AZƏRBAYCAN RESPUBLİKASI ELM VƏ TƏHSİL NAZİRLİYİ

AZƏRBAYCAN TEXNİKİ UNİVERSİTETİ

"Kibertəhlükəsizlik" kafedrası

Əlyazması hüququnda

Məmmədova Nərmin Ləyaqət

Xələfova Aytac Telman

Abbasova Kəmalə Vüqar

İxtisas: 060632 – "İnformasiya texnologiyaları və sistemləri mühəndisliyi"
İxtisaslaşma: "İnformasiya mühafizəsi və təhlükəsizliyi"

Mövzu: Kriptoqrafik tədqiqatlar üçün xüsusi proqram təminatının hazırlanması

MAGİSTRLİK DİSSERTASİYASI

Elmi rəhbər: **t.f.d, dosent M.Y.Orucova**

Bakı-2023

MÜNDƏRİCAT

Giriş

Mövzunun aktuallığı: Elmi-texniki inqilab müasir kompüter texnologiyaları, rabitə vasitələri, eləcə də informasiyanın avtomatlaşdırılmış emalının müasir üsulları əsasında cəmiyyətin informasiyalaşdırılması sahəsində son vaxtlar böyük miqyas almışdır. Bu alət və metodların istifadəsi universal xarakter almışdır və eyni zamanda yaradılmış informasiya və hesablama sistemləri və şəbəkələri həm ərazi bölgüsü baxımından, həm də proseslərin vahid texnologiyaları çərçivəsində qlobal bir şəbəkə ilə bütün dünyada əhatə dairəsini genişləndirir. Məlumatların toplanması, ötürülməsi, saxlanması, axtarışı, işlənməsi və istifadə üçün verilməsi bu sahədə əldə edilən son nailiyyətlərə əsaslanır.Cəmiyyətdə dolaşan informasiyanın həcmi durmadan artır.Informasiya texnologiyalarının təhlükəsizliyinin təmin edilməsi probleminin aktual olması və əhəmiyyəti aşağıdakı səbəblərlə bağlıdır.

1. Müasir kompüterlərin işini sadələşdirərkən onların hesablama gücünün kəskin artması: Kompüterlər daha güclü və mürəkkəb hala gəldikcə, şifrələmə alqoritmləri və digər təhlükəsizlik tədbirləri daha güclü olmalıdır. Artan hesablama gücü, şifrələmənin kırılmasını asanlaşdırır və təhlükəsizliyin qorunmasını daha çətinləşdirir.

2. Məlumatların həcmindəki kəskin artım: İnformasiya daxili və xarici tərəfən toplanan, saxlanılan və emal edilən məlumatların həcmi artmaqdadır. Bu, məlumatların qorunmasını daha çətin və kompleks hələ gətirir.

3. Məlumatların birbaşa və vahid məlumat bazalarında cəmləşdirilməsi: Fərqli məqsədlər üçün toplanan məlumatların birbaşa və vahid bir məlumat bazasında cəmləşdirilməsi, bu bazanın təhlükəsizliyini və məlumatların müxtəlif mənsubiyyətlər tərəfindən istifadəsini təmin etməyi zəruriləşdirir.

4. Fəaliyyət göstərən kompüter parkının genişlənməsi: Fərqli sahələrdə fəaliyyət göstərən kompüter parkının həcmi artmaqdadır. Bu, daha çox kompüterin və şəbəkə cihazlarının təhlükəsizliyinin qorunmasını tələb edir.

5. Minimum təhlükəsizlik tələblərinə cavab verməyən proqram təminatının sürətli inkişafı: Proqram təminatının sürətli inkişafı, onun təhlükəsizliyinə dair tələblərə cavab verə bilməməyə səbəb olur. Bu, təhlükəsizlik zəafiyətlərinin daha çox mövcud olmasına və məlumatların pozulmasına səbəb ola bilər.

6. Qlobal İnternet şəbəkəsinin inkişafı və informasiya emalı sistemlərinin pozulmasının qarşısının alınmaması: Qlobal İnternet şəbəkəsinin inkişafı ilə birlikdə, informasiya emalı sistemlərinin pozulması və məlumatların hərəkətləri üzərindəki kontrolsüzlük artmaqdadır.

Bu səbəblərə görə, informasiya texnologiyalarının təhlükəsizliyi, informasiyanın icazəsi olmayan şəxslərin icazəsiz daxil olmasından qorunmasının zəruriliyini vurğulayır. İnformasiya dəyərli və həssasdır və bu cür məlumatları saxlayan, emal edən və ötürən kompüter sistemlərinə qarşı zərərli hərəkətlər mümkündür. Bu hərəkətlər "informasiya hücumu"kimi adlandırıla bilər.

Tədqiqatın məqsədi: Modifikasiya olunmuş RC6 məlumat şifrələmə alqoritminin işlənməsi, tədqiqi və praktiki tətbiqi.

Tədqiqatın obyekti: Bu disertasiyanın obyekti, kriptoqrafik tədqiqatlar üçün xüsusi proqram təminatının hazırlanmasıdır. Bu proqram təminatı təhlükəsiz və effektiv şifrələmə alqoritmlərini tətbiq etmək və test etmək üçün çərçivə və alətlər yaratmaq məqsədi daşıyır. Məqsəd RC6 alqoritminin performansını və effektivliyini artırmaq üçün dəyişikliklər və yenilikləri həyata keçirməkdir. Bu baxış təhlükəsizlik, performans, açarların idarə edilməsi və digər əsas sahələrdə şifrələmə alqoritminin performansını araşdırmaq məqsədi daşıyır.

Tədqiqat işinin nəzəri və praktiki əhəmiyyəti:

1. Şifrələmə alqoritmlərinin dizaynı və inkişafının əsas nəzəri müddəalarını araşdırılması.

2. Modifikasiya olunmuş RC6 məlumat şifrələmə alqoritminin kriptoqrafik gücünün effektivliyinin təhlili və araşdırılması

Dissertasiya işinin həcmi və quruluşu: Dissertasiya giriş, 3 fəsil , nəticələr və istifadə olunmuş ədəbiyyat siyahısından ibarətdir . Tədqiqat işinin giriş hissəsi seçilmiş mövzunun aktuallığını, tədqiqatın məqsədi, eyni zamanda işin praktiki və nəzəri əhəmiyyətini ortaya qoyur.

Birinci fəsildə əsas nəzəri müddəalar və müasir şifrələmə alqoritmləri təhlil edilir. İkinci fəsildə standart və dəyişdirilmiş RC6 alqoritmlərinin şifrələməsi, şifrəsinin açılması və açarların yaradılması üçün əsas prosedurlar müzakirə olunur.

Üçüncü fəsildə modifikasiya olunmuş RC6-nın kriptoqrafik gücünə dair şifrləmə, deşifrə və onun praktiki tətbiqində açarların yaradılması ilə bağlı tədqiqatlar var.

Abbasova Kəmalə Vüqar qızı: Kriptoqrafik alqoritmlər onların təsnifatını, müqayisəsini araşdırmış və təhlil etmişdir.

Xələfova Aytac Telman qızı: RC6 şifrələnməsi haqqında araşdırma aparmış necə inteqrasiya etmək olar istifadəsi haqqında ətraflı məlumat toplamışdır.

Məmmədova Nərmin Ləyaqət qızı: RC6 şifrələnməsini C# dilinə inteqrasiya etmiş. Standart və Modifikasiya olunmuş RC6 alqoritmlərin şifrələnməsi və deşifrə edilməsi, açarların yaradılması ilə bağlı C# dilində kodları işləmişdi. Və bu kodlardan istifadə edərək FileCoder Şifrələmə Proqramının blok sxemlərini C# proqramlaşdırma dilində yazmışdır.

İXTİSARLARIN SİYAHISI

AES	Advanced Encryption Standard - Qabaqcıl Şifrələmə Standartı
3DES	Triple Data Encryption Standard - Üçlü Məlumat Şifrələmə Alqoritmi
RSA	Rivest–Shamir–Adleman - Rivest–Şamir–Adleman
ECC	Elliptik Curve Cryptography - Elliptik əyri kriptoqrafiya
SHA-256	Secure Hash Alqoritmi 256-bit - Təhlükəsiz Hash Alqoritmi
MD5	Message Digest Alqoritmi 5 - Mesaj Digest 5
DES	Data Encryption Standard - Məlumat Şifrələmə Standartı
GOST 28147-89	GOST block cipher -GOST blok şifrəsi
ECB	Electronic Code Book - Elektron Kod Kitabı
CBC	Cipher Block Chaining - Blockchain Şifrələmə rejimi
OFB	Output Feedback - Çıxış Əlaqəsi rejimi
CFB	Cipher Feedback - Şifrələmə Qeydiyyatı
FEAL	Fast Data Encipherment Alqoritmi - Sürətli Məlumat Şifrələmə Alqoritmi
IDEA	Data Encryption Algorithm- Beynəlxalq Məlumat Şifrələmə Alqoritmi
NIST	National Institute of Standards and Technology -Milli Standartlar və Texnologiya İnstitutu
DSA	Digital Signature Algorithm - Rəqəmsal İmza Alqoritmi
CHAP	Password Authentication Protocol - Şifrə Doğrulama Protokolu

MAC	Message Authentication Code – Mesajın Autentifikasiya Kodu
DTS	Electronic Timestamp Service - Elektron vaxt ştampı xidməti

I FƏSİL Şifrələmə alqoritmlərinin təhlili

1.1Kriptoqrafik alqoritmlərin təsnifatı

Kriptoqrafiya - (Kripto məxfilik, qrafik yazı) icazəsiz girişin qarşısını almaq üçün məlumatların şifrələnməsi və şifrəsinin açılmasının öyrənilməsidir. Şifrələmə prosesindən istifadə etməklə şifrələnmiş mətn həm göndərən, həm də qəbul edən ünvan tərəfindən müəyyən edilməlidir. Müasir məlumat təhlükəsizliyinin inkişafı ilə biz indi məlumatlarımızı elə dəyişə bilərik ki, yalnız nəzərdə tutulan alıcı onu anlaya bilsin[16].

Şifrələmə - kriptoqrafiyanın əsas komponentidir. Məlumatların şifrələmə və deşifrələmə açarları istifadə edərək oxuna bilinməz bir formaya çevrilməsidir. Şifrələmə məlumatı arzuolunmaz şəxslərə gizli saxlamaqla məxfiliyi təmin edir. Məlumat şifrələndikdən sonra şifrələnmiş mətnin yenidən oxunması üçün şifrənin açılması prosesi başlayır. Kriptoqrafiya həm şifrələmə, həm də deşifrə proseslərinə əsaslanır.

Bəzi müasir şifrələmə alqoritmləri:

1. Simmetrik Şifrələmə: Bu klassik yanaşmada eyni açar şifrələmə və şifrənin açılması üçün istifadə olunur. Simmetrik şifrələmə alqoritmlərinə misal olaraq AES və 3DES daxildir. Onlar əsas ölçüsü və performansı ilə fərqlənirlər, lakin hər ikisi yüksək səviyyədə məxfilik təmin edir.

2. Asimmetrik Şifrələmə: Bu tip şifrələmə bir cüt açardan istifadə edir - açıq açar və şəxsi açar. Açıq açar məlumatı şifrələmək üçün istifadə olunur, şəxsi açar isə məlumatların şifrəsini açmaq üçün istifadə olunur. RSA və ECC məşhur asimmetrik şifrələmə alqoritmləridir. Onlar təhlükəsiz açar mübadiləsi və rəqəmsal imza təmin edir.

3. Hashing: Hash funksiyaları verilənləri dəyişən uzunluqlu sətirə çevirən funksiyalardır. Hash funksiyaları məlumatların bütövlüyünü yoxlamaq və rəqəmsal barmaq izlərini yaratmaq üçün geniş istifadə olunur. SHA-256 və MD5 hash funksiyalarının nümunələri mövcuddur.

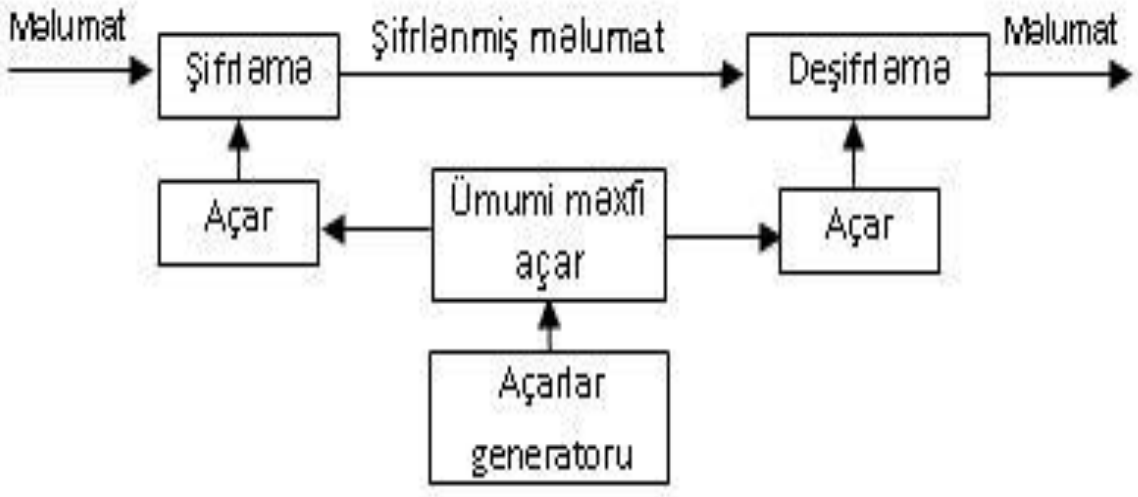
4. Açar Mübadilə Protokolları: Bu protokollar rabitə iştirakçıları arasında şəxsi açarların təhlükəsiz mübadiləsinə imkan verir. Məsələn, Diffie-Hellman protokolu faktiki köçürmə olmadan açarın təhlükəsiz birgə təyin edilməsinə imkan verir.

5. Kvant Şifrələmə: Bu, məlumat ötürülməsini təmin etmək üçün kvant mexanikasının prinsiplərindən istifadə edən yeni sahəyə aiddir. BB84 alqoritmi kimi kvant alqoritmləri kvant hissəciklərinin xüsusiyyətlərindən istifadə edir.

Bütün müasir kriptoqrafik alqoritmlər şifrələmə və şifrənin açılması prosesində mənbə mətnlə yanaşı, yalnız göndərən və qəbul edən tərəfə məxsus məxfi element olan açardan istifadə edən şifrələmə sxemi ilə işləyir. Ümumiyyətlə, bütün kriptoqrafik sistemlər simmetrik və asimmetrik olaraq iki hissəyə bölünür.

Simmetrik Açar Kriptoqrafiyası

Bu sistemlərdə şifrələmə və açma prosesləri eyni açarı istifadə edir. Açarın məxfiliyi əsas prinsipdir. Simmetrik kriptoqrafiya sistemlərində əsas problem, məxfi açarın hər iki tərəfə təhlükəsiz şəkildə ötürülməsindən ibarətdir. Bununla birlikdə, bu sistemlər yüksək sürətə malikdirlər. Açarın qarşı tərəf tərəfindən açıqça yayılması, yalnız həmin açarda şifrələnmiş məlumatın ifşa olunmasına gətirib çıxara bilər. DES, AES və GOST 28147-89 - bu şifrələnmə növlərini simmetrik kriptoqrafik sistemlərə misal göstərmək olar[19].



Şək. 1.1 Simmetrik şifrələmə sistemi

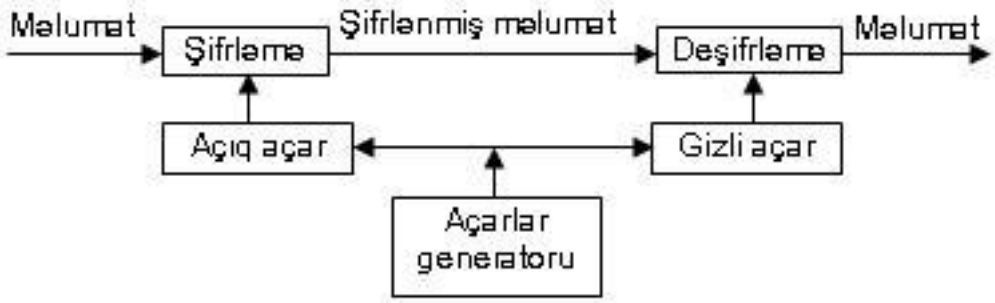
Ən məşhur simmetrik alqoritmlər Cədvəl 1.1-də göstərilmişdir.

Cədvəl 1.1

Növ	**Təsvir**
DES	ABŞ-da məhkəmələr tərəfindən verilən məlumatların şifrələnməsi üçün standart kimi istifadə edilən məşhur şifrələmə alqoritmi. DES 64 bitlik bloku şifrələmək üçün istifadə olunur və 64 bitlik açardan istifadə edir (yalnız 56 bit tələb olunur).
3-DES (Triple DES)	64-bit blok şifrəsi, üç fərqli 56-bit düymədən istifadə edərək DES alqoritmini üç dəfə istifadə edir. Bu, DES-ə qarşı müxtəlif hücumlara qarşı dayanıqlığı və təhlükəsizliyi artırır. Bununla belə, 3-DES artıq ən yaxşı şifrələmə seçimi hesab edilmir. Nisbətən yavaş sürətə və daha mürəkkəbdir, xüsusilə də AES kimi daha müasir şifrələmə alqoritmləri ilə müqayisədə onu daha az effektiv və təhlükəsiz edir. Təhlükəsiz şifrələmə tələb olunarsa, AES kimi daha müasir alqoritmlərdən istifadə etmək tövsiyə olunur. AES daha yüksək səviyyədə təhlükəsizlik və səmərəlilik təmin edir. AES geniş şəkildə qəbul edilmiş standarta çevrilib və bir çox proqramlarda və təhlükəsizlik sistemlərində istifadə olunur.
RC2	RSA Data Security tərəfindən hazırlanmış şifrələmə alqoritmidir. Bu, 64 bitlik blok ölçüsünə malik blok şifrəsidir və dəyişən uzunluqlu açarlardan istifadə edir.RC2 şifrələməni DES-dən təxminən iki dəfə tez təmin edə bilər. Bu, məlumatların daha sürətli şifrələnməsinə imkan verir.
RC4	RSA Data Security tərəfindən hazırlanmış axın şifrələmə alqoritmidir. RC4 bayt yönümlüdür və dəyişən uzunluqlu düymələrdən istifadə edir. RC4, DES ilə müqayisədə təxminən 10 dəfə daha sürətli şifrələmə təmin edə bilər. Bu, məlumatların daha tez şifrələnməsinə imkan verir. RC4 alqoritmi RSA Data Security tərəfindən hazırlanmış və ona məxsusdur.
RC5	RSA Data Security tərəfindən hazırlanmış sürətli blok şifrələmə alqoritmidir. RC5 blok ölçüləri 32, 64 və ya 128 bit və uzunluğu 0-dan 2048 bitə qədər dəyişən düymələrə malikdir. 0-dan 255-ə qədər müxtəlif keçid nömrələri də mövcuddur.
R6	Simmetrik blok şifrələmə alqoritmlərindən biridir. Bu alqoritm, Ronald Rivest tərəfindən 1997-ci ildə təqdim edilmişdir. RC6, məlumatı şifrələmək və açmaq üçün ədədi əməliyyatlardan istifadə edir. RC6 alqoritmi, məlumatı ədədi bloklara bölür və hər bir bloku ayrı-ayrı işləyir. Əsasında, RC6, ədədi əməliyyatlara (əlavə etmə, köməkçi əməliyyatlar, sol və sağ dövriyyələr) əsaslanır və açarın uzunluğu və blok ölçüsü dəyişə bilir. 32, 64, 128 bit bloklar üçün RC6 istifadə edilə bilər. RC6, təhlükəsizlik və effektivlik üzərində yaxşı performansa sahibdir. Beləliklə, RC6, məlumatın gizliliyini təmin etmək üçün çeşitli tətbiqlərdə istifadə olunur. Bunlar şəbəkə təhlükəsizliyi, məlumat bazaları, fayl sistemləri, əməliyyat sistemləri və daha çoxdur.

Asimmetrik açar kriptoqrafiyası

Açıq açar kriptoqrafiyası olaraq da bilinən asimmetrik açar kriptoqrafiyası iki açardan, alıcının istifadə etdiyi gizli açardan və ictimaiyyətə elan edilən açıq açardan ibarətdir. Məlumatları şifrələmək və deşifrə etmək üçün bu üsulda iki fərqli açar istifadə olunur. Açıq açar hər kəs üçün əlçatandır, şəxsi açar isə yalnız bu iki açarı yaradan şəxs üçün əlçatandır.



Şək. 1.2 Asimmetrik şifrələnmə sistemi

Müasir informasiya sistemlərində açıq açarla şifrələmə alqoritmlərindən geniş istifadə olunur. Bunlara misal olaraq RSA açıq şifrələmə sistemləri, El Gamal sistemi , Eliptik Əyri Kriptosistemini və s.

Cədvəl 1.2 ən məşhur asimmetrik alqoritmlər göstərilib.

Cədvəl 1.2

Növ	**Təsvir**
RSA	RSA asimmetrik şifrələmə üçün istifadə edilən məşhur alqoritmdir. Mərhələnin gücü böyük tam ədədlərin faktor intensivliyindən asılıdır. RSA ictimai və şəxsi açar cütü vasitəsilə məlumatı şifrələmək və deşifrə etmək üçün çoxlu ədədi əməliyyatlardan istifadə edir.
ECC	ECC, elliptik node termini ilə ifadə olunan cəbri sistemdən istifadə edərək asimmetrik şifrələmə alqoritmini tətbiq etmək üçün istifadə olunur. Digər asimmetrik şifrələmə alqoritmləri ilə müqayisədə o, daha qısa uzunluqlu kabellərdən istifadə edir və daha yüksək performansa malikdir. ECC RSA, Diffie-Hellman və DSA kimi digər açıq açar sistemləri ilə rəqabət aparır.

El-Gamal	El-Gamal həm şifrələmə, həm də elektron imza üçün istifadə edilə bilən Diffie-Hellman variantıdır. Bütün məlumatlar məlumatların şifrələnməsi və deşifrə edilməsi üçün istifadə olunur.

1.2 DES və AES məlumat şifrələmə standartı

DES və AES şifrələmə standartları informasiya təhlükəsizliyi sahəsində geniş istifadə olunan simmetrik şifrələmə alqoritmləridir. DES 1970-ci illərdə IBM tərəfindən hazırlanmış şifrələmə standartıdır. İlk dəfə 1977-ci ildə ABŞ hökuməti tərəfindən nəşr edilmiş və geniş şəkildə istifadə edilmişdir. Bununla belə, DES zamanla hesablama gücünün artması səbəbindən təhlükəsizlik zəifliyinə məruz qalmışdır. DES-in müəyyən edilmiş çatışmazlıqlarını nəzərə alaraq, standartın ilkin versiyasına daim dəyişikliklər edilir; DES-dən əsas istifadə edən yeni alqoritmlər də mövcuddur - NewDES, TripleDES, LUCIFER, MADRYGA, FEAL, REDOC, LOKI, KHUFU, KHAFRE, IDEA, MMB,
CA-1.1, SKIPJACK, CAST, BLOWFISH, SAFER, 3-WAY, CRAB və başqaları. Bununla belə, hazırda DES standartı iki səbəbə görə istifadə üçün tamamilə qəbuledilməzdir:

1) Onun açarının uzunluğu 56 bitdir ki, bu da kompüterin inkişafının indiki mərhələsində olduqca kiçikdir.

2) İnkişaf zamanı alqoritm aparatın tətbiqinə yönəldilmişdir, yəni qəbuledilməz uzun müddət mikroprosessorlarda yerinə yetirilən əməliyyatları ehtiva edirdi. 1990-cı illərdə müxtəlif krekinq üsulları işlənib hazırlandığından təhlükəsizlik səviyyəsi aşağı düşdü. Buna görə də daha güclü şifrələmə standartının, AES-in hazırlanması zərurəti yaranıb. AES 2001-ci ildə NIST (Milli Standartlar və Texnologiya İnstitutu) tərəfindən müəyyən edilmiş şifrələmə standartıdır. AES DES-i əvəz etmək üçün nəzərdə tutulmuşdur və bütün dünyada qəbul edilir. AES müxtəlif açar uzunluqlarını (128, 192 və 256 bit) dəstəkləyən blok şifrələmə alqoritmidir. Emal sürəti və təhlükəsizliyi baxımından DES-dən daha yaxşıdır və bu gün müxtəlif tətbiqlərdə geniş istifadə olunur.

AES üçün tələblər son dərəcə sadədir [8]:

1) Alqoritm simmetrik olmalıdır.

2) Alqoritm blok şifrəsi olmalıdır.

3) Alqoritmin blok uzunluğu 128 bit olmalı və üç açar uzunluğunu dəstəkləməlidir: 128, 192 və 256 bit.

AES daha təhlükəsiz və daha sürətli alqoritm hesab edilsə də, DES zəifliklərə malik olduğu üçün bu gün istifadədə daha məhduddur.

Cədvəldə ilk 5 AES nümayəndəsi aşağıdakılardır:

Cədvəl 1.3

Alqoritm	**Yaradıcısı**	**Performansı**
MARS	IBM	200 MHz-də 8 MB/s
RC6	R.Rivest & Co	12 MB/s
Rijndael	V.Rijmen & J.Daemen	7 MB/s
Serpent	Universitetlər	2 MB/s
Twofish	B.Schneier & Co	11 MB/s

Bu alqoritmlər arasında Rijndael rəsmi olaraq AES kimi qəbul edilmiş və geniş istifadə olunan standart AES alqoritmidir. Digər alqoritmlər AES ilə oxşar təhlükəsizlik səviyyələrinə malik alternativlərdir. Hər bir alqoritm müxtəlif xüsusiyyətlərə, sürətlərə və təhlükəsizlik profillərinə malikdir.

1.3 Asimmetrik kriptosistemlərə ümumi baxış

Asimmetrik alqoritmlərdə bunlar öyrənilir: RSA açıq açar şifrələmə sistemi, ElGamal şifrələmə sistemi və elliptik əyrilərə əsaslanan şifrələmə sistemləri.

1.3.1 RSA açıq açar sistemi

Açıq açar sistemləri arasında ən populyarı 1977-ci ildə hazırlanmış və onun yaradıcıları Ron Rivest, Adi Şamir və Leonard Adleman tərəfindən adlandırılan RSA şifrələmə sistemi idi. RSA alqoritmi verilənlərin şifrələnməsi və rəqəmsal imza rejimində işləməyə qadir olan ilk tam hüquqlu açıq açar alqoritmi kimi tanınıb [6]. Onlar böyük sadə ədədlərin hesablamada asan tapılmasından istifadə etdilər, lakin bu ədədləri faktorinqlə ayırmaq demək olar ki, mümkün deyil. RSA-nın şifrəsinin

açılması bu faktorizasiyaya (Rabin teoremi) ekvivalent olduğu sübut edilmişdir. Buna görə də, istənilən verilmiş açar uzunluğu üçün şifrəni açmaq üçün tələb olunan əməliyyatların sayı üçün aşağı həddi təxmin etmək olar və bu əməliyyat üçün tələb olunan vaxt müasir kompüterlərin performansını nəzərə alaraq təxmin edilə bilər. RSA alqoritminin təhlükəsizliyini təmin etmək qabiliyyəti, onlarla digər sxemlərlə müqayisədə onu açıq açar sistemlər arasında populyar edən səbəblərdən birinə çevrilmişdir. Buna görə də, RSA alqoritmi bank kompüter şəbəkələrində, xüsusilə uzaq müştərilərlə işləmək üçün istifadə olunur (məsələn, kredit kartı əməliyyatları).

Bu gün RSA alqoritmi SSL, S-HTTP, S/MIME, IPsec (S/WAN), SSH və PCT kimi bir çox standartlarda istifadə olunur. RSA alqoritmi üçün əsas generasiya mərhələsi aşağıdakı əməliyyatlardan ibarətdir[1-3]:

Addım 1:

a. İki fərqli sadə ədəd p və q seçin. Bu dədlər böyük və tək-tək olmalıdır.

b. P və q ədədlərinin mövcudluğu və böyüklüyü yoxlanılmalıdır.

Addım 2: Modulun hesablanması

a. n modulu hesablanır:

$$n = p * q.$$

Addım 3: Eyler (Φ) funksiyasının hesablanması

a. Eyler funksiyası (Φ) hesablanır:

$$\Phi(n) = (p - 1) * (q - 1).$$

b. Eylerin funksiyası n ədədinə görə ədədlərin çoxalması ilə müəyyən edilir.

Addım 4: Ən böyük ortaq bölənin hesablanması (ƏBOB)

a. ƏBOB(n, e) = 1, 1-dən və ə(n)-dən kiçik olmayan bir e dən seçin.

b. e obyektiv olaraq Fermat üsulu ilə seçilir.

Addım 5: Gizli açarın generasiyası

a. Gizli açarın bir hissəsi olacaq, d ədədini uyğun olaraq tapılması

$$d * e \equiv 1 \pmod{\Phi(n)}$$

b. d, ədədi n ədədindən böyük və ya Φ(n)-dən böyük olmalıdır.

Addım 6: Açıq və gizli açılışları müəyyənləşdirin

a. Açıq (n, e) kimi təyin edilmişdir.

b. O, gizli açar (n, d) kimi təyin olunur.

Bu əməliyyatlarla birlikdə RSA alqoritmi üçün əsas generasiya mərhələsi tamamlanır və açıq və gizli açarlar təyin edilir. Bu açarlar, mesajların şifrələnməsi və deşifrələnməsi üçün istifadə olunur. Ancaq qəbul edən tərəfdə şifrələnmiş mesajların şifrəsini açmaq mümkündür və gizli d nömrəsi bu işdə bizə kömək edir. Eyler teoremi çoxdan sübut edilmişdir və xüsusi halı bildirir ki, n iki sadə ədədlə, p və q ilə təmsil oluna bilər. Bu halda

$$(x^{(p-1)(q-1)}) \bmod n = 1$$

tənliyi istənilən x üçün yerinə yetirilir. RSA mesajlarının şifrəsini açmaq üçün bu düsturdan istifadə edəcəyik. Hər iki tərəfi (-y) qüvvətinə qaldıraq:

$$(x^{(-y)(p-1)(q-1)}) \bmod n = 1^{(-y)}\ 1$$

İndi hər iki tərəfi x-ə vuraq:

$$(x^{(-y)(p-1)(q-1)+1}) \bmod n = 1*x = x$$

İndi açıq və gizli açarların necə yaradıldığını xatırlayaq. Evklid alqoritmindən istifadə etməklə d elə seçilmişdir ki,

$$e*d+(p-1)(q-1)*y=1, \text{ yəni } e*d=(-y)(p-1)(q-1)+ 1$$

Nəticə olaraq, göstərici əvvəlki abzasdakı ifadədə

$$(e*d)$$

olaraq dəyişdirilə bilər. Bu halda

$$(x^{e*d})\text{mod } n=x$$

alırıq. Yəni

$$c_i=((m_i)^e)\text{mod } n$$

mesajlarını oxumaq üçün onu m-ə münasibətdə d-nin gücünə qaldırmaq kifayətdir:

$$((c_i)^d)\text{mod } n = ((m_i)^{e*d}) \text{ mod } n = m_i.$$

1.3.2 ElGamal şifrələmə sistemi

Bu sistem RSA-ya alternativdir və eyni kriptoqrafik gücü bərabər açar dəyəri ilə təmin edir[7]. RSA-dan fərqli olaraq, El-Gamal metodu diskret loqarifm məsələsinə əsaslanır. Bu baxımdan o, Diffie-Hellman alqoritminə bənzəyir. Ədədin eksponensial funksiyasını hesablamaq asan olsa da (yəni ədədi qüvvətə yüksəltməklə), verilmiş ədədin arqumentini geri tapmaq (yəni onun loqarifmini tapmaq) olduqca çətindir. Sistemin əsasını təşkil edən parametrlər p və g - birincisi sadə ədəd, ikincisi tam ədəddir. A tərəfi a məxfi açarını yaradır və

$$y = g\text{^}a \text{ mod } p$$

düsturu ilə açıq açarı y hesablayır. B tərəfi A-ya çatdırmaq istədiyi m mesajı üçün təsadüfi k ədədi seçir, bu k ədəd p-dən kiçik olmalıdır və o, aşağıdakı hesablamaları aparır:

$$y1 = g^k \bmod p \text{ və}$$

$$y2 = m \text{ XOR } y^k,$$

burada XOR mod 2 əlavə əməliyyatıdır. B tərəfi (y1, y2) dəyərləri A-ya göndərir. Şifrələnmiş mesajı qəbul edən A Tərəfi mesajı aşağıdakı kimi geri alır:

$$m = (y1^a \bmod p) \text{ XOR } y2$$

DSS standartının bir hissəsi olan NIST tərəfindən hazırlanmış rəqəmsal imza alqoritmi olan DSA qismən bu metoda əsaslanır.

1.3.3 Elliptik tənliklərə əsaslanan kriptosistemlər

Elliptik qrafiklər istənilən sahədə (sonlu, real, rasional və ya mürəkkəb) müəyyən edilə bilən riyazi obyektdir. Sonlu sahələr ümumiyyətlə kriptoqrafiya sahəsində istifadə olunur. Elliptik qrafik(x, y) nöqtələr çoxluğu və aşağıdakı tənliyi təmin edən sonsuz uzaq nöqtə kimi müəyyən edilir [7]:

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

Qrafikdə ki nöqtələr üçün əlavə əməliyyatı kifayət qədər asanlıqla müəyyən edilir və RSA və El-Qamal kriptosistemlərində vurma əməliyyatına oxşar rol oynayır. Həqiqi kriptosistemlərdə elliptik tənliklərə əsaslanan aşağıdakı tənlik istifadə olunur:

$$y^2 = x^3 + ax + b \bmod p$$

burada p sadə ədəddir.Elliptik əyri üzərində diskret loqarifm məsələsi aşağıdakı kimidir: r sıralı elliptik əyridə G nöqtəsi (əyri üzərindəki nöqtələrin sayı) və eyni əyri üzərində başqa bir Y nöqtəsi verilmişdir.

$$Y = xG$$

tənliyinə tək x nöqtəsini tapmaq lazımdır, ona görə də Y x dəfə G-dir.

1.4 Şifrləmə alqoritmlərinin müqayisəsi

RSA kimi kriptoalqoritmlərin proqram təminatı tətbiqləri DES kimi klassik kriptoalqoritmlərin tətbiqindən daha mürəkkəb və daha az səmərəlidir. Modul hesab əməliyyatlarının mürəkkəbliyinə görə, RSA kriptoalqoritmi adətən yalnız kiçik həcmli məlumatların şifrələnməsi üçün istifadə olunur, məsələn, klassik məxfi açarların və ya rəqəmsal imza alqoritmlərinin paylanmasında, göndərilən məlumatların əksəriyyəti isə simmetrik alqoritmlərlə şifrələnir [5].

Cədvəl 1.4-də klassik DES kriptoalqoritmi ilə RSA kriptoalqoritminin bəzi müqayisəli xüsusiyyətləri verilmişdir.

Cədvəl 1.4

Xarakterik	**DES**	**RSA**
Şifrələmə sürəti	Yüksək	Aşağı
İstifadə olunan kriptoqrafik funksiya	Permutasiya və əvəzetmə	Eksponentasiya
Açar uzunluğu	56 bit	500 bitdən çox
Ən Az Xərcli Kriptoanaliz (Alqoritmin Davamlılığını Müəyyən Etmə Problemi)	Klaviatura sahəsinin bütün kombinasiyalarını sınaqdan keçirin	Ədədin əsas faktorizasiyası
Açar generasiya vaxtı	Milisaniyələr	Dəqiqələr
Açar növü	Simmetrik	Asimmetrik

1.5 Rəqəmsal İmza

Elektron-rəqəmsal imza son bir neçə ildə informasiya emalı texnologiyasının kağız əsaslı proseslərin tədricən elektron mühitə keçməsi tendensiyası ilə gündəmə gəlib. Zamanla kağız sənədlərin dövriyyəsinin tamamilə elektron daşıyıcılarla

əvəzlənəcəyi gözlənilir. Bununla belə, ənənəvi kağız sənədlərin sıfır və birlərin elektron sətirləri kimi təqdim edilməsi onları şəxsiyyətsizləşdirir. Kağız sənədlərdə olan qoruyucu xüsusiyyətlər: imzalar, möhürlər və ştamplar, su ştampları, xüsusi teksturalı kağız səthi - elektron sənədlərdə yoxdur. Bununla belə, elektron sənədlər kağız sənədlərdən daha az ehtiyatla qorunmalıdır. Buna görə də kağız sənədlərdə imza və möhürü əvəz edə bilən elektron mühafizə mexanizminin yaradılması vəzifəsi yaranır. Başqa sözlə, qorunan məlumatlara əlavə olunan məlumat olan elektron-rəqəmsal imza mexanizmini hazırlamaq lazımdır. Rəqəmsal imza imzalanmış sənədin məzmunu və yalnız təhlükəsiz rabitədə iştirak edən şəxsə məxsus olan məxfi element (açar) ilə əlaqələndirilir.

Bu mexanizm aşağıdakı funksiyaları təmin edir:

1. Rəqəmsal imza, imzalayanın təsadüfən elektron sənədi imzalamadığını yoxlamalıdır.
2. Rəqəmsal imza yalnız imzalayanın və yalnız onun elektron sənədi imzaladığını təsdiq etməlidir.
3. Rəqəmsal imza imzalanmış sənədin məzmunundan və imzalanma vaxtından asılı olmalıdır.
4. İmzalayanın sonradan sənədi imzalamaqdan imtina etmək imkanı olmamalıdır.

1.5.1 Simmetrik kriptosistemləri istifadə edərək rəqəmsal imza

Rəqəmsal imzanın ilk versiyaları simmetrik şifrələmə sistemlərindən istifadə etməklə hazırlanmışdır. Bu sistemdə mesaj mübadiləsində iştirak edən abunəçilər sənədi imzalamaq və imzanı yoxlamaq üçün eyni məxfi açardan istifadə edirlər. Xüsusi emal rejimlərinə malik istənilən simmetrik kriptosistem kriptoqrafik çevrilmə alqoritmi kimi istifadə edilə bilər (məsələn, DES, QOST 28147-89 və s.).

Simmetrik şifrələmə sistemindən istifadə edərək sənədlərin imzalanması proseduru aşağıdakı kimi ola bilər:

1. Göndərən A və qəbuledici B eyni məxfi açara malikdir, K.
2. Göndərən A rəqəmsal mesajı K düyməsini istifadə edərək şifrələyir və şifrələnmiş mesajı B qəbuledicisinə göndərir. Bu prosesdə xüsusi emal rejimləri

olan şifrələmə sistemindən (məsələn, GOST 28147-89-da autentifikasiya rejimi) istifadə olunur.

3. Qəbuledici B mesajı K düyməsi ilə deşifrə edir.

Yalnız A və B istifadəçilərinin məxfi açarı olduğundan, bu yolla mesajın C kimi hacker tərəfindən deyil, A tərəfindən imzalandığına zəmanət verilir. Lakin bu sxem yalnız hər bir abunəçinin etibarlılığına yüz faiz zəmanət verilə bilən şəbəkələrdə tətbiq oluna bilər. Əks halda, gizli açarı olan abunəçi fırıldaqçılıq potensialına malikdir. Bu dezavantajı aradan qaldırmaq üçün etibarlı interfeysə malik bir sxem təklif olunur. Bu sxemə A və B istifadəçiləri, həmçinin X interfeysi daxildir. X həm göndərici A, həm də B qəbuledicisi ilə əlaqə saxlaya bilir və hər ikisinin KA və KB məxfi açarları var. Bu sxemdə sənədlərin imzalanması proseduru aşağıdakı kimi ola bilər:

1. Göndərən A KA açarından istifadə edərək mesajı şifrələyir və onu X interfeysinə göndərir.
2. Interface X KA düyməsini istifadə edərək mesajın şifrəsini açır.
3. X interfeysi şifrlənmiş mesajı B qəbuledicisinin KB açarı ilə şifrələyir.
4. X interfeysi şifrələnmiş mesajı B qəbuledicisinə göndərir.
5. Qəbuledici B KV açarı ilə X interfeysindən aldığı mesajın şifrəsini açır.

Bu sxem nə dərəcədə effektivdir? Rəqəmsal imza üçün göstərilən tələbləri nəzərə alaraq, sxemi qiymətləndirək:

1. X interfeysi və B qəbuledicisi mesajın həqiqətən A göndəricisindən gəldiyini bilir. İnterfeys X-in etibarı yalnız göndərici A və X interfeysinin KA məxfi açarına malik olmasına əsaslanır. Interface X-in təsdiqi B alıcısına sübut təqdim edir.
2. Yalnız göndərən A KA açarını (və etibarlı interfeys X) bilir. X interfeysi yalnız KA açarında şifrələnmiş mesajı göndərən A-dan qəbul edə bilər. Zərərli C göndərən A adından mesaj göndərməyə cəhd edərsə, X interfeysi onu dərhal 2-ci addımda aşkar edir.
3. Bu, B qəbuledicisi X interfeysindən aldığı rəqəmsal imzanı başqa bir mesaja əlavə etməyə və onu A göndəricisinin mesajı kimi təqdim etməyə çalışdıqda aşkar edilir. X interfeysi bu mesajı B qəbuledicisindən tələb edə və onu KA açarı

ilə şifrələnmiş mesajla müqayisə edə bilər. İki mesaj arasındakı uyğunsuzluq dərhal görünür. Əgər B qəbuledicisi göndərilən mesajı dəyişdirməyə çalışırsa, X interfeysi onu oxşar şəkildə aşkar edir.

4. Göndərən A imzalanmış mesajın göndərilmədiyini rədd edərsə, X interfeysinin təsdiqi əksini bildirir.

Bu diaqramda ən kritik nöqtənin X interfeysi olduğunu müşahidə edə bilərik. Birincisi, X heç kimdən asılı olmamalıdır. İkincisi, X interfeysi tamamilə səhvsiz olmalıdır. Bir neçə min mübahisəli işdən yalnız birində səhv etmək X-in etibarına və əvvəllər imzalanmış sənədlərə xələl gətirəcək.

Beləliklə, simmetrik şifrələmə sistemləri nəzəri cəhətdən istifadə oluna bilsə də, praktikada imza yaratmaq üçün istifadə edilmir, çünki onlar ötürülən məlumatların adekvat etibarlılığını təmin etmək üçün bahalı və mürəkkəb tədbirlər tələb edir.

1.5.2 Asimmetrik kriptosistemlərdən istifadə edərək rəqəmsal imza

Asimmetrik kriptosistemlər vasitəsilə rəqəmsal imzanın yaradılması prosesi elektron məlumatın təhlükəsizliyini və mənbənin məxfiliyini təmin etmək üçün istifadə olunur. Bu üsulda hər bir fərd bir-birinə bağlı açıq və gizli açarlardan istifadə edir[5]. İctimai imza prosesi aşağıdakı addımları yerinə yetirir:

1. Məlumat rəsmi imza üçün hazırlanır. Bu məlumat imzalanacaq akt, elektron məktub və ya hər hansı digər məlumat fondu ola bilər.
2. Məlumatın hazırlanması üçün gizli açılışların istifadə olunduğu funk ilə məlumatın fərdiləşdirilməsi prosesi həyata keçirilir. Bu proses bəzi riyazi məqsədlərlə məlumatın dəyişdirilməsi ilə həyata keçirilir və imza üçün bir çıxış hazırlanır.
3. Əldə edilmiş fərdiləşdirilmiş məlumat imza kimi istifadə olunur və məlumatla birlikdə göndərilir.
4. İnformasiyanın alınması məlumatın təmiz olub-olmadığını yoxlamaq üçün aşkar olandan istifadə etməklə fərdiləşdirilmiş məlumatı faktiki məlumata çevirir.
5. Fərdiləşdirilmiş məlumatın faktiki məlumatla uyğunluğunu yoxlamaq üçün orijinal məlumat funksiya ilə işlənir. Bu əməliyyat məlumatın gizli açılması ilə

məlumatın uyğunluğunu yoxlamaq məqsədilə həyata keçirilir. Əgər göstərilən məlumatın ilkin məlumatla, imza tarixi (rəqəmsal imza) və silinərsə, məlumatın orijinal olduğu hesab edilir və göndərənin şəxsiyyəti yoxlanılır. Bu üsul həm məlumatın təmizliyini və həqiqiliyini, həm də məlumatı göndərən şəxsin kimliyini yoxlamağa imkan verir. Bu elektron ticarətdir.

Bu təsnifatda ətraflı təhlil göstərir ki, açıq şifrələmə sistemlərinə əsaslanan digər rəqəmsal imza kodları bütün sorğulara tam cavab verir. Hal-hazırda RSA, El-Gamal, Schnorr alqoritmləri geniş yayılmışdır.

Cədvəl 1.5-da əsas elektron-rəqəmsal imza nömrələri göstərilir.

Cədvəl 1.5

Növ	**Şərhlər**
DSA	Elektron imza yaratmaq üçün istifadə olunur, lakin şifrələmə üçün deyil. Məxfi fərdiləşdirmə və məlumatın kollektiv yoxlanışı - yalnız bir şəxs məlumatı fərdiləşdirə bilər, lakin hər kəs onun doğruluğunu yoxlaya bilər. Sonlu sahələrdə loqarifmlərin alınmasının hesablanması intensivliklərə əsaslanır.
RSA	RSA elektron imzası mesajın bütövlüyünü və imzanı yaradan şəxsin şəxsiyyətini yoxlamağa imkan verir. O, göndərilən mesajın kriptoqrafik funksiyasını yaradır və onun gizli açarından istifadə edərək onu şifrələyir. Alıcı mesajın parolunu açmaq, mesajın mahiyyətini hesablamaq və bu ikisini müqayisə etmək üçün göndərənin açarından istifadə edir.
MAC	MD və ya SHA və toplama sxemlərindən istifadə edərək elektron imzadan istifadə etməklə, lakin mesaj məlumatı və məxfi açardan istifadə etməklə hesablanır.
DTS	İstifadəçiləri kriptoqrafik cəhətdən təhlükəsiz şəkildə sənəd məlumatları ilə əlaqəli vaxt ştampları ilə təmin edir.

1.6 Parolların yaradılması

Parolların yaradılması istifadəçilərin 128 baytlıq yeni 256 hexadecimal sətir əvəzinə mənalı ifadə, söz və ya simvol ardıcıllığı kimi parolu yadda saxlaya bildiyi üsul hesab olunur. Lakin hər hansı kriptoalqoritm hazırlanarkən nəzərə alınmalıdır ki,

sistemin sonuncu istifadəçisi avtomatik sistem deyil, insandır. Bu, insanın 128 bitlik açarı (32 onaltılıq rəqəm) yadda saxlamasının rahat və hətta mümkün olub-olmadığı sualını doğurur. Bir sözlə, saxlama limiti 8-12 simvol ardıcıllığı diapazonundadır. [3]. Bu problemi həll etmək üçün istənilən uzunluqdakı bir xətti yeni parolla yeni paroldan müəyyən uzunluqdakı sətirə çevirən üsullar hazırlanmışdır. Bu məqsədlə müxtəlif funks (ing. hashing - qarışdırma) istifadə olunur. Hash funksiyası müəyyən bir məlumat blokunun riyazi və ya alqoritmik tərcüməsi adlanır və aşağıdakı xüsusiyyətlərə malikdir.

1. Həş funksiyasının sahəsi sonsuzdur, yani hər hansı bir məlumat bloku üçün bir həş dəyəri alınabilər.
2. Həş funksiyasının dəyər sahəsi məhdud olmalıdır, yəni bir həş dəyəri qəbul edən həş funksiyası mütləq bir dəyər yaratmalıdır.
3. Həş funksiyası tərsinə çevrilməz, yani həş dəyərindən məlumat blokunu geri çevirmək mümkün deyil.
4. Giriş məlumatının yalnız bir bit dəyişdirilməsi, çıxış həş dəyərinin yarıya yaxınını dəyişdirməlidir, yəni həş funksiyası çıxışının bitlərinin təxminən yarısını dəyişdirməlidir.

Bu xüsusiyyətlər həş funksiyasının məlumatın bütünlüyünü yoxlamaq, məlumatın unikal identifikasiyasını təmin etmək və həşlənmiş məlumatların müəyyənləşdirilməsinə imkan verir. Həş funksiyaları şifrələmə, elektron müqavilələr, verilənlər təhlili və digər tətbiqlərdə geniş istifadə olunur. Aşağıdakı cədvəldə ümumi hash funksiyaları verilmişdir.

Cədvəl 1.6

Növ	**Təsvir**
MD2	Ən yavaş, 8 bitlik maşınlar üçün optimallaşdırılmışdır
MD4	Ən sürətli, 32 bitlik maşınlar üçün optimallaşdırılmışdır Bu yaxınlarda sındırıldı
MD5	MD funksiyaları ailəsinin ən ümumi. MD4-ə bənzəyir, lakin təhlükəsizlik təkmilləşdirmələri onu MD4-dən 33% yavaş edir Məlumatların bütövlüyünü təmin edir və təhlükəsiz hesab olunur.

SHA	Dəyişən ölçülü giriş məlumatlarından 160 bitlik hash dəyəri yaradır. NIST tərəfindən təklif edilmiş və ABŞ hökuməti tərəfindən standart olaraq qəbul edilmişdirDSS standartında istifadə üçün nəzərdə tutulmuşdur

1.7 Kriptoqrafik açarların idarə edilməsi

Müəyyən bir informasiya sistemi üçün uyğun kriptoqrafik məlumatın mühafizəsi vasitələrinin seçilməsi ilə yanaşı, açarların idarə edilməsi mühüm məsələdir. Kriptosistemin özü nə qədər mürəkkəb və təhlükəsiz olsa da, açarların istifadəsinə əsaslanır. Əgər iki istifadəçi arasında məxfi məlumat mübadiləsini təmin etmək üçün açarların mübadiləsi prosesi əhəmiyyətsizdirsə, istifadəçilərin sayının yüzlərlə və minlərlə olduğu informasiya sistemində açarların idarə edilməsi ciddi problemdir.

1.7.1 Simmetrik Metodologiya

Bu metodologiyada şifrələmə və deşifrə üçün göndərən və qəbul edən tərəfindən eyni açardan istifadə edilir və rabitə başlamazdan əvvəl razılaşdırılmış açardan istifadə edilməsi nəzərdə tutulur. Açar tutulmazsa, deşifrə prosesi zamanı göndərənin şəxsiyyəti avtomatik olaraq yoxlanılır, çünki yalnız göndərəndə məlumatı şifrələmək üçün istifadə edilə bilən açar var və yalnız qəbul edəndə məlumatın şifrəsini açmaq üçün istifadə edilə bilən açar var. Açarı yalnız göndərən və qəbul edən şəxs bildiyi üçün açarın ələ keçirilməsi halında yalnız bu iki istifadəçinin qarşılıqlı əlaqəsi pozulur. Bu simmetrik (məxfi) açarların təhlükəsiz şəkildə paylanması problemi digər kriptosistemlərə də aid olan problemdir.

Simmetrik düymələri olan sistemlərdən istifadə qaydası:

1. Simmetrik gizli açar təhlükəsiz şəkildə yaradılır, paylanır və saxlanılır.
2. Göndərən mətnin hashini hesablayır, elektron imza yaradır və əldə olunan sətri mətnə əlavə edir.
3. Göndərən məxfi simmetrik açarla sürətli simmetrik şifrələmə/şifrləmə alqoritmini qəbul edən paketə tətbiq edir (mətnin elektron imzası ilə) və şifrəli

mətni əldə edir. Bu yolla autentifikasiya avtomatik həyata keçirilir, çünki yalnız göndərən simmetrik məxfi açarı bildiyi üçün bu paketi şifrələyə bilər. Simmetrik məxfi açarı bildiyi üçün yalnız alıcı bu paketin şifrəsini aça bilər.

4. Göndərən şifrələnmiş mətni ötürərək göndərir. Simmetrik məxfi açar heç vaxt təhlükəsiz olmayan rabitə kanalları üzərindən ötürülmür.
5. Qəbul edən eyni simmetrik şifrələmə/şifrə açma alqoritmini və eyni simmetrik açarı şifrələnmiş mətnə (artıq alıcıda var) tətbiq etməklə orijinal mətni və elektron imzanı bərpa edir. Uğurlu bərpa məxfi açarı bilən şəxs tərəfindən autentifikasiyanı təmin edir.
6. Qəbul edən elektron imzanı mətndən ayırır.
7. Qəbul edən mətnin hashini hesablayaraq başqa bir elektron imza yaradır.
8. Alıcı mesajın bütövlüyünü yoxlamaq üçün bu iki elektron imzanı müqayisə edir.

Bu gün mövcud olan simmetrik metodologiyadan istifadə edən sistemlər:

1. Kerberos verilənlərin autentifikasiyası üçün deyil, şəbəkədəki resurslara girişi təsdiqləmək üçün nəzərdə tutulub. O, bütün istifadəçilərin şəxsi açarlarının surətlərini saxlayan mərkəzi verilənlər bazasından istifadə edir.
2. ATM bank şəbəkələri. Bu sistemlər əvvəlcə sahibləri tərəfindən hazırlanmış və satılmayan sistemlərdir. Bunlar da simmetrik metodologiyalardan istifadə edirlər.

1.7.2 Asimmetrik Metodologiya

Bu metodologiyada şifrələmə və deşifrə üçün müxtəlif açarlardan istifadə olunur, lakin onlar birlikdə yaradılır. Bir açar açıq, digər açar gizli saxlanılır. Şifrələmə və şifrənin açılması hər iki açarla edilə bilər, lakin bir açarla şifrələnmiş məlumat yalnız digər açarla deşifrə edilə bilər. Bütün asimmetrik kriptosistemlər simmetrik kriptosistemlərdə istifadə olunanlardan daha uzun açarlardan istifadə etməlidirlər, çünki onların açarları birbaşa kobud güc hücumlarına məruz qalır və beləliklə,

ekvivalent qoruma səviyyəsini təmin edir. Bu, şifrələmə üçün tələb olunan hesablama resurslarına dərhal təsir edir, lakin elliptik əyrilərdə şifrələmə alqoritmləri bu problemi yüngülləşdirə bilər.

Cədvəl 1.7 – də simmetrik və açıq açarların ekvivalent uzunluqları haqqında məlumatları göstərir.

Cədvəl 7.1

Simmetrik açar uzunluğu	**İctimai açar uzunluğu**
56 bit	384 bit
64 bit	512 bit
80 bit	768 bit
112 bit	1792 bit
128 bit	2304 bit

Asimmetrik açarları olan sistemlərdən necə istifadə etmək olar:

1. Göndərən tərəf:
   - Mətnin hash funksiyası (kriptografik mürəkkəb əməliyyat) hesablanır.
   - Hesablanan hash dəyəri, göndərənin gizli asimmetrik məxfi açarı ilə şifrələnir.
   - Şifrələnmiş hash dəyəri, mətnin özü ilə birlikdə əlavə olunur (elektron imza yaradılır).
   - Bu proseslərin nəticəsi olaraq şifrəli mətn əldə edilir.
2. Şifrələnmiş mətnin ötürülməsi:
   - Şifrəli mətn, alıcıya göndərilir.
3. Qəbul edən tərəf:
   - Alıcı, şifrəli mətni qəbul edir.
   - Alıcı, öncədən əldə etdiyi asimmetrik sertifikat orqanının (CA) açıq açarını istifadə edərək göndərənin asimmetrik açarını doğrulayır.
   - Alıcı, doğruladığı göndərənin asimmetrik açarını istifadə edərək şifrələnmiş mətndən elektron imzanı ayrışdırır.
   - Şifrələnmiş mətnin özündən, alıcı elektron imzasını ayırır.
   - Alıcı, şifrələnmiş mətndən çıxarılır şifrəli sessiya açarını alır.
4. Sessiya açarının açılması:

- Alıcı, öncədən əldə etdiyi asimmetrik sertifikat orqanının (CA) açıq açarını istifadə edərək sessiya açarını açar.
- Açılan sessiya açarı, simmetrik şifrələmə/şifrləmə alqoritmi ilə birlikdə kullanılarak şifrəsi açılan mətn əldə edilir.

Buna görə də, alıcı başlangıcdakı mətni, mətnin dəyişdirilmədiyini yoxlamaq üçün hash funksiyası ilə hesablanan dəyəri göndərənin asimmetrik açarı ilə deşifrə edərək müqayisə edir. Əgər hesablanan və deşifrə edilən hash dəyərləri uyğun gəlirsə, mətnin dəyişdirilmədiyi təsdiq olunur.

II FƏSİL Modifikasiya olunmuş RC6 alqoritminin layihələndirilməsi

2.1 Standart RC6 şifrələməsi

Standart RC6 (Rivest Cipher 6), Ron Rivest, M.J.B. Robshaw, R. Sydney və Y.L. Yin ilə 1997-ci ildə təqdim edilmiş blok şifrələmə alqoritmidir. RC6 simmetrik açar şifrələmə alqoritmidir və müəyyən ölçülü bloklardakı mesajları şifrələmək və deşifrə etmək üçün istifadə olunur.

Əməliyyat prinsipi:

1. RC6 verilmiş mesajı bloklarda emal edir. Hər blok 128 bitdir (16 bayt).
2. Şifrələmə prosesi üçün açarı qəbul edir. Açarın uzunluğu 128, 192 və ya 256 bit ola bilər.
3. Şifrələmə üçün əsas funksiyalara XOR, modifikasiyalar və bit dəyişiklikləri daxildir.
4. Şifrələmə prosesi döngələr və dövrələr vasitəsilə həyata keçirilir. Döngələr daxili funksiyaları təkrarlayır və şifrələmə gücləndirilir.
5. RC6 alqoritminin əsas hərəkəti açar və blok qiymətlərinin birləşməsidir. Bu prosesdən istifadə olunan məhsullar (w, r, b) və dəyişdirilmələr (S) kimi istifadə olunur.

Təhlükəsizlik və performans:

- Standart RC6 təhlükəsizlik və performans baxımından güclü alqoritmdir.

- Açar uzunluğu və dövrlərin sayı artdıqca şifrələmə gücü artır, lakin emal sürəti azalır.İstifadəçilər təhlükəsizlik və performans arasında balans seçməlidirlər.

- Standart RC6 təhlükəsiz analiz, riyazi model və praktiki testlərdən keçdi. Əsasən, bu alqoritm möhkəm şifrələmə metodunu təmsil edir.

RC6 mətni, formanı, səsi və digər məlumatları şifrələmək və onları təhlükəsiz şəkildə çoxaltmaq üçün istifadə olunan alqoritmdir. Bununla belə, RC6-dan istifadə etməyə başlamazdan əvvəl potensial təhlükələrə, sənaye standartlarına və ən son tendensiyalara riayət etmək vacibdir[5].

2.1.1 Standart RC6 açarının yaradılması prosesi

Aşağıda RC6 alqoritminin əsas hesablama alqoritminin blok diaqramı verilmişdir:

Giriş:

- L[0..c-1] sətirində əvvəlcədən yüklənmiş b bayt uzunluğunda istifadəçi tərəfindən müəyyən edilmiş açar
- Şifrələmə dövrlərinin sayı r

Çıxış:

- S[0.. 2r + 3] əsas cədvəlində w-bit sözləri

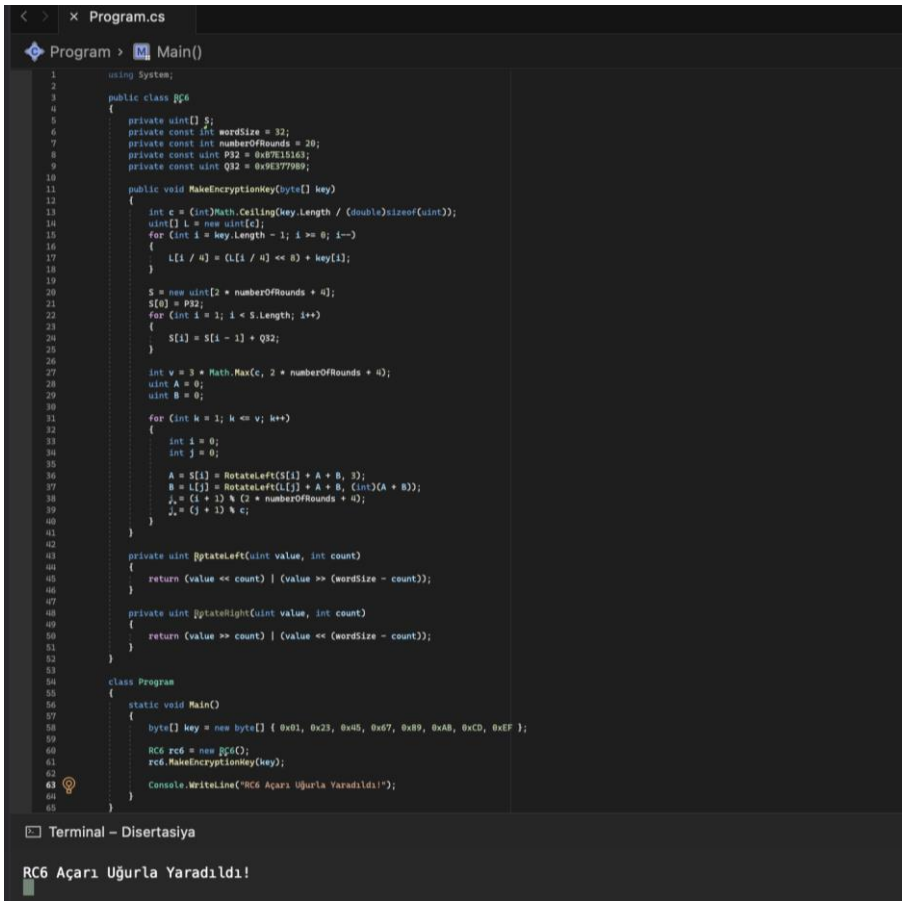
Prosedur:

- S[0] = P32 (sabit dəyər)
- i = 1-dən 2r+3-ə qədər
  - S[i] = S[i-1] + Q32 (sabit dəyər)
- v = 3 * maks{c,2r + 4}
- A = B = i = j = 0
- k = 1-dən v
  - A = S[i] = (S[i] + A + B) <<< 3 (w-bit sola sürüşmə)
  - B = L[j] = (L[j] + A + B) <<< (A + B) (w-bit sola sürüşdürmə)
  - i = (i+1) mod (2r+4)
  - j = (j+1) mod c

Son

Bu blok diaqram RC6 alqoritminin əsas hesablama addımlarını göstərir. Əsas cədvəl S əvvəlcədən təyin edilmiş sabitlərlə işə salınır. Sonra dövrədə S cədvəli yenilənir və A, B, i və j dəyişənlərindən istifadə etməklə hesablamalar aparılır. Bu hesablamalar dövrə boyu müəyyən yerdəyişmələr və birləşmələr tətbiq etməklə həyata keçirilir.

Açar yaratma alqoritminin C# proqramlaşdırma dilində proqram təminatının tətbiqi:



Şək. 2.1 Standart RC6 açarının yaradılması prosesi

2.1.2 Standart RC6 şifrələmə prosesi

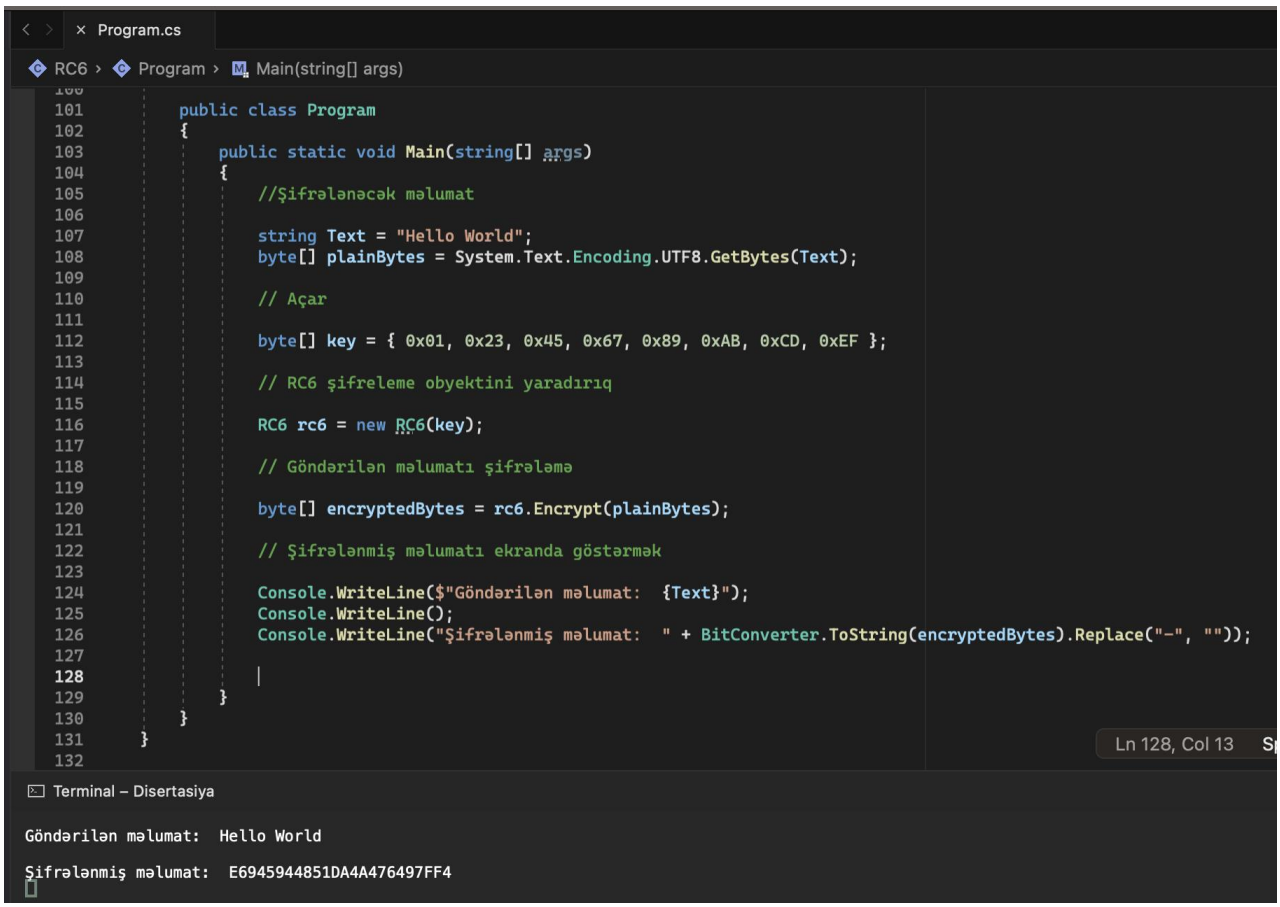
RC6-w/r/b üçün şifrələmə (w bitlərdə sözün uzunluğudur, r təkrarlanan şifrələmə dövrlərinin sıfırdan fərqli sayıdır, b baytlarda açar uzunluğudur) aşağıdakı kimi təsvir edilir:

Program.cs

RC6 › Q32

```
48
49      public byte[] Encrypt(byte[] plainText)
50      {
51          int blockSize = sizeof(uint);
52          int paddedLength = ((plainText.Length + blockSize - 1) / blockSize) * blockSize;
53          uint[] cipherText = new uint[paddedLength / blockSize];
54          for (int i = 0; i < plainText.Length; i += blockSize)
55          {
56              uint block = 0;
57              for (int j = 0; j < blockSize; j++)
58              {
59                  if (i + j < plainText.Length)
60                  {
61                      block = (block << 8) + plainText[i + j];
62                  }
63                  else
64                  {
65                      block <<= 8;
66                  }
67              }
68              cipherText[i / blockSize] = EncryptBlock(block);
69          }
70
71          byte[] result = new byte[cipherText.Length * blockSize];
72          for (int i = 0; i < cipherText.Length; i++)
73          {
74              uint block = cipherText[i];
75              for (int j = blockSize - 1; j >= 0; j--)
76              {
77                  result[i * blockSize + j] = (byte)(block & 0xFF);
78                  block >>= 8;
79              }
80          }
81
82          return result;
83      }
84
85
86
87      private uint EncryptBlock(uint block)
88      {
89          uint A = block + S[0];
90          uint B = block + S[1];
91          for (int i = 1; i <= Rounds; i++)
92          {
93              A = RotateLeft(A ^ B, (int)B) + S[2 * i];
94              B = RotateLeft(B ^ A, (int)A) + S[2 * i + 1];
95          }
96          return A;
97      }
```

Şək. 2.2 Standart RC6 şifrələmə prosesi



Şək. 2.3 Standart RC6 şifrələmə prosesi zamanı alınan nəticə

2.1.3 Standart RC6 şifrəsinin açılması prosesi

Giriş:

- Şifrəli mətn (A, B, C, D)

- Şifrələmə dövrlərinin sayı r

- Açar cədvəli S[0.. 2r + 3] w-bit sözləri

Prosedur:

1. A = A - S[2*r + 2] -> A, S[2*r + 2] ilə çıxarılır.
2. C = C - S[2*r + 3] -> C, S[2*r + 3] ilə çıxarılır.
3. i = r-dən 1-ə qədər -> r-dən 1-ə qədər i-nin dəyəri dəyişdirilir:

- (A, B, C, D) = (D, A, B, C) -> A, B, C, D registrləri dördlüsü yer dəyişir
  t və u dəyişənləri hesablanır.
- t = (B * (2 * B + 1)) << log2(w)

- u = (D * (2 * D + 1)) << log2(w)

- C = ((C - (S[2*i+1] >> t)) xor u) -> C, S[2*i+1]

əməliyyatı tətbiq edildikdən sonra sağa t dəyəri ilə shift olunur və u ilə xorlanır.

A = ((A - (S[2*i] >> u)) xor t) -> A, S[2*i]

əməliyyatı tətbiq edildikdən sonra sağa u dəyəri ilə shift olunur və t ilə xorlanır.

4. D = D - S[1] -> D, S[1] ilə çıxarılır.

5. B = B - S[0] -> B, S[0] ilə çıxarılır.

Çıxış:

- Mənbə mətn (A, B, C, D)

Şifrə açma alqoritminin C# proqramlaşdırma dilində proqram təminatının tətbiqi:

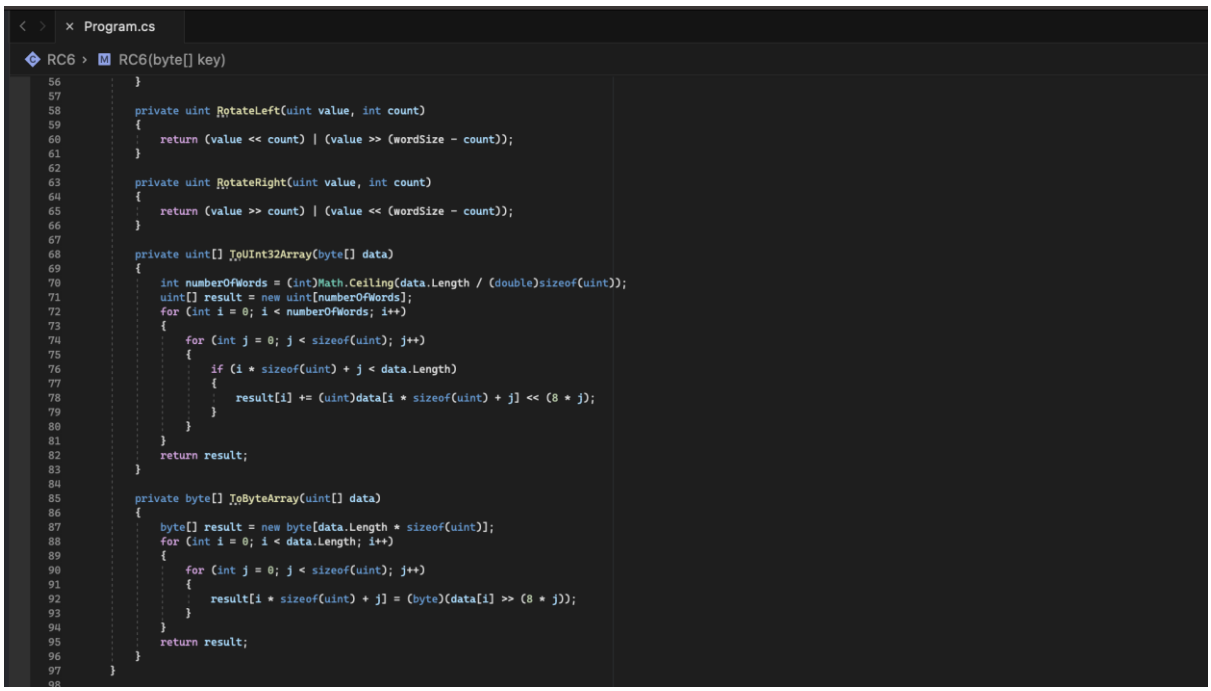
Program.cs

RC6 › RC6(byte[] key)

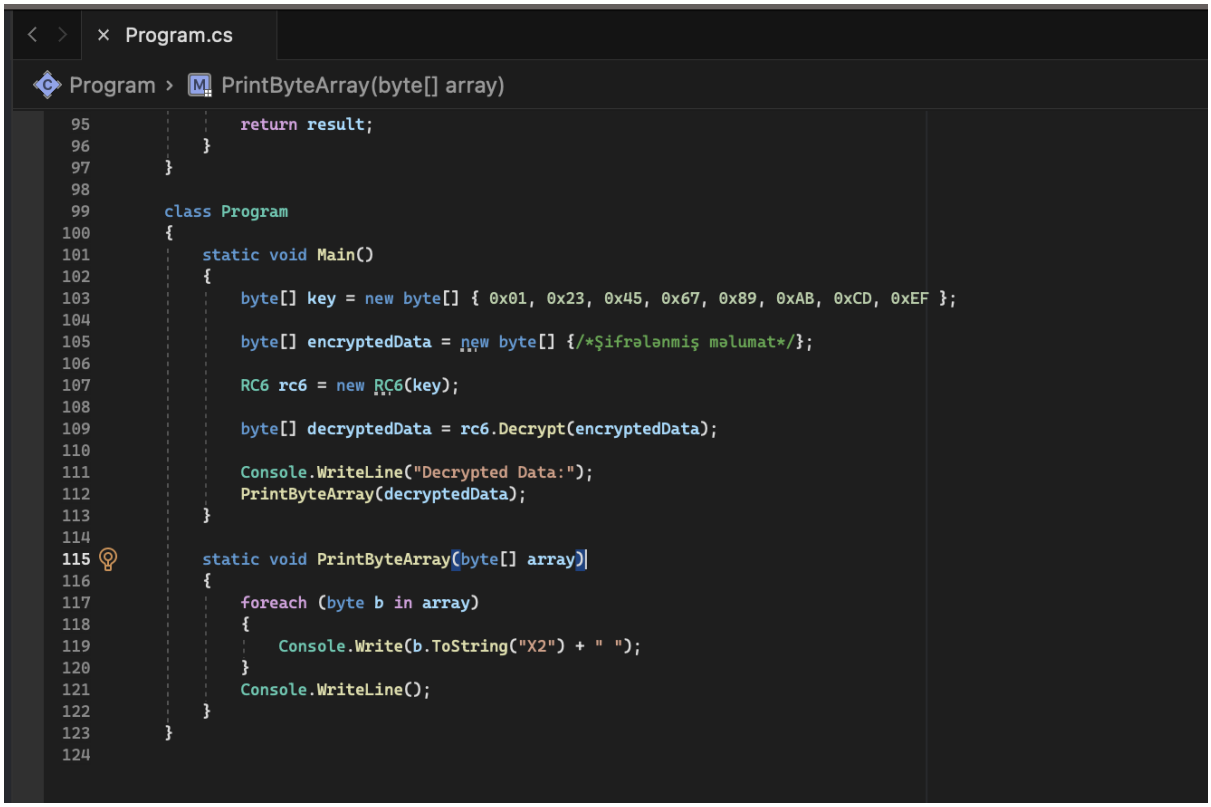
```
using System;

public class RC6
{
    private uint[] S;
    private const int wordSize = 32;
    private const int numberOfRounds = 20;
    private const uint P32 = 0xB7E15163;
    private const uint Q32 = 0x9E3779B9;

    public RC6(byte[] key)
    {
        int c = (int)Math.Ceiling(key.Length / (double)sizeof(uint));
        uint[] L = new uint[c];
        for (int i = key.Length - 1; i >= 0; i--)
        {
            L[i / 4] = (L[i / 4] << 8) + key[i];
        }

        S = new uint[2 * numberOfRounds + 4];
        S[0] = P32;
        for (int i = 1; i < S.Length; i++)
        {
            S[i] = S[i - 1] + Q32;
        }

        int maxIndex = (c > S.Length) ? (3 * c) : (3 * S.Length);
        uint A = 0;
        uint B = 0;


        for (int counter = 0; counter < maxIndex; counter++)
        {
            int i = 0;
            int j = 0;
            A = S[i] = RotateLeft(S[i] + A + B, 3);
            B = L[j] = RotateLeft(L[j] + A + B, (int)(A + B));
            i = (i + 1) % S.Length;
            j = (j + 1) % c;
        }
    }

    public byte[] Decrypt(byte[] encryptedData)
    {
        uint[] temp = ToUInt32Array(encryptedData);
        uint A = temp[0];
        uint B = temp[1];
        for (int i = 2 * numberOfRounds + 3; i >= 2; i--)
        {
            B = RotateRight(B - S[i], (int)A);
            A = RotateRight(A - S[i - 1], (int)B);
        }
        B -= S[1];
        A -= S[0];
        return ToByteArray(new uint[] { A, B });
    }
```

Ln 34, Col 23 Spaces LF

Şək. 2.4 Standart RC6 şifrəsinin açılması prosesi



Şək. 2.5 Standart RC6 şifrəsinin açılması prosesi



Şək. 2.6 Standart RC6 şifrəsinin açılması prosesi

2.2 RC6 Şifrələmə Alqoritminin Modifikasiya Olunmuş Versiyası

RC6 şifrələmə alqoritminin dəyişdirilmiş versiyasının dizaynı aşağıdakı addımları yerinə yetirməklə həyata keçirilə bilər:

1. Standart RC6 alqoritminin təhlili:

Standart RC6 alqoritmini ətraflı təhlil edin və alqoritmin iş prinsiplərini anlayın. RC6-nın təhlükəsizlik xüsusiyyətləri, istifadə olunan açar uzunluğu (512 və 1024 bit) və kriptoanaliz metodlarına davamlı olub-olmaması kimi amilləri qiymətləndirin.

2. Dəyişiklik hədəflərinin təyin edilməsi:

Dəyişdirilmiş alqoritmin hansı məqsədlərə cavab verməli olduğunu müəyyənləşdirin. Məsələn, daha yüksək təhlükəsizlik səviyyəsi, daha sürətli şifrələmə/şifrləmə vaxtları və ya daha çevik açar uzunluqları kimi məqsədlər ola bilər.

3. Struktur dəyişiklikləri:

Alqoritmin strukturunda ediləcək dəyişiklikləri planlaşdırın. Məsələn,
s-qutuları, döngələr, açar proqramlaşdırma kimi struktur elementləri dəyişdirərək alqoritmin performansını və ya təhlükəsizlik səviyyəsini yaxşılaşdıra bilərsiniz.

4. Yeni komponentlərin əlavə edilməsi:

Dəyişdirilmiş alqoritmə yeni komponentlər əlavə edin. Məsələn, əlavə açar təbəqələr, yeni transformasiyalar və ya müxtəlif şifrələmə rejimləri kimi komponentlərdən istifadə edilə bilər.

5. Təhlil və sınaq:

Dəyişdirilmiş alqoritmi təhlil edin və zəiflikləri aşkar etmək üçün müxtəlif kriptoanaliz üsullarını tətbiq edin. Həmçinin, alqoritmin performansını və məlumatların bütövlüyünü yoxlamaq üçün müxtəlif ssenarilər üzərində təcrübə aparın.

6. Optimallaşdırma:

Dəyişdirilmiş alqoritmin işini yaxşılaşdırmaq üçün optimallaşdırma üsullarını tətbiq edin. Məsələn, prosessor üçün xüsusi təlimatlar, paralelləşdirmə və ya yaddaşdan istifadənin optimallaşdırılması kimi üsullardan istifadə edilə bilər.

7. Kriptoqrafik qiymətləndirmə:

Dəyişdirilmiş alqoritmin təhlükəsizlik səviyyəsini qiymətləndirin. Güclü təsadüfilik, davamlı açar sahəsi, hücumlara davamlılıq kimi kriptoqrafik xüsusiyyətləri təhlil edin.

8. Standartlaşdırma və yayılma:

Dəyişdirilmiş alqoritmi standartlaşdırma orqanlarına təqdim etmək və ya onun istifadəsini genişləndirmək üçün lazımi addımları atın.

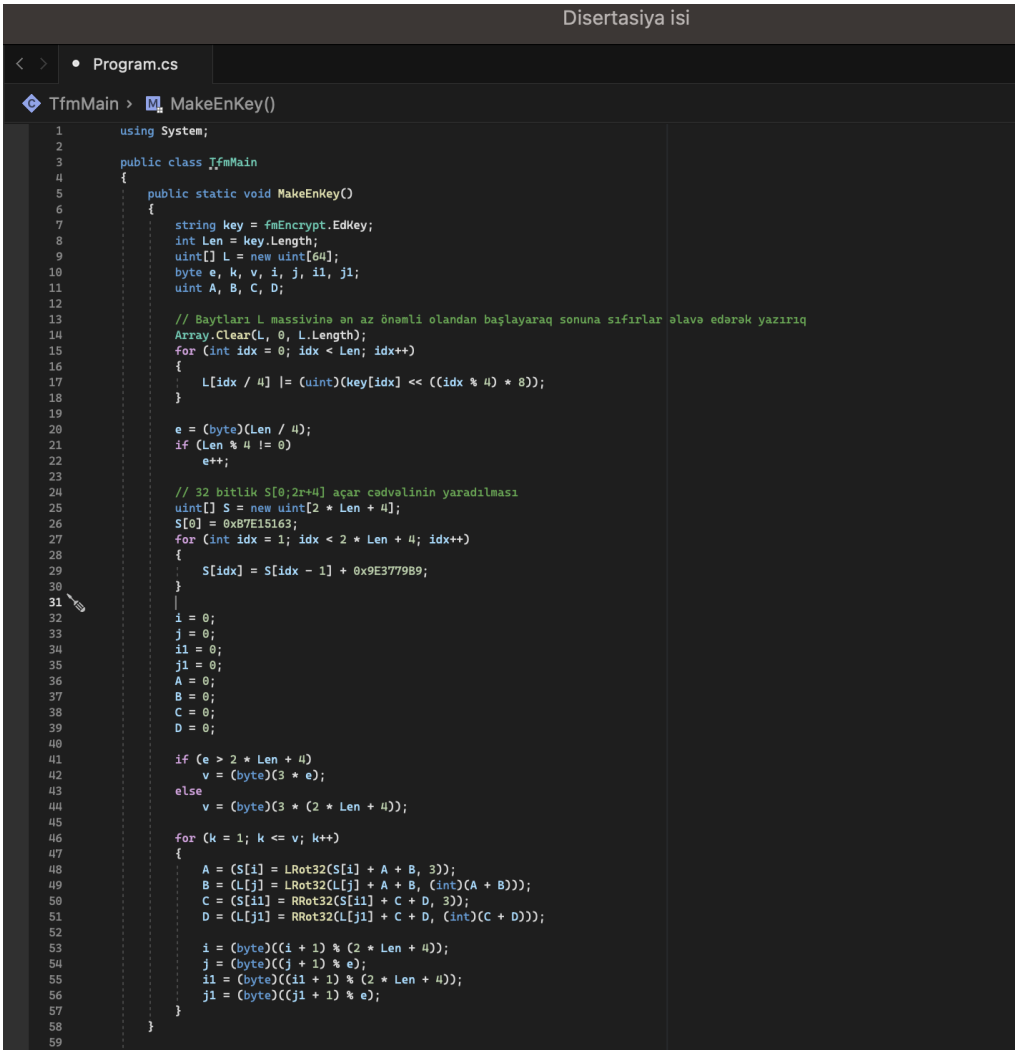
Qeyd: Dəyişdirilmiş şifrələmə alqoritmi hazırlanmışdır.

2.2.1 Modifikasiya olunmuş RC6 açarının yaradılması prosesi

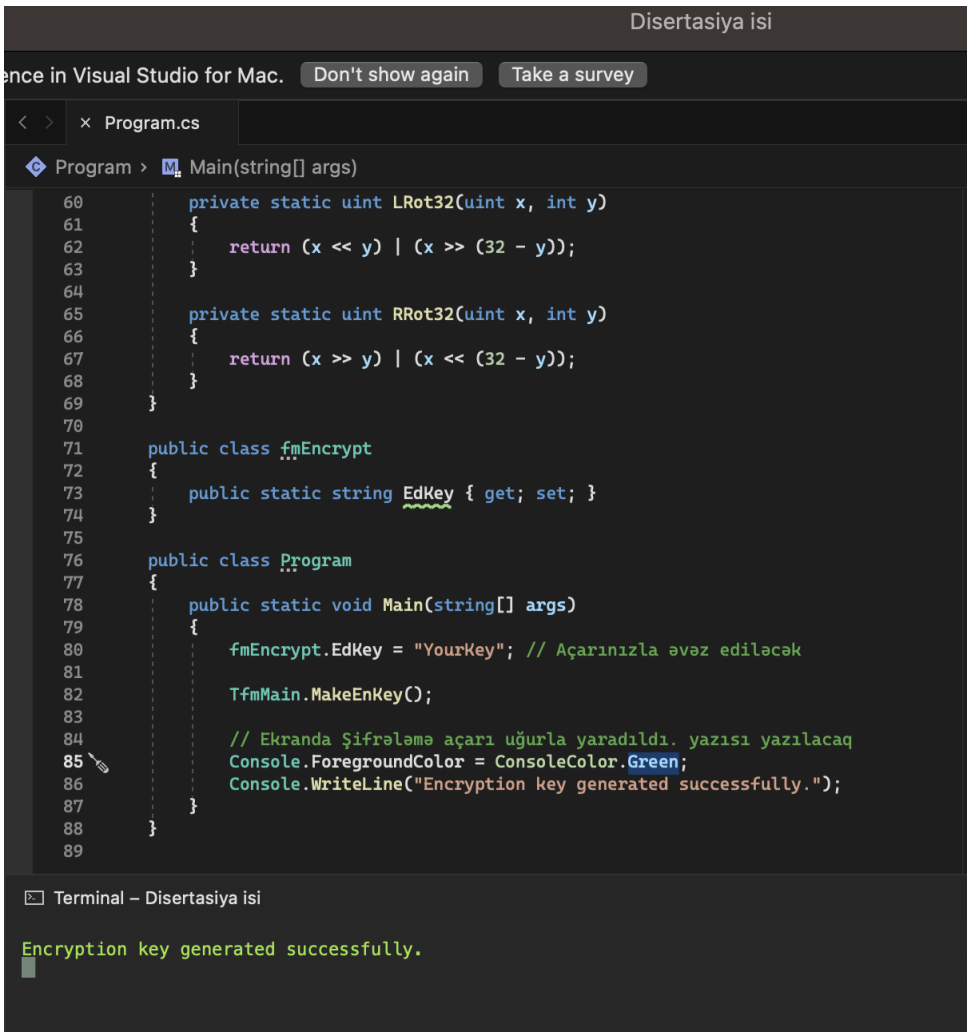
RC6 alqoritminin açar yaradılması proseduru aşağıdakı addımlardan ibarətdir:

1. Giriş kimi, istifadəçi tərəfindən müəyyən edilmiş b-bayt açarı və r hash dövrləri L[0..e-1] götürülür.
2. Çıxış S[0..2r+3] ardıcıllığında w-bit sözlərdən ibarət açar cədvəldir.
3. Birincisi, S[0] dəyəri P32 sabit qiyməti ilə müəyyən edilir.
4. Sonra i = 1-dən 2r+3-ə qədər bütün indekslər üçün S[i] qiyməti S[i-1] + Q32 kimi hesablanır.
5. Sonra v-nin qiyməti 3 * max(e, 2r + 4) kimi müəyyən edilir.
6. A, B, C, D, i, j, i1, j1 dəyişənləri ikirəqəmli ədədə (məsələn, 0) mənimsədilir.
7. Növbəti addımlar i = 1-dən v-ə qədər bütün indekslər üçün təkrarlanır:
   - A = S[i] = (S[i] + A + B) << 3
   - B = L[j] = (L[j] + A + B) << (AA + BB)
   - C = S[i1] = (S[i1] + C + D) >> 3
   - D = L[j1] = (L[j1] + C + D) >> (CC + DD)
   - i = (i + 1) mod (2r + 4)
   - j = (j + 1) mod e
   - i1 = (i1 + 1) mod (2r + 4)
   - j1 = (j1 + 1) mod e
8. Nəhayət, yaradılmış açar cədvəli S[0..2r+3] alınır.

Bu açarların hesablanması proseduru RC6 alqoritminin açar yaratma mərhələsində istifadə olunur. RC6 alqoritmi bu açar cədvəlindən istifadə edərək şifrələmə və deşifrəni həyata keçirir.



Şək. 2.7 Modifikasiya olunmuş RC6 açarının yaradılması prosesi



Şək. 2.8 Modifikasiya olunmuş RC6 açarının yaradılması prosesinin nəticəsi

2.2.2 Modifikasiya Olunmuş RC6 şifrələmə prosesi

RC6-w/r/b üçün şifrələmə (w bitlərdə sözün uzunluğu, r şifrələmə iterasiyalarının sıfırdan fərqli sayı, b baytlarda açar uzunluğu) aşağıdakı kimi təsvir edilir (dəyişikliklər qalın hərflərlə qeyd olunur):

Şifrələmə üçün mofifikasiya olunmuş RC6 alqoritmi

```
void Encryption(uint[] Buf, uint[] S, int r)
{
   uint A = Buf[0];
   uint B = Buf[1];
   uint C = Buf[2];
   uint D = Buf[3];
```

```
uint E = Buf[4];
uint F = Buf[5];

B += S[0];
D += S[1];
F += S[2];

for (int i = 1; i <= r; i++)
{
    uint t = LRot32(B * (2 * B + 1), 5);
    uint v = LRot32(D * (2 * D + 1), 5);
    uint w = LRot32(F * (2 * F + 1), 5);

    A = LRot32(A ^ t, w) + S[2 * i];
    C = LRot32(C ^ v, t) + S[2 * i + 1];
    E = LRot32(E ^ w, v) + S[2 * i + 2];

    uint temp = A;
    A = B;
    B = C;
    C = D;
    D = E;
    E = F;
    F = temp;
}

A += S[2 * r + 1];
C += S[2 * r + 2];
E += S[2 * r + 3];
```

```
    Buf[0] = A;
    Buf[1] = B;
    Buf[2] = C;
    Buf[3] = D;
    Buf[4] = E;
    Buf[5] = F;
}
```

Yuxarıdakı C# kodu dəyişdirilmiş RC6 şifrələmə prosedurunu təmsil edir. Buf parametri daxil edilən mətni və qeydləri ehtiva edən massivdir, S parametri əsas cədvəli təmsil edən massivdir, r parametri isə iterasiya dövrlərinin sayıdır. Şifrələmə əməliyyatı Buf massivində yerinə yetirilir və nəticə şifrəli mətn Buf massivində yerləşdirilir.

Mənbə faylı 192 bitlik (24 bayt) bloklara bölünür. Bu bloklar da öz növbəsində altı registrdən ibarətdir. Bloklar fayldan bir döngədə ardıcıl olaraq oxunur. Birinci blok oxunur və açarın ilk iki 32 bitlik sözü cüt registrlərə modul 2^w əlavə olunur. Cüt registrlər

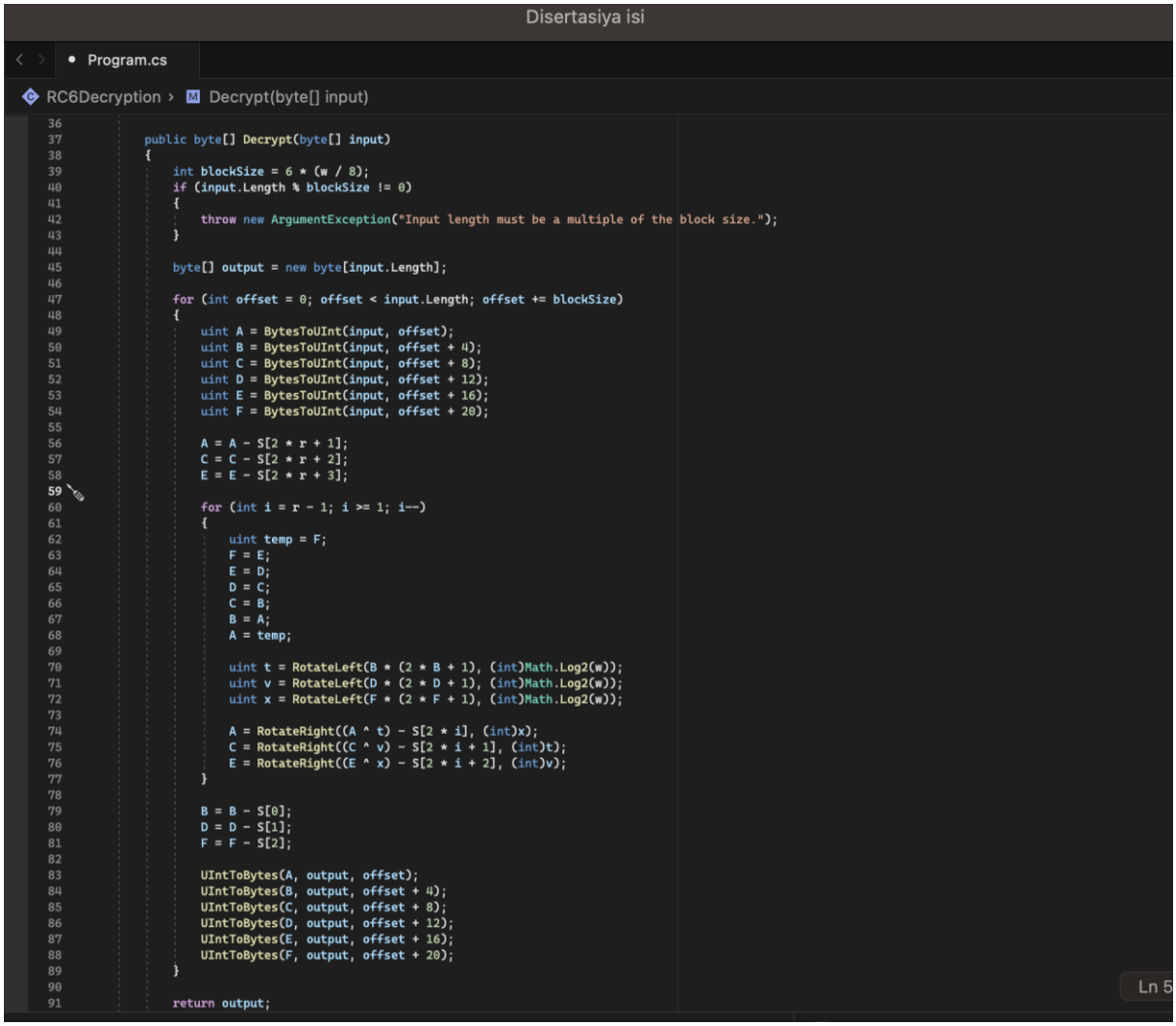
$$f(x)=x*(2*x+1)(\text{mod } 2^w)$$

çevirmə əməliyyatı və $\log_2 32 = 5$ bit ilə sola tsiklik yerdəyişmə ilə dövrələnir. Sonra, çevrilmiş registrlər və orijinal tək registrlər XOR laşdırılır və cüt bloklara çevrildikdən sonra saxlanılan bitlərin sayına görə sola fırlanır. Döngədə son əməliyyat modulu 2^w əlavə etməkdir (2*i)-th və (2*i+1)-th 32-bitlik açar sözlərlə. Dövr bitdikdən sonra açarın son iki 32 bitlik sözü tək registrlərə modul 2^w əlavə olunur . Hər iterasiyada cüt və tək registrləri dəyişdirən maşın sözü ilə yerdəyişmə həyata keçirilir və döngənin yeni iterasiyası başlayır.

2.2.3 Modifikasiya olunmuş RC6 şifrəsinin açılması prosesi

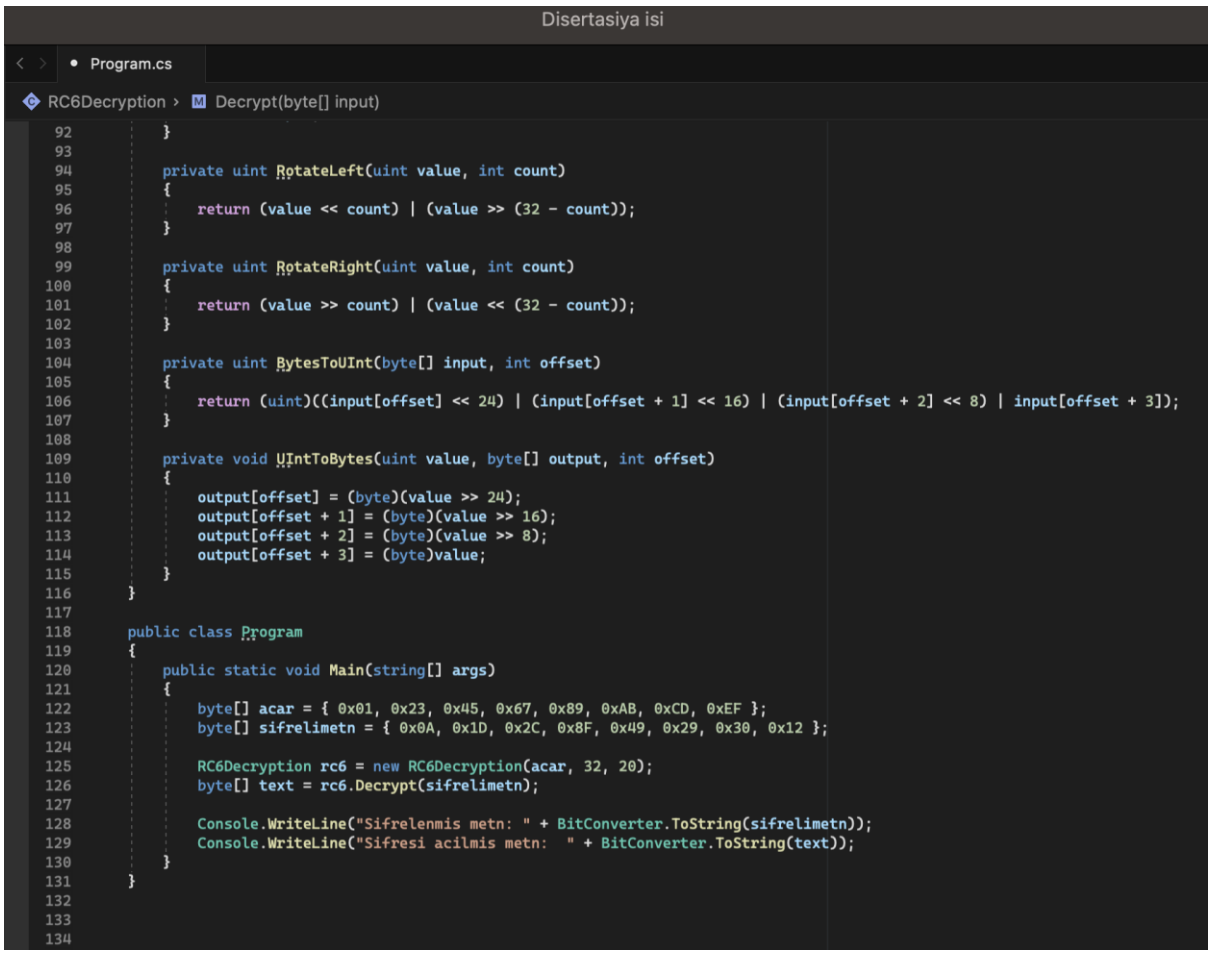
Bu proqram verilmiş açardan və şifrəli mətndən istifadə edərək dəyişdirilmiş RC6 alqoritmi ilə şifrəli mətnin şifrəsini açır. Şifrələmə proqramında, şifrələmə proqramında olduğu kimi, açar və şifrəli mətn bayt ardıcıllığı kimi müəyyən edilir.

Şifrə mətninin şifrəsini açmaq üçün RC6Decryption sinfindən istifadə edirik. Şifrələnmiş mətni Decrypt metoduna ötürməklə orijinal mətni əldə edirik.




```
36
37          public byte[] Decrypt(byte[] input)
38          {
39              int blockSize = 6 * (w / 8);
40              if (input.Length % blockSize != 0)
41              {
42                  throw new ArgumentException("Input length must be a multiple of the block size.");
43              }
44
45              byte[] output = new byte[input.Length];
46
47              for (int offset = 0; offset < input.Length; offset += blockSize)
48              {
49                  uint A = BytesToUInt(input, offset);
50                  uint B = BytesToUInt(input, offset + 4);
51                  uint C = BytesToUInt(input, offset + 8);
52                  uint D = BytesToUInt(input, offset + 12);
53                  uint E = BytesToUInt(input, offset + 16);
54                  uint F = BytesToUInt(input, offset + 20);
55
56                  A = A - S[2 * r + 1];
57                  C = C - S[2 * r + 2];
58                  E = E - S[2 * r + 3];
59
60                  for (int i = r - 1; i >= 1; i--)
61                  {
62                      uint temp = F;
63                      F = E;
64                      E = D;
65                      D = C;
66                      C = B;
67                      B = A;
68                      A = temp;
69
70                      uint t = RotateLeft(B * (2 * B + 1), (int)Math.Log2(w));
71                      uint v = RotateLeft(D * (2 * D + 1), (int)Math.Log2(w));
72                      uint x = RotateLeft(F * (2 * F + 1), (int)Math.Log2(w));
73
74                      A = RotateRight((A ^ t) - S[2 * i], (int)x);
75                      C = RotateRight((C ^ v) - S[2 * i + 1], (int)t);
76                      E = RotateRight((E ^ x) - S[2 * i + 2], (int)v);
77                  }
78
79                  B = B - S[0];
80                  D = D - S[1];
81                  F = F - S[2];
82
83                  UIntToBytes(A, output, offset);
84                  UIntToBytes(B, output, offset + 4);
85                  UIntToBytes(C, output, offset + 8);
86                  UIntToBytes(D, output, offset + 12);
87                  UIntToBytes(E, output, offset + 16);
88                  UIntToBytes(F, output, offset + 20);
89              }
90
91              return output;
```

Şək. 2.9 Modifikasiya olunmuş RC6 şifrəsinin açılması prosesi



Şək. 2.10 Modifikasiya olunmuş RC6 şifrəsinin açılması prosesi

Bu çıxış onu göstərir ki, verilmiş şifrəli mətn dəyişdirilmiş RC6 alqoritmi ilə uğurla deşifrə edilib. Birinci sətir şifrəli mətni, ikinci sətir isə orijinal mətni təmsil edir.

Çıxış

Sifrelenmis metn: 0A - 1D - 2C - 8F - 49 - 29 - 30 - 12

Sifresi acilmis metn: 12 - 34 - 56 - 78 - 9A - BC - DE - F0

III FƏSİL Modifikasiya edilmiş RC6'nın kriptoqrafik gücünün praktiki tətbiqi və öyrənilməsi

3.1 RC6 əsasında FileCoder Şifrələmə Proqramının blok sxemləri

Bu, modifikasiya edilmiş RC6-nın kripto-stabilliyindən istifadə etmək və faylların şifrələnməsini izah etmək və parolu müəyyən etmək üçün hər iki proses üçün mövcud sxemlərin hərtərəfli axtarışıdır. Hər bir blok diaqramın təsviri əlavə edilmişdir. Bütün blok diaqramlarda standart və ya modifikasiya edilmiş RC6 axtarmaq məqsədə uyğun olardı. Blok diaqramların təsvirində hesablanır ki, modifikasiya edilmiş RC6 götürülür, hər bir blokun ölçüsü 24 bayt hesab edilsə də, 16 baytlıq standart RC6-da da qəbul edilə bilər. RC6 blok alqoritmi olduğundan, bloklardakı faylların şifrələnməsi və deşifrə edilməsi prosesi üçün çoxlu sayda texnologiya variantları mümkündür. Bu işdə sifrələnmə/deşifrələnmə prosesinin texnologiyasının iki variantı verilə bilər. Proqramda birləşmə/deşifrələnmə prosesinin birinci variantından istifadə edilmişdir.

3.1.1 Fayl şifrələmə prosesi

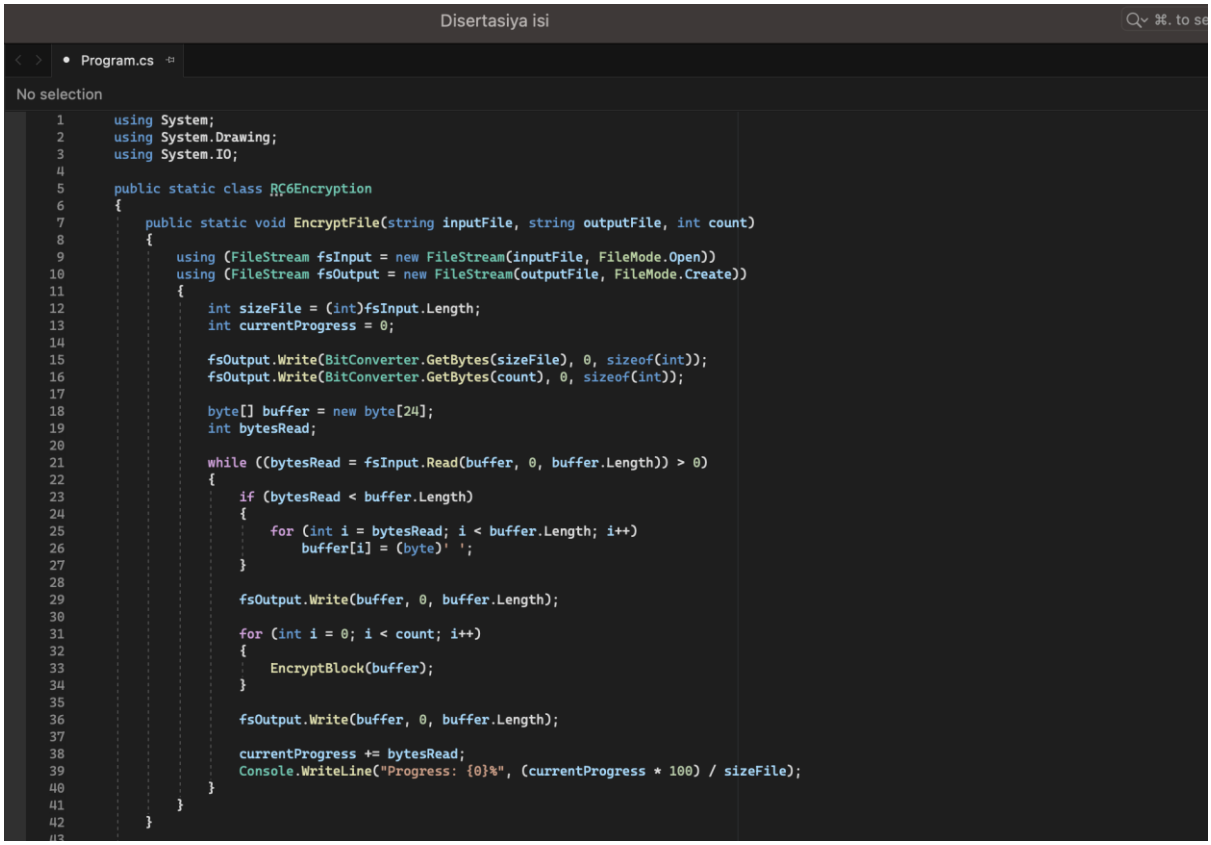
Əvvəlcə, nəticədəki faylda şifrələnməmiş əsas faylın ölçüsünü daxil edən bir dəyişəni yazırıq. Sonra, əsas faylın şifrələnmə dövrünün sayını içərən bir dəyişəni yazırıq. Dövrədə, əsas fayldan ilk 24 bayt oxuyuruq. Əgər bu ilk 24 baytsa, onda onları göstərilən sayda şifrələyib nəticədəki fayla yazırıq. Daha sonra bir daha ilk 24 baytları şifrələyib nəticədəki fayla yazırıq. Növbəti 24 baytı oxuyuruq. Əgər bu ilk 24 bayt deyilsə, onları göstərilən sayda şifrələyib nəticədəki fayla yazırıq. Və beləliklə, əsas fayldakı bütün blokları oxuyana qədər davam edirik. Son bloku oxuyarkən, onun sonuna sıfırları əlavə edirik, əgər blok 24 baytdan azdırsa, və onu da göstərilən sayda şifrələyib nəticədəki fayla yazırıq.

Beləliklə, nəticədəki şifrələnmiş fayl əvvəlcədən 24 baytlıq iki blokla başa çatır. İlk blokda, şifrələnməmiş əsas faylın ölçüsünü daxil edən bir dəyişən yer alır. İkinci blokda, əsas faylın şifrələnmə dövrünün sayı yer alır. Üçüncü blok, 24 bayt ölçüsündədir və göstərilən sayda şifrələnir, dördüncü blok isə üçüncü ilə eyni sayda

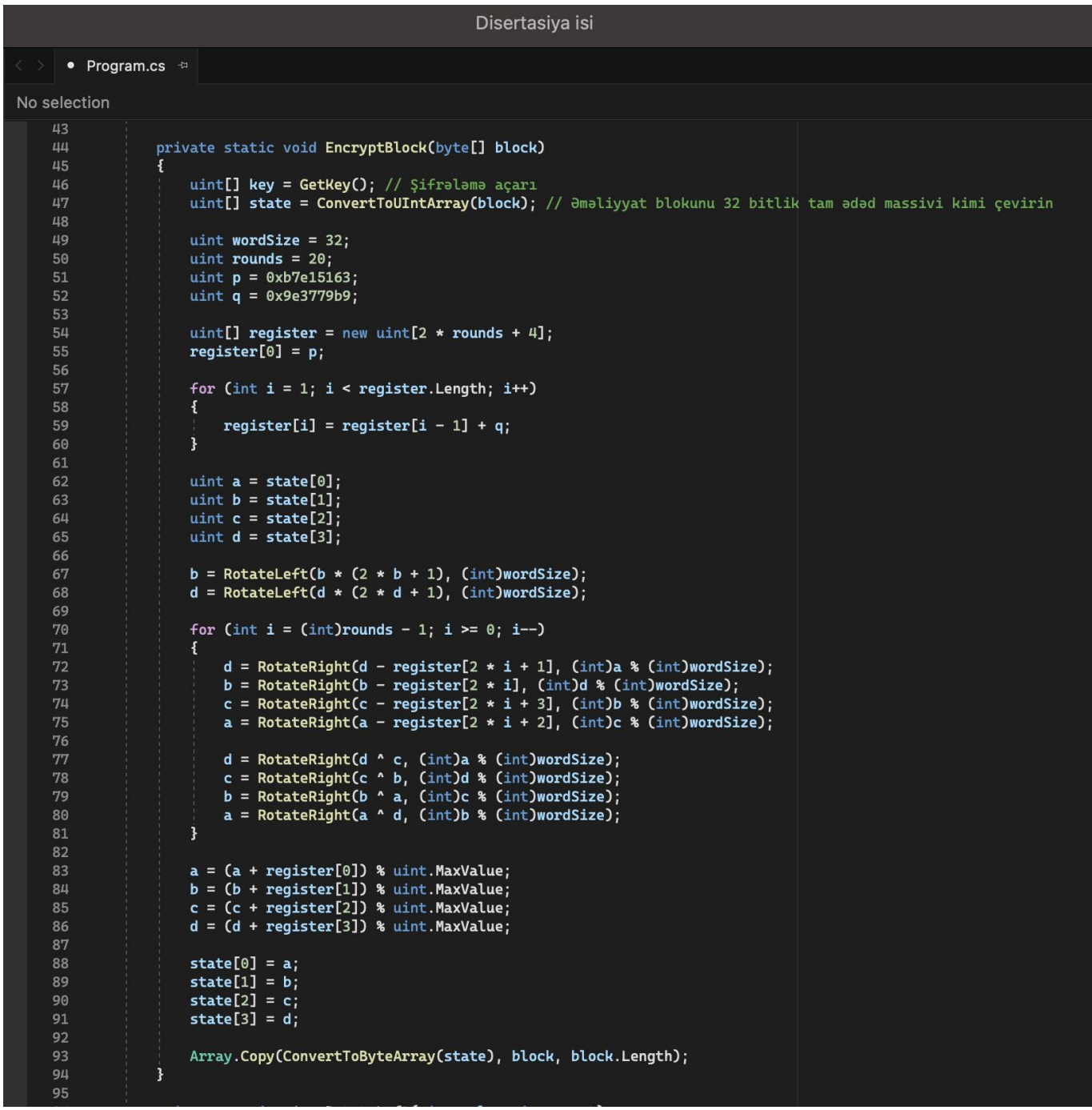
şifrələnir, bir əlavə şifrələnmə ilə. Üçüncü və dördüncü bloklar şifrəni açma zamanı daxil edilən düzgün parolun düzgünlüyünü yoxlamaq üçün istifadə olunur və həmçinin parolun təxminində istifadə olunur. Daha sonra gələn bütün bloklar göstərilən sayda şifrələnir.

Nəticədə, son şifrələnmiş faylın ölçüsü şifrələnməmiş əsas fayldan 32 bayt çoxdur.

Şifrələnmə prosesinin proqram təminatı:C# proqramlaşdırma dilində

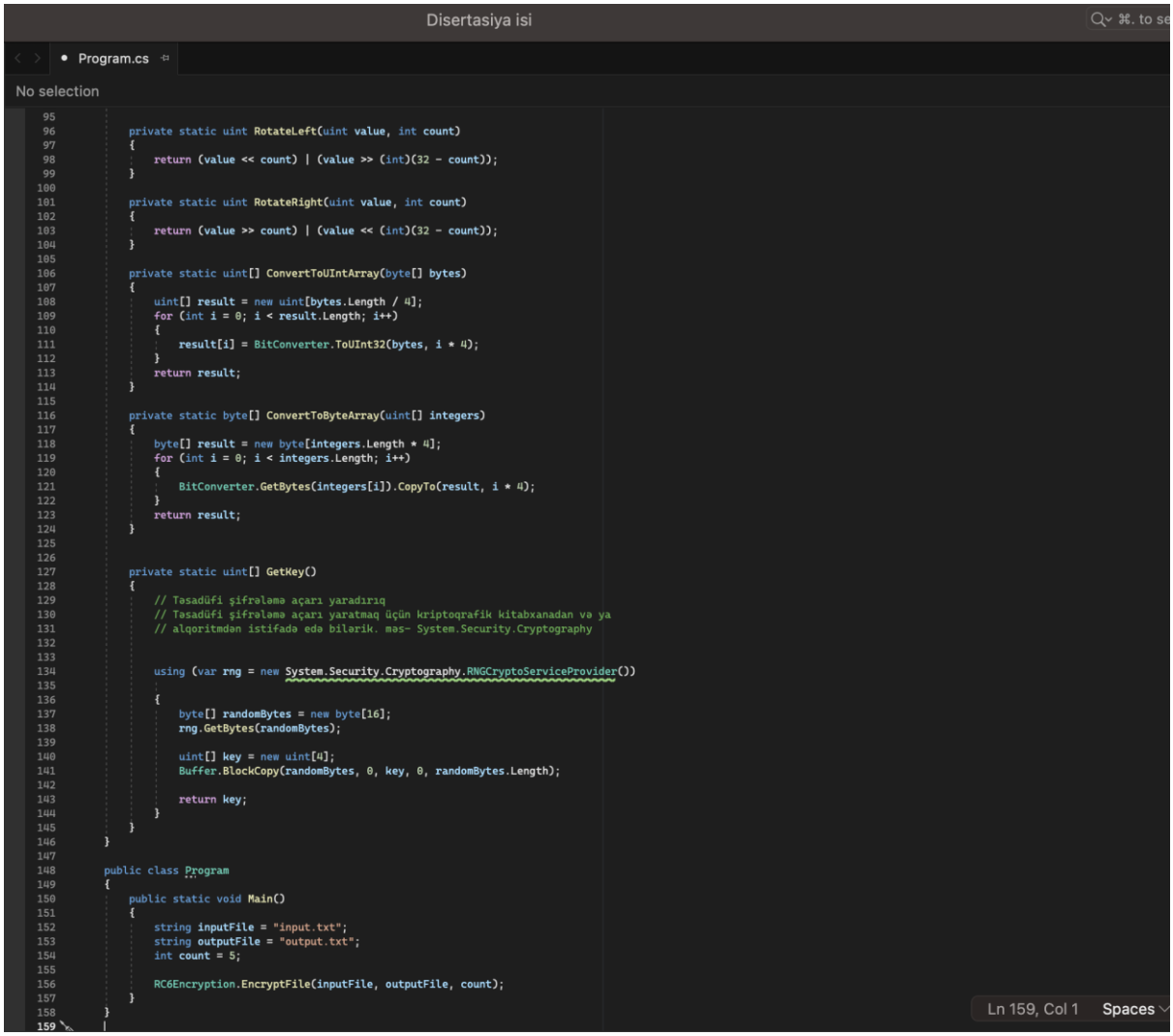


Şək. 3.1 Fayl şifrələmə prosesi




```
43
44      private static void EncryptBlock(byte[] block)
45      {
46          uint[] key = GetKey(); // Şifrələmə açarı
47          uint[] state = ConvertToUIntArray(block); // Əməliyyat blokunu 32 bitlik tam ədəd massivi kimi çevirin
48
49          uint wordSize = 32;
50          uint rounds = 20;
51          uint p = 0xb7e15163;
52          uint q = 0x9e3779b9;
53
54          uint[] register = new uint[2 * rounds + 4];
55          register[0] = p;
56
57          for (int i = 1; i < register.Length; i++)
58          {
59              register[i] = register[i - 1] + q;
60          }
61
62          uint a = state[0];
63          uint b = state[1];
64          uint c = state[2];
65          uint d = state[3];
66
67          b = RotateLeft(b * (2 * b + 1), (int)wordSize);
68          d = RotateLeft(d * (2 * d + 1), (int)wordSize);
69
70          for (int i = (int)rounds - 1; i >= 0; i--)
71          {
72              d = RotateRight(d - register[2 * i + 1], (int)a % (int)wordSize);
73              b = RotateRight(b - register[2 * i], (int)d % (int)wordSize);
74              c = RotateRight(c - register[2 * i + 3], (int)b % (int)wordSize);
75              a = RotateRight(a - register[2 * i + 2], (int)c % (int)wordSize);
76
77              d = RotateRight(d ^ c, (int)a % (int)wordSize);
78              c = RotateRight(c ^ b, (int)d % (int)wordSize);
79              b = RotateRight(b ^ a, (int)c % (int)wordSize);
80              a = RotateRight(a ^ d, (int)b % (int)wordSize);
81          }
82
83          a = (a + register[0]) % uint.MaxValue;
84          b = (b + register[1]) % uint.MaxValue;
85          c = (c + register[2]) % uint.MaxValue;
86          d = (d + register[3]) % uint.MaxValue;
87
88          state[0] = a;
89          state[1] = b;
90          state[2] = c;
91          state[3] = d;
92
93          Array.Copy(ConvertToByteArray(state), block, block.Length);
94      }
95
```

Şək. 3.2 Fayl şifrələmə prosesi




```
95
96          private static uint RotateLeft(uint value, int count)
97          {
98              return (value << count) | (value >> (int)(32 - count));
99          }
100
101         private static uint RotateRight(uint value, int count)
102         {
103             return (value >> count) | (value << (int)(32 - count));
104         }
105
106         private static uint[] ConvertToUIntArray(byte[] bytes)
107         {
108             uint[] result = new uint[bytes.Length / 4];
109             for (int i = 0; i < result.Length; i++)
110             {
111                 result[i] = BitConverter.ToUInt32(bytes, i * 4);
112             }
113             return result;
114         }
115
116         private static byte[] ConvertToByteArray(uint[] integers)
117         {
118             byte[] result = new byte[integers.Length * 4];
119             for (int i = 0; i < integers.Length; i++)
120             {
121                 BitConverter.GetBytes(integers[i]).CopyTo(result, i * 4);
122             }
123             return result;
124         }
125
126
127         private static uint[] GetKey()
128         {
129             // Təsadüfi şifrələmə açarı yaradırıq
130             // Təsadüfi şifrələmə açarı yaratmaq üçün kriptoqrafik kitabxanadan və ya
131             // alqoritmdən istifadə edə bilərik. məs- System.Security.Cryptography
132
133
134             using (var rng = new System.Security.Cryptography.RNGCryptoServiceProvider())
135
136             {
137                 byte[] randomBytes = new byte[16];
138                 rng.GetBytes(randomBytes);
139
140                 uint[] key = new uint[4];
141                 Buffer.BlockCopy(randomBytes, 0, key, 0, randomBytes.Length);
142
143                 return key;
144             }
145         }
146     }
147
148     public class Program
149     {
150         public static void Main()
151         {
152             string inputFile = "input.txt";
153             string outputFile = "output.txt";
154             int count = 5;
155
156             RC6Encryption.EncryptFile(inputFile, outputFile, count);
157         }
158     }
159
```

Şək. 3.3 Fayl şifrələmə prosesi

Not: private static uint[] GetKey() metodu üçün

Bu üsul System.Security.Cryptography.RNGCryptoServiceProvider sinfindən istifadə edərək təsadüfi açar yaradır. Müvafiq sinif təhlükəsiz təsadüfi ədəd generatorunu təmin edir. RandomBytes adlı 16 baytlıq massiv yaradır və bu massivə təsadüfi qiymətlər təyin edir. Sonra Buffer.BlockCopy metodundan istifadə edərək, bu bayt massivi açar adlı 4 uint dəyərdən ibarət massivə kopyalanır və açar kimi qaytarılır.

3.1.2 Faylın şifrəsinin açılması prosesi

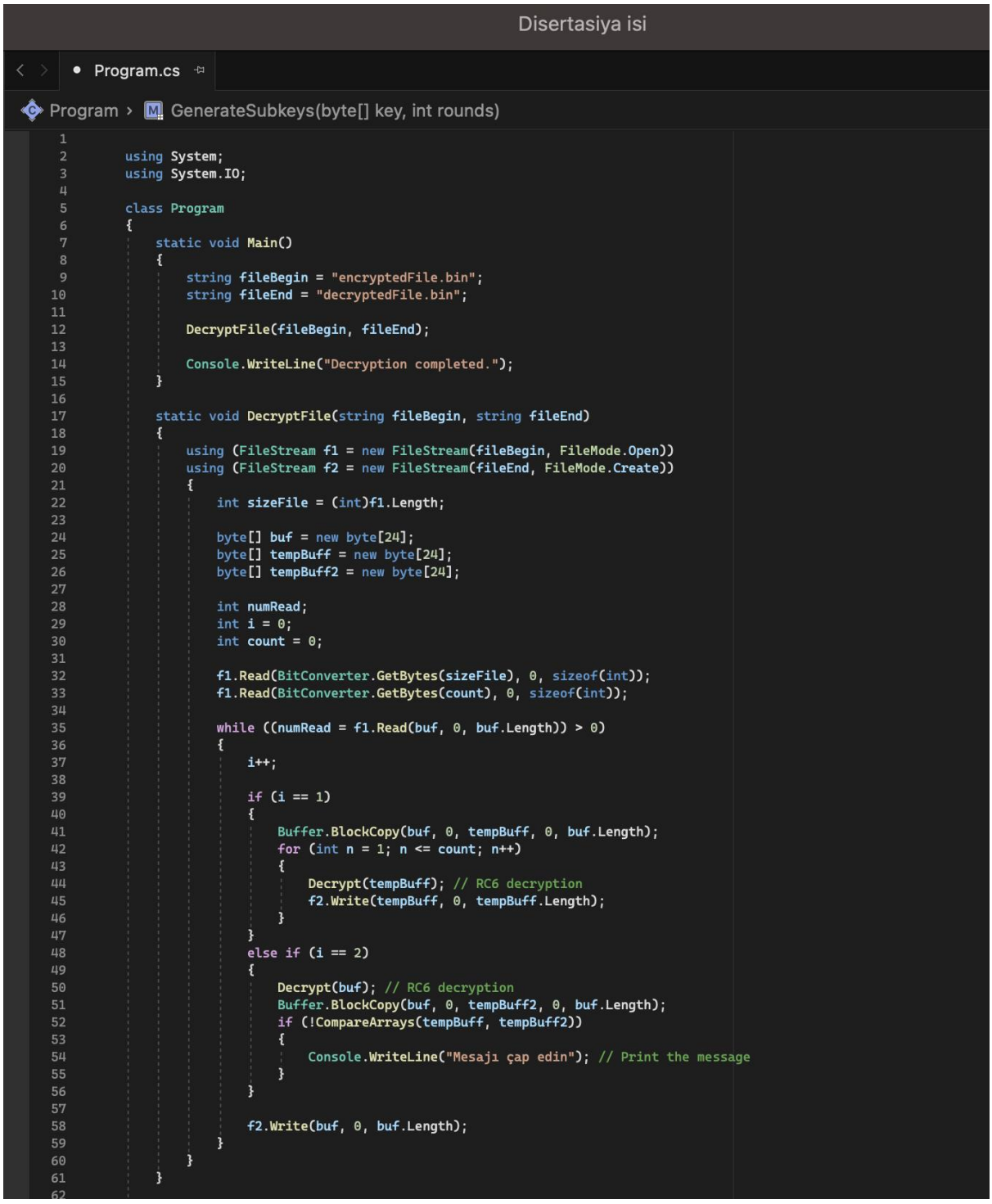
Şifrənin açılması prosesinin blok diaqramının təsviri aşağıdakı kimidir:

1. İlk olaraq şifrələnmiş giriş faylının ilk 4 baytını oxuyaraq və bir dəyişəndə saxlayaraq orijinal açılmamış faylın ölçüsünü əldə edirik.

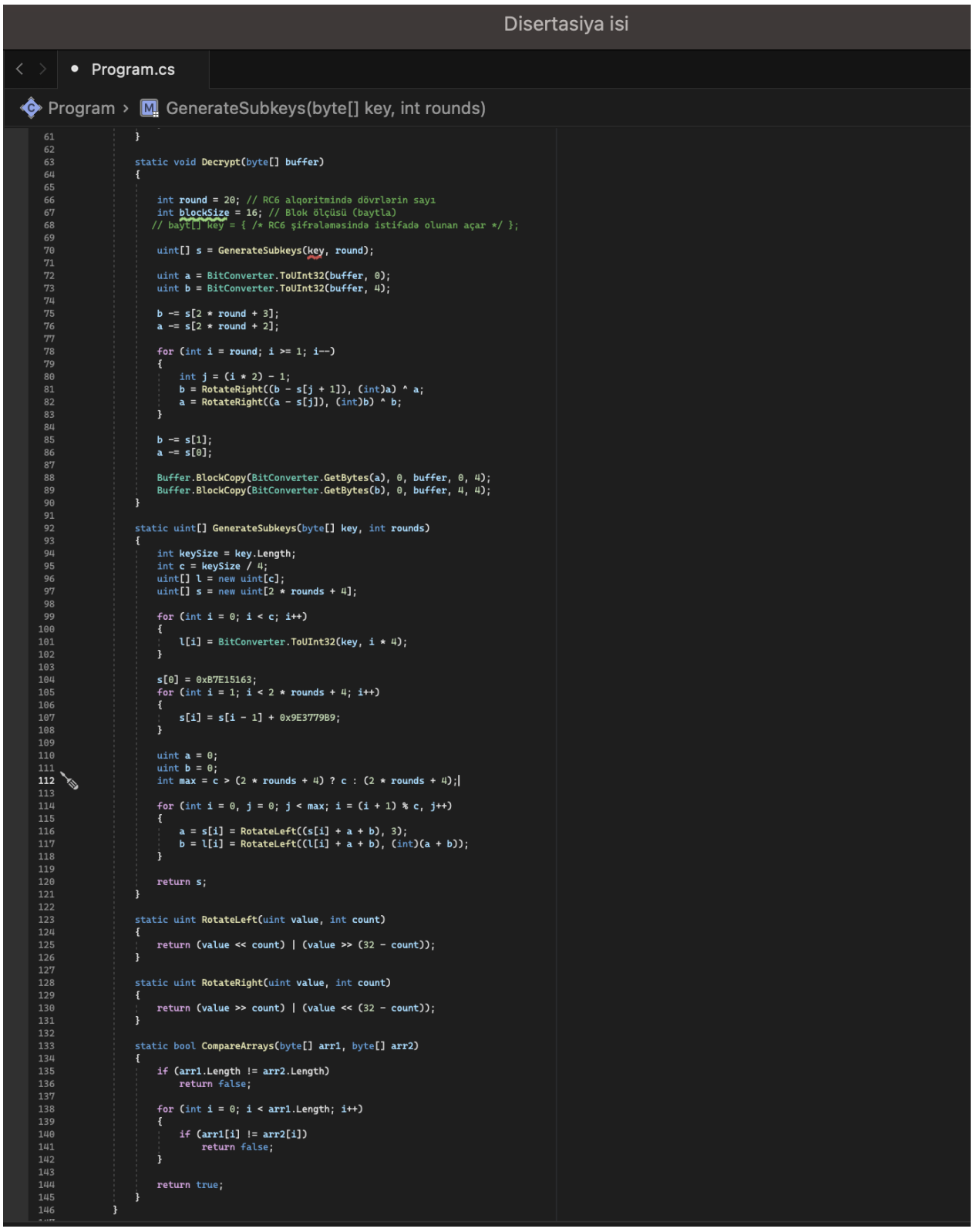
2. Sonra, növbəti 4 baytı oxuyaraq və bir dəyişəndə saxlayaraq orijinal açılmamış faylın şifrələnmə dövrələrinin sayını əldə edirik.
3. Növbəti addımda, bir dövrədə 24 baytı orijinal fayldan oxuyuruq. Əgər bu ilk 24 bayt isə, onları birinci müvəqqəti dəyişəndə saxlayırıq.
4. İlk 24 baytı verilmiş sayda şifrələnmə dövrəsi ilə açırıq və onları son fayla yazırıq. Növbəti 24 baytı oxuyuruq.
5. Əgər bu ikinci 24 baytsa, onları ikinci müvəqqəti dəyişəndə saxlayırıq.
6. Birinci müvəqqəti dəyişəndən əldə edilən ilk 24 baytı ikinci müvəqqəti dəyişəndən əldə edilən ikinci 24 baytla müqayisə edirik. Əgər eyni deyilsə, istifadəçiyə yanlış şifrə daxil etdiyini bildirən bir mesaj çıxırırıq.
7. Növbəti 24 baytı oxuyuruq. Əgər bu neçənci 24 baytsa (neçənci olmadığını yoxlamaq üçün), onları verilmiş sayda şifrələnmə dövrəsi ilə açırıq və onları son fayla yazırıq.
8. Sonuncu mərhələdə, açılmış son fayldan boşluqları silərək (trim edərək) lazımsız baytları silirik, beləliklə onun ölçüsü orijinal açılmamış faylın ölçüsünə bərabər olur.

Nəticədə əldə edilən son açılmış fayl, əvvəlki versiyonlarda olan açılmamış orijinal faylın 32 baytı, açılmış faylın ölçü və şifrələnmə dövrələrinin sayını yoxlamaq üçün nəzərdə tutulan nəzarət bloklarını təşkil edir.

Şifrənin açılması prosesinin proqram təminatı C# proqramlaşdırma dilində



Şək. 3.4 Faylın şifrəsinin açılması prosesi



Şək. 3.5 Faylın şifrəsinin açılması prosesi

3.1.3 Parolun təxmin edilməsi prosesi

Disertasiya isi

Program.cs

No selection

```
//using System;
//using System.IO;

//class Program
//{
//    static void Main()
//    {
//        FindPassword();
//    }

//    static void FindPassword()
//    {
//        uint[] L = new uint[64];
//        byte e, k, v, i, j, i1, j1;
//        uint A, B, C, D;
//        string CurrentKey;
//        FileStream CurFile;

//        CurrentKey = "";

//        // Başlama vaxtını yadda saxla

//        CurFile = new FileStream(FileBegin, FileMode.Open, FileAccess.Read);
//        CurFile.Seek(8, SeekOrigin.Begin);

//        byte[] Temp_Buff = new byte[24];


//        // Fayldan ilk 24 baytı Temp_Buff-a oxuyun

//        CurFile.Read(Temp_Buff, 0, Temp_Buff.Length);

//        // Göstəricini ikinci 24 bayta köçürün

//        CurFile.Seek(32, SeekOrigin.Begin);

//        byte[] Temp_Buff2 = new byte[24];

//        // Fayldan ikinci 24 baytı Temp_Buff2-ə oxuyun

//        CurFile.Read(Temp_Buff2, 0, Temp_Buff2.Length);

//        CurFile.Close();

//        // şifrə yoxlanması
//        while (true)
//        {
//            if (fmPassword.rbNumber.Checked)
//                CurrentKey = DigitalPassword(CurrentKey);

//            if (fmPassword.rbWord.Checked)
//                CurrentKey = WordPassword(CurrentKey);

//            if (fmPassword.rbWordsDigits.Checked)
//                CurrentKey = WordsDigits(CurrentKey);

//            if (fmPassword.rbSpisok.Checked)
//                CurrentKey = ListPassword(CurrentKey, fmPassword.edList.Text);

//            if (fmPassword.rbAll.Checked)
//                CurrentKey = AllPassword(CurrentKey);

//            if (fmPassword.cbShowPass.Checked)
//                fmPassword.lbCurrPass.Caption = CurrentKey;

//            // Deşifrə açarı yaradın

//            string Açar = CurrentKey;
//            int Len = Açar.Length;

//            // əsas dəyişənləri yaradın

//            Array.Clear(L, 0, L.Length);
//            Köçürün(Açar, L, Len + 1);
//            e = (byte)(Len / 4);
//            if (Len % 4 != 0)
//                e++;
```

Şək. 3.6 Parolun təxmin edilməsi

Şək. 3.7 Parolun təxmin edilməsi

Disertasiya isi

Program.cs

No selection

```
152         //    }
153         //
154         //    static string WordsDigits(string currentKey)
155         //    {
156         //        // Sözlər və rəqəmlər parol yaratma məntiqi
157         //        string generatedPassword = "";
158         //        string[] words = currentKey.Split(' ');
159         //        foreach (string word in words)
160         //        {
161         //            for (int i = 0; i < word.Length; i++)
162         //            {
163         //                if (char.IsLetterOrDigit(word[i]))
164         //                {
165         //                    generatedPassword += word[i];
166         //                }
167         //            }
168         //        }
169         //        return generatedPassword;
170         //    }
171         //
172         //    static string ListPassword(string currentKey, string listText)
173         //    {
174         //
175         //        string generatedPassword = "";
176         //        string[] listItems = listText.Split(',');
177         //        foreach (string item in listItems)
178         //        {
179         //            if (currentKey.Contains(item))
180         //            {
181         //                generatedPassword += item;
182         //            }
183         //        }
184         //        return generatedPassword;
185         //    }
186         //
187         //    static string AllPassword(string currentKey)
188         //    {
189         //        // Bütün parol yaratma məntiqi
190         //        string generatedPassword = currentKey.ToUpper();
191         //        return generatedPassword;
192         //    }
193         //
194         //    static uint LRot32(uint x, byte n)
195         //    {
196         //        return (x << n) | (x >> (32 - n));
197         //    }
198         //
199         //    static uint RRot32(uint x, byte n)
200         //    {
201         //        return (x >> n) | (x << (32 - n));
202         //    }
203         //
204         //    static void Köçürün(string str, uint[] arr, int len)
205         //    {
206         //        int j = 0;
207         //        for (int i = 0; i < len; i++)
208         //        {
209         //            arr[i] = (uint)((str[j] << 24) | (str[j + 1] << 16) | (str[j + 2] << 8) | str[j + 3]);
210         //            j += 4;
211         //        }
212         //    }
213         //
214         //    static void Decryption(byte[] buffer)
215         //    {
216         //
217         //
218         //        byte[] key = { } //açarı daxil edin;
219         //    int keySize = key.Length * 8; // açarın bit uzunluğu
220         //        int rounds = 20; // dövr sayı
221         //        int blockSize = 16; // Blok ölçüsü
222         //
223         //        // RC6 şifrələmə alqoritmindən istifadə edərək buferi açın
224         //
225         //        RC6Decrypt(buffer, key, keySize, rounds, blockSize);
226         //    }
227
```

Şək. 3.8 Parolun təxmin edilməsi

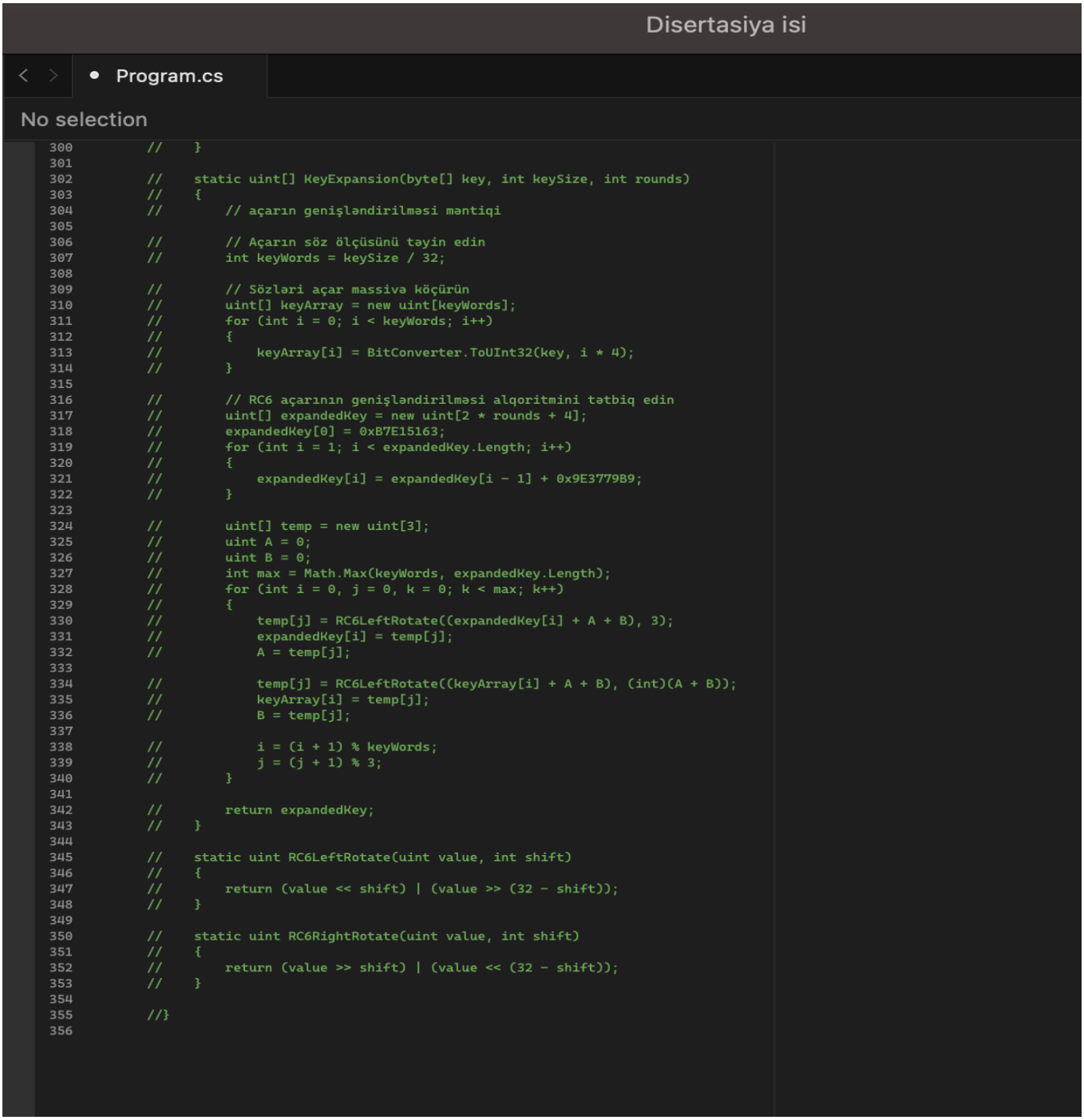
Disertasiya isi

Program.cs

No selection

```
227
228         //    static void RC6Decrypt(byte[] buffer, byte[] key, int keySize, int rounds, int blockSize)
229         //    {
230         //        // RC6 şifreleme algoritması məntiqi
231         //
232         //        // Açarı hazırla
233         //
234         //        uint[] keyWords = KeyExpansion(key, keySize, rounds);
235         //
236         //        // Şifrələmə modulu parametrlərini təyin edin
237         //
238         //        uint blockSizeWords = (uint)(blockSize / 4);
239         //        uint[] block = new uint[blockSizeWords];
240         //        uint[] tempBlock = new uint[blockSizeWords];
241         //
242         //        // Blok ölçüsünü yoxlayırıq
243         //
244         //        if (buffer.Length % blockSize != 0)
245         //        {
246         //            throw new ArgumentException("Invalid buffer size for RC6 decryption.");
247         //        }
248         //
249         //        // Hər bloku həll edin
250         //
251         //        for (int offset = 0; offset < buffer.Length; offset += blockSize)
252         //        {
253         //
254         //            Buffer.BlockCopy(buffer, offset, block, 0, blockSize);
255         //
256         //            // Təhlil üçün RC6 alqoritmini tətbiq edin
257         //            RC6DecryptBlock(block, tempBlock, keyWords, rounds);
258         //
259         //            // Şifrə açılmış bloku buferə yazın
260         //            Buffer.BlockCopy(tempBlock, 0, buffer, offset, blockSize);
261         //        }
262         //    }
263         //
264         //    static void RC6DecryptBlock(uint[] block, uint[] tempBlock, uint[] keyWords, int rounds)
265         //    {
266         //        // RC6 şifrələmə alqoritmi blokdan çıxarma məntiqi
267         //
268         //
269         //        uint A = block[0];
270         //        uint B = block[1];
271         //        uint C = block[2];
272         //        uint D = block[3];
273         //
274         //
275         //        for (int i = rounds; i > 0; i--)
276         //        {
277         //            uint tempD = D;
278         //            D = C;
279         //            C = B;
280         //            B = A;
281         //            A = RC6RightRotate((B * (2 * B + 1)), 5);
282         //            C = RC6RightRotate((C - keyWords[2 * i + 1]), (int)A);
283         //            A = RC6RightRotate((A - keyWords[2 * i]), (int)B);
284         //            D = RC6RightRotate((D - keyWords[2 * i + 1]), (int)C);
285         //            B = RC6RightRotate((B - keyWords[2 * i]), (int)D);
286         //        }
287         //
288         //
289         //        uint temp = A;
290         //        A = B;
291         //        B = C;
292         //        C = D;
293         //        D = temp;
294         //
295         //
296         //        tempBlock[0] = A;
297         //        tempBlock[1] = B;
298         //        tempBlock[2] = C;
299         //        tempBlock[3] = D;
300         //    }
```

Şək. 3.9 Parolun təxmin edilməsi



Şək. 3.10 Parolun təxmin edilməsi

NƏTİCƏ

Məlumatın təhlükəsizliyinin təmin edilməsinin ümumi prinsiplərin konkret tətbiq olunması, məlumatın qorunması üçün təşkilatçı və ya texniki tədbirləri ifadə edə bilər.Bunlardan biri də məlumatların etibarlı və təsdiqlənmiş verilənlər şifrələnməsi ilə təhlükəsizlik tədbirləri qrupuna daxildir. Kriptoqrafik şifrələmə vasitəsilə hər hansı məlumat şifrələnir və uyğun tətbiqlə açıla bilməz. Hər halda seçilmiş kriptoqrafiya metodları zərərçəkənlərə qarşı təhlükəsizlik təmin etmək üçün istifadənin asanlıq, dəyişmə, operativlik və qoruma kombinasiyasını birləşdirməlidir.

Görülən işlərin əsas nəticəsi mətn fayllarının şifrələnməsi proqramının yaradılmasıdır. Bu layihənin həyata keçirilməsi obyekt əsaslı mühitin güclü alətlərindən istifadə etməklə həyata keçirilib.Obyektyönümlü proqramlaşdırma dili olan C# vasitəsi ilə kodlanıb. Bu, istifadəçiyə maksimum diqqət yetirən proqram yaratmağa imkan verir. Hazırlanmış proqram məlumatları şifrələmək və deşifrə etmək üçün hər yerdə istifadə edilə bilər.

İSTİFADƏ EDİLMİŞ ƏDƏBİYYAT

1. Лось А. Б., Нестеренко А. Ю., Рожков М. И. “Криптографические методы защиты информации для изучающих компьютерную безопасность : учебник для вузов— 2-е изд., испр. — Москва : Издательство Юрайт, 2023. — 473 с.
2. Urbanovich P.P. The protection of information based on the methods by cryptography,steganography and obfuscation. Minsk,BGTU Publ.,2016.220 p.
3. Храмова Н.А. Исследование криптосистемы RSA для шифрования информации // Современные наукоемкие технологии. – 2020. – № 9. – С. 88-93.
4. А.А. Болотов, С.Б. Гашков, А.Б. Фролов, А.А. Часовских Элементарное введение в эллиптическую криптографию: алгебраические и алгоритмические основы. - М.: КомКнига, 2006. - 328 с.
5. Аграновский А.В., Хади Р.А. Практическая криптография: алгоритмы и их программирование. - М.: «Солон-Р», 2002.
6. Аграновский А.В., Балакин А.В., Хади Р.А. Классические шифры и методы их криптоанализа. - М: Информационные технологии, 2001.
7. Баричев С.Г., Серов Р.Е. Основы современной криптографии. - М.: "Горячая линия - Телеком", 2001.
8. Чмора А. Современная прикладная криптография. - М: "Гелиос АРБ", 2001 г.
9. Казарин О.В. Программная безопасность компьютерных систем. - М: МГЮЛ, 2003.
10. Мао, Вэньбо. Современная криптография: теория и практика.: Пер. с английского. - Издательство "Вильяме", 2005.
11. Петров А.А. Компьютерная безопасность. Методы криптографической защиты. - М: ДМК, 2000.
12. Рябко Б.Я., Фионов А.Н. Криптографические методы защиты информации, Горячая линия - Телеком, 2005, 229 стр.

13. Щербаков Л.Ю., Доманьшев А.В. Прикладная криптография. Использование и синтез криптографических интерфейсов. - М: Русское издание, 2003.
14. Варфоломеев А.А., Жуков А.Е., Пудовкина М.А. Блокировать криптосистемы. Основные свойства и методы анализа устойчивости. - М.: ПАИМС, 2000.
15. А.А. Мухачев, В.А. Хорошко Практические методы криптографии. - М., изд. Полиграф-Консалтинг, 2005. - 209 с.
16. https://www.edx.org/learn/cryptography
17. https://www.synopsys.com/glossary/what-is-cryptography.html
18. https://onlinedegrees.und.edu/blog/5-cryptography-tools/)
19. https://intellipaat.com/blog/what-is-cryptography/?US
20. https://elib.belstu.by/bitstream/123456789/33420/1/Bernikov_sravnitel%27nyj%20analiz.pdf

ƏLAVƏLƏR

Açar yaratma alqoritminin C# proqramlaşdırma dilində proqram təminatının tətbiqi:

```
using System;

public class RC6
{
   private uint[] S;
   private const int wordSize = 32;
   private const int numberOfRounds = 20;
   private const uint P32 = 0xB7E15163;
   private const uint Q32 = 0x9E3779B9;

   public void MakeEncryptionKey(byte[] key)
   {
      int c = (int)Math.Ceiling(key.Length / (double)sizeof(uint));
      uint[] L = new uint[c];
      for (int i = key.Length - 1; i >= 0; i--)
      {
         L[i / 4] = (L[i / 4] << 8) + key[i];
      }

      S = new uint[2 * numberOfRounds + 4];
      S[0] = P32;
      for (int i = 1; i < S.Length; i++)
      {
         S[i] = S[i - 1] + Q32;
      }

      int v = 3 * Math.Max(c, 2 * numberOfRounds + 4);
      uint A = 0;
      uint B = 0;

      for (int k = 1; k <= v; k++)
      {
         int i = 0;
         int j = 0;

         A = S[i] = RotateLeft(S[i] + A + B, 3);
         B = L[j] = RotateLeft(L[j] + A + B, (int)(A + B));
         i = (i + 1) % (2 * numberOfRounds + 4);
         j = (j + 1) % c;
```

```
        }
    }

    private uint RotateLeft(uint value, int count)
    {
        return (value << count) | (value >> (wordSize - count));
    }

    private uint RotateRight(uint value, int count)
    {
        return (value >> count) | (value << (wordSize - count));
    }
}

class Program
{
    static void Main()
    {
        byte[] key = new byte[] { 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF };

        RC6 rc6 = new RC6();
        rc6.MakeEncryptionKey(key);

        Console.WriteLine("RC6 Açarı Uğurla Yaradıldı!");
    }
}
```

Standart RC6 şifrələmə prosesinin C# proqramlaşdırma dilində yazılışı:

```
using System;

public class RC6
{
    private uint[] S;
    private const int WordSize = 32;
    private const int Rounds = 20;
    private const uint P32 = 0xB7E15163;
    private const uint Q32 = 0x9E3779B9;

    public byte[] Encrypt(byte[] plainText)
    {
        int blockSize = sizeof(uint);
        int paddedLength = ((plainText.Length + blockSize - 1) / blockSize) * blockSize;
        uint[] cipherText = new uint[paddedLength / blockSize];
        for (int i = 0; i < plainText.Length; i += blockSize)
```

```
    {
        uint block = 0;
        for (int j = 0; j < blockSize; j++)
        {
            if (i + j < plainText.Length)
            {
                block = (block << 8) + plainText[i + j];
            }
            else
            {
                block <<= 8;
            }
        }
        cipherText[i / blockSize] = EncryptBlock(block);
    }

    byte[] result = new byte[cipherText.Length * blockSize];
    for (int i = 0; i < cipherText.Length; i++)
    {
        uint block = cipherText[i];
        for (int j = blockSize - 1; j >= 0; j--)
        {
            result[i * blockSize + j] = (byte)(block & 0xFF);
            block >>= 8;
        }
    }

    return result;
}

private uint EncryptBlock(uint block)
{
    uint A = block + S[0];
    uint B = block + S[1];
    for (int i = 1; i <= Rounds; i++)
    {
        A = RotateLeft(A ^ B, (int)B) + S[2 * i];
        B = RotateLeft(B ^ A, (int)A) + S[2 * i + 1];
    }
    return A;
}

public class Program
{
    public static void Main(string[] args)
```

```
        {
            //Şifrələnəcək məlumat

            string Text = "Hello World";
            byte[] plainBytes = System.Text.Encoding.UTF8.GetBytes(Text);

            // Açar

            byte[] key = { 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF };

            // RC6 şifreleme obyektini yaradırıq

            RC6 rc6 = new RC6(key);

            // Göndərilən məlumatı şifrələmə

            byte[] encryptedBytes = rc6.Encrypt(plainBytes);

            // Şifrələnmiş məlumatı ekranda göstərmək

            Console.WriteLine($"Göndərilən məlumat:  {Text}");
            Console.WriteLine();
            Console.WriteLine("Şifrələnmiş məlumat: " +
BitConverter.ToString(encryptedBytes).Replace("-", ""));


        }
    }
}
```

Şifrə açma alqoritminin C# proqramlaşdırma dilində proqram təminatının tətbiqi:

```
using System;
public class RC6
{
    private uint[] S;
    private const int wordSize = 32;
    private const int numberOfRounds = 20;
    private const uint P32 = 0xB7E15163;
    private const uint Q32 = 0x9E3779B9;

    public RC6(byte[] key)
    {
        int c = (int)Math.Ceiling(key.Length / (double)sizeof(uint));
```

```
    uint[] L = new uint[c];
    for (int i = key.Length - 1; i >= 0; i--)
    {
        L[i / 4] = (L[i / 4] << 8) + key[i];
    }

    S = new uint[2 * numberOfRounds + 4];
    S[0] = P32;
    for (int i = 1; i < S.Length; i++)
    {
        S[i] = S[i - 1] + Q32;
    }
    int maxIndex = (c > S.Length) ? (3 * c) : (3 * S.Length);
    uint A = 0;
    uint B = 0;
    for (int counter = 0; counter < maxIndex; counter++)
    {
        int i = 0;
        int j = 0;
        A = S[i] = RotateLeft(S[i] + A + B, 3);
        B = L[j] = RotateLeft(L[j] + A + B, (int)(A + B));
        i = (i + 1) % S.Length;
        j = (j + 1) % c;
    }
}
public byte[] Decrypt(byte[] encryptedData)
{
    uint[] temp = ToUInt32Array(encryptedData);
    uint A = temp[0];
    uint B = temp[1];
    for (int i = 2 * numberOfRounds + 3; i >= 2; i--)
    {
        B = RotateRight(B - S[i], (int)A);
        A = RotateRight(A - S[i - 1], (int)B);
    }
    B -= S[1];
    A -= S[0];
    return ToByteArray(new uint[] { A, B });
}

private uint RotateLeft(uint value, int count)
{
    return (value << count) | (value >> (wordSize - count));
}
```

```
    private uint RotateRight(uint value, int count)
    {
        return (value >> count) | (value << (wordSize - count));
    }

    private uint[] ToUInt32Array(byte[] data)
    {
        int numberOfWords= (int)Math.Ceiling(data.Length / (double)sizeof(uint));
        uint[] result = new uint[numberOfWords];
        for (int i = 0; i < numberOfWords; i++)
        {
            for (int j = 0; j < sizeof(uint); j++)
            {
                if (i * sizeof(uint) + j < data.Length)
                {
                    result[i] += (uint)data[i * sizeof(uint) + j] << (8 * j);
                }
            }
        }
        return result;
    }
    private byte[] ToByteArray(uint[] data)
    {
        byte[] result = new byte[data.Length * sizeof(uint)];
        for (int i = 0; i < data.Length; i++)
        {
            for (int j = 0; j < sizeof(uint); j++)
            {
                result[i * sizeof(uint) + j] = (byte)(data[i] >> (8 * j));
            }
        }
        return result;
    }
}
class Program
{
    static void Main()
    {
        byte[] key = new byte[] { 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF };

        byte[] encryptedData = new byte[] {/*Şifrələnmiş məlumat*/};

        RC6 rc6 = new RC6(key);

        byte[] decryptedData = rc6.Decrypt(encryptedData);
```

```
        Console.WriteLine("Decrypted Data:");
        PrintByteArray(decryptedData);
    }
    static void PrintByteArray(byte[] array)
    {
        foreach (byte b in array)
        {
            Console.Write(b.ToString("X2") + " ");
        }
        Console.WriteLine();
    }
}
```

Modifikasiya olunmuş RC6 açarının yaradılması prosesi C# proqramlaşdırma dilində yazılışı

```
using System;
public class TfmMain
{
    public static void MakeEnKey()
    {
        string key = fmEncrypt.EdKey;
        int Len = key.Length;
        uint[] L = new uint[64];
        byte e, k, v, i, j, i1, j1;
        uint A, B, C, D;

        // Baytları L massivinə ən az önəmli olandan başlayaraq sonuna sıfırlar əlavə edərək yazırıq
        Array.Clear(L, 0, L.Length);
        for (int idx = 0; idx < Len; idx++)
        {
            L[idx / 4] |= (uint)(key[idx] << ((idx % 4) * 8));
        }

        e = (byte)(Len / 4);
        if (Len % 4 != 0)
            e++;

        // 32 bitlik S[0;2r+4] açar cədvəlinin yaradılması
        uint[] S = new uint[2 * Len + 4];
```

```
        S[0] = 0xB7E15163;
        for (int idx = 1; idx < 2 * Len + 4; idx++)
        {
            S[idx] = S[idx - 1] + 0x9E3779B9;
        }

        i = 0;
        j = 0;
        i1 = 0;
        j1 = 0;
        A = 0;
        B = 0;
        C = 0;
        D = 0;

        if (e > 2 * Len + 4)
            v = (byte)(3 * e);
        else
            v = (byte)(3 * (2 * Len + 4));

        for (k = 1; k <= v; k++)
        {
            A = (S[i] = LRot32(S[i] + A + B, 3));
            B = (L[j] = LRot32(L[j] + A + B, (int)(A + B)));
            C = (S[i1] = RRot32(S[i1] + C + D, 3));
            D = (L[j1] = RRot32(L[j1] + C + D, (int)(C + D)));

            i = (byte)((i + 1) % (2 * Len + 4));
            j = (byte)((j + 1) % e);
            i1 = (byte)((i1 + 1) % (2 * Len + 4));
            j1 = (byte)((j1 + 1) % e);
        }
    }

    private static uint LRot32(uint x, int y)
    {
        return (x << y) | (x >> (32 - y));
    }

    private static uint RRot32(uint x, int y)
    {
        return (x >> y) | (x << (32 - y));
    }
}
```

```
public class fmEncrypt
{
   public static string EdKey { get; set; }
}

public class Program
{
   public static void Main(string[] args)
   {
      fmEncrypt.EdKey = "YourKey"; // Açarınızla əvəz ediləcək

      TfmMain.MakeEnKey();

      // Ekranda Şifrələmə açarı uğurla yaradıldı. yazısı yazılacaq

      Console.WriteLine("Encryption key generated successfully.");
   }
}
```

Modifikasiya olunmuş RC6 şifrəsinin açılması prosesi C# proqramlaşdırma dilində

```
using System;
public class RC6Decryption
{
   private uint[] S;
   private int w;
   private int r;

   public RC6Decryption(byte[] key, int wordSize, int rounds)
   {
      w = wordSize;
      r = rounds;

      int c = (int)Math.Ceiling(key.Length / (double)(w / 8));
      uint[] L = new uint[c];
      for (int i = key.Length - 1; i >= 0; i--)
      {
         L[i / (w / 8)] = (L[i / (w / 8)] << 8) + key[i];
      }

      S = new uint[3 * r + 4];
      S[2 * r + 3] = 0xB7E15163;
      for (int i = 2 * r + 2; i >= 0; i--)
      {
```

```
            S[i] = S[i + 1] - 0x9E3779B9;
        }

        uint A = 0;
        uint B = 0;
        for (int k = 3 * Math.Max(c, 2 * r + 4) - 1; k >= 0; k--)
        {
            B = L[k % c] = RotateRight(L[k % c] - A - B, (int)(A + B));
            A = S[k % (2 * r + 4)] = RotateRight(S[k % (2 * r + 4)] - A - B, 3);
        }
    }

    public byte[] Decrypt(byte[] input)
    {
        int blockSize = 6 * (w / 8);
        if (input.Length % blockSize != 0)
        {
            throw new ArgumentException("Input length must be a multiple of the block
size.");
        }

        byte[] output = new byte[input.Length];

        for (int offset = 0; offset < input.Length; offset += blockSize)
        {
            uint A = BytesToUInt(input, offset);
            uint B = BytesToUInt(input, offset + 4);
            uint C = BytesToUInt(input, offset + 8);
            uint D = BytesToUInt(input, offset + 12);
            uint E = BytesToUInt(input, offset + 16);
            uint F = BytesToUInt(input, offset + 20);

            A = A - S[2 * r + 1];
            C = C - S[2 * r + 2];
            E = E - S[2 * r + 3];

            for (int i = r - 1; i >= 1; i--)
            {
                uint temp = F;
                F = E;
                E = D;
                D = C;
                C = B;
                B = A;
                A = temp;
```

```
            uint t = RotateLeft(B * (2 * B + 1), (int)Math.Log2(w));
            uint v = RotateLeft(D * (2 * D + 1), (int)Math.Log2(w));
            uint x = RotateLeft(F * (2 * F + 1), (int)Math.Log2(w));

            A = RotateRight((A ^ t) - S[2 * i], (int)x);
            C = RotateRight((C ^ v) - S[2 * i + 1], (int)t);
            E = RotateRight((E ^ x) - S[2 * i + 2], (int)v);
         }

         B = B - S[0];
         D = D - S[1];
         F = F - S[2];

         UIntToBytes(A, output, offset);
         UIntToBytes(B, output, offset + 4);
         UIntToBytes(C, output, offset + 8);
         UIntToBytes(D, output, offset + 12);
         UIntToBytes(E, output, offset + 16);
         UIntToBytes(F, output, offset + 20);
      }

      return output;
   }

   private uint RotateLeft(uint value, int count)
   {
      return (value << count) | (value >> (32 - count));
   }

   private uint RotateRight(uint value, int count)
   {
      return (value >> count) | (value << (32 - count));
   }

   private uint BytesToUInt(byte[] input, int offset)
   {
      return (uint)((input[offset] << 24) | (input[offset + 1] << 16) | (input[offset + 2]
<< 8) | input[offset + 3]);
   }

   private void UIntToBytes(uint value, byte[] output, int offset)
   {
      output[offset] = (byte)(value >> 24);
      output[offset + 1] = (byte)(value >> 16);
```

```
        output[offset + 2] = (byte)(value >> 8);
        output[offset + 3] = (byte)value;
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        byte[] acar = { 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF };
        byte[] sifrelimetn = { 0x0A, 0x1D, 0x2C, 0x8F, 0x49, 0x29, 0x30, 0x12 };

        RC6Decryption rc6 = new RC6Decryption(acar, 32, 20);
        byte[] text = rc6.Decrypt(sifrelimetn);

        Console.WriteLine("Sifrelenmis-metn:"+ BitConverter.ToString(sifrelimetn));
        Console.WriteLine("Sifresi acilmis metn:  " + BitConverter.ToString(text));
    }
}
```

Fayl Şifrələnmə prosesinin proqram təminatı:C# proqramlaşdırma dilində

```
using System;
using System.Drawing;
using System.IO;

public static class RC6Encryption
{
    public static void EncryptFile(string inputFile, string outputFile, int count)
    {
        using (FileStream fsInput = new FileStream(inputFile, FileMode.Open))
        using (FileStream fsOutput = new FileStream(outputFile, FileMode.Create))
        {
            int sizeFile = (int)fsInput.Length;
            int currentProgress = 0;

            fsOutput.Write(BitConverter.GetBytes(sizeFile), 0, sizeof(int));
            fsOutput.Write(BitConverter.GetBytes(count), 0, sizeof(int));

            byte[] buffer = new byte[24];
            int bytesRead;

            while ((bytesRead = fsInput.Read(buffer, 0, buffer.Length)) > 0)
            {
                if (bytesRead < buffer.Length)
```

```
            {
                for (int i = bytesRead; i < buffer.Length; i++)
                    buffer[i] = (byte)' ';
            }

            fsOutput.Write(buffer, 0, buffer.Length);

            for (int i = 0; i < count; i++)
            {
                EncryptBlock(buffer);
            }

            fsOutput.Write(buffer, 0, buffer.Length);

            currentProgress += bytesRead;
            Console.WriteLine("Progress: {0}%", (currentProgress * 100) / sizeFile);
        }
    }
}

private static void EncryptBlock(byte[] block)
{
    uint[] key = GetKey(); // Şifrələmə açarı
    uint[] state = ConvertToUIntArray(block); // Əməliyyat blokunu 32 bitlik tam ədəd massivi kimi çevirin

    uint wordSize = 32;
    uint rounds = 20;
    uint p = 0xb7e15163;
    uint q = 0x9e3779b9;

    uint[] register = new uint[2 * rounds + 4];
    register[0] = p;

    for (int i = 1; i < register.Length; i++)
    {
        register[i] = register[i - 1] + q;
    }

    uint a = state[0];
    uint b = state[1];
    uint c = state[2];
    uint d = state[3];

    b = RotateLeft(b * (2 * b + 1), (int)wordSize);
```

```
    d = RotateLeft(d * (2 * d + 1), (int)wordSize);

    for (int i = (int)rounds - 1; i >= 0; i--)
    {
        d = RotateRight(d - register[2 * i + 1], (int)a % (int)wordSize);
        b = RotateRight(b - register[2 * i], (int)d % (int)wordSize);
        c = RotateRight(c - register[2 * i + 3], (int)b % (int)wordSize);
        a = RotateRight(a - register[2 * i + 2], (int)c % (int)wordSize);

        d = RotateRight(d ^ c, (int)a % (int)wordSize);
        c = RotateRight(c ^ b, (int)d % (int)wordSize);
        b = RotateRight(b ^ a, (int)c % (int)wordSize);
        a = RotateRight(a ^ d, (int)b % (int)wordSize);
    }

    a = (a + register[0]) % uint.MaxValue;
    b = (b + register[1]) % uint.MaxValue;
    c = (c + register[2]) % uint.MaxValue;
    d = (d + register[3]) % uint.MaxValue;

    state[0] = a;
    state[1] = b;
    state[2] = c;
    state[3] = d;

    Array.Copy(ConvertToByteArray(state), block, block.Length);
}

private static uint RotateLeft(uint value, int count)
{
    return (value << count) | (value >> (int)(32 - count));
}

private static uint RotateRight(uint value, int count)
{
    return (value >> count) | (value << (int)(32 - count));
}

private static uint[] ConvertToUIntArray(byte[] bytes)
{
    uint[] result = new uint[bytes.Length / 4];
    for (int i = 0; i < result.Length; i++)
    {
        result[i] = BitConverter.ToUInt32(bytes, i * 4);
    }
```

```
        return result;
    }

    private static byte[] ConvertToByteArray(uint[] integers)
    {
        byte[] result = new byte[integers.Length * 4];
        for (int i = 0; i < integers.Length; i++)
        {
            BitConverter.GetBytes(integers[i]).CopyTo(result, i * 4);
        }
        return result;
    }


    private static uint[] GetKey()
    {
      // Təsadüfi şifrələmə açarı yaradırıq
      // Təsadüfi şifrələmə açarı yaratmaq üçün kriptoqrafik kitabxanadan və ya
      // alqoritmdən istifadə edə bilərik. məs- System.Security.Cryptography


        using(var rng = new
System.Security.Cryptography.RNGCryptoServiceProvider())

        {
            byte[] randomBytes = new byte[16];
            rng.GetBytes(randomBytes);

            uint[] key = new uint[4];
            Buffer.BlockCopy(randomBytes, 0, key, 0, randomBytes.Length);

            return key;
        }
    }
}

public class Program
{
    public static void Main()
    {
        string inputFile = "input.txt";
        string outputFile = "output.txt";
        int count = 5;

        RC6Encryption.EncryptFile(inputFile, outputFile, count);
```

```
    }
}
```

Faylın şifrəsinin açılması prosesinin proqram təminatı C# proqramlaşdırma dilində

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string fileBegin = "encryptedFile.bin";
        string fileEnd = "decryptedFile.bin";

        DecryptFile(fileBegin, fileEnd);

        Console.WriteLine("Decryption completed.");
    }

    static void DecryptFile(string fileBegin, string fileEnd)
    {
        using (FileStream f1 = new FileStream(fileBegin, FileMode.Open))
        using (FileStream f2 = new FileStream(fileEnd, FileMode.Create))
        {
            int sizeFile = (int)f1.Length;

            byte[] buf = new byte[24];
            byte[] tempBuff = new byte[24];
            byte[] tempBuff2 = new byte[24];

            int numRead;
            int i = 0;
            int count = 0;

            f1.Read(BitConverter.GetBytes(sizeFile), 0, sizeof(int));
            f1.Read(BitConverter.GetBytes(count), 0, sizeof(int));

            while ((numRead = f1.Read(buf, 0, buf.Length)) > 0)
            {
                i++;

                if (i == 1)
                {
```

```
            Buffer.BlockCopy(buf, 0, tempBuff, 0, buf.Length);
            for (int n = 1; n <= count; n++)
            {
               Decrypt(tempBuff); // RC6 decryption
               f2.Write(tempBuff, 0, tempBuff.Length);
            }
         }
         else if (i == 2)
         {
            Decrypt(buf); // RC6 decryption
            Buffer.BlockCopy(buf, 0, tempBuff2, 0, buf.Length);
            if (!CompareArrays(tempBuff, tempBuff2))
            {
               Console.WriteLine("Mesajı çap edin"); // Print the message
            }
         }

         f2.Write(buf, 0, buf.Length);
      }
   }
}

static void Decrypt(byte[] buffer)
{

   int round = 20; // RC6 alqoritmində dövrlərin sayı
   int blockSize = 16; // Blok ölçüsü (baytla)
   bayt[] açarı = { /* RC6 şifrələməsində istifadə olunan açar */ };

   uint[] s = GenerateSubkeys(key, rounds);

   uint a = BitConverter.ToUInt32(buffer, 0);
   uint b = BitConverter.ToUInt32(buffer, 4);

   b -= s[2 * rounds + 3];
   a -= s[2 * rounds + 2];

   for (int i = rounds; i >= 1; i--)
   {
      int j = (i * 2) - 1;
      b = RotateRight((b - s[j + 1]), (int)a) ^ a;
      a = RotateRight((a - s[j]), (int)b) ^ b;
   }

   b -= s[1];
```

```
    a -= s[0];

    Buffer.BlockCopy(BitConverter.GetBytes(a), 0, buffer, 0, 4);
    Buffer.BlockCopy(BitConverter.GetBytes(b), 0, buffer, 4, 4);
}

static uint[] GenerateSubkeys(byte[] key, int rounds)
{
    int keySize = key.Length;
    int c = keySize / 4;
    uint[] l = new uint[c];
    uint[] s = new uint[2 * rounds + 4];

    for (int i = 0; i < c; i++)
    {
        l[i] = BitConverter.ToUInt32(key, i * 4);
    }

    s[0] = 0xB7E15163;
    for (int i = 1; i < 2 * rounds + 4; i++)
    {
        s[i] = s[i - 1] + 0x9E3779B9;
    }

    uint a = 0;
    uint b = 0;
    int max = c > (2 * rounds + 4) ? c : (2 * rounds + 4);

    for (int i = 0, j = 0; j < max; i = (i + 1) % c, j++)
    {
        a = s[i] = RotateLeft((s[i] + a + b), 3);
        b = l[i] = RotateLeft((l[i] + a + b), (int)(a + b));
    }

    return s;
}

static uint RotateLeft(uint value, int count)
{
    return (value << count) | (value >> (32 - count));
}

static uint RotateRight(uint value, int count)
{
    return (value >> count) | (value << (32 - count));
```

```
    }

    static bool CompareArrays(byte[] arr1, byte[] arr2)
    {
        if (arr1.Length != arr2.Length)
            return false;

        for (int i = 0; i < arr1.Length; i++)
        {
            if (arr1[i] != arr2[i])
                return false;
        }

        return true;
    }
}
```

Parolun təxmin edilməsi prosesi C# proqramlaşdırma dilində yazılışı

```
using System;
using System.IO;
class Program
{
    static void Main()
    {
        FindPassword();
    }
    static void FindPassword()
    {
        uint[] L = new uint[64];
        byte e, k, v, i, j, i1, j1;
        uint A, B, C, D;
        string CurrentKey;
        FileStream CurFile;
        CurrentKey = "";
        // Başlama vaxtını yadda saxla

        CurFile = new FileStream(FileBegin, FileMode.Open, FileAccess.Read);
        CurFile.Seek(8, SeekOrigin.Begin);
        byte[] Temp_Buff = new byte[24];
        // Fayldan ilk 24 baytı Temp_Buff-a oxuyun
        CurFile.Read(Temp_Buff, 0, Temp_Buff.Length);
        // Göstəricini ikinci 24 bayta köçürün
        CurFile.Seek(32, SeekOrigin.Begin);
```

```
byte[] Temp_Buff2 = new byte[24];
// Fayldan ikinci 24 baytı Temp_Buff2-ə oxuyun
CurFile.Read(Temp_Buff2, 0, Temp_Buff2.Length);
CurFile.Close();

// şifrə yoxlanmasi
while (true)
{
   if (fmPassword.rbNumber.Checked)
      CurrentKey = DigitalPassword(CurrentKey);

   if (fmPassword.rbWord.Checked)
      CurrentKey = WordPassword(CurrentKey);

   if (fmPassword.rbWordsDigits.Checked)
      CurrentKey = WordsDigits(CurrentKey);

   if (fmPassword.rbSpisok.Checked)
      CurrentKey = ListPassword(CurrentKey, fmPassword.edList.Text);

   if (fmPassword.rbAll.Checked)
      CurrentKey = AllPassword(CurrentKey);

   if (fmPassword.cbShowPass.Checked)
      fmPassword.lbCurrPass.Caption = CurrentKey;

   // Deşifrə açarı yaradın

   string Açar = CurrentKey;
   int Len = Açar.Length;

   // Əsas dəyişənləri yaradın

   Array.Clear(L, 0, L.Length);
   Köçürün(Açar, L, Len + 1);
   e = (byte)(Len / 4);
   if (Len % 4 != 0)
      e++;

   // 32 bitlik sözlərin S[0;2r+4] açar cədvəlinin formalaşması

   uint[] S = new uint[2 * r + 4 + 1];
   S[0] = 0xB7E15163;
   for (i = 1; i <= 2 * r + 3; i++)
      S[i] = S[i - 1] + 0x9E3779B9;
```

```
            i = 0; j = 0; i1 = 0; j1 = 0;
            A = 0; B = 0; C = 0; D = 0;

            if (e > 2 * r + 4)
               v = 3 * e;
            else
               v = 3 * (2 * r + 4);

            for (k = 1; k <= v; k++)
            {
               A = LRot32((S[i] + A + B), 3);
               S[i] = A;
               B = LRot32((L[j] + A + B), (A + B));
               L[j] = B;
               i = (byte)((i + 1) % (2 * r + 4));
               j = (byte)((j + 1) % e);
               C = RRot32((S[i1] + C + D), 3);
               S[i1] = C;
               D = RRot32((L[j1] + C + D), (C + D));
               L[j1] = D;
               i1 = (byte)((i1 + 1) % (2 * r + 4));
               j1 = (byte)((j1 + 1) % e);
            }

            // İkinci 24 baytın şifrəsini açın

            byte[] Buf = Temp_Buff2;
            Decryption();

            // Əgər Buff = Temp_Buff, onda parol tapılıbsa, döngəni dayandırın, parolu göstərin
            if (Temp_Buff[0] == Buf[0] && Temp_Buff[1] == Buf[1] && Temp_Buff[2] == Buf[2] &&
               Temp_Buff[3] == Buf[3] && Temp_Buff[4] == Buf[4] && Temp_Buff[5] == Buf[5])
            {
               Console.WriteLine("Parola tapıldı: " + CurrentKey);
               break;
            }
         }
      }

      static string DigitalPassword(string currentKey)
      {
```

```
    // rəqəmlə parol yaratma məntiqi
    string generatedPassword = "";
    for (int i = 0; i < currentKey.Length; i++)
    {
        if (char.IsDigit(currentKey[i]))
        {
            generatedPassword += currentKey[i];
        }
    }
    return generatedPassword;
}

static string WordPassword(string currentKey)
{
    // sözlə parol yaratma məntiqi
    string generatedPassword = "";
    string[] words = currentKey.Split(' ');
    foreach (string word in words)
    {
        if (word.Length > 3)
        {
            generatedPassword += word.Substring(0, 3);
        }
    }
    return generatedPassword;
}

static string WordsDigits(string currentKey)
{
    // Sözlər və rəqəmlər parol yaratma məntiqi
    string generatedPassword = "";
    string[] words = currentKey.Split(' ');
    foreach (string word in words)
    {
        for (int i = 0; i < word.Length; i++)
        {
            if (char.IsLetterOrDigit(word[i]))
            {
                generatedPassword += word[i];
            }
        }
    }
    return generatedPassword;
}
```

```
static string ListPassword(string currentKey, string listText)
{

    string generatedPassword = "";
    string[] listItems = listText.Split(',');
    foreach (string item in listItems)
    {
        if (currentKey.Contains(item))
        {
            generatedPassword += item;
        }
    }
    return generatedPassword;
}

static string AllPassword(string currentKey)
{
    // Bütün parol yaratma məntiqi
    string generatedPassword = currentKey.ToUpper();
    return generatedPassword;
}

static uint LRot32(uint x, byte n)
{
    return (x << n) | (x >> (32 - n));
}

static uint RRot32(uint x, byte n)
{
    return (x >> n) | (x << (32 - n));
}

static void Köçürün(string str, uint[] arr, int len)
{
    int j = 0;
    for (int i = 0; i < len; i++)
    {
        arr[i] = (uint)((str[j] << 24) | (str[j + 1] << 16) | (str[j + 2] << 8) | str[j + 3]);
        j += 4;
    }
}

static void Decryption(byte[] buffer)
{
```

```
        byte[] key = { } //açarı daxil edin;
    int keySize = key.Length * 8; // açarın bit uzunluğu
        int rounds = 20; // dövr sayı
        int blockSize = 16; // Blok ölçüsü

        // RC6 şifrələmə alqoritmindən istifadə edərək buferi açın

        RC6Decrypt(buffer, key, keySize, rounds, blockSize);
    }

    static void RC6Decrypt(byte[] buffer, byte[] key, int keySize, int rounds, int
blockSize)
    {
        // RC6 şifreleme algoritması məntiqi

        // Açarı hazırla

        uint[] keyWords = KeyExpansion(key, keySize, rounds);

        // Şifrələmə modulu parametrlərini təyin edin

        uint blockSizeWords = (uint)(blockSize / 4);
        uint[] block = new uint[blockSizeWords];
        uint[] tempBlock = new uint[blockSizeWords];

        // Blok ölçüsünü yoxlayırıq

        if (buffer.Length % blockSize != 0)
        {
            throw new ArgumentException("Invalid buffer size for RC6 decryption.");
        }

        // Hər bloku həll edin

        for (int offset = 0; offset < buffer.Length; offset += blockSize)
        {

        Buffer.BlockCopy(buffer, offset, block, 0, blockSize);

            // Təhlil üçün RC6 alqoritmini tətbiq edin
            RC6DecryptBlock(block, tempBlock, keyWords, rounds);

            // Şifrə açılmış bloku buferə yazın
            Buffer.BlockCopy(tempBlock, 0, buffer, offset, blockSize);
```

```
        }
    }

    static void RC6DecryptBlock(uint[] block, uint[] tempBlock, uint[] keyWords, int
rounds)
    {
        // RC6 şifrələmə alqoritmi blokdan çıxarma məntiqi


        uint A = block[0];
        uint B = block[1];
        uint C = block[2];
        uint D = block[3];


        for (int i = rounds; i > 0; i--)
        {
            uint tempD = D;
            D = C;
            C = B;
            B = A;
            A = RC6RightRotate((B * (2 * B + 1)), 5);
            C = RC6RightRotate((C - keyWords[2 * i + 1]), (int)A);
            A = RC6RightRotate((A - keyWords[2 * i]), (int)B);
            D = RC6RightRotate((D - keyWords[2 * i + 1]), (int)C);
            B = RC6RightRotate((B - keyWords[2 * i]), (int)D);
        }


        uint temp = A;
        A = B;
        B = C;
        C = D;
        D = temp;


        tempBlock[0] = A;
        tempBlock[1] = B;
        tempBlock[2] = C;
        tempBlock[3] = D;
    }

    static uint[] KeyExpansion(byte[] key, int keySize, int rounds)
    {
        // açarın genişləndirilməsi məntiqi
```

```
    // Açarın söz ölçüsünü təyin edin
    int keyWords = keySize / 32;

    // Sözləri açar massivə köçürün
    uint[] keyArray = new uint[keyWords];
    for (int i = 0; i < keyWords; i++)
    {
        keyArray[i] = BitConverter.ToUInt32(key, i * 4);
    }

    // RC6 açarının genişləndirilməsi alqoritmini tətbiq edin
    uint[] expandedKey = new uint[2 * rounds + 4];
    expandedKey[0] = 0xB7E15163;
    for (int i = 1; i < expandedKey.Length; i++)
    {
        expandedKey[i] = expandedKey[i - 1] + 0x9E3779B9;
    }

    uint[] temp = new uint[3];
    uint A = 0;
    uint B = 0;
    int max = Math.Max(keyWords, expandedKey.Length);
    for (int i = 0, j = 0, k = 0; k < max; k++)
    {
        temp[j] = RC6LeftRotate((expandedKey[i] + A + B), 3);
        expandedKey[i] = temp[j];
        A = temp[j];

        temp[j] = RC6LeftRotate((keyArray[i] + A + B), (int)(A + B));
        keyArray[i] = temp[j];
        B = temp[j];

        i = (i + 1) % keyWords;
        j = (j + 1) % 3;
    }

    return expandedKey;
}

static uint RC6LeftRotate(uint value, int shift)
{
    return (value << shift) | (value >> (32 - shift));
}
```

```
    static uint RC6RightRotate(uint value, int shift)
    {
        return (value >> shift) | (value << (32 - shift));
    }

}
```

XÜLASƏ

Kriptoqrafik təhlükəsizlik metodlarının tətbiqatı, informasiya təhlükəsizliyi problemlərinin əsas həllinə çatmağı təmin edir. Autentifikasiya, məlumatların şifrələnməsi, bütönlüyün nəzarəti, elektron digəri imza - bugün geniş inkişaf göstərən kəsişən bir qrup tərəfindən tanınan məlumatların təhlükəsizliyində yaxşı məşhur məlumatlardır. Ancaq çoxlu istifadəçilər şəxsi məlumatlarının dəfələrlə qorunmasını gözardı edirlər, ki, çox hallarda heç bir marağa səbəb olmur və həmçinin kompüterə qarşı ümumi qorunma tədbirlərini nəzərdə tutmurlar. Çoxlu istifadəçilər hətta antivirus və istənilən bir kriptosik alogritm vasitəsilə məlumatların məcburi şifrələnməsinin qurulması haqqında eşitmədilər və də düşünmədilər. Bu nöqsanlar bir vaxt sonra kompüterdəki məlumatların müxtəlif viruslarla yoluxmasına və əməliyyat sisteminin yavaşlamasına və ya çöküşünə səbəb olur.

Kriptoqrafiya təhlükəsizlik metodlarının tətbiq edilməsi əsas məlumat təhlükəsizliyi problemlərinin həllinə gətirir. Bu, istifadəçi və idarəçi məlumatı, informasiya resurslarını həmçinin məlumatların ümumi təhlükəsizliyini təmin edən aşağıdakı kriptografiya təhlükəsizlik metodlarının həyata keçirilməsi yolu ilə əldə edilə bilər

SUMMARY

The application of cryptographic security methods provides the main solution to information security problems. Authentication, data encryption, integrity control, electronic signature - these are well-known information in data security, recognized by a cross-sectional group that shows extensive development today. However, many users repeatedly ignore the protection of their personal data, which in most cases does not cause any interest, and also do not consider general protection measures against the computer. Many users have not even heard and did not think about the installation of mandatory encryption of data through an antivirus and any cryptographic alogrithm. These errors eventually lead to the infection of computer data with various viruses and slow down or crash of the operating system.

Applying cryptographic security methods solves the main data security problems. This can be achieved by implementing the following cryptographic security methods that ensure user and administrator information, information resources as well as general security of data

РЕЗЮМЕ

Применение криптографических методов защиты обеспечивает основное решение проблем защиты информации. Аутентификация, шифрование данных, контроль целостности, электронная подпись — это общеизвестная информация в области безопасности данных, признанная поперечной группой, которая сегодня демонстрирует широкое развитие. Однако многие пользователи неоднократно игнорируют защиту своих персональных данных, что в большинстве случаев не вызывает никакого интереса, а также не рассматривают общие меры защиты от компьютера. Многие пользователи даже не слышали и не задумывались об установке обязательного шифрования данных через антивирус и какие-либо криптоалгоритмы. Эти ошибки со временем приводят к заражению компьютерных данных различными вирусами и замедлению или сбою операционной системы.

Применение криптографических методов защиты решает основные проблемы безопасности данных. Это может быть достигнуто за счет реализации следующих методов криптографической защиты, которые обеспечивают информацию пользователя и администратора, информационные ресурсы, а также общую безопасность данных.