcapitolul 4.6 și trecem în paralel prin toate etapele algoritmului cu implementarea din aplicație.

Exemplul 5.9. Pasul 1, configurare: Alice și Bob stabilesc numărul $p = 3797$ și curba eliptică E

și punctul G

$$E : Y^2 = X^3 + 412X + 2356 \text{ (mod } 3797)$$
$$G = (2460, 99) \in E(\mathbb{F}_{3797})$$
$$q = \text{ ord}(G) = 1249$$

Step 1: Configuration

Alice and Bob decide and publish a prime number $p = 3797$, the elliptic curve E over F_{3797} and the point $G \in F_{3797}$:

$$E : Y^2 = X^3 + 412X + 2356 \text{ (mod } 3797)$$
$$G = (2460, 99) \in E(F_{3797})$$
$$q = \text{ord}(G) = 1249$$

Figura 66: Pasul 1 al algoritmul de semnare digitală pe curbe eliptice

Pasul 2, crearea cheii de verificare: Alice alege cheia secretă se semnare $s = 1024$ și calculează

$$V = sG = 1024(2460, 99) = (1753, 1084) \in E(\mathbb{F}_{3797})$$

Alice își publică cheia de verificare $V = (1753, 1084)$

Step 2: Key creation

Alice chosses the secret signing key $s = 1024$, and calculates the verification key

$$\begin{aligned} V &= sG \\ &= 1024 \cdot (2460, 99) \\ &= (1753, 1084) \in E(F_{3797}) \end{aligned}$$

Alice publishes the verification key $V = (1753, 1084)$.

Figura 67: Pasul 2 al algoritmul de semnare digitală pe curbe eliptice

Pasul 3, semnarea: Alice alege ca mesajul $m =$ ”The Book of Five Rings” să fie semnat și determină documentul d ca fiind

$$\begin{aligned} hash(m) &= \text{SHA-256}(m) \\ d &\equiv base_{10}(hash(m)) \text{ (mod } 1249) \\ &\equiv 267 \end{aligned}$$

Step 3: Signing

Alice chooses the message m = "The Book of Five Rings" to be signed, then she computes the hash of the message

$$\begin{aligned}\text{hash}(m) &= \text{SHA-256}(m) \\ &= b3cb03a921f7f0339dead107c315c11fc69dc8d2ecee637587ca21f1ccb9d6f4\end{aligned}$$

Then she determines the document d

$$\begin{aligned}d &= base_{10}(\text{hash}(m)) \bmod 1249 \\ &= 81322695115602024178491823719676790342636482658726481063399593390328736503540 \bmod 1249 \\ &= 267\end{aligned}$$

Figura 68: Pasul 3 al algoritmul de semnare digitală pe curbe eliptice, partea întâi

Alice alege cheia temporală $e = 361$ și calculează punctul R

$$\begin{aligned}R &= eG \\ &= 361 \cdot (2460, 99) \\ &= (3030, 2497) \in E(\mathbb{F}_{3797})\end{aligned}$$

Și semnăturile s_1 și s_2

$$\begin{aligned}s_1 &\equiv R_x \pmod{1249} \\ &\equiv 532 \\ s_2 &\equiv e^{-1}(d + ss_1) \pmod{1249} \\ &\equiv 361^{-1} \cdot (267 + 1024 \cdot 542) \pmod{1249} \\ &\equiv 932\end{aligned}$$

Alice publică semnătura $(s_1, s_2) = (532, 932)$.

Alice chosses the ephemeral key $e = 361$ and computes the point R

$$\begin{aligned}R &= eG \\ &= 361 \cdot (2460, 99) \\ &= (3030, 2497) \in E(F_{3797})\end{aligned}$$

And the signatures

$$\begin{aligned}s_1 &= R_x \bmod 1249 \\ &= 532 \\ s_2 &= e^{-1}(d + ss_1) \bmod 1249 \\ &= 361^{-1} \cdot (267 + 1024 \cdot 532) \bmod 1249 \\ &= 932\end{aligned}$$

Alice publishes the signatures $(s_1, s_2) = (532, 932)$.

Figura 69: Pasul 3 al algoritmul de semnare digitală pe curbe eliptice, partea a doua

Pasul 4, verificarea: Bob determină v_1 și v_2

$$\begin{aligned} v_1 &\equiv ds_2^{-1} \pmod{1249} \\ &\equiv 267 \cdot 932^{-1} \pmod{1249} \\ &\equiv 141 \\ v_2 &\equiv s_1 s_2^{-1} \pmod{1249} \\ &\equiv 532 \cdot 932^{-1} \pmod{1249} \\ &\equiv 1137 \end{aligned}$$

Step 4: Verification

Bob computes v_1 and v_2

$$\begin{aligned} v_1 &= ds_2^{-1} \bmod 1249 \\ &= 267 \cdot 932^{-1} \bmod 1249 \\ &= 141 \\ v_2 &= s_1 s_2^{-1} \bmod 1249 \\ &= 532 \cdot 932^{-1} \bmod 1249 \\ &= 1137 \end{aligned}$$

Figura 70: Pasul 4 al algoritmul de semnare digitală pe curbe eliptice, partea întâi

Apoi, Bob calculează punctul R'

$$\begin{aligned} R' &= v_1 G + v_2 V \\ &= 141 \cdot (2460, 99) + 1137 \cdot (1753, 1084) \\ &= (329, 2357) + (3292, 645) \\ &= (3030, 2497) \in E(\mathbb{F}_{3797}) \end{aligned}$$

Bob verifică semnătura după egalitatea componentelor x ale rezultatelor

$$\begin{aligned} s_1 &\equiv R'.x \pmod{q} \\ 532 &\equiv 3030 \pmod{1249} \\ 532 &\equiv 532 \end{aligned}$$

Pentru că numerele sunt egale semnătura este validă.

Then he computes

$$\begin{aligned} R' &= v_1 G + v_2 V \\ &= 141 \cdot (2460, 99) + 1137 \cdot (1753, 1084) \\ &= (329, 2357) + (3292, 645) \\ &= (3030, 2497) \in E(F_{3797}) \end{aligned}$$

Bob verifies that

$$\begin{aligned} R'_x \bmod q &= s_1 \\ 3030 \bmod 1249 &= 532 \\ 532 &= 532 \end{aligned}$$

Because the numbers are equal, the signature is valid.

Figura 71: Pasul 4 al algoritmul de semnare digitală pe curbe eliptice, partea a doua

5.8 Aproximare spargere ECDLP

Această pagină are rolul de a aproxima în diferite unități de timp cât ar dura să fie rezolvată problema logaritmului discret peste corpuri finite (prezentată în subcapitolul 4.2) pentru un anumit ordin al unei curbe dat de către utilizator. Pentru a prezenta securitatea oferită de problema logaritmului discret peste corpuri finite, comparăm rezultatul în ani cu vârsta universului. De notat faptul că am realizat calculele cu o aproximare destul de mare.

Exemplul 5.10. Spre exemplu, considerăm că avem o curbă eliptică modulo un număr pe 256 de biți, cu ordinul $o = 115792089237316195423570985008687907852837564279074904382605163141518161494337$ (acesta este ordinul protocolului Bitcoin conform [28]). Extragem radicalul din acest număr și va rezulta un număr pe aproximativ 128 de biți, care va fi $\sqrt{o} \approx 10^{37}$. Un CPU de un GHz are o putere de procesare de un miliard de operații pe secundă, ceea ce înseamnă că ar dura $10^{37}/10^9 = 10^{28}$ secunde pentru a sparge ECDLP. Convertim secundele în ani și obținem $10^{28}/(60 \cdot 60 \cdot 24 \cdot 365) = 10^{28}/(3.1 \cdot 10^7) \approx 3.2 \cdot 10^{21}$ ani. Comparăm acest număr cu vârsta universului de $13.7 \cdot 10^9$ ani. Astfel, putem trage concluzia că problema logaritmului discret pe curbe eliptice este foarte sigură.

We will determine the time needed to break the ECDLP for the provided elliptic curve in different time measures based on the Baby Step Giant Step algorithm, which has an complexity of approximately $O(\sqrt{n})$:

$order = 1.157920892373162e + 77 \approx 10^{77.06} \approx 10^{77}$
$\sqrt{order} \approx 10^{38}$

A GHz CPU makes approximately 10^9 operations in a second.
Based on this we can calculate how much time it would take to break the ECDLP:
$10^{38}/10^9 \approx 10^{29}$ seconds
$10^{29}/60 \approx 10^{27.22}$ minutes
$10^{27.22}/60 \approx 10^{25.44}$ hours
$10^{25.44}/24 \approx 10^{24.06}$ days
$10^{24.06}/365 \approx 10^{21.5} \approx 10^{21}$ years

So, it would take $\approx 10^{21}$ years to break ECDLP for the provided elliptic curve's order.
The age of universe is $13.7 \cdot 10^9$ years.
In conclusion, ECDLP is secure.

Figura 72: Aproximarea timpului de spargere a ECDLP

5.9 Simulare spargere ECDLP

În aceast subcapitol vom simula un atac asupra problemei logaritmului discret pe curbe eliptice. Pentru a realiza acest fapt vom folosi algoritmul pași de copil pași de uriaș sau baby step giant step în engleză (prezentat în subcapitolul 4.2 și în [28]).

```
#baby step giant step algorithm to determine the n for Q = nP
def meet_in_the_middle_baby_step_giant_step(a, prime, order, p, q):
    m = int(math.sqrt(order)) + 1
    pk = 0
    steps = 0
    found_pk = False

    #baby steps
    for j in range(1, m):
        jp = multiply_points_f_double_and_add(a, p, j, prime)[0][1]

        #giant steps
        for k in range(1, m + 1):
            multiplier = k*m
            kmp = multiply_points_f_double_and_add(a, p, multiplier, prime)
    [0][1]
            check = add_points_f(a, q, (kmp[0], prime - kmp[1]), prime)[:2]

            #collision found
            if check == jp:
                order_point = order_point_baby_step_giant_step(a, prime, p)
    [0]
                pk = (j + k*m)
                real_pk = (j + k*m) % order_point
                steps = (j - 1) * m + k
                found_pk = True

        if found_pk == True:
            break

    return (pk, real_pk, steps, order_point)
```

Fragmentul de cod 17: Algoritmul pași de copil pași de gigant pentru rezolvarea ECDLP

Returnăm cheia privată n, care verifică ecuația $Q = nP$, însă este posibil ca aceasta să nu fie cea mai mică valoarea pozitivă, de aceea efectuăm modulo ordinul punctului P (care este a doua valoare returnată). Pe lângă aceste valori, returnăm și numărul de pași pe care l-a efectuat algoritmul și ordinul punctului P. Observăm că am dat ca parametru o variabilă $order$, aceasta este ordinul curbei eliptice pe care lucrăm și a fost obținută cu suportul librăriei SimPy.

Exemplul 5.11. Fie numărul prim $p = 4219$ și curba eliptică E

$$E : Y^2 = X^3 + 268X + 1344 \text{ (mod } 4219)$$
$$P = (940, 256) \in E(\mathbb{F}_{4219})$$
$$Q = (1933, 2897) \in E(\mathbb{F}_{4219})$$
$$\text{cu } Q = 1327 \cdot P$$

To determine the private key n, for which $Q = nP$, we have implemented the algorithm Baby Step Giant Step. The algorithm made a total of 416 steps until the first collision.
After that we took the modulo of the number that was returned by the algorithm (initial private key $n = 1327$) by the order of the point (order of point 2132), and resulted in the actual private key.
After this process, the actual wanted private key is $n = log_P Q = 1327$, that means

$$nP = 1327 \cdot (940, 256) = Q(1933, 2897)$$

You can check this in Multiplication in F_p page.

Figura 73: Rezolvarea ECDLP

5.10 Ordinul unui punct

Pentru a determina ordinul unui punct am implementat algoritmul pași de copil pași de uriaș sau baby step giant step în engleză adaptat pentru această sarcină (prezentat în subcapitolul 4.2). Această formă a algoritmului are complexitatea de $O(\sqrt[4]{o})$).

```
#baby step giant step algorithm to determine the order of point
def order_point_baby_step_giant_step(a, prime, p):
    m = int(math.sqrt(math.sqrt(prime))) + 1
    order = 0
    real_order = 0
    steps = 0
    found_order = False

    #baby steps
    for j in range(0, m + 1):
        jp = multiply_points_f_double_and_add(a, p, j, prime)[0][1]

        #giant steps
        for k in range(-m, m + 1):
            multiplier = k*2*m + prime + 1
            check = multiply_points_f_double_and_add(a, p, multiplier,
  prime)[0][1]

            #collision found
            if check[0] == jp[0]:
                order = prime + 1 + k*2*m

                #determine multiple of order
```

```
                if multiply_points_f_double_and_add(a, p, order - j, prime)
    [0][1] == ('inf', 'inf'):
                    order -= j
                    steps = (j - 1) * (2*m + 1) + (m + 1 + k)
                    found_order = True
                    break

                #determine multiple of order
                elif multiply_points_f_double_and_add(a, p, order + j,
    prime)[0][1] == ('inf', 'inf'):
                    order += j
                    found_order = True
                    steps = (j - 1) * (2*m + 1) + (m + 1 + k)
                    break

        if found_order == True:
            break

    #determine real order
    for i in get_divisors(order):
        if multiply_points_f_double_and_add(a, p, i, prime)[0][1] == ('inf'
    , 'inf'):
            real_order = i
            break

    return (real_order, order, steps)
```

Fragmentul de cod 18: Algoritmul pași de copil pași de gigant pentru determinarea ordinului unui punct

Returnăm ordinul $order$, care verifică ecuația $order \cdot P = O(\infty, \infty)$ și numărul de pași pe care l-a efectuat algoritmul. Însă valoarea obținută este posibil să fie un multiplu al ordinului real al punctului. De aceea am construit o funcție suplimentară ce returnează toți divizorii unui număr bazat pe descompunerea în factori primi a acestuia. Cu ajutorul acestei funcții putem testa dacă găsim un divizor ce este ordinul real al punctului.

```
#function that returns the divisors based on the number's factors
def get_divisors(n):
    divisors = [1]
    cn = n
    i = 2

    while i * i <= n:
        if n % i == 0:
            count = 0

            while n % i == 0:
                n //= i
                count += 1

            factors = []
```

```
            for j in range(1, count + 1):
                factors.append(i ** j)

            divs = []
            for d in divisors:
                for f in factors:
                    divs.append(f * d)

            divisors.extend(divs)

        i += 1

    divs = []
    if n != 1:
        for d in divisors:
            divs.append(n * d)

        divisors.extend(divs)

    divisors.sort()
    return divisors
```

Fragmentul de cod 19: Determinarea tuturor divizorilor bazat pe descompunerea în factori primi

Exemplul 5.12. Fie numărul prim $p = 2939$ și curba eliptică E

$$E : Y^2 = X^3 + 1868X + 1273 \pmod{2939}$$
$$P = (128, 32) \in E(\mathbb{F}_{2939})$$
$$\text{cu } 359 \cdot P = O(\infty, \infty)$$

To determine the order of point $P = (128, 32)$, we have implemented the algorithm Baby Step Giant Step. The algorithm made a total of 56 steps until the first collision.
After that we took every divisor of the number that was returned by the algorithm (initial $order = 2872$), because this could be a multiplier of the wanted order of the number.
After this process, the actual wanted order of point $P = (128, 32)$ is 359, that means

$$order \cdot P = 359 \cdot (128, 32) = O('inf', 'inf')$$

You can check this in Multiplication in F_p page.

Figura 74: Ordinul punctului $P = (128, 32) \in E(\mathbb{F}_{2939})$

Observăm că valoarea inițială obținută este $order = 2872$ care verifică $order \cdot P = O(\infty, \infty)$, însă nu este cea mai mică valoare pozitivă cu această proprietate, de aceea verificicăm divizorii numărului $order$ și găsim $359 \cdot P = O(\infty, \infty)$.

5.11 Ordinul unei curbe eliptice peste $\mathbb{F}_p$

Ordinul unei curbe eliptice sau cardinalul acesteia l-am determinat folosind librăria SimPy, mai exact folosind:

```
from sympy.ntheory.elliptic_curve import EllipticCurve
```

Fragmentul de cod 20: Librăria SimPy

Ordinul putea fi determinat și cu algoritmul pași de copil pași de uriaș adaptat, cum a fost realizat în [27].

Exemplul 5.13. Fie numărul prim $p = 3041$ și curba eliptică E

$$E : Y^2 = X^3 + 2574X + 2156 \text{ (mod } 3041)$$

The elliptic curve

$$E : Y^2 = X^3 + 2574X + 2156 \text{ (mod } 3041)$$

has the order of **3120**.
That means that for every point $P \in E(F_{3041})$

$$order \cdot P = 3120 \cdot P = O('inf','inf')$$

You can check this in Multiplication in F_p page.

Figura 75: Ordinul curbei eliptice $E(\mathbb{F}_{3041})$

5.12 Generator curbe eliptice aleatoare

Una din problemele principale ale criptografiei eliptice este că odată cu avansarea tehnologiei cuantice, algoritmii criptografici tradiționali, cum ar fi ElGamal eliptic și ECDSA, ar putea fi vulnerabili la atacurile cuantice puternice. Astfel, dacă nu am putea folosi curbe eliptice cu ordinul mai mare pentru ridica dificultatea, o soluție naivă ar fi crearea unui sistem ce poate schimba curba sau curbele eliptice folosite în securitate înainte ca acestea să fie compromise. În acest scop am creat un generator de curbe eliptice.

```
from sympy.ntheory.elliptic_curve import EllipticCurve
from sympy import randprime
import random

def generate_coefficients(prime):
    a = random.randint(0, prime - 1)
    b = random.randint(0, prime - 1)
```

```
    while (4*a**3 + 27*b**2) % prime == 0:
        a = random.randint(0, prime - 1)
        b = random.randint(0, prime - 1)

    return a, b

def generate_random_elliptic_curve(prime):
    a, b = generate_coefficients(prime)
    e = EllipticCurve(a, b, modulus = prime)
    order_ec = e.order

    return (a, b, order_ec + 1)
```

Fragmentul de cod 21: Generator curbe eliptice

Pentru a crea o curbă eliptică generăm coeficienți a, b aleatori din intervalul $[0, 1, ..., p-1]$, atât timp cât discriminantul este 0. Returnăm coeficienții generați și ordinul curbei eliptice (am adunat 1, deoarece SimPy nu ia în considerare și punctul de la infinit $O(\infty, \infty)$).

Exemplul 5.14. Fie numărul prim $p = 7247$, următoarele sunt exemple de curbe eliptice generate:

The randomly generated elliptic curve is

$$E : Y^2 = X^3 + 5145X + 6508 \pmod{7247}$$

The Order of the generated elliptic curve is 7388.
That means that for every point $p \in E(F_{7247})$

$$order \cdot P = 7388 \cdot P = O('inf', 'inf')$$

You can check this in Multiplication in F_p page.

Figura 76: Exemplul 1 de curbă eliptică generată în $\mathbb{F}_{7247}$

The randomly generated elliptic curve is

$$E : Y^2 = X^3 + 999X + 5999 \pmod{7247}$$

The Order of the generated elliptic curve is 7215.
That means that for every point $p \in E(F_{7247})$

$$order \cdot P = 7215 \cdot P = O('inf', 'inf')$$

You can check this in Multiplication in F_p page.

Figura 77: Exemplul 2 de curbă eliptică generată în $\mathbb{F}_{7247}$

6 Concluzii

Reunind capitolele anterioare, lucrarea de licență a prezentat relevanța și importanța utilizării curbelor eliptice în domeniul criptografiei, acestea oferind niveluri ridicate de securitate și eficiență în comparație cu alte metode criptografice. Un alt atu al criptografiei pe curbe eliptice este performanța oferită de aceasta în domeniu, algoritmii criptografici pe curbe eliptice oferind un echilibru optim între nivelul de securitate și resursele necesare pentru a le calcula. Am prezentat și demonstrat rezistența curbelor eliptice la atacuri cunoscute asupra problemei logaritmului discret, acest aspect asigurând securitatea algoritmilor eliptici. Lucrarea a evidențiat aplicațiile practice ale criptografiei pe curbe eliptice în diverse domenii, cum ar fi transmiterea în siguranță a datelor (schimbul de chei eliptic Diffie-Hellman (ECDH)), criptarea și decriptarea datelor (criptosistemul ElGamal pe curbe eliptice) și semnătura digitală (algoritmul de semnare digitală pe curbe eliptice (ECDSA)). Totuși, curbele eliptice nu sunt importante doar în contextul criptografiei, acestea fiind importante și în alte domenii, precum Blockchain. De asemenea, am implementat o aplicație suport ce calculează și simulează operațiile prezentate pe parcursul lucrării.

Dezavantajul principal al criptografiei pe curbe eliptice este reprezentat de vulnerabilitatea la atacurile cuantice puternice.

O posibilă direcție viitoare a acestei teze ar fi realizarea sintezei și simularea criptositemului Menezes-Vanstone și a criptosistemului RSA pe curbe eliptice (ECRSA) și aprofundarea curbelor Edwards și a proprietăților acestora.

În concluzie, lucrarea de licență a evidențiat beneficiile criptografiei pe curbe eliptice în asigurarea securității comunicațiilor și a datelor, utilizarea curbelor eliptice în algoritmii criptografici reprezintând o alegere populară, eficientă și sigură în fața amenințărilor actuale. Lucrarea a contribuit la asimilarea teoriei criptografiei pe curbe eliptice și a relevanței sale în domeniul criptografiei moderne și înțelegerea tehnică prin algoritmii și operațiile dezvoltate ce prezintă pas cu pas procesul de lucru.

Bibliografie

[1] ANSI-ECDSA. Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA). ANSI Report X9.62, American National Standards Institute, 1998.

[2] Bitcoin Wiki. *Secp256k1*. https://en.bitcoin.it/wiki/Secp256k1, accesat 10.06.2023.

[3] B. Buchanan. *ElGamal and Elliptic Curve Cryptography (ECC)*. https://medium.com/asecuritysite-when-bob-met-alice/elgamal-and-elliptic-curve-cryptography-ecc-8b72c3c3555e, accesat 15.06.2023.

[4] B. Buchanan and J. William. *Elliptic Curve Diffie Hellman (ECDH) with different elliptic curves*. https://asecuritysite.com/ecc/ecdh3, accesat 15.06.2023.

[5] J. W. S. Cassels. *Lectures on Elliptic Curves*, volume 24 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 1991.

[6] Desmos. https://www.desmos.com/calculator, accesat 05.02.2023.

[7] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, IT-22(6):644–654, 1976.

[8] Documentație Bootstrap Navbar. https://getbootstrap.com/docs/4.0/components/navbar/, accesat 08.04.2023.

[9] Documentație Flask. https://flask.palletsprojects.com/en/2.3.x/, accesat 26.12.2022.

[10] Documentație LaTeXMathML. https://www.maths.nottingham.ac.uk/plp/pmadw/lm.html, accesat 12.04.2023.

[11] Documentație MathJax. https://www.mathjax.org/, accesat 12.04.2023.

[12] Documentație Matplotlib. https://matplotlib.org/, accesat 17.01.2023.

[13] Documentație NumPy. https://numpy.org/doc/, accesat 17.03.2023.

[14] Documentație SimPy Number Theory. https://docs.sympy.org/latest/modules/ntheory.html, accesat 25.05.2023.

[15] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31(4):469–472, 1985.

[16] A. Georgescu. *Curs 10, Criptografia bazată pe Curbe Eliptice*, Securitatea Sistemelor Informatice, Curs 2022-2023.

[17] C. L. Gherghe. *Curbe Eliptice 3*, Criptografie si Teoria Codurilor, Curs 2022-2023.

[18] J. Hoffstein et al., *An Introduction to Mathematical Cryptography*, 1 DOI: 10.1007/978-0-387-77994-2 1, @ Springer Science+Business Media, LLC 2008.

[19] J. Jancar. *Secp256k1*. `https://neuromancer.sk/std/secg/secp256k1`, accesat 10.06.2023.

[20] J. Jancar. *secp521r1*. `https://neuromancer.sk/std/secg/secp521r1`, accesat 10.06.2023.

[21] A. W. Knapp. *Elliptic Curves*, volume 40 of *Mathematical Notes*. Princeton University Press, Princeton, NJ, 1992.

[22] S. Lang. *Elliptic Functions*, volume 112 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2nd edition, 1987. With an appendix by J. Tate.

[23] S. Nakov. *Practical Cryptography for Developers*. `https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc`, accesat la 12.04.2023.

[24] R. Pinto. *The Elliptic Curve Digital Signature Algorithm (ECDSA)*. `https://www.linkedin.com/pulse/elliptic-curve-digital-signature-algorithm-ecdsa-rohan-pinto/`, accesat 15.06.2023.

[25] R. Schoof. Counting points on elliptic curves over finite fields. *J. Theor. Nombres Bordeaux*, 7(1):219–254, 1995. Les Dix-huiti'emes Journees Arithmetiques (Bordeaux, 1993).

[26] S. I. Serengil. *Double and Add Method*. `https://sefiks.com/2016/03/27/double-and-add-method/`, accesat la 01.06.2023.

[27] S. I. Serengil. *Counting Points on Elliptic Curves over Finite Field: Order of Elliptic Curve Group*. `https://sefiks.com/2018/02/27/counting-points-on-elliptic-curves-over-finite-field/`, accesat la 28.05.2023.

[28] S. I. Serengil. *Solving Elliptic Curve Discrete Logarithm Problem*. https://sefiks.com/2018/02/28/attacking-elliptic-curve-discrete-logarithm-problem/, accesat la 28.05.2023.

[29] J. H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1986.

[30] J. H. Silverman. *Advanced Topics in the Arithmetic of Elliptic Curves*, volume 151 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1994.

[31] J. H. Silverman. *A Friendly Introduction to Number Theory*. Prentice Hall, Upper Saddle River, NJ, 3rd edition, 2006.

[32] M. S. Stupariu. *Curs 2, Primitive Grafice*, Grafică pe Calculator, Curs 2022-2023.