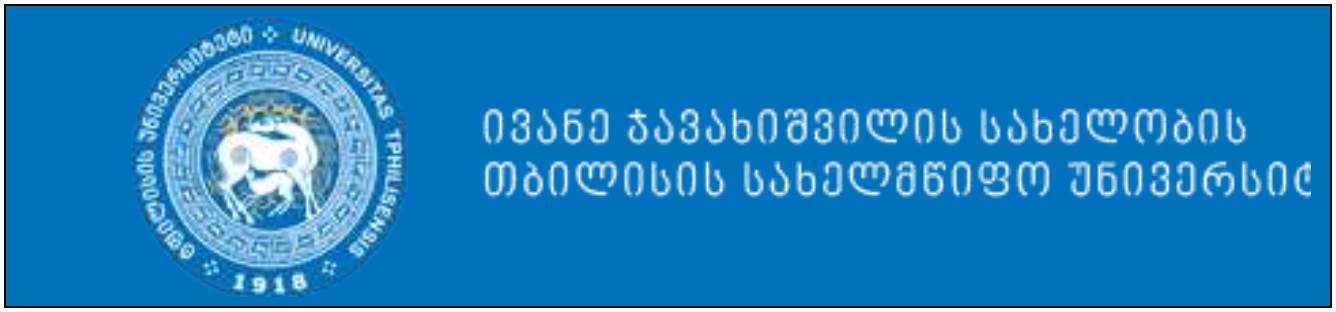


ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტი

ცეზარი მშვენიერაძე

AES სტანდარტის მოდიფიცირებული ვარიანტის შექმნა და ტესტირება

სამაგისტრო პროგრამა - ინფორმაციული ტექნოლოგიები

ნაშრომი შესრულებულია ინფორმაციული ტექნოლოგიების მაგისტრის აკადემიური ხარისხის მოსაპოვებლად

ხელმძღვანელი: ასოცირებული პროფესორი ზურაბ ქოჩლაძე

თბილისი 2019

სარჩევი

ანოტაცია

კვლევის მიზანი იყო AES-ის სტანდარტის მოდიფიცირებული ვარიანტის შექმნა და NIST-ის (სტანდარტებისა და ტექნოლოგიების ინსტიტუტი) მიერ შემოთავაზებულ, შემთხვევითი რიცხვების გენერაციის ტესტებზე შემოწმება, რასაც უნდა დაედგინა მიღებული ალგორითმით დაშიფრული ტექსტების ბიტური მიმდევრობა არის თუა რა შემთხვევითი მიდევრობები ამისთვის შეიქმნა მოდიფიცირებული ვარიანტის პროგრამული იმპლემენტაცია, რომელშიც სტანდარტული AES-ის სტრუქტურიდან ამოვიღე **SriftRow** ფუნქცია ,რომელიც AES-ის უმნიშვნელოვანეს ნაწილს წარმოადგენს და იგი ჩანაცვლდა ჩვენს მიერ მოფიქრებული, თვით შებრუნებადი მატრიცაზე გამრავლების ფუნქციით. შედეგად მივიღეთ, რომ ჩვენს მიერ შექმნილი ვარიანტი ბევრად სწრაფად მუშაობს ვიდრე სტანდარტული AES-ის იმპლემენტაცია, მრავალმა ტესტირებამ, რომელსაც, საბაზისოდ, ყველა ახლად შექმნილი კრიპტოგრაფიული ალგორითმები გადის და აჩვენა, რომ შეცვლილი იმპლემენტაციით მიღებული დაშიფრული ტექსტები აკმაყოფილებს NIST-ის მირე შემოთავაზებულ შემთხვევითი რიცხვების გენერატორის ტესტებს და მისი გამოყენება მიზანშეწონილი არის კრიპტოგრაფიაში, მაგრამ შემდეგ ეტაპზე უნდა შემოწმდეს მისი იმპლემეტაცია არამხოლოდ პროგრამულ დონეზე არამედ ჰარდვეარის დონეზე C-ენის გამოყენებით რათა სიჩქარე ბევრად განსხვავებული იქნება ვიდრე ჩემს მიერ ჩატარებული ცდების დროს.

Abstract

The goal of the research was to create a modified version of the AES standard and to check on Random Number Generation Tests offered by NIST (Standards and Technology Institute) to determine the quality of the result, that gave from encryption. I have created a software code, in the standard AES structure that I took out of the SriftRow function, which is the most important part of AES and has been replaced by our propagation on the self-reverse matrix. As a result, we have created a variant of the standard AES implementation, and the encrypted texts received from the modified implementation passed tests that provided from NIST and its Available to use this algorithm in the cryptography.

შესავალი

დაშიფვრის ახალი ფედერალური სტანდარტი **AES**-ი შექმნეს ბელგიელმა კრიპტოგრაფებმა **იოან დამენმა** და **ვინსენტ რაიმანმა** 1998 წელს და უწოდეს მას **Rijndael**-ი (ალგორითმის სახელი წარმოადგენს ავტორთა გვარების კომბინაციას). ეს ალგორითმი განსხვავებით **DES**-გან მიეკუთვნება იმ ბლოკურ ალგორითმებს, რომლებიც არ იყენებენ ფეისტელის სქემას. მას საფუძვლად უდევს ე.წ. "**კვადრატის**" არქიტექტურა. ეს სახელწოდება არქიტექტურამ მიიღო კრიპტოალგორითმ კვადრატისგან, რომელიც 1996 წელს შექმნეს იმავე ავტორებმა **ლარს კნუდსენთან** ერთად. ალგორითმს შეუძლია იმუშაოს 128,192 და 256 ბიტიან ბლოკებთან (თუმცა სტანდარტს წარმოადგენს 128 ბიტიანი ბლოკი). გასაღების სიგრძე ასევე შეიძლება იყოს 128, 192 და 256 ბიტის სიგრძის, ამასთან შესაძლებელია სხვადასხვა სიგრძის ბლოკებთან სხვადასხვა სიგრძის გასაღებების გამოყენება სტანდარტში გამოიყენება 128 ბიტიანი ბლოკი და გასაღები). ვინაიდან ალგორითმში ფაქტობრივად სრულდება ორი მარტივი ოპერაცია – **XOR**-ით შეკრება და ბაიტების ინდექსირებული ამოღება მეხსიერებიდან, ალგორითმი შეიძლება ეფექტურად იყოს რეალიზებული ყველა ცნობილი პლატფორმისა და აპარატურისთვის.

მე გამოვიყენე სტანდარტული 128 ბიტიანი ბლოკი და გასაღები და AES - ის არქიტექტურიდან (რომელსაც დეტალურად განვიხილავ მომდევნო თავებში) ამოვიღე ShiftRow -ფუნქცია და ის ჩავანაცვლე თვით შებრუნებად მატრიცაზე გამრავლებით, რადგან მატრიცაზე გამრავლება მძიმე ოპერაციაა. ამიტომ რაუნდების რაოდენობა შევამცირე ერთამდე , ათის მაგიერ. რამაც 3,5 ჯერ გაზარდა ბლოკის დაშიფვრის სიჩქარე. შესაბამისად სიჩქარე გაიზარდა , მაგრამ მიღებული შედეგი უკვე უნდა გადამოწმებულიყო ტესტებით. NIST გვთავაზობს მის მიერ შემუშავებულ ტესტებს, რომლებსაც გადის ალგორითმები სანამ ისინი მიიღებენ სერთიფიკატს NIST -ისაგან. აქ ჩვენმა ახლად შემუშავებულმა და იმპლემეტირებულმა ალგორითმა კარგი შედეგები აჩვენა ყველა ტესტი რომელზეც შევამოწმე წარმატებით გაიარა.

მოცემულ ნაშრომში ეტაპობრივად განხილული იქნება ჯერ თეორიული მხარე , ჰილის ალგორითმი, [1] მისი მოდიფიცირებული ვარიანტი, AES-ის არქიტექტურა და მისი მოდიფიცირებული ვარიანტი ჩვენი თეორიის საფუძველზე , საბოლოოდ ტესტირების საფეხურების აღწერა და მათი საშუალებით , მოდიფიცირებული ალგორითმით მიღებული შედეგის ანალიზი და საბოლოოდ დასკვნები. რაც დაგვეხმარება გავაანალიზოთ მიღებული შედეგი და დავსახოთ ამ ალგორითმის განვითარების გზები.

ჰილის ალგორითმი

ვიდრე ჩვენი თეორიის ჩამოყალიბებაზე გადავალთ გავეცნოთ ჰილის ალგორითმს, რომელსაც სტრუქტურულად ჰგავს AES. 1929 წელს ამერიკელმა მათემატიკოსმა ლესტერ ს. ჰილმა წრფივი ალგებრის გამოყენებით შექმნა n-გრამული დაშიფვრის ალგორითმი, რომელიც საშუალებას იძლევა შიფროტექსტის ერთი გამოსასვლელი სიმბოლო დამოკიდებული იყოს n ცალ შესასვლელ სიმბოლოზე. ამ მიზნით მან ღია ტექსტის ასოებს შეუსაბამა რიცხვები ნოლიდან m-მდე (სადაც $m+1$ არის ასოების რაოდენობა ანბანში), ისე, როგორც ამას აკეთებდნენ კლასიკური კრიპტოგრაფიის მრავალ შიფრში. დასაშიფრი ტექსტი გადაიყვანა რიცხვებში და დაყო n სიგრძის ბლოკებად. იმისათვის, რომ დაეშიფრა ეს n ცალი რიცხვი (ანუ ღია ტექსტის n ცალი ასო) ერთდროულად, აიღო კვადრატული მატრიცა $n \times n$-ზე და გაამრავლა ღია ტექსტის შესაბამისი რიცხვების მიმდევრობა, როგორც ვექტორი, მატრიცაზე (ეს მატრიცა წარმოადგენდა შიფრის გასაღებს) მოდულით ოცდაექვსი. მიიღო კვლავ n სიგრძის ვექტორი, რომელიც წარმოადგენს შიფროტექსტს და რომლის თითოეული სიმბოლო დამოკიდებულია შესასვლელი ვექტორის n სიმბოლოზე. ეს იყო ჰილის ალგორითმის ყველაზე მნიშვნელოვანი და არსებითი განსხვავება მანამდე არსებული დაშიფვრის მეთოდებისგან. იმისათვის, რომ შესაძლებელი იყოს დეშიფრაცია, ცხადია, დაშიფვრის მატრიცას უნდა გააჩნდეს შებრუნებული მატრიცა მოდულით m. ამისათვის კი საკმარისია, რომ დამშიფრავი მატრიცის დეტერმინანტი განსხვავდებოდეს ნულისგან და ურთიერთმარტივი იყოს მოდულის ფუძესთან.

მაგალითად, თუ ჩვენ გვინდა, რომ შიფროტექსტის ერთი გამოსასვლელი სიმბოლო დამოკიდებული იყოს სამ შესასვლელ სიმბოლოზე, უნდა ავიღოთ მატრიცა

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

ისეთი, რომ $A\cdot A^{-1}=E$, სადაც E არის ერთეულოვანი მატრიცა და გავამრავლოთ ღია ტექსტის ტრიგრამზე (რიცხვებში გადაყვანის შემდეგ).

$$M\times A=C\,.$$

დეშიფრაცია კი მოხდება ფორმულით:

$$C\times A^{-1}=M\ .$$

განვიხილოთ მარტივი მაგალითი. დავუშვათ გვინდა დავშიფროთ სიტყვა „ძნა“. გასაღები იყოს სიტყვა „შიფრატორი“. თუ ამ სიტყვებს გადავიყვანთ რიცხვებში და გასაღებს ჩავწერთ მატრიცის სახით, მივიღებთ:

$$\mathrm{A}=\begin{pmatrix} 24 & 8 & 20 \\ 16 & 0 & 18 \\ 13 & 16 & 8 \end{pmatrix} \text{ და } \begin{pmatrix} 27 \\ 12 \\ 0 \end{pmatrix}\ .$$

შიფროგრამა კი მიიღება გასაღების ღია ტექსტზე გამრავლებით:

$$\begin{pmatrix} 24 & 8 & 20 \\ 16 & 0 & 18 \\ 13 & 16 & 8 \end{pmatrix}\times \begin{pmatrix} 27 \\ 12 \\ 0 \end{pmatrix}\ (\mathrm{mod}33)=\begin{pmatrix} 18 \\ 3 \\ 15 \end{pmatrix}\ (\mathrm{mod}33)$$

რაც შეესაბამება ტრიგრამს „ტდჟ“.

ამ მატრიცას გააჩნია შებრუნებული მატრიცა რომელიც ტოლია

$$A^{-1} = \begin{pmatrix} 21 & 7 & 6 \\ 31 & 10 & 32 \\ 7 & 14 & 13 \end{pmatrix},$$

ამიტომ, თუ მიღებულ შიფროგრამას გავამრავლებთ ამ მატრიცაზე, მივიღებთ ღია ტექსტს

$$\begin{pmatrix} 21 & 7 & 6 \\ 31 & 10 & 32 \\ 7 & 14 & 13 \end{pmatrix}$$

ცხადია, რაც უფრო დიდი იქნება დამშიფრავი მატრიცის ზომა, მით უფრო მეტი ღია ტექსტის ასო მიიღებს მონაწილეობას გამოსასვლელი შიფროტექსტის ერთი სიმბოლოს გამოთვლაში და მით უფრო კარგად დაიმალება ღია ტექსტის სტრუქტურა შიფროტექსტში, მაგრამ ჰილის ალგორითმის გამოყენება ხელით დაშიფვრის დროს საკმაოდ რთულია, ამიტომ ამ დროს დამშიფრავი მატრიცის ზომაც შესაბამისად პატარაა, რაც ართულებს დასახული მიზნის მიღწევას.

კომპიუტერული კრიპტოგრაფიის განვითარების პირველ ეტაპზე ჰილის ალგორითმის გამოყენებაზე უარი თქვეს იმ მიზეზით, რომ ვექტორის მატრიცაზე გამრავლება წარმოადგენს წრფივ ოპერაციას და თუ ალგორითმში გამოყენებულია $n \times n$-ზე მატრიცა, მის გასატეხად საჭიროა მხოლოდ n^2 წრფივი განტოლების ამოხსნა, მაგრამ ბოლო წლების განმავლობაში გაჩნდა შრომები , რომლებშიც ჰილის ალგორითმის სხვადასხვა მოდიფიკაციები გამოიყენება რომელიმე არაწრფივ ოპერაციასთან ერთად. ეს შეუძლებელს ხდის ალგორითმის მარტივად გატეხვას და ინარჩუნებს ჰილის ალგორითმის ყველა დადებით თვისებას. [1]

ჰილის ალგორითმის მოდიფიცირებული ვარიანტი

[1]ჩვენს მიერ მოდიფიცირებული ჰილის ალგორითმი შეიძლება გამოყენებული იყოს ისეთ შიფრებში, რომლებშიც დასაშიფრი ბლოკი წარმოიდგინება მდგომარეობის მატრიცის სახით (როგორც ეს ხდება მაგალითად **AES** სტანდარტში). განვიხილოთ კრიპტოალგორითმი, რომელშიც ბლოკის ზომა ტოლია 128 ბიტის. ალგორითმში ეს ბლოკი შეგვიძლია

წარმოვადგინოთ 4×4 მატრიცის საშუალებით, რომელსაც უწოდებენ მდგომარეობის მატრიცას.

$$M=\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

აქ თითოეული a_{ij} წარმოადგენს ორობით ბაიტს. დასაშიფრი ორობითი სტრიქონი ჩაიწერება მატრიცაში მარცხნიდან მარჯვნივ ჰორიზონტალურად და ზევიდან ქვევით. თუ გადავიყვანთ თითოეულ ბაიტს ათობით სისტემაში, ნებისმიერი a_{ij} ელემენტისთვის შესრულდება პირობა $0\le a_{ij}\le 255$. ჩვენს მიერ მოდიფიცირებული ალგორითმში ხდება ამ მარტიცის გამრავლება 4×4-ზე განზომილების A მატრიცაზე მოდულით 256.

$$M\times A(\mathrm{mod}\,256),$$

თავისთავად ცხადია, რომ მატრიცის მატრიცაზე გამრავლება აღარ წარმოადგენს ისეთ მარტივ ოპერაციას, როგორიცაა გადანაცვლება ან ჩანაცვლება, რამაც შეიძლება არსებითად იმოქმედოს ალგორითმის სისწრაფეზე. იმისთვის, რომ შევინარჩუნოთ ალგორითმის სწრაფქმედება დასაშვებ ფარგლებში, A მატრიცის ელემენტები უნდა იყოს რაც შეიძლება პატარა რიცხვები. ასეთ შემთხვევაში კი შესაძლებელია, რომ A^{-1} მატრიცის ელემენტები იყოს დიდი რიცხვები, რაც გაზრდის დეშიფრაციის დროს, რაც ასევე არაა სასურველი. ამ მიზეზების გათვალისწინებით ჩვენ შევარჩიეთ თვითშებრუნებადი მატრიცა

$$A=\begin{pmatrix} 2 & -1 & -2 & 2 \\ -1 & -2 & -2 & -2 \\ 1 & 1 & 1 & 2 \\ -1 & 1 & 2 & -1 \end{pmatrix},$$

შესაძლებელი არის რომ ავიღოთ სხვა მატრიცა იგივე თვისებების მქონე მაგრამ, ჩვენი მატრიცის ელემენტებია მხოლოდ ± 1 და ± 2 რიცხვები, რაც ძალიან ამარტივებს მატრიცის მატრიცაზე გამრავლების ოპერაციას (გავიხსენოთ, რომ ორობით სისტემაში ორზე გამრავლება ტოლფასია ათობით სისტემაში ათზე გამარავლების). ის ფაქტი, რომ ასეთ მატრიცაზე

გამრავლების შედეგად მდგომარეობათა მატრიცაში მივიღებთ უარყოფით რიცხვებს, არ წარმოადგენს პრობლემას, რადგანაც მარტივად, მოდულის დამატებით შესაძლებელია მათი კვლავ დადებით რიცხვებში გადაყვანა.

განვიხილოთ კონკრეტული მაგალითი. დავუშვათ მოცემული გვაქვს ღია ტექსტი: “domain parameters”. ASCII კოდების გამოყენებით შევუსაბამოთ ასოებს რიცხვებში ათობით სისტემაში და შემდეგ გადავიყვანოთ ბიტურ სტრიქონში. ათობით სისტემაში მივიღებთ მატრიცას 4×4-ზე:

$$M=\begin{pmatrix}100 & 111 & 109 & 97\\ 105 & 110 & 112 & 97\\ 114 & 97 & 109 & 101\\ 116 & 101 & 114 & 115\end{pmatrix},$$

რაც მოგვცემს ბიტურ სტრიქონს:

01100100 01101111 01101101 01100001 01101001 01101110 01110000 01100001 01110010 01100001 01101101 01100101 01110100 01100101 01110010 01110011.

გავამრავლოთ მიღებული M მატრიცა A მატრიცაზე მოდულით 256. გვექნება:

$$\begin{pmatrix}100 & 111 & 109 & 97\\ 105 & 110 & 112 & 97\\ 114 & 97 & 109 & 101\\ 116 & 101 & 114 & 115\end{pmatrix}\times\begin{pmatrix}2 & -1 & -2 & 2\\ -1 & -2 & -2 & -2\\ 1 & 1 & 1 & 2\\ -1 & 1 & 2 & -1\end{pmatrix}-\ mod\ 256=\begin{pmatrix}101 & 140 & 137 & 99\\ 115 & 140 & 132 & 117\\ 139 & 158 & 44 & 151\\ 130 & 157 & 166 & 143\end{pmatrix}$$

$mod\ 256$,[1]

ამ ალგორითმზე დაყრდნობით უნდა შექმნილიყო პროგრამული კოდი რომელიც საშუალებას მოგვცემდა რეალურ გარემოში მისი სისწორის შემოწმებას, ამიტომ ავიღეთ AES სტანდარტი და მასში გადავწყვიტეთ ამ ყველაფრის იმპლემენტირება.

AES-ის არქიტექტურა

AES-ში ყველა გარდაქმნა სრულდება ბაიტურ მატრიცაზე, რომელსაც ეწოდება მდგომარეობათა მატრიცა. მას აქვს ოთხი სტრიქონი, ხოლო სვეტების რაოდენობა, Nb, დამოკიდებულია ბლოკის სიგრძეზე და უდრის ბლოკის სიგრძე გაყოფილი 32 -ზე, -ზე, ჩვენ შემთხვევაში 4×4 (იხ. სურათი. 1).

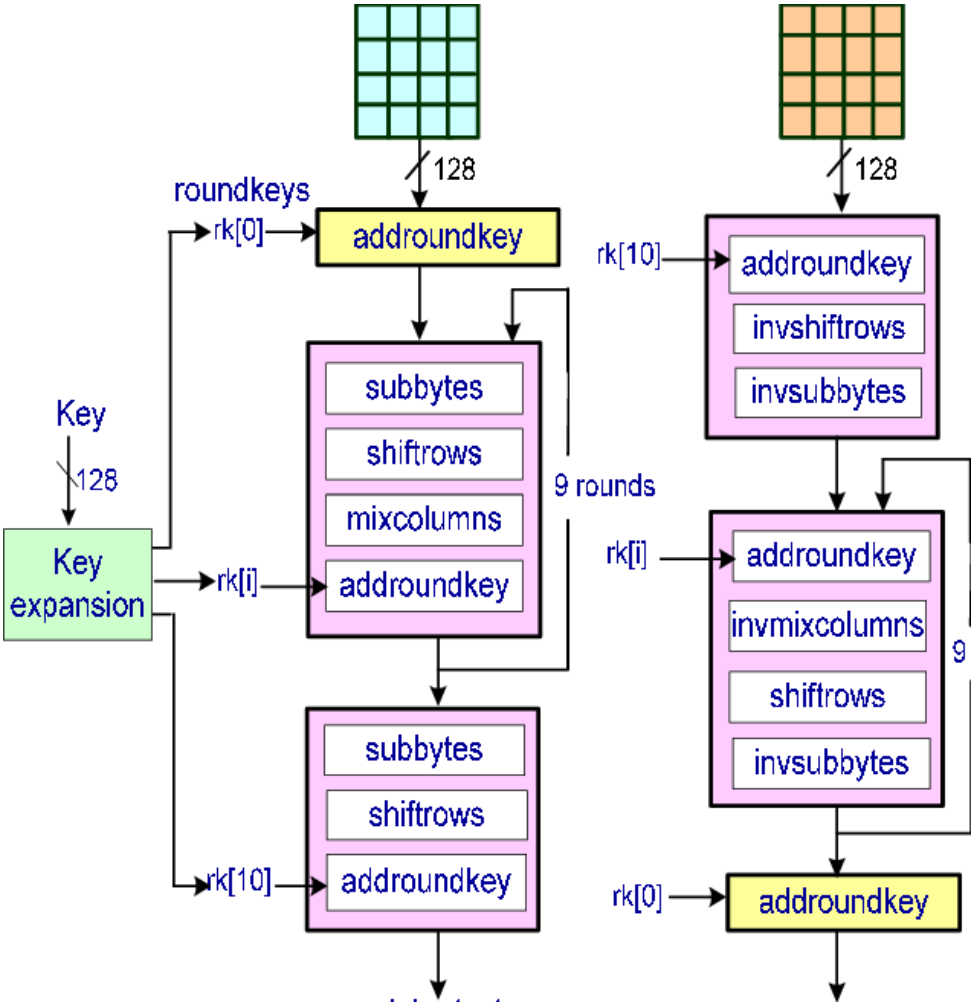
$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

სურათი.1. მდგომარეობათა მატრიცა (დასაშიფრი ბლოკი ტოლია 128 ბიტის)

AES-ში გვაქვს მათემატიკური ოპერაციები. ამიტომ მდგომარეობათა მატრიცის თითოეული ბაიტი გადადის რიცხვში (ნულიდან ორასორმოცდახუთმეტის ჩათვლით) და მუშავდება როგორც რიცხვები. ამასთან მიღებულია რიცხვების წარმოდგენა პოლინომების სახით. პირველ რაუნდში შესვლამდე ღია ტექსტის ბლოკი **XOR** ოპერაციით იკრიბება გასაღებთან (ამას ეწოდება *ტექსტის გათეთრება*). ყოველი რაუნდი, გარდა დამამთავრებელი რაუნდისა, შედგება ოთხი გარდაქმნისგან:

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

დამამთავრებელ რაუნდში გამოტოვებულია ოპერაცია **Mixcolumn (state).**



სურ. 2 AES-ს არქიტექტურა.

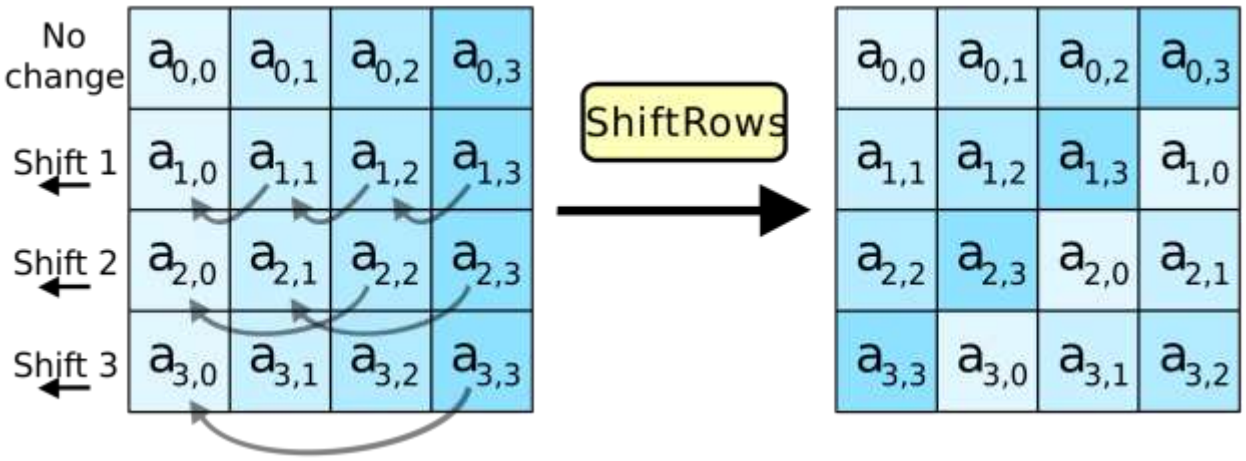
ზემოთ ჩამოთვლილი გარდაქმნებიდან გამოვარჩიოთ შემდეგი გარდაქმნა, **SriftRow (State),** რომელიც ახდენს მდგომარეობის მატრიცის სტრიქონების ციკლურ გადაადგილებას (წაძვრას მარცხნივ). ამასთან ეს წაძვრები სხვადასხვა სტრიქონებისთვის სრულდება განსხვავებულად და დამოკიდებულია მდგომარეობის მატრიცაში სვეტების რაოდენობაზე (**Nb**). ეს დამოკიდებულება მოცემულია ცხრილში 3. სადაც **C0,C1,C2,C3** აღნიშნავენ შესაბამისად პირველ, მეორე, მესამე და მეოთხე სტრიქონებს.

ცხრილი 3. . სტრიქონების წაძვრის დამოკიდებულება სვეტების რაოდენობაზე

Nb	CO	C1	C2	C3
4	0	1	2	3
6	0	1	2	3
10	0	1	3	4

ასეთი პარამეტრები ყველა შესაძლო კომბინაციებიდან ამორჩეული იქნა შემდეგი კრიტერიუმებით:

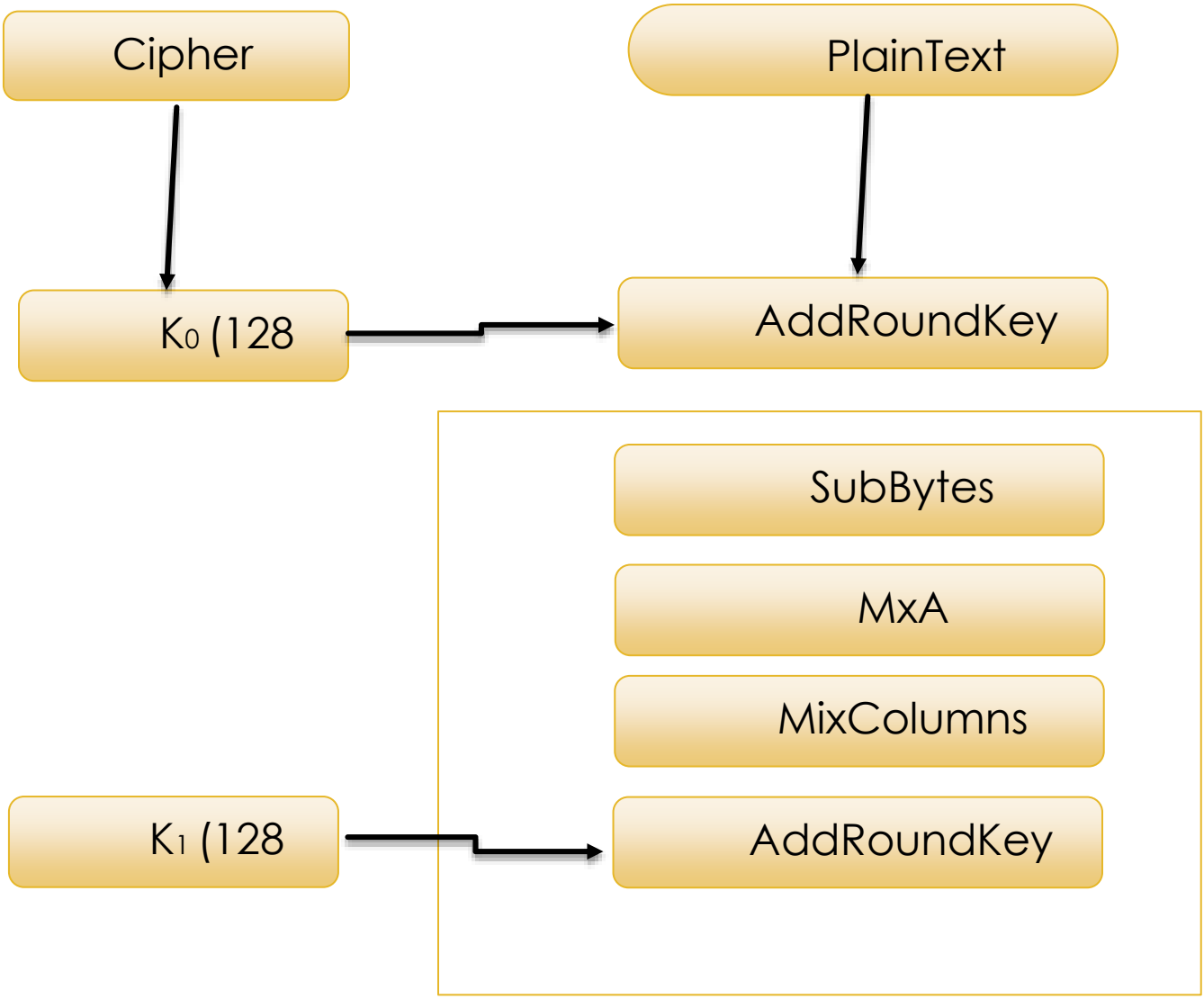
ცხრილი 4.



1. ოთხივე სტრიქონის წაძვრა უნდა ყოფილიყო განსხვავებული და ამასთან $C0 = 0$;
2. ოპერაცია გამძლე უნდა ყოფილიყო შეკვეცილი დიფერენციალებით შეტევის მიმართ;
3. ოპერაცია გამძლე უნდა ყოფილიყო კვადრატული შეტევის მიმართ;
4. ოპერაცია უნდა ყოფილიყო მარტივი.

ამ ქმედების მაგივრად ჩვენ ავიღეთ და შემოვიტანეთ ახალი ფუნქცია , რომელშიც სრულდება მდგომარეობის მატრიცაზე, რომელზეც უკვე ჩატარებულია ByteSub(State) მოქმედება.

შედეგად მივიღეთ შემდეგი ალგორითმი



სადაც MxA ფუნქცია არის მდგომარეობის მატრიცა გამრავლებული თვითშებრუნებად მატრიცაზე მოდულით 256. იხ. სურათი 5.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{1,2} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{1,3} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{1,4} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix} \times \begin{pmatrix} 2 & -1 & -2 & 2 \\ -1 & -2 & -2 & -2 \\ 1 & 1 & 1 & 2 \\ -1 & 1 & 2 & -1 \end{pmatrix} - mod\ 256 =$$

$$\begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{1,2} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{1,3} & b_{3,2} & b_{3,3} & b_{3,4} \\ b_{1,4} & b_{4,2} & b_{4,3} & b_{4,4} \end{pmatrix} mod\ 256$$

სურათი 5.

AES-ის იმპლემენტაცია

ალგორითმების ჩამოყალიბების შემდეგ, უნდა შექმნილიყო C# -ზე ჯერ სტანდარტული AES-ის იმპლემენტაცია რომელიც წარმატებით ჩააბარებდა ტესტებს , რის მიხედვითაც დავრწმუნდებოდით რო საწყისი ალგორითმი არ იქნებოდა წარუმატებლობის მიზეზი. შესაბამისად დავწერე ალგორითმი რომელიც მუშაობდა 128 ბიტიან ბლოკებზე. და წარმატებით ართმევდა თავს ტესტებს. რომელზეც შემდეგ უნდა გაგვეტარებია მოდიფიცირებული ვარიანტიც.

მივიღეთ შემდეგი სტრუქტურის მქონე კოდი (იხ. სურათი 6) რომელიც 128 ბიტიან ბლოკს შიფრავდა 10 რაუნდის გავლის შედეგად და როგორც ზემოთ ვთქვით AES-ის ბოლო რაუნდი განსხვავდება სხვა რაუნდებისგან , რადგან არ ამოღებულია MixColumns ფუნქცია.

```
public void Cipher(byte[] input, byte[] output)  // encipher 16-bit input
{
    this.State = new byte[4, Nb];  // always [4,4]
    for (int i = 0; i < (4 * Nb); ++i)
    {
        this.State[i % 4, i / 4] = input[i];
    }

    AddRoundKey(0);

    for (int round = 1; round <=Nr-1; ++round)  // main round loop
    {
        SubBytes();
        ShiftRows();
        MixColumns();
        AddRoundKey(round);
    }  // main round loop

    SubBytes();
    ShiftRows();
    AddRoundKey(Nr);

    for (int i = 0; i < (4 * Nb); ++i)
    {
        output[i] = this.State[i % 4, i / 4];
    }

}  // Cipher()
```

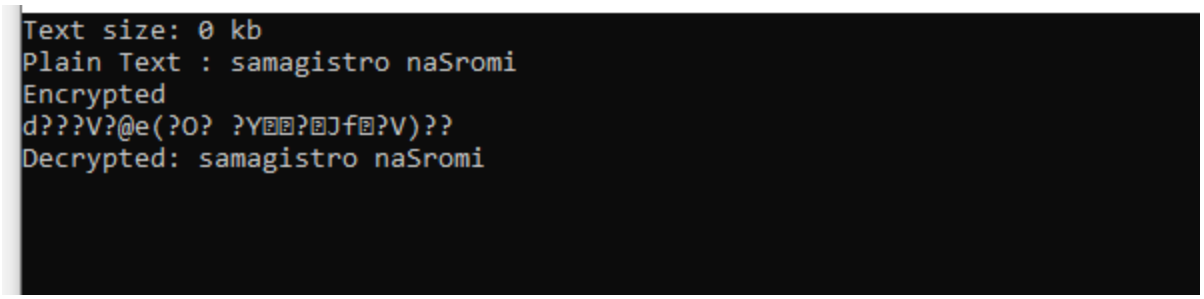
.

სურათი. 6. ბლოკის დაშიფვრის იმპლემენტაცია.

დაშიფვრასთან ერთად აუცილებელია დეშიფრაციის ალგორითმის იმპლემენტაცია, რადგან დავრწმუნდეთ, რომ ალგორითმი სწორად და გამართულად მუშაობს , ამისათვის დავწერე მისი იმპლემენტაცია როგორც სურათი. 7 ზე არის მოცემული. საბოლოოდ როგორც შიფრაცია, ასევე დეშიფრაცია გამართულად მუშაობენ. (იხ. სურათი 8.)

```
public void InvCipher(byte[] input, byte[] output)  // decipher 16-bit input
{
    this.State = new byte[4, Nb];  // always [4,4]
    for (int i = 0; i < (4 * Nb); ++i)
    {
        this.State[i % 4, i / 4] = input[i];
    }

    AddRoundKey(Nr);

    for (int round = Nr - 1; round >= 1; --round)  // main round loop
    {
        InvShiftRows();
        InvSubBytes();
        AddRoundKey(round);
        InvMixColumns();
    }  // end main round loop for InvCipher
    InvShiftRows();
    InvSubBytes();
    AddRoundKey(0);

    for (int i = 0; i < (4 * Nb); ++i)
    {
        output[i] = this.State[i % 4, i / 4];
    }


}  // InvCipher()

1 reference
```

სურათი 7. AES-ში, ბლოკის დეშიფრაციის იმპლემენტაცია 10 რაუნდის შემთხვევაში.



სურათი 8. AES-ის იმპლემენტაციის შედეგად დაშიფრული და დეშიფრირებული შედეგები.

AES-ის მოდიფიცირებული ვარიანტის იმპლემენტაცია

შემდეგი ეტაპი არის არსებული სტანდარტული AES-ის მოდიფიცირებული ვარიანტის იმპლემენტაცია, რომელზეც უკვე უნდა ჩატარდეს დეტალური კვლევა. ამოსათვის AES-ის კოდში შევიდა მნიშვნელოვანი ცვლილებები , პირველ რიგში გაკეთდა მატრიცის მატრიცაზე

```
public void multipleMatrix()
{
    int i, j, k = 0;
    var crr1 = new int[4, 4];
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            crr1[i, j] = 0;
    for (i = 0; i < 4; i++)    //row of first matrix
    {
        for (j = 0; j < 4; j++)    //column of second matrix
        {
            var sum = 0;
            for (k = 0; k < 4; k++)
                sum = sum + State[i, k] * selfInversing[k, j];
            int r = sum % 256;
            crr1[i, j] = r < 0 ? r + 256 : r;
        }
    }
    for (i = 0; i < 4; i++)    //row of first matrix
    {
        for (j = 0; j < 4; j++)    //column of second matrix
        {
            this.State[i, j] = (byte)crr1[i, j];
        }
    }
}
```

სურათი 8. მატრიცის გამრავლება მატრიცაზე მოდულით 256

გამრავლების ფუნქცია, რომელიც , ალგორითმის მიხედვით მდგომარეობის მატრიცას ამრავლებს თვითშებრუნებადი მატრიცაზე, რათქმაუნდა მოდულით 256 (იხ. სურათი 8).

აქედანვე ჩანს რომ ეს გამრავლები ფუნქცია რამდენად მძიმე ქმედებაა ოპერაციების რაოდენობის მხრივ.

```
private int[,] selfInversing ={{ 2,-1,-2, 2 },
                               {-1,-2,-2,-2 },
                               { 1, 1, 1, 2 },
                               {-1, 1, 2,-1 }
                              };
```

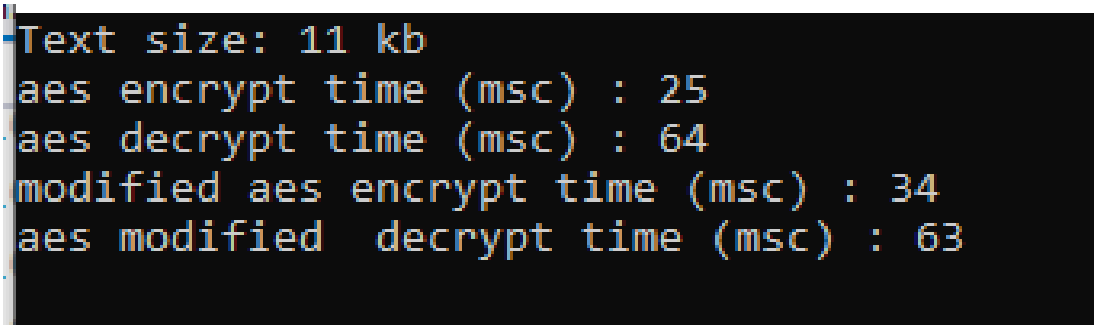
თვით შებრუნებადი მატრიცის იმპლემენტაცია. რომელიც გამოყენებულია გამრავლების ფუნქციაში.

შემდეგ ეტაპზე კი ზემოთ აღწერილ არა მოდიფიცირებულ იმპლემეტაციაში , მიღებული ფუნქციით ჩანაცვლდა ShiftRow და რაუნდების რაოდენობა საწყის ეტაპზე ავიღე ერთის ტოლი რის შედეგადაც სიჩქარე ბევრად უკეთესი იყო ვიდრე სტანდართული იმპლემენტაციის

იხილეთ სურათი 9.

```
Text size: 11 kb
aes encrypt time (msc) : 27
aes decrypt time (msc) : 66
modified aes encrypt time (msc) : 7
aes modified  decrypt time (msc) : 10
```

ხოლო იმის დასადგენად თუ რამდენამდე შეიძლებოდა რაუნდების გაზრდა , ეტაპობრივათ გავზარდე მნიშვნელობა და 8 -ის მნიშვნელამდე სიჩქარე შედარებით უკეთესი იყო ხოლო უკვე 8 და მასზე მეტი მნიშვნელობისთვის სიჩქარე ჩამოუვარდება სტანდარტული იმპლემენტაციისას. იხ.სურათი 10.



სურათი 10. 8 რაუნდის შემთხვევაში სიჩქარე ჩამოუვარდება სტანდარტულისას.

საბოლოო ჩვენმა კოდმა მიიღო სახე (იხ. სურათი 11) სადაც multipleMatrx ფუნქცია არის მატრიცის მატრიცაზე გამრავლება , და მან ამოაგდო ShifRows ფუნქცია, Nr კი წარმოადგენს რაუნდების რაოდენობას რომელიც განისაზღვრება წინასწარ ბლოკის ბიტები რაოდენობის მიხედვით, მაგრამ ჩვენ შემთხვევაში ავიღეთ 1 .

```
public void Cipher(byte[] input, byte[] output)  // encipher 16-bit input
{
    this.State = new byte[4, Nb];  // always [4,4]
    for (int i = 0; i < (4 * Nb); ++i)
    {
        this.State[i % 4, i / 4] = input[i];
    }

    AddRoundKey(0);

    for (int round = 1; round <= Nr; ++round)  // main round loop
    {
        SubBytes();
        multipleMatrix();
        MixColumns();
        AddRoundKey(round);
    }  // main round loop

    for (int i = 0; i < (4 * Nb); ++i)
    {
        output[i] = this.State[i % 4, i / 4];
    }

}  // Cipher()
```

სურათი 11. მოდიფიცირებული AES-ის ბლოკის დაშიფვრის იმპლემენტაცია.

შესაბამისად გადაკეთდა დეშიფრაციის ლოგიკაც და მიიღო შემდეგი სახე როგორც სურათი. 12. ზეა

```
public void InvCipher(byte[] input, byte[] output)  // decipher 16-bit input
{
    this.State = new byte[4, Nb];  // always [4,4]
    for (int i = 0; i < (4 * Nb); ++i)
    {
        this.State[i % 4, i / 4] = input[i];
    }

    AddRoundKey(Nr);

    for (int round = Nr-1; round >= 0; --round)  // main round loop
    {
        InvMixColumns();
        multipleMatrix();
        InvSubBytes();
        AddRoundKey(round);
    }  // end main round loop for InvCipher

    for (int i = 0; i < (4 * Nb); ++i)
    {
        output[i] = this.State[i % 4, i / 4];
    }

}  // InvCipher()
```

სურათი 12.

როგორც სურათიდან ჩანს, მდგომარეობის მატრიცისა და თვითშებრუნებადი მატრიცის ნამრავლის ფუნქციას იმავეს ვიყენებთ , რომელსაც შიფრაციის დროს, რადგან ჩვენს მიერ აღებული მატრიცის შებრუნებული ისევ თავისი თავია და ამიტომ არ გვიწევს ამ ფუნქციის ცვლილება დეშიფრაციისთვის.

საბოლოოდ აიწყო სრულად გამართული შიფრაციის და დეშიფრაციის ფუნქციები რომელებიც მუშაობის სისწრაფით სავსებით მისაღები არის, ამიტომ შემდეგი ეტაპი არის მიღებული შედეგის ტესტირება და შესაბამისად ამ ალგორითმის ავკარგიანობის დადგენა ამ ტესტების შედეგით.

შემთხვევით რიცხვების გენერატორის ტესტები

NIST(სტანდარტებისა და ტექნოლოგიების ნაციონალური ინსტიტუტი) გვთავაზობს მათ მიერ შემუშავებულს ტესტირების პაკეტს[2], რომელიც დაწერილია C ენაზე. ის შედგება 15 ტესტისაგან, რომლებიც შემუშავდა ჰარდვეარის ან პროგრამული კრიპტოგრაფიული შემთხვევითი ან ფსევდო შემთხვევითი გენერატორების მიერ წარმოებული ორობითი მიმდევრობების გასატესტად. ისინი ეძებენ ამ მიმდევრობაში სხვადასხვა ტიპის განსხვავებ არა-შემთხვევით მიმდევრობებს რომლებიც შეიძლება არსებობდნენ ორობით რიგში. ეს 15 ტესტია: (როგორც პროგრამირებაში აქაც ზოგიერთი ფრაზის ან სიტყვის ზუსტი შესატყვისი ვერ მოიძებნა)

1. სიხშირე (Monobit) ტესტი, (The Frequency (Monobit) Test)

2. სიხშირის ტესტი ბლოკში (Frequency Test within a Block)

3. The Runs Test

4. ტესტი ერთიანების უდიდესი მიმდევრობის მოსაძებნად ბლოკში (Tests for the Longest-Run-of-Ones in a Block)

5. ორობითი მატრიცის რანკის ტესტი (The Binary Matrix Rank Test)

6. დისკრეტული ფურიეს გარდაქმნის (სპექტრალური) ტესტი (The Discrete Fourier Transform (Spectral) Test)

7. The Non-overlapping Template Matching Test,

8. The Overlapping Template Matching Test,

9. მაურერის "უნივერსალური სტატისტიკური" ტესტი(Maurer's "Universal Statistical" Test)

10. ხაზოვანი სირთულის ტესტი (The Linear Complexity Test)

11. The Serial Test

12. The Approximate Entropy Test

13. The Cumulative Sums (Cusums) Test

14. The Random Excursions Test

15. The Random Excursions Variant Test

ახალი შექმნილმა ალგორითმის მიერ , მიღებულმა შედეგი ყველა 15 ივე ტესტზე შემოწმდა და გთავაზობთ თითოეული ტესტის აღწერასა და ჩემს მიერ მიღებულ შედეგს.

ჩვენს მიერ განხილული თითოეული ტესტი ითვლის P-value (P-მნიშვნელობა)-ს, რომლის მნიშვნელობაზეცაა ტესტის ჩაბარება დამოკიდებული.

1. სიხშირე (Monobit) ტესტი, (The Frequency (Monobit) Test

ეს ტესტი ფოკუსირებულია მიმდევრობაში ერთიანების და ნოლების პროპორციის დადგენაზე.[2] მისი მიზანია დაადგინოს ერთიანების და ნოლების რაოდენობა რამდენადაა ახლოს , როგორც ნამდვილ შემთხვევით მიმდევრობებშია მოსალოდნელი. ტესტი აფასებს ახლოსაა ერთიანების რაოდენობა შედეგში ½ თან, ამ შემთხვევაში 0-ებისა და 1-იანების განაწილება მიმდევრობაში იქნება ერთნაირი. ყველა შემდგომი ტესტი დამოკიდებულია ამ ცდის ჩაბარებაზე.

ამ ტესტის ფუნქცია გამოიყურება ასე: Frequency(n) სადაც n არის ბიტური მიმდევრობის სიგრძე , ჩვენ შემთხვევაში ავიღე n=1000 , NIST -ის რეკომენდაციაა რომ n ის მნიშვნელობა მეტი ან ტოლი უნდა იყოს 100-ის. ამ შემთხვევაში ზემოთ ნახსენების P-value >= 0.01, წინააღმდეგ შემთხვევაში მიმდევრობა არაა შემთხვევითი, და ან ერთიანების რაოდენობაა ბევრად მეტი ნოლებზე ან პირიქით რაც იწვევს ამ მნიშვნელობის შემცირებას და ტესტის ვერ ჩაბარებას.

როგორც ყველა ტესტში ამ შემთხვევაშიც შევადარებთ, შემთხვევითი მიმდოვრების შედეგს და ჩვენს მიერ დაგენერირებული მიმდევრობას, განსხვავების დასანახად.

ესაა წარმატებით ჩაბარების მაგალითი, რომელიც მოგვცა AES-ის სტანდარტულმა იმპლემენტაციამ, თითოეული მნიშვნელობა არის 10 განსხვავებული 1000-ის ტოლი სიგრძის მქონე ბიტური მიმდევრობაზე ჩატარებული ტესტის P-მნიშვნელობები. საიდანაც კარგად ჩანს, რომ AES-ის მიერ დაბრუნებული შედეგი ნამდვილადაა შემთხვევითი მიმდევრობები.

```
0.779478
0.703945
0.280142
0.794864
0.674485
0.920344
0.347218
0.012419
0.327086
0.298340
```

ხოლო ჩვენს მიერ მოდიფიცირებული ალგორითმის იმპლემენტაციის მიერ მიღებული დაშიფრული ტექსტის ბიტურ მიმდევრობაზე, იმავე პირობებით, როგორითაც სტანდარტულმა გაირა აჩვენა შემდეგი შედეგი

```
0.704336
0.410968
0.612882
0.657969
0.100097
0.612882
0.751830
0.569214
0.751830
0.704336
```

როგორც მივხვდით თითოეულ შემთხვევაში P-მნიშვნელობა მეტია 0,01 ზე რაც ავტომატურად ნიშნავს რომ ალგორითმის მიერ დაგენერირებული შდეგი არის შემთხვევითი მიმდევრობა.

ამის შემდეგ თავისუფლად შეგვიძლია გავაგრძელოთ ტესტირება. რადგან ამ ტესტის ჩავარდნა ავტომატურად ნიშნავდა სხვა ტესტების უარყოფით შედეგს.

2. სიხშირის ტესტი ბლოკში (Frequency Test within a Block)

ვიდრე აღწერაზე გადავალ განვმარტოთ სიმბოლო M და N რომელიც გამოიყენება ამ ტესტში.

M არის ბიტების რაოდენობა ერთ ბლოკში. ჩვენ შემთხვევაში მისი მნიშვნელობა იქნება 128.

N არის M სიგრძის ბლოკების რაოდენობა რომელიც უნდა გაიტესტოს.

ეს ტესტი ფოკუსირებულია M სიგრძის ბლოკში ერთიანების პროპორციის დადგენაზე. მისი მიზანია დაადგინოს ერთიანების სიხშირე რამდენადა არის ახლოს M სიგრძის ბლოკში M/2-თან , როგორც შემთხვევით მიმდევრობებშია მოსალოდნელი. თუ ტესტში M იქნება 1 ის ტოლი მაშინ იგი დადის წინა ტესტის დონეზე (1. სიხშირე (Monobit) ტესტი, (The Frequency (Monobit) Test);

რაც შეეხება ტექსტის ფუნქციურ ნაწილს, იგი გამოიძახება როგორც BlockFrequency(M,n), სადაც M ვიცით უკვე რაც არის, ხოლო n ამ შემთხვევაშიც ბიტური მიმდევრობის სიგრძეა რომელზეც სრულდება ეს ფუნქცია. ის აბრუნებს P-მნიშვნელობას რომელიც წინა ტესტის მსგავსად არ უნდა იყოს ნაკლები 0,01 -ზე.

რეკომენდირებულია რომ n მინიმალური მნიშვნელობა იყოს 100 , ასევე $n \geq MN$, ხოლო M უნდა აკმაყოფილებდეს შემდეგ პირობებს $M \geq 20$, $M > 0.01n$ და $N < 100$.

ჩვენ ავიღეთ შემდეგი მნიშვნელობები

M = 128

n = 1280

გავატარეთ ჩვენი ალგორითმის მიერ დაბრუნებული შედეგი ამ ტესტზე, ავიღეთ n სიგრძის 10 ბიტური მიმდევრობა და ჩავატარეთ ტესტი (იხილეთ ტესტი II -ის შედეგი). სადაც ნათლად ჩანს, რომ ალგორითმა წარმატებით გაიარა ეს ცდაც.

```
0.421516
0.909157
0.979915
0.435027
0.546994
0.938492
0.429596
0.970186
0.355089
0.698653
```

ტესტი II -ის შედეგი. თითოეული წარმოადგენ P-მნიშვნელობას სხვადასხვა მიმდევრობისათვის.

3. The Runs Test

ეს ტესტი ფოკუსირებულია იპოვოს საერთო რიცხვი ერთნაირი ბიტების გადაბმის მიმდევრობაში,[2] სადაც ეს გადაბმა შემოსაზღვრული ამ გადაბმაში მონაწილე ბიტისგან განსხვავებული ბიტით , მაგალითად ნოლიანების მიმდევრობა რომელიც ტავში და ბოლოში სჰემოსაზგვრულია ერთიანებით და პირიქით. მიზანი ტესტის არის იმის განსაზღვრა, არის თუ არა სხვადასხვა სიგრძის ერთიანების და ნოლების გადაბმების სიგრძეები, როგორც ეს მოსალოდნელია შემთხვევითი თანმიმდევრობისთვის. კერძოდ, ეს ტესტი განსაზღვრავს თუ რყევები (გადასვლები ნოლიანებიდან ერტიანებსჰი ან პირიქით) არის ძალიან სწრაფ ან ძალიან ნელი.

ტესტის ფუნქცია Runs(n) იღებს ბიტური სტრიქონს სიგრძეს, როლის მიხედვითაც იგი ალგორითმის შედეგიდან ამოიღებს ზუსტად n-ური სიგრძის ბიტურ მიმდევრობას და ჩაატარებს მასზე გამოთვლებს.

მისი შედეგი ანუ P-მნიშვნელობა განსაძღვრავს მიმდევრობის შემთხვევითობას. როგორც წინა ტესტებში.

ასევე ამ ტესტ გააჩნია $V_n(\text{abs})$ რომლის მნიშვნელობაც განსაზღვრავს რყევების სიხშირეს, სწრაფია თუ ნელი ეს ემრყეობა.

როგორც ვთქვით ტესტის ფუნქცია იღებს მხოლოდ n მნიშვნელობას ამიტომ როგორც პირველ ტესტში, ჩვენი ალგორითმის მიერ დაბრუნებული შედეგი ამ ტესტზე, ავიღეთ n=1000 სიგრძის 10 ბიტური მიმდევრობა და ჩავატარეთ ტესტი (იხილეთ ტესტი III -ის შედეგი). სადაც ნათლად ჩანს, რომ ალგორითმა წარმატებით გაიარა ეს ცდაც. ასევე V_n(abs) მაღალი მნიშვნელობა ნიშნავს რომ მერყეობა 0 დან 1 ში საკმაოდ სწრაფია.

```
                RUNS TEST
-----------------------------------------
COMPUTATIONAL INFORMATION:
-----------------------------------------
(a) Pi                          = 0.494000
(b) V_n_obs (Total # of runs) = 496
(c) V_n_obs - 2 n pi (1-pi)
    -----------------------     = 0.175691
      2 sqrt(2n) pi (1-pi)
-----------------------------------------
p_value = 0.803775
```

```
0.803775
0.720090
0.956008
0.608467
0.362737
0.793981
0.571323
0.533671
0.057339
0.373402
```

ტესტი(Runs test) III -ის შედეგი.

მარცხენა მხარეს V_n(abs) მნიშვნელობა არის 496 რაც მაღალია 1000 ბიტის მიმდევრობისათვის, ხოლო მარცხნივ არის P-მნიშვნელობები, რაც ადასტურებს მიმდევრობების შემთხვევითობას.

4. ტესტი უდიდესი მიმდევრობის მოსაძებნად ბლოკში (Tests for the Longest-Run-of-Ones in a Block)

ეს ტესტის ინტერესია ერთიანები ყველაზე გრძელი მიმდევრობის მონახვა M ბიტის სიგრძის ბლოკში[2]. მისი მიზანია განსაზღვროს სიგრძე ერთიანების უდიდესი მიმდევრობი გატესტილ მიმდევრობაში და რამდენად ემთხვევა ეს შედეგი შემთხვევითი მიმდევრობისას მოსალოდნელ შედეგს. აღსანიშნავია. გასათვალისწინებელია, რომ ერთიანების მიმდევრობის ეს მნიშვნელობა ასევე გვაძლევს მოსალოდნელ მნიშვნელობას ნოლიანების მიმდევრობისა და შესაბამისად ვტესტავთ მხოლოდ ერთიანებისათვის.

ამ ტესტის ფუნქცია იღებს LongestRunOfOnes(n) იღებს მხოლოდ n ბიტური სტრიქონის სიგრძის მნიშვნელობას ხოლო M ბლოკის სიგრძის მნიშვნელობები შეიძლება იყო M = 8, M = 128 და M = 10^4 რომლებიც გამოითვლებიან n-ის სიგრძის მიხედვით

n ის მინიმალური მნიშვნელობა	M
128	8
6272	128
750000	10^4

ტესტი ასევე ითვლის χ^2(abs) -ს, რომლის ზომაც განსაძღვრავს, რამდენად კარგადაა განასწილებული ერთიანების მიმდევრობები ბლოკში. შესაბამისად დიდი მნიშვნელობის შემთხვევაში მიმდევრობებში ქვექნება ერთიანების დიდი დაჯგუფებები , ხოლო მცირე ზომის შემემთხვევაში პირიქით.

ჩვენ შემთხვევაში ტესტირებისათვის ავიღე n=10000 შესაბამისად M = 128 და გავატარეთ ტესტი. რომელმაც აჩვენა რომ P - მნიშვნელობა აქაც მივიღეთ 0,01 ზე მეტი რაც ნიშნავს რომ მიმდევრომა შემთხვევითია, ხოლო χ^2(abs) მნიშვნელობა დაბალი, ამიტომ P-მნიშვნელობა გამოვიდა საკმაოდ მაღალი, რაც უფრო დიდია χ^2(abs)-ის ზომა P-მნიშვნელობაც მცირდება

```
          LONGEST RUNS OF ONES TEST
---------------------------------------------
COMPUTATIONAL INFORMATION:
---------------------------------------------
(a) N (# of substrings)  = 78
(b) M (Substring Length) = 128
(c) Chi^2                = 0.670176
---------------------------------------------
      F R E Q U E N C Y
---------------------------------------------
<=4  5  6  7  8  >=9 P-value  Assignment
   8  18  22  14   7    9 SUCCESS             p_value = 0.984565
```

```
          LONGEST RUNS OF ONES TEST
-------------------------------------------
COMPUTATIONAL INFORMATION:
-------------------------------------------
(a) N (# of substrings)  = 78
(b) M (Substring Length) = 128
(c) Chi^2                = 4.813229
-------------------------------------------
      F R E Q U E N C Y
-------------------------------------------
<=4  5  6  7  8  >=9 P-value  Assignment
   7  19  18  19  10    5 SUCCESS              p_value = 0.439097
```

ამ სურათიდან ნათლად ჩანს რომ χ^2(abs) ის ზომა გავლენას ახდენს P-მნიშვნელობაზე.

5. ორობითი მატრიცის რანკის ტესტი (The Binary Matrix Rank Test)

```
                        RANK TEST
-------------------------------------------
COMPUTATIONAL INFORMATION:
-------------------------------------------
(a) Probability P_32 = 0.288788
(b)             P_31 = 0.577576
(c)             P_30 = 0.133636
(d) Frequency   F_32 = 11
(e)             F_31 = 19
(f)             F_30 = 8
(g) # of matrices    = 38
(h) Chi^2            = 2.077158
(i) NOTE: 0 BITS WERE DISCARDED.
-------------------------------------------
SUCCESS         p_value = 0.353957
```

6. დისკრეტული ფურიეს გარდაქმნის (სპექტრალური) ტესტი (The Discrete Fourier Transform (Spectral) Test)

```
                              FFT TEST
                    ------------------------------------------
                    COMPUTATIONAL INFORMATION:
                    ------------------------------------------
                    (a) Percentile = 94.600000
                    (b) N_l        = 473.000000
                    (c) N_o        = 475.000000
                    (d) d          = -0.580381
                    ------------------------------------------
SUCCESS             p_value = 0.561658
```

7. The Non-overlapping Template Matching Test,

```
                NONPERIODIC TEMPLATES TEST
------------------------------------------------------------------------------
                COMPUTATIONAL INFORMATION
------------------------------------------------------------------------------
        LAMBDA = 0.234375       M = 128 N = 8   m = 9   n = 1024
------------------------------------------------------------------------------
               F R E Q U E N C Y
Template   W_1  W_2  W_3  W_4  W_5  W_6  W_7  W_8    Chi^2   P_value Assignment Index
------------------------------------------------------------------------------
000000001    0    1    0    0    1    0    0    1  8.412121 0.394282 SUCCESS   0
000000011    0    0    0    0    0    0    0    0  1.818182 0.986087 SUCCESS   1
000000101    0    0    0    0    1    0    0    1  6.214141 0.623259 SUCCESS   2
000000111    0    0    0    0    0    0    0    0  1.818182 0.986087 SUCCESS   3
000001001    0    1    0    0    2    0    0    0 16.686869 0.033540 SUCCESS   4
000001011    0    0    0    0    0    1    0    1  6.214141 0.623259 SUCCESS   5
000001101    0    1    0    0    0    0    0    1  6.214141 0.623259 SUCCESS   6
000001111    0    0    1    0    0    1    0    0  6.214141 0.623259 SUCCESS   7
000010001    1    1    0    0    0    0    0    0  6.214141 0.623259 SUCCESS   8
```

9. მაურერის "უნივერსალური სტატისტიკური" ტესტი(Maurer's "Universal Statistical" Test)

```
                UNIVERSAL STATISTICAL TEST
                -------------------------------------------
                COMPUTATIONAL INFORMATION:
                -------------------------------------------
                (a) L         = 7
                (b) Q         = 1280
                (c) K         = 141577
                (d) sum       = 877055.238169
                (e) sigma     = 0.002768
                (f) variance  = 3.125000
                (g) exp_value = 6.196251
                (h) phi       = 6.194899
                (i) WARNING:  1 bits were discarded.
                -------------------------------------------
SUCCESS         p_value = 0.625410
```

10. ხაზოვანი სირთულის ტესტი (The Linear Complexity Test)

```
-----------------------------------------------------
       L I N E A R  C O M P L E X I T Y
-----------------------------------------------------
       M (substring length)     = 500
       N (number of substrings) = 2000
-----------------------------------------------------
       F R E Q U E N C Y
-----------------------------------------------------
  C0   C1   C2   C3   C4   C5   C6    CHI2    P-value
-----------------------------------------------------
       Note: 0 bits were discarded!
  17   51  255 1000  510  128   39  3.399921 0.757234
```

11. The Serial Test

```
                        SERIAL TEST
                ----------------------------------------------
                 COMPUTATIONAL INFORMATION:
                ----------------------------------------------
                (a) Block length    (m) = 2
                (b) Sequence length (n) = 1000000
                (c) Psi_m               = 18.732976
                (d) Psi_m-1             = 9.363600
                (e) Psi_m-2             = 0.000000
                (f) Del_1               = 9.369376
                (g) Del_2               = 0.005776
                ----------------------------------------------
FAILURE         p_value1 = 0.009236
SUCCESS         p_value2 = 0.939419
```

12. The Approximate Entropy Test

```
                        APPROXIMATE ENTROPY TEST
                ------------------------------------------
                COMPUTATIONAL INFORMATION:
                ------------------------------------------
                (a) m (block length)    = 5
                (b) n (sequence length) = 100000
                (c) Chi^2               = 46.130286
                (d) Phi(m)              = -3.465528
                (e) Phi(m+1)            = -4.158445
                (f) ApEn                = 0.692917
                (g) Log(2)              = 0.693147
                ------------------------------------------
SUCCESS         p_value = 0.050651
```

13. The Cumulative Sums (Cusums) Test

```
                      CUMULATIVE SUMS (FORWARD) TEST
                -----------------------------------------
                COMPUTATIONAL INFORMATION:
                -----------------------------------------
                (a) The maximum partial sum = 404
                -----------------------------------------
SUCCESS         p_value = 0.402556

                      CUMULATIVE SUMS (REVERSE) TEST
                -----------------------------------------
                COMPUTATIONAL INFORMATION:
                -----------------------------------------
                (a) The maximum partial sum = 341
                -----------------------------------------
SUCCESS         p_value = 0.559334
```

14. The Random Excursions Test

```
                          RANDOM EXCURSIONS TEST
                -----------------------------------------
                COMPUTATIONAL INFORMATION:
                -----------------------------------------
                (a) Number Of Cycles (J) = 1490
                (b) Sequence Length (n)  = 1000000
                (c) Rejection Constraint = 500.000000
                -----------------------------------------
SUCCESS         x = -4 chi^2 =  2.116987 p_value = 0.832732
SUCCESS         x = -3 chi^2 =  0.681615 p_value = 0.983962
SUCCESS         x = -2 chi^2 =  4.003613 p_value = 0.548896
SUCCESS         x = -1 chi^2 =  1.762416 p_value = 0.880944
SUCCESS         x =  1 chi^2 =  5.144966 p_value = 0.398446
SUCCESS         x =  2 chi^2 =  1.883404 p_value = 0.865032
SUCCESS         x =  3 chi^2 =  0.633042 p_value = 0.986440
SUCCESS         x =  4 chi^2 =  2.133962 p_value = 0.830316
```

15. The Random Excursions Variant Test

```
                    RANDOM EXCURSIONS VARIANT TEST
               ---------------------------------------------
               COMPUTATIONAL INFORMATION:
               ---------------------------------------------
               (a) Number Of Cycles (J) = 1490
               (b) Sequence Length (n)  = 1000000
               ---------------------------------------------
SUCCESS        (x = -9) Total visits = 1177; p-value = 0.164338
SUCCESS        (x = -8) Total visits = 1148; p-value = 0.105748
SUCCESS        (x = -7) Total visits = 1182; p-value = 0.117620
SUCCESS        (x = -6) Total visits = 1260; p-value = 0.203960
SUCCESS        (x = -5) Total visits = 1343; p-value = 0.369393
SUCCESS        (x = -4) Total visits = 1379; p-value = 0.442167
SUCCESS        (x = -3) Total visits = 1443; p-value = 0.700209
SUCCESS        (x = -2) Total visits = 1536; p-value = 0.626608
SUCCESS        (x = -1) Total visits = 1523; p-value = 0.545502
SUCCESS        (x =  1) Total visits = 1510; p-value = 0.714088
SUCCESS        (x =  2) Total visits = 1480; p-value = 0.915771
SUCCESS        (x =  3) Total visits = 1348; p-value = 0.244704
SUCCESS        (x =  4) Total visits = 1230; p-value = 0.071832
SUCCESS        (x =  5) Total visits = 1180; p-value = 0.058368
SUCCESS        (x =  6) Total visits = 1141; p-value = 0.053903
SUCCESS        (x =  7) Total visits = 1089; p-value = 0.041615
SUCCESS        (x =  8) Total visits = 1067; p-value = 0.045422
SUCCESS        (x =  9) Total visits = 1027; p-value = 0.039680
```

ბოლო 10 ტესტის შედეგი მოგაწოდეთ პირდაპირ და როგორი შედეგი მივიღეთ, რადგან მათი ახსნა სათითაოდ უკვე აღარ შედის ამ კვლევის ფარგლებში. თითოეულ ტესტზე დადებითი შედეგი ნიშნავს რომ ჩვენი ალგორითმი გამართულად მუშაობს.

დასკვნა

კვლევის მიზანი იყო წინასწარ შემუშავებული თეორიის[1] მიხედვით პროგრამული უზრუნველყოფის შექმნა, შემდგომ მისი ანალიზი NIST ის მიერ შემოთავაზებული შემთხვევითი და ფსევდო შემთხვევითი რიცხვების გენერატორის ტესტებზე[2]. შემუშავდა და დაიწერა შესაბამისი კოდი, რომელიც მუშაობს გამართულად, ხოლო მისმა შედეგებმა, ანუ შიფრაციის შედეგად მიღებულმა ბიტურმა სტრიქონებმა , გაიარეს ზემოთ ხსენებული ტესტები. თითოეულმა ცდამ აჩვენა რომ გენერირებული შედეგი წარმოადგენ შემთხვევით მიმდევრობებს და შესაბამისად ჩვენმა ცვლილებებმა, AES-ის სტრუქტურაში, არ შეუცვლია ალგორითმის თვისებები, პირიქით გაიზარდა ალგორითმის სისწრაფე, რაც ჩვენი , მიზანიც იყო.

მიუხედავად ამისა, ჩემს მიერ შემუშავებული პროგრამული უზრუნველყოფა დაწერილია C# ზე ,რაც არ იძლევა რეალური სიჩქარე დადგინდეს ამ ალგორითმის. ამისათვის ამ ალგორითმის შემდგომი შემოწმება უნდა მოხდეს C ენაზე. რომელსაც აქვს საშუალება ჰარდვეარის დონეზე მოხდეს ალგორითმის იმპლემენტაცია, და ნათლად გამოჩნდება მატრიცების ნამრავლის ფუნქციონალი როგორ სიჩქარეს აჩვენებს და ნამდვილად იქნება ისეთი სწრაფი როგორც ეს ჩვენ შემთხვევაში აჩვენა?.

საბოლოოდ კვლევამ სირთულიდან გამომდინარე საკმაოდ დიდი დრო წაიღო და შედეგიც სასიამოვნო მივიღეთ. შემდეგი კვლევა აუცილებელია რადგან მიღებული რეზულტატები ცხადყოფს, რომ ალგორითმს დიდი პოტენციალი გააჩნია და შესაძლებელია კიდევ უფრო დახვეწა და განვითარება.

გამოყენებული ლიტერატურა

[1] Z. Kochladze Modified Version of the Hill's Algorithm. GESJ: Computer Sciences and Telecommunication. No.3 (43), 2014, pp. 33-36.

[2] NIST SP 800-22, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications