საქართველოს ტექნიკური უნივერსიტეტი
1922 წლიდან

ნოდარ ლომინაძე, თამარ ასათიანი
თამარ ლომინაძე, რუსუდან პაპიაშვილი

კრიპტოგრაფიის საფუძვლები

სახელმძღვანელო

სტუ-ის „IT-კონსალტინგის სამეცნიერო ცენტრი”

ნოდარ ლომინაძე, თამარ ასათიანი,
თამარ ლომინაძე, რუსუდან პაპიაშვილი

კრიპტოგრაფიის საფუძვლები

დამხმარე სახელმძღვანელო

დამტკიცებულია:

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრის“ სარედაქციო კოლეგიის მიერ

თბილისი 2024

UDC (უაკ) 004.312.26
ლ-816

წარმოდგენილ დამხმარე სახელმძღვანელოში დეტალურადაა განხილული მოდულური არითმეტიკა, Xor არითმეტიკა და გალუას ველის არითმეტიკა, რომლებიც ქმნიან საფუძველს კრიპტოგრაფიული ალგორითმების დამუშავებისთვის. განხილულია ჰეშირების მეთოდები, რაც ფართე გამოყენებას პოულობს სხვადასხვა დანიშნულების ალგორითმებში. გაუმჯობესებული სიმეტრიული შიფრაციის ალგორითმები AES (Advanced Encryption System), რომელიც დღეისათვის საუკეთესოა სიმეტრიული შიფრაციის ალგორითმებს შორის. AES გალუას მთვლელის რეჟიმით, რომელიც საშუალებას იძლევა ერთდროულად მოხდეს როგორც მონაცემების შიფრაცია, ისე მისი ავთენტობა.

ასევე განხილულია ასიმეტრიული შიფრაციის დიფი-ჰელმანის ალგორითმი, რომელიც ფართო გამოყენებას პოულობს კლიენტ-სერვერული მისალმების (Handshake) ალგორითმებში. ასიმეტრიული შიფრაციის ალგორითმი RSA, რომელიც დღეისათვის საკმაოდ ფართოდაა გავრცელებული, მაგრამ მილევად რეჟიმშია. ასიმეტრიული შიფრაციის ელიპტიკურ წირზე დაფუძნებული ალგორითმები, რომლებიც მახასიათებლებით დღეისათვის საუკეთესოა. წარმოდგენილი მოდელების კომპლექსურად გამოყენება კომპიუტერულ ქსელებში ტრანსპორტის დონის უსაფრთხოების (TLS, Transport Layer Sequirity) განხორციელების საკითხებში.

დამხმარე სახელმძღვანელო გამიზნულია ამ საკითხებით დაინტერესებული სტუდენტებისა და მკითხველისათვის.

რეცენზენტები:

საქართველოს ტექნიკური უნივერსიტეტის საინფორმაციო ტექნოლოგიების დეპარტამენტის პროფესორი – **რევაზ კაკუბავა**

საქართველოს ტექნიკური უნივერსიტეტის საინფორმაციო სისტემების დეპარტამენტის ხელმძღვანელი, პროფესორი – **მედეა თევდორაძე**

ISBN 978-9941-8-7348-5

Nodar Lominadze, Tamar Asatiani,
Tamar Lominadze, Rusudan Papiashvili

The Fundamentals of Cryptography

Approved by:

The editorial board of the "IT-Consulting scientific center" of the Georgian Technical University (GTU).

Reviewers:

Professor **Revaz Kakubava**, Department of Information Technology, Georgian Technical University

Professor **Medea Tevdoradze**, Head of the Department of Information Systems, Georgian Technical University

UDC (უაკ) 004.312.26
ლ-816

In the presented handbook, comprehensive discussions are provided on modular arithmetic, XOR arithmetic, and Galois field arithmetic, which form the foundational principles for cryptographic algorithms and their implementations. The guide offers detailed coverage of Hash methods, which enable the generation of diverse and distinct algorithmic outputs. Additionally, it examines symmetric encryption algorithms, particularly the Advanced Encryption Standard (AES), which is currently the most widely used symmetric encryption method globally. AES employs a Galois field multiplier framework, allowing concurrent operations to ensure both the encryption of messages and their authenticity.

The handbook also delves into asymmetric encryption algorithms, including Diffie-Hellman, which is integral to client-server handshake protocols. It further explores the RSA algorithm, which remains in use for various applications despite its relatively slower performance. Moreover, it provides an in-depth analysis of elliptic curve-based asymmetric encryption algorithms, valued for their efficiency and widespread application in contemporary cryptographic systems. The discussions extend to the complexities of secure transport layer protocols, such as Transport Layer Security (TLS), and their critical role in safeguarding communications within computer networks.

This handbook is designed to serve as a valuable resource for students and readers interested in the aforementioned topics.

ISBN 978-9941-8-7348-5

სარჩევი

წინასიტყვაობა

წინამდებარე ნაშრომი განეკუთვნება სფეროს, რომლის შინაარსი კლოდ შენონმა (Cload Shanon) განსაზღვრა როგორც არასაიმედო ელემენტებიდან საიმედო სისტემების შექმნის საკითხები, რაც მნიშვნელოვან ადგილს იკავებს თანამედროვე ინფორმატიკის პრაქტიკულად ყველა მიმართულებაში და მათ შორის ინფორმაციული სისტემების უსაფრთხოების უზრუნველყოფის ამოცანებში.

თუმცა ინფორმაციული უსაფრთხოების სფეროში უამრავი სამუშაო იქნა ჩატარებული ათწლეულების მანძილზე, რომლებიც პრაქტიკაში ინერგებოდნენ საერთაშორისო ორგანიზაციების მიერ რეკომენდირებული სტანდარტების სახით, ამავე დროს ხდებოდა ახალი სტანდარტების შემუშავება, რაც წარმოქმნიდა მოქმედი, მოძველებული სტანდარტების ახალით ჩანაცვლების და ამგვარად, მნიშვნელოვანი სამუშაოების ჩატარების აუცილებლობას.

დღეისათვის ჩამოყალიბდა ალგორითმების შედარებით მცირე რაოდენობა, რომლებიც წინამორბედებთან შედარებით მეტი სირთულით ხასიათდებიან, მაგრამ უზრუნველყოფენ გაუმჯობესებულ ეფექტურობას და ჰაკერული შეტევებისადმი მეტ მდგრადობას. სწორედ ამ ალგორითმებზეა ორიენტირებული წინამდებარე ნაშრომი.

ნაშრომის შინაარსი შემდეგნაირად შეიძლება ჩამოყალიბდეს:

პირველ თავში, მათემატიკური საფუძვლები, სხვა საკითხებთან ერთად განხილულია მოდულური არითმეტიკა, Xor არითმეტიკა და გალუას ველის არითმეტიკა, რომლებიც ქმნიან საფუძველს კრიპტოგრაფიული ალგორითმების დამუშავებისთვის.

მეორე თავში, ჰეშირების ფუნქციები, განხილულია ჰეშირების მეთოდები, რაც ფართე გამოყენებას პოულობს სხვადასხვა დანიშნულების ალგორითმებში.

მესამე თავში განიხილება გაუმჯობესებული სიმეტრიული შიფრაციის ალგორითმები AES (Advanced Encryption System), რომელიც დღეისათვის საუკეთესოა სიმეტრიული შიფრაციის ალგორითმებს შორის.

მეოთხე თავში განხილულია AES გალუას მთვლელის რეჟიმით, რომელიც საშუალებას იძლევა ერთდროულად მოხდეს როგორც მონაცემების შიფრაცია, ისე მისი ავთენტობა.

მეხუთე თავში განხილულია ასიმეტრიული შიფრაციის დიფი-ჰელმანის ალგორითმი, რომელიც ფართო გამოყენებას პოულობს კლიენტ-სერვერული მისალმების (Handshake) ალგორითმებში.

მეექვსე თავში განხილულია ასიმეტრიული შიფრაციის ალგორითმი RSA, რომელიც დღეისათვის საკმაოდ ფართოდაა გავრცელებული, მაგრამ მილევად რეჟიმშია.

მეშვიდე თავში განხილულია ასიმეტრიული შიფრაციის ელიპტიკურ წირზე დაფუძნებული ალგორითმები, რომლებიც მახასიათებლებით დღეისათვის საუკეთესოა, თუმცა მისი გამოყენება კრიპტოგრაფიის მეთოდების ჩაღრმავებულ ცოდნას მოითხოვს.

მერვე თავში განხილულია წინა თავებში წარმოდგენილი მოდელების კომპლექსურად გამოყენება კომპიუტერულ ქსელებში ტრანსპორტის დონის უსაფრთხოების (TLS, Transport Layer Sequirity) განხორციელების საკითხებში.

შეიძლება დარწმუნებით ითქვას, რომ ამ ნაშრომის გაცნობა მკითხველს საშუალებას მისცემს გააღრმავოს თავისი წარმოდგენა ინფორმაციული უსაფრთხოების სფეროში გამოყენებული თანამედროვე ალგორითმებისა და მეთოდების შესახებ.

თავი 1. მათემატიკური საფუძვლები

1.1 რიცხვთა სიმრავლეები და მათზე განმარტებული ოპერაციები

ამბობენ, რომ რიცხვთა სიმრავლეზე განმარტებულია არითმეტიკული ოპერაცია, თუ ამ სიმრავლიდან აღებულ ნებისმიერი ორ რიცხვზე მოცემული ოპერაციის შესრულების შედეგად მიიღება რიცხვი, რომელიც ამავე სიმრავლეს ეკუთვნის.

ნატურალურ რიცხვთა სიმრავლე, N

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

N = {1, 2, 3, ...},
ან 0-დან უსასრულობამდე,
N= {0,1,2,3, …}.

ნატურალურ რიცხვთა სიმრავლეზე განმარტებულია შეკრებისა და გამრავლების ჩვეულებრივი არითმეტიკული ოპერაციები, რადგან ორი მთელი დადებითი რიცხვის ჯამი და ნამრავლი ისევ მთელი დადებითი რიცხვია და ეკუთვნის ნატურალურ რიცხვთა სიმრავლეს. 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.

ნატურალურ რიცხვთა სიმრავლე სასრული რაოდენობის ელემენტებით

როგორც ზემოთ ითქვა, ნატურალურ რიცხვთა სიმრავლე უსასრულოა და მასზე განმარტებულია მხოლოდ შეკრებისა და გამრავლების ოპერაციები.

ეხლა განვიხილოთ თუ რა ოპერაციებია განმარტებული ნატურალურ რიცხვთა სასრულ სიმრავლეზე

$$S= \{0,1,2,3,4,5,6\}$$

ამ სიმრავლეზე არაა განმარტებული შეკრების ოპერაცია, რადგან მისი ნებისმიერი ორი ელემენტის ჯამი არ ეკუთვნის ამავე სიმრავლეს. ეს სამართლიანია აგრეთვე გამოკლების, გამრავლების და გაყოფის ოპერაციებისათვის. ამგვარად, ნატურალურ რიცხვთა სასრულ სიმრავლეზე არცერთი ჩვეულებრივი არითმეტიკული ოპერაცია არაა განმარტებული. ამავე დროს კრიპტოგრაფიაში ფართო გამოყენებას პოულობს ე.წ. მოდულური არითმეტიკა, რომელიც ქვემოთ იქნება განხილული. იგი ეყრდნობა ნატურალურ რიცხვთა სასრულ სიმრავლეებს და მასში ოთხივე არითმეტიკული ოპერაცია არის განმარტებული.

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 საპოვნელად 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, **ერატოსთენეს 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, რიცხვის მარტივობაზე შემოწმების (**Primality test**) რომელიმე ალგორითმით. უარყოფითი შედეგის შემთხვევაში ავიღოთ n-ის ახალი მნიშვნელობა და შევამოწმოთ იგი მარტივობაზე. 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, რაც ქვემოთ იქნება განხილული. 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**), რომელიც ქვემოთ იქნება განხილული, საჭირო ხდება რიცხვი M-ის გამყოფების სიმრავლის პოვნა. მაგალითად, რიცხვ, M=42 გამყოფების სიმრავლეა {2, 3, 6, 7, 14, 21}.

რაც შეეხება ალგორითმების პროგრამულ რეალიზაციას, აქ გამოიყენება სხვადასხვა დაპროგრამების ენებში განხორციელებული დიდი რაოდენობის ბიბლიოთეკები, რომლებიც ათასობით ფუნქციებს მოიცავენ.

მთელ რიცხვთა სიმრავლე, Z

მთელ რიცხვთა სიმრავლე შედგება ყველა დადებითი და უარყოფითი რიცხვისა და რიცხვ 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.

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 ∞. (ლ. მძინარიშვილი, ნ. კაჭახიძე, დ. უგულავა, ნ. ხომერიკი, 2018)

1.2 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 მარცხნიდან 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].

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, ყველა ჩამოთვლილი სიმბოლო წარმოდგება 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

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.

1.3 ტექსტური მონაცემების კოდირება: ASCII, Unicode

ეს პარაგრაფი მოიცავს ორი ფართოდ გავრცელებული კოდირების მეთოდების მოკლე დახასიათებას, როგორიცაა:

ASCII, American Standart Code for Information Interchange - ამერიკული სტანდარტული კოდი მონაცემთა გაცვლისთვის, რომელიც ორიენტირებულია ინგლისურენოვანი ტექსტური მონაცემების კომპიუტერებს შორის გაცვლისათვის.

Unicode, Universal Encoding, უნივერსალური კოდირება, ორიენტირებული მთელ მსოფლიოში არსებულ დამწერლობებში წარმოდგენილი ტექსტური ინფორმაციების კომპიუტერისთვის გასაგებ ენაზე წარმოდგენის უნივერსალურ ფორმატზე.

ASCII კოდი

როგორც ცნობილია, კომპიუტერში მონაცემები წარმოდგენილია ბიტური ფორმით, როცა ბიტი, b, ღებულობს მნიშვნელობას სიმრავლიდან {0,1}. მაგრამ ალგორითმებში მონაცემები, იქნება ეს რიცხვითი თუ ტექსტური, წარმოდგენილია ბიტების ან ბაიტების სახით. ზოგადად ბაიტი B (ოქტეტი) წარმოადგენს 8 ბიტისაგან შედგენილ მიმდევრობას $B=b_7\,b_6\,b_5\,b_4\,b_3\,b_2\,b_1\,b_0$, რაც იძლევა $2^8=256$ მნიშვნელობას დიაპაზონში 00000000 ÷ 11111111.

ASCII კოდი, შეიქმნა ინგლისურენოვანი ტექსტების კომპიუტერული დამუშავებისთვის. შერჩეული იქნა 128 სიმბოლო, რომელიც კოდირებული (დანომრილი) იქნა ათობითი რიცხვებით დიაპაზონიდან 0 ÷ 127, რაც ორობით წარმოდგენაში 7 ბიტს მოითხოვდა დიაპაზონში 0000000,0000001,⋯,1111111. საბოლოოდ, სიმბოლოები წარმოდგენილ იქნა ბაიტებით (რვა ბიტიანი ოქტეტებში) $B=b_7\,b_6\,b_5\,b_4\,b_3\,b_2\,b_1\,b_0$, რომელშიც b_7 უდიდესი, ხოლო b_0 უმცირესი წონის ბიტია. ამგვარად კოდირება ხორციელდება ფორმატით:

$$0, x_6\,x_5\,x_4\,x_3\,x_2\,x_1\,x_0,$$

DEC	HEX	Binary	Symbol
32	20	00100000	SP
33	21	00100001	!
34	22	00100010	"
35	23	00100011	#
36	24	00100100	$
37	25	00100101	%
38	26	00100110	&
39	27	00100111	'
40	28	00101000	(
41	29	00101001	)
42	2A	00101010	*
43	2B	00101011	+
44	2C	00101100	,
45	2D	00101101	-
46	2E	00101110	.
47	2F	00101111	/
48	30	00110000	0
49	31	00110001	1
50	32	00110010	2
51	33	00110011	3
52	34	00110100	4
53	35	00110101	5
54	36	00110110	6
55	37	00110111	7
56	38	00111000	8
57	39	00111001	9
58	3A	00111010	:
59	3B	00111011	;
60	3C	00111100	<
61	3D	00111101	=
62	3E	00111110	>
63	3F	00111111	?

DEC	HEX	Binary	Symbol
64	40	01000000	@
65	41	01000001	A
66	42	01000010	B
67	43	01000011	C
68	44	01000100	D
69	45	01000101	E
70	46	01000110	F
71	47	01000111	G
72	48	01001000	H
73	49	01001001	I
74	4A	01001010	J
75	4B	01001011	K
76	4C	01001100	L
77	4D	01001101	M
78	4E	01001110	N
79	4F	01001111	O
80	50	01010000	P
81	51	01010001	Q
82	52	01010010	R
83	53	01010011	S
84	54	01010100	T
85	55	01010101	U
86	56	01010110	V
87	57	01010111	W
88	58	01011000	X
89	59	01011001	Y
90	5A	01011010	Z
91	5B	01011011	[
92	5C	01011100	\
93	5D	01011101	]
94	5E	01011110	^
95	5F	01011111	_

სურათი 1. ASCII კოდირებაში გამოყენებული სიმბოლოების სიმრავლე და მათი კოდირება (გაგრძელება მომდევნო გვერდზე).

DEC	HEX	Binary	Symbol
96	60	01100000	`
97	61	01100001	a
98	62	01100010	b
99	63	01100011	c
100	64	01100100	d
101	65	01100101	e
102	66	01100110	f
103	67	01100111	g
104	68	01101000	h
105	69	01101001	i
106	6A	01101010	j
107	6B	01101011	k
108	6C	01101100	l
109	6D	01101101	m
110	6E	01101110	n
111	6F	01101111	o
112	70	01110000	p
113	71	01110001	j
114	72	01110010	r
115	73	01110011	s
116	74	01110100	t
117	75	01110101	u
118	76	01110110	v
119	77	01110111	w
120	78	01111000	x
121	79	01111001	y
122	7A	01111010	z
123	7B	01111011	{
124	7C	01111100	|
125	7D	01111101	}
126	7E	01111110	~
127	7F	01111111	DEL

სურათი 1.1. სურათი 1-ის გაგრძლება

სადაც ბაიტის უდიდესი ბიტი $b_7=0$ ყოველი სიმბოლოს კოდისათვის, ხოლო სიმბოლოების უნიკალურად კოდირება (დანომვრა) ხდება ორობითი რიცხვებით, რომლებიც მიიღებიან ზემოთ მოყვანილ ფორმატში x_i - ის 0-ით ან 1-ით ჩანაცვლების შედეგად.

ASCII კოდირებაში გამოყენებული სიმბოლოები იყოფა ორ ქვესიმრავლედ. პირველში შედის ე.წ. დაბეჭდვადი სიმბოლოები, რომლებიც გამოიტანებიან კომპიუტერის მონიტორზე ვიზუალურად გამოყენებისთვის, ხოლო მეორეში გაერთიანებულია არაბეჭდვადი სიმბოლოები, რომელთა დანიშნულებაა ტექსტური ინფორმაციის კომპიუტერებს შორის გადაცემასთან დაკავშირებული პროცესების მართვასთან. ASCII კოდის ცხრილი ნაჩვენებია სურათი 1-ზე, რომელშიაც არ არის წარმოდგენილი მმართველი სიმბოლოები ნომრებით (კოდებით) 0=დან 31-მდე.

როგორც სურათი 1 გვიჩვენებს ცხრილი შეიცავს როგორც სიმბოლოს (ვიზუალურ გამოსახულებას) ისე მის კოდურ პოზიციას ათვლის ათობით (Decimal), თექვსმეტობით (Hexadecimal) და ორობით (Binary) სისტემაში. როგორც ვხედავთ, ყველა სიმბოლოს ორობით წარმოდგენაში შესაბამისი ბაიტის უდიდესი ბიტი $b_7=0$.

UNICODE

როგორც ითქვა ASCII კოდი გამიზნული იყო მხოლოდ ინგლისურენოვანი ტექსტების კოდირებისთვის, მაგრამ თანდათანობით წარმოიშვა კოდირების მეთოდების დამუშავების აუცილებლობა მსოფლიოს სხვა ძველი და თანამედროვე დამწერლობებისათვის, რასაც საბოლოოდ მოყვა ერთი ყოვლისმომცველი კოდირების მეთოდის დამუშავება (Unicode - უნივერსალური კოდი), რაც გულისხმობს მსოფლიო დამწერლობებში გამოყენებული 1112064 სიმბოლოს კომპიუტერში წარმოდგენის შესაძლებლობას.

როგორც ზემოთ ითქვა ASCII კოდირების სისტემაში გამოყენებულ ყოველ 128 სიმბოლოდან შეესაბამება ერთ-ერთი 0,1,2 ⋯,126, 127 ათობითი რიცხვებიდან, ანუ ხდება სიმბოლოების ცალსახად წარმოდგენა რიცხვების სახით, რისთვისაც საკმარისია ერთი ბაიტის გამოყენება, რომელშიც b_7 ბიტი ყოველთვის ნულის ტოლია, ხოლო დანარჩენი 7 ბიტი $b_6 b_5 \cdots b_0$ წარმოქმნის 2^7 კომბინაციას, რაც საკმარისია 128 სიმბოლოს ცალსახად დასანომრად.

ანალოგიური მიდგომაა გამოყენებული Unicode კოდირებაში, სადაც განიხილება მილიონზე მეტი სიმბოლო და მათი ცალსახად დანომვრა მოითხოვს რამდენიმე ბაიტს. Unicode-ის შემთხვევაში ნომრების მიმდევრობა წარმოადგენს როგორც

$$u+0, u+1, u+2, \cdots$$

და თითოეულ ნომერს კოდის პოზიცია (Code Position) ან კოდური წერტილი (Code Point) ეწოდება. დღეისათის გავრცელებულია Unicode-ს სამი ძირითადი ვარიანტი:

- Unicode (Unicode Transformation Format), UTF-8. აქ თითოეული კოდური პოზიციის კოდირებისთის გამოიყენება ერთიდან 4 ბაიტამდე.
- Unicode UTF-16, სადაც თითოეული სიმბოლოს კოდირებისთვის გამოიყენება ერთი ორბაიტიანი რიცხვი ან ორი ორბაიტიანი რიცხვი.
- Unicode UTF-32, სადაც თითოეული სიმბოლოს კოდირებისთვის გამოიყენება ოთხბაიტიანი რიცხვი.

ამ სამი მიმართულებიდან ყველაზე უფრო გავრცელებულია UTF-8, რაც დღეისათვის ინტერნეტში გავრცელებული ტექსტური მონაცემების 85-98% მოიცავს.

UTF-8 კოდირება

როგორც ზემოთ ითქვა UTF-8 კოდირებაში გამოიყენება ერთიდან ოთხ ბაიტამდე, როგორც სურათი 2-ზეა წარმოდგენილი.

პირველი კოდური პოზიცია (Hex)	ბოლო კოდური პოზიცია (Hex)	ბაიტი 1	ბაიტი 2	ბაიტი 3	ბაიტი 4	კოდური პოზიციების რაოდენობა
u+0000	u+007F	0xxxxxx	///////////////	///////////////	///////////////	128
u+0080	u+07FF	110xxxxx	10xxxxxx	///////////////	///////////////	1920
u+0800	u+FFFF	1110xxxx	10xxxxxx	10xxxxxx	///////////////	61440
u+10000	u+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx	1048576

სურათი 2

კოდური პოზიციების მთლიანი სიმრავლე დაყოფილია 4 ქვესიმრავლედ შემდეგნაირად:

პირველ ქვესიმრავლეში შედის პოზიციები დიაპაზონიდან

$$U+0000 \div U+007F$$

რაც განსაზღვრავს 128 კოდურ პოზიციას და საკმარისია ASCII კოდში შემავალი სიმბოლოების კოდირებისთვის UTF-8 კოდის ფორმატით,

$$B$$
$$0xxxxxxx$$

ამგვარად ASCII კოდში შემავალი სიმბოლოს კოდირება ხდება ისეთი ბაიტით, რომლის უდიდესი ბიტი b_7=0, ხოლო დანარჩენი ბიტები იყენებენ ერთერთს სიმრავლიდან {0,1}, რაც იძლევა 128 სიმბოლოს კოდირების საშუალებას. ამგვარად, როგორიც არ უნდა იყოს UTF-8 კოდით წარმოდგენილ ტექსტში ბაიტების მიმდევრობა, ყველა ბაიტი, რომლის უდიდესი ბიტი b_7=0, წარმოადგენს ASCII კოდის რომელიმე სიმბოლოს და მის შესახებ მონაცემები მოიპოვება ASCII ცხრილის გამოყენებით;

მეორე ქვესიმრავლეში x ღებულობს მნიშვნელობას სიმრავლიდან {0,1} და პოტენციალურად იძლევა 2^{11}=2048 სიმბოლოს კოდირების საშუალებას, თუმცა ყველა პოზიცია არ არის გამოყენებული და პრაქტიკულად გამოყენებულია 1920 პოზიცია. სხვანაირად, ამ ინტერვალში კოდირებულ სიმბოლოთა რაოდენობა არის 1920. თუ UTF-8 კოდირებულ ტექსტის შესაბამის ბაიტების მიმდევრობაში გვხვდება ბაიტი რომლის უდიდესი სამი ბიტია 110 ⋯⋯, ეს ნიშნავს, რომ სიმბოლოს კოდირება ხდება ორი ბაიტით და მას უნდა მოჰყვებოდეს ბაიტი ბიტებით 10 ⋯ და ეს ორი ბაიტი ერთობლივად უნა იქნას განხილული.

მესამე ქვესიმრავლე შეიცავს კოდურ პოზიციებს დიაპაზონიდან

$$U + 800 \div U + FFF$$

ამ შემთხვევაში სიმბოლოების კოდირება ხდება სამი ბაიტით და ეს სამი ბაიტი უნდა ქმნიდეს მიმდევრობას:

B_1	B_2	B_3
1110xxxx	10xxxxxx	10xxxxxx

თუ ასეთი მიმდევრობა არ დგება, ადგილი აქვს კოდირების შეცდომას. ეს დიაპაზონი ქმნის 61440 სიმბოლოს კოდირების შესაძლებლობას.

მეოთხე ქვესიმრავლე მოიცავს კოდურ პოზიციებს დიაპაზონიდან

$$\mathrm{U} + 10000 \div \mathrm{U} + 10\mathrm{FFFF}$$

რაც ქმნის მილიონზე მეტ სიმბოლოს კოდირების საშუალებას. აქ სიმბოლოს კოდი წარმოდგება ოთხბაიტიანი ფორმატით:

B_1	B_2	B_3	B_4
11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

რაც საშუალებას იძლევა განხოლციელდეს 2^{21} სიმბოლოს კოდირება, რაც პრაქტიკულად შეუზღუდავი შესაძლებლობაა.

სურათი 3 გვიჩვენებს რამდენიმე სიმბოლოს UTF-8 კოდირების მაგალითზე, როგორიცაა დოლარის სიმბოლო $ (Dollar sign); ფუნტის სიმბოლო £

(Poun sigh), ევროს სიმბოლო € (Euro sign) და გოთური დამწერლობის ერთ-ერთი სიმბოლო 𐌈 (Gothic letter), რომელთა UTF-8 კოდი წარმოდგება ერთი, ორი, სამი და ოთხი ბიტით.

Character ბოლო სიმბოლო	კოდური პოზიცია (Hex)	Binary Code position კოდური პოზიცია ორობითში	Binary UTF-8 ორობითი UTF-8	Hex UTF-8
$	U+0024	0000␣0000␣0010␣0100	00100100	24
£	U+00A3	0000␣0000␣1010␣0011	11000010␣10100011	C2a3
€	U+20AC	0010␣0000␣1010␣1100	11100010␣10000010␣10 101100	E282AC
𐌈	U+10348	0001␣0000␣0011␣0100 ␣1000	11110000␣10010000␣10 001101␣10001000	F0908D88

სურათი 3

ცხრილში მოცემულია სიმბოლოთა კოდური პოზიცია ათვლის თექვსმეტობით და ორობით სისტემაში. UTF-8 კოდი ორობით სისტემებში. ნახაზის ბოლო სვეტში მოცემულია სიმბოლოს UTF-8 კოდი თექვსმეტობით სისტემაში.

განვიხილოთ UTF-8 კოდირებული ტექსტის გარჩევის (დეკოდირების) მაგალითი. ვთქვათ, მოცემულია ტექსტის UTF-8 კოდი ბაიტების მიმდევრობის სახით:

B_1	B_2	B_3	B_4	B_5	B_6
00100100	00101100	11000010	10100011	00101100	11100010

B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}
10000010	10101100	00101100	111100	10010000	10001101	10001000

განვიხილოთ ბაიტების ეს მიმდევრობა მარცხნიდან მარჯვნივ.

1. ბაიტ B_1-ის უდიდესი ბიტი b_7=0. ეს ნიშნავს, რომ მასში წარმოდგენილია სიმბოლო კოდით 00100100, რაც ASCII ცხრილის შესაბამისად, იძლევა სიმბოლოს $ (დოლარის ნიშანი).

2. ბაიტ B_2-ის უდიდესი ბიტი b_7=0. ეს ნიშნავს, რომ სიმბოლო კოდირებულია კოდური პოზიციით 00101100, რაც ASCII ცხრილის შესაბამისად, იძლევა სიმბოლოს , .

3. ბაიტი B_3-ის ორი უდიდესი ბიტი 1-ის ტოლია. ეს ნიშნავს, რომ სიმბოლო კოდირებულია ორი B_3 და B_4 ბაიტებით. აქ ხაზგასმული ბიტები განსაზღვრავენ კოდირების ფორმატს, ხოლო ამ ბაიტების დანარჩენი ბიტების კონკატენაცია იძლევა სიმბოლოს UTF-8 კოდურ პოზიციას

$$00010||100011 = 0001010011$$

რაც წარმოადგენს ფუნტის სიმბოლოს £.

4. ბაიტი B_5 ASCII კოდირების თანახმად, იძლევა სიმბოლოს , .

5. B_6-ში კომბინაცია 1110 გვიჩვენებს, რომ სიმბოლო კოდირებულია სამი B_6, B_7 და B_8 ბაიტებით. ზემოთქმულის ანალოგიურად იძლევა სიმბოლოს € (Euro ევრო).

6. B_9-ში b_7=0, ანუ გვაქვს ASCII კოდი სიმბოლოსთვის , .

7. ბაიტი B_{10} შეიცავს 11110, რაც ნიშნავს, რომ ეს სიმბოლო კოდირებულია ოთხი ბაიტით B_{10}, B_{11}, B_{12}, B_{13}, რაც იძლევა გოთური დამწერლობის ასოს Θ (Gothic letter).

საბოლოოდ, ბაიტების მოცემული მიმდევრობა დეშიფრაციისას იძლევა ტექსტს:

$, £, €, Θ

უნიკოდის UTF-16 და UTF-32 ფორმატები

როგორც ზემოთ ითქვა, UTF-8 და ASCII კოდირების თავსებადი მეთოდებია და ამიტომ UTF-8 ფართო გამოყენებას პოულობს ტექსტური მონაცემების კოდირებაში. მაგალითად, Web სისტემაში გვერდების 98%-ში სწორედ UTF-8 კოდირება გამოიყენება. თუმცა, ზოგიერთ სისტემაში გამოიყენება კოდირების UTF-16 და UTF-32 ფორმატები, რომლებიც ASCII კოდირებასთან არათავსებადი არიან, რაც განაპირობებს მათი გავრცელების შეზღუდულ არეალს. როგორც UTF-16 ისე UTF-32 იყენებს კოდური პოზიციების მიმდევრობას

$$U + 0, U + 1, U + 2, \cdots$$

მაგრამ ისინი განსხვავდებიან ეფექტურობის მაჩვენებლებით.

UTF-16

კოდირების ამ სისტემის დამუშავების საწყის ეტაპზე იგულისხმებოდა, რომ 2^{16} კოდური პოზიცია საკმარისი იქნებოდა და ამიტომ პოზიციების კოდირება ხდება 16 ბიტით ანნუ ორი ოქტეტით (ბაიტით), რაც ქმნიდა 16 ბიტიან კოდურ ერთეულს. შემდგომში აღმოჩნდა, რომ სიმბოლოების რაოდენობა აღემატებოდა 2^{16}-ს, რამაც გამოიწვია კოდირების გაფართოების საჭიროება. კერძოდ, ძირითადი (ხშირად ხმარებდი) სიმბოლოების კოდირება ხდება 16 ბიტიანი (ორ ბაიტიანი) ერთეულით, ხოლო იშვიათად ხმარებული სიმბოლოების კოდირება ხდება ორი 16 ბიტიანი ერთეულის გამოყენებით. ამგვარად გამოდის, რომ UTF-16 კოდირებაშ სიმბოლოები ცვლადი სიგრძის კოდებით (ორბაიტიანი ან ოთხბაიტიანი) გამოისახება.

UTF-32

კოდირების ამ ფორმატში უნიკოდის სისტემაში გაერთიანებული თითოეული სიმბოლოს კოდირება ხდება 32 ბიტიანი (4 ბაიტიანი) ოროობითი რიცხვით, რაც წარმოქმნის 32 ბიტიანი უნიკოდის ტრანსფორმაციის ფორმატს. ეს არის კოდირება კოდის ფიქსირებული სიგრძით, რაც ზოგადად ამარტივებს გამოთვლებს, თუმცა აგრძელებს კოდირებული მონაცემების სიგრძეს და მოითხოვს დიდი ზომის მეხსიერებას.

1.4 მოდულური არითმეტიკა

მოდულური არითმეტიკა ფართო გამოყენებას პოულობს მათემატიკის სხვადასხვა სფეროებში, მათ შორის კრიპტოგრაფიაში. ქვემოთ განხილული იქნება მოდულური არითმეტიკის ძირითადი კანონები. (ლ. მძინარიშვილი, ნ. კაჭახიძე, დ. უგულავა, ნ. ხომერიკი, 2018) 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.

მოდულური ოპერატორი

მოდულური ოპერატორი აღინიშნება სიტყვით modulus, შემოკლებით mod. (ზოგ შემთხვევაში, მაგალითად ზოგიერთ დაპროგრამების 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

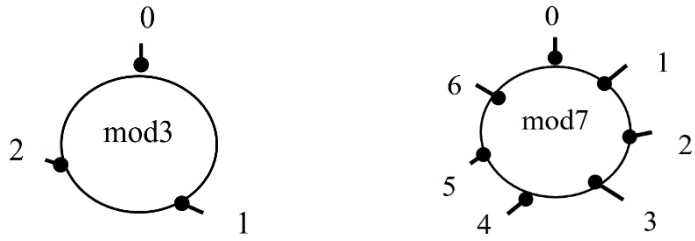
მაგალითად, $\frac{5}{3}$ = 1 და ნაშთი 2, რაც ჩაიწერება როგორც 5 mod3 = 2.

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

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) modC = (AmodC + BmodC) modC
მაგალითად, თუ A = 13, B=16 და C=5, მაშინ
(13+16) mod5 = (13mod5 + 16mod5) mod5 = (3+1) mod5 = 4 mod5 = 4

მოდულური გამოკლება

გამოკლება მოდულურ არითმეტიკაში ხორციელდება შემდეგნაირად:
(A-B) modC = (AmodC - BmodC) modC
მაგალითად, თუ A= 13, B=16 და C=5, გვექნება
(13-16) mod5 = (13mod5 - 16mod5) mod5 = (3-1) mod5 = 2 mod5 = 2

მოდულური გამრავლება

მოდულური გამრავლება ხორციელდება ფორმულით:
(A·B) modC = (AmodC · BmodC) modC
მაგალითად, A = 13, B=16 და C=5 შემთხვევაში გვექნება:
(13·16) mod5 = (13mod5 · 16mod5) mod5 = (3·1) mod5 = 3 mod5 =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

$$(A \cdot A^{-1}) \mod C = 1.$$

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 $A \cdot A^{-1} \mod 6 = 1$. ამისათვის განვიხილოთ ყველა შესაძლო ვარიანტი გადასინჯვის მეთოდით, ანუ მიმდევრობით განვიხილოთ რიცხვები 0, 1, 2, 3, 4, 5. გვექნება

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

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

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

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

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

$(5 \cdot 5) \mod 6 = 1 \leftarrow$ ინვერსიული ელემენტი ნაპოვნია!

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

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

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

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

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

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

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

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

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.

მოდულური სადარობა (კონგრუენტობა - Congruence Modulo)

ხშირად მათემატიკურ ტექსტებში გვხვდება გამოსახულება

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

სადაც ≡ სადარობის, 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 სადარი mod 5-ით, რადგან
$26 \bmod 5 = 1$ და $17 \bmod 5 = 2$ და ამგვარად $1 \neq 2$.

1.5 Xor არითმეტიკა (არითმეტიკა მოდულით 2)

ეს არითმეტიკული სისტემა განსხვავდება კომპიუტერში გამოყენებული ორობითი არითმეტიკისაგან. ტექნიკური ხასიათის ლიტერატურაში იგი გამოიყენება X_{or} არითმეტიკის დასახელებით, რაც უკავშირდება X_{or} ლოგიკურ ფუნქციას (Exclusive Or, გამორიცხული ან), აგრეთვე დასახელებით „არითმეტიკა მოდულით 2“.

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.

შეკრება

ორი რიცხვის შეკრებისას ოპერაცია ყველა თანრიგში სრულდება სხვა თანრიგებისაგან დამოუკიდებლად, ანუ, პარალელურად და თვითოეულ თანრიგში ჯამი გამოითვლება შემდეგი ცხრილის მიხედვით:

$\oplus$	0	1
0	0	1
1	1	0

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

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

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:

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

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:

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

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:

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

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

გამოკლება

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

$\ominus$	0	1
0	0	1
1	1	0

აქაც, ისევე, როგორც X_{or} შეკრების შემთხვევაში, გამოთვლები წარმოებს მოდულით 2. მაგალითად, $0\text{-}1 \rightarrow (-1) \bmod 2 = 1$.

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,

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

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

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
		⊗						
					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:

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

გაყოფა

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)		2)		1	1	0	0	1	0		
		X								+								
				1	0	1	(გამყოფი)							0	0	1	ნაშთი	
			1	0	1	0					1	1	0	0	1	1	გასა-ყოფი	
		0	0	0	0													
	1	0	1	0														
	1	1	0	0	1	0												

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:

გასაყო-ფი		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	
		⊗									
						1	0	1	1	გამყოფი	
				1	1	1	0	0	0		
			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		

2)		1		1	0	0	0	1	0	0	0
	⊕										
								0	1	1	ნაშთი
		1		1	0	0	0	1	0	1	1	გასაყოფი

1.6 გალუას ველის არითმეტიკა

გალუას ველი

სიმრავლეებისა და მათზე განმარტებული ოპერაციების განხილვისას აღნიშნულ იქნა, რომ ნატურალერ რიცხვთა სიმრავლე უსასრულოა და მასზე განმარტებულია მხოლოდ ორი ოპერაცია: შეკრება და გამრავლება. ამავე დროს კრიპტოგრაფიაში ფართო გამოყენებას პოულობს ოპერაციები სასრული რაოდენობის ელემენტების შემცველ სიმრავლეებზე, რომლის მაგალითს წარმოადგენს გალუას ველი (Galois Field, GF), რომელიც აღინიშნება როგორც GF(p) და ხასიათდება შემდეგი თვისებებით: (Corbellini, 2015)

-ამ გამოსახულებაში p წარმოადგენს მარტივ რიცხვს;

- სასრული სიმრავლე განისაზღვრება როგორც

$$Fp= \{0,1,2,\ldots..,p-1\};$$

-ამ სიმრავლეზე განმარტებული შეკრების, გამოკლების, გამრავლების და გაყოფის ოპერაციები სრულდება მოდულური არითმეტიკის წესების მიხედვით, რაც განხილულია პარაგრაფში „მოდულური არითმეტიკა“. მაგალითად, თუ p=7, გვექნება სასრული სიმრავლე {0,1,2,3,4,5,6} და შეკრებისა და გამრავლების ოპერაციები სრულდებიან როგორც ეს შემდეგ მაგალითებშია ნაჩვენები: (5+6) mod7=4, (5x6)mod7=2.

+	0	1
0	0	1
1	1	0

,

x	0	1
0	0	0
1	0	1

გაფართოებული გალუას ველი GF(2^m)

გაფართოებული გალუას ველი წარმოდგება გამოსახულებით GF(2^m), სადაც გამოსახულებაში 2^m რიცხვი 2 აღნიშნავს, რომ ხდება GF(2)={0,1} ველის გაფართოება ისე, რომ გაფართოებული ველის თითოეული ელემენტი შეიცავს m რაოდენობის ბიტს. მაგალითად, თუ m=2, GF(2^2) შეიცავს ორბიტიან მონაცემების 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 (ტეტრადებს), 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}. აგრეთვე ხშირად გამოიყენება გაფართოება $GF(2^{128})$.

პოლინომური წარმოდგენა

გაფართოებული გალუას ველის არითმეტიკა ეყრდნობა ბიტური მიმდევრობის პოლინომის სახით წარმოდგენას და ოპერაციებს პოლინომებზე. მაგალითად, GF(22) 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 $1*x^7+0*x^6+1*x^5+1*x^4+0*x^3+1*x^2+0*x^1+1*x^0=x^7+ x^5+ x^4+ x^2+1$.

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)=x3+x+1, m=4-თვის P(x)=x4+x+1, ხოლო m=8-თვის პოლინომი P(x)=x8+ x4+ x3+ x2+1.

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

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 Xor 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.

გამრავლება

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

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

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 განაყოფის ნაშთი:

$$\begin{array}{rrrrrrrl|rrrrr} x^4 & + & x^3 & + & x^2 & + & x & & x^3 & + & x & + & 1 \\ \cline{12-14} x^4 & + & x^2 & + & x & & & & x & + & 1 & & \\ \cline{1-7} & & x^3 & & & & & & & & & & \\ & & x^3 & + & x & + & 1 & & & & & & \\ \cline{3-7} & & 0 & + & x & + & 1 & \rightarrow & \multicolumn{4}{l}{11} \\ & & & & & & & & \multicolumn{4}{l}{\text{(ნაშთი)}} \end{array}$$

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, $011 \oplus 111 = 100$, 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 შევასრულოთ 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 გამოყენება მიზანშეწონილია ალგორითმების აპარატულად რეალიზაციისთვის. რაც შეეხება ხელით გამოთვლებს, გამრავლების შემთხვევაში ვსარგებლობთ 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) გაფართოებული ველებისთვის, თუმცა GF(2^8) შემთხვევაში ცხრილი შიცავს 2^8=256 სვეტს და სტრიქონს და გამოყენებისთვის მოუხერხებელია.

თავი 2. მონაცემთა ჰეშირება

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

2.1 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 ,H,

$$H(m) = h,$$

სადაც ჰეში, h არის m-ს სახე (emage), ხოლო m არის h-ის პირველსახე (preemage). როგორც ითქვა, m შეიძლება იყოს ნებისმიერი ზომის ბიტური მონაცემი 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 მოთხონაა ის, რომ იგი ცალმხრივი (one-way) ფუნქციაა: ნებისიმიერი m-თვის ადვილად (სწრფაად) ხდება მისი სახის (ანასახის) დადგენა, მაგრამ მოცემული h-ათვის მისი წინასახის დადგენა, ანუ $h=H^{-1}(m)$ გარდასახვის, განხორციელება რთულია.

2.2 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 მეორე m_2 მონაცემის პოვნა, რომლისთვისაც $H(m_2)=h$. ე.ი. ჰეშის ერთ მნიშვნელობას ორი განსხვავებული პირველსახე 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 ჰეშების 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;

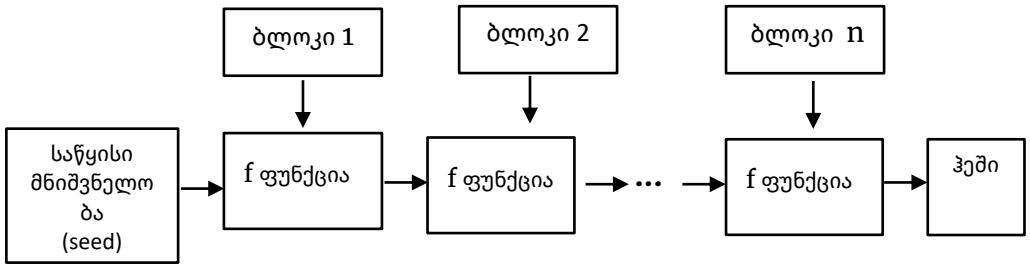
– ნებისმიერი m-თვის H(m)-ის გამოთვლა სწრაფი უნდა იყოს;

– ჰეშ ფუნქცია უნდა ხასიათდებოდეს ზვავისებური ეფექტით. ეს ნიშნავს, რომ თუ 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.

2.3 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 ერთერთი ალგორითმის გამარტივებული სქემა.



სურათი 1

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. შეიძლება ითქვას, რომ ეს ფუნქცია ასრულებს მონაცემთა ორი ბლოკის ერთ ბლოკად წარმოდგენის, ანუ შეყურსვის ოპერაციას.

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-ის (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 SHA256 ჰეშ ალგორითმი, რომელიც წარმოქმნის 256 ბიტიან ჰეშს და SHA-512 რომელიც წარმოქმნის 512 ბიტიან ჰეშს.

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

2.4. ჰეშირების გამოყენების მაგალითები

როგორც ზემოთ ითქვა, ჰეშირების ფუნქციის მახასიათებლებს შორის ერთ-ერთი უმთავრესია კოლიზიების არსებობა-არარსებობის შესაძლებლობა, როცა ორი განსხვავებული m_1 და m_2 მონაცემისთვის მიიღება ჰეშის ერთიდაიგივე მნიშვნელობა,

$$H(m_1) = H(m_2) = h,$$

სადაც H ჰეშირების ფუნქცია, ხოლო h თვით ჰეშის რიცხვითი მნიშვნელობაა. თეორიულად ნებისმიერი ფუნქცია შეიძლება წარმოქმნიდეს კოლიზიას, მაგრამ მისი პრაქტიკულად აღმოჩენა იმდენად რთული და მოცულობით გამოთვლებთან იყოს დაკავშირებული, რომ კოლიზიის ძიება აზრს კარგავდეს.

პრაქტიკულად წლების მანძილეზე, დამუშავებულ იქნა ჰეშ-ფუნქციების ოჯახები

SHA0, SHA1, SHA2, SHA3 და სხვა.

სადაც SHA (Sequre Hash Algorithm) აღნიშნავს უსაფრთხო ჰეშ-ალგორითმს. აქ თითოეული ოჯახი აერთიანებს რამდენიმე ჰეშ-ფუნქციას, რომლებიც დაფუძნებულია მერკელ-დამარდის კონსტრუქციაზე (Merkle–Damgard Construction), რომლის გამარტივებული ბლოკ-სქემა ნაჩვენებია სურათზე 1 -ზე.

SHA0 და SHA1 ოჯახებისთვის, კრიპტანალიზის საფუძველზე, აღმოჩენილ იქნა კოლიზიები და ამ მიზეზით, როგორც არასაიმედო, ისინი ამოღებულ იქნენ გამოყენებიდან. რაც შეეხება SHA2 და SHA3 ოჯახებს. ისინი დღეისათის კოლიზიებისგან თავისუფლად ითვლებიან და ფართო გამოყენებას პოულობენ კრიპტოგრაფიის სფეროში.

SHA2 ოჯახში წარმოდგენილი ფუნქციების მაგალითებია

SHA2-256, SHA2-512,

რომლებიც 256 ბიტიან და 512 ბიტიან ჰეშებს წარმოქმნიან, ხოლო SHA3 -ის მაგალითებია

SHA3-256, SHA3-512,

მაგალითად SHA3-256, უფრო საიმედოდ ითვლება, ვიდრე SHA2-256: ორივე წარმოქმნის 256 ბიტიან ჰეშს, მაგრამ განსხვავებული ალგორითმებით.

რამდენადაც 256-ბიტიანი და 512-ბიტიანი ჰეშების ორობითი ფორმით წარმოდგენა მოუხერხებელია, ამიტომ მათი წარმოდგენისთვის გამოიყენება თექვსმეტობითი ფორმა.

მაგალითად, მონაცემისათვის “Hello”გვექნება ჰეში

SHA2-256

(“Hello”)=2cf24dba5fb0a30e26w83b2dc5b9029e18161w5c1fa7425e780433b29389824

ეს ფუნქცია გამოიყენება კრიპტოვალუტა Bitcoin-ის ბლოკჩეინში.

SHA3 -ის ოჯახის განვითარების საფუძველზე დამუშავებულ იქნა ჰეშფუნქციების ახალი ოჯახი „Keccak”, რომელშიც გამოიყენება ჰეშფუნ-ქციების დამუშავების მიდგომა „Sponge Construction”. მაგალითად Keccak-256 წარმოადგენს SHA3 -ის მემკვიდრეს იგი გამოიყენება კრიპტოვალუტა Ethereum-ის ბლოკჩეინში. უნდა აღინიშნოს აგრეთვე ჰეშფუნქციები Shake-128 (msg, Length) Shake-256 (msg, Length), რომელთაც შეუძლიათ მოცემული მონაცემისათვის (msg) წარმოქმნან მოცემული სიგრძის (Length) ჰეში.

განვიხილოთ ჰეშფუნქციების გამოყენების რამდენიმე მაგალითი.

მაგალითი 1. პაროლების საიმედოდ დამახსოვრება.

მომხმარებლის პაროლი, როგორც წესი წარმოადგენს ღია ტექსტს და იგი ინახება პაროლების მონაცემთა ბაზაში როგორც ჰეში, H (password). სისტემაში პაროლის შეტანისას სისტემა გამოითვლის ღია ტექსტის ჰეშს და ამოწმებს, არის თუ არა იგი მონაცემთა ბაზაში. ეს საშუალებას იძლევა გაკონტროლდეს პაროლის ვალიდურობა.

მაგალითი 2. დოკუმენტების ჰეშირება.

თუ დოკუმენტები ინახებიან გარე მეხსიერების მოწყობილობაში ფაილების სახით, გამოითლება H (Document) და მიღებული ჰეში ინახება სათანადო მონაცემთა ბაზაში. ეს საშუალებას იძლევა გაირკვეს, ხომ არ მოხდა დოკუმენტში ან მის ჰეშში რაიმე სახის ცვლილება.

მაგალითი 3. ჰეშირების გამოყენება კრიპტოვალუტის სისტემების ბლოკჩეინებში.

ჰეშირება მრავალმხრივ გამოყენებას პოულობს კრიპტოვალუტის სისტემის ბლოკჩეინებში. ბლოკჩეინის მორიგი, ახალი ბლოკის ფორმირებისას სისტემაში გაერთიანებულ კვანძებს (მაინერებს) მიეცემათ შემდეგი კომბინატორული ამოცანა: მოცემული x-თვის იპოვეთ ისეთი რიცხვი p, რომ H(x‖p)-ს ორობითი მნიშვნელობის პირველი n ბიტი იყოს ერთიანი. კვანძს,

რომელიც პირველი ამოხსნის ამ ამოცანას, მიენიჭება ბლოკჩეინის ახალი, მომდევნო ბლოკის შექმნის პრიორიტეტი, სრულდება სამუშაო Proof of Work (PoW).

როგორც ბიტკოინი (Bitcoin), ისე ეთერიუმი (Ethereum) და სხვა კრიპტოვალუტის სისტემები წარმოადგენენ დეცენტრალიზებულ სისტემებს და მათში გაერთანებულ ნებისმიერ კვანძა შეუძლია განახორციელოს მაინერის (Miner) ფუნქცია, მაგრამ მათ განკარგულებაში არსებული გამოთვლითი სიმძლავრის მიხედვით განსხვავებული ალბათობით. შედარებით სუსტი კვანძების შემთხვევაში ისინი ეყრდნობიან კომპიუტერის ცენტრალურ პროცესორულ მოწყობილობას (Central Processing Unit, CPU) და გრაფიკულ პროცესორულ მოწყობილობებს (Graphic Processing Unit, GPU). მაგრამ მძლავრ ორგანიზაციებს შეუძლიათ გამოიყენონ მთლიანად აპარატურული მოწყობილობა ASIC (Application Specific Integrated Circuit), რომელიც დიდი სწრაფქმედებით ხასიათდება. ამ მიზეზით, იმისათვის, რომ შეიზღუდოს ქსელში პრივილეგიური კვანძების წარმოქმნა, კრიპტოვალუტის სისტემების ბლოკჩეინებში იყენებენ სპეციალიზებულ ჰეშირების ფუნქციებს. ეს ფუნქციები ისეა აგებული რომ მოითხოვენ დიდი ზომის ოპერატიულ მეხსიერებას, რისი ინტეგრალური სქემით განხორციელება იმდენად ძნელი და ძვირი საქმეა, რომ ეს აზრს კარგავს. ასეთ ჰეშირების ფუნქციებს ASIC - მედეგ ფუნქციებს უწოდებენ. ისინი საშუალებას იძლევიან PoW სამუშაოს შესრულების შესაძლებლობის ალბათობა შეძლებისდაგვარად გათანაბრდეს ქსელში ჩართული ყველა კვანძისათვის.

თავი 3. 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.

3.1 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.

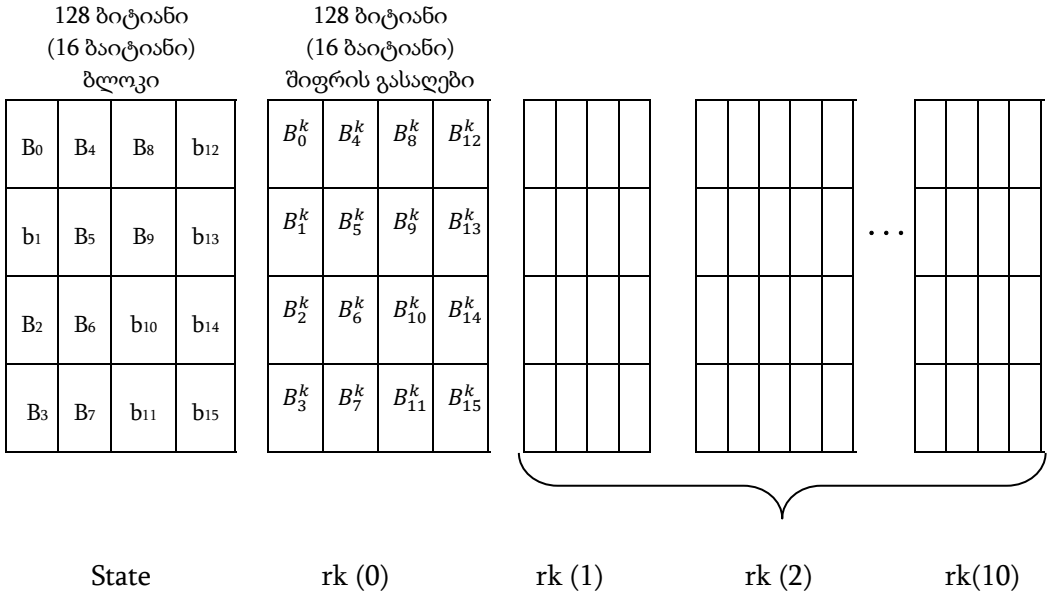
128 ბიტიანი გასაღების, K, ბაიტურად წარმოდგენისათვის საკმარისია ბიტების მიმდევრობა, დაწყებული მარცხნიდან, დავყოთ რვაბიტიან ქვეჯგუფებად. პირველი რვა ბიტი განსაზღვრავს შიფრის ბაიტს B_0^k, მომდევნო რვა ბიტი იძლევა B_1^k და ა. შ. ბოლო რვა ბიტი განსაზღვრავს B_{15}^k.

რაც შეეხება დასაშიფრავ გზავნილს, M (Message), ხდება მისი შემადგენელი სიმბოლოების შეცვლა ბიტური წარმოდგენით, მაგალითად, ASCII (American Standard Code for Informatoion Interchange) კოდის ან უნივერსალური კოდირების, უნიკოდის, 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 ზემოთაღწერილი 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

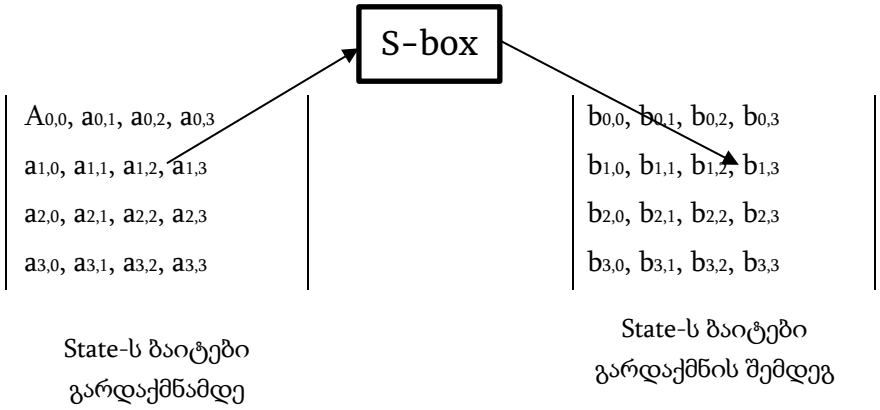
$$\text{State } (i,j) = \begin{vmatrix} S_{0,0}, S_{0,1}, S_{0,2}, S_{0,3} \\ S_{1,0}, S_{1,1}, S_{1,2}, S_{1,3} \\ S_{2,0}, S_{2,1}, S_{2,2}, S_{2,3} \\ S_{3,0}, S_{3,1}, S_{3,2}, S_{3,3} \end{vmatrix}$$

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.

3.2 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.



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 2

3.3 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.

A, b, c, d	⟶	a, b, c, d
e, f, g, h		f, g, h, e
i, j, k, l		k, l, i, j
m, n, o, p		n, o, p, m

სურათი 1 მასივის ელემენტების ციკლური ძვრა მარცნივ

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

3.4 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.**

3.5 გარდაქმნა AddRound Key (State, RoundKey)

მოცემული გარდაქმნა იყენებს ორ მონაცემს: მასივ State-ს და მასივ RoundKey. მოქმედება სრულდება ბაიტურად, როცა ხდება ბაიტ State (i, j)-ის ორის მოდულით 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**).

3.6 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 რაუნდები 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.

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.

მერე 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.

ფინალური მეათე რაუნდი სრულდება როგორც

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

როგორც ვხედავთ ფინალურ რაუნდში არ გამოიყენება გარდაქმნა MixColumn.

3.7 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. 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.

თავი 4. AES გალუას მთვლელის რეჟიმით

სიმეტრიული შიფრაციის მეორე მაგალითია AES გალუას მთვლელის რეჟიმით (AES with Galois Counter Mode, AES with GCM). მაშინ, როცა AES სისტემის ძირითადი მიზანია მონაცემთა გადაცემისას მისი კონფიდენციალობის უზრუნველყოფა დაშიფვრის გზით, AES with GCM ორიენტირებულია არა მხოლოდ კონფიდენციალობის, არამედ გადასაცემ მონაცემთა ავთენტობის (საიმედოდ გადაცემის) უზრუნველყოფაზე. ამგვარად, ეს სისტემა აერთიანებს ამ ორი მოთხოვნის ერთდროულად დაკმაყოფილებას, რაც განაპირობებს მის მაღალეფექტურობას.

სისტემა AES ხასიათდება შემდეგი მახასიათებლებით:

იგი წარმოადგენს ბლოკური შიფრაციის სისტემას, რომელშიც გამოყენებული სიმეტრიული შიფრაციის გასაღების დასაშვები სიგრძეებია 128 ბიტი, 192 ბიტი და 256 ბიტი. უფრო ხშირად გამოიყენება 128 და 256 ბიტის სიგრძის გასაღებები და აღწერებში აღინიშნება როგორც K (Key);

დასაშიფრავი მონაცემი P წარმოდგება როგორც 128 ბიტიანი ბლოკების მიმდევრობა $P_1, P_2, \ldots P_n^*$ და თუ მონაცემის სიგრძე არ არის 128-ს ჯერადი, მაშინ ხდება ბოლო P_n^* ბლოკის შევსება ნულოვანი ბიტებით მარჯვენა ბოლოდან ისე, რომ ბოლო ბლოკიც გახდეს 128 ბიტიანი. ამგვარად, ადგილი აქვს ტოლობას $P = P_1||P_2|| \cdots ||P_n^*$, სადაც || კონკატენაციის სიმბოლოა;

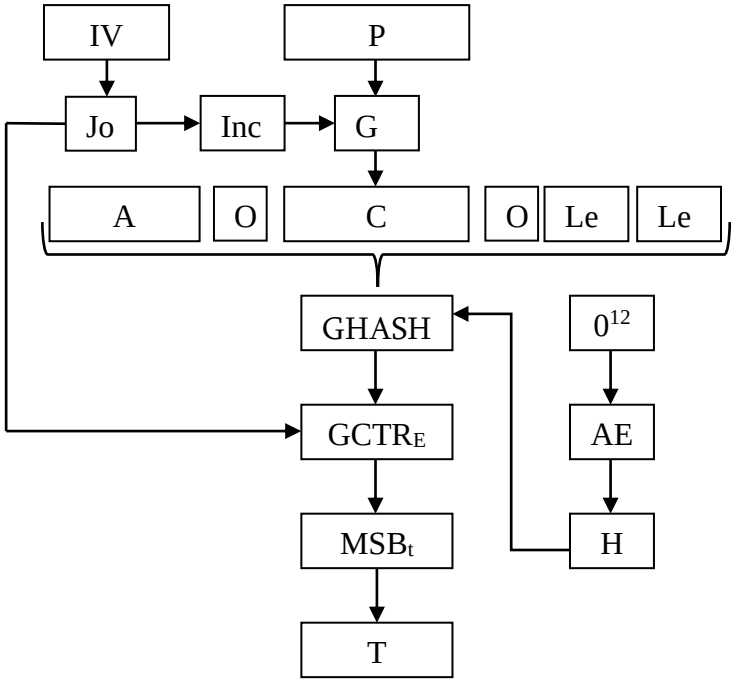
AES სისტემაში არაა გათვალისწინებული მონაცემთა გადაცემისას შემთხვევითი ან მიზანდასახული ჩარევით გამოწვეული შეცდომების აღმოჩენის მექანიზმი.

ქვემოთ მოყვანილია სისტემის აღწერა წყაროების D.McGrew, John Viega, The Galois/Counter Mode of Operation და M. Dvorkin, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-3880 გამოყენებით. ძირითადად მოყვანილია მხოლოდ სისტემის ფუნქციონირების გრაფიკული წარმოდგენა, რადგან სათანადო ალგორითმების აღწერა დიდ ადგილს დაიკავებდა.

გალუას მთვლელის რეჟიმი მოიცავს ორ ძირითად ფუნქციას: ავთენტური შიფრაცია (Authenticated Encription) და ავთენტური დეშიფრაცია (Authenticated Decryption). ეს ნიშნავს, რომ გვაქვს ორი კატეგორიის მონაცემი - ღია ტექსტი P, რომელიც ექვემდებარება შიფრაცია/დეშიფრაციის ოპერაციებს მონაცემთა კონფიდენციალობის უზრუნველყოფისათვის და დამატებითი მონაცემი A, (Aditional Authenticated Data), რომლის შიფრაცია/დეშიფრაცია არ

ხდება, მაგრამ ხდება მისი შემოწმება ავთენტობაზე. მონაცემთა ამგვარად გადაცემის მაგალითს წარმოადგენს უსაფრთხოება, როცა ინტერნეტში გადაცემისას პაკეტში ინკაფსულირებული მონაცემი იშიფრება, ხოლო პაკეტის სათაურში (Header) განთავსებული მონაცემები არ იშიფრება, რადგან ისინი გამოიყენებიან როგორც მმართველი მონაცემები ქსელურ აპარატურაში (მაგალითად მარშრუტიზატორებში) გადაადგილებისას.

სურათი 1 გვიჩვენებს ავთენტიკური შიფრაციის ალგორითმის ბლოკსქემას განზოგადებული სახით და ქვემოთ მოცემულია მისი შემადგენელი ელემენტების დაკონკრეტებული აღწერები.



სურათი 1. $GCM - AE(IV, P, A) = (C, T)$

აქ:

IV (The Initiation Vector) ინიციალიზაციის ვექტორია და შეიცავს რამდენიმე ველს, რომლებიც მიუთითებენ საჭირო მონაცემებზე, რაც შეიძლება სხვადასხვანაირად შეიქმნას, მაგრამ გავრცელებული სტანდარტების რეკომენდაციების თანახმად იგი უფრო ხშირად შედგება 96 ბიტიანი სტრიქონისგან;

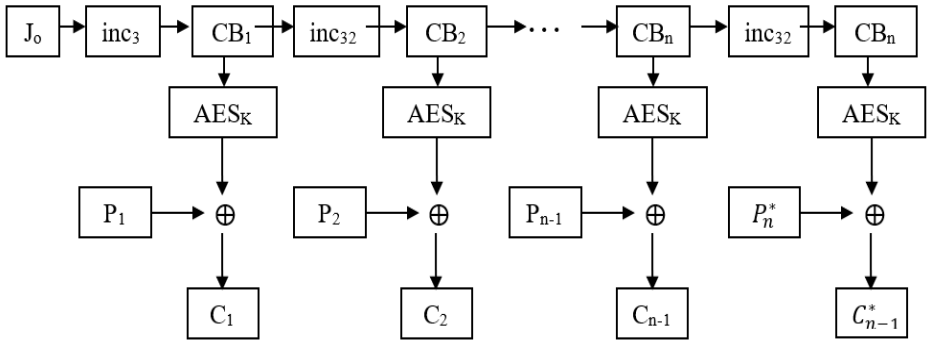
ბლოკი J_0 წარმოადგენს 128 ბიტიან მონაცემს და იგი წარმოიქმნება IV მონაცემის 96 ბიტის გაფართოებით 128 ბიტიან ბლოკამდე შემდეგი ფორმულის გამოყენებით:

$J_0 = IV \,||0^{31}||1;$

ეს ნიშნავს, რომ ხდება შემადგენელი 96 ბიტის კონკატენაცია 31 ნულისგან შემდგარ ბიტურ სტრიქონთან, რომელსაც მოყვება ბიტი 1. საბოლოოდ მიიღება 128 ბიტიანი ბლოკი. როგორც სურათი 1 გვიჩვენებს, ეს მონაცემი გამოიყენება, როგორც შიფრაციის, ისე აუტენტიფიკაციის პროცედურებში.

კონფიდენციალური მონაცემის შიფრაცია

ამ ფუნქციის განხორციელების შედეგად დასაშიფრავი ტექსტისგან მიიღება შიფრტექსტი, როგორც ამას სურათი 2 გვიჩვენებს.



სურათი 2. $GCTR_k\ (J_o, X_1\ ||X_2 \cdots X_n^*) =$ C1|| C2|| … $||C_n^*$

აქ $CB_1, CB_2, \ldots, CB_{n-1}, CB_n$ აღნიშნავენ 128 ბიტიან მთვლელების მნიშვნელობების მიმდევრობას, სადაც CB_i-ს მნიშვნელობა მიიღება წინმსწრები CB_{i-1}-ის მნიშვნელობისგან inc_{32} ფუნქციის გამოყენებით. ეს ფუნქცია ახდენს მთვლელის მნიშვნელობის გაზრდას ანუ ინკრემენტაციას (Incrimentation) შემდეგნაირად:

128 ბიტიანი CB_{i-1}-ის ბიტების მიმდევრობა დავყოთ ორ ქვემიმდევრობად, σ_1 და σ_2. მიმდევრობა σ_1მოიცავს 96 ბიტს, დაწყებული მარცხნიდან, ხოლო σ_2 მოიცავს ბოლო 32 ბიტს. ამგვარად, $CB_{i-1} = \sigma_1||\sigma_2$;

წარმოვიდგინოთ σ_2-ში შემავალი ბიტები როგორც ორობითი რიცხვი და ვიპოვოთ მისი შესაბამისი ათობითი რიცხვი x;

ვიპოვოთ $x^1 = (x+1) \bmod 2^{32}$;

წარმოვადგინოთ x^1 როგორც 32 ბიტიანი ორობითი რიცხვი, რასაც შეესაბამება ბიტების მიმდევრობა σ_2';

CB_i, ანუ მომდევნო მთვლელის მნიშვნელობა იქნება:

$$CB_i = \sigma_1||\sigma_2';$$

როგორც ვხედავთ, ამ გამოთვლებში σ_1 არ იცვლება,

სურათი 2 გვიჩვენებს, რომ ხდება თითოეული მთვლელის მონაცემის დაშიფვრა ფუნქცია AES_k გამოყენებით, სადაც AES შიფრია, ხოლო K სიმეტრიული შიფრაციის გასაღები. ამის შემდეგ ხდება მიღებული მონაცემის შეკრება $P = P_1||P_2||\cdots||P_n^*$-ის შემადგენელ ბლოკებთან ორობითი ანუ X_{02} არითმეტიკის გამოყენებით. საბოლოოდ მიიღება $P = P_1, P_2, \ldots, P_n$ ღია ტექსტის ბლოკების შესაბამისი შიფრტექსტების მიმდევრობა $C_1, C_2, \ldots, C_n$. საბოლოოდ ხდება გარდაქმნა.

$$P = (P_1, P_2, \cdots, P_n^*) \rightarrow C(C_1, C_2, \cdots, C_n^*)$$

მონაცემთა ავთენტურობის კონტროლი

ზემოთ, სურათი 2, განხილულ იქნა კონფიდენციალური მონაცემის $P = P_1||P_2||\cdots||P_n^*$ შიფრაციის ალგორითმი, რაც AES GCM სისტემის ორიდან ერთერთი ფუნქციაა. ეხლა განვიხილოთ ამ სისტემის მეორე ფუნქცია, რაც მონაცემების ავთენტურობის შემოწმებაში მდგომარეობს. როგორც სურათი 1 გვიჩვენებს, ამაში მონაწილეობს $GHASH_H$ ფუნქცია. ეს ფუნქცია შესასვლელზე ღებულობს, როგორც დამატებით, არაავთენტიკურ მონაცემს A, ისე ავთენტიკურ მონაცემ P -ს შიფრს, C, და სხვა მონაცემებს. რეალურად მონაცემი A წარმოდგება როგორც 128 ბიტიანი ბლოკების მიმდევრობა $A = A_1||A_2||\cdots||A_m^*$, სადაც m ბლოკების რაოდენობაა, ხოლო ბოლო ბლოკი A_m^* შეიძლება იყოს არასრული. ანალოგიურად, $C = (C_1||C_2||\cdots||C_n^*)$ შიფრბლოკების მიმდევრობის ბოლო C_n^* ბლოკი შეიძლება იყოს არასრული.

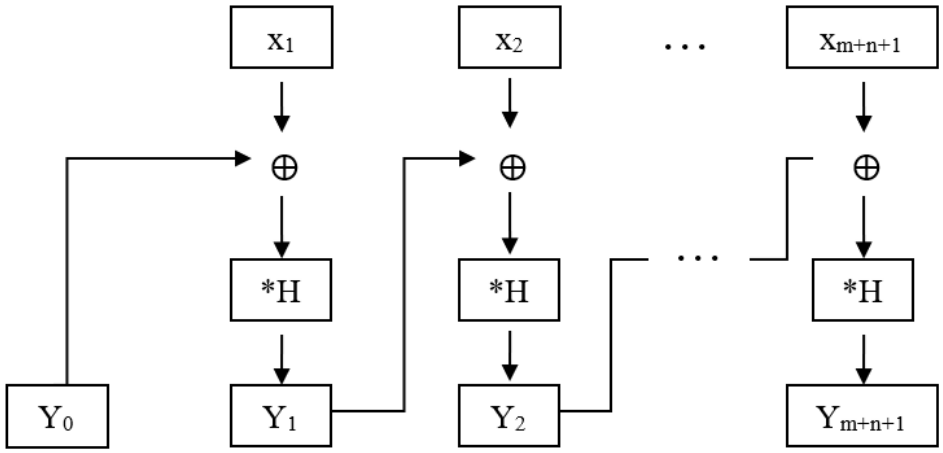
რაც შეეხება A და C მონაცემების სიგრძეებს len(A) და len(C) (A და C მონაცემების სიგრძეებს ბიტებში), ისინი წარმოდგებიან 64 ბიტიანი მონაცემებით და ერთობლიობაში ქმნიან ერთ 128 ბიტიან ბლოკს. ამგვარად, ეს მონაცემები შეიძლება წარმოვიდგინოთ როგორც 128 ბიტიანი ბლოკების კონკატენაცია.

თუ მოვახდენთ აქ წარმოდგენილ ბლოკების გამჭოლ გადანომვრას და მიღებულ მიმდევრობაში i-ურ ბლოკს ავღნიშნავთ სიმბოლოთი x_i , გვექნება მიმდევრობა

$$\underbrace{A_1||\ A_2||\ \ldots\ ||A_m^*}_{\text{m ბლოკი}}\ \underbrace{||\ C_1||\ C_2||\ \ldots\ ||C_n^*}_{\text{n ბლოკი}}\ \underbrace{||\ \text{len(A)}\ ||\ \text{len(C)}}_{\text{1 ბლოკი}}$$

$$x_1, x_2, \cdots x_m, x_{m+1}, \cdots x_{m+n}, x_{m+n+1}$$

ამ მონაცემების ავთენტურობის შესამოწმებლად გამოიყენება $GHAS_H$ ფუნქცია, რომელიც ნაჩვენებია სურათი 3-ზე.



სურათი 3. $\mathrm{GHASH}_H(x_1, x_2, \cdots x_{m+n+1}) = Y_{m+n+1}$

აქ აღწერილი პროცედურის საწყის მონაცემებს წარმოადგენს ზემოთ აღწერილი მონაცემები $x_1, x_2, \cdots x_{m+n+1}$ და Y_0, რომელიც წარმოადგენს 128 ნულოვანი ბიტისაგან (0^{128}) შემდგარ ბლოკს, იტერაციული გამოთვლების საწყის მონაცემს, ხოლო ფუნქციის შედეგი არის Y_{m+n+1}. ბლოკი H, როგორც სურათი 1 გვიჩვენებს, მიიღება ნულოვანი ბლოკის (0^{122}) დაშიფვრით AESk შიფრის გამოყენებით. ოპერაცია $\oplus$ აღნიშნავს ორი ბლოკის შეკრებას X_{or} არითმეტიკის გამოყენებით, ხოლო * აღნიშნავს გაფართოებული გალუას ველის ($GF(2^{128})$) ორი ელემენტის გამრავლებას გალუას ველის არითმეტიკის გამოყენებით.

ამგვარად, $\mathrm{GHASH}_H(x_1, x_2, \cdots x_{m+n+1})$ იძლევა მონაცემის ავთენტურობის შემოწმების საშუალებას, რომელიც ეყრდნობა როგორც დამატებით, არაკონფიდენციალურ მონაცემებს A, ისე კონფიდენციალურ მონაცემების შიფრს C-ს.

GHASH წარმოქმნის მაკონტროლებელ მონაცემს, რომელიც კრიპტოგრაფიული ჰეშის ანალოგია, მაგრამ ამ უკანასკნელიდან განსხვავებით, ხდება მისი დამატებითი დამუშავება, როგორც ამას სურათი 1 გვიჩვენებს. კერძოდ ხდება მისი დამატებითი გარდაქმნა GCTRk ფუნქციის გამოყენებით. მიღებული 128 ბიტიანი ბლოკის მარცხნიდან პირველი t ბიტი გამოიყენება როგორც მონაცემთა ავთენტურობის საბოლოო მაჩვენებელი და აღინიშნება სიმბოლო T (Tag, ჭდე, იარლიყი). საბოლოოდ, ზემოთ აღწერილი ავთენტიკური დეშიფრაციის ფუნქცია განისაზღვრება როგორც

$$GCM - AE(IV, P, A) = (C, T)$$

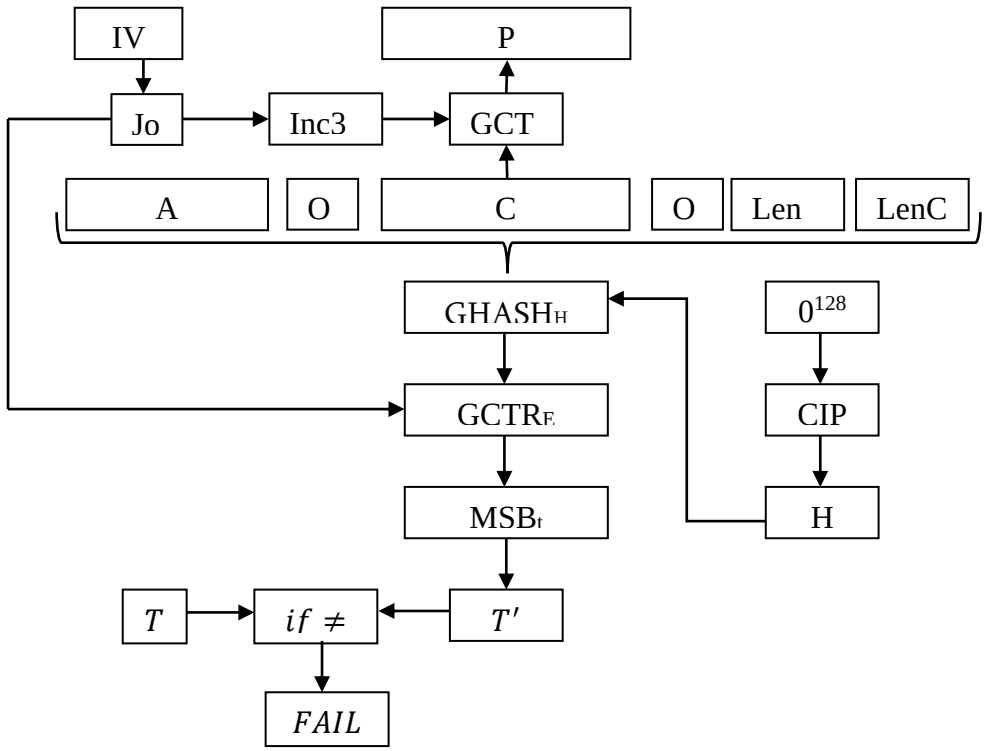
ეს ორი მონაცემი, C და T, გადაეცემა კომუნიკაციაში მონაწილე მონაცემთა მიმღებ მხარეს, სადაც სრულდება დეშიფრაციის ოპერაცია.

დეშიფრაცია

დეშიფრაციის ფუნქციის გრაფიკული წარმოდგენა მოცემულია სურათ 4-ზე, რომელიც აღწერს $GCM - AD_k(IV, C, A, T) = P$ ან $FAIL$ ფუნქციას. ეს ნიშნავს, რომ გადაცემისას შეცდომის აღმოჩენის შემთხვევაში ფიქსირდება მტყუნების მდგომარეობა, FAIL, ხოლო წინააღმდეგ შემთხვევაში დიქსირდება ღია ტექსტი P.

როგორც სურათი 1 და სურათი 4 შედარება გვიჩვენებს, $GHASH_K$ გამოთვლა ორივე შემთხვევაში მსგავსად ხდება. რაც შეეხება სხვაობას შიფრაციისა დეშიფრაციას შორის ეს ემყარება X_{or} არითმერტიკის შემდეგ თვისებას:

$AES_K \oplus P_i = C_i$ და $AES_K \oplus C_i = P_i$



სურათი 4. $GCM - AD_k(IV, C, A, T) = P_{or} FAIL$

მართლაც, $\oplus$ შეკრების ოპერაციის შემთხვევაში თუ $AES_K \oplus P_i = C_i$, მაშინ

$$AES_K \oplus C_i = AES_K \oplus (AES_K \oplus P_i) = (AES_K + AES_K) \oplus P_i = 0 \oplus P_i = P_1$$

(ეს გამომდინარეობს X_{or} არითმეტიკის თვისებიდან, რომ ნებისმიერი A თვის $A \oplus A = 0$. მაგალითად, 101 ⊕ 101=000).

თავი 5. ასიმეტრიული კრიპტოგრაფია: დიფი-ჰელმანის ალგორითმი

დიფი-ჰელმანის (Whit Diffie, Martin Hellman) ალგორითმი წარმოადგენს ასიმეტრიული (ღია გასაღებით) კრიპტოგრაფიის ერთ-ერთ პროტოკოლს, რომელიც ეყრდნობა მერკლის (Ralph Mercle) მიერ შესრულებულ წინმსწრებ კვლევებს და ბევრი მკვლევარის აზრით იგი უნდა იწოდებოდეს როგორც დიფი-ჰელმან-მერკლის პროტოკოლი. იგი გამოყენებულ იქნა 1976 წელს და საშუალებას იძლევა ქსელურ კომუნიკაციაში მონაწილე მხარეებმა საიმედოდ წარმოქმნან ზიარი კრიპტოგრაფიული გასაღები არასაიმედო (დაუშიფრავი) საკომუნიკაციო არხის გამოყენებით. ეს პროტოკოლი განსაკუთრებულად ფართე გამოყენებას პოულობს უკანასკნელ პერიოდში, რადგან იგი საშუალებას იძლევა მონაცემთა გადაცემისას მიღწეულ იქნეს ე.წ. სრულყოფილი ფორვარდული საიდუმლოება (Perfect Forward secrecy, PFS).

5.1 მოდულურ ახარისხებაზე დაფუძნებული დიფი-ჰელმანის ალგორითმი

ციფრული ლოგარითმის პრობლემა (Digital Logarithm Problem, DLP)

ეს პრობლემა უკავშირდება გამოთვლებს, რომლებიც დაკავშირებულია გამოსახულებასთან

$$a = b^n mod p,$$

სადაც a, b და n მთელი დადებითი რიცხვებია, ხოლო p მარტივი რიცხვია. თუ ცნობილია b, n და p, მაშინ a -ს გამოთვლა „იოლი“ (easy) ამოცანაა, რადგან დამუშავებულია მისი გამოთვლის ეფექტური ალგორითმები, რომლებიც განხორციელებულია კრიპტოგრაფიულ ბიბლიოთეკებსა და online კალკულატორებში და შეიძლება გამოყენებულ იქნეს p-ს დიდი მნიშვნელობის შემთხვევებში. მაგალითად, მოდულური ახარისხების კალკულატორში (Online Modular Exponentiation Calculator). იმ შემთხვევაში, თუ მოცემულია a, b, p და უნდა განისაზღვროს n, დღეისათვის არ არსებობს ამ ამოცანის თანამედროვე კომპიუტერზე ეფექტურად ამოხსნის ალგორითმი და იგი ცნობილია როგორც დისკრეტული ლოგარითმის პრობლემა (Discrete Logaritm Propblem).

5.2 P მარტივი რიცხვის პრიმიტიული ფესვი მოდულით P

დიფი-ჰელმანის ალგორითმის გამოყენებისას იგულისხმება, რომ კომუნიკაციაში მონაწილე ორივე მხარისთვის ცნობილია ორი მონაცემი: p და g. p წარმოადგენს მარტივ რიცხვს, გამოიყენება როგორც მოდული გამოთვლებში და განსაზღვრავს სიმრავლეს $F_p = \{1, 2, \dots, P-1\}$, რომელიც ქმნის მულტიპლიკაციურ ჯგუფს. ეს ნიშნავს, რომ თუ a, $b \in F_p$, მაშინ $(a.b) \bmod_p \in F_p$. რიცხვი g ახარისხების ოპერაციაში წარმოადგენს ფუძეს და მას p-ს პრიმიტიული ფესვი (root) ეწოდება. პრიმიტიული ფესვი არის ისეთი $r \in F_p$, რომელიც მიმდევრობით x = 0, 1, 2, …, P-2 ხარისხებში აყვანისას წარმოქმნის F_p სიმრავლეს. მაგალითად, P=5 შემთხვევაში $F_5 = \{1, 2, 3, 4\}$ და თუ ამ სიმრავლის თითოეულ ელემენტს ავიყვანთ x = 0, 1, 2, 3 ხარისხებში, გვექნება

r=1	r=2	r=3	r=4
$1^0 \ mod \ 5 = 1$	$2^0 \ mod \ 5 = 1$	$3^0 \ mod \ 5 = 1$	$4^0 \ mod \ 5 = 1$
$1^1 \ mod \ 5 = 1$	$2^1 \ mod \ 5 = 2$	$3^1 \ mod \ 5 = 3$	$4^1 \ mod \ 5 = 4$
$1^2 \ mod \ 5 = 1$	$2^2 \ mod \ 5 = 4$	$3^2 \ mod \ 5 = 4$	$4^2 \ mod \ 5 = 1$
$1^3 \ mod \ 5 = 1$	$2^3 \ mod \ 5 = 3$	$3^3 \ mod \ 5 = 2$	$4^3 \ mod \ 5 = 4$

როგორც ეს მაგალითი გვიჩვენებს ახარისხებისას r=1 და r=4 არ წარმოქმნიან F_5 სიმრავლის ყველა ელემენტს, ხოლო r=2 და r=3 წარმოქმნიან. ეს ნიშნავს, რომ r=2 და r=3 არიან p=5 მარტივი რიცხვის პრიმიტიული ფესვები და ნებისმიერი მათგანი შეიძლება გამოყენებულ იქნეს როგორც ფუძე ახარისხების ოპერაციისას. ეს ნიშნავს, რომ გამოთვლებში შეიძლება გამოვიყენოთ როგორც g=2, ისე g=3. P-ს დიდი (პრაქტიკული) მნიშვნელობისთვის შეიძლება ვისარგებლოთ p-ს პრიმიტიული ფესვების ჩამომთვლელი online კალკულატორით (Primitive Root Online Calculator). განხილული მაგალითის შემთხვევაში დიფი-ჰელმანის ალგორითმის გამოყენებისათვის ორთავე მხარეს უნდა ჰქონდეს შეთანხმებული მონაცემი, (p, g) = (5, 2) ან (p, g) = (5, 3).

5.3 დიფი-ჰელმანის ალგორითმი

დავუშვათ ალისას და ბობს სურთ საიმედოდ გაცვალონ დიდი ზომის მონაცემი დაშიფრულად სიმეტრიული შიფრაციის სისტემის, მაგალითად AES-ის გამოყენებით. ამისათვის მათ საიმედოდ უნდა წარმოქმნან სიმეტრიული საიდუმლო კრიპტოგრაფიული გასაღები არასაიმედო (დაუშიფრავი) საკომუნიკაციო არხის გამოყენებით, რომელსაც შეიძლება აკვირდებოდეს ბოროტი განზრახვის მქონე მესამე პირი. ამის საშუალებას იძლევა დიფი-ჰელმანის ალგორითმი, რომელიც მდგომარეობს შემდეგი ნაბიჯების განხორციელებაში:

1. ალისა და ბობი ორივე იყენებს ერთსა და იმავე (p, g) წყვილს, რომელიც შეიძლება საჯაროდ ცნობილი (არასაიდუმლო) იყოს. აქ p არის დიდი მარტივი რიცხვი, g არის p-ს პრიმიტიული ფესვი მოდულით p, რომელ;იც წარმოქმნის სიმრავლე Fp-ს.
2. ალისა შემთხვევითად ირჩევს ერთ-ერთ რიცხვს a ∈ Fp და განიხილავს მას როგორც თავის საიდუმლო გასაღებს. ანალოგიურად, ბობი ირჩევს რიცხვ b და განიხილავს მას როგორც თავის საიდუმლო გასაღებს.
3. ალისა გამოთვლის $A = g^a \ mod \ p$, რომელიც წარმოადგენს მის ღია გასაღებს და უგზავნის მას ბობს. ანალოგიურად, ბობი გამოთვლის თავის ღია გასაღებს $B = g^b \ mod \ p$ და უგზავნის მას ალისას.
4. ალისა გამოთვლის $S = B^a \ mod \ p = (g^b mod \ p)^a \ mod \ p = g^{ab} \ mod \ p.$ ბობი გამოთვლის $S = A^b \ mod \ p = (g^a mod \ p)^b \ mod \ p = g^{ab} \ mod \ p.$ ამგვარად, ორივე მხარე საიდუმლოდ ინახავს S როგორც საერთო საიდუმლო კრიპტოგრაფიულ გასაღებს.

რამდენადაც ზემოთგანხილული მონაცემების გაცვლა ხდებოდა არასაიმედო (დაუშიფრავი) კავშირის გამოყენებით, ბოროტგამზრახველს შეეძლო საკომუნიკაციო არხზე დაკვირვებით გაეგო A და B და განესაზღვრა საიდუმლო გასაღებები ტოლობებიდან $A = g^a \ mod \ p$ და $B = g^b mod \ p$. მაგრამ, რადგან ცნობილი A, g და p მნიშვნელობებისთვის $A = g^a \ mod \ p$ ტოლობიდან a-ს განსაზღვრა p-ს დიდი მნიშვნელობისთვის პრაქტიკულად მიუღებლად დიდ დროს მოითხოვს და ეს მიდგომა აზრს კარგავს. იგივე პრობლემა წარმოიქმნება ბობის შემთხვევაშიც.

შედეგად, ალისამ და ბობმა წარმოქმნეს კრიპტოგრაფიული გასაღები მისი არასაიმედო საკომუნიკაციო არხში გადაცემის გარეშე!

ამგვარად, ალისა და ბობს აქვთ ზიარი კრიპტოგრაფიული გასაღები, რომელიც სხვადასხვანაირად შეიძლება იქნეს გამოყენებული. თუ, მაგალითად მონაცემთა დაშიფვრისას განზრახულია AES კრიპტოგრაფიული სისტემის

გამოყენება 128 ბიტიანი სიმეტრიული გასაღებით და S-ის ბიტური განზომილება არის 128 ბიტი, მაშინ S პირდაპირ შეიძლება გამოყენებულ იქნეს როგორც სიმეტრიული გასაღები. წინააღმდეგ შემთხვევაში შეიძლება განისაზღვროს, მაგალითად, 256 ბიტიანი ჰეში ფუნქციით SHA256 და მისი პირველი 128 ბიტი გამოყენებულ იქნას როგორც სიმეტრიული გასაღები.

თავი 6. ასიმეტრიული კრიპტოგრაფია: RSA

6.1 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 სისტემებისაგან, სადაც როგორც შიფრაცია, ისე დეშიფრაცია ხდება ერთი საიდუმლო გასაღების (Private Key) გამოყენებით, RSA კრიპტოსისტემაში გამოიყენება ორი გასაღები: ღია გასაღები (Public Key) და ფარული (Private Key). ღია ანუ საჯარო გასაღები ადვილად ხელმისაწვდომია, ხოლო ფარული გასაღები საიდუმლოდ უნდა ინახებოდეს.

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 $0 \le m < n$, ადგილი აქვს ტოლობას

$(m^e)^d \equiv m \pmod{n}$,

სადაც ≡ სადარობის 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 სპეცია-

ლიზებული online კალკულატორები, რომლებიც საშუალებას იძლევიან შემოწმდეს რომელიმე დასახელებული რიცხვი მარტივობაზე (Primality Testing). 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) $\lambda(n)$-ის მნიშვნელობა n-თვის. ჩვენი შემთხვევისთვის როცა $n=p\cdot q$, სადაც p და q მარტივი რიცხვებია, ეს ფუნქცია ადვილად გამოითვლება,

$$\lambda(n) = LCM(p-1, q-1),$$

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 $\lambda(n)$, შეიძლება განისაზღვროს e და d. e შეიძლება იყოს რიცხვი რომელიც აკმაყოფილებს პირობებს:

$e < \lambda(n)$ და e და $\lambda(n)$ ურთიერთმარტივია.

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

$$e\cdot d = 1 \mod \lambda(n)$$

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

(e, n),

ხოლო ფარულ გასაღებს წყვილი

(d, n).

ცხადია, p, q და $\lambda(n)$ 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 ფარული 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 მარტივობაზე 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) ურთიერთმარტივობაზე. ვისარგებლოთ კალკულატორით Coprime Calculator.

5. ვიპოვოთ e-ს მოდულური მულტიპლიკაციური ინვერსია 29d≡1mod9900. ვისარგებლოთ კალკულატორით Modular Multiplicative Inverse Calculator. d=7169.

6. დავშიფროთ ASCII ასო, რომლის ათობითი ნომერია 49. გამოვთვალოთ შიფრი m=49-თვის. გვექნება $c=49^{29}$mod20099. ვისარგებლოთ კალკულატორით Modular exponentiation. C=16540.

7. მოვახდინოთ დეშიფრაცია. $m\equiv c^{d}$mod20099. ვისარგებლოთ Modular exponentiation კალკულატორით. **m=49**.

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

6.2 RSA კრიპტოსისტემის გამოყენების მაგალითი

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

შევადგინოთ B-ს ღია და ფარული 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 (Modulus).

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 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 მულტიპლიკაციური 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 ფარული (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 $0 \le m < n$, 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, $c=m^e \mod n$	11	62	20	20	53	

განვიხილოთ, მაგალითად სიმბოლო H. მისთვის გვაქვს m=72, ხოლო ამ სიმბოლოს შიფრი არის $c=m^e \mod n = 72^5 \mod 91=11$. სიმბოლო E-სთვის გვექნება $c=m^e \mod n = 69^5 \mod 91=62$ 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 ფარული გასაღების (d, n) = (5, 91) გამოყენებით, როგორც ეს სურათზეა ნაჩვენები:

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

აქ გამოთვლები წარმოებს ფორმულით $m=c^d \mod n$.

შიფრ 11-თვის გვექნება $m=11^5 \mod 91= 72$ (ანუ H), 62-თვის გვექნება $m=62^5 \mod 91= 69$ (ანუ E) და ა.შ.

ცხადია m-ის შესაძლო მნიშვნელობაა $m<n$. მაგალითად, თუ შიფრის გასაღები n იქნებოდა დიდი რიცხვი, მაშინ რამდენიმე სიმბოლოს ნომრები შეგვეძლო ერთობლივად წარმოგვედგინა როგორც ერთი 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 ის, რომ იგი მნიშვნელოვნად ამარტივებს გასაღებების მართვას.

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

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

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 ავთენტიკაციისა და ციფრული ხელმოწერის ალგორითმებში. (ANDREW S. TANENBAUM, NICK FEAMSTER, DAVID WETHERALL, 2021)

თავი 7. 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 ფარულ გასაღებებს და ხასიათდება არანაკლები საიმედოობით. (Corbellini, 2015)

7.1 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

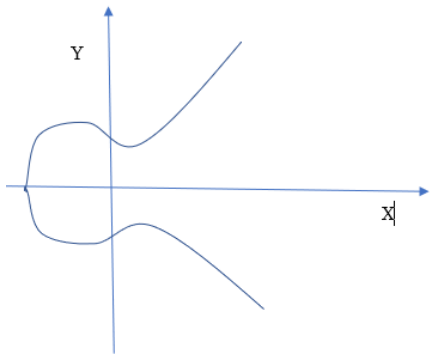
$$y^2 = x^3 + ax + b \text{ და}$$

$$D = 4a^3 + 27 \cdot b^2 \neq 0 \text{ უტოლობით}$$

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

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

სურათი1 გვიჩვენებს ელიპტიკური წირის გრაფიკებს a და b კოეფიციენტების (პარამეტრების) სხვადასხვა მნიშვნელობებისთვის.



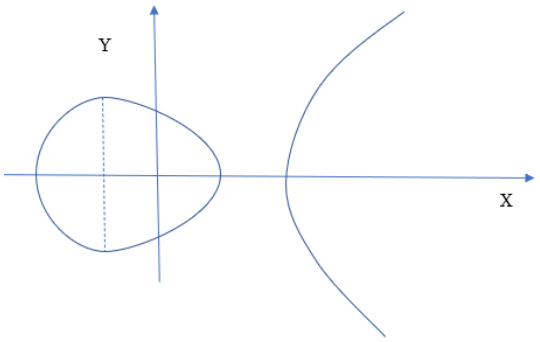
$$y^2 = x^2 - x + 1$$

a=-1, b=1

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

D>0 (ვალიდური წირი)

ა)



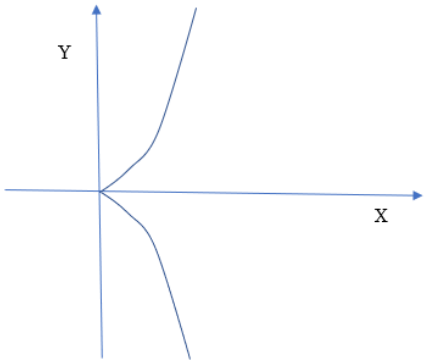
$$y^2 = x^3 - 3x + 1$$

a=-3, b=1

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

D<0 (ვალიდური წირი)

ბ)



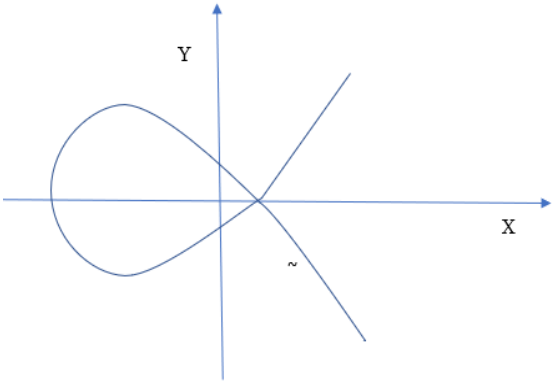
$$y^2 = x^3$$

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 სიმეტრიულია 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 ერთად იზრდება დადებით და უარყოფით უსასრულობამდე. წირის კრიპტოგრაფიული ვარგისიანობისთვის მოითხოვება, რომ მის ნებისმიერ წერტილში შესაძლებელი იყოს მხების ცალსახად გატარება,რაც გამოისახება პირობებით $D>0$ ან $D<0$. რაც შეეხება შემთხვევას (გ), აქ $D = 0$ და წირი კრიპტოგრაფიისტვის უვარგისია, რადგან (0, 0) წერტილში მხების ცალსახად გატარება შეუძლებელია და ეს წერტილი წარმოადგენს განსაკუთრებულ, სინგულარულ წერტილს, რის გამოც თვით წირი გამოუყენებადია. ანალოგიური სიტუაცია გვაქვს დ) შემთხვევაში. აქაც დისკრიმინანტი $D = 0$ და სინგულარობა გამოიხატება იმით, რომ შტოები იკვეთებიან (1, 0) წერტილში და წირისადმი მხების ცალსახად გატარება შეუძლებელია.

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

წერტილების შეკრების გეომეტრიული მეთოდი

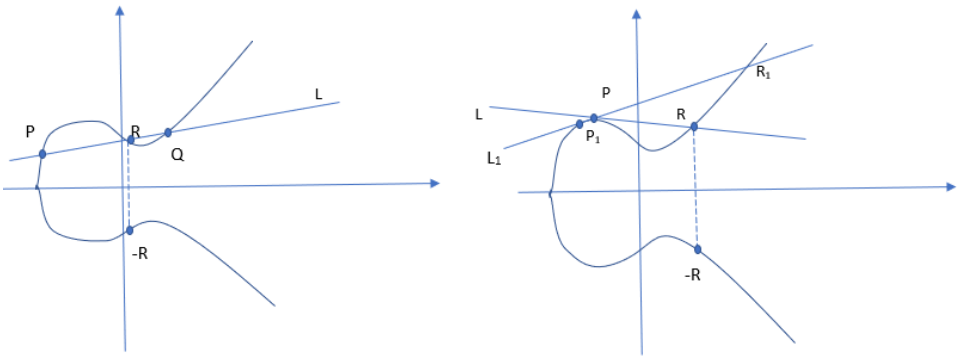
ელიპტიკურ წირზე დაფუძნებულ კრიპტოგრაფიაში ძირითადია წირის ორი წერტილის შეკრების ოპერაცია, რომელიც აღინიშნება + (შეკრების) სიმბოლოთი, თუმცა რადიკალურად განსხვავდება სხვა კონტექსტებში გამოყენებული შეკრების ოპერაციებისაგან.

განვიხილოთ შეკრების ოპერაცია სურათი2-ზე 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

სურათი2 ა)-ზე განხილულია 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 წერტილი P -სკენ, 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 ვიპოვოთ 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.

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

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

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 $n \cdot P$.

7.2 ელიპტიკური კრიპტოგრაფია სასრული ველის გამოყენებით

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 $Fp=\{0,1,2,\ldots,p-1\}$ ველის შემთხვევაში ცვლადები ღებულობენ მნიშვნელობებს

სასრული სიმრავლიდან და გამოთვლები წარმოებს მოდულით P (modP). ამგვარად, ზემოთ მოყვანილი გამოსახულებები მიიღებენ სახეს:

$\{(x, y) \in F^2p , | y^2 \equiv x^3 + ax + b \pmod{p}\} \cup \{0\}$,

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

აქ ≡ სადარობის სიმბოლოა, რაც ნიშნავს, რომ ორი გამოსახულება სადარია, 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

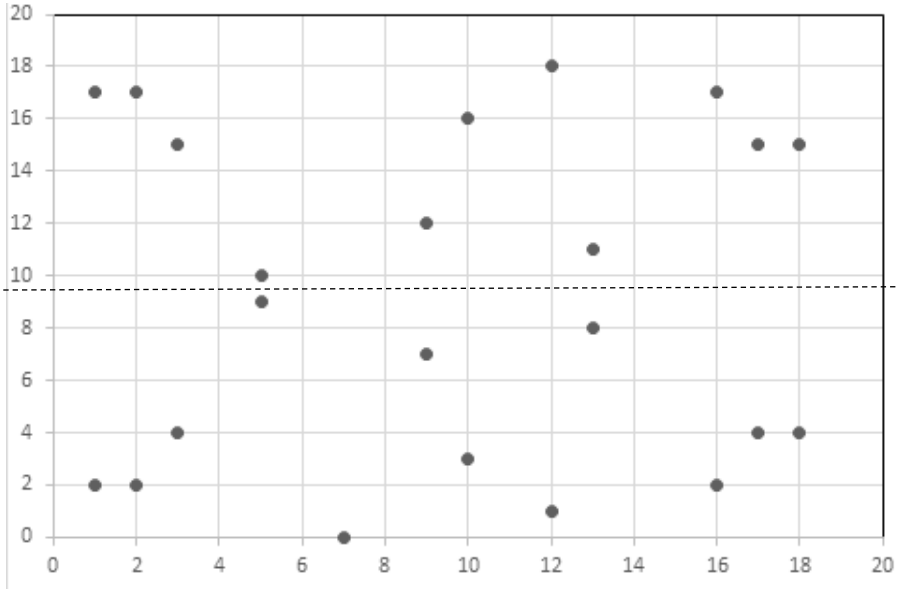
$4 \cdot 2^3 + 27 \cdot 1^2 \not\equiv 0 \pmod{5}$ ე. ი. $(4 \cdot 2^3 + 27 \cdot 1^2) \bmod 5 \not\equiv 0 \bmod 5$,

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

მოყვანილი გამოთვლები გვიჩვენებენ, რომ წირი $y^2 \equiv x^3 + 2 \cdot x + 1$ არ შეიცავს სინგულარულ წერტილს 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 $(P-1)\ (P-1) = 18 \cdot 18 = 324$.



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

განვიხილოთ ამ სიმრავლეზე განმარტებული ელიპტიკური ფუნქცია

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

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) $y^2 \equiv x^3 - 7x + 10 \pmod{19}$ წირს. ჩავატაროთ გამოთვლები:

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

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

$16 = 16$ (ეკუთვნის)

რაც შეეხება წერტილს (2, 3) გვაქვს

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

$9 \bmod 19 = 5 \bmod 19$

$9 \neq 5$ (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.

ელიპტიკური წირის ჯგუფის რიგი (Group Order)

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

$Y^2 = x^3 + a \cdot x + b \pmod{p}$

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:

$$Y^2 = x^3 + a \,.\, x + b \pmod{p}$$

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 დაპროგრამების ენების ბიბლიოთეკებში და აქ არ მოგვყავს გარკვეული სირთულეების გამო.

სკალარული ნამრავლი და ციკლური ქვეჯგუფი

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

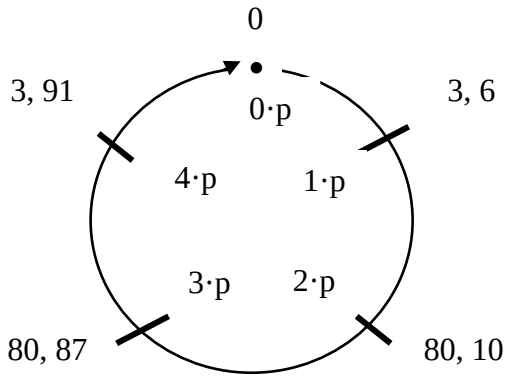
$$n \cdot p = \underbrace{p + p + \ldots + p}_{n-\text{ჯერ}}$$

სადაც p ელიპტიკური წირის წერტილი, ხოლო n ნატურალური რიცხვია.

განვიხილოთ სკალარული ნამრავლის თვისებები. როცა წირი მოცემულია განტოლებით $y^2 \equiv x^3 + 2{\cdot}x + 3 \pmod{97}$, 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:

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

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·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·P რიცხვებისთვის n= 0, 1, 2, … 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 · P;
4. უმცირესი n, რომლისთვისაც n · P = 0 იქნება P-ს რიგი (შესაბამისი ციკლის ელემენტების რაოდენობა).

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·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

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

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. რაც შეეხება მოცემული P და Q-ითვის ისეთი 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.

7.3 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.

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. 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).

7.4 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.

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.

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

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.

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 $P = k \cdot G$, სადაც G ბაზური, ხოლო P ციკლის ერთერთი წერტილია;

3. გამოთვალე $r = x_P \bmod n$ სადაც x_P არის P წერტილის აბსცისა;

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

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

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

7. წყვილი (r, s) არის ხელმოწერა.

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 $u_1 = S^{1-1} \cdot z \bmod n$;

2. გამოთვალე მთელი რიცხვი $u_2 = S^{1-1} \cdot r \bmod n$;

3. გამოთვალე წერტილი $P = u_1 \cdot G + u_2 \cdot H_A$.

ხელმოწერა ვალიდურია მხოლოდ იმ შემთხვევაში, როცა $r = x_P \pmod n$.

დიფი-ჰელმანის ეფემერული ალგორითმი

ზემოთ განხილული დიფი-ჰელმანის ალგორითმები კომუნიკაციაში მონაწილე მხარეებს საშუალებას აძლევს შექმნან სიმეტრიული დაშიფრისათვის საჭირო კრიფტოგრაფიული გასაღები ისე, რომ არ ხდება მისი გადაცემა არასაიმედო (დაუშიფრავი) საკომუნიკაციო არხის გამოყენებით. თუ ეს გასაღები გამიზნულია მრავალჯერადი გამოყენებისთვის, მაშინ იგი სტატიკური ხასიათისაა და საჭიროა მხარეებს უნდა ჰქონდეთ მისი ფლობის დამადასტურებელი სერთიფიკატები, რაც აუცილებელია მხარეების ავთენტიკაციისთვის, რაც იწვევს სისტემის საიმედობისა და სწრაფქმედების გაუარესებას. შედეგად, თანდათანობითი გავრცელება ჰპოვა ე.წ. დიფი-ჰელმანის ეფემერულმა ალგორითმმა, სადაც „ეფემერული“ (Ephemeral) მიუთითებს კრიპტოგრაფიული გასაღების მოკლევადიანად, ზღვრულად კი ერთჯერად გამოყენებაზე. ამ შემთხვევაში ახალი გასაღები იქმნება ყოველი კომუნიკაციის დაწყებამდე და უქმდება კომუნიკაციის დამთავრებისთანავე. ეს მიდგომა საშუალებას იძლევა განხორციელდეს ე.წ. სრულყოფილი ფორვარდული საიდუმლოება (Perfect Forward Secrecy, PFS). ეს ნიშნავს, რომ თითოეული ტრანზაქციისთვის გამოიყენება უნიკალური კრიპტოგრაფიული გასაღები, რაც მნიშვნელოვნად ამცირებს დაგროვილი (ღია ტექსტი, შიფრტექსტი) წყვილების ანალიზის საფუძველზე შიფრაციის გასაღების განსაზღვრის შესაძლებლობას. ტექნიკურ ლიტერატურაში დიფი-ჰელმანის ეფემერული ალგორითმები აღინიშნებიან როგორც DHE (Diffie-Hellman Ephemeral) და ECDHE (Elliptic Curve Diffie-Hellman Ephemeral).

თავი 8. ტრანსპორტის დონის უსაფრთხოება

8.1 ქსელურ კომუნიკაციებში უსაფრთხოების პროტოკოლების დამუშავებასა და გავრცელებასთან დაკავშირებული პრობლემები

მიუხედავად კიბერუსაფრთხოებაში მიღწეული შედეგებისა, ამ მიმართულებით დღეისათვის არსებობს გამოწვევები სხვადასხვა მიმართულებით, რომელთა დაძლევა მასშტაბურ კვლევით და საინჟინრო სამუშაოების შესრულებას მოითხოვს. ამ სამუშაოთა შესრულების საჭიროება მოტივირებულია არსებული მდგომარეობით, რაც შეიძლება შემდეგნაირად დახასიათდეს:

1. ინფორმაციული უსაფრთხოების სისტემების დამუშავების საქმეში ერთერთ ძირითად საყრდენს წარმოადგენს მათემატიკური მოდელები, რომლებიც შემუშავებულ იქნა ხანგრძლივი დროის განმავლობაში და რომლებიც ამჟამად გავრცელებული გამოთვლით ტექნიკის პირობებში ქმნიან საშუალებებს მონაცემთა დაცვის სისტემების პროექტირების და იმპლემენტაციის სფეროში. ამავე დროს მათემატიკა არ გამორიცხავს, რომ ამჟამად გამოყენებულ მოდელებისათვის დამუშავებულ იქნეს ახალი გამოთვლითი სქემები, რომელთა გამოყენება ძირს გამოუთხრის ამჟამად საიმედოდ მიჩნეულ მეთოდებს, რაც მოითხოვს მოძველებული ალგორითმების ახლით შეცვლას და ეს მასშტაბურ კომპიუტერულ ქსელებში დიდ დროსა და ძალისხმევასთან არის დაკავშირებული.

2. დღეისათვის უკვე ვრცელდება მოსაზრება, რომ მომავალ ხუთ წელში შეიძლება შეიქმნას ახალი ტექნოლოგიის ქვანტური კომპიუტერი, რომელიც ამჟამად არსებულ კომპიუტერთან შედარებით რამდენიმე რიგით უფრო სწრაფქმედი იქნება და მათი გამოყენების შემთხვევაში ალგორითმები, რომლებიც დღეისათვის ითვლებიან ძნელად შესასრულებლად , გადაინაცვლებენ ადვილად შესრულებად ალგორითმების კატეგორიაში. ამჟამად მიმდინარეობს იმის შეფასება, თუ რამდენად ფართოდ გავრცელებული იქნება ქვანტური კომპიუტერი და იმის დადგენა თუ მისი არსებობის პირობებში რომელი ალგორითმები და რა პარამეტრებით შეინარჩუნებენ ქმედითუნარიანობას. ამგვარად, მიმდინარეობს მოსამზადებელი სამუშაოები სერიოზული გაურკვევლობის პირობებში.

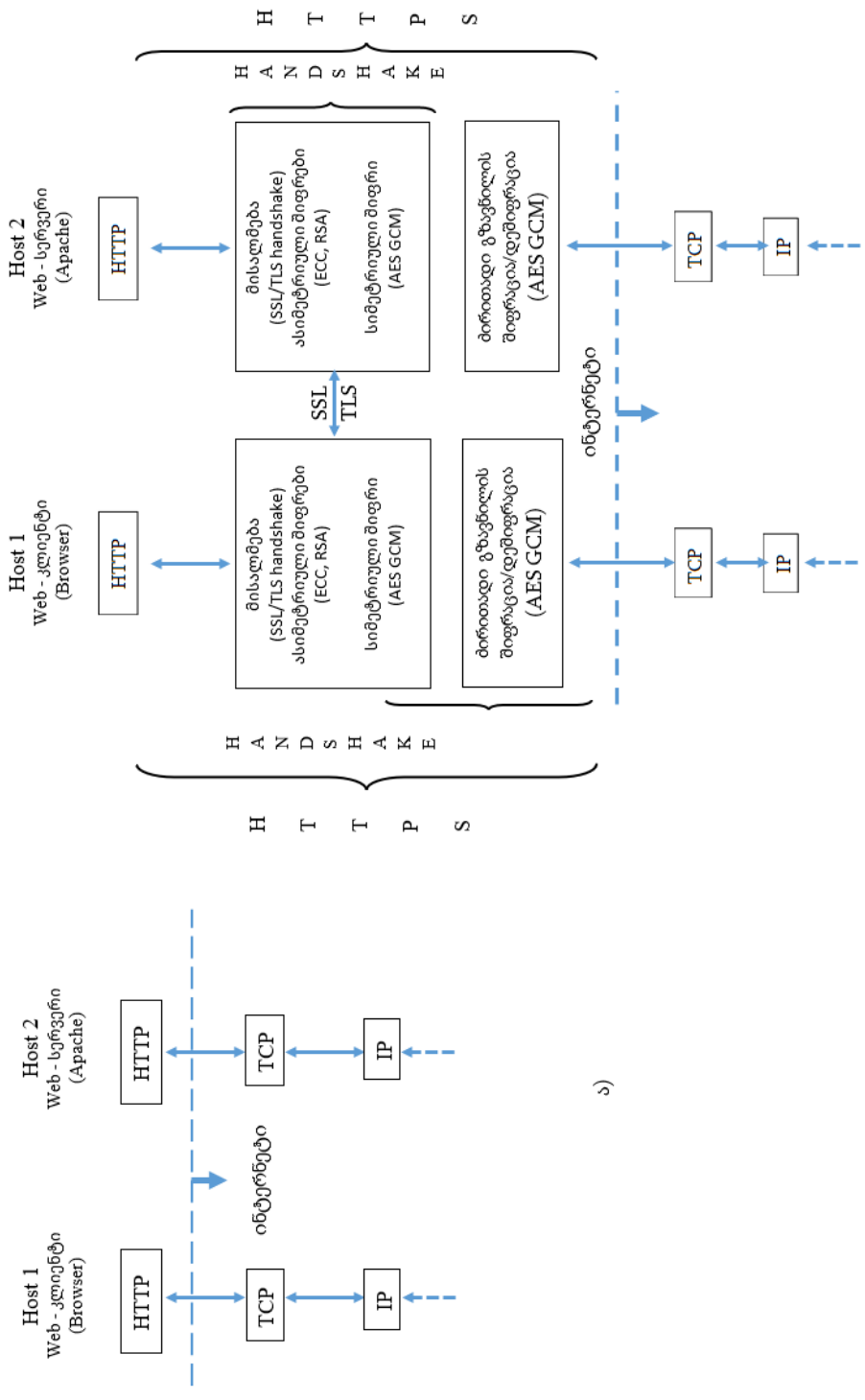
3. ინფორმაციული უსაფრთხოების სფეროში ახალი პროტოკოლების და სტანდარტების შექმნა და მათი გავრცელება დიდი მოცულობის და მრავალმხრივი სამუშაოების შესრულებას მოითხოვს. კერძოდ, უნდა მოხდეს ახალი

ალგორითმის მახასიათებლების გამოკვლევა სპეციალისტების მიერ, რასაც წლები სჭირდება რომ შეიქმნას მათი გამოყენების სათანადო სტანდარტები და ისინი უნდა რეკომენდებულ იქნეს გამოყენებისთვის რომელიმე ორგანიზაციისგან, როგორიცაა, მაგალითად, NIST (National Institute of Standards and Technology). ამის შემდეგ იწყება მისი თანდათანობითი გავრცელება, რაც აგრეთვე წლებს მოითხოვს. როგორც ვხედავთ ეს საკმაოდ ნელი პროცესია და ამას განაპირობებს ისიც, რომ ამ დროს ქსელში ერთდროულად გამოიყენება როგორც მოძველებული, ისე ახალი, გაუმჯობესებული სტანდარტები, რაც განპირობებულია იმით, რომ რიგ ორგანიზაციებს არ სურთ სტანდარტის შეცვლასთან დაკავშირებით დამატებითი სამუშაოს შესრულება. მეორე მხრივ, როგორც წესი, ქსელში უნდა უზრუნველყოფილ იქნეს ნებისმიერ ორ კვანძს შორის კომუნიკაციების შესაძლებლობა, რაც კვანძებს აიძულებს უზრუნველყონ როგორც მოძველებული, ისე განახლებული სტანდარტები.

4. ტექნიკური საშუალებები, როგორც წესი, სპეციფიკაციის შესაბამისად, მდგრადად მუშაობენ, მაგრამ სიტუაცია მკვეთრად იცვლება, როცა საქმეში ერევა სხვადასხვა მიზნების მქონე ადამიანები და ორგანიზაციები. მაგალითადდ, ჰაკერები იკვლევენ პროტოკოლებში არსებულ სისუსტეებს და შეცდომებს მათი არამართებული გამოყენების მიზნით, ხოლო ზოგიერთი ორგანიზაციები პროგრამისტებისგან მოითხოვენ, რომ პროტოკოლის განმახორციელებელ პროგრამულ უზრუნველყოფაში პროგრამისტებმა გაითვალისწინონ ე.წ. „უკანა შესასვლელები“ (Back Doors), რაც მათ საშუალებას მისცემს ნაკლები დანახარჯებით მოახდინონ დიდი ზომის ინფორმაციიდან საჭირო ინფორმაციის ამოღება.

8.2 ტრანსპორტის დონის უსაფრთხოება

ზემოთქმულის და კიდევ სხვა ფაქტების გათვალისწინებით განვიხილოთ ინფორმაციული უსაფრთხოების განვითარების ძირითადი ეტაპები და პერსპექტივები გამოყენებითი სისტემის HTTPS-ის მაგალითზე, რომელიც წარმოადგენს HTTP პროტოკოლის გაუმჯობესებულ ვერსიას, (HTTP Sequre), და რომლის ფუნქციონირების ზოგადი სქემა ნაჩვენებია სურათი1-ზე.


Host 1
Web - კლიენტი
(Browser)
Host 2
Web - სერვერი
(Apache)
HTTP
TCP
IP
ინტერნეტი
ა)
HTTPS
HANDSHAKE
მისალმება
(SSL/TLS handshake)
ასიმეტრიული მიფრები
(ECC, RSA)
სიმეტრიული მიფრი
(AES GCM)
SSL
TLS
მირითადი გზავნილის
მიფრაცია/დეშიფრაცია
(AES GCM)

სურათი 1

HTTP	Hyper Text Transfer protocol, ჰიპერტესტირების გადაცემის პროტოკოლი
HTTPS	HTTP secure, უსაფრთხო HTTP
SSL	Sequre Sochet Leyar უსაფრთხო სოკეტის შრე
TLS	Trasport Layer Sequrity ტრანსპორტის დონის უსაფრთხოება
ECC	Elyptic Curve Cryptography ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფია
RSA	ასიმეტრიული კრიპტოგრაფიის მეთოდი
TCP	Transmission Control Protocol
IP	Internet Protocol

ბ) სურათი 1

როგორც სურათი 1 გვიჩვენებს, გამოყენებითი სისტემები HTTP და HTTPS ეყრდნობიან ინტერნეტს TCP/IP პროტოკოლებს. პროტოკოლი TCP, კლიენტსა და სერვერს შორის ლოგიკური კავშირის დამყარების გზით, უზრუნველყოფს მათ შორის მონაცემების საიმედო გადაცემას მიუხედავად იმისა, მონაცემი დაუშიფრავია თუ დაშიფრული. იგი მონაცემს განიხილავს, როგორც დანომრილი ბაიტების მიმდევრობას და ქსელში მონაცემის დაკარგვის შემთხვევაში ახდენს მონაცემის აღდგენას განმეორებითი გადაგზავნის გზით ე.წ. „მცურავი ფანჯრის“ გამოყენებით. რაც შეეხება IP პროტოკოლს, იგი დაფუძნებულია ინტერნეტში მონაცემთა პაკეტების დეიტაგრამულ გადაცემაზე და აქაც შეცდომების შემთხვევაში ხდება მონაცემის პაკეტების განმეორებითი გადაცემის გზით. რამდენადაც TCP/IP პროტოკოლების სისტემა ეყრდნობა ლოკალურ და კორპორაციულ ქსელებს, ამ უკანასკნელებში მონაცემთა გადაცემის უსაფრთხოება გავლენას ახდენს მთლიანობაში ინტერნეტის უსაფრთხოებაზე და მათშიც გათვალისწინებულია მონაცემთა გადაცემისას შეცდომების აღმოჩენისა და გასწორების მეთოდები. უნდა აღინიშნოს, რომ ინტერნეტის პროტოკოლების სტეკის ქვემო დონეებში, მოწყობილობის მნიშვნელობის მიხედვით, ხშირად გამოიყენება მონაცემთა კოდირების მეთოდები, რომლებიც იძლევიან არა მარტო შეცდომების აღმოჩენის, არამედ მათი ავტომატურად გასწორების საშუალებებს. ასეთებია, მაგალითად, კომპიუტერის ძირითად მეხსიერებაში ბიტური შეცდომის აღმოჩენა და გასწორება ჰემინგის კოდის (Hamming Code)

გამოყენებით და საკომუნიკაციო არხში პაკეტური შეცდომის აღმოჩენა და გასწორება კოდირების რიდ-სოლომონის (Reed-Solomon Encoding) მეთოდის გამოყენებით.

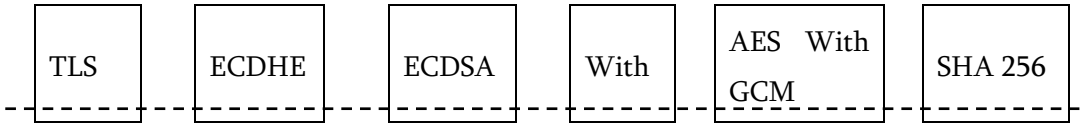
ამგვარად, შეიძლება ითქვას, რომ ინტერნეტის TCP/IP და სხვა ქვემდებარე პროტოკოლები უზრუნველყოფენ მონაცემთა უსაფრთხოდ გადაცემის მახასიათებელს (Data Integrety), თუმცა მათში არ არის გათვალისწინებული მონაცემთა კონფიდენციალობის ანუ საიდუმლოების უზრუნველყოფის საშუალებები, რამაც წარმოქმნა HTTPS შექმნის საჭიროება.

როგორც სურათი1 ბ) გვიჩვენებს, SSL/TLS პროტოკოლი განთავსებულია HTTP და TCP პროტოკოლებს შორის, როგორც პროტოკოლების შუალედური, დამატებითი შრე. მისი დანიშნულებაა განახორციელოს დიალოგი (მისალმება) კლიენტისა და სერვერის ამ დონეებს შორის კრიპტოგრაფიულ შესაძლებლობათა შეთანხმების მიზნით, რაც აუცილებელია შემდგომში HTTP პროტოკოლებს შორის მონაცემების გაცვლისათვის სიმეტრიული (სწრაფი) შიფრაციის ალგორითმის გამოყენებით. ამ მიზნის მისაღწევად მისალმების პროცედურებში გამოიყენება ასიმეტრიული კრიპტოგრაფია ღია გასაღებით, როგორიცაა, მაგალითად, RSA, დიფი-ჰელმანის ალგორითმი და ECC.

მისალმების პროცედურის დანიშნულებაა შემდეგი სამი ძირითადი მიზნის მიღწევა:

- კლიენტსა და სერვერს შორის კომუნიკაციისთვის აუცილებელი შეთანხმებული შიფრების კრებულების დადგენა;
- კლიენტისა და სერვერის (უფრო ხშირად სერვერის) ავთენტიკაცია;
- მონაცემთა შიფრაცია/დეშიფრაციისთვის საჭირო სიმეტრიული, ანუ სესიის გასაღების დადგენა.

ამ მიზნის მიღწევა ეფუძნება შიფრის და შიფრების კრებულის ცნებებს. შიფრი (Cipher) წარმოადგენს კრიპტოგრაფიაში გამოყენებულ ისეთ ალგორითმებს, როგორიცაა, მაგალითად, შიფრაცია, დეშიფრაცია და სხვ. შიფრების კრებული (Cipher Suite) წარმოადგენს შიფრების ერთობლიობას, რაც უზრუნველყოფს მხარეებს შორის კომუნიკაციას. მაგალითად, განვიხილოთ შიფრების შემდეგი კრებული:



შიფრების ამ კრებულში:

- TLS განსაზღვრავს ერთერთს ბოლო დროს მოქმედ ვერსიებიდან, TLS 1.2 ან TLS 1.3. ვერსია TLS 1.2-ის გამოყენება რეკომენდებულია ბოლო 15-20 წლიან პერიოდში და მრავალმხრივი გამოცდის შემდეგ ჩამოყალიბებულია ვერსია, რომელიც შეიცავს 37 რაოდენობის შიფრთა კრებულს. რაც შეეხება TLS 1.3, იგი განახლებული ვერსიაა, ითვლება თვისობრივად გაუმჯობესებულად და მოიცავს შიფრების 5 კრებულს;

- ECDHE (Elliptic Curve Diffie – Hellman Ephemeral) წარმოადგენს დიფი-ჰელმანის ალგორითმს, რომელიც იყენებს ელიპტიკურ წირზე დაფუძნებულ კრიპტოგრაფიას და განკუთვნილია ერთჯერადი გამოყენების სიმეტრიული გასაღების შესაქმნელად. ამგვარად, იგი საშუალებას იძლევა განხორციელდეს ე.წ. სრულყოფილი ფორვარდული კრიპტოგრაფია (Perfect Forward Cryptography), რაც მედეგია კრიპტანალიზისადმი, რადგან ერთი და იმავე გასაღებით გზავნილების დაშიფვრა არ ხდება;

- ECDSA (Ellyptic Corve Digital Signature Algorithm) წარმოადგენს ელიპტიკურ წირზე დაფუძნებულ ციფრული ხელმოწერის ალგორითმს და ითვლება დღეისათვის საუკეთესო ალგორითმად;

- AES 128 GCM წარმოადგენს AES (Advanced Encryption Standard) სიმეტრიული შიფრაციის განსაკუთრებულად ეფექტურ ალგორითმს, 128 ბიტიანი სიმეტრიული გასაღებით და გალუას მთვლელის მეთოდით (Galois Counter Mode), რაც საშუალებას იძლევა გადაიგზავნოს მონაცემები, რომლებიც შეიცავენ როგორც კონფიდენციალურ, დასაშიფრავ, ისე ღია ტექსტით გადასაცემ ნაწილებს;

- SHA 256 წარმოადგენს კრიპტოგრაფიული ჰეშირების ალგორითმს, რომელიც გამოიყენება როგორც ციფრული ხელმოწერის, ისე მონაცემთა გადაცემის სისწორის კონტროლის საკითხებში.

რადგანაც როგორც კლიენტში (ბრაუზერში) ისე სერვერში შეიძლება ინსტალირებული იყოს რეკომენდებული შიფრთა კრებულიდან მხოლოდ რამდენიმე, მისალმებისას საჭიროა პირველ რიგში გაირკვეს ის შიფრთა კრებულები, რომლებიც ინსტალირებული აქვს როგორც კლიენტს, ისე სერვერის მხარეს და ამორჩეულ იქნეს საერთო კრებული, რომლის გამოყენებითაც ხდება

მონაცემების გაცვლა კომუნიკაციაში მონაწილე მხარეებს შორის. უკანასკნელ პერიოდში გავრცელებულია TLS პროტოკოლის ვერსია TLS 1.2 და TLS 1.3. TLS 1.2 მოიცავს 37 შიფრთა კრებულს, ხოლო TLS 1.3, რომელიც წარმოადგენს TLS 1.2-ის გაუმჯობესებულ ვერსიას, მოიცავს მხოლოდ 5 კრებულს, როგორც ამას გვიჩვენებს სურათი 2.

37 კრებული

1.	TLS 1.2	ECDHE	ECDSA	AES_128_GCM	SHA256
2.	TLS 1.2	ECDHE	ECDSA	AES_256_GCM	SHA384
.					
.					
.					
36.	TLS 1.2	DHE	RSA	AES_128_CBC	SHA256
37.	TLS 1.2	DHE	RSA	AES_256_CBC	SHA256

ა)

5 კრებული

1.	TLS 1.3	ECDHE	ECDSA	AES_256_GCM	SHA384
2.	TLS 1.3	ECDHE	ECDSA	CHACHA20_POLY1305	SHA256
3.	TLS 1.3	ECDHE	ECDSA	AES_128_GCM	SHA256
4.	TLS 1.3	ECDHE	ECDSA	AES_128_CCM_8	SHA256
5.	TLS 1.3	ECDHE	ECDSA	AES_128_CCM	SHA256

ბ)

სურათი 2

როგორც სურათი გვიჩვენებს, TLS 1.2 შემთხვევაში თითოეული შიფრთა კრებული ხასიათდება 5 მონაცემით და მოიცავს 37 კრებულს, რომელთაგან ზოგიერთი შეიძლება იყოს მოძველებული და ინერციით შემორჩენილი ბრაუზერებსა და სერვერებში. რაც შეეხება TLS 1.3, აქ განსაზღვრულია მხოლოდ 5 კრებული და გარდა ამისა, ყველა მათგანი იყენებს შიფრებს ECDHE და ECDSA, რაც ამარტივებს ამორჩევის პროცესს. (სურათზე ეს ნაჩვენებია წყვეტილ გვერდებიან მართკუთხედში მოქცეული ელემენტებით). გარდა ამისა ელიპტიკური კრიპტოგრაფიის გამოყენება საშუალებას იძლევა განხორციელდეს სრულყოფილი ფორვარდული საიდუმლოება, რაც უზრუნველყოფილია ECDHE-ის გამოყენებით და სწრაფქმედებით, რაც მიიღწევა ECDSA-ის გამოყენებით. (Nohe, 2019)

ბოლო 10-15 წლის მანძილზე შიფრების კრებულებში უფრო ხშირად გამოიყენებოდა RSA კრიპტოგრაფია, როგორც კარგად დამუშავებული და გავრცელებული. ამავე დროს ბოლო 10 წლის მანძილზე ვითარდებოდა ელიპტიკურ წირზე დაფუძნებული კრიპტოგრაფია, რომელიც, თუმცა გამოირჩევა შედარებითი სირთულით და მოითხოვს მეტ ყურადღებას ინსტალაციის ეტაპზე, მაგრამ ხასიათდება RSA შიფრზე უკეთესი სწრაფქმედებით და ჰაკერების შეტევებისადმი მედეგობით. ამ ორი შიფრის შედარებაზე წარმოდგენას გვაძლევს შემდეგი ცხრილი:

Sequrity (bits)	RSA (key length)	ECC (key length)
80	1024	160-223
112	2048	224-255
128	3072	256-383
192	7680	384-511
256	15360	512 +

აქ მახასიათებელი sequrity მიუთითებს იმაზე, თუ რა ზომის გამოთვლები იქნება ჩასატარებელი შიფრის გასატეხად უხეში ძალის (Brute Force) გამოყენების შემთხვევაში, რაც იზომება როგორც 2^{80}, 2^{112} და ა.შ. 2^{256}. როგორც ცხრილი გვიჩვენებს, მაგალითად, RSA შიფრს გასაღებით 2048 ბიტი შეესაბამება ECC გასაღები სიგრძით დიაპაზონში 224-255, ხოლო RSA შიფრს გასაღებით 7680 ბიტი შეესაბამება ECC შიფრი გასაღებით დიაპაზონში 384--511 ბიტი, რაც

მიუთითებს ECC შიფრის უპირატესობაზე RSA შიფრთან შედარებით.საზოგადოებაში გავრცელებულია მოსაზრება ინფორმატიკის მიღწევების სწრაფად გავრცელების შესახებ. მაგრამ აქ უნდა განსხვავდეს კარგად დამუშავებული საშუალებების გავრცელების სისწრაფე მათი დამუშავების ხანგრძლივობასთან შედარებით. რაც შეეხება ახალი საშუალებების შექმნას, იგი დიდ დროს მოითხოვს. მაგალითად, TLS პროტოკოლის დამუშავება დაიწყო 2006 წლამდე და პირველი ვერსია TLS 1.1 შეიქმნა 2006 წელს, მომდევნო გაუმჯობესებული ვერსია შეიქმნა 2009 წელს და მოქმედია დღევანდლამდე მილევად რეჟიმში. რაც შეეხება საბოლოო ვერსიას TLS 1.3 იგი შეიქმნა 2009 წელს და დღეისთვის წარმოადგენს ძირითად ვერსიას. მის შექმნასა და გავრცელებას დასჭირდა 10 წელი და 28 შუალედური პროექტის რეალიზება ინფორმატიკის სფეროში უმსხვილესი კორპორაციების მონაწილეობით.

მითითებული ლიტერატურა

ANDREW S. TANENBAUM, NICK FEAMSTER, DAVID WETHERALL. (2021). *COMPUTER NETWORKS.* Pearson Education.

Corbellini, A. (2015). Elliptic Curve Cryptography: a Gentle introduction. მოპოვებული https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/#scalar-multiplication-დან

Nohe, P. (2019 წლის 30 April). Taking a Closer Look at the SSL/TLS Handshake. მოპოვებული https://www.thesslstore.com/blog/explaining-ssl-handshake-დან

ლ. მძინარიშვილი, ნ. კაჭახიძე, დ. უგულავა, ნ. ხომერიკი. (2018). *დისკრეტული მათემატიკა.* საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“.

გადაეცა წარმოებას 15.12 2024. ოფსეტური ქაღალდის ზომა
60X84 1/16. პირობითი ნაბეჭდი თაბახი 16,25. 100 გვ, ტირაჟი 50 ეგზ.

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრი“
თბილისი, მ.კოსტავას 77

ISBN 978-9941-8-7348-5