LATVIJAS UNIVERSITĀTE
FIZIKAS, MATEMĀTIKAS UN OPTOMETRIJAS FAKULTĀTE
MATEMĀTIKAS NODAĻA

DROŠS ATTĒLU ŠIFRĒŠANAS ALGORITMS, IZMANTOJOT *XOR* OPERATORU UN RUBIKA KUBU

BAKALAURA DARBS

Autors: **Jānis Rekners**
Studenta apliecības Nr.: jr18025
Darba vadītājs: Dr. Math Raivis Bēts

RĪGA 2022

ANOTĀCIJA

Šajā bakalaura darbā "Drošs fotogrāfiju šifrēšanas algoritms, izmantojot XOR operatoru un Rubika kubu" ir izveidots iztirzājums par fotogrāfiju šifrēšanas algoritmu. Darbā ir definēti pamatjēdzieni, kuri nepieciešami, lai izprastu algoritmu, aprakstīti šifrēšanas un atšifrēšanas soļi, aplūkoti piemēri, veikta algoritma praktiskā analīze, kā arī izdarīti secinājumi.

Atslēgvārdi: kriptogrāfija, XOR operators, Rubika kubs, fotogrāfiju šifrēšana

ABSTRACT

This bachelor thesis named “A Secure Image Encryption Algorithm Based on the XOR Operator and the Rubik’s Cube” contains a description of an image encryption algorithm. This paper contains basic concepts which will be needed to better understand the algorithm, a description of the steps of encryption and decryption, a few examples, practical analysis of the algorithm and conclusions.

Keywords: cryptography, XOR operator, Rubik’s cube, image encryption

SATURA RĀDĪTĀJS

IEVADS

20. gadsimta tehnoloģiskā revolūcija nozīmēja, ka daudz svarīgāka kļuva dokumentu aizsargāšana, lai pie tiem nepiekļūtu nelabvēlīgi cilvēki vai to grupas. Tika izstrādātas vairākas tehnikas, ar kurām cilvēki digitālajā vidē varēja pārsūtīt viens otram datus ar samazinātu risku kādam citam iejaukties privātajā dzīvē, noslēpumu dalīšanā vai vispār jebkādas citas svarīgas informācijas dalīšanā. Mūsdienās lielākā daļa izsmalcinātu šifrēšanas algoritmu sevī ietver sarežģītu matemātiku, kas pēc iespējas vairāk apgrūtina tās atšifrēšanu tiem, kam šī informācija nav paredzēta.

Šajā bakalaura darba apskatītajā metodē īpaša uzmanība tiek pievērsta fotogrāfiju šifrēšanai. Šādu algoritmu drošība ir plaši pētīta. Ir pierādīts, ka algoritmi, kas ir balstīti uz permutācijām parasti nav īpaši droši, jo mūsdienu datori spēj iziet cauri visām iespējamām kombinācijām bez relatīvām problēmām. Taču, kaut gan tie tiek izmantoti biežāk, algoritmi, kas ir balstīti uz gadījuma kustībām parasti prasa lielāku datora atmiņas apjomu. Tāpēc bieži algoritms sastāv no daļēji permutācijām un gadījuma kustībām.

"Rubika kubs" ir plaši atpazīstama un slavena puzle. Tas ir kubs ar unikālu krāsu un 9 uzlīmēm uz katras skaldnes. Oriģinālajā puzlē ir iekšējs mehānisms, kas ļauj katru skaldi pagriezt neatkarīgi no citām. Tās mērķis ir skaldnes turpināt griezt tā, lai uz katras no 6 kuba skaldnēm atrastos tikai vienas krāsas uzlīmes. Šajā darbā tiks izmantots svarīgs Rubika kuba princips – ja tiek pareizi saliktas piecas no sešām skaldnēm, tad arī sestā būs pareizi salikta, ja uz katras no skaldnēm saliktajā pozīcijā ir unikālas krāsas. Tas ir tāpēc, ka, ja uz pārējām piecām skaldnēm visas uzlīmes jau ir pareizajās pozīcijās, tad uz sestās skaldnes atradīsies tikai atlikušās uzlīmes, jo tām nebūtu citas pozīcijas, kur atrasties.

Šajā darbā tiks piedāvāts algoritms, kas izmanto gan iepriekšminēto Rubika kuba principu, gan XOR operatoru, lai šifrētu attēlus. Šim algoritmam ir divas daļas: pirmā ir Rubika kuba princips: attēlu sadalīšana pa vairākiem attēliem izmērā 3×3 pikseļi, Rubika kuba ģenerēšana ap šiem 3x3 attēliem un to sajaukšana, bet otrā – XOR operatora pielietošana uz fotogrāfijas rindām un kolonnām. Šis viss notiek izmantojot uzģenerētas atslēgas. Šos soļus būs iespējams atkārtot tik iterācijas, cik vēlams šifrēšanai – jo vairāk iterāciju, jo grūtāk atšifrēt trešajai pusei.

Šajā algoritmā arī tiks izmantots tā saucamais "Dieva skaitļa" jeb *God's number* princips, kas nosaka, ka Rubika kubu no jebkuras pozīcijas var atrisināt 20 vai mazāk kustībās ([7]). Šīs kustības tiek definētas kā jebkuras Rubika kuba skaldnes pagriešana par 90 vai 180 grādiem.

“Dieva skaitlis”, ja kustības tiek definētas kā skaldnes pagriešana tikai un vienīgi par 90 grādiem, vēl nav atrasts.

Ir vairāki veidi, kā aprakstīt vienu Rubika kuba kustību. Vispopulārākā notācija Rubika kuba kustībai ir $\{R, L, U, D, F, B\}$ sistēma, kur katrs burts apzīmē Rubika kuba skaldi (R – labā puse, L - kreisā puse, U – augšpuse, D – apakšpuse, F – priekšpuse, B – aizmugure). Piemēram, R apzīmē Rubika kuba labās puses pagriešanu par 90 grādiem pulksteņrādītāja virzienā, bet R' apzīmē Rubika kuba labās puses pagriešanu par 90 grādiem pretējā pulksteņrādītāja virzienā. Rubika kuba labās puses pagriešanu par 180 grādiem apzīmē ar $R2$. Piemēram, viens no populārākajiem Rubika kuba algoritmiem ir tā saucamais *Sune*, kurš tiek aprakstīts ar $R\ U\ R'\ U\ R\ U2\ R'$. Tas, kā Rubika kuba risinātājs tur Rubika kubu attiecībā pret savu skatupunktu nosaka, kuru skaldi apzīmē kā labo pusi, kreiso pusi utt. Lai gan ir daudz veidu, kā apzīmēt vienu Rubika kuba kustību, nav vienas noteiktas notācijas, kā apzīmēt, piemēram, Rubika kuba sajaukšanu vai atrisināšanu, tāpēc šajā darbā šiem mērķiem tiks ieviesta nepieciešamā notācija.

Galvenā darba motivācija bija zinātniskais raksts [1], kurā tiek piedāvāts attēlu šifrēšanas algoritms, kas sevī ietver pikseļu rindu un kolonnu pārvietošanu un pikseļu krāsu maiņu, izmantojot XOR algoritmu. Šī bakalaura darba ietvaros tiek piedāvāts oriģināls attēla šifrēšanas algoritms, kas balstās uz 3-dimensionāliem Rubika kubiem un iepriekšminēto principu par to 5 skaldņu atrisināšanu.

Šī darba mērķis ir aprakstīt attēlu šifrēšanas algoritmu. Lai šos mērķus sasniegtu, darbā tika veikti šādi uzdevumi:

1) algoritma šifrēšanas un dešifrēšanas soļu aprakstīšana,
2) praktisku piemēru veikšana,
3) algoritma ātruma un drošības apskatīšana.

1. PAMATJĒDZIENI

Šajā nodaļā apskatīsim jēdzienus, kas palīdzēs izprast turpmāko saturu. Parasti, kad runā par kriptogrāfiju, ar to saprot teksta šifrēšanu. Līdz ar to daudzi kriptogrāfijas jēdzieni satur vārdu "teksts". Lai gan šis darbs ir saistīts ar attēlu šifrēšanu, šie jēdzienu nosaukumi tiks atstāti tādi, kādi ir, jo no tiem ir viegli saprast, kas ar to ir domāts, ja runā nevis par teksta šifrēšanu, bet par fotogrāfiju šifrēšanu.

1. definīcija. Kortežu

$$\langle P, C, K, E, D \rangle$$

Sauc par kriptosistēmu, ja

- P – pamattekstu kopa (teksts, ko šifrē);
- C – kriptotekstu kopa (šifrēšanas rezultējošais teksts);
- K – atslēgu kopa (ar ko atšifrē/aizšifrē tekstu);
- $E: P \times K \to C$ – šifrs (šifrēšanas, attēlojums);
- $D: C \times K \to P$ – dešifrējošais attēlojums

Šifram jābūt invertējamai funkcijai, citādāk nevarēs atklāt pamattekstu pēc tā aizšifrēšanas. ([2]). Protams, visdrošākais šifrs būtu pamatteksta visa pārveidošana nesaprotamos gadījuma simbolos, bet šis šifrs nav invertējams, un tādējādi , tas nav atšifrējams.

Viens no visvienkāršākajiem šifriem ir katra burta pamattekstā aizstāšana ar citu burtu. Šis šifrs nav īpaši drošs, jo, saprotot vienu vārdu, viss teksts ir daudz vieglāk uzlaužams. Daudz drošāks šifrs ir, piemēram, XOR operatora šifrs, kurā pamattekstu pārveido binārajā kodā izmantojot ASCII tekstu ([3]).

2. definīcija. XOR operators (definēts ar simbolu $\oplus$) ir operācija, kas paņem vektorus A un B, kas sastāv no $\{0,1\}$, kas ir vienāda garuma un saskaita pēc dalījuma ar 2, izveidojot vektoru C.

XOR operators strādā šādi: ja pirmie cipari vektoros A un B neatšķiras, vektora C pirmais cipars būs 0, bet, ja atšķiras, tad 1. Šis attiecas uz visiem cipariem, kas atrodas vektoros A un B, kad veido C.

Piemēram, teksts "*abcd*" ar ASCII palīdzību tiktu pārveidots uz "01100001 01100010 01100011 01100100", kur pirmais burts ir pirmie astoņi cipari, otrais burts ir otrie astoņi utt. Varam izveidot atslēgu, kas ir 8-ciparu binārs skaitlis, piemēram 01000001. Šo atslēgu drīkst zināt tikai lietotāji, kam ir atļauts dalīties ar pamattekstu. Operators aplūko tekstu burtu pa burtam. Ja atslēgas pirmais cipars ir vienāds ar burta pirmo ciparu, tad rezultātā iegūst 0. Bet, ja tā nav, tad iegūst 1. Tālāk, ja atslēgas otrais cipars sakrīt ar burta otro ciparu, tad tā vietā

ieliek 0, citādāk 1. Tā turpina līdz astotajam ciparam, pēc tam sāk nākamo burtu, un tā dara līdz teksta beigām. Mūsu gadījumā kriptoteksts būtu “00100000 00100011 00100010 00100101”, to pārveidojot atpakaļ tekstā sanāktu “ #”%” (izskatās pēc 3 rakstuzīmēm, bet šajā gadījumā burts *a* pārvērtās par atstarpi). Ar XOR operatoru var saprast arī skaitļa atlikumu dalījumā ar 2. Šajā darbā XOR operators tiks lietots tieši šajā kontekstā.

Viena no XOR funkcijas svarīgākajām īpašībām ir tas, ka tā ir invertējama un simetriska. Tas nozīmē, ka, ja šim kriptotekstam “ #”%” pielietotu mums zināmo atslēgu tieši tādā pašā veidā kā šifrējot, mēs iegūtu pamattekstu “*abcd*”. Šis nozīmē, ka saņēmējam šis teksts ir viegli atšifrējams.

Ja runā par attēlu šifrēšanu, tad arī tur ir iespējams pielietot XOR operatoru. Katru pikseli ir iespējams pārveidot par bināru kodu, kas nozīmē, ka ar tiem var strādāt tieši tāpat, kā ar burtiem. Šajā darbā apskatītajā algoritmā XOR operators tiks izmantots, lai matricas rindas un kolonnas pārvietotu par noteiktu pozīciju skaitu. Tālāk apskatīsim, kā strādā šis algoritms.

$$\boldsymbol{A_0} = \begin{pmatrix} \boldsymbol{a} & \boldsymbol{b} & \boldsymbol{c} \\ \boldsymbol{d} & \boldsymbol{e} & \boldsymbol{f} \\ \boldsymbol{g} & \boldsymbol{h} & \boldsymbol{i} \end{pmatrix} \tag{1}$$

Šo ideju var viegli paskaidrot ar matricu. Pieņemsim, ka dota matrica A_0. Šeit matricai pirmās un trešās rindas elementus pārvietosim par vienu vienību pa labi, pēdējo elementu novietojot atpakaļ pirmajā pozīcijā. Šādi izdarot, iegūst matricu A_1.:

$$\boldsymbol{A_1} = \begin{pmatrix} \mathbf{c} & \mathbf{a} & \mathbf{b} \\ \boldsymbol{d} & \mathbf{e} & \mathbf{f} \\ \boldsymbol{i} & \mathbf{g} & \mathbf{h} \end{pmatrix} \tag{2}$$

Kā redzams, matricas A_0 elementi a_{1j} un a_{3j} kļuvuši par attiecīgi a_{1j+1} un a_{3j+1}, ja j = 1, 2, bet elementi a_{13} un a_{33} kļuvuši par attiecīgi a_{11} un a_{31}, kas ilustrē elementu pārvietošanu un nokļūšanu atpakaļ pirmajā pozīcijā. Esam izveidojuši jaunu matricu A_1, taču mūsu mērķis ir “sajaukt” sākotnējo matricu A_0, tāpēc līdzīgi rīkosimies arī ar kolonnām. Pirmo un otro kolonnu pārvietosim par vienu vienību uz augšu, pēdējos elementus noliekot pirmā elementa vietā.

$$\boldsymbol{A_2} = \begin{pmatrix} \mathbf{d} & \mathbf{e} & \mathbf{b} \\ \boldsymbol{i} & \mathbf{g} & \mathbf{f} \\ \boldsymbol{c} & \mathbf{a} & \mathbf{h} \end{pmatrix} \tag{3}$$

Rezultātā izveidojas matrica A_2. Kā redzams, ir izmainījušās visu kolonnu un rindu elementi un to secība. Šajā algoritmā tiks izmantots princips, kas kādas matricas elementus sajauc pa rindām un pa kolonnām. Protams, šis bija tikai piemērs, un uz matricas rindām un kolonnām varēja veikt visādas patvaļīgas kustības, lai to sajauktu.

Protams, līdzīgi ir iespējams izdarīt ar attēliem. Attēli ir veidoti no N rindām un M kolonnām pikseļu. Ja paņem rindas un kolonnas ar šiem pikseļiem, ir iespējams izdarīt līdzīgi minētajam piemēram, iepriekšējās fotogrāfijas vietā izveidojot jaunu.

Šajā darbā arī tiks izmantots iepriekšminētais Rubika kuba princips, kurš nosaka, ka sajauktam kubam saliekot piecas no sešām skaldnēm pareizajās pozīcijās, arī sestā skaldne vienmēr tiks salikta, jo šīs sestās skaldnes uzlīmēm nebūtu cita vieta, kur atrasties, izņemot savās pareizajās pozīcijās. Vienīgais izņēmums būtu tad, ja uz Rubika kuba vienas krāsas uzlīmju vietā uz katras skaldnes atrastos fotogrāfija. Šajā gadījumā ir iespēja, ka attēla centrs var tikt pagriezts par $90 \cdot n$ grādiem ($n \in Z/\{0\}$). Taču šajā darbā aprakstītajā algoritmā šis izņēmums nebūs ietekmējošs faktors, jo "uzlīmes" vietā tiks izmantots viens vienīgs pikselis, kura gadījumā nav svarīgi, vai tas ir pagriezts par $90 \cdot$ n grādiem.

2. ŠIFRĒŠANAS ALGORITMS

Šajā nodaļā tiek aprakstīts algoritms, kuru izmantosim, lai šifrētu attēlus. Vispirms sadalīsim attēlu vairākos mazākos 3×3 pikseļu attēlos. Ja attēla dimensijas to neļauj izdarīt (rindu vai kolonnu pikseļu skaits nedalās ar 3 veselā skaitlī), tiks pievienotas "tukšas" rindas vai kolonnas. Tālāk ap šiem 3×3 attēliem tiks nejauši ģenerētas 5 Rubika kuba skaldnes, uz kurām esošās pikseļu krāsas neatrodas oriģinālajā 3×3 attēlā. Šeit svarīgi tas, ka jebkura krāsa, kas atrodas uz vienas Rubika kuba skaldnes nedrīkst būt identiska ar krāsu, kas ir uz citas Rubika kuba skaldnes, bet krāsas uz vienas skaldnes drīkst būt gan pilnīgi atšķirīgas, gan pilnīgi vienādas. Tālāk visus šos Rubika kubus nejaušā veidā sajauc un saglabā 3×3 pikseļu kombināciju, kas atrodas uz oriģinālās skaldnes. Šādi, tās saliekot kopā, izveidojas jauns attēls. Šajā algoritmā visas pārējās skaldnes tiks uzskatītas par atslēgu. Tad tiek izmantots XOR operators, lai sajauktu šī attēla rindas un kolonnas. Pēc visa šī var izvēlēties apturēt algoritmu vai veikt vēl iterācijas, kas algoritmu sāk no sākuma. Šī šifra atšifrēšanai izmanto XOR algoritmu rindu un kolonnu pareizajām pozīcijām un Rubika kuba principu, visiem Rubika kubiem saliekot 5 skaldnes, tādējādi atgriežoties pie sākotnējā attēla.

2.1. Rubika kuba šifrēšanas algoritms

Pieņemsim, ka I_0 ir attēls, kura izmērs ir $M \cdot N$ pikseļi. Šeit I_0 reprezentē pikseļu kopu jeb pikseļu vērtību matricu attēlam I_0. Šifrēšanas algoritma soļi ir šādi:

1) Ja M vai N nedalās ar 3 bez atlikuma, tad rindu apakšā un/vai kolonnu labajā pusē ievieto attiecīgi 1 vai 2 rindas un/vai kolonnas, tā, lai jaunais rindu un kolonnu skaits dalītos ar 3 bez atlikuma. Pikseļi šajās rindās un kolonnās drīkst būt tādās pašās krāsās kā tie, kas ir uz I_0, bet ne obligāti. Jauno attēlu apzīmē ar I_1 un jauno dimensiju skaitu pēc rindu un kolonnu ievietošanas ar M' un N';
2) Nodefinē algoritma iterāciju skaitu $ITER_{max}$, pirmo iterāciju sāk ar $ITER = 0$;
3) Nomaināms iterācijas skaitu: $ITER = ITER + 1$;
4) I_1 sadala attēlos ar izmēru 3×3 pikseļi. Šo 3×3 attēlu jeb matricu kopu apzīmēsim ar J;
5) Ap visiem attēliem kopā J nejauši ģenerē piecas 3×3 skaldnes tā, lai izveidotos Rubika kubs. Šīs piecas skaldnes katram attēlam kopā J drīkst gan atšķirties, gan būt vienādas. Svarīgi ir tas, ka uz nevienas no sešām skaldnēm neatrodas pikseļa krāsa, kas atrodas uz citas skaldnes vienā Rubika kubā. Uz vienas skaldnes nav obligāti jābūt vienai vienīgai krāsai. Šo Rubika kubu kopu apzīmēsim ar J_R;

6) Ar Rubika kubu no kopas J_R skaldnēm veic 20 gadījuma kustības, izveidojot jaunu kopu ar sajauktiem Rubika kubiem. Šo kopu apzīmēsim ar J_R^{-1};
7) Rubika kubus no kopas J_R^{-1} saliek tajās pašās pozīcijās, kur tie atradās attēlā I_1 ar skaldnēm tādā pašā pozīcijā, tādējādi izveidojot attēlu, ko apzīmēsim ar I_2;
8) Nejauši ģenerē vektorus K_R un K_C, kuru garums ir attiecīgi M' un N'. Vektors K_R sastāv no skaitļiem $\{0, 1, \dots, M' - 1\}$, bet vektors K_C sastāv no skaitļiem $\{0, 1, \dots, N' - 1\}$. Šo vektoru elementu vērtības nedrīkst būt visur vienādas;
9) Katrai fotogrāfijas I_2 rindai i:
   a) pikseļus pēc heksadecimālās sistēmas pārvērš skaitliskā vērtībā,
   b) aprēķina rindas i elementu summu, šo summu apzīmē ar $a(i)$:

$$a(i) = \sum_{j=1}^{N\prime} I_0(i,j), \quad i = 1, 2, \dots, M', \tag{4}$$

   c) nosaka summas $a(i)$ atlikumu dalījumā ar 2, šo apzīmē ar $M_{a(i)}$,
   d) rindu i pa labi vai pa kreisi pārvieto $K_R(i)$ pozīcijas un visus pikseļus, kas šķērso matricas robežas, pēc kārtas novieto pirmā pikseļa vietā. Šo pārvietošanu dara šādi:

$$\begin{cases} ja\ M_{a(i)} = 0, tad\ pārvieto\ pa\ labi, \\ ja\ M_{a(i)} = 1, tad\ pārvieto\ pa\ kreisi \end{cases} \tag{5}$$

10) Katrai fotogrāfijas I_2 kolonnai j:
   a) pikseļus pēc heksadecimālās sistēmas pārvērš skaitliskā vērtībā,
   b) aprēķina kolonnas j elementu summu, šo summu apzīmē ar $b(j)$:

$$b(j) = \sum_{i=1}^{M\prime} I_0(i,j), \quad j = 1, 2, \dots, N', \tag{6}$$

   c) nosaka summas $b(j)$ atlikumu dalījumā ar 2, šo apzīmē ar $M_{b(j)}$,
   d) kolonna j uz augšu vai uz leju tiek pārvietota par $K_C(i)$ pozīcijām. Šo pārvietošanu dara šādi:

$$\begin{cases} ja\ M_{b(j)} = 0, tad\ pārvieto\ uz\ augšu \\ ja\ M_{b(j)} = 1, tad\ pārvieto\ uz\ leju \end{cases} \tag{7}$$

11) Ja sasniegtā iterācija sakrīt ar izvēlēto $ITER_{max}$, tad šifrēšanas algoritms ir pabeigts, un aizšifrētā bilde tiek apzīmēta ar I_{SCR}. Ja nesakrīt, tad algoritms atgriežas pie 3. soļa.

Šī šifrēšanas algoritma atslēgu kopa ir 5 Rubika kuba skaldnes no kopas J_R, vektori K_R un K_C, skaitļi M un N un skaitlis $ITER_{max}$. Lai iegūtu ātru un efektīvu algoritmu, kas nepatērē daudz datora atmiņas, ieteicams izmantot $ITER_{max} = 1$. Turpretim, ja $ITER_{max} > 1$, tad veidojas daudz drošāks algoritms, ko grūtāk atšifrēt. Katrs šī algoritma izmantotājs mainot $ITER_{max}$ var izveidot savu līdzsvaru starp ātrumu un drošumu. Apakšnodaļas 2.3. piemērā un 3. nodaļā $ITER_{max}$ tika uzstādīts uz 1.

2.2. Rubika kuba atšifrēšanas algoritms

Atšifrēto fotogrāfiju I_0 iegūst no šifrētās fotogrāfijas I_{SCR} un atslēgām K_C, K_R, Rubika kubiem no kopas J_R^{-1} un $ITER_{max}$ šādi:

1) Sāk ar $ITER = 0$;
2) $ITER = ITER + 1$;
3) Katrai fotogrāfijas I_{SCR} kolonnai j:
   a) aprēķina kolonnas j elementu summu, šo summu apzīmē ar $b_{SCR}(j)$:

$$b_{SCR}(j) = \sum_{i=1}^{M'} I_{SCR}(i,j), \;\; j = 1, 2, \dots, N', \tag{8}$$

   b) nosaka summas $b_{SCR}(j)$ atlikumu dalījumā ar 2 un to apzīmē ar $M_{b_{SCR}(j)}$,
   c) kolonnu j uz leju vai uz augšu pārvieto par $K_C(i)$ pozīcijām. Šo pārvietošanu dara šādi:

$$\begin{cases} ja\ M_{b_{SCR}(j)} = 0, tad\ pārvieto\ uz\ leju, \\ ja\ M_{b_{SCR}(j)} = 1, tad\ pārvieto\ uz\ augšu \end{cases} \tag{9}$$

4) Katrai fotogrāfijas I_{SCR} rindai i:
   a) aprēķina rindas i elementu summu, šo summu apzīmē ar $a_{SCR}(i)$:

$$a_{SCR}(i) = \sum_{j=1}^{N'} I_{SCR}(i,j), \;\; i = 1, 2, \dots, M', \tag{10}$$

   b) nosaka summas $a_{SCR}(i)$ atlikumu dalījumā ar 2, šo apzīmē ar $M_{a_{SCR}(i)}$,
   c) rindu i pa labi vai pa kreisi pārvieto par $K_R(i)$ pozīcijām. Šo pārvietošanu dara šādi:

$$\begin{cases} ja\ M_{a_{SCR}(i)} = 0, tad\ pārvieto\ pa\ kreisi, \\ ja\ M_{a_{SCR}(i)} = 1, tad\ pārvieto\ pa\ labi \end{cases} \tag{11}$$

5) Šajā solī visi attēla pikseļi ir novietoti pareizajās pozīcijās pie saviem Rubika kubiem no kopas J_R^{-1}. Tālāk visu Rubika kubu no kopas J_R^{-1} 5 skaldnes, kas nepieder attēla galvenajai skaldnei, saliek šādi:

$$J_R^{-1}(k,l) \ggg J_R^A(k,l), \tag{12}$$

kur *k* un *l* apzīmē Rubika kubu pozīciju attēlā, kur $k \in \left\{1, \dots, \frac{M'}{3}\right\}$, $l \in \left\{1, \dots, \frac{N'}{3}\right\}$, ar J_R^A apzīmē Rubika kubus no kopas J_R bez galvenās skaldnes, bet ar operatoru $\ggg$ apzīmē operatora kreisajā pusē esošā Rubika kuba 5 skaldņu pārvēršanu par operatora labajā pusē esošā Rubika kuba 5 skaldnēm. Šādi ir iegūti Rubika kubi no kopas J_R. Ideālā gadījumā dators šo soli veiktu ar vismazāko iespējamo gājienu jeb Rubika kuba pagriezienu skaitu;

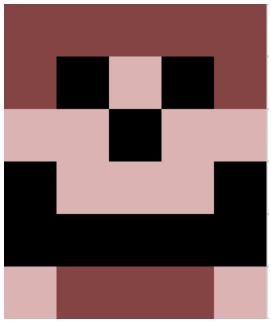
6) Iegūtos Rubika kubus no kopas J_R novieto ar pareizo skaldni uz augšu, lai iegūtu attēlu. Atšifrētājs zinās, kura ir pareizā skaldne, jo tā būs tā, kas nav daļa no atslēgas.

7) Ja ir $ITER = ITER_{max}$, tad algoritms ir beidzies un ir iegūts attēls I_1, pretēji atgriežas pie 2. soļa;

8) No attēla I_1 apakšas noņem $(M' - M)$ rindas un no labās puses noņem $(N' - N)$ kolonnas, lai atbrīvotos no rindām un kolonnām, kas tika pievienotas, lai rindu un kolonnu skaits dalītos ar 3 bez atlikuma, iegūstot sākotnējo attēlu I_0.

2.3. Algoritma vizualizācija

Šajā nodaļā apskatīsim vizualizācijas piemēru, kas palīdzēs izprast, kā īsti strādā algoritms. Kā pamattekstu izmantosim 6×5 attēlu, kas sevī ietver 3 atšķirīgu krāsu pikseļus:



2.1. att. Sākotnējais kriptoteksts I_0: "Cilvēka seja"

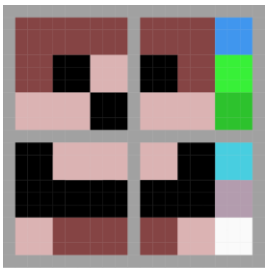
1) Šī attēla pikseļu rindu skaits dalās ar 3 bez atlikuma, bet kolonnu skaits gan nē. Tādēļ pievieno vienu kolonnu labajā pusē ar nejauši ģenerētām krāsām:

2.2. att. Attēls I_1

2) Nosaka $ITER_{max} = 1$; $ITER = 0$;

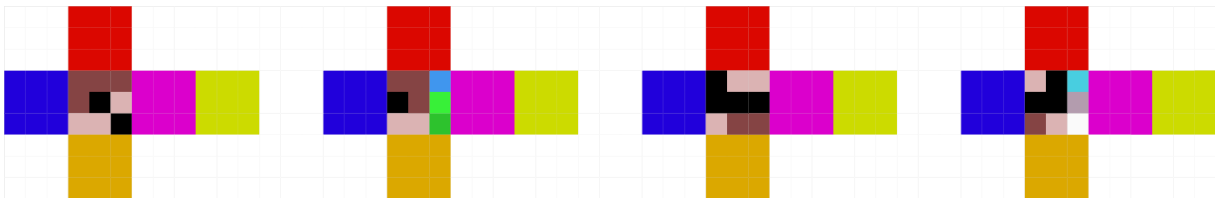
3) $ITER = ITER + 1 = 1$;

4) I_1 sadala attēlos ar izmēru 3×3 pikseļi:



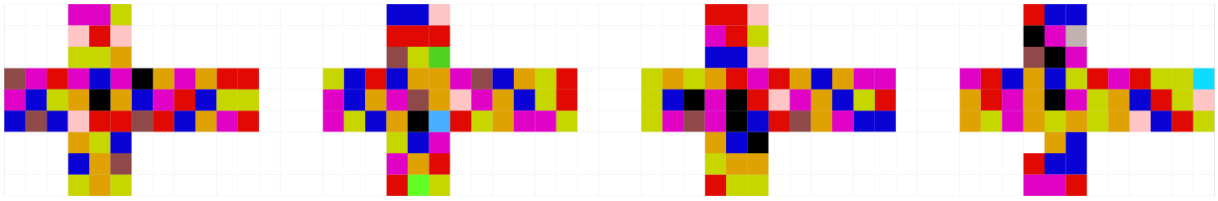
2.3. att. 3×3 attēli no kopas J

5) Nejauši ģenerē 5 skaldnes ap katru 3×3 attēlu no kopas J. Šajā piemērā vienkāršības dēļ uz skaldnēm atradīsies tikai viena krāsa, bet algoritma drošības dēļ šīm krāsām vajadzētu atšķirties. Tāpat arī vienkāršības dēļ šīs 5 skaldnes būs vienādas visiem Rubika kubiem:



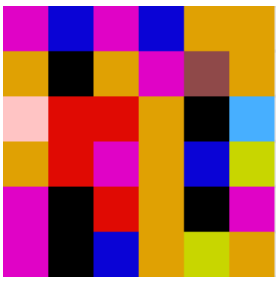
2.4. att. Rubika kubi no kopas J_R

6) Uz Rubika kubu no kopas J_R skaldnēm veic 20 gadījuma kustības, izveidojot kopu J_R^{-1}:



2.5. att. Rubika kubi no kopas J_R^{-1}

7) Rubika kubus no kopas J_R^{-1} skaldnes saliek tajās pašās pozīcijās, kurās tās bija attēlā I_1, izveidojot jaunu attēlu:



2.6. att. I_2

8) Nejauši ģenerē vektorus K_R un K_C, kuru garums ir 6 un sastāv no skaitļiem $\{0, 1, 2, 3, 4, 5\}$:

$$K_R = \{5, 2, 4, 2, 4, 1\}$$

$$K_C = \{3, 2, 5, 5, 1, 3\}$$

9) Katrai attēla I_2 rindai i:

a) Pikseļus pēc heksadecimālās sistēmas pārvērš skaitliskās vērtībās:

1. rinda: $\{14681030, 656342, 14681030, 656342, 14721283, 14721283\}$;
2. rinda: $\{14721283, 0, 14721283, 14681030, 9390409, 14721283\}$;
3. rinda: $\{16762052, 14682627, 14682627, 14721283, 0, 1698111\}$;
4. rinda: $\{14721283, 14682627, 14681030, 14721283, 656342, 13161984\}$;
5. rinda: $\{14681030, 0, 14682627, 14721283, 0, 14681030\}$;
6. rinda: $\{14681030, 0, 656345, 14721283, 13161984, 14721283\}$.

b) Aprēķina rindu elementu summu, to apzīmē ar $a(i)$:

$a(1) = 60057310$;

$a(2) = 68735288$;

$a(3) = 62546700$;

$a(4) = 72624549$;

$a(5) = 58765970$;

$a(6) = 57941925$.

c) Nosaka $a(i)$ atlikumu dalījumā ar 2, to apzīmē ar $M_{a(i)}$:

$M_{a(1)} = 0$;

$M_{a(2)} = 0$;

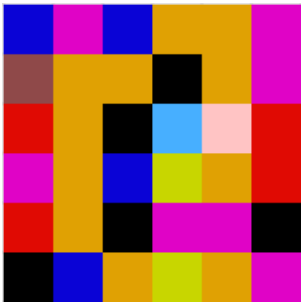
$M_{a(3)} = 0$;

$M_{a(4)} = 1$;

$M_{a(5)} = 0$;

$M_{a(6)} = 1$.

d) Rindu i pa labi vai pa kreisi pārvieto par $K_R(i)$ pozīcijām. Šajā gadījumā 1., 2., 3. un 5. rindu pārvietos pa labi, bet 4. un 6. rindu pārvietos pa kreisi:



2.7. att. Notikusi rindu pārvietošana

10) Katrai fotogrāfijas I_2 kolonnai j:

a) Pikseļus pēc heksadecimālās sistēmas pārvērš skaitliskās vērtībās:

1. kolonna: $\{14681030, 14721283, 16762052, 14721283, 14681030, 14681030\}$;
2. kolonna: $\{656342, 0, 14682627, 14682627, 0, 0\}$;
3. kolonna: $\{14681030, 14721283, 14682627, 14681030, 14682627, 656345\}$;
4. kolonna: $\{656342, 14682627, 14721283, 14721283, 14721283, 14721283\}$;
5. kolonna: $\{14721283, 9390409, 0, 656342, 0, 13161984\}$;
6. kolonna: $\{14721283, 14721283, 1698111, 13161984, 14681030, 14721283\}$.

b) Aprēķina kolonnu elementu summu, to apzīmē ar *b(j)*:

$b(1) = 90247708$;

$b(2) = 30021596$;

$b(3) = 74114942$;

$b(4) = 74224101$;

$b(5) = 37930018$;

$b(6) = 73704974$.

c) Nosaka *b(j)* atlikumu dalījumā ar 2, to apzīmē ar $M_{b(j)}$:

$M_{b(1)} = 0$;

$M_{b(2)} = 0$;

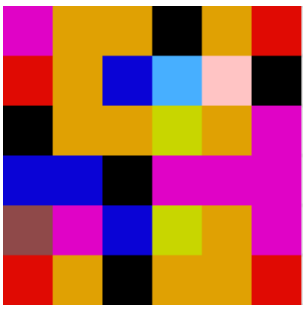
$M_{b(3)} = 0$;

$M_{b(4)} = 1$;

$M_{b(5)} = 0$;

$M_{b(6)} = 0$.

d) Kolonnu j uz augšu vai uz leju pārvieto par $K_C(i)$ pozīcijām. Šajā gadījumā 1., 2., 3., 5. un 6. kolonnu pārvietos uz augšu, bet 4. rindu pārvietos uz leju:



2.8. att. Notikusi kolonnu pārvietošana

11) Par cik izvēlētais iterāciju skaits $ITER_{max}$ ir 1, algoritms ir beidzies un ir iegūts šifrēts attēls I_{SCR}. Šo attēlu saņēmējam sūta kopā ar Rubika kubiem no attēla 2.5.

Tālāk tiks apskatīts atšifrēšanas algoritms. Tiks izmantots attēls 2.8., atslēgas $N = 6$ un $M = 5$, Rubika kubu 5 skaldnes no attēla 2.4. jeb kopa J_R^A un vektori K_R un K_C.

1) Sāk ar $ITER = 0$;

2) $ITER = ITER + 1 = 1$;

3) Katrai fotogrāfijas I_{SCR} kolonnai j:

a) aprēķina kolonnas j elementu summu, šo summu apzīmē ar $b_{SCR}(j)$:

$b_{SCR}(1) = 90247708$;

$b_{SCR}(2) = 30021596$;

$b_{SCR}(3) = 74114942$;

$b_{SCR}(4) = 74224101$;

$b_{SCR}(5) = 37930018$;

$b_{SCR}(6) = 73704974$.

b) nosaka summas $b_{SCR}(j)$ atlikumu dalījumā ar 2, šo apzīmē ar $M_{b_{SCR}(j)}$:

$M_{b_{SCR}(1)} = 0$;

$M_{b_{SCR}(2)} = 0$;

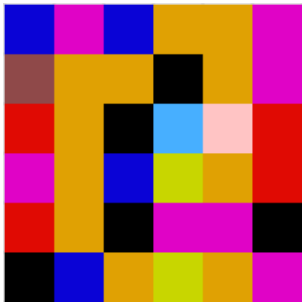
$M_{b_{SCR}(3)} = 0$;

$M_{b_{SCR}(4)} = 1$;

$M_{b_{SCR}(5)} = 0$;

$M_{b_{SCR}(6)} = 0$.

c) kolonnu j uz leju vai uz augšu pārvieto par $K_C(i)$ pozīcijām. Šajā gadījumā 1., 2., 3., 5. un 6. kolonnu pārvietos uz leju, bet 4. rindu pārvietos uz augšu.



2.9. att. Attēls pēc kolonnu pārvietošanas

4) Katrai fotogrāfijas I_{SCR} rindai i:

a) aprēķina rindas i elementu summu, šo summu apzīmē ar $a_{SCR}(i)$:

$a_{SCR}(1) = 60057310;$

$a_{SCR}(2) = 68735288;$

$a_{SCR}(3) = 62546700;$

$a_{SCR}(4) = 72624549;$

$a_{SCR}(5) = 58765970;$

$a_{SCR}(6) = 57941925.$

b) nosaka summas $a_{SCR}(i)$ atlikumu dalījumā ar 2, šo apzīmē ar $M_{a_{SCR}(i)}$:

$M_{a_{SCR}(1)} = 0;$

$M_{a_{SCR}(2)} = 0;$

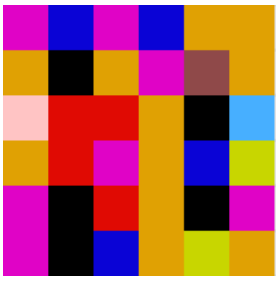
$M_{a_{SCR}(3)} = 0;$

$M_{a_{SCR}(4)} = 1;$

$M_{a_{SCR}(5)} = 0;$

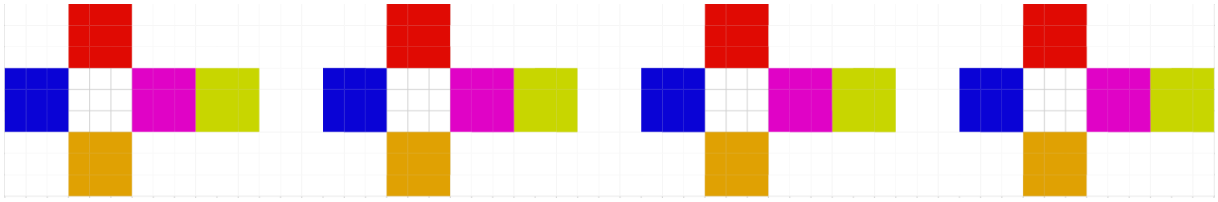
$M_{a_{SCR}(6)} = 1.$

c) Rindu i pa labi vai pa kreisi pārvieto par $K_R(i)$ pozīcijām. Šajā gadījumā 1., 2., 3. un 5. rindu pārvietos pa kreisi, bet 4. un 6. rindu pārvietos pa labi:

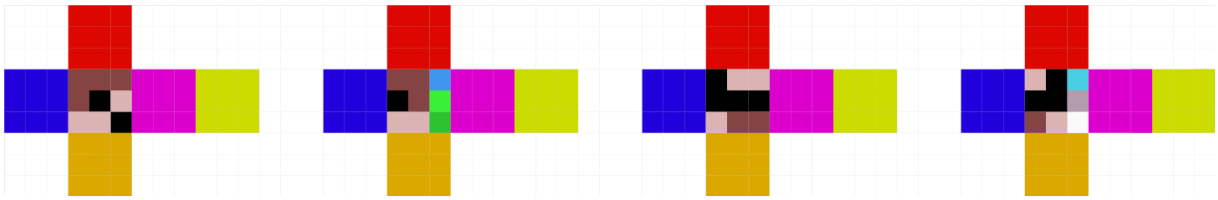


2.10. att. Attēls pēc rindu pārvietošanas

5) Izmanto Rubika kubus no attēla 2.5., lai piecas no to skaldnēm pārvērstu Rubika kubos, kas redzami atslēgā:

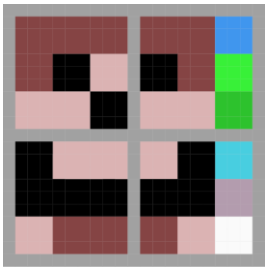


2.11. att. Rubika kubu piecas skaldnes jeb atslēgu kopa J_R^A



2.12. att. Rubika kubi no attēla 2.5. pēc operācijas $J_R^{-1}(k,l) \ggg J_R^A(k,l)$

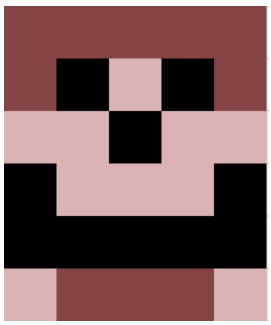
6) Iegūtos Rubika kubus novieto pareizā secībā ar pareizo skaldni uz augšu:



2.13. att.

7) $ITER = ITER_{max}$, tādēļ vairāk iterāciju nav jāveic;

8) No attēla 2.12. noņem kolonnu no labās puses, kas nebija daļa no oriģinālā attēla:



2.14. att.

Šajā brīdī algoritms ir beidzies, un ir veiksmīgi iegūts oriģinālais attēls.

3. ALGORITMA PRAKTISKĀ ANALĪZE

Šajā nodaļā tiks aprakstīti algoritma alternatīvi, tā ātrums, drošības līmenis, kā arī kopējais praktiskums.

3.1. Algoritma ātrums

Šifrēšanas algoritma ātrums ir ļoti svarīgs parametrs un kopā ar algoritma drošības līmeni tas nosaka algoritma kopējo praktiskumu. Šajā nodaļā apskatīsim, cik ilgā laikā jaudīgs dators spētu izpildīt šo algoritmu uz vairāku izmēru attēliem. Teiksim, ka tiktu izmantots viens no jaunākajiem procesoriem tirgū - Intel® Core™ i5-12600HX procesors ar maksimālo frekvenci 4.6 GHz jeb 4.6 miljards darbības sekundē ([4]), 16 GB RAM un 1 TB cietā diska atmiņas. Šifrēšanas algoritma 1.-3. soļi tiktu izpildīti uzreiz – aptuveni 4×10^{-9} sekundēs, tāpēc šo soļu izpildes ātrums vērā netiks ņemts gala aprēķinā. 5. soļa ātrums ir atkarīgs no tā, cik liels ir attēls – jo tas lielāks, jo vairāk Rubika kubu jāģenerē. Iepriekšminētais dators spētu nejauši ģenerēt 250 triljonus pikseļus sekundē ([5]). Tabulā 3.1. ir attēloti pikseļu ģenerēšanas ātrumi atkarībā no attēla lieluma:

Tabula 3.1.

Attēla izmērs (pikseļi)	Nepieciešamais pikseļu skaits	Ģenerēšanas laiks (sek.)
64 x 64	21780	$8{,}712 \times 10^{-8}$
128 x 128	83205	$3{,}328 \times 10^{-7}$
256 x 256	332820	$1{,}331 \times 10^{-6}$
512 x 512	1315845	$5{,}263 \times 10^{-6}$
1024 x 1024	5263380	$2{,}105 \times 10^{-5}$

Ja rēķina šifrēšanas algoritma ātrumu 7. solī, jāņem vērā, ka Rubika kubs no jebkuras pozīcijas var tikt atrisināts 20 kustībās vai mazāk ([7]), tāpēc nav jēgas to jaucot izpildīt vairāk nekā 20 kustību. Šajā algoritmā dators katram Rubika kubam ģenerē 20 nejaušas kustības, kas sajauc Rubika kubu. Šo darbību ātrums, protams, atkal ir atkarīgs no attēla lieluma. Šis izpildes ātrums ir attēlots tabulā 3.2.:

Tabula 3.2.

Attēla izmērs (pikseļi)	Kustību skaits	Ģenerēšanas laiks (sek.)
64 x 64	9680	$3{,}872 \times 10^{-8}$
128 x 128	36980	$1{,}479 \times 10^{-7}$
256 x 256	147920	$5{,}917 \times 10^{-7}$
512 x 512	584820	$2{,}339 \times 10^{-6}$
1024 x 1024	2339280	$9{,}357 \times 10^{-6}$

Ir arī vērts ņemt vērā 8.-10. soļu ātrumus. 8. solī tiek veikta viena darbība uz katru rindu un kolonnu, bet 9. un 10. solī uz katru rindu un kolonnu tiek veiktas 3 darbības. To kopējais ātrums ir attēlots tabulā 3.3:

Tabula 3.3.

Attēla izmērs (pikseļi)	Darbību skaits	Ātrums (sek.)
64 x 64	924	$3{,}696 \times 10^{-9}$
128 x 128	1806	$7{,}224 \times 10^{-9}$
256 x 256	3612	$1{,}445 \times 10^{-8}$
512 x 512	7196	$2{,}878 \times 10^{-8}$
1024 x 1024	14364	$5{,}746 \times 10^{-8}$

Lai iegūtu kopējo algoritma šifrēšanas daļas ātrumu, tiek saskaitīti visi soļi no tabulām 3.1.-3.3.:

Tabula 3.4. Kopējais šifrēšanas ātrums

Attēla izmērs (pikseļi)	Kopējais darbību skaits	Ātrums (sek.)
64 x 64	32384	$1{,}29536 \times 10^{-7}$
128 x 128	121991	$4{,}87964 \times 10^{-7}$
256 x 256	484352	$1{,}93741 \times 10^{-6}$
512 x 512	1907861	$7{,}63144 \times 10^{-6}$
1024 x 1024	7617021	$3{,}04681 \times 10^{-6}$

Atšifrēšanas algoritma 3. un 4. solis ir līdzīgi šifrēšanas algoritma 9. un 10. solim, tāpēc to izpildes laiku var nolasīt tabulā 3.3.

Atšifrēšanas algoritma 5. solis nenoliedzami patērēs visvairāk laika no visiem soļiem. Visātrākie datora algoritmi spēj Rubika kubu salikt sākuma pozīcijā 0,38 sekundēs ([6]). Šis laika daudzums tiek izmantots, lai aprēķinātu 5. soļa ātrumu tabulā 3.5.:

Tabula 3.5.

Attēla izmērs (pikseļi)	Rubika kubu skaits	Patērētais laiks (sek.)
64 x 64	484	183,92
128 x 128	1849	702,62
256 x 256	7396	2810,48
512 x 512	29241	11111,58
1024 x 1024	116964	44446,32

Kā redzams, 5. soļa izpilde prasa daudz laika – aptuveni 12 stundas tiek atvēlētas attēlam izmērā 1024×1024 pikseļi. Šim solim nepieciešamais laika apjoms padara visu iepriekšējo soļu izpildes ātrumu par vērā neņemamu. No šiem rezultātiem var secināt, ka, izmantojot algoritmu, prioritātei vajadzētu būt nevis šifra ātrumam, bet drošībai.

3.2. Algoritma drošība

Šifrētajam attēlam vajadzētu maksimāli atšķirties no oriģinālā. Parasti šo atšķirību var kvantificēt ar diviem rādītājiem ([1]). Pirmais no šiem rādītājiem ir izmainīto pikseļu skaits jeb *number of pixels change rate* (*NPCR*), kas procentuāli parāda, cik pikseļi izmainās no oriģinālās fotogrāfijas. Otrs rādītājs ir apvienotā vidējā izmaiņu intensitāte jeb *unified average changing intensity* (*UACI*), kas mēra vidējo pikseļu atšķirības intensitāti. Pieņemsim, ka $I_1(i,j)$ un $I_{SCR}(i,j)$ ir pikseļu vērtību matricas respektīvi oriģinālajai pēc rindu un kolonnu pievienošanas un šifrētajai fotogrāfijai, kur i – matricas rindas, bet j – matricas kolonnas. Ar vienādojumiem (13) un (15) definēsim *NPCR* un *UACI* rādītājus:

$$NPCR = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N} D(i,j)}{M \cdot N} \cdot 100\%, \tag{13}$$

$$kur\ D(i,j) = \begin{cases} 0, & ja\ I_0(i,j) = I_{SCR}(i,j), \\ 1, & ja\ citādi \end{cases} \tag{14}$$

$$UACI = \left(\sum_{i=1}^{M}\sum_{j=1}^{N} \frac{|I_0(i,j) - I_{SCR}(i,j)|}{16777216} \right) \cdot \frac{100\%}{M \cdot N} \tag{15}$$

Darba [1] autori ir noteikuši, ka, lai sasniegtu ideālu fotogrāfiju šifrēšanas algoritmu, *NPCR* vērtībai jābūt pēc iespējas augstākai un *UACI* vērtībai jābūt aptuveni 33%. Taču darbā [1] tika izmantotas melnbaltas bildes, kuru *UACI* vērtība ir svarīgāka nekā krāsainiem attēliem – 33% atšķirība melnbaltā attēlā ir daudz mazāka nekā krāsainā. Tāpēc šim algoritmam

pieņemamas *UACI* vērtības būtu jebkur starp 10% un 90%. Tabula 3.6. parāda *NPCR* un *UACI* attēliem dažādos izmēros. Ir redzams, ka *NPCR* vērtības ir ļoti tuvas 100% un *UACI* vērtības arī ir adekvātas. Iemesls, kāpēc *UACI* vērtības ir neatkarīgas no attēla izmēra ir tāds, ka neatkarīgi no tā, kā sajauc Rubika kubu, katras skaldnes centrs vienmēr paliks tajā pašā krāsā, tāpēc katrā attēlā mainīsies $^{8}/_{9}$ no pikseļu krāsām. Augstā *NPCR* vērtība norāda uz to, ka pikseļu pozīciju maiņa ir bijusi gadījuma lielums.

Tabula 3.6.

Attēla izmērs (pikseļi)	NPCR (%)	UACI (%)
64 x 64	99,9254	88,8889
128 x 128	99,9593	88,8889
256 x 256	99,9846	88,8889
512 x 512	99,9990	88,8889
1024 x 1024	99,9998	88,8889

3.3. Algoritma stiprās un vājās puses, kā arī alternatīvas

Apakšnodaļās 3.1. un 3.2. ir noskaidrots, ka algoritms ir pietiekami drošs, bet tā atšifrēšanas daļa ir ļoti lēna, kas arī ir tā lielākā vājība. No šī varētu secināt, ka šis šifrēšanas algoritms šajā formā var būt pielietots tikai un vienīgi tad, kad algoritma izpildes laiks nav svarīgs, bet šifra drošība ir nepieciešama būt augstā līmenī. Ir pāris veidu, kā algoritmu izmainīt tā, ka tā izpilde prasa mazāku laika apjomu, iemainot to pret mazāku drošības līmeni:

1) Šifrēšanas algoritma 5. solī izveidot Rubika kubu 5 skaldnes ap kopas J_R attēliem tā, lai tās visiem attēliem kopā J būtu vienādas. Šis tikai samazinātu nepieciešamo pikseļu skaitu, kas datoram jāģenerē, bet atšifrēšanas algoritma 5. soļa izpildes laiks paliktu relatīvi neskarts, jo datoram tik un tā būtu jāatrod veids, kā katru Rubika kubu salikt, jo tie ir sajaukti neatkarīgā veidā viens no otra;
2) Šifrēšanas algoritma 6. solī katru Rubika kubu no kopas J_R sajaukt ar vienu un to pašu kustību secību, izveidojot vienādi sajauktus Rubika kubus. Šis ne tikai paātrinātu datoru solī, kur tas ģenerē nejaušas kustībās, lai sajauktu Rubika kubus, bet arī ļautu ietaupīt pat vairākas stundas lielākiem attēliem Rubika kubu salikšanas solī, jo dators, saliekot pirmo Rubika kubu, uzreiz zina, kā salikt visus pārējos, būtībā padarot katra nākamā Rubika kuba salikšanu par vienu darbību. Šai algoritma alternatīvai ir vislielākais potenciāls izmantošanai praktiskā vidē. Šīs alternatīvas vājībā ir tāda, ka, uzlaužot vienu Rubika kubu, tiek uzlauzts arī viss šifrs.

Ir arī vismaz viens veids, kā padarīt algoritmu drošāku, izmantojot XOR operatoru. Šis gan, visticamāk, algoritmu padarītu lēnāku. Šis solis tika pielietots darbā no avota [1] un tas ir sekojošs:

Izmantojot vektoru K_R, XOR operatoru pielieto katrai attēla I_{SCR} rindai:

$$I_1(2i-1,j) = I_{SCR}(2i-1,j) \oplus K_R(i) \tag{16}$$

$$I_1(2i,j) = I_{SCR}(2i,j) \oplus rot180\big(K_R(i)\big), \tag{17}$$

kur $\oplus$ un $rot180(K_R(i))$ respektīvi apzīmē XOR operatoru un vektora K_R apgriešanu otrādi, pēdējam elementam kļūstot par pirmo un pirmajam par pēdējo. Indekss $2i-1$ apzīmē nepāra rindu, bet $2i$ – pāra.

Līdzīgi tiek darīts ar attēla kolonnām un atslēgu K_C. Šis pārvērstu pikseļu un atslēgu vērtības bināros skaitļos un mainītu pikseļu krāsu, kas nozīmētu, ka būtu gandrīz neiespējami noteikt, kuriem no šifra pikseļiem vajadzētu atrasties uz oriģinālā attēla, un kuri ir daļa no ģenerētajām Rubika kubu skaldnēm. Šis solis būtu viegli izmantojams atšifrēšanā, jo, kā jau iepriekš minēts, XOR operators ir atgriežama funkcija.

Kombinācijā ar alternatīvu 2), visticamāk, algoritms šādā formā būtu vislabākais no lietojamības viedokļa, tāpēc šo alternatīvo algoritmu apskatīsim padziļinātāk.

3.3.1. Algoritma alternatīvas apraksts

Ja runā par šifrēšanas algoritmu, tad 6. solis ir alternatīvajā versijā jāizmaina:

"6) Ar Rubika kubu no kopas J_R skaldnēm veic 20 gadījuma kustības, **kas visiem Rubika kubiem ir vienādas**, izveidojot jaunu kopu ar sajauktiem Rubika kubiem. Šo kopu apzīmēsim ar J_R^{-1}."

Pēc 10. soļa tiek ieviesti divi jauni soļi, kas izmaina attēla pikseļu krāsas:

11) Izmantojot vektoru K_R, XOR operatoru pielieto katrai attēla I_{SCR} rindai:

$$I_1(2i-1,j) = I_{SCR}(2i-1,j) \oplus K_R(i) \tag{18}$$

$$I_1(2i,j) = I_{SCR}(2i,j) \oplus rot180\big(K_R(i)\big), \tag{19}$$

12) Izmantojot vektoru K_C, XOR operatoru pielietojam katrai attēla I_{ENC} kolonnai:

$$I_{ENC}(i,2j-1) = I_1(2i-1,j) \oplus K_C(i) \tag{20}$$

$$I_{ENC}(i,2j) = I_1(2i,j) \oplus rot180\big(K_C(i)\big), \tag{21}$$

Rezultātā oriģinālā algoritma 11. solis paliktu par šī algoritma 13. soli.

Ja runā par šīs alternatīvas atšifrēšanas algoritmu, tad pēc 2. soļa tiktu ieviesti divi soļi, kas pikseļus atgrieztu atpakaļ pareizajās krāsās:

3) Izmantojot vektoru K_C, XOR operatoru pielieto katrai attēla I_{ENC} kolonnai šādi:

$$I_1(i, 2j-1) = I_{ENC}(2i-1, j) \oplus K_C(i) \quad (22)$$

$$I_1(i, 2j) = I_{ENC}(2i, j) \oplus rot180\big(K_C(i)\big), \quad (23)$$

4) Izmantojot vektoru K_R, XOR operatoru pielieto katrai attēla I_1 rindai šādi:

$$I_{SCR}(2i-1, j) = I_1(2i-1, j) \oplus K_R(i) \quad (24)$$

$$I_{SCR}(2i, j) = I_1(2i, j) \oplus rot180\big(K_R(i)\big) \quad (25)$$

Katra soļa indekss pēc šiem alternatīvajiem soļiem tiktu pavirzīts par 2 indeksiem uz priekšu.

Ja oriģinālajā algoritmā ievieš šos soļus, tad algoritms paliek daudz ātrāks, jo datoram, kas atrisina Rubika kubus, principā ir tikai jāatrod atrisinājums vienam Rubika kubam. Tālāk apskatīsim šīs alternatīvā algoritma ātrumu. Šeit ievērojami mainīsies laika apjomi, kas redzami tabulā 3.5.:

Tabula 3.7.

Attēla izmērs (pikseļi)	Rubika kubu skaits	Patērētais laiks (sek.)
64 x 64	1	0,38
128 x 128	1	0,38
256 x 256	1	0,38
512 x 512	1	0,38
1024 x 1024	1	0,38

Visus pārējos laika apjomus, kas tiek patērēti uz gadījuma kustību ģenerēšanu un XOR operatoru, var neņemt vērā, jo tie tiktu mērīti nanosekundēs. Alternatīvais algoritms būtu praktiski lietojams, jo tas šifrētu un aizšifrētu attēlu ļoti ātri, pat ātrāk nekā šifrēšanas algoritms, kas minēts avotā [1]. Vēl vairāk, šis alternatīvais algoritms spētu šifrēt un atšifrēt attēlu gandrīz vienādā ātrumā neatkarīgi no tā izmēra.

SECINĀJUMI

Šajā darbā tika aprakstīts fotogrāfiju šifrēšanas algoritms. Šis algoritms ir balstīts uz Rubika kuba principa, lai pārvietotu un sajauktu fotogrāfijas pikseļus. Lai aizšifrētu fotogrāfiju vēl vairāk, tiek izmantots XOR operators ar nejauši uzģenerētu atslēgu. Šīs XOR funkcijas atslēga tiek pielietota uz attēla rindām un kolonnām, lai tās sajauktu vēl vairāk. Praktiskā analīze ir noteikusi, ka algoritms ir salīdzinoši drošs, bet lēns, kas neļauj algoritmu pielietot praktiski. Taču ir noteikts alternatīvs algoritms, kas to padara ievērojami ātrāku un, līdz ar to, praktiski lietojamu. Nenoliedzami, šīs alternatīvas stiprā puse ir tā, ka to ir iespējams izpildīt salīdzinoši vienādā laikā, neatkarīgi no sākotnējā attēla izmēra. Alternatīvā algoritma izpildes ātrums to padara efektīvāku no ātruma viedokļa par algoritmu, kas minēts avotā [1], taču, lai pārliecinātos par kopējo efektivitāti, šis alternatīvais algoritms būtu jāizstrādā un jāveic drošības statistiskā analīze.

Šī darba mērķis bija izveidot un analizēt oriģinālu attēlu šifrēšanas algoritmu, tā drošību, efektivitāti un alternatīvas. Nākamais solis algoritma analīzē varētu būt nodaļā 3.3.1. noteiktā alternatīvā algoritma analīze, lai secinātu, vai šis algoritms varētu būt praktiski pielietojami, un, ja ir, tad šī algoritma izstrāde un padziļināta statistikā drošības analīze.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] Khaled Loukhaoukha u.c. *A Secure Image Encryption Algorithm Based on Rubik's Cube Principle* 2012. Pieejams: https://www.hindawi.com/journals/jece/2012/173931/ skatīts 01.05.2021.

[2] Douglas R. Stinson. (1995) *Cryptography: theory and practice.* CRC Press, - 434 lpp.

[3] IEEE, *Encryption of images using XOR cipher* 2017. Pieejams: https://ieeexplore.ieee.org/abstract/document/7919607 skatīts 02.05.2021.

[4] *12th Generation Intel® Core™ i5 Processors* Pieejams: https://www.intel.com/content/www/us/en/products/details/processors/core/i5.html Skatīts 11.05.2022.

[5] Kyungduk Kim *Massively parallel unltrafast random bit generation with a chip-scale laser* 2021. Pieejams: https://www.science.org/doi/abs/10.1126/science.abc2666 Skatīts 29.05.2022.

[6] Ben Katz *The Rubik's Contraption* 2018. Pieejams: https://build-its-inprogress.blogspot.com/2018/03/the-rubiks-contraption.html Skatīts 29.05.2022.

[7] Rik van Grol *The Quest for God's Number* Pieejams: https://www.tandfonline.com/doi/abs/10.4169/194762110X535961?journalCode=umho20 Skatīts 29.05.2022.

Bakalaura darbs “Drošs attēlu šifrēšanas algoritms, izmantojot *XOR* operatoru un Rubika kubu” izstrādāts LU Fizikas, matemātikas un optometrijas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Jānis Rekners

_____ _____

(paraksts) (datums)

Rekomendēju darbu aizstāvēšanai:

Vadītājs: Dr. Math. Raivis Bēts

_____ _____

(paraksts) (datums)

Darbs iesniegts Matemātikas nodaļā 2022. gada __. jūnijā.

_____

(Dekāna pilnvarotā persona Inita Šneidere)

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

2022. gada__ . jūnijā.