МАТЕМАТИЧКА ГИМНАЗИЈА

МАТУРСКИ РАД

из предмета

Програмирање и програмски језици

на тему

Криптографија елиптичких кривих

Ученик:
Момчило Топаловић, $IV_{\text{Д}}$

Ментор:
др Филип Марић

Београд, мај 2018.

Садржај

1 Увод

Криптографија је део наше свакодневне рутине. Било да се дописујемо, подижемо паре са банкомата или једноставно користимо интернет, криптографски протоколи нам омогућавају да све те радње обавимо на сигуран начин, чак иако о томе врло често нисмо ни свесни. Због њих можемо да се дописујемо за жељеном особом а да не бринемо да ли још неко има приступ нашој конверзацији, због њих једино ми можемо да приступимо рачуну као и да претражујемо интернет знајући да само ми имамо приступ нашим налозима.

Иако ћемо формалније о криптографији причати у §2, има смисла да дати протокол оцењујемо на основу његове сигурности и времена извршавања. Такође је интуитивно јасно да ако будемо захтевали већу сигурност, време извршавања ће да се повећа.

1.1 Сложеност

Као што смо већ наговестили, потребно је да уведемо неко мерило ефикасности алгоритама. Један начин да то постигнемо је да уведемо *асимптоцку сложеност*, тј. да посматрамо понашање функције дужине извршавања програма од величине улаза n за довољно велике n ($n \to \infty$). Ради једноставности, функцију трајања времена извршавања ћемо звати *временском сложеношћу*. Циљ је да уведемо неку врсту поретка, тј. да видимо када је неки алгоритам са временском сложеношћу f бољи од алгоритма са временском сложеношћу g.

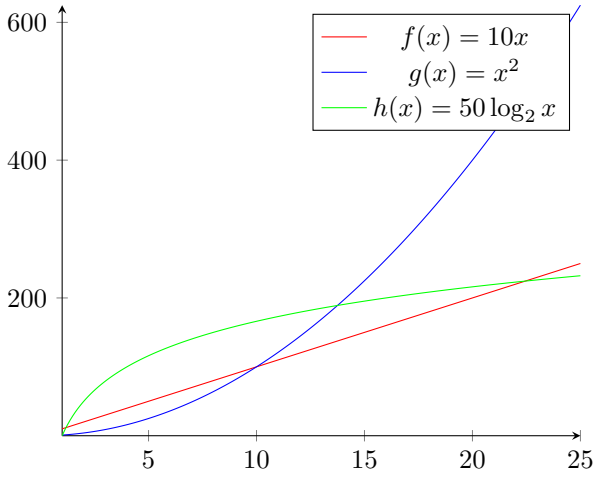
Дефиниција 1.1 (Велика O нотација)**.** *Нека су f и g две функције које сликају $\mathbb{N}$ у $\mathbb{R}^+$. Кажемо да*

$$f(n) = O(g(n)) \quad (n \to \infty)$$

ако постоје позитивне константе n_0 и M такве да $(\forall n > n_0)\ \big(f(n) \leq Mg(n)\big)$.

често ћемо подразумевати да $(x \to \infty)$.
Велика O нотација нам, у неку руку, говори када важи $f \leq g$. Други начин да се протумачи претходна дефиниција је: f је асимптотски ограничена одозго са g.

Пример 1.1.1. Посматрајмо функције $f(x) = 10x$, $g(x) = x^2$ и $h(x) = 50 \log_2 x$. Тада $h = O(f)$ и $f = O(g)$. Приметимо да свака функција има интервал на коме је најмања - g на интервалу $(1, 10)$, f на интервалу $(10, 22)$ и h на $(23, +\infty)$. Као што ћемо видети касније, постоје алгоритми који, иако асимптотски најбржи од тренутно откривених, немају праткичну примену баш из разлога што им је интервал где су они бољи од асимптоцки лошијих облика $(M, +\infty)$, за веома велике M.



Слика 1: Поређење функција f, g и h

Приметимо да $O(f(x))$ представља класу функција које асимптотски не расту брже од f, због чега би можда правилније било писати $f \in O(g(x))$, али се испостави да је ова ознака згодна за рачун - када у неком изразу будемо писали $O(f(x))$ мислмо на неку $g(x)$ тд. $g(x) \in O(f(x))$.

Лема 1.1. *За све функције $f, g, h : \mathbb{N} \to \mathbb{R}^+$ и за сваку константу $c > 0$, важе следећа својства:*

1. $f = O(f)$

2. $f = O(g), g = O(h) \implies f = O(h)$
3. $f = O(g) \implies cf = O(g), f = O(cg)$

Докази ових тврђења су прилично једноставни и следе директно из дефиниције, па су овом раду изостављени.

Пример 1.1.2. Нека је $f(n) = n^3$, $g(n) = 10^{200}n$. Тада важи $g = O(f)$. Приметимо да је g асимптотски мања од f, а да је $f(n) < g(n)$ за $n < 10^{100}$, што је веће од броја честица у видљивом универзуму, па се ни не може дати као улаз неком алгоритму. Шта више, $(\forall n)\ (g(n) \geq 10^{200})$, тј. за било какав улаз, алгоритам са сложеношћу g ће се извршавати милионима година (што не важи за f).

Дефиниција 1.2 (Мала o нотација)**.** *Нека су f и g две функције које сликају $\mathbb{N}$ у $\mathbb{R}^+$. Кажемо да*

$$f(n) = o(g(n)) \quad (n \to \infty)$$

ако за сваку константу $c > 0$ постоји n_0 тд. $(\forall n > n_0)\ \big(f(n) \leq cg(n)\big)$.

Ова нотација нам говори када је $f < g$. За њу важе аналгона тврђења другој и трећој ставци леме 1.1.

Дефиниција 1.3 (Ω нотација)**.** *Нека су f и g две функције које сликају $\mathbb{N}$ у $\mathbb{R}^+$. Кажемо да*

$$f(x) = \Omega(g(x)) \quad (x \to \infty)$$

ако $g(x) = O(f(x))$.

Ω нотација нам, у неку руку, говори када важи $f \geq g$. Други начин да се протумачи претходна дефиниција је: f је асимптотски ограничена одоздо са g.

Дефиниција 1.4 (Θ нотација)**.** *Нека су f и g две функције које сликају $\mathbb{N}$ у $\mathbb{R}^+$. Кажемо да*

$$f(x) = \Theta(g(x)) \quad (x \to \infty)$$

ако $f(x) = O(g(x))$ и $f(x) = \Omega(g(x))$.

Θ нотација нам говори када су неке две функције асимптотски понашају слично.

Пример 1.4.1. Посматрајмо функције $f(x) = 10x$ и $g(x) = 10x + x\sin x$. Тада $f = \Theta(g)$. Приметимо да не мора да постоји $\lim_{x\to\infty} \frac{f(x)}{g(x)}$ - у овом случају $\frac{f(x)}{g(x)}$ варира од 0.9 до 1.1.

Лема 1.2. *За све функције $f, g, h : \mathbb{N} \to \mathbb{R}^+$ важе следећа тврђења:*

1. $f = \Theta(f)$
2. $f = \Theta(g) \implies g = \Theta(f)$
3. $f = \Theta(g), g = \Theta(h) \implies f = \Theta(h)$

Слично као и код леме 1.1, ова лема следи директно из дефиниције, па је изостављен.
Приметимо да наведена три својства подсећају на услове када је нека бинарна релација релације еквиваленције. Следи да Θ дели скуп функција из $\mathbb{N}$ у $\mathbb{R}^+$ на класе еквиваленције где су две функције f и g у истој класи ако $f = \Theta(g)$.
Када будемо описивали ефикасност неког алгоритма, то ћемо радити користећи Θ нотацију, док ћемо Ω и O користити када будемо поредили нека два алгоритма.
Прихваћено је да је неки алгоритам *ефикасан* ако му је временска сложеност ограничена полиномом по величини улаза. Стога уводимо слдеће класе сложености.

Дефиниција 1.5. *Нека је A неки алгоритам са временском сложеношћу f у зависности од величине улаза n. Тада:*

1. *A је полиномне сложености ако $f = O(n^c)$ за неко c.*
2. *A је експоненцијалне сложености ако $f = \Omega(n^c)$ за свако c.*
3. *A је субекспоненцијалне сложености ако $f = O(2^{o(n)})$.*

4. A је пуне експоненцијалне сложености ако $f = \Omega(2^n)$.

Како се већина криптографских система заснива на претпоставци да неки проблем нема ефикасно решење, у проучавању алгоритама који их пробијају користимо следећу нотацију.

Дефиниција 1.6. *L-нотација је асимптоцка нотација дефинисана на следећи начин у зависности од величине улаза n:*

$$L_n[\alpha, c] = e^{(c+o(1))(\ln n)^{\alpha}(\ln \ln n)^{1-\alpha}}.$$

Приметимо да за $\alpha = 0$ добијамо полиномне сложеност по $\log n$, а да за $\alpha = 1$ добијамо пуну експоненцијалну, То значи да је α добар индикатор колико смо близу полиномне сложености.
За крај, навешћемо теорему коју ћемо користити у анализи сложености неких алгоритама.

Теорема 1.1 (Мастер теорема)**.** *За дату рекурентну једначину*

$$T(n) = aT\Big(\frac{n}{b}\Big) + f(n)$$

важи:

$$T(n) = \begin{cases} \Theta(n^{\log_b a}), & f(n) = \Theta(n^{\log_b a - \varepsilon}) \\ \Theta(n^{\log_b a} \log^{k+1} n), & f(n) = \Theta(n^{\log_b a} \log^k n) \\ \Theta(f(n)), & f(n) = \Theta(n^{\log_b a + \varepsilon}) \end{cases}$$

Доказ. Нека је $n = b^{\alpha}$. Тада је

$$T(b^{\alpha}) = aT(b^{\alpha-1}) + f(b^{\alpha}) = \sum_{k=0}^{\alpha} a^{\alpha-k} f(b^k).$$

Ако је $f(n) = \Theta(n^{\log_b a - \varepsilon})$, тада је $f(b^x) = a^{x(1-\varepsilon')}$, где је $\varepsilon' = \varepsilon / \log_b a$, па важи:

$$\begin{aligned} T(n) = T(b^{\alpha}) &= \sum_{k=0}^{\alpha} \Theta\big(a^{\alpha-k} a^{k(1-\varepsilon')}\big) = \Theta\Big(a^{\alpha} \sum_{k=0}^{\alpha} a^{-k\varepsilon'}\Big) \\ &= \Theta\Big(a^{\alpha} \frac{1 - a^{-\varepsilon'(\alpha+1)}}{1 - a^{-\varepsilon'}}\Big) = \Theta(a^{\alpha}) = \Theta(n^{\log_b a}). \end{aligned}$$

Приметимо да овде сложеност потиче од листова којих има $\Theta\big(n^{\log_b a}\big)$.
Ако је $f(n) = \Theta(n^{\log_b a} \log^k n)$ тада је $f(b^x) = \Theta(x^k a^x)$, па важи:

$$T(b^{\alpha}) = \sum_{k=0}^{\alpha} \Theta\big(x^k a^{\alpha}\big) = \Theta\big(x^{\alpha+1} a^{\alpha}\big) = \Theta\big(n^{\log_b a} \log^k n\big)$$

Овде је сваки слој мање-више једнак, па је сложеност $f(n) \log n$.
Ако је $f(n) = \Theta(n^{\log_b a + \varepsilon})$, тада сличим рачуном као у првом случају добијамо да важи:

$$T(b^{\alpha}) = \Theta\Big(a^{\alpha} \frac{1 - a^{\varepsilon'(\alpha+1)}}{1 - a^{\varepsilon'}}\Big) = \Theta\big(a^{\alpha+\varepsilon'}\big) = \Theta\big(n^{\log_b a + \varepsilon}\big) = \Theta(f(n)).$$

Приметимо да овде сложеност потиче од корена стабла. □

Пример 1.6.1 (Сложеност merge sort-а)**.** Претпоставимо да нам је дат неки низ $(a_i)_{i=1}^n$ који желимо да сортирамо. Основна идеја merge sort-а је да поделимо низ на два дела једнаких дужина, рекурзивно их сортирамо, и на крају спојимо. Функцију merge која спаја два сортирана низа у један можемо имплементирати на следећи начин:

$$\begin{aligned} \text{merge}([\quad], (b_i)_{i=1}^B) &= (b_i)_{i=1}^B \\ \text{merge}((a_i)_{i=1}^A, [\quad]) &= (a_i)_{i=1}^A \\ \text{merge}((a_i)_{i=1}^A, (b_i)_{i=1}^B) &= \begin{cases} [a_1, \text{merge}((a_i)_{i=2}^A, (b_i)_{i=1}^B)] & \text{, ако } a_1 < b_1 \\ [b_1, \text{merge}((a_i)_{i=1}^A, (b_i)_{i=2}^B)] & \text{, иначе} \end{cases} \end{aligned}$$

Дакле, merge можемо имплементирати у временској сложености $\Theta(n)$. Посматрајмо псеудо-код следећег алгоритма за сортирање низа.

Алгоритам 1.1.1 Merge sort

Улаз: $(a_i)_{i=1}^n$
Излаз: сортирани низ $(b_i)_{i=1}^n$

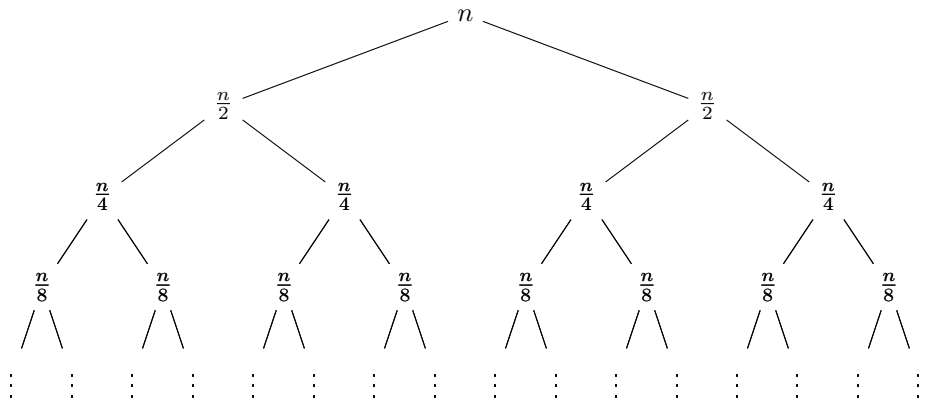
1: **ако** $n = 1$ **онда**
2: **врати** (a_1)
3: $k \leftarrow \lfloor \frac{n}{2} \rfloor$
4: $x \leftarrow \text{mergesort}((a_i)_{i=1}^k)$
5: $y \leftarrow \text{mergesort}((a_i)_{i=k+1}^n)$
6: $b \leftarrow \text{merge}(x, y)$
7: **врати** b

Нека је $T(n)$ временска сложеност merge sort-а. Како функција mergesort за улаз величине $n > 1$ има два рекурзивна позива за улаз величине $\lfloor \frac{n}{2} \rfloor$ и $\lceil \frac{n}{2} \rceil$, као и позив функције merge за улаз величине n следи да важи:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n).$$

Овде можемо применити први случај теореме 1.1, па важи:

$$T(n) = \Theta(n \log n).$$



Слика 2: Индуковано стабло за рекурентну $T(n) = 2T(\frac{n}{2}) + n$

2 Увод у криптографију

Иако смо навели зашто је криптографија нама битна, историјски, главни покретач развоја ове области је био војни код. Војска која је могла да сазна позиције непријатељских трупа, не одавајући своје је била у огромној предности.
Ради једноставности, под *отвореним текстом* подразумевамо оригинални текст (поруку) коју желимо да пошаљемо, док под термином *шифрат* подразумевамо енкриптовани текст који шаљемо.
Такође ћемо, ради прегледности, промењиве познате потенцијалном нападачу приказивати плавом бојом, док ћемо промењиве које су у датом тренутку њему непознате приказивати црвеном.

Пример 2.0.1 (Цезарова шифра)**.** Нека је дат низ карактера $(a_i)_{i=1}^n$ који желимо да пошаљемо ($a_i \in \{0, 1, \cdots, \alpha - 1\}$, α је број знакова у језику). Тада је порука коју ми саљемо само померена за k карактера. Ако је нпр. $k = 3$ тада а↦г, х↦п, итд. Ако смо хтели да шифрујемо поруку *Alea iacta est*[1], добили бисмо *Epie megxe iwx*. Ако бисмо сада желели да дешифрујемо, потребно је само да обрнемо процес - померимо све карактере за k карактера у другом смеру.
Формално, имамо функције за енкрипцију (e_k) и декрипцију (d_k) дефинисане као $e_k(x) = (x + k) \mod \alpha$ и $d_k(x) = (x - k) \mod \alpha$ (α је број карактера у језику).

а	б	в	г	⋯	ц	ш
г	д	ђ	е	⋯	б	в

(а) енкрипција

а	б	в	г	⋯	ц	ш
ч	ц	ш	а	⋯	х	ц

(б) декрипција

Слика 3: шематски приказ Цезарове шифре (српска азбука)

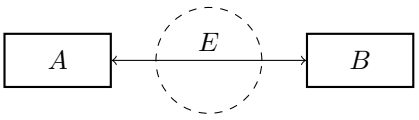
Овим смо постигли неколико ствари:

- Само наши савезници могу да сазнају шта ту пише.
- Наши непријатељи не могу да пошаљу такву поруку (потребан им је кључ k, као и начин енкрипције).

2.1 Основни модел

Да бисмо моделовали сва дешавања приликом неке комуникације, користићемо следећи модел.

- Две стране (A и B) размењују податке.
- Једине информације које оне могу да пошаљу су преко једног канала.
- Прислушкивач (E) у сваком тренутку може да:
  - прочита све пакете постлате преко канала,
  - измени било који пакет послат преко канала.
- E има значајне процесорске способности.
- E зна описе свих коришћених протокола.



Слика 4: Основни криптографски модел

Остаје да одговоримо на питање шта захтевамо од неког криптографског система да бисмо га сматрали безбедним. Можемо, у зависности од потребе, поставити неке од следећих захтева (претпостављамо да A саље поруку B-у)

- *Тајност* - E не може да добије отворени текст из шифрата.
- *Интегритет података* - B може да буде сигуран да E није мењао податке
- *Аутентикација порекла података* - B може да буде сигуран да је A послао поруку.

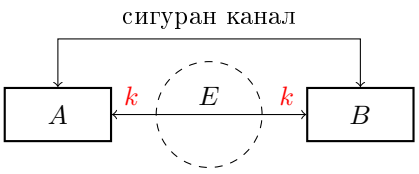
[1] лат. Коцка је бачена - Јулије Цезар 10. јан. 49 п.н.е.

- *Потврда ентитета* - B може да буде уверен у идентитет A.
- *Немогућност избегавања одговорности* - A не може да негира чињеницу да је послао поруку
- *Анонимност* - B не може да сазна ко му шаље поруку

2.2 Симетрична криптографија

Протоколи коришћени у симетричној криптографији ослањају се на алгоритме који користе исти кључ и за енкрипцију и за декрипцију. Да би и A и B добили приступ истом тајном кључу, они морају да пре тога комуницирају преко сигурног канала (нпр. курир коме верујемо). Главна мана овог модела је управо то што морамо да обезбедимо сигуран канал, док је главна предност то што су значајно бржи.
Неки од примери коришћених алгоритама укључују:

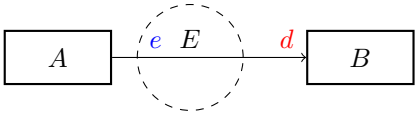
- *Data Encryption Standard* (DES)
- *Advanced Encryption Standard* (AES)
- *Rivest Cipher 4* (RC4)



Слика 5: Основни модел симетричне криптографије

2.3 Асиметрична криптографија

Протоколи коришћени у асиметричној криптографији ослањају се на алгоритме који користе различите кључеве за енкрипцију (јавни) и декрипцију (тајни). Претпоставимо да A жели да пошаље поруку B-у. Тада B генерише пар кључева (e, d), и објављује кључ за шифровање e, док кључ за дешифровање d чува за себе. На овај начин свако може да пошаље B-у поруку тако да нико други не може да их растумачи (зато што само он има тајни кључ). Због овога се асиметрична криптографија чешће зове криптографија јавног кључа. Као што ћемо видети касније, овај модел се ослања на претпоставкама да су одређени проблеми из теорије бројева тешки.



Слика 6: Основни модел асиметричне криптографије

Системе са јавним кључем можемо грубо поделити у три групе:

- RSA системе
- DL (*discrete logarithm*) системе
- EC (*elliptic curve*) системе

2.3.1 RSA системи

Ривест, Шамир и Адлеман су 1977. предложили овај систем [1]. Иако је Клифорд Кокс предложио еквивалентан систем 1973, пошто је радио за британску тајну службу, његов рад је објављен за ширу јавност тек 1997. У овом делу рада ћемо представити основну верзију алгоритма. Напредније верзије се ослањају на исте претпоставке, те нам није у овом раду од интереса. Сам алгоритам се ослања на следећу теорему.

Теорема 2.1 (Ојлерова теорема)**.** *За дати пар узајамно простих бројева a и n важи:*

$$a^{\varphi(n)} \equiv 1 \mod n,$$

где је φ Ојлерова функција ($\varphi(n) = |\{x \leq n \mid NZD(x, n) = 1\}|$).

Претпоставимо да Алиса жели да пошаље поруку m Бобу, тако да Ева не може да прочита m. На почетку Боб генерише јавни кључ (e, n) и приватни кључ (d, n). Јавни кључ ће објавити тако да свако може да Бобу пошаље шифровану поруку, али једино он може да је дешифрује користећи

приватни кључ. Он то ради тако што прво генерише два проста броја p и q. Њих користи како би израчунао $(\phi, n) = ((p-1)(q-1), pq)$. Затим бира произвољно e тд. $1 < e < \phi$ и $\text{NZD}(e, \phi) = 1$ и израчунава $d = e^{-1} \mod \phi$. Овим је Боб успешно израчунао приватни (d, n) и јавни кључ (e, n), након чега објављује јавни кључ (e, n).

Након тога Алиса шаље $m^e \mod n$ Бобу. Боб израчунава $(m^e)^d = m^{ed}$, што је по теореми [2.1] једнако m.

RSA проблем представља проблем проналажења отвореног текста m на основу шифрата m^e, као и јавног кључа (e, n). Доказано је да је проналажење приватног кључа (d, n) из јавног кључа (e, n) еквивалентно за проблемом факторисања броја n (*integer factorization problem*, IFP). Иако није доказано, претпоставља се да је RSA проблем еквивалентан са IFP.

A зна	A рачуна	E	B рачуна	B зна
m			p, q	p, q
m			$\phi = (p-1)(q-1)$	p, q, ϕ
m			$n = pq$	p, q, ϕ, n
m			e	p, q, ϕ, n, e
m			$d = e^{-1} \mod \phi$	p, q, ϕ, n, e, d
m, n, e		$\xleftarrow{e,n}$		p, q, ϕ, n, e, d
m, n, e, M	$M = m^e \mod n$			p, q, ϕ, n, e, d
m, n, e, M		$\xrightarrow{M}$		p, q, ϕ, n, e, d, M
m, n, e, M			$m = M^d \mod n$	$p, q, \phi, n, e, d, M, m$

Слика 7: шематски приказ основног RSA алгоритма

Напомена 2.1 (Избор простих бројева)**.** *Ради сигурности бирамо l-тоцифрене p и q, где се обично узима да је $l \in \{1024, 2048, 4096\}$. Остаје још да образложимо како да изаберемо p и q тако да они буду прости. Нека је $\pi(n)$ број простих мањих од n. Како је $\pi(n) \sim \frac{n}{\ln n}$ [2], закључујемо да је једна од стратегија да узимамо насумично бројеве и тестирамо да ли су прости.*

Тренутно најбољи алгоритми који решавају IFP укључују:

- General number field sieve [3]- $L_n[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}]$
- Multiple polynomial quadratic sieve [4]- $L_n[\frac{1}{2}, 1]$
- Lenstra's elliptic-curve factorization algorithm [5] - $L_n[\frac{1}{2}, \sqrt{2}]$

2.3.2 DL системи

Ослањају се на *discrete logarithm problem* (DLP), тј. проблем израчунавања x ако нам је дато $a^x \mod p$ (знамо $\{a, p, a^x \mod p\}$). Један алгоритам за шифровање који се базира на DLP-у дао је Тахер Елгамал[2][6]. Претпоставимо да су нам дати доменски параметри (p, q, g) који редом представљају модул (p мора бити прост), ред генератора и генератор. Такође, претпоставимо да смо успешно генерисали приватни кључ x и јавни кључ $y = g^x \mod p$.

Алгоритам 2.3.1 Основна ElGamal енкрипција

Улаз: Доменски параметри (p, q, g), јавни кључ y, отворени текст m
Излаз: Шифрат (c_1, c_2)

1: $k \in_R \{1, 2, \cdots q-1\}$
2: $c_1 \leftarrow g^k \mod p$
3: $c_2 \leftarrow my^k \mod p$
4: **врати** (c_1, c_2)

[2] Taher Elgamal, египатски криптолог

Алгоритам 2.3.2 Основна ElGamal декрипција

Улаз: Доменски параметри (p, q, g), приватни кључ x, шифрат (c_1, c_2)
Излаз: Отворени текст m

1: **врати** ${c_1}^{-x} c_2 \mod p$

Доказ. Треба доказати да је ${c_1}^{-x} c_2 \mod p = m$. Како $c_1 \equiv g^k \mod p$ и $c_2 \equiv m y^k \mod p$ имамо:

$$ {c_1}^{-x} c_2 \equiv g^{-kx} m y^k \equiv g^{-kx} m g^{kx} \equiv m \mod p $$

□

2.3.3 EC системи

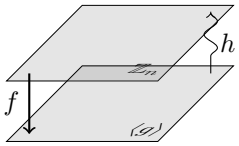
Као што смо већ напоменули, DL системи се заснивању на претпоставци да је DL проблем ($b \equiv a^x$ у групи $(\mathbb{Z}_p, \cdot)$) тежак. Посматрајмо сличан систем у коме смо узели неку групу која није $(\mathbb{Z}_p, \cdot)$. Приметимо да избор групе $(\mathbb{Z}_p, +)$ не би био добар пошто би се овакав систем заснивао на претпоставци да је $xa = b$ тежак. Ако бисмо могли да израчунамо мултипликативни инверз броја a, тада би се овај проблем свео на

$$ x = a^{-1} b. $$

Међутим, као што ћемо видети у §3.3, ови проблеми се могу ефикасно решити.
Дакле, треба нам група $G = \langle g \rangle$ реда n у коме је тешко израчунати дискретан логаритам. Такође, треба нам и нека функција $f : x \mapsto g^x$, као и h која је функција чије је рачунање лако, а има релативно мало парова $x \neq y$ таквих да је $h(x) = h(y)$ (идеално би било када би h била 1-1). Ову функцију ћемо звати int и њен значај ћемо видети тек у §5.

Сигурност овакве групе се своди на тежину рачунања f^{-1}, док се брзина своди на лакоћу израчунавања f и g. Већ смо видели да је један пример таког система $(G, f, g) = ((\mathbb{Z}_p, \cdot), x \mapsto g^x \mod p, x \mapsto x)$. Касније ћемо видети да је још једна таква тројка $(G, f, g) = (\langle P \rangle, x \mapsto xP, (x, y) \mapsto \mathrm{int}(x))$, где је P нека тачка на некој елиптичкој кривој, а int нека функција која слика у $\mathbb{N}$ (ако је аргумент већ у скупу природних бројева, онда је int идентитет, али то не мора увек да важи).

У овом раду ће нам од сада главни фокус бити да објаснимо како и зашто овај систем функционише. Како се свака операција у групи $\langle P \rangle$ своди на серију операција у коначном пољу над којим је $\langle P \rangle$ дефинисано, прво ћемо у §3 причати о аритметици коначних поља. После тога ћемо у §4 објаснити како представљамо елиптичке криве, тачке на њој и како оперишемо са њима. На крају ћемо, у §5, дати примере неких алгоритама који се базирају на елиптичким кривама.



Слика 8: шематски приказ DL система у општој групи

3 Аритметика коначних поља

Као што ћемо видети у делу §4, свака елиптичка крива је дефинисана над неким пољем. У интересу су нам само коначна поља како бисмо могли да гарантујемо да ће се алгоритми описану у §5 завршити.

3.1 Групе

Како свако поље чине две групе, прво ћемо увести појам групе и извести неколико основних тврђења која ће нам бити корисна касније.

Дефиниција 3.1. *Група $(G, \oplus)$ се састоји из скупа G заједно са бинарном операцијом $\oplus : G \times G \to G$ који задовољавају следеће аксиоме:*

1. *(Асоцијативност):* $(\forall a, b, c \in G)\ (a \oplus (b \oplus c) = (a \oplus b) \oplus c)$
2. *(Постојање неутрала):* $(\exists e \in G)(\forall a \in G)\ (a \oplus e = e \oplus a = a)$
3. *(Постојање инверза):* $(\forall a \in G)(\exists b \in G)\ (a \oplus b = e)$

Често ћемо групу $(G, \oplus)$ звати само G ако се из контекста може закључити шта је операција. Такође ћемо неутрал често писати као 0 ако је операција сабирање, или као 1 ако је операција множење.
Приметимо да у општем случају елементи групе не комутирају ($(\forall a, b \in G)\ (a \oplus b = b \oplus a)$). Ако елементи групе G комутирају, онда ту групу зовемо *Абеловом*[3] или *комутативном*.
Под *редом* коначне групе G подразумевамо $|G|$, док под *редом* елемента x коначе групе G, у ознаци $\mathrm{ord}(x)$, подразумевамо $\min\{k > 0 \mid x^k = e\}$.

Пример 3.1.1. Типични примери група су: $(\mathbb{Z}, +)$, $(\mathbb{R}, +)$, $(\mathbb{R}\setminus\{0\}, \cdot)$,$(\mathbb{C}, +)$, $(\mathbb{C}\setminus\{0\}, \cdot)$. Приметимо да $(\mathbb{Z} \setminus \{0\}, \cdot)$ није група зато што не постоји инверз елемента 2, као ни $(\mathbb{R}, \cdot)$ зато што не постоји инверз елемента 0.

Пример 3.1.2. Пример још једне групе је $(\mathbb{Z}_n, +)$. Занима нас да ли је $(\mathbb{Z}_n \setminus \{0\}, \cdot)$ исто група за свако n. Нека је n сложен. Тада се n може записати као $n = ab$ за неке $a, b > 1$. Међутим, тада не постоји инверз елемената a и b. Овим смо доказали да $(\mathbb{Z}_n \setminus \{0\}, \cdot)$ није група за сложено n. Међутим, испоставља се да $(\mathbb{Z}_p \setminus \{0\}, \cdot)$ јесте група за просто p.

Пример 3.1.3. Посматрајмо уређени пар $(GL_n(\mathbb{R}), \cdot)$, тј. скуп инвертибилних матрица степена n за неко фиксирано n над пољем $\mathbb{R}$ у односу на множење. Како овако дефинисан уређени пар задовољава дефиницију [3.1], а њени елементи не комутирају, ово је пример некомутативне групе.

Пример 3.1.4. Кажемо да је дата функција f аритметичка ако она слика $\mathbb{N}$ у $\mathbb{C}$. За дату аритметичку функцију кажемо да је мултипликативна ако важи:

- $f(1) = 1$
- Ако $(a, b) = 1$ тада $f(ab) = f(a)f(b)$

Дефинишимо Дирихлеову[4] конволуцију $(*)$ као операцију над аритметичким функцијама која задовољава:

$$(g * f)(n) = \sum_{d \mid n} g(d) f\left(\frac{n}{d}\right).$$

Лако се види да је $(A, *)$ комутативна група са неутралом

$$\epsilon(n) = \begin{cases} 1, & \text{ако } n = 1 \\ 0, & \text{иначе} \end{cases}.$$

Приметимо да је инверз функције 1 $(1(n) = 1)$ Мебијусова[5] функција μ.

[3] Niels Henrik Abel, норвешки математичар, 1802-1829.
[4] Johann Peter Gustav Lejeune Dirichlet, немачки математичар, 1805-1859.
[5] August Ferdinant Möbius, немачки математичар, 1790-1868.

Дефиниција 3.2. *Кажемо да су две групе $(G, \oplus)$ и $(H, \otimes)$ изоморфне ако постоји бијекција $f : G \to H$ таква да*

$$(\forall a, b \in G)\ (f(a \oplus b) = f(a) \otimes f(b)).$$

Функцију f ћемо звати изоморфизмом.

Пример 3.2.1. Нека је прва група $(\mathbb{R}, +)$, а друга $(\mathbb{R}^+, \cdot)$. Како важи

$$e^{x+y} = e^x \cdot e^y$$

и како је e^x бијекција између $\mathbb{R}$ и $\mathbb{R}^+$, закључујемо да су ове две групе изоморфне.

Дефиниција 3.3. *Директан производ група $(G, \oplus)$ и $(H, \otimes)$ је група $(G \times H, \cdot)$ где*

$$(g_1, h_1) \cdot (g_2, h_2) = (g_1 \oplus g_2, h_1 \otimes h_2)$$

Дефиниција 3.4. *Циклична група $G = \langle a \rangle$ са генератором a је група $\{a^n \mid n \in \mathbb{Z}\}$*

Теорема 3.1. *Свака циклична група $\langle a \rangle$ коначног реда n је изоморфна са $\mathbb{Z}_n$.*

Доказ. Посматрајмо функцију $f : \langle a \rangle \to \mathbb{Z}_n$ дефинисану као:

$$f(a^x) = x \mod n$$

Лако се види да је f добро дефинисана бијекција. Како $f(a^x \cdot a^y) = f(a^{x+y}) = (x + y) \mod n = f(a^x) + f(b^y)$, видимо да је f изоморфизам. □

Пример 3.4.1. Посматрајмо групу групу $(\mathbb{Z}_{17} \setminus \{0\}, \cdot)$. Она је генерисана генератором $g = 3$. Нека је f функција описана у доказу теореми 3.1. Тада је

$$4 \equiv f(12) \equiv f(7 + 5) \equiv f(7)f(5) \equiv 11 \cdot 5 \equiv 4 \mod 17$$

3.2 Коначна поља

Дефиниција 3.5. *Поље $(F, +, \cdot)$ се састоји из скупа F заједно са две бинарне операције $+ : F \times F \to F$ и $\cdot : F \times F \to F$ које задовољавају следеће аксиоме:*

1. *$(F, +)$ је Абелова група*
2. *$(F \setminus \{0\}, \cdot)$ је Абелова група*
3. *(Дистрибутивни закон): $(\forall a, b, c \in F)(a + b) \cdot c = (a \cdot c) + (b \cdot c)$*

Пример 3.5.1. Неки примери поља су: $(\mathbb{Q}, +, \cdot)$, $(\mathbb{Z}_p, +, \cdot)$, $(\mathbb{R}, +, \cdot)$ и $(\mathbb{C}, +, \cdot)$. Приметимо да $(\mathbb{Z}, +, \cdot)$ није поље зато што $(\mathbb{Z}, \cdot)$ није група, као ни да $(\mathbb{H}, +, \cdot)$ зато што елементи групе $(\mathbb{H}, \cdot)$ не комутирају, где је $\mathbb{H}$ скуп кватерниона.

Аналогно као код група уводимо *ред* поља. Инверз сабирања елемента a зовемо *адитивним инверзом* и обележавамо са $-a$. Инверз множења елемента a зовемо *мултипликативним инверзом* и обележавамо са a^{-1}.

Дефиниција 3.6. *Карактеристика поља F је $|\langle 1 \rangle|$ при сабирању ако је $\langle 1 \rangle$ коначан скуп и 0 у супротном.*

Теорема 3.2. *Коначно поље реда n постоји акко је $n = p^\alpha$ за неки прост број p и тада је такво поље јединствено одређено (до на изоморфизам). Такође, важи да је карактеристика таквог поља једнака p.*

Дефиниција 3.7. *Коначно поље $\mathbb{F}_n$ зовемо простим ако је n прост.*

Приметимо да на основу теореме [3.2] следи да су једина (до на изоморфизам) проста поља облика $\mathbb{Z}_p$.

Дефиниција 3.8. *Кажемо да је коначно поље $\mathbb{F}_q$ екстензионо ако q није прост*

На основу теореме [3.2] видимо да таква поља постоје за $n = p^\alpha, \alpha > 1$. Следећа теорема нам даје пример таквог поља за свако такво n, а самим тим и сва екстензиона коначна поља.

Теорема 3.3. *Нека је S скуп свих полинома степена мањег од α над пољем $\mathbb{Z}_p$ и нека је $g(x)$ неки иредуцибилни полином (над $\mathbb{Z}_p$) степена α. Тада је $(S, +, \cdot_g)$ поље, где је $\cdot_g$ множење по модулу g.*

Иако наведено тврђење нећемо доказивати, биће нам потребно да ефикасно нађемо један такав полином g.

Лема 3.1. *Полином g описан у теореми [3.3] постоји.*

Доказ. Нека је $N_p(m)$ број моничних иредуцибилних полинома степена m над $\mathbb{F}_p$. Тада

$$N_p(m) = \frac{1}{m} \sum_{d \mid n} \mu(d) p^{m/d}.$$

Такође, важи и

$$\frac{1}{2m} \leq \frac{N_p(m)}{p^m} \approx \frac{1}{m}.$$

□

Други део леме [3.2] нам говори да је број таквих полинома довољан да је једна стратегија да једноставно узимамо насумичне полиноме и проверавамо да ли су иредуцибилни или не. За проверу иредуцибилности можемо користити Рабинов тест, који се заснива на следећој теореми

Теорема 3.4. *Нека је $f(x)$ неки полином степена n над $\mathbb{F}_p$. Тада је f иредуцибилан акко:*

- $f \mid (x^{p^n} - x)$,
- *За сваког простог делиоца q броја n важи $NZD(f, x^{p^{n/q}} - x) = 1$.*

Пример 3.8.1. Нека је $p = 2$ и $n = 2$. Како је $x^2 + 1 = (x + 1)^2$ (у $\mathbb{Z}_2$), полином $f(x) = x^2 + 1$ није иредуцибилан. У овом случају не важи први услов - $x^4 - x = (x^2 + 1)^2 + x + 1$.

Пример 3.8.2. Нека је $p = 3$ и $n = 3$. Како је $x^3+2 = x(x+1)(x+2)$ (у $\mathbb{Z}_3$), полином $f(x) = x^3+2$ није иредуцибилан. Полином $x^{27} - x$ се факторише на следећи начин,

$$\begin{aligned} x^{27} - x =& x(x+1)(x+2)(x^3+2x+1)(x^3+2x+2)(x^3+x^2+2)(x^3+x^2+x+2) \\ & (x^3+x^2+2x+1)(x^3+2x^2+1)(x^3+2x^2+x+1)(x^3+2x^2+2x+2), \end{aligned}$$

те важи први услов. Међутим приметимо да се за $q = 3$ други услов своди на $\text{NZD}(x^3-x, x^3+2) = 1$, што, како су то исти полиноми, не важи.

Дефиниција 3.9. *Бинарно поље је свако поље чија је карактеристика једнака 2.*

Како је адитивни инверз сваког елемента у пољу $\mathbb{Z}_2$ једнак самом себи, на основу теореме 3.2 и модела 3.3 закључујемо да

$$(\forall x \in \mathbb{F}_{2^m})\ (-x = x).$$

3.3 Аритметика у простим пољима

У овом делу ћемо објаснити како да репрезентујемо бројеве у оперативној меморији, као и како да вршимо *елементарне операција* над њима. Под елементарим операцијама мислимо на:

- сабирање
- тражење адитивног инверза (тј. одузимање)
- множење
- тражење мултипликативног инверза

Ради једноставности, до краја овог поглавља, сматрамо да сви бројеви припадају фиксном пољу $\mathbb{F}_p$.

3.3.1 Репрезентација

Прво морамо да видимо како можемо да репрезентујемо бројеве у оперативној меморији. Претпоставимо да имамо неки број $A = (a_{n-1} \cdots a_1 a_0)_B$ за неку базу B. Тада ћемо број A да памтимо као низ цифара a. Углавном се узима да је B величине једног регистра, што је углавном 2^{64}.

Такође, биће нам занимљиво још једно тумачење ове репрезентације. Наиме, нека је $P(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$. Тада је $A = P(B)$. Ово можда делује као да смо само написали шта значи да је A написан у бази B, али нам омогућава да множимо бројеве тако што множимо полиноме, што се испоставља да лакше.

Од сада, па до краја поглавља, претпостављамо да је n фиксирано. То можемо зато што можемо само додати водеће нуле. За n можемо узети

$$n = \lceil \log_B p \rceil.$$

3.3.2 Сабирање и одузимање

Како у сабирању имамо пренос два броја ($7 + 5 = 2$ са преносом 1), увешћемо ознаку $(x, \varepsilon) \leftarrow y$ где је

$$x = y \mod B$$
$$\varepsilon = \begin{cases} 0 & \text{, ако } 0 \leq y < B \\ 1 & \text{, иначе} \end{cases}.$$

Прво дајемо помоћне алгоритме за сабирање и одузимање. Они подсећају на алгоритме које и ми користимо, па у овом делу неће бити објашњено како и зашто раде.

Алгоритам 3.3.1 Сабирање са преносом

Улаз: Бројеви $a, b \in \{0, 1, 2, \cdots p-1\}$
Излаз: $(a + b \mod B^n, \varepsilon)$, где је ε пренос

1: $\varepsilon \leftarrow 0$
2: **за** $i \in \{0, 1, 2, \cdots n-1\}$
3: $\quad (c_i, \varepsilon) \leftarrow a_i + b_i + \varepsilon$
4: **врати** (c, ε)

Алгоритам 3.3.2 Одузимање са позајмљивањем

Улаз: Бројеви $a, b \in \{0, 1, 2, \cdots p-1\}$
Излаз: $(a - b \mod B^n, \varepsilon)$, где је ε *позајмица*

1: $\varepsilon \leftarrow 0$
2: **за** $i \in \{0, 1, 2, \cdots n-1\}$
3: $\quad (c_i, \varepsilon) \leftarrow a_i - b_i - \varepsilon$
4: **врати** (c, ε)

Како овим алгоритмима можемо да добијемо $\varepsilon \neq 0$ или $c \notin [0, p)$, морамо да коригујемо добијене резултате. Када будемо писали $(c, \varepsilon) \leftarrow x + y$ или $(c, \varepsilon) \leftarrow x - y$ користимо већ наведене алгоритме.

Алгоритам 3.3.3 Сабирање

Улаз: Бројеви $a, b \in \{0, 1, 2, \cdots p-1\}$
Излаз: $a + b \mod p$

1: $(c, \varepsilon) \leftarrow a + b$
2: **ако** $\varepsilon = 1$ **онда**
3: $\quad (c, 0) \leftarrow c - p$
4: **ако** $c \geq p$ **онда**
5: $\quad (c, 0) \leftarrow c - p$
6: **врати** c

Алгоритам 3.3.4 Одузимање

Улаз: Бројеви $a, b \in \{0, 1, 2, \cdots p-1\}$
Излаз: $a - b \mod p$

1: $(c, \varepsilon) \leftarrow a - b$
2: **ако** $\varepsilon = 1$ **онда**
3: $\quad (c, 0) \leftarrow c + p$
4: **врати** c

Лако се види да је временска сложеност и сабирања и одузимања $\Theta(n)$

3.3.3 Редукција по модулу

Ово је помоћни алгоритам који се користи у множењу. Множење ћемо, слично као и сабирање и одузимање, урадити у групи $(\mathbb{Z}, \cdot)$, па ћемо након тога извршити корекцију (редукцију) по модулу p. Приметимо да за два броја $a, b \in [0, p)$ имамо да $0 \leq ab \leq (p-1)^2$. То нам даје слободу да у следећем алгоритму претпоставимо да је њихов производ $s \leq (p-1)^2$. Приметимо да редукција по модулу није ништа друго но $s \mod p = s - p\lfloor \frac{s}{p} \rfloor$.
Овде је најскупља операција дељење, па је у Баретовој редукцији основна идеја да се $q = \lfloor \frac{s}{p} \rfloor$ оцени са $\hat{q}$ користећи јефтиније операције. Напоменимо да у следећем алгоритму још увек имамо дељење са B^x и редукцију по модулу B^x, међутим, то нам не представља никакав проблем с обзиром на то да узимамо $B = 2^w$, те се та дељења и редукције своде на битовно и (&), или битовни десни померај ($\gg$).

Алгоритам 3.3.5 Баретова редукција

Улаз: $s \in [0, (p-1)^2)$, p, $\mu = \lfloor \frac{B^{2n}}{p} \rfloor$
Излаз: $s \mod p$

1: $\hat{q} \leftarrow \lfloor \frac{\mu \lfloor \frac{a}{B^{k-1}} \rfloor}{B^{k-1}} \rfloor$
2: $x \leftarrow a \mod B^{k+1}$
3: $y \leftarrow p\hat{q} \mod B^{k+1}$
4: **ако** $x \geq y$ **онда**
5: $\quad r \leftarrow x - y$
6: **иначе**
7: $\quad r \leftarrow x + b^{k+1} - y$
8: **док** $r \geq p$
9: $\quad r \leftarrow r - p$
10: **врати** r

Може се доказати да важи $q - 2 \leq \hat{q} \leq q$ [7, стр. 36].

Овај алгоритам је још увек спорији од множења, али је асимптотски исти. Приметимо да иако ћемо после сваког множења позивати ову функцију, овде нам баш треба множење без редукције, тако да овде нема рекурзије.

Напомена 3.1. *Посматрајмо $p = 2^{512} - 1$. Ово је Мерсенов прост, па је редукција по његовом модулу знатно олакшана. Конкретно, нека је $(a_i)_{i=0}^{1041}$ број који желимо да редукујемо. Тада је*

$$a \equiv (a_i)_{i=0}^{520} + (a_i)_{i=521}^{1041} \mod p.$$

Дакле, за неке специјалне просте, редукција се може обавити знатно брже него коришћењем алгоритма 3.3.5. Скоро Мерсенови прости имају сличне особине. Такозвани NIST прости имају ово својство [8].

3.3.4 Дељење са константом мањом од B

У алгоритмима за множење ће нам требати да умемо да делимо са константама мањим од B. Када будемо писали $(z, \varepsilon) \leftarrow \frac{x}{y}$, мислимо на $(z, \varepsilon) \leftarrow (\lfloor x/y \rfloor, x \mod y)$.

Алгоритам 3.3.6 Дељење константним фактором мањим од B

Улаз: $a = (a_i)_{i=0}^{n-1} \in \mathbb{N}_0$, $b \in [0, B)$
Излаз: $(c, \varepsilon) = {}^{a}/_{b}$

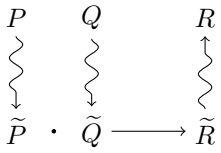
1: $\varepsilon \leftarrow 0$
2: **за** $i \in \{n-1, n-2, \cdots, 1, 0\}$
3: $\quad (c_i, \varepsilon) \leftarrow \frac{B\epsilon + a_i}{b}$
4: **врати** (c, ε)

3.3.5 Множење

Посматрајмо следећи алгоритам у зависности од природног параметра s.
Нека су дата два броја $p = p_{ks-1}p_{ks-2}\cdots p_1p_0$ и $q = q_{ks-1}q_{ks-2}\cdots q_1q_0$ (можемо додати водеће нуле, тако да и p и q буду дужине ks за неко k). Основна идеја је да поделимо p и q на s бројева дужине k које ћемо рекурзивно да измножимо.
Посматрајмо p и q у бази B^k. Нека је $P(x) = \sum_{i=0}^{k-1} P_i x^i$, и $Q(x) = \sum_{i=0}^{k-1} Q_i x^i$, и нека је $R(x) = \sum_{i=0}^{2k-2} R_i x^i$, где је $R(x) = P(x)Q(x)$.
Нека је $\widetilde{f} = \{(x, f(x) \mid x \in S\}$, за неко фиксирано S ($|S| = 2p+1$). Приметимо да ако је $f = \sum_{i=0}^{2p-2} f_i x^i$ полином, тада је

$$\begin{bmatrix} \widetilde{f}_0 \\ \widetilde{f}_0 \\ \vdots \\ \widetilde{f}_{2p-1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & s_0 & s_0^2 & \cdots & s_0^{2p-1} \\ 1 & s_1 & s_1^2 & \cdots & s_1^{2p-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & s_{2p-1} & s_{2p-1}^2 & \cdots & s_{2p-1}^{2p-1} \end{bmatrix}}_{\Lambda} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{2p-1} \end{bmatrix},$$

па је и $f = \Lambda^{-1}\widetilde{f}$ (Λ зависи само од S, па се Λ^{-1} може израчунати само једанпут).
Имамо $\widetilde{P} = \Lambda' P$ и $\widetilde{Q} = \Lambda' Q$, где Λ' представља првих k колона. Како се претходне операције могу свести на константан број множења константама и сабирања, ово може урадити у линеарној сложености по k (ако претпоставимо да је s фиксирано). Такође, сада можемо израчунати $\widetilde{R} = \widetilde{P} \cdot \widetilde{Q}$. Овде нам треба $2s-1$ множења бројева дужине k. Коначно $R = \Lambda^{-1}\widetilde{R}$.



Напомена 3.2. *Како је* $\det\Lambda$ *неки полином по* $s_0, s_1, \cdots s_{2p-1}$. *Како за* $s_i = s_j$ *видимо да је* $\det\Lambda = 0$, *следи су нуле тог полинома* $s_i - s_j$ *за* $j > i$. *Сада се лако види да је*

$$\det\Lambda = \prod_{i<j} s_j - s_i,$$

па је $\det\Lambda \neq 0$, *одаке следи да је она инвертибилна.*

Алгоритам 3.3.7 Toom-Cook

Улаз: $(a_i)_{i=0}^{k_a-1}, (b_i)_{i=0}^{k_b-1} \in [0, p-1)$, s
Излаз: ab

1: $k = \max\{\lceil {}^{k_a}/_{s} \rceil, \lceil {}^{k_b}/_{s} \rceil\}$
2: **за** $i \in \{0, 1, \cdots, k-1\}$
3: $\quad A_i = (a_p)_{p=is}^{(i+1)s-1}$
4: $\quad B_i = (b_p)_{p=is}^{(i+1)s-1}$
5: $\widetilde{A} \leftarrow \Lambda' A$
6: $\widetilde{B} \leftarrow \Lambda' B$
7: **за** $i \in \{0, 1, \cdots, k-1\}$
8: $\quad \widetilde{C_i} \leftarrow \widetilde{A_i}\widetilde{B_i}$
9: $C \leftarrow \Lambda^{-1}\widetilde{C}$
10: $C \leftarrow \mathrm{norm}(C)$
11: **врати** C

Овде функција norm врши нормализацију у смислу да ставља све коефициенте у интервал $[0, B)$ у временској сложености $\Theta(sk)$. Приметимо да рекурзија потиче од линије 8.

Пример 3.9.1 (Toom-2)**.** Toom-2 је еквивалентан са Карацубиним алгоритмом.
Основна идеја је да применимо *divide and conquer* метод, тј. да једно множење два броја дужине k сведемо на множења величине $\frac{k}{2}$. Приметимо да важи $(aB^k+b)(cB^k+d) = acB^{2k}+(ad+bc)B+bd = x_2B^{2k}+x_1B^k+x_0$, где је $(x_2, x_1, x_0) = (ac, (a+b)(c+d)-x_2-x_0, bd)$. Ако узмемо да $a, b, c, d < B^k$, Овим смо успешно свели проблем множења два k-тоцифрена броја на 3 множења $\frac{k}{2}$-тоцифрених броја.
Нека је $T(n)$ временска сложеност овог алгоритма за два броја дужине n. Тада важи $T(n) = 3T(\frac{n}{2}) + \Theta(n)$, па је по [1.1]

$$T(n) = \Theta(n^{\log_2 3}).$$

Пример 3.9.2 (Toom-3)**.** Често се под Toom-Cook алгоритмом подразумева Toom-3.
Узмимо $S = \{0, \pm 1, -2, \infty^6\}$. Приметимо да:

$$\begin{bmatrix} p(0) \\ p(1) \\ p(-1) \\ p(-2) \\ p(\infty) \end{bmatrix} = \begin{bmatrix} 1 & 0^1 & 0^2 \\ 1^0 & 1^1 & 1^2 \\ (-1)^0 & (-1)^1 & (-1)^2 \\ (-2)^0 & (-2)^1 & (-2)^2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_0 \\ n_1 \\ n_2 \end{bmatrix}.$$

Аналогно урадимо и за q. Сада помножимо $p(\cdot)$ и $q(\cdot)$. Ово захтева 5 s-цифрених множења. Сада важи:

$$\begin{bmatrix} r(0) \\ r(1) \\ r(-1) \\ r(-2) \\ r(\infty) \end{bmatrix} = \begin{bmatrix} p(0)q(0) \\ p(1)q(1) \\ p(-1)q(-1) \\ p(-2)q(-2) \\ p(\infty)q(\infty) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0^1 & 0^2 & 0^3 & 0^4 \\ 1^0 & 1^1 & 1^2 & 1^3 & 1^4 \\ (-1)^0 & (-1)^1 & (-1)^2 & (-1)^3 & (-1)^4 \\ (-2)^0 & (-2)^1 & (-2)^2 & (-2)^3 & (-2)^4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\Lambda} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix},$$

одакле следи да је,

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1/2 & 1/3 & -1 & 1/6 & -2 \\ -1 & 1/2 & 1/2 & 0 & -1 \\ -1/2 & 1/6 & 1/2 & -1/6 & 2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r(0) \\ r(1) \\ r(-1) \\ r(-2) \\ r(\infty) \end{bmatrix}.$$

Нека је $T(n)$ временска сложеност овог алгоритма за два броја дужине n. Тада важи $T(n) = 5T(\frac{n}{3}) + \Theta(n)$, па је по [1.1]

$$T(n) = \Theta(n^{\log_3 5}).$$

Следећи алгоритми су асимптотски бржи од Toom-3:

- Schönhage-Strassen [9] - $O(n \log n \log\log n)$
- Fürer [10] - $O(n \log n 2^{O(\log^* n)})$

Међутим, како у криптографији елиптичких кривх користимо релативно мале бројеве, ови алгоритми нам нису од интереса у овом раду.

Напомена 3.3. *Сличном анализом као у примерима 3.9.1 и 3.9.2 видимо да је за произвољно s сложеност алгоритма Toom-s једнака*

$$\Theta(n^{\log_s 2s-1}) \to \Theta(n) \qquad (x \to \infty).$$

Међутим, ова сложеност се односи на фиксирано s. Права сложеност би била

$$\Theta(s^2 n^{\log_s 2s-1}).$$

[6] $f(\infty) = \lim_{x\to\infty} \frac{f(x)}{x^{\deg f}}$

3.3.6 Мултипликативни инверз

До сада смо дали алгоритме за сабирање, множење и за адитивни инверз. Преостаје нам још да образложимо како да за дати број a и модул p израчунамо $a^{-1} \mod p$. Основна идеја алгоритма је да чувамо следеће инверианте

$$u = ax_1 + py_1,$$
$$v = ax_2 + py_2.$$

Ако би у неком тренутку добили $u = 1$, тада би $a^{-1} \cong x_1 \mod p$, и слично ако бисмо добили $v = 1$, тада би $a^{-1} \cong x_2 \mod p$ Уочимо једну такву шесторку бројева која ће нам представљати почетак алгоритма - $(u, v, x_1, y_1, x_2, y_2) = (a, p, 1, 0, 0, 1)$. Ако бисмо у сваком кораку могли да смањимо u и/или v, алгоритам би морао да се у једном тренутку заврши. У следећем излагању ћемо користити нотацију $[u, x_1, y_1]$ ако $u = ax_1 + py_1$ и $[u, x_1, y_1; v, x_2, y_2]$ ако $[u, x_1, y_1]$ и $[v, x_2, y_2]$. Ако будемо писали $\frac{x}{2}$, то значи да је x дељиво са 2.

$$[2u, 2x, y] \implies [u, x, y/2] \tag{1}$$
$$[2u, 2x + 1, y] \implies [u, x + \frac{p+1}{2}, \frac{y-a}{2}] \tag{2}$$
$$[u, x_1, y_1; v, x_2, y_2] \implies [u - v, x_1 - x_2, y_1 - y_2; v, x_2, y_2] \tag{3}$$

Алгоритам 3.3.8 Мултипликативни инверз

Улаз: $a \in [0, p-1)$, p
Излаз: $a^{-1} \mod p$

```
1:  (u, v) ← (a, p)
2:  (x1, x2) ← (1, 0)
3:  док u > 1 и v > 1
4:      док 2 | u
5:          u ← u/2
6:          ако 2 | x1 онда
7:              x1 ← x1/2                          ▷ Једначина (1)
8:          иначе
9:              x1 ← (x1+p)/2                      ▷ Једначина (2)
10:     док 2|v
11:         v ← v/2
12:         ако 2 | x2 онда
13:             x2 ← x2/2                          ▷ Једначина (1)
14:         иначе
15:             x2 ← (x2+p)/2                      ▷ Једначина (2)
16:     ако u ≥ v онда                             ▷ Једначина (3)
17:         u ← u − v
18:         x1 ← x1 − x2
19:     иначе                                      ▷ Једначина (3)
20:         v ← v − u
21:         x2 ← x2 − x1
22: ако u = 1 онда
23:     врати x1
24: иначе
25:     врати x2
```

Напомена 3.4. *Могли смо да чувамо и само једну инваријанту*

$$ax + py = u,$$

али тада бисмо добили Еуклидов алгоритам. Бинарна инверзија, иако има исту временску сложеност као Еуклидов алгоритам, има боље перформансе зато што се ослања на бинарне операције.

3.4 Екстензиона поља

Теорема 3.3 нам даје један начин да моделујемо екстензиона поља. Наиме, имаћмо један редукциони полином g, који постоји на основу леме 3.2, и све операције ћемо радити по његовом модулу. Како нам је битно да је неки систем што сигурнији, док је у исто време и што бржи, разматраћемо само бинарна поља. Екстензиона поља карактеристике $p > 2$ нам нису од интереса зато што нам не повећавају сигурност, али нам знатно успоравају систем.

До краја овог поглавља смо фиксирали поље $\mathbb{F}_{2^m}$, као и редукциони полином g.

3.4.1 Репрезентација

Слично као и елементе простих поља, и елементе бинарних поља памтимо као низ коефицијената полинома $(a_i)_{i=0}^{m-1}$.
Када будемо писали $(a_i)_{i=0}^{m-1}$, мислимо на полином $a(x) = \sum_{i=0}^{m-1} a_i x^i$

3.4.2 Сабирање и одузимање

Приметимо да за два полинома $(a_i)_{i=0}^{m-1}$ и $(b_i)_{i=0}^{m-1}$, $a + b$ се дефинише као $(a_i + b_i)_{i=0}^{m-1}$. Међутим, што је и једна од великих предности бинарних поља, $a_i + b_i = a_i \oplus b_i$ (где је $\oplus$ ексклузивно или). Дакле, за сабирање имамо следећи алгоритам:

Алгоритам 3.4.1 Сабирање

Улаз: $(a_i)_{i=0}^{m-1}, (b_i)_{i=0}^{m-1} \in \mathbb{F}_{2^m}$, m
Излаз: $c = a + b$

1: **за** $i \in \{0, 1, \cdots, m-1\}$
2: $\quad c_i \leftarrow a_i \oplus b_i$
3: **врати** c

Како што смо већ напоменули, у бинарним пољима важи да је за свако x, $-x = x$, па је алгоритам 3.4.1 уједно и алгоритам за одузимање.

3.5 Редукција

Како је редукциони полином степена $\deg g = m$, следи да се g може записати као

$$g(x) = x^m + r(x), \tag{4}$$

за неки полионом $r(x)$ степена мањег од m. Слично као и у простим пољима, можемо претпоставити да је тражени полином степена мањег од $2m - 1$, тј. треба редуковати полином

$$c(x) = \sum_{i=0}^{2m-2} c_i x^i.$$

Приметимо да је

$$r(x) \equiv -r(x) \equiv g - r(x) \equiv x^m \mod g.$$

Следи да

$$\begin{aligned} c(x) &\equiv \sum_{i=0}^{2m-2} c_i x^i \equiv \sum_{i=0}^{m-1} c_i x^i + x^m \sum_{i=0}^{m-1} c_{i+m} x^i \\ &\equiv \sum_{i=0}^{m-1} c_i x^i + r(x) \sum_{i=0}^{m-1} c_{i+m} x^i \mod g. \end{aligned}$$

Претпоставимо да смо већ израчунали $f_i(x) = x^i r(x) \mod g$ за свако $i \in \{0, 1, \cdots, m-1\}$ (ова операција не зависи од улаза, те се може урадити само јаданпут). Ово нам даје једноставан алгоритам за редукцију.

Алгоритам 3.5.1 Редукција у $\mathbb{F}_{2^m}$

Улаз: Полином $(c_i)_{i=0}^{2m-2}$, низ полинома $(f_i(x))_{i=0}^{m-1}$, m
Излаз: $c \mod g$

1: $C \leftarrow (c_i)_{i=0}^{m-1}$
2: **за** $i \in \{m, m+1, \cdots, 2m-2\}$
3: $\quad C \leftarrow C + f_{i-m}$
4: **врати** C

Иако на први поглед делује да алгоритам 3.5.1 ради у квадрадној сложености, сетимо се да из леме 3.2 следи да су потенцијални редукциони полиноми релативно густи. Ово нам даје слободу да изаберемо $r(x)$ тд. $r(x)$ има мали број коефициената различитих од 0. Нека је $r(x) = \sum_{i=0}^{D-1} x^{j_i}$. Тада је, користећи једначину (4),

$$z^{m+k} \equiv \sum_{i=0}^{D-1} x^{k+j_i} \mod g.$$

Овим смо показали да је сложеност алгоритма 3.5.1 ради у сложености $\Theta(mD)$, где у пракси можемо узети да је $D < 10$.

Напомена 3.5. *Напоменимо још да слично као код простих поља, постоје полиноми за брзу редукцију [8].*

3.6 Множење

Слично као и код простих поља, идеја је да дате полиноме $(a_i)_{i=0}^{m-1}$ и $(b_i)_{i=0}^{m-1}$ прво помножимо у $\mathbb{Z}_2[x]$, па онда редукцијом добијемо жељени резултат. За то можемо користити Карацубин алгоритам, који је описан у примеру 3.9.1. Како је алгоритам за полиноме скоро исти као и алгоритам за бројеве (зато што множење бројева редукујемо на множење полинома), тај алгоритам нећемо овде наводити.

3.7 Мултипликативни инверз

Овде поново имамо случај да је алгоритам који смо описали у простим пољима до одређене мере применљив и овде. Једина разлика је у томе што не проверавамо да ли је неки полином дељив са 2, неко са полиномом x, као и што не делимо са 2, већ са полиномом x.

Алгоритам 3.7.1 Мултипликативни инверз

Улаз: $a \in \mathbb{F}_{2^m}$, m, g
Излаз: $a^{-1} \mod g$

1: $(u, v) \leftarrow (a, g)$
2: $(p_1, p_2) \leftarrow (1, 0)$
3: **док** $u > 1$ **и** $v > 1$
4: **док** $x \mid u$
5: $u \leftarrow \frac{u}{x}$
6: **ако** $2 \mid p_1$ **онда**
7: $p_1 \leftarrow \frac{p_1}{x}$
8: **иначе**
9: $p_1 \leftarrow \frac{p_1+g}{x}$
10: **док** $2 \mid v$
11: $v \leftarrow \frac{v}{x}$
12: **ако** $2 \mid p_2$ **онда**
13: $p_2 \leftarrow \frac{p_2}{x}$
14: **иначе**
15: $p_2 \leftarrow \frac{p_2+g}{x}$
16: **ако** $\deg u \geq \deg v$ **онда**
17: $u \leftarrow u - v$
18: $p_1 \leftarrow p_1 + p_2$
19: **иначе**
20: $v \leftarrow v - u$
21: $p_2 \leftarrow p_2 + p_1$
22: **ако** $u = 1$ **онда**
23: **врати** p_1
24: **иначе**
25: **врати** p_2

Напомена 3.6. *Слично као и код простих поља, постоји Еуклидов алгоритам за полиноме који чува само једну инваријанту. Међутим, он користи сложеније операције попут дељења, док алгоритам 3.7.1 користи операције десног помераја ($\gg$) и одузимања.*

4 Елиптичке криве

У овом делу рада ћемо причати о елиптичким кривама. Видећемо да се над њеним тачкама може уочити Абелова група која је основа свих ЕС система.

4.1 Дефиниција

Дефиниција 4.1. *Елиптичка крива E над пољем $\mathbb{F}$ је дефинисана једначином:*

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

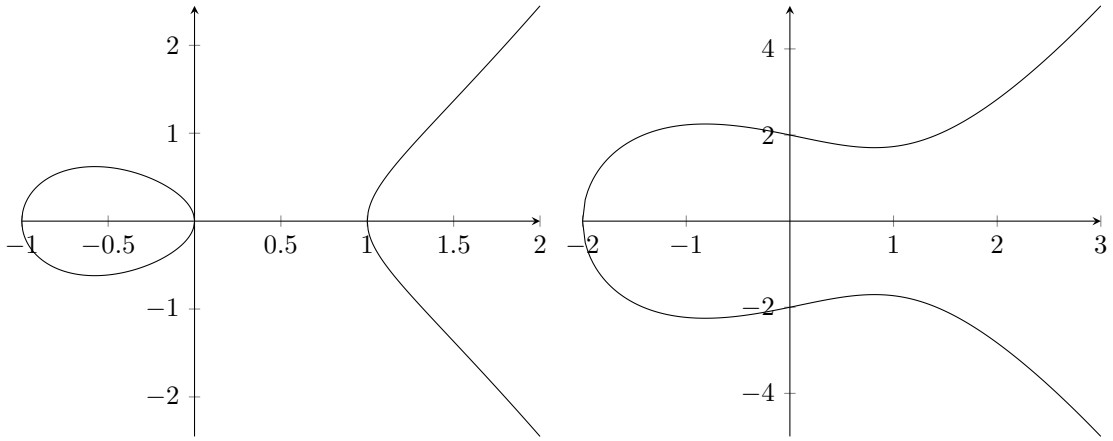
где $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$ и $\Delta \neq 0$, где је Δ дискриминанта E дефинисана као:

$$\begin{aligned} \Delta &= -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ d_2 &= a_1^2 + 4a_2 \\ d_4 &= 2a_4 + a_1a_3 \\ d_6 &= a_3^2 + 4a_6 \\ d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^3 - a_4^2. \end{aligned} \tag{2}$$

Скуп тачака на кривој E, у ознаци $E(\mathbb{F})$, је скуп

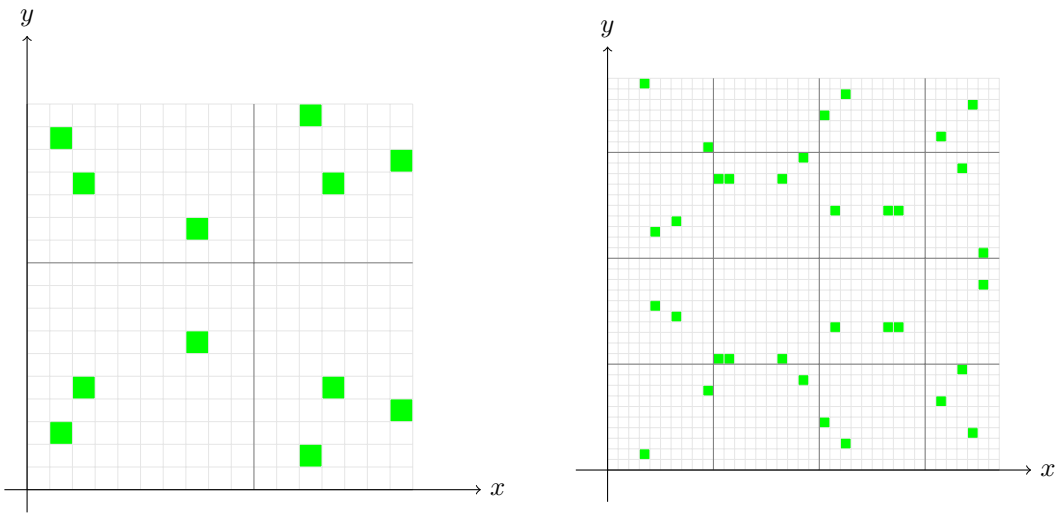
$$\{(x, y) \in \mathbb{F}^2 \mid y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\infty\}.$$

Тачку ∞ ћемо од сада обележавати са $\mathcal{O}$.



(а) $y^2 = x^3 - x$ (б) $y^2 = x^3 - 2x + 4$

Слика 9: Елиптичке криве над $\mathbb{R}$



(а) $y^2 = x^3 + 5x - 2$ на пољу $\mathbb{F}_{17}$ (б) $y^2 = x^3 + 2x + 5$ на пољу $\mathbb{F}_{37}$

Слика 10: Елиптичке криве над коначним пољима

Под доменским параметрима подразумевамо уређену петорку $(\mathbb{F}, E, P, n, h)$, где су:

- $\mathbb{F}$ - поље над којим је посматрана крива дефинисана; подразумевамо да $\mathbb{F}$ садржи и кореишћени редукциони полином,
- E - једначина криве; во може бити уређени пар (a, b) или уређена тројка (a, b, c) у зависности од потребе. E можемо послати и као *random seed* коришћен за генерисање насумичне криве,
- P - тачка на кривој; ова тачка служи као генератор у цикличној групи $\langle P \rangle$,
- n - ред тачке P,
- h - кофактор, $h = \frac{\#E(\mathbb{F})}{n}$

4.2 Класе изоморфних кривих

Дефиниција 4.2. *Нека су E_1 и E_2 две елиптичке криве дефинисане над K дате једначинама:*

$$E_1 : \quad y^2 + a_1xy + a_3 = x^3 + a_2x^2 + a_4x + a_6$$
$$E_2 : \quad y^2 + b_1xy + b_3 = x^3 + b_2x^2 + b_4x + b_6$$

Кажемо да су E_1 и E_2 изоморфне ако постоји четворка $u, r, s, t \in K$ $(u \neq 0)$ таква да трансформација координата

$$(x, y) \to (u^2x + r, u^3y + u^2sx + t) \tag{3}$$

трансформише једначину E_1 и једначину E_2. Трансформацију (3) ћемо звати прихватљивом променом координата.

Теорема 4.1 (Изоморфне класе)**.** *Нека је дата елиптичка крива $E : y^2 + a_1xy + a_3 = x^3 + a_2x^2 + a_4x + a_6$ над K.*

1. *Ако је $char(K) = 2, a_1 \neq 0$, тада је E изоморфна кривој дефинисаној једначином:*

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

   за неке $a, b \in K$. Кажемо да је таква крива не-суперсингуларна и има дискриминанту $\Delta = b$.

2. *Ако је $char(K) = 2, a_1 = 0$, тада је E изоморфна кривој дефинисаној једначином:*

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

   за неке $a, b, c \in K$. Кажемо да је таква крива суперсингуларна и има дискриминанту $\Delta = c^4$.

3. *Ако је* $char(K) = 3, a_1^2 \neq -a_2$, *тада је* E *изоморфна кривој дефинисаној једначином:*

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

за неке $a, b \in K$. *Кажемо да је таква крива не-суперсингуларна и има дискриминанту* $\Delta = -a^b$.

4. *Ако је* $char(K) = 3, a_1^2 = -a_2$, *тада је* E *изоморфна кривој дефинисаној једначином:*

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

за неке $a, b \in K$. *Кажемо да је таква крива суперсингуларна и има дискриминанту* $\Delta = -a^3$.

5. *Ако је* $char(K) \neq 2, 3$, *тада је* E *изоморфна кривој дефинисаној једначином:*

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

за неке $a, b \in K$. *Дискриминанта такве криве је* $\Delta = -16(4a^3 + 27b^2)$.

Доказ. Теорему доказујемо користећи следеће промене координата.

1. $\text{char}(K) = 2, a_1 \neq 0$:

$$(x, y) \to \left(a_1^2 x + \frac{a_3}{a_1}, a_1^3 y + \frac{a_1^2 a_4 + a_3^2}{a_1^3}\right)$$

2. $\text{char}(K) = 2, a_1 = 0$:

$$(x, y) \to (x + a_2, y)$$

3. $\text{char}(K) = 3, a_1^2 \neq -a_2$:

$$(x, y) \to \left(x + \frac{a_4 - a_1 a_3}{a_1^2 + a_2}, y + a_1 x + a_1 \frac{a_4 - a_1 a_3}{a_1^2 + a_2} + a_3\right)$$

4. $\text{char}(K) = 3, a_1^2 = -a_2$:

$$(x, y) \to (x, y + a_1 x + a_3)$$

5. $\text{char}(K) \neq 2, 3$:

$$(x, y) \to \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1 x}{216} - \frac{a_1^3 + 4a_1 a_2 - 12a_3}{24}\right)$$

□

4.3 Закон групе

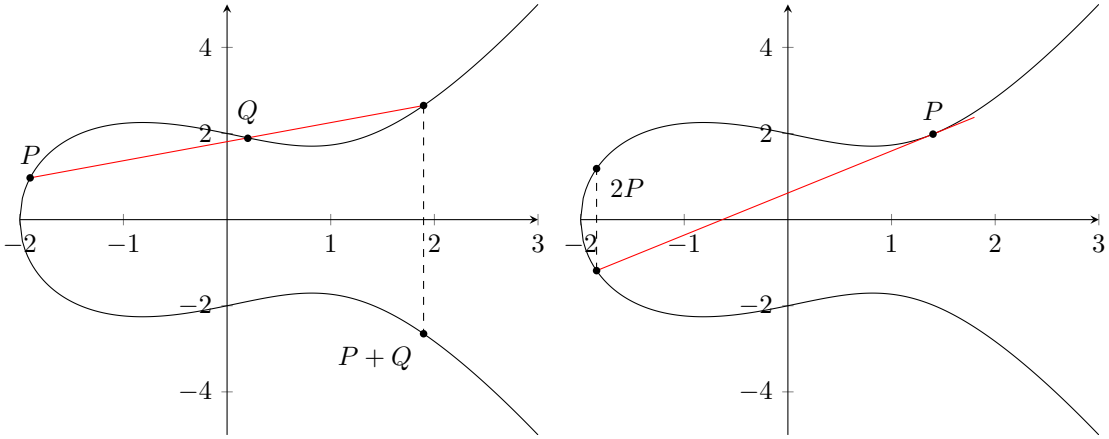
У овом делу рада нам је циљ да нађемо неку операцију (коју ћемо звати сабирање) над елиптичким кривама. Прво ћемо дати како би једна таква операција гласила у пољу реалних бројева, па ћемо тек онда објаснити како би она изгледала у коначним пољима.

4.3.1 Мотивација у $\mathbb{R}$

Посматрајмо елиптичку криву $E(\mathbb{R})$ дефинисану једначином $y^2 = x^3 + ax + b$. Тада је збир две тачке P и Q дефинисан на следећ начин:

- $\mathcal{O}$ је неутрал.
- $-(x, y) = (-x, y)$ - односно инверз неке тачке је њој осно симетрична тачка.
- Нека $P \neq \pm Q$ и нека је R трећи пресек праве PQ са кривом. Тада је $P + Q = -R$.
- Нека је R друга тачка пресека тангенте у тачки P и криве. Тада је $P + P = -R$. Ово је слично претходном случају.

Приметимо да се, за разлику од до сада посматраних група, сабирање две различите тачке знатно разликује од сабирања две исте.



(а) Пример сабирања (б) Пример дуплирања

Слика 11: $y^2 = x^3 - 2x + 4$

За $P = (x_1, y_1)$, $Q = (x_2, y_2$ и $P + Q = (x, y)$ се може показати да се последње две ставке могу записати као:

$$(x, y) = (x_1, y_1) + (x_2, y_2) = \left(\left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2, \frac{y_2 - y_1}{x_2 - x_1}(x_1 - x) + y_1\right)$$
$$(x, y) = (x_1, y_1) + (x_1, y_1) = \left(\left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1, \frac{3x_1^2 + a}{2y_1}(x_1 - x) - y_1\right)$$

4.3.2 Закон групе у коначним пољима

За криве над пољем чија карактеристика није ни 2 ни 3 ($y^2 = x^3 + ax + b$), закон групе је исти као и за криве над пољем $\mathbb{R}$:

- $\mathcal{O}$ је неутрал.
- $-(x, y) = (-x, y)$.
- Нека $P = (x_1, y_1) \neq \pm Q = (x_2, y_2)$. Тада је $P + Q = (x, y)$ дефинисано као:

$$(x, y) = (x_1, y_1) + (x_2, y_2) = \left(\left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2, \frac{y_2 - y_1}{x_2 - x_1}(x_1 - x) + y_1\right).$$

- За $P = (x_1, y_1)$, $P + Q = (x, y)$ је дефинисано као

$$(x, y) = (x_1, y_1) + (x_1, y_1) = \left(\left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1, \frac{3x_1^2 + a}{2y_1}(x_1 - x) - y_1\right).$$

За не-суперсингуларне криве над пољем карактеристике 2 ($y^2 + xy = x^3 + ax^2 + b$):

- $\mathcal{O}$ је неутрал.
- $-(x, y) = (x, x + y)$.
- Нека $P = (x_1, y_1) \neq \pm Q = (x_2, y_2)$. Тада је $P + Q = (x, y)$ дефинисано као:

$$(x, y) = (\lambda^2 + \lambda + x_1 + x_2 + a, x_1^2 + \lambda(x_1 + x) + x + y_1)$$

  где је $\lambda = \frac{y_1 + y_2}{x_1 + x_2}$.

- За $P = (x_1, y_1)$, $P + Q = (x, y)$ је дефинисано као

$$(x, y) = (\lambda^2 + \lambda + a, x_1^2 + \lambda x + x),$$

  где је $\lambda = x_1 + \frac{y_1}{x_1}$.

За суперсингуларне криве над пољем карактеристике 2 ($y^2 + cy = x^3 + ax + b$):

- $\mathcal{O}$ је неутрал.
- $-(x, y) = (x, y + c)$.
- Нека $P = (x_1, y_1) \neq \pm Q = (x_2, y_2)$. Тада је $P + Q = (x, y)$ дефинисано као:

$$(x, y) = \left(\left(\frac{y_1 + y_2}{x_1 + x_2} \right)^2 + x_1 + x_2, \frac{y_1 + y_2}{x_1 + x_2}(x_1 + x) + y_1 + c \right).$$

- За $P = (x_1, y_1) \neq -P$, $P + P = (x, y)$ је дефинисано као

$$(x, y) = \left(\left(\frac{x_1 + a}{c}^2 \right), \frac{x_1^2 + a}{c}(x_1 + x) + y_1 + c \right).$$

Лема 4.1. *За наведену операцију $+$ важи да је $(E(\mathbb{F}), +)$ Абелова група.*

Доказ ове леме изостављамо зато што се доказ своди на једноставну проверу алгебарских једнакости.

4.4 Својства

Теорема 4.2 (Структура групе)**.** *За дату елиптичку криву E над $\mathbb{F}_q$ важи:*

$$E(\mathbb{F}_q) \cong \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$$

за неке $n_2 \mid n_1$ и $n_2 \mid q - 1$.

Теорема 4.3 (Hasse)**.** *За дату елиптичку криву E над $\mathbb{F}_q$, број тачака на E, $\#E(\mathbb{F}_q)$ задовољава следећу неједнакост:*

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

Како $\sqrt{q} = o(q)$, имамо да $\#E(\mathbb{F}_q) \approx q$.

4.5 Сабирање тачака

Иако смо у §4.3 описали како се сабирају тачке, испоставља се да ако пређемо у пројективну раван, можемо да добијемо доста на ефикасности. Овај део ће бити последњи писан јер, искрено, има битнијих ствари.

4.6 Множење тачака

Дефиниција 4.3. *Адициони ланац за дати број A је низ $(a_i)_{i=0}^n$ који задовољава:*

- $a_0 = 1$,
- $a_n = A$,
- $(\forall k > 0)$ $a_k = a_i + a_j$ *за неке* $0 \leq i, j, < k$.

Генерализација проблема налажења адиционог ланца најмање дужине (сличан проблем, само за низ A) је доказана да је НП-комплетно [11]. Претпоставља се да је и сам проблем налажења оптималног ланца НП-комплетнан, али није доказано. Приметимо да бинарно степеновање није увек оптимално. Како је

$$a^{15} = a^3\big((a^3)^2\big)^2,$$

видимо да је један адициони ланац за $A = 15$ дужине 5 ($[1, 2, 3, 6, 12, 15]$), док нам бинарно степеновање даје ланац дужуне 6 ($[1, 2, 3, 6, 7, 14, 15]$).

Дефиниција 4.4. *Addition-subtracion ланац за дати број A је низ $(a_i)_{i=0}^n$ који задовољава:*

- $a_0 = 1$,
- $a_n = A$,
- $(\forall k > 0)$ $a_k = a_i \pm a_j$ *за неке* $0 \leq i, j < k$.

Није познато да ли је проблем налажења оптиманлног *addition-subtracion* ланца НП-тешко. Приметимо да претходна два проблема нису еквивалентна ($a^{31} = {}^{a^{32}}/_{a}$).
Како нам за дату тачку P рачунање $-P$ знатно брже од сабирања, видимо да су нам *addition-subtracion* ланци битнији у криптографији елиптичких кривих.

Сваки адициони ланац L је уједно и *addition-subtracion* ланац. Представљамо основни алгоритам за продалажење адиционог ланца

Алгоритам 4.6.1 Бинарно множење

Улаз: Доменски параметри $(\mathbb{F}, E, P, n, h)$, x
Излаз: xP

$S \leftarrow \mathcal{O}$
док $x > 0$
  ако $2 \mid x$ **онда**
    $S \leftarrow S + P$
  $P \leftarrow P + P$
  $x \leftarrow \lfloor \frac{x}{2} \rfloor$
врати S

Овај алгоритам захтева $\lceil \log_2 x \rceil + r \sim \frac{3}{2} \lceil \log_2 x \rceil$, где је r број цифара 1 у бинарном запису броја x. Овај алгоритам не користи чињеницу да ми можемо и да одузимамо тачке сличном брзином којом их и додајемо. Конкретно, увешћемо NAF сваког броја као низ цифара $(k_i)_{i=0}^{l-1}$, где $k_i \in \{0, \pm 1\}$, где нема узастопних не-нула цифара. Под претпоставком да имамо NAF неког броја x, можемо израчунати xP у сложености $l + r$, где је r број не-нула цифара у NAF-у.

Алгоритам 4.6.2 NAF множење

Улаз: Доменски параметри $(\mathbb{F}, E, P, n, h)$, x
Излаз: xP

$S \leftarrow \mathcal{O}$
$(k_i)_{i=0}^{l-1} \leftarrow \mathrm{NAF}(x)$
за $i \in \{l-1, l-2, \cdots, 0\}$
  $S \leftarrow S + S$
  ако $k_i = 1$ **онда**
    $S \leftarrow S + P$
  ако $k_i = -1$ **онда**
    $S \leftarrow S - P$
врати S

Остаје да видимо како да израчунамо NAF неког броја k. То можемо урадити применом трансформација облика:

$$[0, \underbrace{1, 1, \cdots, 1, 1}_{n}, 1] \to [1, \underbrace{0, 0, \cdots, 0, 0}_{n}, -1]$$

што се може елегантно урадити на следећи начин:

Алгоритам 4.6.3 *Non-adjacent form (NAF)*

Улаз: k
Излаз: $\mathrm{NAF}(k)$

$i \leftarrow 0$
док $k > 0$
  ако $2 \nmid k$ **онда**
    $k_i \leftarrow 2 - (k \mod 4)$
    $k \leftarrow k - k_i$
  иначе
    $k_i \leftarrow 0$
  $k \leftarrow \frac{k}{2}$
  $i \leftarrow i + 1$
врати $(k_s)_{s=0}^{i-1}$

Може се показати да је просечан број сабирања тачака коришћењем алгоритма 4.6.2 $\frac{4}{3} \lceil \log_2 x \rceil$.

5 Алгоритми

Након што смо објаснили шта значи сабирати тачке на кривој, можемо да објаснимо и како функционишу неки протоколи који их користе.

Као што смо наговестили у §2, потребна нам је нека функција која нас *враћа* из света тачака на кривој у свет бројева. Због тога за дати бинарни полином $(a_i)_{i=0}^{m-1}$ дефинишемо $\overline{a}$ као $a(2)$ (над $\mathbb{Z}$). У наредним алгоритмима претпостављамо да имамо функцију $\mathrm{int} : E(\mathbb{F})$ која је дефинисана као

$$\mathrm{int}(x,y) = \begin{cases} x & \text{, ако је } \mathbb{F} \text{ просто поље} \\ \overline{x} & \text{, ако је } \mathbb{F} \text{ бинарно поље} \end{cases}.$$

Како ће нам за неке алгоритме бити потребно да генеришемо псеудо-случајне бројеве, ради читљивост, процес бирања случајног броја x из скупа S ћемо обележавати са $x \in_R S$. Напоменимо да је генерисање доменских параметара врло често непотребно, и да се могу користити већ генерисане криве са унапред изабраним тачкама [8], због чега су алгоритми потребни за то изостављени у овом раду.

5.1 Elliptic Curve Discrete Logarithm Problem

Исто као што се RSA заснива на претпоставци да је *integer factorization problem* тежак, или DL на претпоставци да је *discrete logarithm problem* тежак, тако и EC системи претпостављају да је *elliptic curve discrete logarithm problem*, проблем решавања једначине $xP = Q$, тежак. Навешћемо неколико алгоритама који покушавају да што ефикасније реше ECDLP.

5.1.1 Pohlig-Hellman алгоритам

Pohlig-Hellman алгоритам редукује рачунање $\log_P Q$ на речунање ECDL у подгрупама простог реда групе $\langle P \rangle$.

Теорема 5.1 (Кинеска теорема о остацима)**.** *Нека је дат систем конгруенција $x \equiv x_i \mod m_i$, где за $i \neq j$ важи $(m_i, m_j) = 1$. Тада постоји јединствено $x \mod m$ које задовољава дати систем, где је $m = \prod_i m_i$.*

Нека је $\mathrm{ord}(P) = n = \prod_{i=1}^{k} {p_i}^{\alpha_i}$. Нека је $x = \log_P Q$. Основна идеја сведемо на рачунање ЕЦДЛ-а у простим подгрупама $\langle P \rangle$, тј. је да израчунамо $x_i = x \mod p_i^{\alpha_i}$, па да применом теореме 5.1 израчунамо x.

Приметимо да из услова да је $x = \log_P Q$ имамо:

$$Q - xP = \mathcal{O}. \tag{1}$$

Нека је $c = \frac{n}{p_i^{\alpha_i}}$. Приметимо да из (1) следи да

$$cQ - x(cP) = \mathcal{O}. \tag{2}$$

Међутим, како је $\mathrm{ord}(cP) = p_i^{\alpha_i}$, видимо да је једначина (2) еквивалентна са

$$cQ - x_i(cP) = \mathcal{O}.$$

Одавде видимо да је

$$x_i = \log_{cP} cQ.$$

Сада можемо без умањења општости претпоставити да је $\mathrm{ord}(P) = p^\alpha$. Нека је $x = \sum_{i=0}^{\alpha-1} x_i p^i$. Сада имамо да:

$$Q = \sum_{i=0}^{\alpha-1} x_i p^i P.$$

Множењем са $p^{\alpha-1-k}$ добијамо:

$$p^{\alpha-1-k}Q = \sum_{i=0}^{\alpha-1} x_i p^{\alpha-1+i-k} P = \sum_{i=0}^{k} x_i p^{\alpha-1+i-k} P.$$

Претпоставимо да смо израчунали $x_0, x_1, \cdots, x_{k-1}$. Сређивањем претходног израза видимо да:

$$x_k(p^{\alpha-1}P) = p^{\alpha-1-k}Q - \sum_{i=0}^{k-1} x_i p^{\alpha-1+i-k}P.$$

Како је $\mathrm{ord}(p^{\alpha-1}P) = p$, видимо да знањем $x_0, x_1, \cdots, x_{k-1}$ можемо израчунати x_k рачунањем ECDL у групи чији је ред једнак p.
Овим смо доказали да се рачунање ECDL може ефикасно свести на рачунање ECDL у простим подгрупама.

Пример 5.0.1. Овај пример је цопы-пастед из књиге, треба откуцати још брда ствари како бих могао да дам свој пример... [7, стр. 156].

До краја §5.1 ћемо сматрати да је n прост.

5.1.2 Полардов ρ алгоритам

Нека нам је дата коначна циклична група $(G, \cdot)$ реда n, где је n прост. желимо да за дате a и b одредимо x такво да $a^x = b$. Основна идеја је да нађемо два пара (α_1, β_1) и (α_2, β_2) таква да

$$a^{\alpha_1} b^{\beta_1} = a^{\alpha_2} b^{\beta_2}. \tag{3}$$

Претпоставимо да смо нашли једно решење једначине (3). Тада

$$a^{\alpha_2-\alpha_1} b^{\beta_2-\beta_1} = 1$$

видимо да мора да важи:

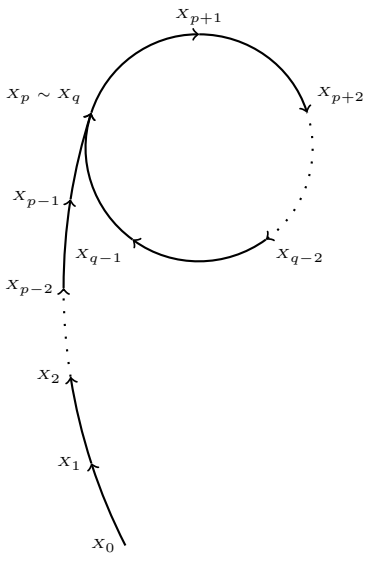
$$a^{\alpha_2-\alpha_1} a^{x(\beta_2-\beta_1)} = 1,$$

па и

$$\alpha_1 - \alpha_2 \equiv x(\beta_2 - \beta_1) \mod n.$$

Из претходне једначине следи да је

$$x = (\beta_2 - \beta_1)^{-1}(\alpha_1 - \alpha_2) \mod n.$$



Слика 12: Шемацки приказ Полардовог ρ алгоритма

Остаје да образложимо како налазимо (α_1, β_1) и (α_2, β_2). Нека је $x_i = (p_i, q_i)$. Кажемо да је $x_i \sim x_j$ ако $a^{p_i} b^{q_i} = a^{p_j} b^{q_j}$. Основна идеја је да конструишемо низ $(x_i)_{i=0}^{\infty}$ тако што итерирамо псеудо-насумичну функцију f (тј. $x_i = f^i(x_0)$). Како има коначно много (n) класа, постојаће индекси i и j тд. $x_i \sim x_j$. Нека су најмањи такви индекси p и q. Тада $(\forall i \geq 0)\ (x_{p+i} \sim x_{q+i})$, тј. овај низ је цикличан почевши од индекса p.
Дакле, редуковали смо проблем на проналажење циклуса у повезаној листи. Како p и q могу да буду незанемарљиве величине, потребан нам је алгоритам са меморискoм сложеношћу $O(1)$.
Можемо користити Флојдов алгоритам. Идеја је да имамо два показивача i и j и да у сваком тренутку чувамо инваријанту $j = 2i$. Након што $i, j > p$, имаћмо да се $i - j \mod p - q$ у сваком кораку смањује за 1, па ће у једном тренутку $x_i \sim x_j$.

Алгоритам 5.1.1 Полардов ρ алгоритам

Улаз: Доменски параметри $(\mathbb{F}, E, P, n, h)$, тачка Q
Излаз: $x = \log_P Q$

$(P_1, \alpha_1, \beta_1) \leftarrow (\mathcal{O}, 0, 0)$
$(P_2, \alpha_2, \beta_2) \leftarrow (P_1, \alpha_1, \beta_1)$
док тачно
    $(\alpha_1, \beta_1) \leftarrow f(\alpha_1, \beta_1)$
    $P_1 \leftarrow (\alpha_1 P + \beta_1 Q)$
    $(\alpha_2, \beta_2) \leftarrow f(f(\alpha_2, \beta_2))$
    $P_2 \leftarrow (\alpha_2 P + \beta_2 Q)$
    ако $P_1 = P_2$ **онда**
        $\alpha \leftarrow \alpha_1 - \alpha_2$
        $\beta \leftarrow \beta_2 - \beta_1$
        ако $\beta = 0$ **онда**
            врати 'Неуспех'
    врати $\alpha\beta^{-1} \mod n$

5.2 Генерисање кључева

Као што смо то објаснили у §2.3, потребан нам је пар приватног кључа и јавног кључа (x, Q), за које важи $xP = Q$. То можемо да урадимо тако што прво насумично изаберемо x, па тек онда израчунамо $Q = xP$.

Алгоритам 5.2.1 Генерисање приватног и јавног кључа

Улаз: Доменски параметри $(\mathbb{F}, E, P, n, h)$
Излаз: Приватни кључ d и јавни кључ Q

$d \in_R \{1, 2, \cdots, n-1\}$
$Q \leftarrow dP$
врати (d, Q)

5.3 Elliptic Curve Integrated Encryption Scheme

У овом алгоритму претпостављамо да имамо следеће функције:

- H - нека сигурна хеш функција (нпр. SHA-512),
- MAC (*message authentication code*) - нпр. HMAC,
- ENC - ENC_k представља симетричну енкрипцију кључем k,
- KDF (*key derivation function*) - функција која за дати улаз враћа неки кључ; ова функција је дериват функције H у смислу да потребан l-битни кључ, можемо добити конкатенацијом $H(X, 0)$, $H(X, 1)$, $\cdots$.

Идеја је да користимо елиптичке криве како бисмо сигурно пренели кључ за енкрипцију симетричним кључем (функције ENC и DEC) тако што ћемо користити функцију KDF. На то ћемо још додати и аутентикацију (функција MAC).

Алгоритам 5.3.1 ECIES енкрипција

Улаз: Доменски параметри $(\mathbb{F}, E, P, n, h)$, јавни кључ Q, отворени текст m
Излаз: Шифрат (R, C, t)

$k \in_R [1, n-1]$
$R \leftarrow kP$
$Z \leftarrow hkQ$
ако $Z = \infty$ **онда**
    иди на корак 1
$(k_1, k_2) \leftarrow \text{KDF}(\text{int}(Z), R)$
$C \leftarrow \text{ENC}_{k_1}(m)$
$t \leftarrow \text{MAC}_{k_2}(C)$
врати (R, C, t)

Алгоритам 5.3.2 ECIES декрипција

Улаз: Доменски парамтери $(\mathbb{F}, E, P, n, h)$, приватни кључ d, шифрат (R, C, t)
Излаз: Отворени текст m или одбијање шифрата

ако R није на кривој E **онда**
    врати 'Одбијено'
$Z \leftarrow hdR$
ако $Z = \infty$ **онда**
    врати 'Одбијено'
$(k_1, k_2) \leftarrow \mathrm{KDF}(\mathrm{int}(Z), R)$
$t' = \mathrm{MAC}_{k_2}(C)$
ако $t' \neq t$ **онда**
    врати 'Одбијено'
$m = \mathrm{DEC}_{k_1}(C)$
врати m

Доказ. Ако претпоставимо да је израчунато Z исто у алгоритмима 5.3.1 и 5.3.2, онда је (k_1, k_2) једнако у оба алгоритма, те под претпоставкама да су наведене функције (MAC, ENC и DEC) ваљање, онда је стварно израчунато m једнако оригиналном m. Нека је Z_e израчунато Z у енкрипцији, а Z_d у декрипцији. Тада важи

$$Z_d = hdR = hdkP = hkQ = Z_e.$$

□

5.4 Elliptic Curve Digital Signature Algorithm

У свакодневном животу, ми се потписујемо на документа како бисмо потврдили да је неки документ потекао од нас или да смо га прочитали. Криптографски аналог претходном је дигитални потпис (*digital signature*). Основна идеја је да на неки начин обрнемо процес енкрипције, тј. да свако има кључ за декрипцију, а само ми имамо кључ за енкрипцију. На овај начин можемо да потврдимо некоме наш идентитет тако што енкриптујемо неку поруку хешовану поруку $H(m)$, што једимо ми можемо.

Алгоритам 5.4.1 ECDSA генерисање потписа

Улаз: Доменски параметри $(\mathbb{F}, E, P, n, h)$, приватну кључ d, отворнени текст m
Излаз: Потпис (r, s)

$k \in_R [1, n-1]$
$\overline{x} \leftarrow \mathrm{int}(kP)$
$r \leftarrow \overline{x} \mod n$
ако $r = 0$ **онда**
    иди на корак 1
$e \leftarrow H(m)$
$s \leftarrow k^{-1}(e + dr) \mod n$
ако $s = 0$ **онда**
    иди на корак 1
врати (r, s)

Алгоритам 5.4.2 ECDSA провера потписа

Улаз: Доменски параметри $(\mathbb{F}, E, P, n, h)$, јавни кључ Q, отворени текст m, потпис (r, s)
Излаз: Примање или одбијање потписа

ако $r \notin [1, n-1]$ **или** $s \notin [1, n-1]$ **онда**
    врати 'Одбијено'
$e \leftarrow H(m)$
$w \leftarrow s^{-1} \mod n$
$(u_1, u_2) \leftarrow (ew \mod n, rw \mod n)$
$X \leftarrow u_1 P + u_2 Q$
ако $X = \infty$ **онда**
    врати 'Одбијено'
$\overline{x} \leftarrow \text{int}(X)$
$v \leftarrow \overline{x} \mod n$
ако $v = r$ **онда**
    врати 'Прихваћено'
иначе
    врати 'Одбијено'

Доказ. Ако важи да је $X = kP$, тада би важило $u = r$, што нам и треба. Како

$$\begin{aligned} X &= u_1 P + u_2 Q = (u_1 + u_2 d)P = (ew + wrd)P \\ &= w(e + rd)P = s^{-1}(e + rd)P = k(e + dr)^{-1}(e + dr)P \\ &= kP, \end{aligned}$$

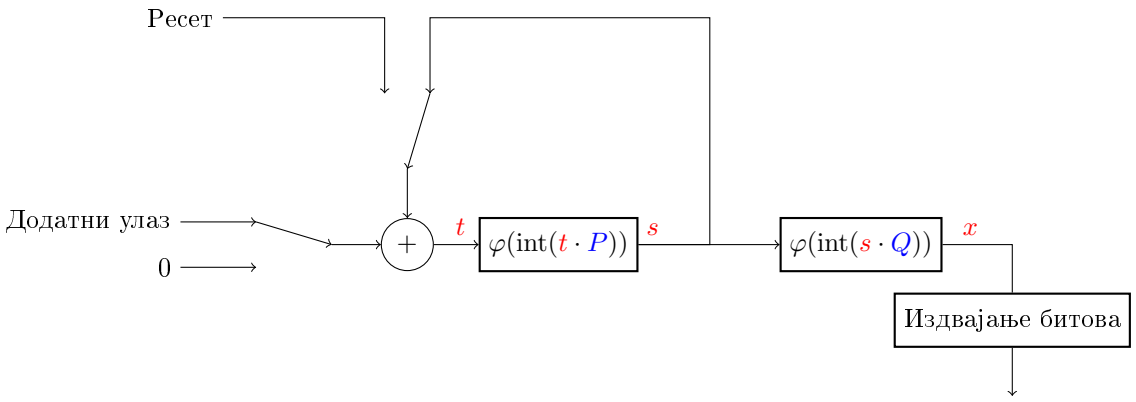
следи да је алгоритам коректан. □

5.5 Dual Elliptic Curve Deterministic Random Bit Generator

Најосновнији модел било ког псеудо-случајног генератора се састоји из:

- S - тренутно стање (углавном је S један број),
- g - функција излаза (у датом тренутку генератор ће нам вратити $g(S)$),
- f - функција стања (након што позовемо генератор, стање S се промени у $f(S)$).

Да би неки псеудо-случајни генератор био криптографски сигуран, неопходно је да нападач не може да установи следећи излаз на основу низа претходних. Зато се користе за f и g користе криптографски сигурне хеш функције. Нажалост, сигурност хеш функција обично долази из њихове компликованости, што резултује у томе да ми не знамо да ли неко зна како да пробије дату хеш функцију. Због тога је NIST објавио псеудо-случајни генератор чија се сигурност заснива на тежини ECDLP-а [12]. Идеја је да f буде функција која зависи од једне тачке P, док g зависи од друге тачке Q.



Слика 13: Шематски приказ Dual EC DRBG-а

Међутим, битно је да P и Q нису међусобно зависне, или барем, да нико не зна ту зависност. Да бисмо образложили зашто, узмимо поједностављени модел где је функција φ замењена са

функцијом идентитета. Прво приметимо да можемо да пробијемо први ниво одбране и да откријемо x (само проверимо све могуће вредности за x), па самим тим и $R = sQ$. Одавде не можемо да израчунамо s, па самим тим ни следеће стање (под претходним претпоставкама).
Претпоставимо сада да постоји зависност $dQ = P$, и да ми знамо d. Тада је следеће s једнако $s' = \text{int}(sP) = \text{int}(dsQ)$, па је и следећи излаз једнак

$$\text{int}(\text{int}(dR)Q).$$

Овим смо ми успешно предвидили наредни излаз. Иако је ово само поједностављен модел, сличан поступак се може применити и на стандардни модел [13].

Разлог зашто је овај алгоритам обавијен у контраверзу, је зато што је потребно да дата имплементација користи тачно одређене P и Q како би добила сертификат. Међутим, како је NSA предлагала избор за P и Q, а нису објавили како су дошли до њих, конзензус је да у овом алгоритму постоји *backdoor*.

Иако је овај псеудо-случајан генератор око 2000 пута спорији од генератора који се базирају на хеш функцијама, битан је зато што је то први генератор који је *доказиво* сигуран.

6 Закључак

Криптографија је једна од најбитнијих области информатике. Од мера предострожности да нико не мења овај рад у његовој изради, преко чувања података на нашим рачунарима, до војних комуникација, она нам даје потребну сигурност. У овом раду је описан само један њен део, који је, иако занемарљиве величине у односу на остатак криптографије, веома битан, поготову као систем који обједињује више подсистема.

На крају бих волео да се захвалим ментору овог рада и професору информатике, Филипу Марићу, на пруженој поцоћи приликом израде овог рада, као и на неизмерној подршци током целог средњошколског школовања.

Литература

[1] R. L. Rivest, A. Shamir, and L. Adelman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–216, feb 1978.

[2] D. Zagier. Newman's short proof of the prime number theorem. *The American Mathematical Monthly*, 204(8):705–708, oct 1997.

[3] A. K. Lenstra, H. W. Lenstra Jr., J. M. Pollard, and J. P. Buhler. *The Development of the Number Field Sieve*. Springer, 1993.

[4] R. D. Silverman. The multiple polynomial quadratic sieve. *Mathematics of Computation*, 48:329–339, 1987.

[5] Jr. H. W. Lenstra. Factoring integers with elliptic curves. *Annals of Mathematics*, 126(3):649–673, nov 1987.

[6] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, jul 1985.

[7] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, New York, 2003.

[8] National Institute of Standards and Technology. *FIPS 186-4: Digital Signature Standard*. July 2013.

[9] A. Schönhage and V. Strassen. Fast multiplication of large numbers. *Springer Computing*, 7(3-4):281–292, sep 1981.

[10] M. Fürer. Fast integer multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2007.

[11] P. Downey and R. Sethi B. Leong. Computing sequences with addition chains. *SIAM Journal on Cumputing*, 10(3):638–646, 1981.

[12] E. Braker and J. Jelsey. Recommendation for random number generation using deterministic random bit generators. Technical report, National Institute of Technologies, 2006.

[13] S Checkoway, M. Fredikson, R. Niederhagen, A. Everspaugh, M. Green, T. Lange, T. Ristenpart, D. J. Bernstein, J. Maskiewicz, and H. Shacham. On the practical exploitability of dual ec in tls implementations. In *In Proceedings of the 23rd USENIX security symposium*, 2014.