ასათიანი თ. ლომინაძე თ. ლომინაძე ნ. პაპიაშვილი რ.

კრიპტოგრაფიის საფუძვლები

2021

ზოგადი განხილვა

წინამდებარე ნაშრომში განხილულია კრიპტოგრაფიაში გამოყენებული მეთოდები, რომლებიც ინდივიდუალური თუ კომპლექსურად გამოყენებისას საშუალებას იძლევიან საკომუნიკაციო სისტემაში მონაცემთა გადაცემისას მიღწეულ იქნას უსაფრთხოების ისეთი მახასიათებლები, როგორიცაა:

- მონაცემთა კონფიდენციალობა (Data Secrecy);
- კომუნიკაციაში მონაწილე მხარეების აუთენტიფიკაცია;
- გზავნილის საიმედოდ გადაცემა (Data Integrity);
- შესრულებულ კომუნიკაციაზე მომავალში უარის თქმის შეუძლებლობა (Data non-Repudiation);

ამჟამად ინტერნეტში გავრცელებული სისტემებისგან ამორჩეულ და განხილულ იქნა ისეთი სისტემები, როგორიცაა:

- გაუმჯობესებული კრიპტოგრაფიული სისტემა AES (Advenced Encryption Standart);
- კრიპტოგრაფიული ჰეშ-ფუნქციები (Cryptographic Hash Functions)
- კრიპტოგრაფიული სისტემა RSA;
- ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფია (Ellyptic Curve Cryptography);

რამდენადაც ზემოთჩამოთვლილი სისტემების გაგება მოითხოვს სათანადო მათემატიკურ ცოდნას, ამიტომ საკმაოდ სრულადაა განხილული ისეთი საკითხები, როგორიცაა:

- მოდულური გამოთვლები (Modular Arithmetic);
- Xორარითმეტიკა (Exclusive or Arithmetic);
- გალუას ველის არითმეტიკა (Galois Field Arithmetic);
- ელიპტიკური ფუნქციები განმარტებული ნამდვილ რიცხვთა სიმრავლეზე და მათზე ოპერაციები;
- ელიპტიკური ფუქციები განმარტებული სასრულ სიმრავლეზე და მათზე ოპერაციები.

მიგვაჩნია, რომ ამ საკითხების ათვისება მკითხველს მისცემს სათანადო ცოდნას განვითარებადი კრიპტოგრაფიული მოდელების ეფექტურად ათვისებაში.

სარჩევი

ზოგადი განხილვა 1

თავი 1. მათემატიკური საფუძვლები 5

§1. რიცხვთა სიმრავლეები და მათზე განმარტებული ოპერაციები 5

1.1. ნატურალურ რიცხვთა სიმრავლე 5

1.2. მარტივი და შედგენილი რიცხვები 5

1.3. მთელ რიცხვთა სიმრავლე, Z 6

1.4. ნამდვილ რიცხვთა სიმრავლე R (Real) 6

§2. მონაცემთა წარმოდგენა ათვლის ორობით, რვაობით და თექვსმეტობით სისტემებში 7

§3. მოდულური არითმეტიკა 10

3.1. მოდულური ოპერატორი 11

3.2. მოდულური არითმეტიკული ოპერაციები 13

3.3. მოდულური კონგრუენტობა (Congruence Modulo) 15

§4. Xor არითმეტიკა (არითმეტიკა მოდულით 2) 16

4.1. შეკრება 16

4.2. გამოკლება 18

4.3. გამრავლება 19

4.4. გაყოფა 19

§5. გალუას ველის არითმეტიკა 21

5.1. გალუას ველი 22

5.2. გაფართოებული გალუას ველი $GF(2^m)$ 22

5.3. პოლინომური წარმოდგენა 23

5.4. პრიმიტიული (მარტივი) პოლინომი 23

5.5. შეკრება 24

5.6. გამრავლება 24

5.7. პრიმიტიული ელემენტი 25

თავი 2. კრიპტოგრაფიის მეთოდები 27

§1. მონაცემთა ჰეშირება 27
1.1. კრიპტოგრაფიული ჰეშ-ფუნქციები 27
1.2 კრიპტოგრაფიული ჰეშ-ფუნქციების თვისებები 28
1.3. ჰეშირების ალგორითმი 29
§2. სიმეტრიული ბლოკური შიფრაცია: AES 30
2.1. Rijndael-ის ზოგადი დახასიათება 31
2.2. გარდაქმნა ByteSub 33
2.3. გარდაქმნა shiftRow 34
2.4. გარდაქმნა MixColumn 35
2.5. გარდაქმნა AddRound Key (State, RoundKey) 35
2.6. შიფრაცია 35
2.7. დეშიფრაცია 37
§3. კრიპტოგრაფიული სისტემა ღია გასაღებით: RSA 38
3.1. ზოგადი განხილვა 38
3.2. RSA კრიპტოსისტემის გამოყენების მაგალითი 41
§4. ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფია 44
4.1. ელიპტიკური წირი ნამდვილ რიცხვთა სიმრავლეზე 44
4.1.1. ბინარული ოპერაცია – წერტილების შეკრების გეომეტრიული და ალგებრული მეთოდები 46
4.1.2. წერტილის სკალარზე გამრავლება 48
4.2. ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფია სასრულო ველის გამოყენებით 50
4.2.1. ალგებრული სტრუქტურა – მთელი რიცხვების ველი მოდულით P 50
4.2.2. მოდულური შეკრება 51
4.2.3. მოდულური ინვერსია შეკრების მიხედვით (Additive Inverse) 51
4.2.4. მოდულური გამოკლება 51
4.2.5. მოდულური გამრავლება 51
4.2.6. მოდულური ინვერსია გამრავლების მიხედვით 52

4.2.7. გაყოფა მოდულით P 53
4.2.8. ელიპტიკური წირი Fp ველში 53
4.2.9. წერტილების შეკრება 55
4.2.10. ელიპტიკური წირის ჯგუფის რიგი (Group Order) 57
4.2.11. სკალარული ნამრავლი და ციკლური ქვეჯგუფი 58
4.2.12. ბაზური წერტილის პოვნა 60
4.2.13. ელიპტიკურ წირებზე დაფუძნებული კრიპტოგრაფია 61
4.2.14. ასიმეტრიული კრიპტოგრაფია – საიდუმლო და ღია გასაღებები 62
4.2.15. შიფრაციის მაგალითები: ECDH, ECDSA 62
4.2.16. დიფი-ჰელმანის ალგორითმი ელიპტიკური წირისათვის, ECDH 63
4.2.17. ელიპტიკურ წირზე დაფუძნებული ციფრული ხელმოწერა 64
4.2.18. ხელმოწერა ECDSA ალგორითმის გამოყენებით 64
4.2.19. ხელმოწერის ვერიფიკაცია 65
ციტირებული ლიტერატურა 66

თავი 1. მათემატიკური საფუძვლები

§1. რიცხვთა სიმრავლეები და მათზე განმარტებული ოპერაციები

1.1. ნატურალურ რიცხვთა სიმრავლე

ნატურალურ რიცხვთა სიმრავლე აღინიშნება ასო N (Natural) და შეიცავს ყველა დადებით მთელ რიცხვს, დაწყებული 1-დან უსასრულობამდე,

$$N = \{1, 2, 3, ...\}$$

ნატურალურ რიცხვთა სიმრავლეზე განმარტებულია შეკრებისა და გამრავლების ოპერაციები, რადგან ორი მთელი დადებითი რიცხვის ჯამი და ნამრავლი ისევ მთელი დადებითი რიცხვია. რაც შეეხება გამოკლებისა და გაყოფის ოპერაციებს, ისინი არ არიან განმარტებული ნატურალურ რიცხვთა სიმრავლეზე, რადგან ორი ნატურალური რიცხვის სხვაობა შეიძლება იყოს უარყოფითი და ორი ნატურალური რიცხვის განაყოფი შეიძლება იყოს წილადი, რომლებიც არ მიეკუთვნებიან სიმრავლე N-ს.

ამავე დროს ნატურალური რიცხვები ხასიათდებიან მთელი რიგი თვისებებით, რომლებიც გადამწყვეტ როლს თამაშობენ კრიპტოგრაფიული სისტემების დამუშავებაში.

1.2. მარტივი და შედგენილი რიცხვები

მთელ დადებით რიცხვს ეწოდება მარტივი, თუ იგი უნაშთოდ იყოფა მხოლოდ 1-ზე და თავისთავზე. კრიპტოგრაფიასთან დაკავშირებულ ალგორითმების აღწერაში ხშირად შევხვდებით, მაგალითად, გამოთქმას – „ავიღოთ 256 ბიტიანი მარტივი რიცხვი“. ეს ძალიან დიდი რიცხვია და მის საპოვნად შეიძლება გამოვიყენოთ ერთ-ერთი შემდეგი ორი მიდგომიდან:

– პირველ შემთხვევაში ეს რიცხვი შეიძლება ავირჩიოთ მარტივი რიცხვების სიიდან, თუ ეს სია წინასწარ გვაქვს შედგენილი. ამგვარი სია შეიძლება შედგენილ იქნეს, მაგალითად, **ერატოსთენის საცერის (Eratosthenes sieve)** გამოყენებით. ეს ალგორითმი საშუალებას იძლევა მოცემული N-თვის შევადგინოთ ყველა მარტივი რიცხვების სია (2, 3, 5, 7, 11, 17, 19, 23, 29, 31, ..., N-მდე).

როცა N ძალიან დიდი რიცხვია, მარტივი რიცხვების სიის შენახვა არაეფექტურია და გამოიყენება სხვა ალგორითმი.

– მეორე შემთხვევაში შეიძლება ავიღოთ მოთხოვნილი სიდიდის რიცხვი დიაპაზონიდან (0, ... N) და შევამოწმოთ, არის თუ არა იგი მარტივი. მარტივობაზე შემოწმების (**Primality test**) რომელიმე ალგორითმით. უარყოფითი შედეგის შემთხვევაში ავიღოთ n-ის ახალი მნიშვნელობა და შევამოწმოთ იგი სიმარტივეზე. ეს პროცესი გავიმეოროთ მანამ, სანამ არ ვიპოვით მოთხოვნილი ზომის მარტივ რიცხვს.

ხშირად წარმოიქმნება შედგენილი რიცხვის, n-ის მარტივ მამრავლებად დაშლის ამოცანა (**Prime factorization**), რაზეც დაფუძნებულია RSA კრიპტოგრაფია. n-ის ზრდასთან ერთად რიცხვის მარტივ მამრავლებად დაშლის ამოცანის ამოხსნისათვის მოთხოვნილი გამოთვლების მოცულობა იმდენად სწრაფად იზრდება, რომ დიდი n-ის შემთხვევაში დღეისათვის ცნობილი მეთოდებით ამ ამოცანის ამოხსნის მცდელობა პრაქტიკულად აზრს კარგავს.

ელიპტიკურ წირზე დაფუძნებულ კრიპტოგრაფიაში (**Eliptic curve Cryptography, ECC**) საჭირო ხდება რიცხვი M-ის გამყოფების სიმრავლის პოვნის ამოცანა. მაგალითად, რიცხვი M=42 გამყოფების სიმრავლეა {2, 3, 6, 7, 14, 21}.

1.3. მთელ რიცხვთა სიმრავლე, Z

მთელ რიცხვთა სიმრავლე შედგება ყველა დადებითი და უარყოფითი რიცხვისა და რიცხვი 0-გან:

$$Z= \{... -3, -2, -1, 0, 1, 2, 3...\}$$

მთელ რიცხვთა სიმრავლეზე განმარტებულია შეკრების, გამოკლების და გამრავლების ოპერაციები, რადგან მთელი რიცხვების წყვილზე ამ ოპერაციების გამოყენება იძლევა მთელ რიცხვს. რაც შეეხება გაყოფის ოპერაციას, გაყოფის შედეგი შეიძლება აღმოჩნდეს წილადი, რაც არ ეკუთვნის Z-ს, ამიტომ მთელ რიცხვთა სიმრავლეზე გაყოფის ოპერაცია არაა განმარტებული.

1.4. ნამდვილ რიცხვთა სიმრავლე R (Real)

ნამდვილ რიცხვთა სიმრავლე შეიცავს ყველა მთელ და წილად რიცხვებს, აგრეთვე ტრანსცენდენტულ რიცხვებს $\pi, \sqrt{2}, \sqrt{5}$ და ა.შ. ნამდვილ რიცხვთა სიმრავლეზე განმარტებულია ოთხივე არითმეტიკული ოპერაცია, გარდა 0-ზე გაყოფისა, რადგან რიცხვის 0-ზე გაყოფის შედეგი არ წარმოადგენს რეალურ რიცხვს და მას აღნიშნავენ სიმბოლოთი ∞.

§2. მონაცემთა წარმოდგენა ათვლის ორობით, რვაობით და თექვსმეტობით სისტემებში

კომპიუტერში როგორც მონაცემები, ისე პროგრამები წარმოდგებიან ორობითი ფორმით და მონაცემთა დამუშავება ხდება ორობითი ოპერაციების გამოყენებით. ორობითი რიცხვის მაგალითია

b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
1	0	1	1	0	0	1	0	1	0	1	0

სადაც მარჯვენა b_0 არის წონით უმცირესი, ხოლო მარცხენა b_{11} წონით უდიდესი ბიტი. ამ ორობითი რიცხვის რვაობით სისტემაში წარმოსადგენად, რომელშიაც გამოიყენება რვა ციფრი 0, 1, 2, 3, 4, 5, 6, 7, ვსარგებლობთ შემდეგი შესაბამისობის ცხრილით

რვაობითი		ორობითი	რვაობითი		ორობითი
0	–	000	4	–	100
1	–	001	5	–	101
2	–	010	6	–	110
3	–	011	7	–	111

იმისათვის, რომ ორობითი რიცხვი ჩავწეროთ რვაობით სისტემაში, ორობითი რიცხვის ციფრები, დაწყებული მარჯვნიდან, დავყოთ სამეულებად (ტრიადებად). თუ ორობითი რიცხვის ციფრთა რაოდენობა სამის ჯერადი არ არის, მას მარჯვნიდან მივუწეროთ ერთი ან ორი 0, რადგან თუ მთელ რიცხვს მარცხნივ მივუწერთ ნულებს, ამით მისი სიდიდე არ შეიცვლება. ამგვარად, ზემოთ მოყვანილი ორობითი რიცხვისათვის გვექნება

O 5 4 5 2,

სადაც ასო O (octal, რვაობითი) მიუთითებს, რომ მოცემულია რიცხვი რვაობითი ფორმით. თუ გვინდა რვაობითი რიცხვის გადაყვანა ორობითში, საჭიროა რვაობითი ციფრები შევცვალოთ შესაბამისი ტრიადებით.

აღწერილის ანალოგიურად, თუ გვინდა ორობითი რიცხვი ჩავწეროთ თექვსმეტობით სისტემაში, ვსარგებლობთ თექვსმეტობითი

0 1 2 3 4 5 6 7 8 9 a b c d e f

ციფრების ორობით ციფრებთან შესაბამისობის ცხრილით

თექვსმეტობითი	ორობითი	თექვსმეტობითი	ორობითი
0	0000	8	1000
1	0001	9	1001
2	0010	a	1010
3	0011	b	1011
4	0100	c	1100
5	0101	d	1101
6	0110	e	1110
7	0111	f	1111

დავყოფთ მოცემული ორობითი რიცხვის ციფრებს, დაწყებული მარჯვნიდან, ოთხეულებად (ტეტრადებად) და თითოეულ ტეტრადას შევუსაბამებთ თექვსმეტობით ციფრს. ზემოთ მოყვანილი ორობითი რიცხვისათვის გვექნება

Ox b2a

აქ პრეფიქსი Ox მიუთითებს, რომ რიცხვი მოცემულია თექვსმეტობითი ფორმით.

როგორც ზემოთ განხილული მაგალითები გვიჩვენებს, რიცხვი რვაობით ან თექვსმეტობით სისტემებში გაცილებით კომპაქტურად ჩაიწერება, ვიდრე ორობითში, რაც ფართოდ გამოიყენება დიდი ზომის რიცხვების გამოსახატავად. მაგალითად, კრიპტოგრაფიაში ფართოდ გამოყენებული 256 ბიტიანი ორობითი რიცხვი თექვსმეტობითში გამოიხატება 256 : 4 = 64 თექვსმეტობითი სიმბოლოთი, რაც გაცილებით კომპაქტური წარმოდგენაა.

ზემოთ მოყვანილი მაგალითები ეხება მცირე ზომის მონაცემებს. კრიპტოგრაფიაში ხმარებული მონაცემების განზომილებაზე წარმოდგენას გვაძლევს ერთ-ერთი მონაცემი, რომელიც გამოიყენება სისტემა Bitcoin-ში ელექტრონული ხელისმოწერის პროცედურაში:

Ox = 8x79be667ef9dcbbac55a06295ce8770b07029bfcdb2dc28d959f2815b16f81798

ეს მონაცემი, მიუხედავად იმისა, რომ თექვსმეტობით სისტემაშია წარმოდგენილი, ძნელად აღსაქმელია, თუმცა გაცილებით კომპაქტურია, ვიდრე 256-ბიტიანი ორობითი წარმოდგენა.

ხშირად გზავნილი წარმოდგენილია ტექსტური ფორმით და საჭირო ხდება მისი წარმოდგენა სხვადასხვა ფორმით, საბოლოოდ კი ორობითი ფორმით, რადგან კრიპტოგრაფიული ალგორითმების კომპიუტერით განხორციელებისას ოპერაციები ძირითადად წარმოებს ორობით მთელ რიცხვებზე.

ტექსტური მონაცემების კოდირების ერთ-ერთი გავრცელებული შესაძლებლობაა ASCII (American Standard Code for Information Interchange) კოდი, რომელიც დამუშავებული იქნა ინგლისურენოვანი ტექსტებისთვის და მოიცავს 128 სიმბოლოს, რომელთა ორობით კოდში წარმოდგენისათვის გამოიყენება შვიდბიტიანი ორობითი რიცხვები დიაპაზონში [0, 2^7-1].

ქვემოთ მოყვანილია ამ კოდის ფრაგმენტი ინგლისური მაღალი რეგისტრის ასოებისათვის a, b, c, ... x, y, z.

DEC	OCT	HEX	BTN	Simbol
97	141	61	01100001	a
98	142	62	01100010	b
99	143	63	01100011	c
100	144	64	01100100	d
101	145	65	01100101	e
102	146	66	01100110	f
103	147	67	01100111	g
104	150	68	01101000	h
105	151	69	01101001	i
106	152	6A	01101011	j
107	153	6B	01101100	k
108	154	6C	01101100	l
109	155	6D	01101101	m

DEC	OCT	HEX	BTN	Simbol
110	156	6E	01101110	n
111	157	6F	01101111	o
112	160	70	01110000	p
113	161	71	01110001	q
114	162	72	01110010	r
115	163	73	01110011	s
116	164	74	01110100	t
117	165	75	01110101	u
118	166	76	01110110	v
119	167	77	01110111	w
120	170	78	01111000	x
121	171	79	01111001	y
122	172	7A	01111010	z

აქ DEC (DECIMAL) ნიშნავს ათობითს, OCT (OCTal) - რვაობითს, HEX (HEXADECIMAL) - თექვსმეტობითს, BIN (BINARY) - ორობითს, ხოლო Simbol ასოს გამოსახულებაა.

თუმცა ASCII კოდირებაში სიმბოლოები 7 ბიტით არის წარმოდგენილი, მაგრამ, რადგან ტექსტური მონაცემების წარმოდგენისათვის ხშირად გამოიყენება რვაბიტიანი ბაიტები, ამიტომ 7 ბიტის ბაიტში მოსათავსებლად ხდება 8 ბიტიანი ბაიტის მარცხენა ბიტის 0-ით შევსება. ამგვარად, მაგალითად a სიმბოლოს ASCII კოდი 1100001 წარმოდგება როგორც 01100001. ამ მიზეზით, როგორც ცხრილებიდან ჩანს, ყველა ჩამოთვლილი სიმბოლო წარმოდგენილია ბაიტით, რომლის მარცხენა ბიტი ნულის ტოლია. განვიხილოთ

მაგალითი, წარმოვადგინოთ ტექსტი „Apple“ ორობითი გამოსახულებით. ამისათვის თითოეული ასო წარმოვადგინოთ ორობითი ექვივალენტით და მოვახდინოთ მათი კონკატენაცია. გვექნება

01100001	01110000	01110000	01101100	01100101
A	P	P	L	E

აქამდე განვიხილავდით რიცხვებს ორობით, რვაობით და თექვსმეტობით ათვლის სისტემებში. 256 ბიტიანი ორობითი რიცხვი, როგორც ცნობილია ორობითი არითმეტიკიდან ღებულობს მნიშვნელობებს დიაპაზონში 0-დან 2^{256}-1-მდე. მოვახდინოთ 2^{256}-ის მიახლოებითი შეფასება ათვლის ათობითი სისტემისთვის.

$$2^{256} = \underbrace{2^{10} \cdot 2^{10} \ldots 2^{10} \cdot 2^{6}}_{\text{25-ჯერ}} > \underbrace{10^{3} \cdot 10^{3} \ldots 10^{3}}_{\text{25-ჯერ}} \cdot 64 = 64 \cdot 10^{75}$$

ამ გამოსახულებაში $2^{10} = 1024$ შევცვალეთ $10^3 = 1000$-ით. როგორც ვხედავთ $64 \cdot 2^{75}$ ხომ დიდი ათობითი რიცხვია, მაგრამ 2^{256} მასზე გაცილებით მეტია.

როგორც ზემოთ ითქვა, ორობითი რიცხვი რვაობითში ან თექვსმეტობითში რომ წარმოდგეს, ხშირად საჭირო ხდება მარცხენა მხარეს რამდენიმე ნულოვანი ბიტის დამატება, რომ ბიტების საერთო რაოდენობა იყოფოდეს 3-ზე ან 4-ზე. ეს მოვლენა ცნობილია მონაცემთა შევსების (padding) სახელწოდებით. ანალოგიური საჭიროება წარმოიქმნება კრიპტოგრაფიაში, როცა ტექსტური მონაცემის ორობითი წარმოდგენა იყოფა სტანდარტული ზომის ბლოკებად, მაგალითად 128 ბიტიან, ანუ 16 ბაიტიან ბლოკებად. ამ დროს მონაცემის ბლოკების შექმნა იწყება მარცხნიდან მარჯვნივ და თუ ბოლო ბლოკი არ არის სტანდარტული ზომის, ხდება მისი შევსება (padding), რომელიმე სპეციალურად შერჩეული მონაცემით.

§3. მოდულური არითმეტიკა

მოდულური არითმეტიკა ფართო გამოყენებას პოულობს მათემატიკის სხვადასხვა სფეროებში, მათ შორის კრიპტოგრაფიაში. ქვემოთ განხილული იქნება მოდულური არითმეტიკის ძირითადი კანონები. რაც შეეხება ამ სფეროს უფრო სრულ განხილვას, იგი შეიძლება მოძიებულ იქნეს ინტერნეტის სხვადასხვა წყაროებში, როგორიცაა, მაგალითად, საიტი Khanacademy.org, ან ინფორმაციაში, რომელსაც აბრუნებს ინტერნეტი შემდეგი საკვანძო სიტყვებით ძიებისას: Khan Academy Modular Arithmetic.

3.1. მოდულური ოპერატორი

მოდულური ოპერატორი აღინიშნება სიტყვით modulus, შემოკლებით mod. (ზოგ შემთხვევაში, მაგალითად ზოგიერთ პროგრამირების ენაში და კალკულატორში, გამოიყენება სიმბოლო %).

ზოგადად შეიძლება ითქვას, რომ მოდულური ოპერატორი დაკავშირებულია გაყოფის ოპერაციასთან. როგორც ცნობილია ორი დადებითი მთელი A და B რიცხვების გაყოფისას ადგილი აქვს შემდეგს: $\frac{A}{B}$ = Q ნაშთი R,

სადაც A გასაყოფია (Dividend), B გამყოფია (Divisor), Q განაყოფია (Quotient), R-კი ნაშთია (Remainder).

ხშირად გაყოფის ოპერაციისას საინტერესოა არა განაყოფი Q, არამედ ნაშთი R, რაც გამოისახება მოდულური ოპერაციით

A modB = R

მაგალითად, $\frac{5}{3}$ = 1 და ნაშთი 2, რაც ჩაიწერება როგორც 5 mod3 = 2.

გაყოფის ოპერაციის თვისების გათვალისწინებით, ცხადია რომ A modB ანუ R ღებულობს მნიშვნელობებს სიმრავლიდან {0, 1, 2, … B-1}. განვიხილოთ მაგალითი, გამოვთვალოთ მნიშვნელობები mod3 რიცხვების მიმდევრობისათვის 0, 1, 2, 3, ...

$\frac{0}{3}$ = 0 ნაშთი 0 (0 mod3 = 0)

$\frac{1}{3}$ = 0 ნაშთი 1 (1 mod3 = 1)

$\frac{2}{3}$ = 0 ნაშთი 2 (2 mod3 = 2)

$\frac{3}{3}$ = 1 ნაშთი 0 (3 mod3 = 0)

$\frac{4}{3}$ = 1 ნაშთი 1 (4 mod3 = 1)

$\frac{5}{3}$ = 1 ნაშთი 2 (5 mod3 = 2)

$\frac{6}{3}$ = 2 ნაშთი 0 (6 mod3 = 0)

$\frac{7}{3}$ = 2 ნაშთი 1 (7 mod3 = 1)

$\frac{8}{3}$ = 2 ნაშთი 2 (8 mod3 = 2)

როგორც ზემოთ მოყვანილი მაგალითი გვიჩვენებს, რიცხვების მოდულური მნიშვნელობა მეორდება ციკლურად. ეს თვისება საშუალებას იძლევა ჩამოყალიბდეს მოდულური მნიშვნელობის ვიზუალურად წარმოდგენის მეთოდი.

ავიღოთ წრე და დავყოთ იგი იმდენ ნაწილად, რა მოდულითაც ხდება გამოთვლები, ქვემო ნახაზზე გამოსახულია ორი ნახაზი mod3 და mod7 შემთხვევებისთვის.

mod3-ის შემთხვევაში, იმისათვის, რომ ვიპოვოთ 5mod3, დავიწყოთ 0-ით აღნიშნული წერტილიდან და მისგან საათის ისრის მიმართულებით გავაკეთოთ 5 ნაბიჯი წრიულად და ბოლო წერტილი მოგვცემს 5mod3 მნიშვნელობას, ამ შემთხვევაში 2-ს. ანალოგიურად, 7mod3 შემთხვევაში ვაკეთებთ 7 ნაბიჯს, რაც გვაძლევს 1-ს: 6mod7 შემთხვევაში გვექნება 6mod7=6, ხოლო 12mod7 მოგვცემს 5-ს. რაც შეეხება უარყოფითი რიცხვის მოდულურ მნიშვნელობას, საკმარისია გამოვიყენოთ იგივე წრე, ოღონდ ნაბიჯები უნდა გავაკეთოთ საათის ისრის საწინააღმდეგო მიმართულებით. მაგალითად -2mod3 გვაძლევს 1-ს, ანუ -2mod3=1. ანალოგიურად, -5mod7 შემთხვევაში, დაწყებული 0-დან, ვაკეთებთ 5 ნაბიჯს საათის ისრის საწინააღმდეგოდ, რაც გვაძლევს -5mod7=2.

ზემოთ მოყვანილი მაგალითები მიუთითებენ შემდეგი ზოგადი წესის გამოყენების შესაძლებლობაზე:

– იმისათვის, რომ ვიპოვოთ დადებითი A მთელი რიცხვის მნიშვნელობა მოდულით B საჭიროა A-ს გამოვაკლოთ B იმდენჯერ, სანამ სხვაობა არ აღმოჩნდება [0, (B-1)] დიაპაზონში. მაგალითად 17mod5-თვის გვექნება:

17mod5 = 12mod5 = 7mod5 = 2mod5 = 2

– რომ ვიპოვოთ უარყოფითი A რიცხვის მნიშვნელობა modB, საჭიროა A-ს დავუმატოთ B იმდენჯერ, სანამ ჯამი არ მოხვდება [0, B-1] დიაპაზონში. მაგალითად, -17mod5 შემთხვევაში გვექნება:

-17mod5 = -12mod5 = -7mod5 = -2mod5 = 3mod5 = 3

3.2. მოდულური არითმეტიკული ოპერაციები

ზემოაღწერილი მოდულური ოპერატორის თვისებები საშუალებას იძლევა განისაზღვროს ძირითადი არითმეტიკული ოპერაციები მოდულური გამოთვლების შემთხვევაში, რომლებიც აქ მოცემულია მათი სისწორის დამტკიცების გარეშე. (დამტკიცება შეიძლება ინახოს საიტზე Khan Academy.com. Modular Arithmetic).

მოდულური შეკრება

მოდულურ არითმეტიკაში შეკრების ოპერაცია გამოისახება ტოლობით:

$$(A+B) \text{ mod} C = (A\text{mod}C + B\text{mod}C) \text{ mod} C$$

მაგალითად, თუ A = 13, B=16 და C=5, მაშინ

$(13+16) \text{ mod}5 = (13\text{mod}5 + 16\text{mod}5) \text{ mod}5 = (3+1) \text{ mod}5 = 4 \text{ mod}5 = 4$

მოდულური გამოკლება

გამოკლება მოდულურ არითმეტიკაში ხორციელდება შემდეგნაირად:

$$(A-B) \text{ mod} C = (A\text{mod}C - B\text{mod}C) \text{ mod} C$$

მაგალითად, თუ A= 13, B=16 და C=5, გვექნება

$$(13-16) \text{ mod}5 = (13\text{mod}5 - 16\text{mod}5) \text{ mod}5 = (3-1) \text{ mod}5 = 2 \text{ mod}5 = 2$$

მოდულური გამრავლება

მოდულური გამრავლება ხორციელდება ფორმულით:

$$(A \cdot B) \text{ mod} C = (A\text{mod}C \cdot B\text{mod}C) \text{ mod} C$$

მაგალითად, A = 13, B=16 და C=5 შემთხვევაში გვექნება:

$$(13 \cdot 16) \text{ mod}5 = (13\text{mod}5 \cdot 16\text{mod}5) \text{ mod}5 = (3 \cdot 1) \text{ mod}5 = 3 \text{ mod}5 = 3$$

მოდულური გაყოფა

ჩვეულებრივ არითმეტიკაში მოცემული A რიცხვის ინვერსიული (შებრუნებული) რიცხვი აღინიშნება როგორც A^{-1}. ეს არის ისეთი რიცხვი, რომელიც აკმაყოფილებს ტოლობას $A \cdot A^{-1} = 1$. მაგალითად, 5-ის ინვერსიული რიცხვია $\frac{1}{5}$ $(5 \cdot \frac{1}{5} = 1)$.

რაც შეეხება მოდულურ გამოთვლებს, აქ როგორც ზემოთ ითქვა, გამოსახულება AmodC მნიშვნელობა ეკუთვნის სიმრავლეს {0, 1, 2, ... , C-1}, რადგან იგი წარმოადგენს ნაშთს, რომელიც მიიღება, A რიცხვის C-ზე გაყოფისას. რაც შეეხება მოდულურ გამოთვლებში

A რიცხვის ინვერსიულ რიცხვს, ისიც აღინიშნება როგორც A^{-1} და არის ერთ-ერთი {0, 1, 2, . . . , C-1} სიმრავლიდან, რომელიც აკმაყოფილებს ტოლობას $(A \cdot A^{-1})$ modC= 1.

განვიხილოთ მაგალითები, რომლებიც გვიჩვენებენ, რომ მოდულური გაყოფის შესაძლებლობა დაკავშირებულია ინვერსიული ელემენტის არსებობა-არარსებობასთან.

მაგალითი 1. ვთქვათ A=5 და C=6. ვიპოვოთ A^{-1}, ანუ რიცხვი {0, 1, 2, ... , 5} სიმრავლიდან, რომლისთვისაც გვექნება $A \cdot A^{-1}$ mod6=1. ამისათვის განვიხილოთ ყველა შესაძლო ვარიანტი გადასინჯვის მეთოდით, ანუ მიმდევრობით განვიხილოთ რიცხვები 0, 1, 2, 3, 4, 5. გვექნება

$(5 \cdot 0) \bmod 6 = 0$

$(5 \cdot 1) \bmod 6 = 5$

$(5 \cdot 2) \bmod 6 = 4$

$(5 \cdot 3) \bmod 6 = 3$

$(5 \cdot 4) \bmod 6 = 2$

$(5 \cdot 5) \bmod 6 = 1$ ← ინვერსიული ელემენტი ნაპოვნია!

ამგვარად, რიცხვ 5-ის ინვერსია mod6 არის 5, რადგან $(5 \cdot 5) \bmod 6 = 1$.

მაგალითი 2. ვთქვათ A=3 და C=6. წინა მაგალითის მსგავსად შევამოწმოთ ყველა შესაძლო ვარიანტი. გვექნება

$(3 \cdot 0) \bmod 6 = 0$

$(3 \cdot 1) \bmod 6 = 3$

$(3 \cdot 2) \bmod 6 = 0$

$(3 \cdot 3) \bmod 6 = 3$

$(3 \cdot 4) \bmod 6 = 0$

$(3 \cdot 5) \bmod 6 = 3$

რამდენადაც {0, 1, 2, 3, 4, 5} სიმრავლის არცერთი ელემენტი არ იძლევა ერთის ტოლ მნიშვნელობას, ამიტომ c = 6 შემთხვევაში რიცხვი 3-თვის ინვერსიული ელემენტი არ არსებობს, რაც ნიშნავს, რომ მოდულური გაყოფა შეუძლებელია.

ზემოთ მოყვანილი მაგალითები ადასტურებენ შემდეგ ზოგად თვისებას: AmodC გამოსახულებისთვის რიცხვ A-ს გააჩნია ინვერსიული (შებრუნებული) ელემენტი მხოლოდ იმ შემთხვევაში, თუ A და C რიცხვებს არ გააჩნიათ საერთო მარტივი

თანამამრავლი, ანუ არ არსებობს ისეთი მარტივი რიცხვი, რომელზედაც როგორც A ისე C იყოფა უნაშთოდ.

პირველი მაგალითის შემთხვევაში რიცხვებს 5 და 6 არ გააჩნდა საერთო მარტივი თანამამრავლი, ხოლო მეორე მაგალითის შემთხვევაში რიცხვებს A=3 და C=6 გააჩნიათ საერთო მარტივი თანამამრავლი 3.

რამდენადაც მარტივ რიცხვებს მარტივი გამყოფები არ გააჩნია, ამიტომ მოდულურ გამოთვლებში, როგორც წესი, მოდულის მნიშვნელობად აიღება მარტივი რიცხვი, რაც უზრუნველყოფს რიცხვებისთვის ინვერსიული რიცხვების არსებობას.

ზემოთ მოყვანილი ალგორითმი, რაც მდგომარეობს ინვერსიული ელემენტის ძიებისას ყველა შესაძლო ვარიანტის გადასინჯვაში, მისაღებია მოდულის მცირე მნიშვნელობისთვის. როცა მოდული დიდი რიცხვით გამოისახება, ეს მიდგომა არაეფექტური ხდება გამოთვლების მოცულობის თვალსაზრისით და გამოიყენება სხვა, უფრო ეფექტური ალგორითმები. ერთ-ერთ ასეთ მეთოდს წარმოადგენს **გაფართოებული ევკლიდეს ალგორითმი (Extended Euclidean algorithm)**, რომელიც სხვა მონაცემებთან ერთად საშუალებას იძლევა განისაზღვროს რიცხვის ინვერსიის მნიშვნელობა.

3.3. მოდულური კონგრუენტობა (Congruence Modulo)

ხშირად მათემატიკურ ტექსტებში გვხვდება გამოსახულება

$$A \equiv B \pmod{C}$$

სადაც ≡ კონგრუენტობის, შესაბამისობის, ტოლობის სიმბოლოა, ხოლო (modC) ნიშნავს, რომ მოდულური გამოთვლები ხორციელდება ტოლობის ორივე მხარისთვის.

მაგალითად $27 \equiv 17 \pmod 5$ რადგან

$$27 \bmod 5 = 17 \bmod 5$$

$$2 \bmod 5 = 2 \bmod 5$$

$$2 = 2$$

რაც შეეხება 26 და 17, ისინი არ არიან კონგრუენტული mod 5-ით, რადგან

$26 \bmod 5 = 1$ და $17 \bmod 5 = 2$ და ამგვარად $1 \neq 2$.

§4. Xor არითმეტიკა (არითმეტიკა მოდულით 2)

ეს არითმეტიკული სისტემა განსხვავდება კომპიუტერში გამოყენებული ორობითი არითმეტიკისაგან და იგი გამოიყენება მოწყობილობებს შორის მონაცემთა გადაცემისას წარმოქმნილი შეცდომების აღმოჩენის (და არა გასწორების) ალგორითმებში. ტექნიკური ხასიათის ლიტერატურაში იგი გამოიყენება X_{or} არითმეტიკის დასახელებით, რაც უკავშირდება X_{or} ლოგიკურ ფუნქციას (Exclusive Or, გამორიცხული ან), ხოლო კოდირების თეორიის მათემატიკურ კვლევებთან დაკავშირებულ ლიტერატურაში მისთვის გამოიყენება დასახელება „არითმეტიკა მოდულით 2“.

როგორც ცნობილია, შორ მანძილებზე გადაცემისას მონაცემი გადაიცემა როგორც ბიტების მიმდევრობა, თანაც, როგორც წესი, მონაცემში ბიტების რაოდენობა დიდ დიაპაზონში შეიძლება იცვლებოდეს.

ზოგადად, n ბიტიანი მიმდევრობა შეიძლება ჩაიწეროს როგორც $b_{n-1}, b_{n-2}, \ldots, b_1, b_0$, სადაც b_0 უმცირესი, ხოლო b_{n-1} უდიდესი წონის ბიტია. როგორც წესი, გადაცემა იწყება დიდი ბიტიდან: ჯერ გადაიგზავნება ბიტი b_{n-1}, შემდეგ ბიტი b_{n-2}, და ა.შ. ბოლოს ბიტი b_0. თუმცა, იშვიათად გვხვდება მოწყობილობები, რომლებშიაც ბიტების გადაცემა ხდება ზემოთ თქმულის საწინააღმდეგო მიმდევრობით.

როგორც წესი, გადასაცემი მონაცემი შეიძლება შეიცავდეს ბიტების ნებისმიერ რაოდენობას, დაწყებული რამდენიმე ბიტიდან, გაგრძელებული კილობიტებით და მეგაბიტებით.

X_{or} არითმეტიკაში განიხილება ოთხი ძირითადი არითმეტიკული ოპერაცია, რომლებისთვისაც გამოყენებული იქნება შემდეგი სიმბოლოები: $\oplus$ შეკრებისთვის, $\ominus$ გამოკლებისთვის, $\otimes$ გამრავლებისთვის და $\oslash$ გაყოფისთვის. განვიხილოთ ეს ოპერაციები ცალ-ცალკე.

4.1. შეკრება

თუ გადასაცემ მონაცემებს წარმოვიდგენთ როგორც რიცხვებს, მაშინ ორი რიცხვის შეკრებისას ოპერაცია ყველა თანრიგში სრულდება სხვა თანრიგებისაგან დამოუკიდებლად, ანუ, პარალელურად და თვითოეულ თანრიგში ჯამი გამოითვლება შემდეგი ცხრილის მიხედვით:

⊕	0	1
0	0	1
1	1	0

როგოც ვხედვთ, ერთბიტიანი რიცხვების შეკრება ხდება ორის მოდულით, ანუ X_{or} ლოგიკური ცხრილის შესაბამისად. მართლაც, მაგალითად

$(1+1)\mathrm{mod}2=2\mathrm{mod}2=0$

განვიხილოთ შეკრების მაგალითი:

$$\begin{array}{ccccccccccc} & 1 & 0 & 1 & 0 & 1 & & & 1 & 0 & 1 & 0 & 1 \\ \oplus & & & & & & \rightarrow & \oplus & & & & & \\ & & & 1 & 1 & 0 & & & 0 & 0 & 1 & 1 & 0 \end{array} = 1\ 0\ 0\ 1\ 1$$

აქ თანრიგთა რაოდენობის გათანაბრების მიზნით მეორე შესაკრებს წინ მიეწერება ორი ნული, რაც რიცხვის მშივნელობას არ ცვლის.

ჩვეულებრივ, ორობით შეკრებასთან შედარებით X_{or} შეკრებას რამდენიმე „უცნაური“ თვისება გააჩნია. მაგალითად, რიცხვის თავისთავთან შეკრებისას მიიღება ნულოვანი შედეგი:

$$\begin{array}{ccccccccc} & & 1 & 0 & 1 & 0 & 1 & & \\ \oplus & & & & & & & \rightarrow & 0\ 0\ 0\ 0\ 0 \\ & & 1 & 0 & 1 & 0 & 1 & & \end{array}$$

რაც არ ეთანხმება ჩვეულებრივ წარმოდგენას შეკრების ოპერაციაზე.

შევადაროთ ორობითი და X_{or} შეკრების ოპერაციები სწრაფქმედების თვალსაზრისით. ჩვეულებრივ არითმეტიკაში ორი რიცხვის შეკრება იწყება დაბალი თანრიგიდან ერთიანის ან ნულის გადატანით მომდევნო (მაღალ) თანრიგში, როგორც ეს მაგალითზეა ნაჩვენები:

$$\begin{array}{ccccccccccccc} & 1 & & 0 & & 1 & & 1 & & 1 & & 1 & \\ & \leftarrow & & \leftarrow & & \leftarrow & & \leftarrow & & \leftarrow & & \leftarrow & \\ & & 1 & & 0 & & 1 & & 1 & & 0 & & 1 \\ \oplus & & & & & & & & & & & & \\ & & 1 & & 0 & & 1 & & 0 & & 1 & & 1 \end{array} = 1\ 0\ 1\ 1\ 0\ 0\ 0$$

აქ თანრიგებს შორის კავშირია და, n თანრიგა რიცხვების შემთხვევაში, გადატანა ხდება n-ჯერ მიმდევრობით. თუ ერთ თანრიგში შეკრების დროს აღვნიშნავთ T, მაშინ

შეკრების ოპერაციისთვის საჭირო დრო იქნება nT. X_{or} შეკრების შემთხვევაში იმავე რიცხვებისთვის გვექნება:

$$\begin{array}{cccccccc} & 1 & & 1 & & & 1 & \\ & \uparrow & & \uparrow & & & \uparrow & \\ & 1 & 0 & 1 & 1 & 0 & 1 & \\ \oplus & & & & & & & \rightarrow \quad 0\ 0\ 0\ 1\ 1\ 0 \\ & 1 & 0 & 1 & 0 & 1 & 1 & \end{array}$$

ეს ნიშნავს, რომ ოპერაცია ⊕ თანრიგებში მიმდინარეობს პარალელურად და მთლიანად ოპერაციის შესრულების დრო უდრის T. ამგვარად, ოპერაცია + სჭირდება nT-ჯერ მეტი დრო, ვიდრე ოპერაცია ⊕-ს, რაც ამ უკანასკნელის სწრაფქმედებაზე მიუთითებს.

4.2. გამოკლება

ისევე, როგორც X_{or} შეკრების შემთხვევაში, გამოკლების ოპერაცია თანრიგებში ხდება დამოუკიდებლად და პარალელურად შემდეგი ცხრილის შესაბამისად:

⊖	0	1
0	0	1
1	1	0

აქაც, ისევე, როგორც X_{or} შეკრების შემთხვევაში, გამოთვლები წარმოებს მოდულით 2. მაგალითად, 0-1 → (-1) mod 2=1.

შეკრებისა და გამოკლების ცხრილები ერთმანეთს ემთხვევა და გამოდის, რომ ეს ოპერაციებიც ერთმანეთს ემთხვევა. ამგვარად, X_{or} არითმეტიკაში გამოკლების ოპერაცია ყოველთვის შეიძლება შეიცვალოს შეკრებით და გამოკლების შესრულების საჭიროება არ არსებობს. მაგალითად,

$$\begin{array}{cccccc} & 1 & 0 & 1 & 0 & 1 & 1 \\ \ominus & & & & & & \\ & 0 & 1 & 1 & 1 & 0 & 1 \\ \hline & 1 & 1 & 0 & 1 & 1 & 0 \end{array} \quad \rightarrow \quad \begin{array}{cccccc} & 1 & 0 & 1 & 0 & 1 & 1 \\ \oplus & & & & & & \\ & 0 & 1 & 1 & 1 & 0 & 1 \\ \hline & 1 & 1 & 0 & 1 & 1 & 0 \end{array}$$

ამგვარად, X_{or} არითმეტიკაში ორი რიცხვის შეკრება და გამოკლება ერთიდაიმავე შედეგს იძლევა.

4.3. გამრავლება

გამრავლების ბიტური ოპერაცია $\otimes$ X_{or} არითმეტიკაში განისაზღვრება ცხრილით:

$\otimes$	0	1
0	0	0
1	0	1

პროცედურულად X_{or} გამრავლების ოპერაცია ემთხვევა ორობითი გამრავლების ოპერაციას, ოღონდ სვეტებში ბიტების შეკრება ხდება მოდულით 2.

მაგალითად 101101 და 1011 რიცხვების გამრავლებისთვის გვექნება:

			1	0	1	1	0	1
		$\otimes$						
					1	0	1	1
			1	0	1	1	0	1
		1	0	1	1	0	1	
	0	0	0	0	0	0		
1	0	1	1	0	1			
1	0	0	0	1	1	1	1	1

მარჯვნიდან მეოთხე სვეტის ჯამი, მაგალითად, გამოითვლება შემდეგნაირად:

$$(1+1+0+1) \bmod 2 = 3 \bmod 2 = 1$$

4.4. გაყოფა

მონაცემთა გადაცემისას წარმოქმნილი შეცდომების აღმოჩენისა და გასწორების ალგორითმებში X_{or} გაყოფა ძირითადი ოპერაციაა, რადგან სწორედ ორი რიცხვის გაყოფისას მიღებული ნაშთი წარმოადგენს მაკონტროლებელ მონაცემებს.

პროცედურულად X_{or} გაყოფის ოპერაცია მსგავსია მთელი რიცხვების გაყოფისა ორობით არითმეტიკაში, თუმცა არსებობს მნიშვნელოვანი განსხვავება შუალედურ გამოთვლებში. შედარებისთვის განვიხილოთ გაყოფის მაგალითი ჯერ ორობით, შემდეგ კი X_{or} არითმეტიკაში.

ჯერ შევასრულოთ 110011 რიცხვის გაყოფა რიცხვზე 101.

```
გასაყოფი   1 1 0 0 1 1 | 1 0 1     გამყოფი
         -              | 1 0 1 0   განაყოფი
           1 0 1
           ------
             1 1 0
             0 0 0
             -----
               1 0 1
               1 0 1
               -----
                 0 0 1
                 0 0 0
                 -----
                     1     ნაშთი
```

გამოთვლების სისწორე მოწმდება ტოლობით:

განაყოფი X გამყოფი + ნაშთი = გასაყოფი

მართლაც,

```
1)      1 0 1 0   (განაყოფი)
    X
          1 0 1   (გამყოფი)
        ---------
        1 0 1 0
    0 0 0 0
1 0 1 0
---------------
1 1 0 0 1 0
```

```
2)      1 1 0 0 1 0
    +
          0 0 1   ნაშთი
    -------------
    1 1 0 0 1 1   გასაყოფი
```

რაც შეეხება X_{or} გაყოფას, იგი ორობითი გაყოფისგან განსხვავდება შემდეგი თვისებებით:

1. X_{or} არითმეტიკაში გამოკლება შეკრების ექვივალენტურია, ამიტომ თითოეულ ბიჯზე გამოკლების მაგივრად სრულდება შეკრება;
2. თითოეულ ბიჯზე შეკრება ხდება მოდულით 2;
3. ორობით არითმეტიკაში თითოეულ ბიჯზე განაყოფის სათანადო პოზიციაში ფიქსირდება 1, თუ საკლები მეტია მაკლებზე და 0 წინააღმდეგ შემთხვევაში.

X_{or} არითმეტიკაში, რადგან გამოკლება იცვლება შეკრებით და, ამგვარად, უარყოფითი რიცხვი არ არსებობს, გაყოფის ოპერაციაში გამოიყენება რიცხვთა შედარების სრულიად განსხვავებული წესი. ამბობენ, რომ ერთი რიცხვი თავსდება მეორეში, თუ მათი უმაღლესი წონის ერთიანი ერთსადაიმავე პოზიციაში აქვთ. მაგალითად, რიცხვი 1011 თავსდება რიცხვში 1001 და პირიქით, 1001 თავსდება 1011-ში. თუ რიცხვებს უმაღლესი წონის ერთიანი განსხვავებულ პოზიციებში აქვთ, მაშინ ერთ-ერთი თავსდება მეორეში, მაგრამ პირიქით არა. მაგალითად, 0101 თავსდება 1011-ში, მაგრამ 1011 არ თავსდება

0101-ში. X_{or} გაყოფის ოპერაციაში თუ შუალედური ნაშთი მოიცავს გამყოფს, მაშინ განაყოფის სათანადო პოზოციაში ჩაიწერება 1 და წინააღმდეგ შემთხვევაში 0.
განვიხილოთ X_{or} გაყოფის ოპერაცია 110001011 რიცხვის 1011-ზე გაყოფის მაგალითზე:

გასაყოფი	1	1	0	0	0	1	0	1	1	1	0	1	1			გამყოფი
⊕										1	1	1	0	0	0	განაყოფი
	1	0	1	1												
		1	1	1	0											
		1	0	1	1											
			1	0	1	1										
			1	0	1	1										
				0	0	0	0									
				0	0	0	0									
					0	0	0	1								
					0	0	0	0								
						0	0	1	1							
						0	0	0	0							
							0	1	1	ნაშთი						

შევამოწმოთ X_{or} გაყოფის სისწორე ზოგადი წესით:

განაყოფი ⊗ გამყოფი ⊕ ნაშთი =გასაყოფი

მართლაც,

1)			1	1	1	0	0	0	განაყოფი	2)		1	1	0	0	0	1	0	0	0	
	⊗										⊕										
					1	0	1	1	გამყოფი									0	1	1	ნაშთი
			1	1	1	0	0	0				1	1	0	0	0	1	0	1	1	გასაყოფი
		1	1	1	0	0	0														
	0	0	0	0	0	0															
1	1	1	0	0	0																
1	1	0	0	0	1	0	0	0													

§5. გალუას ველის არითმეტიკა

გალუას ველის არითმეტიკა ფართო გამოყენებას პოულობს მონაცემთა გადაცემისას წარმოქმნილი შეცდომების გამასწორებელი კოდის (Error Correction Code, ECC, შეცდომების გამასწორებელი კოდი) დამუშვებაში, როცა ხდება არა მხოლოდ შეცდომების აღმოჩნა, არამედ განუსაზღვრელობის პირობებში მისი ავტომატურად გასწორება. ქვემოთ მოყვანილია გალუას ველის არითმეტიკის ზოგიერთი თვისება, ხოლო შემდეგ თავში განხილულია შეცდომების აღმოჩენისა და გასწორების რიდ-სოლომონის (Reed-Solomon, R-S) ალგორითმი, რომელიც ამ არითმეტიკას ეყრდნობა.

5.1. გალუას ველი

გალუას ველი (Galois Field, GF), სხვანაირად ველი სასრული ელმენტებით, წარმოადგენს ელემენტების სიმრავლეს, რომლისთვისაც:

– განსაზღვრულია შეკრებისა და გამრავლების ოპერაციები (ეს ოპერაციები არ ემთხვევა ჩვეულებრივ არითმეტიკაში განსაზღვრულ ოპერაციებს);

– GF-ში შემავალ ელემენტებზე შესრულებული ოპერაციების შედეგი ეკუთვნის ისევ GF ველს;

– სრულდება კომუტატორების, დისტრიბუტულობის და ასოციატურობის თვისებები;

– ნებისმიერი a ელემენტისთვის, a∈ GF, მოიძებნება მოპირდაპირე ელემენტი - a∈ GF, ისეთი, რომ a+(-a)=0;

– ნებისმიერი არანულოვანი ელემენტისათვის a∈ GF, მოიძებნება შებრუნებული ელემენტი a^{-1}∈ GF, ისეთი, რომ a· a^{-1}=1.

ორი უკანასკნელი თვისება მიუთითებს, რომ GF შეიცავს 0 და 1.

გალუას ველი შეიძლება აგებულ იქნეს მოდულური გამოთვლების საფუძველზე იმ შემთხვევაში, თუ მოდული m მარტივი რიცხვია. განვიხილოთ მაგალითები:

გალუას ველი მოდულით 2.

გალუას ველი მოდულით 2 წარმოიქმნება იმ ნაშთების სიმრავლით, რომლებიც მიიღებიან მთელი რიცხვების 2-ზე გაყოფისას. რადგან ამ შემთხვევაში ნაშთი შეიძლება იყოს 0 ან 1, ამიტომ GF(2)={0,1}. ეს სიმრავლე აკმაყოფილებს ველის თვისებებს. მართლაც, GF(2)-ის ორი ელემენტის ჯამი მოდულით 2 ეკუთვნის GF(2) სიმრავლეს. შეკრება და გამრავლება მოდულით 2 გამოისახება ცხრილებით:

+	0	1
0	0	1
1	1	0

,

x	0	1
0	0	0
1	0	1

რაც ემთხვევა X_{or} არითმეტიკის ოპერაციებს. 0-ის მოპირდაპირე რიცხვია ისევ 0, რადგან 0+0=0, ხოლო ერთის შებრუნებული რიცხვია ისევ 1, რადგან 1·1=1.

5.2. გაფართოებული გალუას ველი GF(2^m)

მონაცემთა გადაცემისას წარმოქმნილი შეცდომის გასწორების ალგორითმებში მონაცემები განიხილება, არა როგორც ბიტების მიმდევრობა, არამედ რამდენიმე

ბიტისაგან შემდგარი სიმბოლოების მიმდევრობა. ამგვარ წარმოდგენას შეესაბამება ე.წ. გაფართოებული გალუას ველი, $GF(2^m)$, სადაც გამოსახულებაში 2^m რიცხვი 2 აღნიშნავს, რომ მიმდევრობა შედგება ბიტებისაგან, ანუ $GF(2)=\{0,1\}$ ელემენტებისგან, ხოლო თითოეული სიმბოლო შეიცავს m რაოდენობის ბიტს. მაგალითად, თუ m=2, $GF(2^2)$ შეიცავს ორბიტიან მონაცემების (სიმბოლოების) სიმრავლეს {00, 01, 10, 11}. m=3 შემთხვევაში ველი შეიცავს სამბიტიან მონაცემებს (ტრიადებს), ანუ $GF(2^3)=\{000, 001, 010, 011, 100, 101, 110, 111\}$. m=4 შემთხვევაში ველი შეიცავს ბიტების ოთხეულებს (ტეტრიადებს), ანუ $GF(2^4)=\{0000, 0001, ..., 1111\}$. უფრო ხშირად გვხვდება მონაცემების წარმოდგენა რვაბიტიანი ბაიტების (ოქტეტების) სახით და ვღებულობთ ველს $GF(2^8)=\{00000000, 00000001, ... , 11111111\}$.

5.3. პოლინომური წარმოდგენა

გაფართოებული გალუას ველის არითმეტიკა ეყრდნობა ბიტური მიმდევრობის პოლინომის სახით წარმოდგენას და ოპერაციებს პოლინომებზე. მაგალითად, $GF(2^2)$ შემთხვევაში, ელემენტი 10 წარმოდგება როგორც პოლინომი $1*x^1+0*x^0=x$.

$GF(2^3)$ შემთხვევაში, მაგალითად, მიმდევრობას 101 შეესაბამება

$1*x^2+0*x^1+1*x^0=x^2+1$ პოლინომი.

$GF(2^4)$ ველის 1011 ელემენტს შეესაბამება $1*x^3+0*x^2+1*x^1+1*x^0=x^3+x+1$ პოლინომი.

ანალოგიურად, $GF(2^8)$ ველის ელემენტს 10110101 შეესაბამება პოლინომი $1*x^7+0*x^6+1*x^5+1*x^4+0*x^3+1*x^2+0*x^1+1*x^0=x^7+ x^5+ x^4+ x^2+1$.

5.4. პრიმიტიული (მარტივი) პოლინომი

გაფართოებულ გალუას ველში ოპერაციები წარმოებს მოდულით P(x), სადაც P(x) მარტივ პოლინომს წარმოადგენს. როგორც ცნობილია, პოლინომი არის მარტივი, თუ იგი არ იშლება მარტივ მამრავლებად. ლიტერატურაში თითოეული m-თვის რეკომენდებულია მარტივი პოლინომი ველის წარმოქმნისთვის. მაგალითად, m=3-თვის რეკომენდებულია პოლინომი $P(x)=x^3+x+1$, m=4-თვის $P(x)=x^4+x+1$, ხოლო m=8-თვის პოლინომი $P(x)=x^8+ x^4+ x^3+ x^2+1$.

შემდგომში სიმარტივისთვის გამოთვლები ძირითადად ჩატარებულია m=3 შემთხვევისთვის, პოლინომით $P(x)=x^3+x+1$.

5.5. შეკრება

$GF(2^3)$ შეიცავს სამბიტიან მიმდევრობებს და, ამგვარად, ისინი წარმოდგებიან პოლინომებით, რომელთა ხარისხი ნაკლებია 3-ზე. მაგალითად, ელემენტს 101 შეესაბამება პოლინომი $1*x^2+0*x^1+1*x^0=x^2+1$, ხოლო ელემენტს 010 შეესაბამება $0*x^2+1*x^1+0*x^0=x$. ამგვარად, გვექნება

$101+010 \rightarrow x^2+1+x = x^2+x+1 \rightarrow 111$

ცხადია, პოლინომების შეკრების მაგივრად შეგვიძლო შეგვესრულებინა შეკრების ოპერაცია პირდაპირ ელემენტებზე, რაც იმავე შედეგს მოგვცემდა:

$$\begin{array}{ccc} 1 & 0 & 1 \\ \oplus & & \\ 0 & 1 & 0 \\ \hline 1 & 1 & 1 \end{array}$$

(შევნიშნოთ, რომ შეკრება ხდება თანრიგებში დამოუკიდებლად, მოდულით 2, ანუ X_{or} არითმეტიკის შესაბამისად).

რადგანაც მეორე ხარისხის პოლინომების ჯამი არ შეიძლება აღემატებოდეს 3, ანუ, ველის წარმომქმნელი $P(x)=x^3+x+1$ პოლინომის ხარისხს, მოდულური გამოთვლა საჭირო აღარაა.

5.6. გამრავლება

გამრავლების შემთხვევაში ორი მეორე ხარისხის ნამრავლი შეიძლება აღემატებოდეს 3 და საჭირო ხდება ნამრავლის გამოანგარიშება მოდულით $P(x)$. განვიხილოთ ნამრავლი 101X110. პოლინომური წარმოდგენისას გვექნება

$(1*x^2+0*x+1)*(x^2+x) \rightarrow) (x^2+1)*(x^2+x) = x^4+x^3+x^2+x$

რადგან მიღებული ნამრავლის ხარისხი აღემატება $P(x)=x^3+x+1$ ხარისხს, საჭიროა გამოვთვალოთ $(x^4+x^3+x^2+x) \mathrm{mod} P(x)$, ანუ ვიპოვოთ ამ პოლინომების გაყოფის ნაშთი:

$$\begin{array}{cccccccc|cccc} x^4 & + & x^3 & + & x^2 & + & x & & x^3 & + & x & + 1 \\ \hline x^4 & & x^2 & & x & & & & x & + & 1 & \\ \cline{1-5} & & x^3 & & + & & x & & & & & \\ & & x^3 & + & x & & +1 & & & & & \\ & & 0 & + & 0 & & +x & \rightarrow & \text{ნაშთი} & & & \end{array}$$

001

ამგვარად, 101X110=001 (შემოწმდეს)

5.7. პრიმიტიული ელემენტი

გაფართოებული გალუას ველი GF(2^3)={000,001, 010,011, 100, 100, 101, 110, 111} შეიცავს პრიმიტიულ, ანუ მარტივ ელემენტს, ისეთს, რომ მისი მიმდევრობით ახარისხება გვაძლევს ველის ყველა არანულოვან ელემენტს. პრიმიტიულ ელემენტს ტრადიციულად აღნიშნავენ სიმბოლოთი α. ველს შეიძლება ჰქონდეს რამდენიმე პრიმიტიული ელემენტი. ვაჩვენოთ, რომ GF(2^3) პრიმიტიული ელემენტია $\alpha = 010$, ანუ მისი მიმდევრობით 0, 1, 2, ... ხარისხებში აყვანისას მივიღებთ ველის ყველა არანულოვან ელემენტს.

1. $\alpha^0 = (010)^0 = 1 = 001$

2. $\alpha^1 = (010)^1 = 010$

3. $\alpha^2 = (010)^2 = 010 \cdot 010 = 100$

4. $\alpha^3 = (010)^3 = \alpha^2 \cdot 010 = 1000 \qquad \alpha^3 = (\alpha^2 \cdot \alpha) mod 1011$

რადგან მივიღეთ ოთხთანრიგა ნამრავლი, იგი უნდა გავყოთ $x^3 + x + 1$ პოლინომის შესაბამის რიცხვზე ანუ 1011ზე. გვექნება

1000 | 1011	
1011	1
011	(ნაშთი)

საბოლოოდ, $\alpha^3 = 011$

5. $\alpha^4 = \alpha^3 \cdot \alpha = (011 \cdot 010) mod 1011$

თუ შევასრულებთ გამოთვლებს α^3 გამოთვლის ანალოგიურად, გვექნება $\alpha^4 = 110$.

6. $\alpha^5 = \alpha^4 \cdot \alpha = (110 \cdot 010) mod 1011 = 111$

7. $\alpha^6 = \alpha^5 \cdot \alpha = (111 \cdot 010) mod 1011 = 101$

გამოთვლების შედეგები შეიძლება სისტემატიზებულ იქნეს ცხრილში.

ველის ელემენტები ხარისხობრივი გამოსახულებები					ველის ელემენტები რიცხობრივი გამოსახულებები
	0	0	0		
α^0	0	0	1		
α^1	0	1	0		
α^2	1	0	0		
α^3	0	1	1		
α^4	1	1	0		
α^5	1	1	1		
α^6	1	0	1		

GF(2^3) ველის ელემენტების წარმოდგენა ხარისხების სახით საშუალებას იძლევა შევადგინოთ შეკრების და გამრავლების ცხრილები, რაც აადვილებს გამოთვლებს.

შეკრების შემთხვევაში, მაგალითად, 011⊕111=100, რასაც შეესაბამება გამოსახულება $\alpha^3 \oplus \alpha^5 = \alpha^2$. თუ ამგვარად გამოთვლებს ჩავატარებთ ველის ელემენტების ყველა წყვილისთვის, მივიღებთ შეკრების ცხრილს (ტაბულას).

	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	0	α^3	α^6	α^1	α^5	α^4	α^2
α^1	α^3	0	α^4	α^0	α^2	α^6	α^5
α^2	α^6	α^4	0	α^5	α^1	α^3	α^0
α^3	α^1	α^0	α^5	0	α^6	α^2	α^4
α^4	α^5	α^2	α^1	α^6	0	α^0	α^3
α^5	α^4	α^6	α^3	α^2	α^0	0	α^1
α^6	α^2	α^5	α^0	α^4	α^3	α^1	0

რაც შეეხება გამრავლებას, აქაც უნდა შვასრულოთ გამრავლება მოდულით 1011. მაგალითად, $\alpha^5 \otimes \alpha^3 = (111 \cdot 011) mod 1011 = 010 = \alpha^1$.

თუ ამგვარ გამოთვლებს შევასრულებთ ყველა წყვილისთვის, მივიღებთ გამრავლების ცხრილს.

	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^1	α^1	α^2	α^3	α^4	α^5	α^6	α^0
α^2	α^2	α^3	α^4	α^5	α^6	α^0	α^1
α^3	α^3	α^4	α^5	α^6	α^0	α^1	α^2
α^4	α^4	α^5	α^6	α^0	α^1	α^2	α^3
α^5	α^5	α^6	α^0	α^1	α^2	α^3	α^4
α^6	α^6	α^0	α^1	α^2	α^3	α^4	α^5

შეკრებისა და გამრავლების ცხრილების გამოყენებით მიზანშეწონილია კონტროლის ალგორითმების აპარატულად რეალიზაციისთვის. რაც შეეხება ხელით გამოთვლებს, გამრავლების შემთხვევაში ვისარგებლოთ შემდეგი მარტივი წესით. რადგან GF(2^3) შეიცავს 7 არანულოვან ელემენტს, ორი ხარისხოვანი გამოსახულების გამრავლებისას ნამრავლის ხარისხი ტოლია თანამამრავლთა ხარისხის მაჩვენებლების ჯამის, მოდულით 7. მაგალითად,

$$\alpha^5 \otimes \alpha^6 = (\alpha^{(5+6) mod 7} = \alpha^{11 mod 7} = \alpha^4$$

$GF(2^3)$-ის მსგავსად, ანალოგიური ცხრილები შეიძლება შედგეს $GF(2^4)$ და $GF(2^8)$, თუმცა $GF(2^8)$ შემთხვევაში ცხრილი შიცავს $2^8=256$ სვეტს და სტრიქონს და გამოყენებისთვის მოუხერხებელია.

თავი 2. კრიპტოგრაფიის მეთოდები

§1. მონაცემთა ჰეშირება

ქვემოთ განხილული იქნება თანამედროვე კრიპტოგრაფიული ჰეშ-ფუნქციები, რომლებიც ფართო გამოყენებას პოულობს კრიპტოგრაფიაში. ამავე დროს უნდა აღინიშნოს, რომ ჰეშირებას გამოყენების ხანგრძლივი ისტორია აქვს ისეთ სფეროებში, როგორიცაა, მაგალითად:

– კომპილატორები, სადაც ხდება პროგრამის მაღალი დონის პროგრამირების ენიდან მანქანურ ენაზე გადათარგმნა და საჭიროა გამოყენებული იდენტიფიკატორების ცხრილების შედგენა იდენტიფიკატორების მახასიათებლების დამახსოვრებისა და მათი მოძებნის მიზნით;

– მონაცემთა გადაცემის სისწორის კონტროლი, როგორც კომპიუტერში, ისე კომპიუტერულ ქსელებში, სადაც მაკონტროლებელი სიდიდე (ჰეში) მიიღება როგორც ორობით სისტემაში წარმოდგენილი გასაკონტროლებელი რიცხვის სპეციალურად შერჩეული სტანდარტულ რიცხვზე გაყოფის შედეგად მიღებული ნაშთი (როგორც წესი, გაყოფის ოპერაცია სრულდება X_{02} არითმეტიკის წესების შესაბამისად).

კრიპტოგრაფიის განვითარების კვალობაზე თანდათანობით გამოიკვეთა გამკაცრებული მოთხოვნები ჰეშირების პროცედურების საიმედოობისადმი, რამაც განაპირობა კრიპტოგრაფიული ჰეშირების ალგორითმის შექმნა, რომლებიც მჭიდროდ არიან დაკავშირებული კრიპტოგრაფიული შიფრაცია/დეშიფრაციის მეთოდებთან.

1.1. კრიპტოგრაფიული ჰეშ-ფუნქციები

კრიპტოგრაფიული ჰეშ-ფუქცია H წარმოადგენს ფუნქციას, რომელიც ნებისმიერი ზომის ბიტურ მონაცემებს, m (message, გზავნილი), გარდასახავს ფიქსირებული სიგრძის ბიტურ მონაცემად, რასაც m-ის ჰეში ეწოდება. ამ ფუნქციის განსაზღვრის არეს წარმოადგენს ნებისმიერი ბიტური მონაცემი, რადგან თუ საწყისი მონაცემი წარმოდგენილია სხვა, მაგალითად ტექსტური ფორმით, ხდება მისი ორობით მონაცემად გარდაქმნა სიმბოლოების ბიტური კოდირების გამოყენებით, რაც შეეხება ამ

ფუნქციის მნიშვნელობათა სიმრავლეს, იგი შეიძლება იყოს ფიქსირებული სიგრძის, მაგალითად 128 ბიტიანი, 256 ბიტიანი ან სხვა სიგრძის ორობითი მონაცემი. ამგვარად ჰეშ-ფუნქციისათვის ვღებულობთ ფუნქციურ ასახვას, f,

$$H(m) = h,$$

სადაც ჰეში, h, m-ს სახეა (emage), ხოლო m არის h-ის პირველსახე (preemage). როგორც ითქვა, m შეიძლება იყოს ნებისმიერი ზომის ბიტური რიცხვი და H(m) ფუნქციის განსაზღვრის არე ძალიან ფართოა (შემოუსაზღვრავია), ხოლო ჰეში, თუ მას განვიხილავთ როგორც ორობით რიცხვს, მოქცეულია დიაპაზონში (0 _ 2^d-1), სადაც d არის ჰეშის ბიტების რაოდენობა.

ფუნქცია H(m)-ის დამახასიათებელი მოთხოვნაა ის, რომ იგი ცალმხრივი (one-way) ფუნქციაა: ნებისიმიერი m-თვის ადვილად (სწრაფად) ხდება მისი სახის (ანასახის) დადგენა, მაგრამ მოცემული h-ათვის მისი წინასახის დადგენა, ანუ $h=H^{-1}(m)$ გარდასახვის, განხორციელება რთული და არაცალსახაა.

1.2 კრიპტოგრაფიული ჰეშ-ფუნქციების თვისებები

m მონაცემის ჰეშის მნიშვნელობა h შეიძლება განვიხილოთ როგორც „თითის ანაბეჭდი“, იგი იმის მსგავსად ახასიათებს მონაცემს, როგორც თითის ანაბეჭდი ახასიათებს ადამიანს: თითოეული ადამიანს შეესაბამება უნიკალური თითის ანაბეჭდი და უკიდურესად ძნელია ვიპოვოთ ორი ადამიანი, რომელთაც ერთნაირი თითის ანაბეჭდი გააჩნიათ, რაც ფართოდ გამოიყენება კრიმინოლოგიურ გამოკვლევებში. იმისათვის, რომ ჰეშ-ფუნქცია იყოს საიმედო ინსტრუმენტი კრიპტოგრაფიაში, მას უნდა გააჩნდეს შემდეგი თვისებები:

– გამოთვლითი თვალსაზრისით მძიმე უნდა იყოს h-ის მიხედვით მისი პირველსახის m-ის პოვნა. ეს უკიდურესად გაუძნელებს კრიპტოგრაფიულ შეტევას ჰაკერს, რომელიც ცდილობს h-ით იპოვოს m;

– თუ H ფუნქცია m_1 მონაცემისთვის იძლევა $H(m_1)=h$, ძნელი უნდა იყოს ისეთი m_2 მონაცემის პოვნა, რომლისთვისაც $H(m_2)=h$. ე.ი. ჰეშის ერთ მნიშვნელობას ორი განსხვავებული მნიშვნელობა არ უნდა შეეფარდებოდეს; ეს ნიშნავს, რომ თუ ჰაკერმა იცის მონაცემი m და მისი ჰეში h, მისთვის ძნელი უნდა იყოს m-ის სხვა მონაცემით ჩანაცვლება;

– ძნელი უნდა იყოს ისეთი m_1 და m_2 პოვნა, რომ $H(m_1)=H(m_2)$. ეს თვისება გამორიცხავს კოლიზიებს ჰაშების მნიშვნელობათა შორის და განსაკუთრებით მკაცრ მოთხოვნას წარმოადგენს;

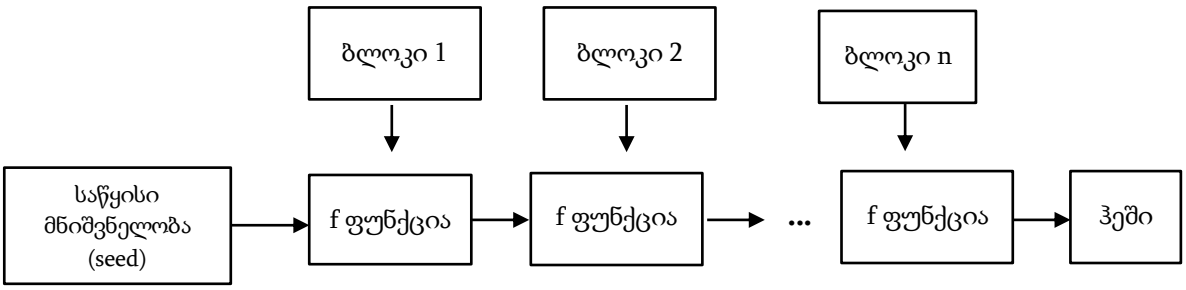
– ჰეშ-ფუნქცია არის დეტერმინისტული ფუნქცია, რაც ნიშნავს, რომ ერთიდაიგივე m-თვის H ფუნქციის გამოყენება უნდა იძლეოდეს ერთსადაიმავე პასუხს;

– ნებისმიერი m-თვის H(m)-ის გამოთვლა სწრაფი უნდა იყოს;

– ჰეშ ფუნქცია უნდა ხასიათდებოდეს ზვავისებური ეფექტით. ეს ნიშნავს, რომ ორი m_1 და m_2 ერთმანეთისგან განსხვავდება მცირედ, თუნდაც ერთი ბიტით. ჰეშ-ფუნქციის შესრულების პროცესში ეს სხვაობა უნდა გაძლიერდეს და უნდა მიღებული იქნეს h_1 და h_2 ჰეშები, რომლებიც, ერთმანეთისგან ძლიერ განსხვავდებიან.

1.3. ჰეშირების ალგორითმი

ჰეშირების ალგორითმი პროცედურულად საკმაოდ ახლო დგას ბლოკური შიფრაციის ალგორითმებთან, რომელიც უფრო ზუსტად მომდევნო პარაგრაფში იქნება განხილული. როგორც შიფრაციის, ისე ჰეშირების ალგორითმებში მონაცემი m იყოფა ტოლი სიდიდის ბლოკებად m_1, m_2, m_3, ... , m_n, დაწყებული მარცხნიდან. საჭიროების შემთხვევაში, თუ m მონაცემის სიგრძე ბლოკის სიგრძის ჯერადი არ არის, ხდება მისი შევსება დამატებითი მონაცემებით (padding) მონაცემის ბოლოში. სურათზე ნაჩვენებია პეშირების ალგორითმის გამარტივებული სქემა.



აქ გამოთვლები დაყოფილია რაუნდებად. ყოველ რაუნდში f ფუნქცია ასრულებს დაშიფრვის ოპერაციას, რომელშიაც მონაცემთა ბლოკს იყენებს როგორც გასაღებს (key), ხოლო წინა რაუნდიდან გადმოცემულ შიფრს როგორც დასაშიფრავ მონაცემს. პირველ რაუნდში გამოიყენება საწყისი მონაცემი (seed) და ბლოკი 1 და მიღებული შედეგი გადაეცემა მეორე ბლოკს და ა.შ. ბლოკების ამოწურვამდე. მიღებული შედეგი წარმოადგენს საწყისი m მონაცემის ჰეშს.

ამგვარად, თუ ბლოკის სიგრძე ბაიტებში იქნება, მაგალითად, 128 ბიტი, ფუნქცია f ყოველ ნაბიჯზე შესავალზე ღებულობს ორ 128 ბიტიან რიცხვს და გამოსავალზე გამოიმუშავებს ისევ 128 ბიტიან რიცხვს, რომელიც გადაეცემა მომდევნო რაუნდს. საბოლოოდ შეიძლება ითქვას, რომ აღწერილი პროცედურა ახდენს ბლოკის მონაცემების თანდათანობით შეყურსვას, რის შედეგად გამოსავალზე მიიღება m მონაცემის 128 ბიტიანი ჰეში.

წლების მანძილზე დამუშავებული იქნა ჰეშირების ალგორითმების რამდენიმე ოჯახი, რომელთა გავრცელება ხდება სახელით – უსაფრთხო ჰეშირების ალგორითმი, SHA (Sequre Hash Algorithm), ორგანიზაცია NIST (National institute of Standards and Technology) რეკომენდაციით დამუშავებულ იქნა ოჯახები SHA-1, SHA-2, SHA-3, SHA-4, სადაც მომდევნო ოჯახი ხასიათდება გაზრდილი ეფექტურობით. ამჟამად ყველაზე მეტად გავრცელებულია SHA-2, რომელშიც შედიან SHA-256, ჰეშ ალგორითმი, რომელიც წარმოქმნის 256 ბიტიან ჰეშს და SHA-512 512 ბიტიანი ჰეშით.

ჰეშირების მეთოდები, შიფრაცია/დეშიფრაციის მეთოდებთან ერთად, პოულობს ფართო გამოყენებას მონაცემთა უსაფრთხოების დაცვაში ისეთი მახასიათებლების უზრუნველყოფაში, როგორიცაა აუთენტიკაცია, მონაცემთა უცვლელობის დაცვა, ელექტრონული ხელისმოწერა და სხვა.

§2. სიმეტრიული ბლოკური შიფრაცია: AES

შიფრაციის სტანდარტი AES (Advenced Encryption Standart, გაუმჯობესებული შიფრაციის სტანდარტი) წარმოადგენს შიფრაციის სტანდარტების ოჯახის, სახელწოდებით Rijndael (ჰოლანდიურად წარმოითქმება როგორც „რაინდოლ“), ერთ-ერთ ვარიანტს. Rijndael დამუშავებულ იქნა ჰოლანდიელი კრიპტოგრაფების დომენის (John Daemen) და რაიმენის (Vincent Rijmen) მიერ, ის გამარჯვებული გამოვიდა საერთაშორისო კონკურსში და მისი ერთ-ერთი ვარიანტი მიღებულ იქნა სტანდარტად NIST-ის (National Institute of Standards and Technology) მიერ 2001 წლიდან სახელწოდებით AES.

2.1. Rijndael-ის ზოგადი დახასიათება

ბლოკური კრიპტოგრაფიული სისტემის ორ ძირითად მონაცემს წარმოადგენს შიფრის გასაღები K (Cypher Key) და მონაცემთა ბლოკი ღია ტექსტით B. გასაღების ზომა შეიძლება იყოს დაწყებულ 128 ბიტიდ ან 32 ბიტიანი ბიჯით 256 ბიტამდე, ანუ 128, 160, 192, 224 და 256 ბიტი. თუმცა, პრაქტიკულ გამოყენებას უფრო ხშირად პოულობენ 128 და 256 ბიტიანი გასაღებები. რაც შეეხება ღია ტექსტის ბლოკს, მისი ზომაც ბიტებში შეიძლება იყოს 128-დან 256-მდე ბიჯით 32.

შემდგომში განხილული იქნება ვარიანტი 128 ბიტიანი გასაღებით და 128 ბიტიანი ღია ტექსტის ბლოკით, რაც AES კრიპტოსისტემის გამოყენების ძირითად ვარიანტს შეესაბამება.

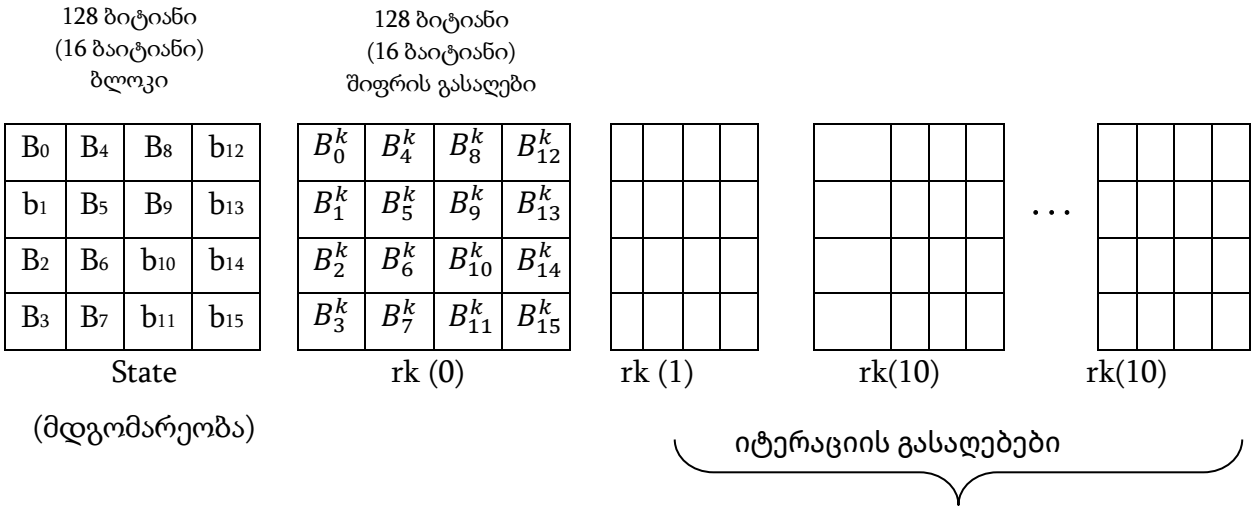
რადგან Rijndael ალგორითმებში დამუშავების ძირითად ერთეულს წარმოადგენს ბაიტი, ამიტომ როგორც შიფრის გასაღები K, ისე ღია ტექსტის ბლოკი, წარმოდგებიან როგორც ბაიტების მიმდევრობა.

128 ბიტიანი გასაღების, K, ბაიტურად წარმოდგენისათვის საკმარისია ბიტების მიმდევრობა, დაწყებული მარცხნიდან, დავყოთ რვაბიტიან ქვეჯგუფებად. პირველი რვა ბიტი განსაზღვრავს შიფრის ბაიტს B_0^k, მომდევნო რვა ბიტი იძლევა B_1^k და ა. შ. ბოლო რვა ბიტი განსაზღვრავს B_{15}^k.

რაც შეეხება დასაშიფრავ გზავნილს, M (Message), ხდება მისი შემადგენელი სიმბოლოების შეცვლა ბიტური წარმოდგენით, მაგალითად, ASCII (American Standard Code for Informatoion Interchange) კოდის გამოყენებით. ამის შედეგად გზავნილი M წარმოდგება როგორც ბიტების მიმდევრობა და ხდება მისი დაყოფა 128 ბიტიან ბლოკებად, დაწყებული მარცხნიდან. იმ შემთხვევაში, თუ გზავნილის ბიტების რაოდენობა არ აღმოჩნდება 128-ის ჯერადი, ხდება ბოლო ბლოკის შევსება დამხმარე მონაცემით (Padding). M გზავნილის დაშიფრვის ამოცანა დაიყვანება მიღებული ბლოკების მიმდევრობით დაშიფრვაზე და მიღებული შიფრების გაერთიანება (კონკატენაცია) იძლევა M-ის შიფრს.

თითოეული ღია ტექსტის 128 ბიტიანი ბლოკი, ზემოთ აღწერილის მსგავსად, წარმოდგება როგორც ბაიტების მიმდევრობა $B_0, B_1, B_2, \ldots, B_{15}$.

გამოთვლებში ზემოაღწერილი ბაიტების მიმდევრობა წარმოდგება ორგანზომილებიანი მასივების სახით, როგორც სურათი 1-ზეა ნაჩვენები.



სურათი 1

როგორც სურათი გვიჩვენებს მონაცემები წარმოდგენილია ორგანზომილებიანი მასივების სახით. მაგალითად მასივი state, რომელშიაც აისახება გამოთვლების როგორც საწყისი, ისე შუალედური და საბოლოო მდგომარეობა, თავიდან შეიცავს 128 ბიტიან ღია ტექსტის შესაბამის ბაიტებს. პირველ სვეტში განლაგებულია ბაიტები B_0, B_1, B_2, B_3, მეორეში B_4, B_5, B_6, B_7 და ა. შ. ანალოგიურად, მასივი rk (0) შეიცავს შიფრის გასაღების, K, შემადგენელ ბაიტებს და გასაღები ინახება rk (0) მასივში მთლიანი გამოთვლების მანძილზე. რადგან AES კრიპტოსისტემაში გამოთვლები ხორციელდება რაუნდების სახით, rk(1), rk(2) და ა.შ. მასივები შეიცავენ რაუნდის გასაღებებს, რომლებიც წარმოიქმნებიან rk(0)-ში შენახული შიფრის გასაღების საფუძველზე. იმისდა მიხედვით, თუ როგორ პლატფორმაზე ხორციელდება ალგორითმის იმპლემენტაცია, ყველა რაუნდის გასაღებები შეიძლება შეიქმნას გამოთვლების დაწყებისას, ან რაუნდის გასაღები შეიძლება შეიქმნას იტერაციის დასაწყისში (მაგალითად იმ შემთხვევაში, როცა პლატფორმა დაფუძნებულია ცენტრალურ პროცესორზე ერთბაიტიანი რეგისტრებით და შეზღუდული მეხსიერებით).

შემდგომში გამოყენებული იქნება აგრეთვე ორგანზომილებიანი მასივის წარმოდგენა ორგანზომილებიანი მატრიცის სახით, მაგალითად, მასივი State შეიძლება წარმოდგენილ იქნას როგორც

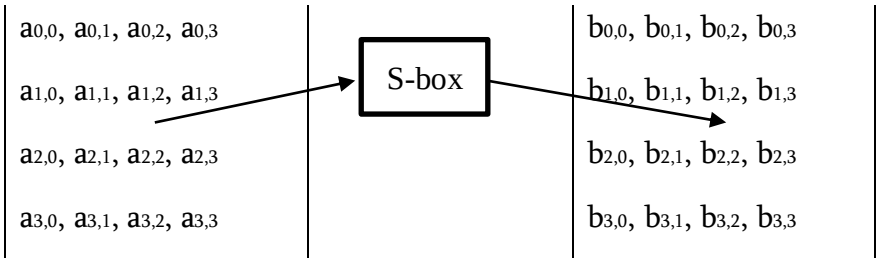
$$\text{State } (i,j) = \begin{vmatrix} S_{0,0}, S_{0,1}, S_{0,2}, S_{0,3} \\ S_{1,0}, S_{1,1}, S_{1,2}, S_{1,3} \\ S_{2,0}, S_{2,1}, S_{2,2}, S_{2,3} \\ S_{3,0}, S_{3,1}, S_{3,2}, S_{3,3} \end{vmatrix}$$

სადაც i=0, 1, 2, 3 სტრიქონის ნომერია, j=0, 1, 2, 3 სვეტის ნომერია, ხოლო State (i, j) = $S_{i,j}$.

State-ის გარდაქმნებში გამოყენებული პრიმიტივები, როგორც ითქვა, მონაცემთა სტრუქტურა State გამოთვლების დასაწყისში შეიცავს ღია ტექსტის ბლოკს წარმოდგენილს ბაიტების სახით. გამოთვლებში ხდება ამ მონაცემების გარდაქმნა და, ამგვარად, State შეიცავს გამოთვლების შუალედურ შედეგებს, ხოლო საბოლოოდ იგი შეიცავს ბაიტებს, რომლებიც განსაზღვრავენ საწყისი ღია ტექსტის (Plaintext) შესაბამის შიფრტექსტს (ciphertext). ეს მიზანი მიიღწევა იტერაციული პროცედურების რეალიზაციის გზით, რომელიც დაფუძნებულია რამდენიმე ქვემოთ მოყვანილი გარდამქმნელი პრიმიტივის (ფუნქციის) გამოყენებაზე.

2.2. გარდაქმნა ByteSub

ფუნქცია ByteSub (Byte Substitution, ბაიტის ჩანაცვლება) მოქმედებს State-ს ბაიტებზე ინდივიდუალურად. ამ ოპერაციაში გამოიყენება წინასწარ შექმნილი ცხრილი დასახელებით S-box (Substitution box). მასივ State-ის ელემენტი (ბაიტი) State (i,j) გამოიყენება როგორც S-box-ის ინდექსი და State-ის (i,j) ბაიტი შეიცვლება S-box-დან ამოღებული ელემენტით. ამ ოპერაციის სქემატურ წარმოდგენას იძლევა სურათი 2.



როგორც სურათი 2 გვიჩვენებს აქ ხდება მატრიცა (State)-ს a2,2ბაიტის ჩანაცვლება b 2,2 ახალი მნიიშვნელობით. ეს ახალი მნიშვნელობა გამოთვლება S-box ბლოკის საშუალებით. ჩვენს შემთხვევაში ჩასანაცვლებელი a 2,2 ბაიტის პირველი 4 ბიტი გამოვიყენოთ როგორც S-box ის სტიქონის ნომერი, ხოლო მომდევნო 4 ბიტი გამოვიყენოთ, როგორც S-box -ის სვეტის ნომერს. ამგვარად სტრიქონისა და სვეტის გადაკვეთის ელემენტზე მოთავსებულია ჩამნაცვლებელი ბაიტის მნიშვნელობა, რომელიც ჩვენს შემთხვევაში გამოთვლება როგორც a2,2-ის მოდულური მულტიპლიკაციური ინვერსია. სურათი 3.

გამოთვლები ყველა უჯრისათვის გამოითვლება ინდივიდუალურად და პარალელურად.

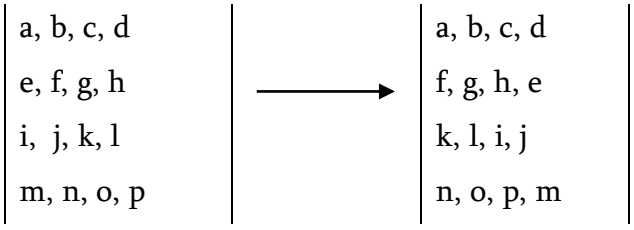
ამ პროცედურის გამოყენება State-ს ყველა ბაიტისადმი აღინიშნება როგორც **ByteSub (State)**.

X	y															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	1	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	A0	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	4	C7	23	C3	18	96	5	9A	7	12	80	E2	EB	27	B2	75
4	9	83	2C	1A	1B	6E	5°	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	0	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	5C	CF
6	D0	EF	AA	FB	43	4D	33	85	45	19	2	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	6	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	8
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	80
D	70	3E	B5	66	48	3	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

სურათი 3.

2.3. გარდაქმნა shiftRow

გარდაქმნა shiftRow (სტრიქონის დაძვრა) წარმოადგენს გადანაცვლების (permutation) ოპერაციას, რომელიც ხორციელდება State-ის სტრიქონებზე. გამოყენებული ალგორითმის თანახმად State მასივის პირველი სტრიქონი რჩება უცვლელად. მეორე სტრიქონის ელემენტები ციკლურად დაიძვრებიან მარცხნივ ერთი პოზიციით, მესამე სტრიქონის ელემენტები 2 პოზიციით, ხოლო მეოთხე სტრიქონის ელემენტები 3 პოზიციით. ამ ელემენტების გადანაცვლების შედეგები ნაჩვენებია სურათი 3-ზე.



სურათი 3. მასივის ელემენტების ციკლური ძვრა მარცნივ

ეს გარდაქმნა სრულდება State-ს ყველა სტრიქონზე და იგი აღინიშნება როგორც shiftRow (State).

2.4. გარდაქმნა MixColumn

გარდაქმნა MixColumn ხორციელდება თითოეული სვეტისათვის შემდეგი მატრიცული ოპერაციის საფუძველზე.

$$\begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{vmatrix} = \begin{vmatrix} 02, 03, 01, 01 \\ 01, 02, 03, 01 \\ 01, 01, 02, 03 \\ 03, 01, 01, 02 \end{vmatrix} \cdot \begin{vmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{vmatrix},$$

სადაც | b_0, b_1, b_2, b_3 | სვეტის ახალი მნიშვნელობაა, ხოლო | a_0, a_1, a_2, a_3 | არის სვეტის მნიშვნელობა გარდაქმნამდე. აქ b_0, b_1, b_2, b_3 და a_0, a_1, a_2, a_3 ღებულობენ ბაიტურ მნიშვნელობებს, ხოლო '01', '02', '03' მატრიცის ელემენტებია თექვსმეტობით სისტემაში.

ამ გარდაქმნის შედეგად ხდება State-ს სვეტების შემადგენელი ბაიტების ახალი მნიშვნელობების გამოთვლა ძველი მნიშვნელობების კომბინირების საფუძველზე.

ამ გარდაქმნის ყველა სვეტისათვის გამოყენება აღინიშნება როგორც **MixColumn.**

2.5. გარდაქმნა AddRound Key (State, RoundKey)

მოცემული გარდაქმნა იყენებს ორ მონაცემს: მასივ State-ს და მასივ RoundKey. მოქმედებია სრულდება ბაიტურად, როცა ხდება ბაიტ State (i, j)-ის ორის მოდული შეკრება RoundKey (i, j) ბაიტთან.

$$State(i, j) := State(i, j) \oplus RoundKey(i, j),$$

$$i=0, 1, 2, 3 \quad j=0, 1, 2, 3.$$

ამგვარად, ხდება მასივ State-ს განახლება State და RoundKey მასივების ბაიტებზე X_{02} ოპერაციის ჩატარების საფუძველზე, რაც აღინიშნება როგორც **AddRoundKey (State, RoundKey**).

2.6. შიფრაცია

როგორც ამას სურათი 1 გვიჩვენებს, გამოთვლების დასაწყისში მასივი State შეიცავს დასაშიფრავი 16 ბიტიანი ბლოკის (ღია ტექსტის) ბაიტებს, ხოლო მასივი rk(0) შეიცავს

შიფრის გასაღების (cipher Key) ბაიტებს. ღია ტექსტის დაშიფვრის პროცედურა მდგომარეობს შემდეგი რაუნდების შესრულებაში:

- სრულდება რაუნდი ნომერი 0;
- ერთმანეთის მიყოლებით სრულდება რაუდები ნომრებით 1÷ 9;
- სრულდება რაუნდი ნომრით 10.

ამ რაუნდების შესრულების შედეგად მასივ State-ში აღმოჩნდება საწყისი ღია ტექსტის შესაბამისი დაშიფრული ტექსტის ბაიტები, რომელთა კონკატენაცია წარმოქმნის ღია ბლოკის შესაბამის დაშიფრულ ტექსტს (Block cipher text).

აღწერილი პროცედურა სრულდება ღია M გზავნილის ყველა ბლოკისათვის და მათი კონკატენაცია იძლევა M გზავნილის შესაბამის შიფრტექსტს (CipherText).

რაც შეეხება გარდაქმნებს, ისინი რაუნდების მიხედვით მდგომარეობენ შემდეგში.

2.6.1. რაუნდში 0 სრულდება პროცედურა AddRound Key (State, rk(o), რომლის შედეგად ხდება State-ის ბაიტების გარდაქმნა State (i, j): = State (i, j) $\oplus$ rk(o) (i, j) ყველა ბაიტისთვის.

2.6.2. სრულდება რაუნდები ნომრებით 1-დან 9-მდე.

ამ რაუნდებში ყველაში გამოიყენება მდგომარეობის აღმწერი მასივი State. ამავე დროს თითოეულ რაუნდში გამოიყენება რაუნდის შესაბამისი გასაღების მასივი: პირველ რაუნდში მასივი rk(1), მეორეში rk(2) და ა.შ. ზოგადად რაუნდში შესრულებადი გარდაქმნები შეიძლება გამოისახოს როგორც განზოგადოებული პროცედურა

```
Round (State, RoundKey)
{
    ByteSub (State);
    ShiftRow (State);
    MixColumn (State);
    AddRoundKey (State, RoundKey);
}
```

როგორც ვხედავთ, გამოთვლები წარმოებს ძირითადად მასივ State-თან დაკავშირებით, ხოლო RoundKey-ის ჩანაცვლება ხდება პირველ რაუნდში მასივით rk(1), მეორეში rk(2) და ა.შ.

2.6.3. ფინალური მეათე რაუნდი სრულდება როგორც

```
Round (State, RoundKey)
{
    ByteSub (State);
    ShiftRow (State);
    AddRoundKey (State, RoundKey);
}
```

როგორც ვხედავთ ფინალურ რაუნდში არ გამოიყენება გარდაქმნა MixColumn.

2.7. დეშიფრაცია

მას შემდეგ, რაც M გზავნილის გამგზავნ კომპიუტერში მიღებულია M-ის შიფრტექსტი, ხდება მისი გადაგზავნა მიმღებთან. მიმღები მხარე ახდენს შიფრტექსტის დაყოფას 128 ბიტიან ბლოკებად და ახდენს ამ ბლოკების დეშიფრაციას. ამისათვის იგი ბლოკის შიფრტექსტის შემადგენელ ბაიტებს ათავსებს State მასივში და ა.შ. გარდა ამისა ხდება 16 ბაიტიანი გასაღების შეტანა rk(0) სვეტებში და რაუნდის გასაღებების შეტანა rk(1), rk(2) და ა. შ. მასივებში. გარდა ამისა, გამოიყენება შიფრაციის დროს გამოყენებული გარდაქმნების ინვერსიული, შებრუნებული გარდაქმნები.

დეშიფრაციის თითოეულ ციკლში ხდება გარდაქმნების განხორციელება შებრუნებული მიმდევრობით. მაგალითად, შიფრაციის დროს ბოლო რაუნდი ხორციელდებოდა სქემით

```
Round (State, RoundKey)
{
    ByteSub (State);
    ShiftRow (State);
    AddRoundKey (State, RoundKey);
```

},

დეშიფრაცია იწყება შიფრაციის ბოლო რაუნდის საპირისპირო გარდაქმნებით InvRound (State, RoundKey)

{

AddRoundKey (State, RoundKey);

InvShiftRow (State);

InvByteSub (State);

}

როგორც ვხედავთ, შიფრაციისას ბოლო რაუნდის ბოლო გარდაქმნა იყო AddRoundKey (State, RoundKey), დეშიფრაციის დროს კი პირველი ოპერაცია არის AddRoundKey (State, RoundKey). ეს ოპერაცია მდგომარეობს State-ს და RoundKey-ს ბაიტების შეკრებაში მოდულით 2, და ამგვარად, იგი ავტოინვერსიულია. შიფრაციის დროს ბოლო რაუნდის ბოლოსწინა ოპერაციაა ShiftRow (State), რაც მდგომარეობს State-ს სტრიქონების ციკლურად ძვრაში მარცხნივ. დეშიფრაციისას ბოლო რაუნდის მეორე ოპერაციაა InvShiftRow (State), რაც მდგომარეობს State-ს სტრიქონების ციკლურად ძვრაში მარჯვნივ, რაც ანეიტრალებს ShiftRow (State) ოპერაციის მოქმედებას. ანალოგიურად, დეშიფრაციის ბოლო რაუნდის გარდაქმნა InvByteSub (State) ანეიტრალებს ByteSub (State) გარდაქმნას.

ამგვარად, დეშიფრაციის პირველი რაუნდი ანეიტრალებს შიფრაციის ბოლო რაუნდის ზემოქმედებას State-ზე. ეს პროცესი გრძელდება რაუნდებისათვის ნომრებით 10, 9, ... , 1, 0, რის შედეგადაც State-ში აღმოჩნდება შიფრტექსტის შესაბამისი ღია ტექსტი.

§3. კრიპტოგრაფიული სისტემა ღია გასაღებით: RSA

3.1. ზოგადი განხილვა

ამჟამად ფართოდ გავრცელებული ასიმეტრიული კრიპტოგრაფიის სისტემა RSA შეიქმნა 1977 წელს და მისი დასახელება წარმოდგება შემქმნელების – **R**ivest, **S**hamiz, **A**dleman – გვარების პირველი ასოებისაგან. განსხვავებით სიმეტრიული კრიპტოგრაფიის სისტემენისაგან, სადაც როგორც შიფრაცია, ისე დეშიფრაცია ხდება ერთი საიდუმლო გასაღებისა (Private Key) გამოყენებით, RSA კრიპტოსისტემაში გამოიყენება ორი

გასაღები: ღია გასაღები (Public Key) და დახურული (Private Key). ღია ანუ საჯარო გასაღები ადვილად ხელმისაწვდომია და იგი გამოიყენება დეშიფრაციის (Decryption) ოპერაციაში.

RSA კრიპტოსისტემა ეყრდნობა რიცხვთა თეორიაში მიღებულ შედეგებს და მისი ძირითადი შინაარსი შეიძლება შემდეგნაირად ჩამოყალიბდეს:

შესაძლებელია ვიპოვოთ მთელი რიცხვები e, d და n ისეთი, რომ ნებისმიერი მთელი m რიცხვისათვის, რომელიც აკმაყოფილებს უტოლობას $0 \leq m < n$, ადგილი აქვს ტოლობას

$$(m^e)^d \equiv 1 \bmod n,$$

სადაც ≡ კონგრუენტულობის სიმბოლოა. როგორც ვხედავთ, ამ ფორმულაში ძირითად ოპერაციებს წარმოადგენს ახარისხება და მოდულური გამოთვლები.

1. კრიპტოსისტემის შექმნა იწყება n-ის განსაზღვრით, რომელიც მარტივად ჩამოყალიბდება:

განსაზღვრეთ $n = p \cdot q$,

სადაც p და q მარტივი რიცხვებია. რიცხვთა მცირე დიაპაზონისთვის ეს ადვილად კეთდება, რადგან $p \cdot q$ პოვნა იოლი ოპერაციაა. მაგრამ ამოცანა მოულოდნელად გართულდება თუ იქნება მოთხოვნა „p და q დიდი მარტივი რიცხვებია“. წამოიჭრება შეკითხვა: როგორ ვიპოვოთ დიდი მარტივი რიცხვები? ამისათვის შეიძლება გამოვიყენოთ ერატოსთენეს საცერი (Eratosthene's Sieve), რომელიც საშუალებას იძლევა შევადგინოთ მარტივი რიცხვების სია, რომელიმე დიდ N რიცხვამდე. რამდენადაც მარტივი რიცხვები საკმაოდ მკვრივად არიან განლაგებული მთელ რიცხვთა სიმრავლეში, ამიტომ მივიღებთ ძალიან დიდ სიას, რომლის დამახსოვრება და გამოყენება პრობლემებთანაა დაკავშირებული. ეხლა წარმოვიდგინოთ რეალური სიტუაცია, როცა მოითხოვება, ვთქვათ 512 ბიტიანი მარტივი რიცხვების გამოყენება. ამ შემთხვევაში უკვე ცხრილების გამოყენება სრულიად არაეფექტურია და ამ მიზნით დამუშავებულია სპეციალიზებული online კალკულატორები, რომლებიც საშუალებას იძლევიან შემოწმდეს რომელიმე დასახელებული რიცხვი სიმარტივეზე (Primality Texting). ამ შემთხვევაში მოწმდება არის თუ არა აღებული დიდი რიცხვი მარტივი. თუ არა აიღება ახალი რიცხვი და ეს გრძელდება სანამ არ იქნება ნაპოვნი დიდი მარტივი რიცხვი. ამგვარად, საბოლოოდ განისაზღვრება n-ის მნიშვნელობა.

RSA კრიპტოსისტემაზე შეტევის ძირითადი მიმართულებაა n-ს მარტივ მამრავლებად დაშლა, რადგან p და q განსაზღვრის შემთხვევაში, როგორც ქვემოთ ვნახავთ, ადვილია კრიპტოსისტემის გატეხვა. ამიტომ RSA სისტემაში გამოიყენება მართლაც დიდი რიცხვები, რომელთა მარტივ მამრავლებად დაშლას იმდენად დიდი დრო სჭირდება,

რომ ამ მიმართულებით მცდელობას აზრი არ აქვს და ეს მდგომარეობა კიდევ დიდ ხანს იქნება შენარჩუნებული.

2. მთელი რიცხვების e და d განსაზღვრისათვის გამოიყენება n-ის მნიშვნელობა. კერძოდ უნდა განისაზღვროს ე.წ. კარმაიკლის ფუნქცია (Carmichael Totien Function) ʎ(n)-ის მნიშვნელობა n-თვის. ჩვენი შემთხვევისთვის როცა n=p·q, სადაც p და q მარტივი რიცხვებია, ეს ფუნქცია ადვილად გამოითვლება,

$$ʎ(n) = LCM(p-1, q-1),$$

სადაც LCM (Least Common Multiple) p-1 და q-1 რიცხვების უმცირესი საერთო ჯერადია (უ.ს.ჯ.). დიდი რიცხვების შემთხვევაში უ.ს.ჯ.-ის გამოთვლისათვის გამოიყენება სათანადო კალკულატორი, აგებული ევკლიდეს ალგორითმზე.

მას შემდეგ, რაც ნაპოვნია ʎ (n), შეიძლება განისაზღვროს e და d. e შეიძლება იყოს ნებისმიერი რიცხვი, რომელიც აკმაყოფილებს პირობებს

$$e < ʎ(n) \text{ და } e \text{ და } ʎ(n) \text{ ურთიერთმარტივია.}$$

რაც შეეხება d-ს იგი არის e-ს მულტიპლიკაციური ინვერსია მოდულით ʎ (n), ანუ

$$e \cdot d = 1 \bmod ʎ(n)$$

ამგვარად განსაზღვრული n, e და d პირობებში ღია გასაღებს წარმოადგენს წყვილი

$$(e, n),$$

ხოლო დახურულ გასაღებს წყვილი (d, n).

ცხადია, p, q და ʎ (n) საიდუმლო რიცხვებია, რომლებიც კრიპტოსისტემის შედგენის შემდეგ ნადგურდებიან (ან საიდუმლოდ ინახებიან ყოველი შემთხვევისთვის).

განვიხილოთ მაგალითები

მაგალითი 1.

ვთქვათ p-=3 და q=11 ანუ **n=33.**

ʎ (n) = LCM (p-1, q-1) = LCM (2, 10)=10, ანუ ʎ (33) = 10

ავიღოთ e=3, რადგან 3<10 და 3 და 10 ურთიერთმარტივია.

გამოვთვალოთ d. რადგან 3·d=1mod10, გვექნება **d=7.**

ამგვარად, ღია გასაღები იქნება.

(3,33)

ხოლო დახურული გასაღები იქნება.

(7, 33)

მაგალითი 2. განვიხილოთ მაგალითი საშუალო დიაპაზონის მონაცემებით, როცა ხელით გამოთვლები პრაქტიკულად აზრს კარგავს. RSA კრიპტოსისტემის შექმნა გულისხმობს შემდეგ ნაბიჯებს.

1. ავიღოთ ორი მარტივი რიცხვი p=101, q=199.

ამ რიცხვების სიმარტივეზე შესამოწმებლად გამოვიყენოთ primality test online calculator.

2. ვიპოვოთ n=101·199 = 20099, **n=20099**

ამ ნამრავლის გამოსათვლელად გამოვიყენოთ Big Numbers Miltiplication Calculator

3. ვიპოვოთ კარმაიკლის ფუნქციის მნიშვნელობა: ʎ (n)= ʎ (20099)= LCM (100, 198) ʎ **(n)=9900**. ვისარგებლოთ Online Carmichael Function Calculator.

4. ავიღო **e=29** და შევამოწმოთ e და ʎ (n) ურთიერთსიმარტივეზე. ვისარგებლოთ კალკულატორით Coprime Calculator.

5. ვიპოვოთ e-ს მოდულური მულტიპლიკაციური ინვერსია $29d \equiv 1 \bmod 9900$. ვისარგებლოთ კალკულატორით Modular Multiplicative Inverse Calculator. D=7169.

6. დავშიფროთ ASCII ასო, რომლის ათობითი ნომერია 49. გამოვთვალოთ შიფრი m=49-თვის. გვექნება $c=49^{29} \bmod 20099$. ვისარგებლოთ კალკულატორით Modular exponentiation. C=16540.

7. მოვახდინოთ დეშიფრაცია. $m \equiv c^d \bmod 20099$. ვისარგებლოთ Modular exponentiation კალკულატორით. **m=49**.

8. ამგვარად, შექმნილი RSA კრიპტოგრაფული სისტემის ღია გასაღებია (e, n) = (29, 20099), ხოლო დახურული (7169,20099).

მაგალითი 2 წარმოდგენას გვაძლევს RSA კრიპტოსისტემასთან დაკავშირებულ გამოთვლებზე. მაგრამ მხედველობაში უნდა იქნეს მიღებული, რომ რეალური სისტემები გაცილებით დიდ რიცხვებთან არის დაკავშირებული და სწორედ ეს უზრუნველყოფს RSA სისტემის საიმედოობას.

3.2. RSA კრიპტოსისტემის გამოყენების მაგალითი

მაგალითისათვის განვიხილოთ კომუნიკაცია ელისსა (Alice, A) და ბობს (Bob, B) შორის, როცა კომუნიკაციის წამომწყებია A. კომუნიკაციისათვის B-ს სჭირდება ბობის ღია გასაღები გზავნილის დასაშიფრად.

შევადგინოთ B-ს ღია და დახურული გასაღებები. ამისათვის:

1. ავირჩიოთ ორი მარტივი რიცხვი p=13 და q=7.

2. გამოვთვალოთ n=p·q=13·7=91. შემდგომში რიცხვი n=91 შიფრაცია/დეშიფრაციის ალგორითმებში გამოიყენება როგორც მოდული (Modules), ხოლო 91-ის ორობით წარმოდგენაში ბიტების რაოდენობა, როგორც შიფრის გასაღების (key) სიდიდე. ჩვენს შემთხვევაში, რადგან 6y< 91 < 128, შიფრის სიდიდე იქნება key=7.

3. გამოვთვალოთ უმცირესი საერთო ჯერადი Z= LCM (p-1, q-1) = LCM (1296) = 12. რიცხვი Z=12 გამოიყენება შიფრაციის ექსპონენტის e და დეშიფრაციის ექსპონენტის d გამოთვლაში.

4. ავირჩიოთ e-ს მნიშვნელობა, რომელიც ურთიერთმარტივი იქნება, Z-თან არ ექნება საერთო გამყოფი Z-თან და ნაკლებია Z-ზე. ასეთად გამოდგება e=5, რადგან 5 და 12 საერთო გამყოფი არ აქვთ. e გამოიყენება როგორც ღია გასაღების კომპონენტი.

5. გამოვთვალოთ დეშიფრაციის ექსპონენტა d. რადგან d არის e-ს მოდულური მულტი-პლიკაციური ინვერსია მოდულით z, ამიტომ (d·e)mod12=1, ანუ (d·5)mod12=1. ამ განტოლებას აკმაყოფილებს d=5.

ამგვარად ბობის ღია გასაღები იქნება (e, n)=(5, 91), ხოლო დახურული (d, n) = (5, 91).

დავუშვათ, რომ ელისი აგზავნის ტექსტს ASCII კოდის მაღალი რეგისტრების სიმბოლოების გამოყენებით, რომელთაც შეესაბამებათ ათობითი რიცხვები (ASCII კოდში რიგითი ნომერი) როგორც ქვემო სურათზეა ნაჩვენები

A	B	C	D	E	F	G	H	I	J	K	L	M
65	66	67	68	69	70	71	72	73	74	75	76	77
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
78	79	80	81	82	83	84	85	86	87	88	89	90

აქ A-ს შეესაბამება რიცხვი 65, B-ს შეესაბამება 66 და ა.შ.

დავუშვათ აგრეთვე, რომ ელისის გზავნილია M = HELLO. ამისათვის საჭიროა ეს გზავნილი წარმოდგეს, როგორც რიცხვი m, მაგრამ მეორე მხრივ მოითხოვება, რომ კმაყოფილდებოდეს უტოლობა 0 ≤ m < n, ანუ m <91. ამ მოთხოვნას აკმაყოფილებს გზავნილის თითოეული სიმბოლო, ხოლო რამდენიმე სიმბოლოს ერთად წარმოდგენა n-ის გაცილებით დიდ მნიშვნელობას მოითხოვს. ამიტომ, პრაქტიკულად გზავნილი

“HELLO” უნდა წარმოდგეს გზავნილების მიმდევრობად “H”, “E”, “L”, “L”, “O”, და წარმოდგება რიცხვით. რომელიც მისი რიგითი ნომრის ტოლია.

ამგვარად, ელისი ახდენს გზავნილის თითოეული სიმბოლოს შიფრაციას ბობის ღია გასაღებით (e, n) = (5, 91), როგორც სურათზეა ნაჩვენები.

სიმბოლო	H	E	L		L	O	ელისის მხარე
სიმბოლოს ნომერი, m	72	69	76		76	79	
შიფრი, $c=m^e mdn$	11	62	20		20	53	

განვიხილოთ, მაგალითად სიმბოლო H. მისთვის გვაქვს m=72, ხოლო ამ სიმბოლოს შიფრი არის $c=m^e modn = 72^5 mod91=11$. სიმბოლო E-სთვის გვექნება $c=m^e modn = 69^5 mod91=62$ და ა.შ. (გამოთვლები სრულდება მოდულირი ახარისხების online კალკულაციით).

ამგვარად, ელისი ბობს უგზავნის ასოების შიფრების კრებულს 11, 62, 20, 20, 53.

ბობი, გზავნილის მიღების შემდეგ ახდენს მათ დეშიფრაციას თავისი დახურული გასაღების (d, n) = (5, 91) გამოყენებით, როგორც ეს სურათზეა ნაჩვენები:

მიღებული შიფრები, c	11	62	20	20	53	ბობის მხარე
დეშიფრაცია $m=c^d modn$	72	69	76	76	79	
სიმბოლოები	H	E	L	L	O	

აქ გამოთვლები წარმოებს ფორმულით $m=c^d modn$.

შიფრ 11-თვის გვექნება $m=11^5 mod91= 72$ (ანუ H), 62-თვის გვექნება $m=62^5 mod91= 69$ (ანუ E) და ა.შ.

ცხადია m-ის შესაძლო მნიშვნელობა m<n. მაგალითად, თუ შიფრის გასაღები n იქნება 1024 ბიტიანი, მაშინ რამდენიმე სიმბოლოს ნომრები შეგვეძლო წარმოგვედგინა როგორც ერთი 6 თანრიგიანი ათობითი რიცხვი m, რაც გაართულებს გამოთვლებს და გაამარტივებდა პროცედურებს. საერთოდ უნდა ითქვას, რომ ასიმეტრიული კრიპტოსისტემები რომელთა მაგალითი RSA, გაცილებით ნელმოქმედებია, ვიდრე კრიპტოსისტემები სიმეტრიული გასაღებით, როგორიცაა, მაგალითად, AES. RSA კრიპტოსისტემის ერთ-ერთ ღირსებად უნდა ჩაითვალოს, რომ იგი მნიშვნელოვნად ამარტივებს გასაღებების მართვას.

RSA კრიპტოსისტემის განხილვისას თავიდანვე აღინიშნა, რომ შიფრაცია/ დეშიფრაციის ალგორითმი დაფუძნებულია ტოლობაზე

$$(m^e)^d \equiv 1 \ modn,$$

სადაც m მონაცემია (ღია ტექსტია), e შიფრაციის ექსპონენტაა, d - დეშიფრაციის ექსპონენტა, ხოლო n მოდულია. ეს ტოლობა შეიძლება გადავწეროთ შემდეგნაირად:

$$(m^e)^d = m^{e \cdot d} = (m^d)^e,$$

რაც ნიშნავს, რომ შეგვიძლია მოვიქცეთ სხვანაირად, კერძოდ, შეიძლება მოვახდინოთ შიფრაცია საიდუმლო გასაღებით, ანუ ვიპოვოთ $m = c^e$. ეს მიდგომა გამოყენებას პოულობს აუთენტიკაციისა და ციფრული ხელმოწერის ალგორითმებში.

§4. ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფია

ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფია (Elliptic Curve Cryptography), RSA კრიპტოგრაფიის პარალელურად, ფართო გამოყენებას პოულობს კრიპტოგრაფიის სხვადასხვა, განსაკუთრებით ელექტრონული ხელმოწერის ამოცანებში. ისეთ სისტემებში, როგორიცაა Bitioin, Ethhereum და ბევრი სხვა, ხელმოწერა ხორციელდება ალგორითმით ECDSA (Elliptic Curve Digital Singanture Algorithm, ელიპტიკურ წირზე დაფუძნებული ხელისმოწერის ალგორითმი), რომელიც RSA სისტემასთან შედარებისას იყენებს გაცილებით ნაკლები ზომის ღია და დახურულ გასაღებებს და ხასიათდება არანაკლები საიმედოობით.

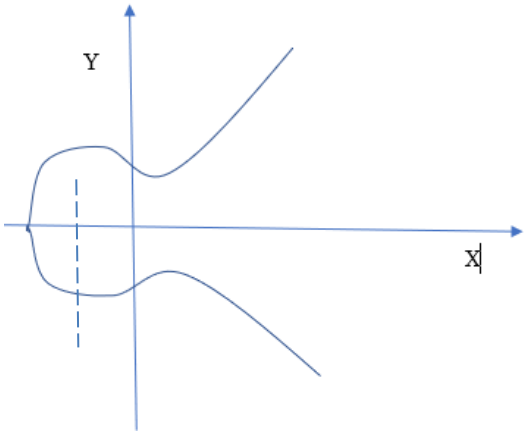
4.1. ელიპტიკური წირი ნამდვილ რიცხვთა სიმრავლეზე

ელიპტიკური წირები შეიძლება წარმოდგენილ იქნეს სხვადასხვა ფორმით თუმცა კრიპტოგრაფიაში ძირითადად გამოიყენება წირი ვეიერშტრასის ნორმალური ფორმით (Weierstrass Normal Form), რომელიც წარმოადგენს (xoy) სიბრტყეზე მოცემულ მესამე ხარისხის წირს, განსაზღვრულს ტოლობით $y^2 = x^3 + ax + b$ და $D = 4a^3 + 27 \cdot b^2 \neq 0$ უტოლობით

ამ გამოსახულებებში x, y, a, b ნამდვილი რიცხვებია და წირი შეიძლება განვიხილოთ, როგორც წერტილების სიმრავლე

$$S = \{x, y \in R^2 \mid y^2 = x^3 + a \cdot x + b,\ 4 \cdot a^3 + 27 \cdot b^2 \neq 0\}$$

ნახ. 222-1-ზე ნაჩვენებია ელიპტიკური წირის გრაფიკები a და b კოეფიციენტების (პარამეტრების) სხვადასხვა მნიშვნელობებისთვის.



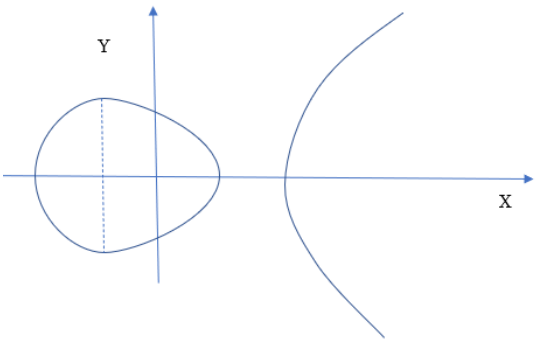
$y^2 = x^2 - x + 1$

a=-1, b=1

$D = 4 * a^3 + 27 * b^2 = 4 * (1)^3 + 27 * 1^2 = 23$

D>0 (ვალიდური წირი)

ა)



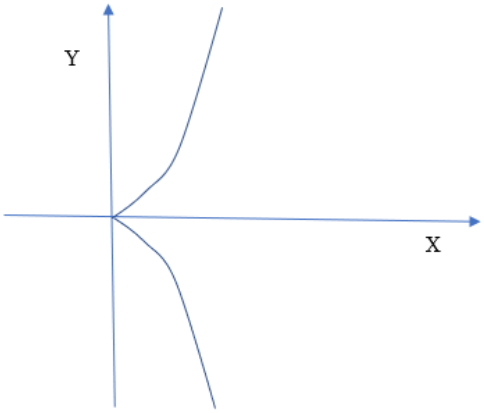
$y^2 = x^3 - 3x + 1$

a=-3, b=1

$D = 4 * a^3 + 27 * b^2 = 4 * (-3)^3 + 27 * 1^2 = -81$

D<0 (ვალიდური წირი)

ბ)



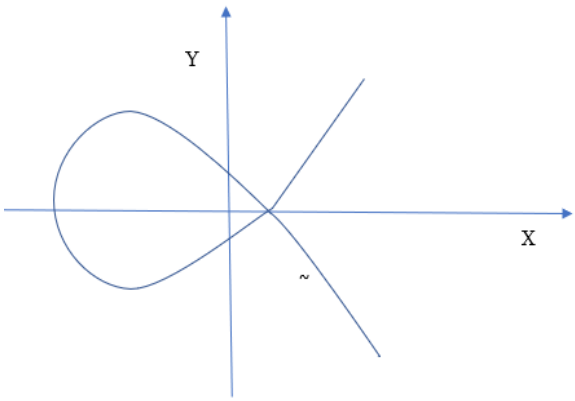
$y^2 = x^3$

a=0, b=0

$D = 4 * 0 + 27 * 0 = 0$

D=0 (სინგულარული წირი)

გ)



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

a=-3, b=2

$D = 4 * (-1)^3 + 27 * 4 = 0$

D=0 (სინგულარული წირი)

დ)

სურათი 1

როგორც ნახაზები გვიჩვენებენ, წირების გრაფიკები სიმეტრიული არიან x ღერძის მიმართ. მართლაც, რადგან $y^2 = x^3 + ax + b$, ამიტომ $y \pm \sqrt{x^3 + ax + b}$. რომ ვიპოვოთ

წირის (x, y) წერტილის ინვერსიული წერტილი, საკმარისია y კოორდინატა, შევცვალოთ -y კოორდინატით.

P, Q და R წერტილების ინვერსიული წერტილებია -P, -Q და -R, რომლებიც წარმოადგენენ საწყისი წერტილების სარკისებურ ასახვას x ღერძის მიმართ. წირის თითოეული შტო x კოორდინატას ზრდასთან შემთხვევაში $D < 0$ და შესაბამისი წირი ორელემენტიანია და ვარგისია კრიპტოგრაფიული გამოყენებისთვის. რაც შეეხება შემთხვევას (გ), აქ $D = 0$ და წირი უვარგისია, რადგან (0, 0) წერტილში მხების ცალსახად გატარება შეუძლებელია და ეს წერტილი წარმოადგენს განსაკუთრებულ, სინგულარულ წერტილს, რის გამოც თვით წირი წარმოადგენს სინგულარულს და გამოუყენებადია. ანალოგიური შემთხვევა გვაქვს დ) შემთხვევაში. აქაც დისკრიმინანტი $D = 0$ და სინგულარობა გამოიხატება იმით, რომ შტოები იკვეთებიან და (1, 0) წერტილში წირისადმი მხების ცალსახად გატარება შეუძლებელია.

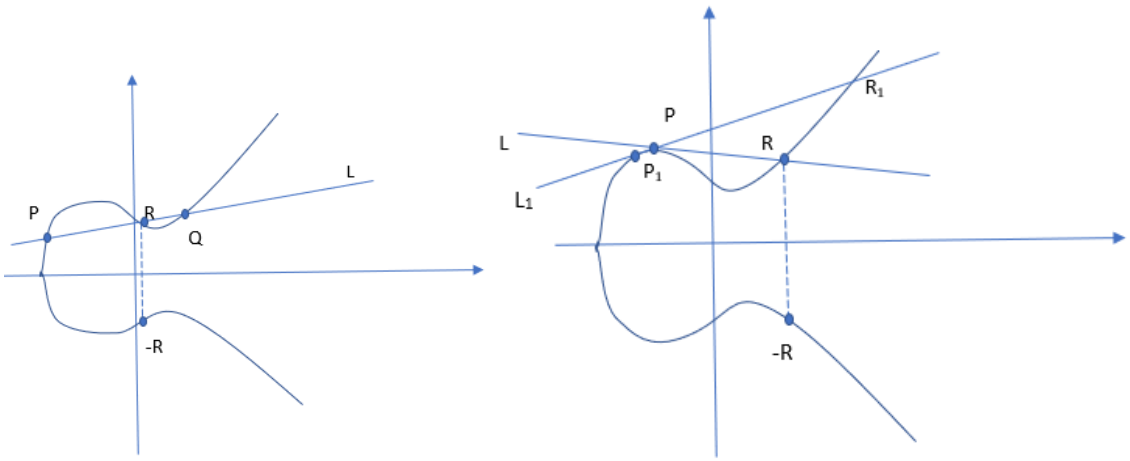
ზემოთ თქმული გვიჩვენებს, რომ ელიპტიკური წირის პრაქტიკულ გამოყენებას წინ უნდა უსწრებდეს წირის შემოწმება ვალიდურობაზე მისი შესაძლო სინგულარობის გამოვლენის მიზნით.

4.1.1. ბინარული ოპერაცია – წერტილების შეკრების გეომეტრიული და ალგებრული მეთოდები

ელიპტიკურ წირზე დაფუძნებულ კრიპტოგრაფიაში ძირითადია წირის ორი წერტილის შეკრების ოპერაცია, რომელიც აღინიშნება + (შეკრების) სიმბოლოთი, თუმცა რადიკალურად განსხვავდება სხვა კონტექსტებში გამოყენებული შეკრების ოპერაციებისაგან.

განვიხილოთ შეკრების ოპერაცია სურათი 2 წარმოდგენილი $y^2 = x^3 - 3x + 3$ წირის მაგალითზე. აქ მოცემულია ორი წერტილი P და Q და საჭიროა ვიპოვოთ მათი ჯამის შესაბამისი წერტილი, რომელიც აგრეთვე ამ წირს ეკუთვნის. ამისათვის უნდა ჩავატაროთ შემდეგი ოპერაციები:

1. P და Q წერტილებზე გავატაროთ წრფე, რომელიც წირს გადაკვეთს R წერტილში.
2. ვიპოვოთ R გადაკვეთის წერტილის ინვერსიული -R წერტილი (ეს შესაძლებელია, რადგან ელიპტიკური წირი სიმეტრიულია x ღერძის მიმართ).
3. მიღებული -R წერტილი წარმოადგენს P და Q წერტილების ჯამს, ანუ $P + Q = -R$.



P+Q+R=0

P+Q=R

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

ა)

P+P =-R

2P=-R

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

ბ)

სურათი 2

აქ განხილული იყო ორი განსხვავებული P და Q წერტილების შეკრების ოპერაცია. ამავე დროს, ხშირად წარმოიქმნება წერტილის თავის თავთან შეკრების ოპერაცია P+P. ეს შემთხვევაა განხილული სურათი 2 ბ-ზე. წარმოვადგინოთ P წერტილთან ახლოს მდებარე P_1 წერტილი და მოვახდინოთ მათი შეკრება. გავატაროთ მათზე მკვეთი L_1, რაც მოგვცემს R_1 წერტილს. ეხლა მივასწრაფოთ P_1 P -კენ, რის შედეგად მკვეთი L_1 მიუახლოვდება P წერტილში გატარებულ L მხებს და საბოლოოდ დაემთხვევა მას. შედეგად მივიღებთ წერტილს R და მის ინვერსიას -R. ამგვარად, P+P ჯამის საპოვნელად საჭიროა P-ზე გავატაროთ მხები, ვიპოვოთ მისი გადაკვეთა წირთან და მოვახერთად უსასრულოდ იზრდება. ყველა $y^2 = x^3 + ax + b$ ტოლობა არ განსაზღვრავს კრიპტოგრაფიული გამოყენებისთვის ვარგის წირს. იმისათვის, რომ წირი ვარგისი ანუ ვალიდური იყოს, საჭიროა $D=4\cdot a^3+27\cdot b^2$ დისკრიმინანტის რიცხვითი მნიშვნელობა განსხვავებული იყოს ნულისაგან. მაგალითად ზემოთ მოყვანილი ნახაზის ა) შემთხვევაში $D = 23 > 0$ და წირი კრიპტოგრაფიისთვის ვარგისია. ასევე ბ) დინოთ გადაკვეთის წერტილის ინვერსია. ეს მოგვცემს P+P=-R. სხვანაირად შეიძლება ითქვას, რომ P+P=2*P, ანუ, წერტილის თავისთავთან შეკრება იწვევს მისი მნიშვნელობის გაორმაგებას.

ზემოთ განხილული ორი წერტილის შეკრების გეომეტრიული მეთოდი ხასიათდება თვალსაჩინოებით, მაგრამ მისი კომპიუტერული რეალიზაცია მოითხოვს ალგორითმს, რომელიც დაფუძნებულია წერტილების ალგებრულ შეკრებაზე. ამ შემთხვევაში საჭირო ხდება ოპერაციების წარმოება წერტილების კოორდინატებზე. რომ შევკრიბოთ ორი P (x_P, y_P) და Q (x_Q, y_Q) წერტილი, პირველ რიგში უნდა შედგეს ამ ორ წერტილზე გამავალი წრფის განტოლება, რომელიც ზოგადად გამოისახება წრფივი $c \cdot x + d \cdot y - e = 0$ განტოლებით. ამის შემდეგ უნდა ვიპოვოთ ამ წრფის $y^2 = x^3 + ax + b$ წირთან გადაკვეთის R (x_R, y_R) წერტილი, მოვახდინოთ მისი ინვერსია x ღერძის მიმართ, რაც მოგვცემს წერტილს -R (x_R, $-y_R$). ამ ოპერაციების ჩატარება მოითხოვს გამოთვლებს, რაც დაკავშირებულია კუბური განტოლების ფესვების პოვნასთან, რაც აღწერისათვის საკმაოდ მოცულობითი და რთულია და ამ მიზეზით აქ არ განვიხილავთ.

4.1.2. წერტილის სკალარზე გამრავლება

ელიპტიკურ წირებზე დაფუძნებულ კრიპტოგრაფიაში ერთ-ერთი მთავარი ოპერაციაა წერტილის გამრავლება სკალარზე, $n \cdot P$, სადაც n სკალარული, ამ შემთხვევაში ნატურალური რიცხვია, ხოლო P ელიპტიკურ წირზე მდებარე წერტილია. თუ დავეყრდნობით ტოლობას

$$n \cdot P = \underbrace{P + P + P \ldots + P}_{n \text{ შესაკრები}} p$$

მაშინ სკალარული ნამრავლის გამოთვლა დაიყვანება ორი წერტილის შეკრების მეთოდის გამოყენებაზე შემდეგი მიმდევრობით

$$P,\ P+P=2P,\ 2P+P=3P,\ 3P+P=4P,\ \ldots$$

ამგვარად n ნაბიჯიანი შეკრების შემდეგ, მიიღება სკალურული ნამრავლი $n \cdot P$.

აღნიშნული მეთოდის ძირითადი ნაკლია მისი დაბალი სწრაფქმედება n-ის დიდი მნიშვნელობისთვის. მაგალითად, პრაქტიკაში ხშირად გამოყენებადი შემთხვევაა, როცა n გამოისახება 256 ბიტიანი რიცხვით და საჭირო ხდება უფრო სწრაფი ალგორითმების დამუშავება. ერთ-ერთ ასეთ ალგორითმს წარმოადგენს „გააორმაგე და შეკრიბე“, რაც დაფუძნებულია n-ის ათვლის ორობით სისტემაში წარმოდგენაზე. განვიხილოთ ეს მეთოდი კონკრეტულ მაგალითზე. ვთქვათ n=91, რასაც შეესაბამება ორობითი რიცხვი $b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0 = 011011$, ანუ

$$91 = 1 \cdot 2^6 - 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

ამგვარად, P წერტილის 91-ზე სკალარული ნამრავლისათვის გვექნება

$91 \cdot P = 1 \cdot 2^6 \cdot P - 0 \cdot 2^5 \cdot P + 1 \cdot 2^4 \cdot P + 1 \cdot 2^3 \cdot P + 0 \cdot 2^2 \cdot P + 1 \cdot 2^1 \cdot P + 1 \cdot 2^0 \cdot P$

ამ სქემის შესაკრებები შეიცავენ მამრავლებს $2^6 \cdot P$, $2^5 \cdot P$, $2^4 \cdot P$, $2^3 \cdot P$, $2^2 \cdot P$, $2^1 \cdot P$, $2^0 \cdot P$, სადაც თითოეული მათგანი მიიღება წინმსწრების გაორმაგებით. მაგალითად, გამოითვლება როგორც $2^3 \cdot P + 2^3 \cdot P$, ანუ $2^3 \cdot P$ წერტილის თავისთავთან შეკრებით: $2^4 \cdot P = 2^3 \cdot P + 2^3 \cdot P = 2^4 \cdot P$. გარდა ამისა, სკალარული ნამრავლი მიიღება იმ წევრების შეკრებით, რომელთა შესაბამისი ბიტი ერთის ტოლია. თქმულის გათვალისწინებით შეიძლება ჩამოყალიბდეს შემდეგი იტერაციული ალგორითმი:

S იყოს ცვლადი, რომელიც შეიცავს გამოთვლების შუალედურ შედეგებს. დასაწყისში S=0. D იყოს ცვლადი, რომელიც ინახავს გაორმაგების შუალედურ შედეგებს. თავიდან D=P=2 გამოთვლები ვაწარმოოთ ბიტების მნიშვნელობათა გათვალისწინებით, მარჯვნიდან მარცხნივ.

ნაბიჯი 0. $b_0=1$.

შეკრება: S: = S+D, გვექნება $S = 0+P = P$ (ანუ $1 \cdot P$ ანუ $2^0 \cdot P$).

გაორმაგება: D: = D+P, გვექნება $D = 2^0 \cdot P + 2^0 \cdot P = 2^1 \cdot P$).

ნაბიჯი 1. $b_1=1$.

შეკრება: S: = D+ S, გვექნება $S = 2^1 \cdot P + 2^0 \cdot P$

გაორმაგება: D: = D+P, გვექნება $D = 2^1 \cdot P + 2^1 \cdot P = 2^2 \cdot P$.

ნაბიჯი 2. $b_2=0$.

გაორმაგება: D: = D+ D, $D = 2^2 \cdot P + 2^2 \cdot P = 2^3 \cdot P$.

ნაბიჯი 3. $b_3=1$.

შეკრება: S: = D+ S, $S = 2^3 \cdot P + 2^1 \cdot P + 2^0 \cdot P$

გაორმაგება: D: = D+D, $D = 2^3 \cdot P + 2^3 \cdot P = 2^4 \cdot P$.

ნაბიჯი 4. $b_4=1$.

შეკრება: S: = D+ S, $S = 2^4 \cdot P + 2^3 \cdot P + 2^1 \cdot P + 2^0 \cdot P$

გაორმაგება: D: = D+D, $D = 2^4 \cdot P + 2^4 \cdot P = 2^5 \cdot P$.

ნაბიჯი 5. $b_4=0$.

გაორმაგება: D: = D+D, $D = 2^5 \cdot P + 2^5 \cdot P = 2^6 \cdot P$.

ნაბიჯი 6. $b_6=1$.

შეკრება: $S: = D+ S, \quad S = 2^6 \cdot P + 2^4 \cdot P + 2^3 \cdot P + 2^1 \cdot P + 2^0 \cdot P$

საბოლოოდ გვაქვს $S = 91 \cdot P = 2^6 \cdot P + 2^4 \cdot P + 2^3 \cdot P + 2^1 \cdot P + 2^0 \cdot P$

სკალარული $n \cdot P$ ნამრავლის გამოთვლის ზემოთ მოყვანილი ორი მეთოდიდან პირველი მოითხოვს შეკრების ოპერაციის ჩატარებას, ამიტომ მისი სწრაფქმედება $0(n)$ რიგისაა. რაც შეეხება მეორე მეთოდს, სადაც ხდება n-ის წარმოდგენა ორობითი რიცხვით, რომელშიაც ბიტების რაოდენობა განისაზღვრება $\lg n$, მასში შეკრებათა რაოდენობა $0(\lg n)$ რიგისაა და ამგვარად იგი გაცილებით სწრაფქმედია პირველთან შედარებით.

4.2. ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფია სასრული ველის გამოყენებით

წინა პარაგრაფში განხილული იყო ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფია, როცა წირი განსაზღვრულია ნამდვილ რიცხვთა (xoy) სიბრტყეზე და წირის შემადგენელი წერტილების x და y კოორდინატები ღებულობდნენ მნიშვნელობებს ნამდვილ რიცხვთა უსასრულო სიმრავლიდან. ამგვარად, მათემატიკური ოპერაციების შედეგები არ იყო შემოსაზღვრული და იცვლებოდნენ ფართო დიაპაზონში, რაც ართულებდა მონაცემების დამახსოვრებას და მათზე ოპერაციების შესრულებას და, ამგვარად, იწვევდა ალგორითმების დაბალ ეფექტურობას. გამოთვლების ეფექტურობის გაუმჯობესების მიზნით დამუშავებულ იქნა ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფიის მეთოდები, როცა წირი განმარტებულია სასრული სიმრავლეზე. ეს ნიშნავს, რომ წირის შემადგენელი წერტილების x და y კოორდინატები ღებულობენ მნიშვნელობებს სასრული სიმრავლიდან და მათზე შესრულებული ოპერაციების შედეგები ეკუთვნიან ამავე სასრული სიმრავლეს.

4.2.1. ალგებრული სტრუქტურა – მთელი რიცხვების ველი მოდულით P

სასრული ველი წარმოადგენს სიმრავლეს სასრული რაოდენობის ელემენტებით. ასეთი ველის მაგალითს წარმოადგენს მთელი რიცხვების სიმრავლე მოდულით P, სადაც P მარტივი რიცხვია. მთელი რიცხვები მოდულით P ქმნიან სიმრავლეს {0, 1, 2, ... , P-1}, რომელზედაც განმარტებულია შეკრების (+), გამრავლების (·) და სხვა ოპერაციები, რაც წარმოებს მოდულური არითმეტიკის წესების შესაბამისად. ტექნიკურ ლიტერატურაში სასრული ველი უფრო ხშირად აღინიშნება როგორც GF(P) (გალუას ველი მოდულით P) და Fp, ხოლო ქვემოთ გამოყენებულ იქნება სახელი Fp, როგორც კომპაქტური ჩანაწერი.

განვიხილოთ მოდულური გამოთვლების მაგალითები F_7 ველისთვის, სადაც გამოთვლები წარმოებს $p=7$ მოდულით, რის შედეგად არითმეტიკული ოპერაციის

შედეგი წარმოადგენს ერთ-ერთ მთელ რიცხვს სიმრავლიდან {0, 1, 2, 3, 4, 5, 6}. ოპერაციის შესრულების წესები მოყვანილია სათანადო დამტკიცების გარეშე.

4.2.2. მოდულური შეკრება

ორი დადებითი A და B რიცხვების შეკრება მოდულით P სრულდება შემდეგი წესის შესაბამისად.

$$(A+B) \bmod p = (A \bmod p + B \bmod p) \bmod p$$

მაგალითად, თუ A = 15 და B = 8, მოდულური შეკრების ოპერაციისთვის გვექნება,

$$(15 + 8) \bmod 7 = (15 \bmod 7 + 8 \bmod 7) \bmod 7 = (1+1) \bmod 7 = 2 \bmod 7 = 2.$$

4.2.3. მოდულური ინვერსია შეკრების მიხედვით (Additive Inverse)

მთელი A რიცხვის ინვერსიას შეკრების მიხედვით, ანუ მის მოპირდაპირე რიცხვს წარმოადგენს ისეთი მთელი B რიცხვი, რომელიც აკმაყოფილებს ტოლობას

$$(A+B) \bmod 7 = 0.$$

მაგალითად 2-ის მოპირდაპირე რიცხვია 5, რადგან (2+5) mod7=0.

4.2.4. მოდულური გამოკლება

მოდულური გამოკლება ხდება შემდეგი ფორმულის მიხედვით

$$(A-B) \bmod p = (A \bmod p + (-B) \bmod p) \bmod p$$

მაგალითად, (3-7) mod7=(3mod7+(-7) mod7) mod7=(3+0) mod7=3

როგორც ზემოთ მოყვანილი შეკრების ოპერაციის მაგალითი გვიჩვენებს გამოკლების ოპერაცია შეკრების ოპერაციის მსგავსია.

4.2.5. მოდულური გამრავლება

მოდულური გამრავლების ოპერაცია ხორციელდება შემდეგი ტოლობის შესაბამისად

$$(A \cdot B) \bmod C = (A \bmod C \cdot B \bmod C) \bmod C$$

მაგალითად, თუ A=16, B=21 და C=7, გვექნება

$$(16 \cdot 21) \bmod 7 = (16 \bmod 7 \cdot 21 \bmod 7) \bmod 7 = (2 \cdot 0) \bmod 7 = 0 \bmod 7 = 0$$

მართლაც, $(16 \cdot 21) \bmod 7 = 336 \bmod 7 = 0$.

4.2.6. მოდულური ინვერსია გამრავლების მიხედვით

მოდულურ გამოთვლებში არ არის განმარტებული გაყოფის ოპერაცია, მაგრამ გამოიყენება რიცხვის ინვერსიის პოვნის ალგორითმი. ჩვეულებრივ არითმეტიკაში A რიცხვის ინვერსიას წარმოადგენს რიცხვი A^{-1}, რადგან $A \cdot A^{-1} = 1$. მაგალითად, რიცხვ 3-ის ინვერსია არის $3^{-1} = \frac{1}{3}$. რადგან ველის განმსაზღვრელი სიმრავლე არ შეიცავს წილადებს, გამოიყენება მოდულური გამოთვლება: A რიცხვის ინვერსია არის ისეთი A^{-1}. რომლისთვისაც კმაყოფილდება ტოლობა $(A \cdot A^{-1}) \bmod C = 1$.

რიცხვის ინვერსიის პოვნის ლოგიკურად მარტივი მეთოდი მდგომარეობს ველის ყველა ელემენტის მოსინჯვაში, როგორც ეს ქვემოთ მოყვანილ მაგალითშია განხილული. ვთქვათ გვინდა ვიპოვოთ რიცხვი 5-ის ინვერსია მოდულით 7. მოვსინჯოთ {0, 1, 2, 3, 4, 5, 6} სიმრავლეში ელემენტები მიმდევრობით:

$(5 \cdot 0) \bmod 7 = 0$

$(5 \cdot 1) \bmod 7 = 5$

$(5 \cdot 2) \bmod 7 = 3$

$(5 \cdot 3) \bmod 7 = 1$ ← $5^{-1} = 3$. რადგან $(5 \cdot 3) \bmod 7 = 1$

$(5 \cdot 4) \bmod 7 = 6$

$(5 \cdot 5) \bmod 7 = 4$

$(5 \cdot 6) \bmod 7 = 2$

ეხლა განვიხილოთ შემთხვევა, როცა გვინდა ვოპოვოთ 2-ის ინვერსია 2^{-1} მოდულით 6. ჩავატაროთ გამოთვლები ზემოგანხილულის ანალოგიურად:

$(2 \cdot 0) \bmod 6 = 0$

$(2 \cdot 1) \bmod 6 = 2$

$(2 \cdot 2) \bmod 6 = 4$

$(2 \cdot 3) \bmod 6 = 0$

$(2 \cdot 4) \bmod 6 = 2$

$(2 \cdot 5) \bmod 6 = 4$

როგორც ვხედავთ ამ შემთხვევაში სიმრავლე {0, 1, 2, 3, 4, 5} არ შეიცავს ელემენტს, რომელიც წარმოადგენს 2-ის შებრუნებულ ელემენტს. ეს ნიშნავს რომ 2-ს მოდულით 6

შებრუნებული ელემენტი არ გააჩნია. ამის მიზეზია ის, რომ 2 და 6 ურთიერთ მარტივი რიცხვები არაა, მათ აქვთ საერთო თანამამრავლი - მარტივი რიცხვი 2.

ინვერსიული ელემენტის გამოთვლის ზემოთ განხილული მეთოდი ლოგიკურად მარტივია, მაგრამ იგი არაეფექტურია სწრაფქმედების თვალსაზრისით, რადგან P-ს დიდი მნიშვნელობისათვის იგი დიდი მოცულობის გამოთვლებს მოითხოვს. პრაქტიკული განზომილების ამოცანებში, როცა P-ს მნიშვნელობა ასეულებს აღწევს, გამოიყენება **ევკლიდეს გაფართოებული ალგორითმი** (Extended Euclidean algorithm), რომლის აღწერა ადვილად მოსაძიებელია ინტერნეტში.

4.2.7. გაყოფა მოდულით P

ჩვეულებრივ არითმეტიკაში x/y ნიშნავს ორი რიცხვის გაყოფას. Fp ველში ჩანაწერი x/y ნიშნავს $x \cdot y^{-1}$, სადაც y^{-1} არის y-ის მულტიპლიკაციური ინვერსია. ამგვარად, რომ შესრულდეს Fp ველში გაყოფა მოდულით P, საჭიროა ჯერ ვიპოვოთ y-ის ინვერსია y^{-1} და შემდეგ ვიპოვოთ $(x \cdot y^{-1})$ modP. მაგალითად, თუ გვინდა ვიპოვოთ $(4 \cdot 5^{-1})$ mod7 უნდა ვიპოვოთ 5^{-1}. როგორც ზემოთ განხილული მაგალითი გვიჩვენებს $5^{-1} = 3$. ამგვარად გვექნება:

$(4 \cdot 5^{-1})$ mod7 = $(4 \cdot 3)$ mod7 = 12 mod7 = 5.

4.2.8. ელიპტიკური წირი Fp ველში

წინა პარაგრაფში განხილული იყო ელიპტიკური წირები განსაზღვრული ნამდვილ რიცხვთა სიმრავლეზე, სადაც წირი მოცემული იყო ვეიერშტრასის ფორმით და წირის შემადგენელი (x, y) წერტილების სიმრავლე აკმაყოფილებდნენ პირობებს

$\{(x, y) \in R^2 \mid y^2 = x^3 + a \cdot x + b\} \cup \{0\}$

$4a^3 + 27 \cdot b^2 \neq 0$,

სადაც x და y ღებულობენ მნიშვნელობებს ნამდვილ რიცხვთა R სიმრავლიდან, $y^2 = x^3 + a \cdot x + b$ წირის განტოლებაა. 0 უსასრულობაში მდებარე იდეალური წერტილია, ხოლო $4 \cdot a^3 + 27 \cdot b^2 \neq 0$ წირის ვალიდურობის პირობაა. სასრული Fp ველის შემთხვევაში ცვლადები ღებულობენ მნიშვნელობებს სასრულო სიმრავლიდან და გამოთვლები წარმოებს მოდულით P (modP). ამგვარად, ზემოთ მოყვანილი გამოსახულებები მიიღებენ სახეს:

$\{(x, y) \in F^2p \mid y^2 \equiv x^3 + ax + b \ (\text{mod}P)\} \cup \{0\}$,

$4a^3 + 27 \cdot b^2 \not\equiv 0 \pmod{P}$

აქ ≡ კონგრუენტობის სიმბოლოა, რაც ნიშნავს, რომ ორი გამოსახულება კონგრუენტულია, თუ ორთავე მათგანი, modP-თ გამოთვლის შემდგომ, აკმაყოფილებენ მოცემულ თანაფარდობას. მაგალითად,

$12 \equiv 2 \pmod{5}$, რადგან

$12 \bmod 5 = 2 \pmod{5}$

$2 = 2$

რაც შეეხება a და b, ისინი არიან მთელი რიცხვები Fp სიმრავლიდან.

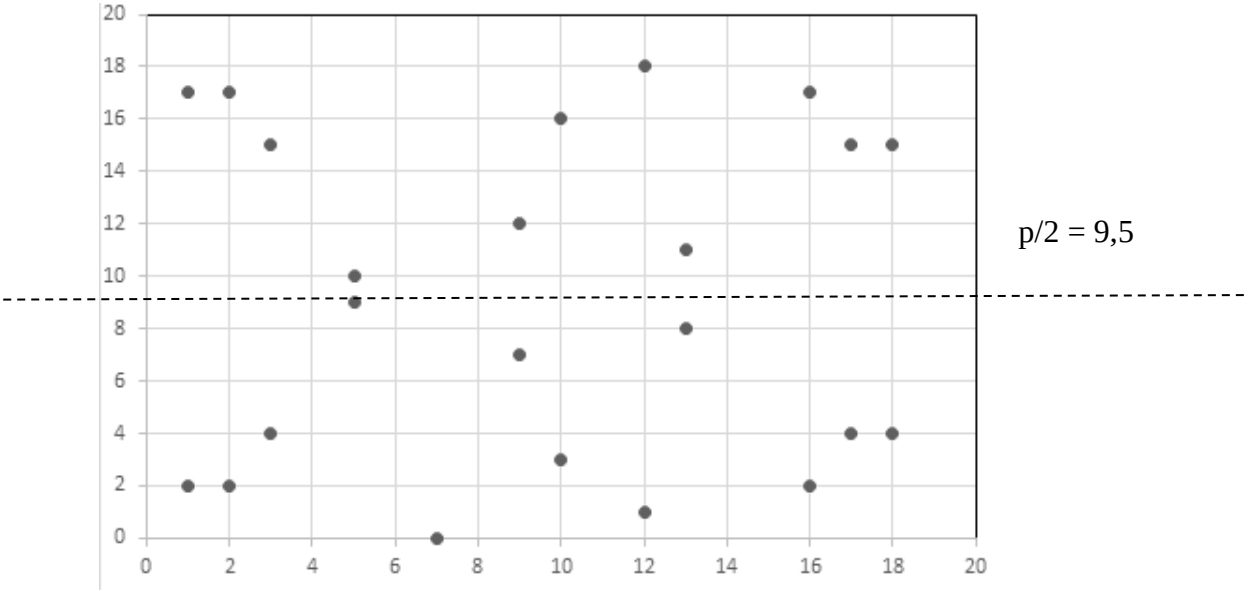
განვიხილოთ ვალიდურია თუ არა ელიპტიკური წირი $y^2 \equiv x^3 + 2x + 1$. ვალიდურობისათვის საჭიროა კმაყოფილდებოდეს უტოლობა

$4 \cdot 2^3 + 27 \cdot 1^2 \not\equiv 0 \pmod{5}$, ე. ი. $(4 \cdot 2^3 + 27 \cdot 1^2) \bmod 5 \not\equiv 0 \bmod 5$,

$(32 + 27) \bmod 5 \not\equiv 0 \bmod 5$, $59 \bmod 5 \not\equiv 0$, $4 \not\equiv 0$.

მოყვანილი გამოთვლები გვიჩვენებენ, რომ წირი $y^2 \equiv x^3 + 2 \cdot x + 1$ არ შეიცავს სინგულარულ ელემენტებს და იგი ვალიდურია კრიპტოგრაფიული გამოყენებისთვის.

ქვემოთ მოყვანილია F_{19} ველის გრაფიკული გამოსახულება. რადგან ველი განსაზღვრულია P=19 მოდულით, ამიტომ როგორც x, ისე y კოორდინატები ღებულობენ მნიშვნელობებს სიმრავლიდან {0, 1, 2, … , 18} და, ამგვარად გვაქვს დისკრეტული (x, y) წერტილების რაოდენობა (P-1) (P-1) = 18 · 18 = 324.



$$y^2 \equiv x^3\text{-}7x + 10 \pmod{19}$$

განვიხილოთ ამ სიმრავლეზე განმარტებული ელიპტიკური ფუნქცია

$$y^2 \equiv x^3\text{-}7x + 10 \pmod{19}$$

ნახაზზე გამოსახულია იმ წერტილების სიმრავლე, რომლებიც აკმაყოფილებენ ამ ტოლობას და, როგორც ვხედავთ მათი რაოდენობა არის 23 წერტილი 324-დან. რადგან

$$y = \pm \sqrt{x^3 - 7x + 10},$$

შესაბამისად, ფიქსირებული x-თვის გვაქვს y-ის ორი მნიშვნელობა, რომლებიც განლაგებული არიან სიმეტრიულად $y = \frac{p}{2}$ ჰორიზონტალური ხაზის მიმართ.

იმისათვის, რომ ელიპტიკურ წირზე შესრულდეს მათემატიკური ოპერაციები, საჭირო ხდება ელიპტიკური წირის შემადგენელ ელემენტების (წერტილების) სიმრავლის განსაზღვრა. ამისათვის საჭირო ხდება ყველა (x, y) წერტილისთვის განისაზღვროს მისი წირისადმი კუთვნილება. მაგალითად, შევამოწმოთ ეკუთვნის თუ არა წერტილი (x, y) = (3, 4) $y^2 \equiv x^3\text{-}7x + 10 \pmod{19}$ წირს. ჩავატაროთ გამოთვლები:

$4^2 \equiv 3^3\text{-}7\cdot 3 + 10 \pmod{19}$

$4^2 \bmod 19 = 16 \bmod 19$

$16 = 16$ (ეკუთვნის)

რაც შეეხება წერტილს (2, 3) გვაქვს

$3^2 \equiv 2^3 - 7\cdot 2 + 11 \pmod{19}$

$9 \bmod 19 = 5 \bmod 19$

$19 \neq 5$ (არ ეკუთვნის).

ეს მეთოდი პრინციპულად ვარგისია, მაგრამ იგი ძალიან არაეფექტური (ნელი) ხდება P-ს დიდი მნიშვნელობისათვის, როცა იგი უტოლდება ათეულებს და ასეულებს და მოითხოვება დიდი მოცულობის გამოთვლები.

4.2.9. წერტილების შეკრება

ნამდვილ რიცხვთა სიმრავლეზე განმარტებული ელიპტიკური წირისათვის, როგორც ეს წინა პარაგრაფში იქნა განხილული, ზოგადად, წრფე ელიპტიკურ წირს კვეთს სამ P, Q, R წერტილებში და ამ წერტილებისათვის ადგილი აქვს ტოლობას

$$P + Q + R = 0,$$

სადაც ორი P და Q წერტილების ჯამისათვის გვაქვს გამოსახულება

$$P + Q = - R$$

რაც შეეხება დისკრეტულ Fp სიმრავლეზე განმარტებულ წრფეს, იგი წარმოადგენს Fp სიმრავლის ისეთი (x, y) წერტილების ერთობლიობას, რომლებიც აკმაყოფილებენ წრფის განტოლებას:

$$a \cdot x + by + c \equiv 0 \pmod{p}.$$

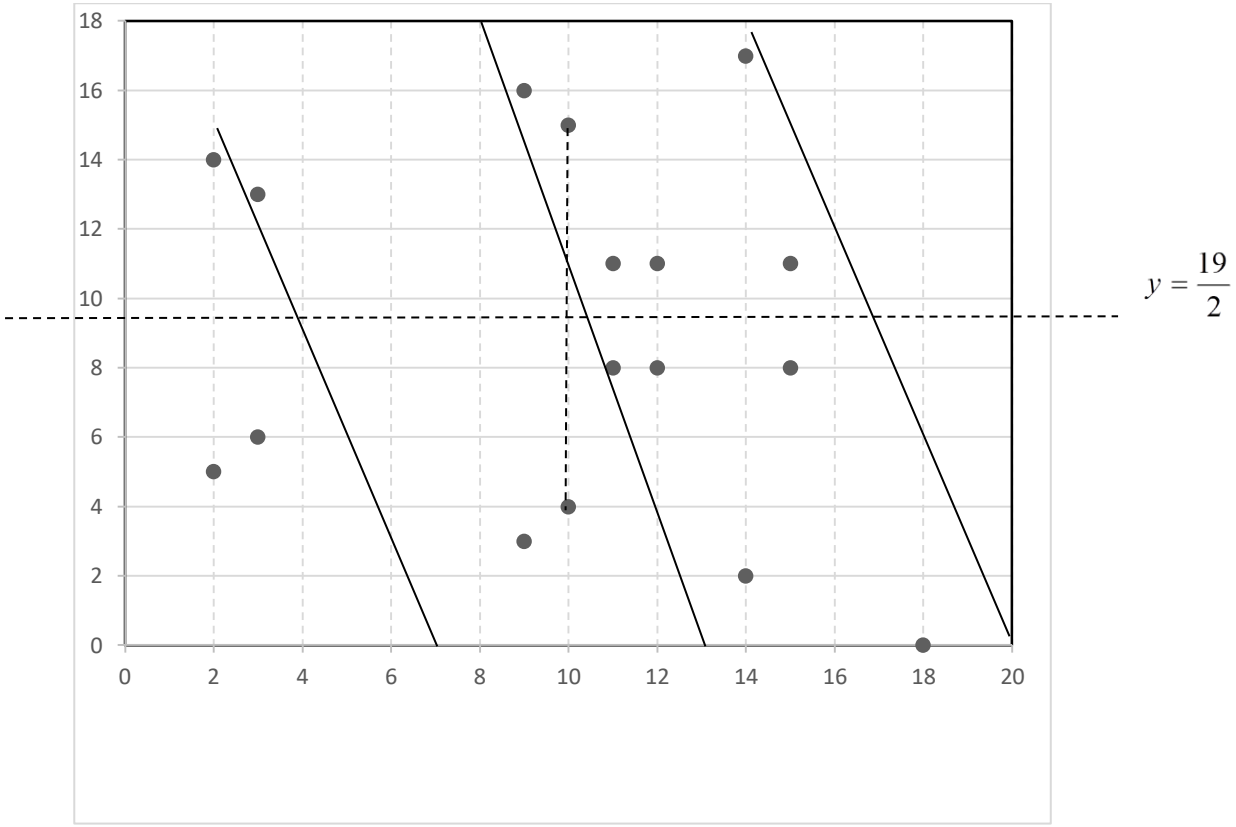
ქვემოთ მოყვანილ ნახაზზე ნაჩვენებია $y^2 \equiv x^3 + 2 \cdot x + 3$ ელიპტიკური წირის წერტილების განლაგება F_{19}-ის შემთხვევაში. როგორც ნახაზი გვიჩვენებს, აქ ხდება $a \cdot x + b \cdot y + c \equiv 0$ წრფის „გადახვევა“ $F_{19} \times F_{19}$ სიმრავლის მიმართ, ეს წირი შეიცავს სამ წერტილს P, Q და R და ადგილი აქვს ტოლობებს:

$$P = (3, 6), Q = (15, 8), R = (10, 4),$$

$$P + Q = - R = (10, - 4 \bmod 19) = 10, 15)$$

(აქ გათვალისწინებულია, რომ R წერტილის სიმეტრიული -R წერტილს აქვს იგივე აბსცისა, რაც R წერტილს, ხოლო მისი ორდინატა R წერტილის ორდინატის მოპირდაპირე რიცხვია).

ზემოთ განხილული ელიპტიკური წირის ორი ელემენტის შეკრების ოპერაციის კომპუტერული რეალიზაცია დაფუძნებულია შეკრების ალგებრულ მეთოდზე, რისი აღწერაც გარკვეულ სირთულეებთან არის დაკავშირებული და აქ არ არის განხილული. დაინტერესებულ მკითხველს ამ საკითხის განხილვა შეუძლია ადვილად მოიპოვოს ანდრია კორბელინის პოსტებში საერთო სათაურით Andrea Corbellini, Elliptic Curve Cryptograhy: a Gentle Introduction.



$y^2 = x^3 - 2x + 3$ ველში F_{19}

$P = (3, 6), Q = (15, 8), R = (10, 4)$

$P + Q = -R = (10, 15)$

4.2.10. ელიპტიკური წირის ჯგუფის რიგი (Group Order)

სასრულო ველზე განმარტებული ელიპტიკური წირი შეიცავს სასრულო რაოდენობის (x, y) წერტილებს, რომლებიც აკმაყოფილებენ წირის განტოლებას

$$Y^2=x^2+a.x+b \ (\text{mod} p)$$

და როგორც x (აბსცისა), ისე y (ორდინატა) ღებულობენ მნიშვნელობებს {0, 1, 2, ... , P-1} სიმრავლიდან.

კრიპტოგრაფიული სისტემის დამუშავებისას საჭიროა ზუსტად განისაზღვროს წირის შემადგენელი წერტილების რაოდენობა N. ეს წერტილები ქმნიან ალგებრულ სტრუქტურას - ჯგუფს (Group) და ჯგუფში შემავალი ელემენტების რაოდენობას N ეწოდება ჯგუფის რიგი.

ელიპტიკური წირის შესაბამისი ჯგუფის რიგის განსაზღვრა შეიძლება ხორციელდებოდეს მარტივი წესით, როცა ყველა (x, y) წერტილისთვის ხდება

შემოწმება, აკმაყოფილებენ თუ არა (x, y) წერტილის კოორდინატები წირის განტოლებას:

$$Y^2 = x^3 + a \cdot x + b \ (\mathrm{mod} p)$$

და ყველა დადებითი შედეგების რაოდენობა იძლევა წირის შესაბამისი ჯგუფის ელემენტების რაოდენობას, ანუ ჯგუფის რიგს N. ამ მეთოდის გამოყენება P-ს დიდი მნიშვნელობისათვის დიდი მოცულობის გამოთვლებს მოითხოვს. ამიტომ P-ს პრაქტიკული მნიშვნელობებისათვის გამოიყენება ე.წ. შუფის ალგორითმი (S'choof's Algorithm), რომელიც განხორციელებულია სხვადასხვა პროგრამირების ენების ბიბლიოთეკებში და აქ არ მოგვყავს გარკვეული სირთულეების გამო.

4.2.11. სკალარული ნამრავლი და ციკლური ქვეჯგუფი

ისევე, როგორც ნამდვილ რიცხვთა სიმრავლის შემთხვევაში Fp ველის შემთხვევაში სკალარული ნამრავლი განისაზღვრება როგორც

$$n \cdot p = \underbrace{p + p + \ldots + p}_{n\text{-ჯერ}}$$

სადაც p ელიპტიკური წირის წერტილი, ხოლო n ნატურალური რიცხვია.

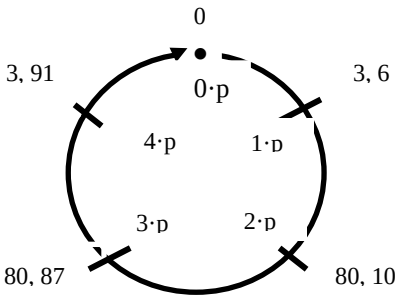
ნამდვილ რიცხვთა სიმრავლეზე განმარტებული წირისათვის წინა პარაგრაფში აღწერილი იყო სკალარული ნამრავლის გამოთვლის ეფექტური ალგორითმი „გააორმაგე და შეკრიბე". იგივე ალგორითმი შეიძლება გამოყენებულ იქნეს Fp ველზე განმარტებული წირისათვის, ოღონდ წერტილების შეკრებისას გამოთვლები უნდა წარმოებდეს მოდულური არითმეტიკის წესების შესაბამისად.

განვიხილოთ სკალარული ნამრავლის თვისებები ანდრეა კორბელინის პოსტიდან [ ].

აქ წირი მოცემულია განტოლებით $y^2 \equiv x^3 + 2 \cdot x + 3 \ (\mathrm{mod}\ 97)$, ხოლო P = (3, 6).

როგორც ქვემოთ მოყვანილი ნახაზი გვიჩვენებს, P წერტილის ნატურალურ რიცხვებზე გამრავლებისას, დაწყებული 0-დან მიიღება მნიშვნელობები:

$0 \cdot P = 0$, $1 \cdot P = (3, 6)$, $2 \cdot P = (80, 10)$, $3 \cdot P = (80, 87)$, $4 \cdot P = (3, 91)$



თუ გავაგრძელებთ გამოთვლებს მომდევნო ნატურალური რიცხვებისთვის, გვექნება:

$5 \cdot p = 0$, $6 \cdot p = (3, 6)$, $7 \cdot p = (80, 10)$, $8 \cdot p = (80, 87)$, $9 \cdot p = (3, 91)$, $10 \cdot p = 0$, $11 \cdot p = (3, 6)$, $12 \cdot p = (80, 10)$ ...

გამოდის, რომ რა რიცხვზეც არ უნდა გავამრავლოთ $p = (3, 6)$ წერტილი, მიიღება ერთ-ერთი ნახაზზე მოცემული ხუთი წერტილიდან, ზოგადად შეიძლება დაიწეროს, რომ

$$n \cdot P = (n \bmod 5)\, P$$

ნებისმიერი ნატურალური n რიცხვისთვის.

თუ P-ს მაგივრად განვიხილავთ წირის სხვა Q წერტილს და გამოვთვლით სკალარულ ნამრავლს $n{\cdot}Q$ ზემოთ განხილულის თანახმად, შეიძლება მივიღოთ სხვა ციკლურად განმეორებადი წერტილების სიმრავლე.

ზოგადად, თუ წირის ყველა ელემენტის სიმრავლე ანუ წირი, ქმნის ჯგუფს ელემენტების რაოდენობით N, მაშინ მისი ნებისმიერი P ელემენტის სკალარული ნამრავლი $n{\cdot}P$ რიცხვებისთვის $n= 0, 1, 2, \ldots$ ქმნის ციკლს, რომლის ელემენტების სიმრავლე წარმოადგენს წირის ელემენტების სიმრავლის ქვესიმრავლეს. რაც მეტია ციკლური ქვეჯგუფის ელემენტების სიმრავლე, მით უფრო ეფექტურია მისი გამოყენება კრიპტოგრაფიაში და ამიტომ წარმოიშობა ისეთი ბაზური P წერტილის პოვნის ამოცანა, რომელიც უზრუნველყოფს ციკლს მაქსიმალური რაოდენობის ელემენტებით.

ამგვარად, თუ წირის შემადგენელი ელემენტების რაოდენობას ვუწოდებთ მის რიგს N, მაშინ წირის ერთ-ერთი P წერტილის ნატურალურ რიცხვებზე გამრავლებისას მიღებულ წერტილთა რაოდენობა n წარმოადგენს P-ს რიგს.

წირის რიგი N და წერტილის რიგი n ერთმანეთთან დაკავშირებულია ლაგრანჟის თეორემით (Lagrange's Theorem): n არის N-ის გამყოფი.

ეს თეორემა საშუალებას გვაძლევს ვიპოვოთ P-ს რიგი ანუ მის მიერ წარმოქმნილი ციკლის ელემენტთა რაოდენობა შემდეგი ალგორითმის გამოყენებით:

1. ვიპოვოთ ელიპტიკური წირის რიგი (წერტილთა რაოდენობა) N შუფის ალგორითმით;
2. ვიპოვით N-ის ყველა გამყოფი;
3. N-ის ყველა გამყოფისთვის გამოვთვალოთ $n \cdot P$;
4. უმცირესი n, რომლისთვისაც $n \cdot P = 0$ იქნება P-ს რიგი (შესაბამისი ციკლის ელემენტების რაოდენობა).

4.2.12. ბაზური წერტილის პოვნა

როგორც ზემოთ ითქვა, ელიპტიკური წირის ნებისმიერი P წერტილი წარმოქმნის ციკლს და ციკლში შემავალი ელემენტების რაოდენობას ციკლის რიგი ეწოდება, ხოლო წერტილი P ამ ციკლის ბაზურ წერტილს წარმოადგენს.

ელიპტიკურ წირზე დაფუძნებულ კრიპტოგრაფიაში Fp ველზე განსაზღვრული წირის შემთხვევაში საჭიროა ვიპოვოთ ისეთი ციკლი (ქვეჯგუფი), რომლის წერტილების რაოდენობა n მარტივი რიცხვია და ამასთან იგი შეიცავს წირის წერტილების მაქსიმალურ რაოდენობას.

ლაგრანჟის თეორემის თანახმად, ციკლში შემავალი წერტილები ქმნიან ქვეჯგუფს, რომლის რიგი ანუ წერტილების რაოდენობა n არის წირის წერტილების საერთო რაოდენობის, N-ის გამყოფი. ამგვარად, $h = N/n$ ყოველთვის მთელი რიცხვია და მას ქვეჯგუფის კოფაქტორი ეწოდება.

რამდენადაც N არის ნებისმიერი n-ის ჯერადი, ამიტომ ადგილი აქვს ტოლობას $N \cdot P = 0$. ეს ტოლობა შეიძლება ჩაიწეროს როგორც

$$N \cdot P = n(h \cdot P) = 0$$

წერტილს $G = h \cdot P$ ეწოდება ქვეჯგუფის გენერატორი, რადგან $n \cdot G$ სკალარული ნამრავლი

$$n \cdot G = \underbrace{G + G + \ldots + G}_{\text{n-ჯერ}}$$

წარმოქმნის ციკლს, რომლის წერტილთა რაოდენობა n მარტივი რიცხვია.

ამგვარად, შეიძლება ჩამოყალიბდეს ბაზური წერტილის პოვნის შემდეგი ალგორითმი:

1. გამოთვალე ელიპტიკური წირის რიგი N შუფის ალგორითმის გამოყენებით;
2. იპოვე N-ის მამრავლები და აირჩიე მათგან n, რომელიც წარმოადგენს უდიდეს მარტივ რიცხვს N-ის გამყოფთა შორის;
3. გამოთვალე კოფაქტორი $h = N/n$;
4. აირჩიე წირის შემთხვევითი წერტილი P;
5. გამოთვალე $G = h \cdot P$;
6. თუ $G = 0$, მაშინ დაუბრუნდი ნაბიჯს 4. წინააღმდეგ შემთხვევაში ნაპოვნია ქვეჯგუფი (ციკლის რიგი n და კოფაქტორი h).

ამგვარად, ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფიული ალგორითმები გამოთვლებში იყენებენ შემდეგ დომენურ პარამეტრებს:

- მარტივი რიცხვი p, რომელიც განსაზღვრავს Fp სასრული ველის ზომას;
- ელიპტიკური წირის განტოლების a და b კოეფიციენტები;
- ბაზური წერტილი G, რომელიც წარმოქმნის გამოთვლებში გამოყენებულ ქვეჯგუფს (ციკლს);
- ქვეჯგუფის რიგი n (ციკლში წერტილების რაოდენობა);
- ქვეჯგუფის კოფაქტორი h.

ეს პარამეტრები ქმნიან ექვს ელემენტიან კორტეჟს (p, a, b, G, n, b).

„იოლი“ და „რთული“ პრობლემები

როგორც ზემოთ ითქვა, წერტილის სკალარზე გამრავლება ნიშნავს შემდეგს: მოცემულია წირის წერტილი P და სკალარი (ნატურალური რიცხვი) n; საჭიროა ვიპოვოთ წერტილი

$$Q = n \cdot P.$$

ეს ამოცანა ითვლება „იოლად“, რადგან სათანადო გამოთვლები შეიძლება შესრულდეს ეფექტურად, მაგალითად „გააორმაგე და შეკრიბე“ ალგორითმის გამოყენებით. რაც შეეხება მოცემული P და Q-ითვის ისეთ n-ის პოვნას, რომელიც აკმაყოფილებს ტოლობას

$$Q = n \cdot P.$$

არის „რთული“, რადგან დღეისათვის არ არსებობს ამ ამოცანის გადაწყვეტის ეფექტური ალგორითმი, იგი მოითხოვს ვარიანტების გადასინჯვას. რაც n-ის დიდი მნიშვნელობისათვის პრაქტიკულად შეუძლებელია.

4.2.13. ელიპტიკურ წირებზე დაფუძნებული კრიპტოგრაფია

აქამდე განხილული იყო ელიპტიკური წირების თვისებები ორი შემთხვევისთვის: როცა წირის წერტილების კოორდინატები ღებულობენ მნიშვნელობებს ნამდვილ რიცხვთა სიმრავლიდან (უწყვეტი წირები) და როცა ისინი ღებულობენ მნიშვნელობებს სასრული სიმრავლიდან (დისკრეტული წირები). ქვემოთ მოყვანილია დისკრეტულ ფუნქციებზე დაფუძნებული კრიპტოგრაფიის მაგალითები, რადგან ისინი ფართოდ გამოიყენებიან გამოთვლების ეფექტურობის გამო.

4.2.14. ასიმეტრიული კრიპტოგრაფია – საიდუმლო და ღია გასაღებები

ელიპტიკურ კრიპტოგრაფიაში კერძო, ანუ საიდუმლო და საჯარო ანუ ღია გასაღებები განისაზღვრება შემდეგნაირად:

კერძო გასაღები, d, წარმოადგენს მთელ რიცხვს, რომელიც აირჩევა სიმრავლიდან {1, 2, … , n-1}, სადაც n არის ქვეჯგუფის რიგი, ანუ ქვეჯგუფში შემავალი წერტილების რაოდენობა;

საჯარო ანუ ღია გასაღები გამოითვლება ფორმულით H = d·G, სადაც G ქვეჯგუფის ბაზური წერტილია, ხოლო d · G ბაზური წერტილის d-ზე სკალარული ნამრავლი. რამდენადაც H ელიპტიკური ფუნქციის ერთ-ერთი წერტილია, მას აქვს ორი კოორდინატა ± აბსცისა x და ორდინატა y. როცა ცნობილია d და G, H-ის გამოთვლა წარმოადგენს „იოლ“ ამოცანას, საკმარისია ვიპოვოთ სკალარული ნამრავლი, მაგალითად „გააორმაგე და შეკრიბე“ ალგორითმით. იმ შემთხვევაში, როცა მოცემულია H და G, მაშინ d-ს პოვნა არის „ძნელი“ ამოცანა, რადგან ეს ასე თუ ისე უკავშირდება დიდი რაოდენობის ვარიანტების გადასინჯვის ამოცანას, რაც თანამედროვე კომპიუტერების გამოყენებით ძალიან დიდ დროს მოითხოვს. თუმცა, დღეისათვის მათემატიკურად დამტკიცებული არ არის, რომ არ შეიძლება აღმოჩენილ იქნეს ალგორითმი, რომელიც H = d · G განტოლებიდან d-ს პოვნის „ძნელ“ ამოცანას „იოლ“ ამოცანად აქცევს (ამოცანის ამოხსნისათვის საჭირო დროის თვალსაზრისით).

4.2.15. შიფრაციის მაგალითები: ECDH, ECDSA

ამ პარაგრაფში განიხილება ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფიის გამოყენების ორი გავრცელებული მაგალითი: ECDH და ECDSA.

ECDH (Elliptic Curve Diffie-Hellman, დიფი-ჰელმანის ალგორითმი ელიპტიკური წირისათვის) განკუთვნილია არა უშუალოდ მონაცემების დაშიფვრისათვის, არამედ სიმეტრიული გასაღებით შიფრაციისათვის საჭირო სიმეტრიული გასაღების შექმნისათვის.

რაც შეეხება ECDSA (Elliptic Curve Digital Singonture Algorithm, ელიპტიკური წირის გამოყენებით ციფრული ხელისმოწერის ალგორითმი), იგი გამოიყენება ელექტრონული ხელმოწერისათვის ისეთ სისტემებში, როგორიცაა, მაგალითად, Bitcoin, Ethhereum, TLS და სხვა მსგავსი სისტემები.

4.2.16. დიფი-ჰელმანის ალგორითმი ელიპტიკური წირისათვის, ECDH

ცნობილია, რომ მონაცემების შიფრაცია-დეშიფრაცია სიმეტრიული გასაღებით გაცილებით სწრაფად ხდება, ვიდრე ღია გასაღებით. ამავე დროს სიმეტრიული გასაღების ხანგრძლივად გამოყენება დაკავშირებულია მისი საიდუმლოდ შენახვის და მართვის პრობლემებთან. ამიტომ ხშირად გამოიყენება შიფრაციის პროცესის გაყოფა ორ ეტაპად. პირველ ეტაპზე ღია გასაღების გამოყენებით ხდება სიმეტრიული გასაღების შექმნა, ხოლო მეორე ეტაპზე ხდება მონაცემების დაშიფვრა, გადაცემა და დეშიფრაცია სიმეტრიული გასაღების გამოყენებით. ელიპტიკურ წირზე დაფუძნებულ კრიპტოგრაფიაში ეს ხორციელდება შემდეგნაირად.

ვთქვათ ელისს (Alice) და ბობს (Bob) სურთ გაცვალონ დაშიფრული მონაცემები. ამისათვის საჭიროა მათ გამოიყენონ დომენური პარამეტრები, რომლებიც განსაზღვრავენ ელიპტიკურ წირს, ბაზურ წერტილს G და ქვეჯგუფის რიგს n. ამის შემდეგ ხორციელდება შემდეგი ნაბიჯები:

1. ელისი ირჩევს თავის საიდუმლო გასაღებს d_A და გამოთვლის თავის ღია გასაღებს $H_A = d_A \cdot G$. ანალოგიურად, ბობი ირჩევს თავის საიდუმლო გასაღებს d_B და გამოთვლის თავის ღია გასაღებს $H_B = d_B \cdot G$. ცხადია d_A და d_B მნიშვნელობები აიღებიან სიმრავლიდან {1, 2, … , n-1}, სადაც n ქვეჯგუფში შემავალი წირის წერტილების რაოდენობაა.

2. ელისი და ბობი ერთმანეთს უცვლიან თავიანთ H_A და H_B ღია გასაღებებს როგორც დაუშიფრავ მონაცემებს.

3. ელისი თავისი საიდუმლო d_A-ს გამოყენებით გამოთვლის $S= d_A \cdot H_B$. ამის მსგავსად ბობი გამოთვლის $S= d_B \cdot H_A$. ამგვარად, ორივე ღებულობს S-ის ერთსადაიმავე მნიშვნელობას რადგან

$$S = d_A \cdot H_B = d_A \cdot d_B \cdot G = d_B \cdot H_A = d_B \cdot d_A \cdot G = d_A \cdot d_B \cdot G$$

ამგვარად, ელისს და ბობს აქვთ საერთო S, რომელიც წარმოადგენს ელიპტიკური წირის წერტილს და აქვს ორი კოორდინატა x და y. ამ წერტილის x კოორდინატა, რომლის მნიშვნელობა შედის {1, 2, ... , n-1} სიმრავლეში, შეიძლება გამოყენებულ იქნეს მონაცემთა სიმეტრიული გასაღებით კოდირებისათვის, მაგალითად კოდირების ისეთ სისტემებში, როგორიცაა AES ან 3DES.

აღწერილი ალგორითმი ხშირად გვხვდება დასახელებით ECDHE ნაცვლად ECDH, სადაც ბოლო ასო E აღნიშნავს „Ephemeral“, რაც მიუთითებს რომ მიღებული სიმეტ-რიული გასაღები განკუთვნილია ერთჯერადი ან მოკლევადიანი გამოყენებისთვის.

4.2.17. ელიპტიკურ წირზე დაფუძნებული ციფრული ხელმოწერა

ECDSA (Elliptic Curve Digital Singnature Algorithm) წარმოადგენს DSA (Digital Signature Algorithm) ელექტრონული ხელმოწერის ალგორითმის მოდიფიკაციას ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფიისთვის. ამ სქემის თანახმად:

- მონაცემის გამგზავნს, ელისს, სურს განახორციელოს გზავნილის ხელმოწერა მისი საიდუმლო გასაღების d_A-ის გამოყენებით;
- მონაცემის მიმღებს, ბობს, სურს მოახდინოს გზავნილის ვალიდაცია (სისწორეზე შემოწმება) ელისის ღია გასაღების, H_A-ის გამოყენებით.

იგულისხმება, რომ მხოლოდ ელისმა იცის თავისი საიდუმლო გასაღები და მხოლოდ მას შეუძლია მოახდინოს ციფრული ხელმოწერა. რაც შეეხება ელისის ღია გასაღებს, იგი ცნობილია როგორც ბობის, აგრეთვე სხვა მხარისათვის და, ამგვარად, ნებისმიერ დაინტერესებულ მხარეს შეუძლია შეამოწმოს გზავნილის ვალიდურობა. გარდა ამისა იგულისხმება, რომ ყველა მხარისათვის ცნობილია დომენური პარამეტრები:

– მარტივი რიცხვი p, რომელიც განსაზღვრავს სასრული ველის ზომას;

– ელიპტიკური წირის კოეფიციენტები a და b;

– ბაზური წერტილი G, რომელიც წარმოქმნის გამოყენებულ ქვეჯგუფს (ციკლში შემავალი წერტილების სიმრავლეს);

– ქვეჯგუფში (ციკლში) შემავალი წერტილების რაოდენობა n;

– ქვეჯგუფის კოფაქტორი h.

ეს მონაცემები წარმოდგება კორტეჟის (P, a, b, G, n, h) სახით, რომელიც ცნობილია მონაცემების გაცვლაში მონაწილე მხარეებისათვის.

4.2.18. ხელმოწერა ECDSA ალგორითმის გამოყენებით

როგორც გზავნილის ხელმოწერის, ისე მისი ვერიფიკაციის ალგორითმებში გამოიყენება m (message) გზავნილის ჰეშირების მეთოდი: ECDSA იყენებს არა პირდაპირ გზავნილს, არამედ მის ჰეშს. ჰეშირების მეთოდი ცნობილია ორივე მხარისათვის და იგი უნდა იყენებდეს კრიპტოგრაფიულად საიმედო ჰეშ-ფუნქციას. თუ ჰეშის სიგრძე ბიტებში აღემატება n-ის სიგრძეს ბიტებში, მაშინ ხდება მისი შეკვეცა. კერძოდ, აიღება ჰეშის n ბიტი, დაწყებული მარცხნიდან, რომელიც აღინიშნება როგორც პარამეტრი z.

ციფრული ხელმოწერის ალგორითმი, რომელსაც ელისი ასრულებს, მდგომარეობს შემდეგი ნაბიჯების შესრულებაში:

1. აირჩიე შემთხვევითი მთელი რიცხვი k სიმრავლიდან {1, 2, ... , n-1};

2. გამოთვალე წერტილი P = k · G, სადაც G ბაზური, ხოლო P ციკლის ერთ-ერთი წერტილია;

3. გამოთვალე r = $x_P$modn სადაც x_p არის P წერტილის აბსცისა;

4. თუ r = 0, აირჩიე k-ს სხვა მნიშვნელობა და გაიმეორე ზემოაღწერილი ნაბიჯები;

5. გამოთვალე s = k^{-1} (z + r ·d_A) modn, სადაც k^{-1} არის k -ს მულტიპლიკაციური ინვერსია მოდულით n;

6. თუ s =0, გაიმეორე ზემოთმოყვანილი ნაბიჯები თავიდან;

7. წყვილი (r, s) არის ხელმოწერა.

4.2.19. ხელმოწერის ვერიფიკაცია

ხელმოწერის ვერიფიკაცია ხდება შემდეგი მონაცემების გამოყენებით:

– ელისის ღია გასაღები H_A;

– მესიჯ m-ის შეკვეცილი ჰეში z;

– ხელმოწერა (r, s).

ვერიფიკაციის პროცედურა მდგომარეობს შემდეგში:

1. გამოთვალე მთელი რიცხვი $u_1 = S^{1-1}$·z modn;

2. გამოთვალე მთელი რიცხვი $u_2 = S^{1-1}$·r modn;

3. გამოთვალე წერტილი P= u_1· G + u_2 · H_A.

ხელმოწერა ვალიდურია მხოლოდ იმ შემთხვევაში, როცა r = x_p (modn).

ციტირებული ლიტერატურა

1. Э.Таленваум, Сомрьютерные сети, Питер, (Глава 8. Безопасность в сетях).

2. Andrea Corbellini, Elliptic Curve Cryptography: A Gentle Entroduction.

3. Khan Academy, Modular Arithmetic