



UNIVERZITET SINGIDUNUM

Fakultet za informatiku i računarstvo

APLIKACIJA ZA ŠIFROVANJE FAJLOVA U C# PROGRAMSKOM JEZIKU

- Diplomski rad -

Mentor:

Doc. dr Saša Adamović

Kandidat:

Kristina Kunjadić-Ćulibrk

Beograd, 2015.



FAKULTET ZA INFORMATIKU I RAČUNARSTVO

UNIVERZITET SINGIDUNUM

Fakultet za informatiku i računarstvo

Beograd, Danijelova 32

Kandidat: Kristina Kunjadić-Ćulibrk

Broj indeksa: 2011/200525

Smer: Informatika i računarstvo

Tema: Aplikacija za šifrovanje fajlova u C# programskom jeziku

Datum odobrenja rada:
Beograd, 03. septembar 2015.

MENTOR

Doc. dr Saša Adamović

DEKAN

Prof. dr Dragan Cvetković

Aplikacija za šifrovanje fajlova u C# programskom jeziku

Sažetak: U ovom radu predstavljena je sopstvena aplikacija čija je primarna svrha zaštita poverljivih podataka. Aplikacija, pored šifarskog sistema, ima ugrađen i mehanizam koji izračunava heš vrednost otvorenog fajla, i on može pomoći u utvrđivanju njegove autentičnosti ukoliko se šalje putem javne mreže ili se sumnja na neovlašćeni pristup ili izmenu. Biće navedene neke od sličnih aplikacija koje se najviše koriste u svrhu zaštite, i biće objašnjen način njihovog funkcionisanja. Teorijsko istraživanje podrazumeva pregled najpoznatijih simetričnih šifrskih algoritama kao i heš funkcija, sa posebnim osvrtom na Rijndael algoritam i SHA-1 algoritam, koji su sastavni deo aplikacije. Biće objašnjen način upotrebe aplikacije i dat pregled njenih funkcija.

Ključne reči: simetrični šifarski sistemi, heš funkcije, aplikacija za šifrovanje fajlova

The application for encrypting files in C# programming language

Abstract: In this paper, personally designed application is going to be presented. It's primary purpose is the protection of confidential data. The application, in addition to the encryption system, has a built-in mechanism that calculates the hash value of an open file. It can help in determining file's authenticity if sent via the public network or if it is suspected to an unauthorized access or alteration. This study will list some of the similar applications that are widely used for protection of data, and will explain the way of their functioning. Theoretical study includes an overview of the most popular symmetric algorithms and hash functions, with special emphasis on Rijndael algorithm and SHA-1 algorithm, which are an integral part of the application. The way of how to use the application will be explained, and the overview of its functions will be given.

Keywords: symmetric cipher systems, hash functions, application for file encryption

SADRŽAJ

1. Uvod.....	5
1.1 Predmet istraživanja.....	5
1.2 Doprinosi rada	6
1.3 Struktura rada.....	6
2. Pregled u oblasti istraživanja	7
2.1 VeraCrypt.....	7
2.2 GNU Privacy Guard	10
2.3 AES Crypt	13
2.4 7-zip	15
2.5 AxCrypt	17
3. Teorijske osnove istraživanja	19
3.1 Pregled simetričnih šifarskih algoritama	19
3.1.1 DES.....	19
3.1.2 Blowfish	22
3.1.3 AES.....	22
3.2 Rijndael algoritam	24
3.3 Heš funkcije	27
3.3.1 Merklova struktura.....	28
3.3.2 MD5	29
3.3.3 SHA	30
3.4 SHA-1	31
4. Pregled predloženog rešenja	35
4.1 Uputstvo za upotrebu.....	35
4.2 Microsoft Visual Studio	37
4.3 C# programski jezik.....	38

4.4 Pregled funkcija korišćenih u kodu aplikacije	38
5. Zaključak.....	40
5.1 Rezultati i doprinosi.....	40
5.2 Predlog za budući rad	40
Literatura.....	41
Dodatak 1	43
Dodatak 2	45
Dodatak 3	48

Spisak slika

2.1 Kreiranje volume fajla	7
2.2 Ponuđeni izbor šifarskih sistema i heš funkcija	8
2.3 Zahtev za unos lozinke	8
2.4 Učitavanje volume fajla na računar	9
2.5 Novi disk „M:“	9
2.6 Komande za prebacivanje fajlova	9
2.7 Zatvaranje volume fajla	10
2.8 Biranje fajla za šifrovanje	11
2.9 Pokretanje postupka šifrovanja kroz Windows Explorer	12
2.10 Izbor šifarskog sistema koji će se koristiti u procesu šifrovanja	12
2.11 Unošenje lozinke primaoca	13
2.12 Započinjanje procesa šifrovanja koristeći AES Crypt	14
2.13 Unošenje lozinke	14
2.14 Započinjanje procesa dešifrovanja koristeći AES Crypt	14
2.15 Pokretanje 7-zip aplikacije nad fajlom koji se šifruje	15
2.16 Prozor za podešavanje opcija šifrovanja	16
2.17 Unošenje tajne lozinke	16
2.18 Pokretanje AxCrypt alata kroz Windows Explorer	17
2.19 Izbor šifrovanja lozinkom i/ili fajl-ključem	18
3.1 Fejstelova struktura za simetrične šifre ^[7]	20
3.2 Grafički prikaz jedne runde DES algoritma ^[7]	21
3.3 Grafički prikaz strukture AES algoritma u slučaju korišćenja 128-bitnog ključa ^[13]	23
3.4 Grafički prikaz Rijndael algoritma ^[15]	26
4.1 Izgled aplikacije „CryptoFile“ nakon njenog pokretanja	35
4.2 Prozor za odabir fajla koji će se šifrovati	36
4.3 Prikaz foldera „Šifrovanje“	36

4.4 Izgled heš fajla nakon procesa dešifrovanja.....	37
--	----

Spisak tabela

Tabela 3.1 Specifikacije osnovnih SHA algoritama ^[16]	30
--	----

1. Uvod

1.1 Predmet istraživanja

Poslovanje, ili bilo kakav drugi vid digitalne komunikacije u današnje vreme neizostavno uključuje upotrebu računara i samim time skladištenje i čuvanje velike količine digitalnih informacija. Sa razvojem mnogobrojnih načina za zaštitu ličnih ili poslovnih informacija, koji podrazumevaju algoritme različitog nivoa kompleksnosti, paralelno su se razvijali i algoritmi za nedozvoljen pristup, izmenu ili čak krađu takvih informacija. Iz tog razloga, javlja se potreba za kompletnom kontrolom i zaštitom naših podataka. Uključenje bilo koje druge strane u taj proces, bilo da je to proizvođač softvera za zaštitu podataka koji koristimo ili prosto osoba sa kojom delimo računar, može znatno ugroziti našu bezbednost.

Istraživanjem trenutnog stanja na tržištu, kada su u pitanju rešenja za zaštitu podataka, došli smo do zaključka da svakako postoji veliki broj aplikacija koje su upravo razvijene u ovu svrhu. Međutim, kako smo već napomenuli rizike koje ovakav izbor zaštite nosi, bilo da je on apsolutno besplatan ili plaćen, kao najbolje rešenje ovog problema još uvek predstavlja razvoj sopstvenog softvera za zaštitu informacija.

Osnovna funkcija aplikacije, koja predstavlja temu ovog rada, biće šifrovanje svih vrsta fajlova Rijndael kriptografskim algoritmom koji predstavlja iterativni blokovski, simetrični šifarski algoritam, i predstavlja varijaciju AES algoritma. Ključ će biti veličine 128 bita, i korisnik će ga samostalno unositi u vidu šifre koja sadrži najmanje 8 slova ili brojeva po sopstvenom izboru.

Takođe, aplikacija će posedovati ugrađenu funkciju za automatsko izračunavanje heš vrednosti fajla pre šifrovanja i nakon dešifrovanja, i upis dobijene vrednosti u tekstualni fajl. Na taj način, korisnik će imati uvid u sam integritet fajla i biti u mogućnosti da zaključi da li je dobijeni fajl upravo onaj koji je inicijalno i šifrovan. U svrhu računanja heš vrednosti koristiće se jedna od najrasprostranjenijih postojećih heš funkcija SHA1.

Grafički korisnički interfejs aplikacije biće prilagođen prosečnom korisniku i omogućiće lako i jednostavno snalaženje u aplikaciji, dok će sama aplikacija vršiti prevenciju od bilo kakve moguće greške od strane korisnika. Na taj način upotreba aplikacije će biti sigurna i oslobođena od potencijalnih grešaka koje bi mogle ugroziti sam proces zaštite odabranog fajla.

Aplikacija će biti razvijena u programskom alatu Microsoft Visual Studio u programskom jeziku C#.

1.2 Doprinosi rada

Doprinos ovog rada predstavlja sopstveno kriptografsko rešenje za šifrovanje fajlova, bilo koje vrste, i ono podrazumeva visok stepen sigurnosti podataka. Aplikacija poseduje intuitivno okruženje za lak i jednostavan pristup osnovnim funkcijama.

Korisnik šifrjuje svoje podatke sopstvenim tajnim ključem, a zatim šifrovane podatke može da pošalje putem bilo koje javne mreže i da bude siguran u to da im niko, ko ne poseduje znanje o tajnom ključu, ne može neovlašćeno pristupiti.

Takođe, aplikacija poseduje funkcionalnost izračunavanja heš vrednosti fajla pre šifrovanja i nakon dešifrovanja. Ova funkcionalnost se obavlja automatski, i daje korisniku mogućnost provere integriteta dešifrovanog fajla, jednostavnim upoređivanjem dve heš vrednosti.

1.3 Struktura rada

U prvom poglavlju dato je obrazloženje zašto postoji potreba za razvojem sopstvenog rešenja za šifrovanje podataka i date su osnovne karakteristike aplikacije. Takođe, navedeni su doprinosi ostvareni izradom ovakvog rešenja, a na samom kraju poglavlja predstavljena je i struktura rada.

U drugom poglavlju dat je pregled već postojećih aplikacija, sa sličnim funkcionalnostima kao aplikacija koja predstavlja temu ovog rada. Pregled obuhvata pet najpopularnijih alata za šifrovanje podataka koji se trenutno koriste širom sveta i najčešći su izbor korisnika.

U trećem poglavlju predstavljene su teorijske osnove istraživanja koje su pomogle u odabiru šifarskog sistema i heš funkcije. Istraživanje obuhvata pregled šifarskih sistema DES, Blowfish i AES, kao i Rijndael algoritam na koji je stavljen akcenat nešto detaljnijim opisom. Kada su u pitanju heš funkcije u ovom poglavlju su objašnjeni Merklova struktura (koja predstavlja osnov za mnoge heš funkcije), zatim algoritmi MD5 i SHA-1, dok je SHA-1 objašnjen nešto detaljnije imajući u vidu njegovu implementaciju u rešenje.

U četvrtom poglavlju dat je pregled predloženog rešenja koje podrazumeva njegovo uputstvo za upotrebu, kratak opis korišćenog alata Microsoft Visual Studio i programskog jezika C#, kao i pregled i objašnjenje funkcija korišćenih u kodu programa. Kod glavnih funkcija naveden je na samom kraju rada u vidu dodatka.

U petom poglavlju kroz zaključak rada, ukratko su objašnjeni postignuti rezultati i doprinosi i dat je predlog za budući rad.

2. Pregled u oblasti istraživanja

U ovoj oblasti rada, fokusiraćemo se na neke od trenutno najpoznatijih alata koji se koriste za šifrovanje fajlova, način na koji oni rade kao i na algoritme koje koriste.^[1]

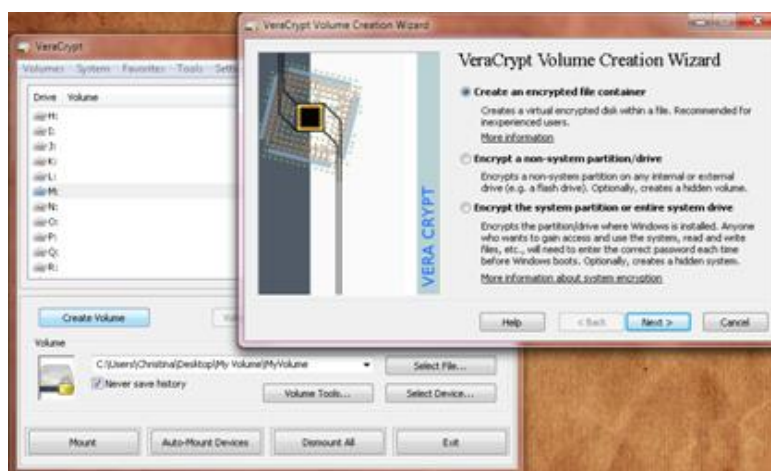
2.1 VeraCrypt

VeraCrypt predstavlja besplatan open-source alat koji služi za šifrovanje fajlova. Aplikacija podržava rad na Windows, Linux i Mac OS X operativnim sistemima.^[1]

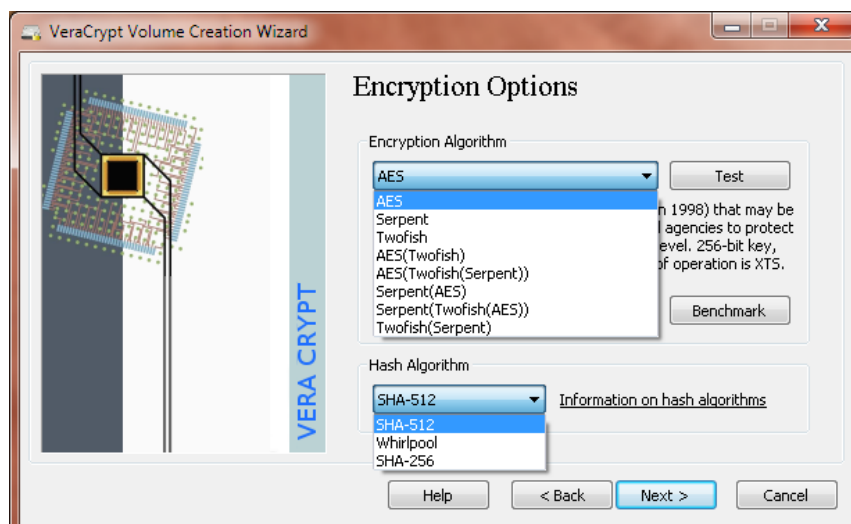
U zavisnosti od izbora korisnika, za šifrovanje se mogu koristiti simetrični šifarski sistemi AES, Serpent i Twofish kao i sve njihove kombinacije. Dužina ključa je 256 bita. Kada su u pitanju heš algoritmi ovaj program pruža korisniku mogućnost izbora između RIPEMED-160, SHA-512 i Whirlpool algoritma. Odabrani heš algoritam koristi se zajedno sa slučajnim podacima generisanim od strane programa u RAM memoriji kao parametar za funkciju Random Number Generator koja generiše glavne, pomoćne ključeve, kriptografsku "so" koja se koristi za "začinjavanje" ili poboljšavanje lozinke i fajl-ključeva. Svi generisani parametri koriste se za zaštitu određenog diska – datoteke.

Slučajni podaci generisani od strane programa zasnivaju se na kretanjima miša, upotrebe tastature ali i raznih parametara vezanih za operativni sistem na kome se program koristi.

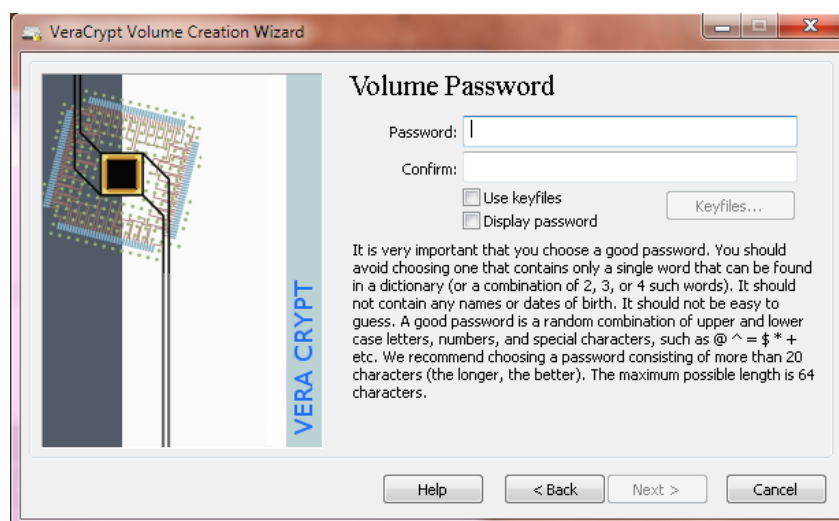
Nakon instalacije programa, potrebno je napraviti poseban fajl tj. Container koji će predstavljati skladište podataka koje želimo da zaštitimo. On se kreira kroz interfejs VeraCrypt-a i u daljem programu naziva se „volume” fajl (Slika 2.1). Tokom procesa njegovog kreiranja, bira se šifarski sistem, heš algoritam i lozinka koji će se nadalje koristiti za pristup formiranom container-u (Slika 2.2 i 2.3). Lozinka može da sadrži maksimalno 64 karaktera.



2.1 Kreiranje volume fajla



2.2 Ponudeni izbor šifarskih sistema i heš funkcija

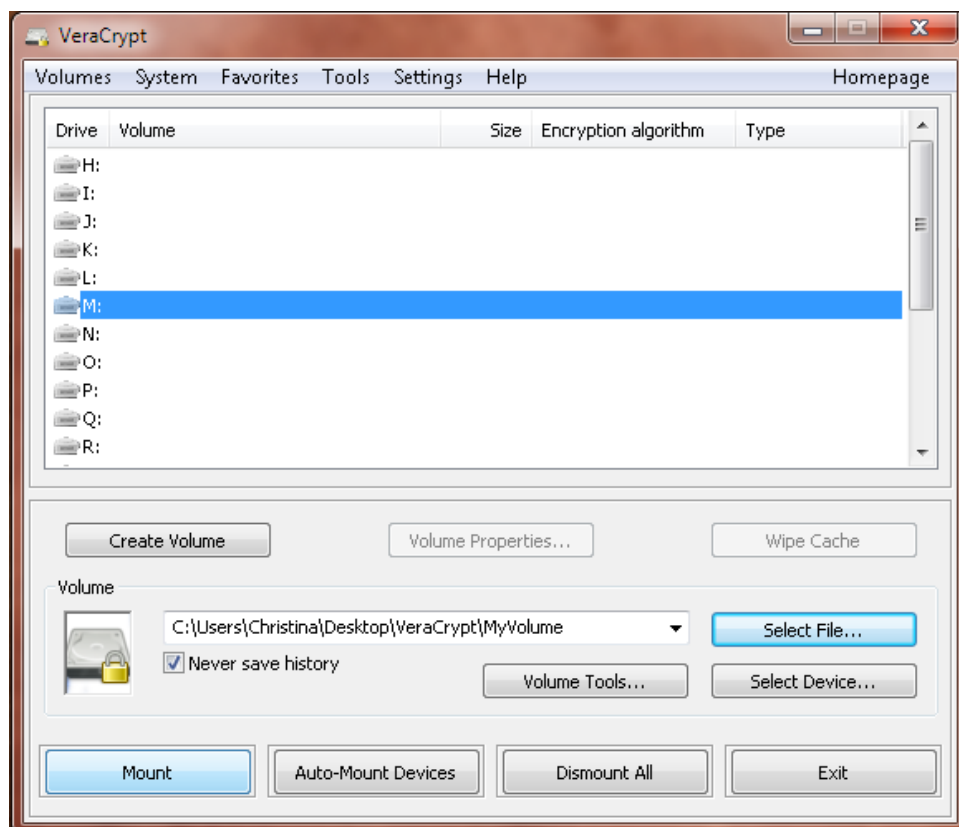


2.3 Zahtev za unos lozinke

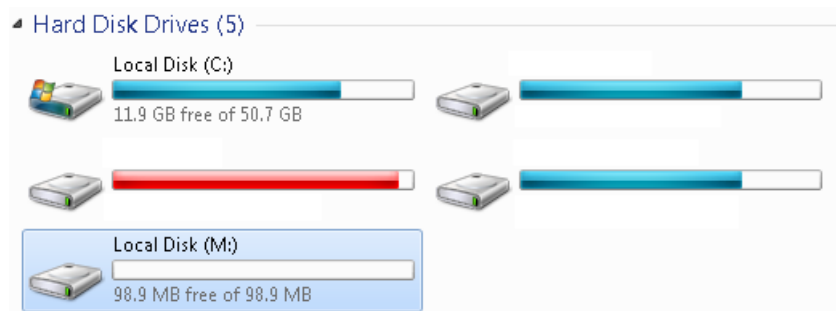
Nakon uspešne inicijalizacije volume fajla, neophodno ga je učitati kao otvoren folder na računar kome pristupamo. To se čini pomoću funkcije mount (Slika 2.4), odabira prethodno kreiranog fajla i unosa njegove već definisane lozinke.

Završetkom ovog procesa, u folderu „Computer“, pojavljuje se nov disk koji predstavlja otvorenu verziju izabranog volume fajla (Slika 2.5).

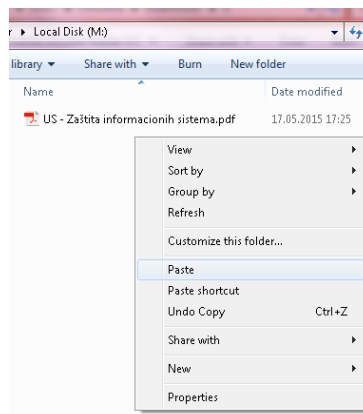
Na ovaj disk se mogu prebaciti svi fajlovi koje je potrebno zaštititi koristeći osnovne komande copy-paste (Slika 2.6).



2.4 Učitavanje volume fajla na računar

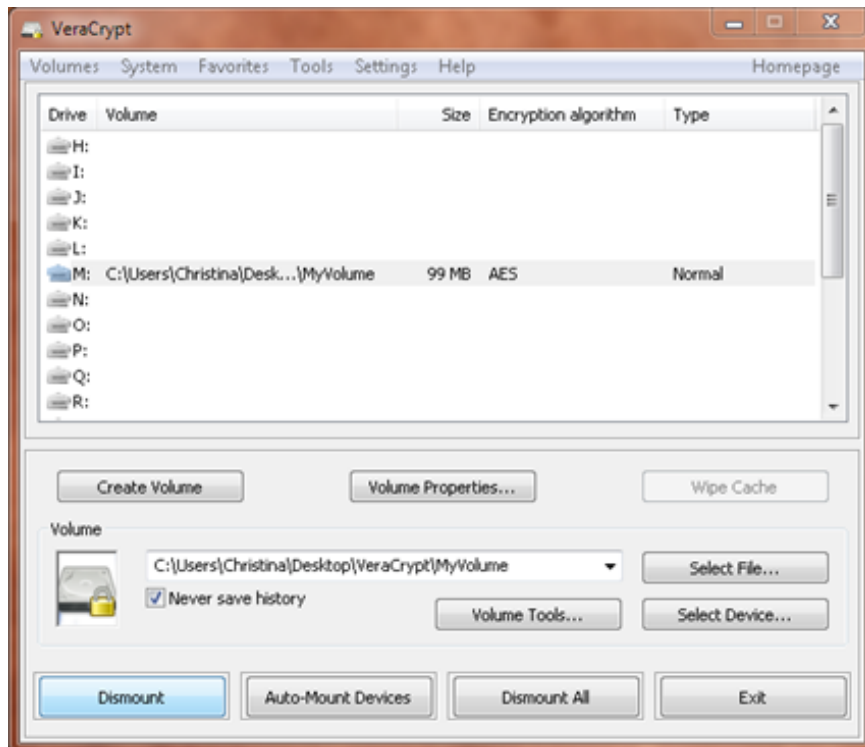


2.5 Novi disk „M:“



2.6 Komande za prebacivanje fajlova

Nakon što su podaci uspešno prebačeni, otvorena verzija volume fajla se može zatvoriti jednostavnim klikom na dugme **dismount** i pristup volume fajlu je onemogućen (Slika 2.7). Podacima koji se nalaze u šifrovanom volume fajlu nije moguće pristupiti bez korišćenja ispravne lozinke.



2.7 Zatvaranje volume fajla

Ono što ovaj program čini posebnim je takođe ugrađena funkcija koja podrazumeva *on-the-fly-encryption*. *On-the-fly-encryption* znači da se podaci automatski šifruju tačno pre njihovog čuvanja i dešifruju nakon njihovog učitavanja iz volume fajla. Ovo podrazumeva da se podaci lako mogu prebaciti u bilo koji drugi folder na računar. Međutim, njihovim pomeranjem gubi se i njihova zaštita jer su podaci zaštićeni samo dok se nalaze u volume fajlu.

Volume fajl se može čuvati na jednom mestu ili slati, bez rizika da će mu bilo ko, ko ne poseduje lozinku, neovlašćeno pristupiti. ^[2]

2.2 GNU Privacy Guard

GNU Privacy Guard je open-source softver koji predstavlja kompletnu i besplatnu implementaciju OpenPGP standarda, što znači da korisnicima pruža mogućnost zaštite elektronske komunikacije ali i skladišta podataka. Podržava rad na Windows, Linux i Mac OS operativnim sistemima. ^[1]

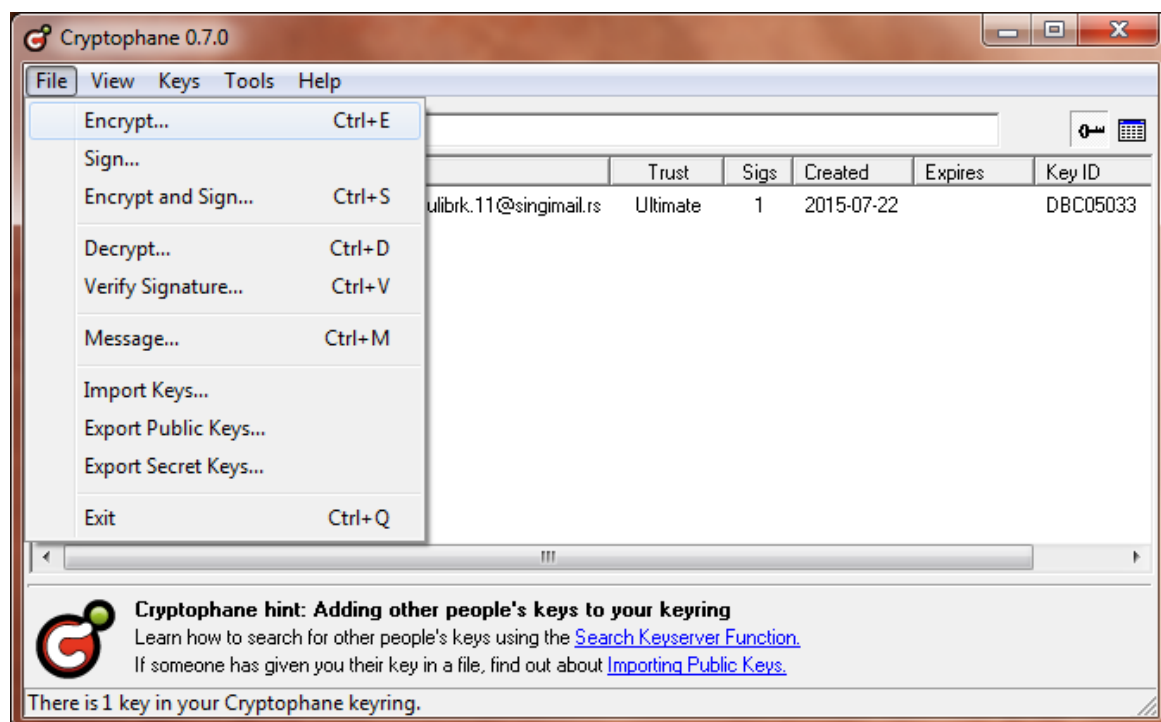
Ovaj moćan softver poseduje veliki broj funkcija koje omogućavaju: upravljanje ključevima, digitalno potpisivanje, autentifikaciju i šifrovanje

elektronske komunikacije.^[3] Međutim, u ovom radu fokus će biti stavljen na upotrebu funkcija za šifrovanje i dešifrovanje konkretnih fajlova.

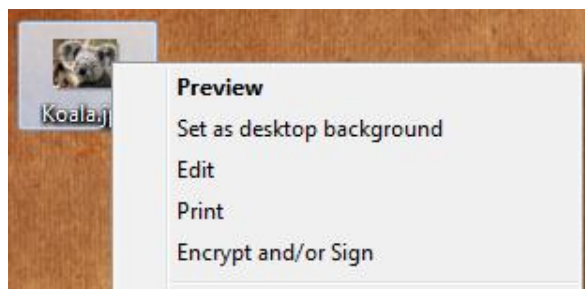
GnuPG podržava simetrične algoritme: AES, Blowfish, Twofish i Triple DES, asimetričan algoritam RSA, kao i DSA koji se koristi isključivo za digitalno potpisivanje. Takođe, ima mogućnost generisanja heš vrednosti putem SHA1, SHA256 i SHA512 algoritama.

GnuPG je program koji najveću slobodu izbora, kada je u pitanju šifrovanje podataka, daje korisniku ukoliko se koristi kroz konzolnu aplikaciju. Međutim, u ovom radu i u svrhu istraživanja, koristiće se Cryptophane - jedna od Windows grafičkih aplikacija koje je predviđena za rad sa GnuPG, a poseduje jednostavne opcije za šifrovanje i dešifrovanje podataka.^[3]

Nakon instalacije i pokretanja Cryptophane-a, funkcija za šifrovanje dostupna je klikom na File -> Encrypt posle čega se bira sam fajl koji će biti šifrovan (Slika 2.8). Takođe pokretanje postupka šifrovanja je moguće i bez otvaranja Cryptophane-a, direktno kroz Windows Explorer, desnim klikom na odabrani fajl (Slika 2.9). Bitno je napomenuti da ovaj program nema mogućnost šifrovanja više fajlova istovremeno već to čini isključivo pojedinačno.

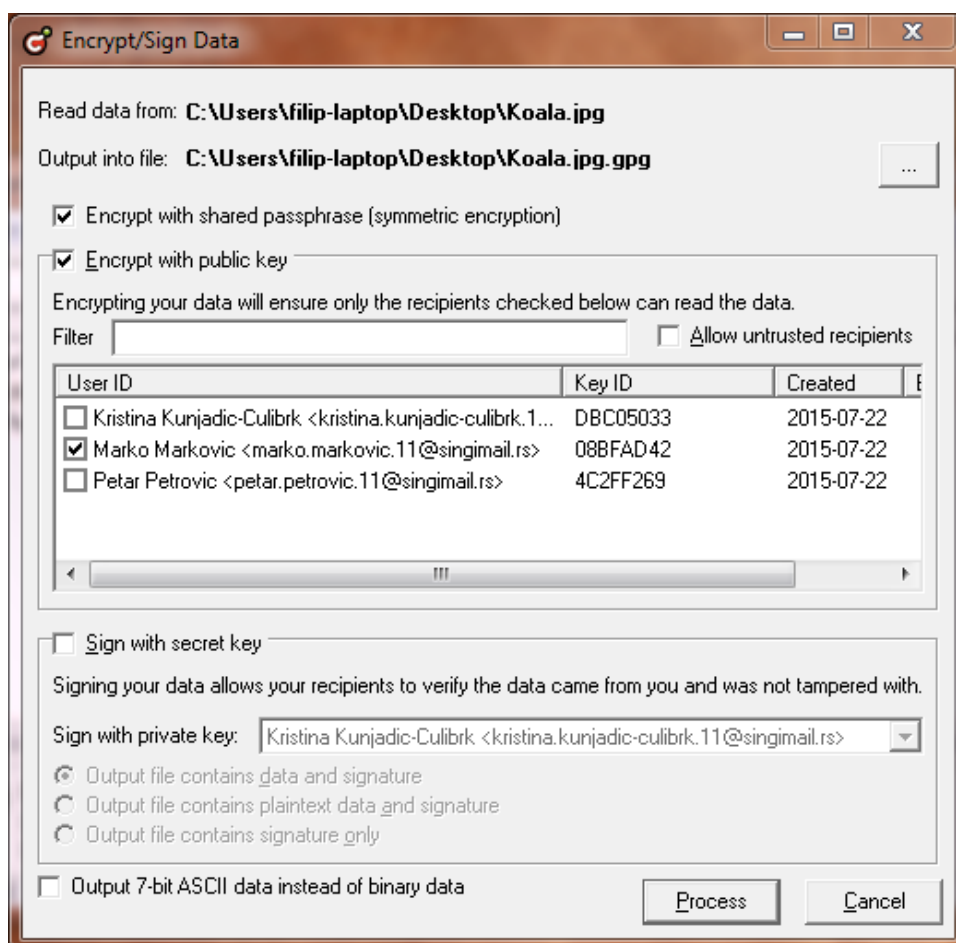


2.8 Biranje fajla za šifrovanje



2.9 Pokretanje postupka šifrovanja kroz Windows Explorer

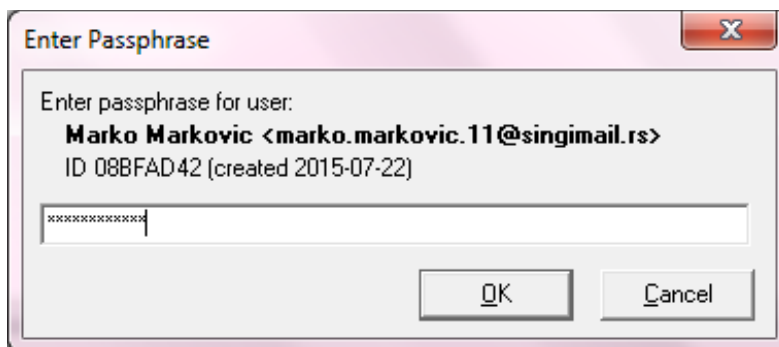
Kada su u pitanju ključevi koje će program koristiti za zaštitu podataka, Crypophane daje korisniku mogućnost izbora. U slučaju odabira simetričnog šifarskog sistema, fajl se šifrjuje lozinkom koju korisnik sam unosi, dok se u slučaju odabira asimetričnog šifarskog sistema, koristi ranije importovan javni ključ primaoca (i/ili ranije kreiran/ importovan tajni ključ pošiljaoca – u tom slučaju je već reč o digitalnom potpisivanju). Moguće je kombinovanje šifrovanja i digitalnog potpisa, što pruža visok nivo zaštite. Takođe, postoji mogućnost odabira korisnika koji će biti ovlašćeni za pristup šifratu, što znači da je neovlašćeni pristup onemogućen. U ovom slučaju korišćen je asimetričan šifarski sistem, bez upotrebe digitalnog potpisa (Slika 2.10).



2.10 Izbor šifarskog sistema koji će se koristiti u procesu šifrovanja

Generisan šifrat nosi naziv originalnog fajla kao i naziv njegove ekstenzije, s tim što se na sam kraj dodaje posebna ekstenzija „.gpg“ koja označava da je reč o fajlu šifrovanom GnuPG softverom.

Kada primalac dobije njemu namenjeni šifrat, dešifrovanje se odvija slično kao i šifrovanje - odabirom komande File->Decrypt, nakon koje sledi odabir šifrata i određivanje naziva dešifrovanog fajla. Na kraju se obavlja unošenje lozinke tajnog ključa primaoca (Slika 2.11) i time se proces dešifrovanja uspešno izvršava.

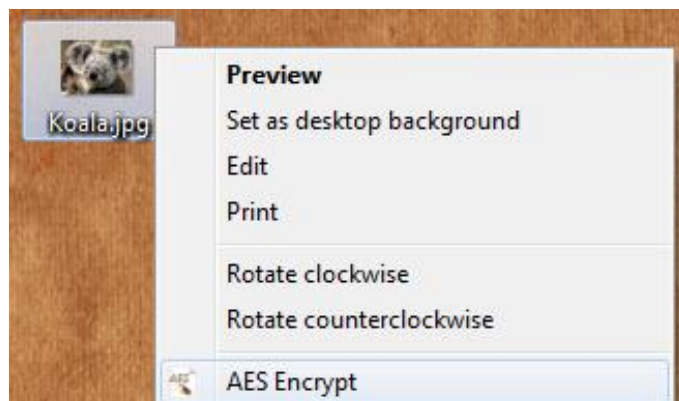


2.11 Unošenje lozinke primaoca

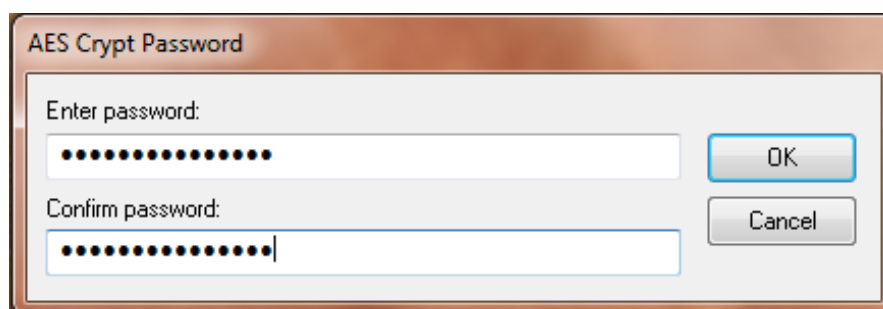
2.3 AES Crypt

AES Crypt predstavlja open-source softver za šifrovanje fajlova koji koristi simetrični AES blokovski algoritam. Dostupan je u verzijama pogodnim za korišćenje na Windows, Linux i Mac OS operativnim sistemima, ali i u obliku aplikacije pogodne za Android i iOS platforme na mobilnim uređajima (pod nazivom „Crypt4All Lite“ za Android i „AESCrypt“ za iOS). Takođe, za sve korisnike, koji žele da implementiraju u svoj program algoritme koje koristi ova aplikacija, za preuzimanje su dostupne biblioteke u Java i C# programskim jezicima.

Šifrovanje fajlova se odvija jednostavno i ne zahteva nikakvo predznanje korisnika o kriptografiji ili samom procesu šifrovanja. Nakon skidanja programa i njegove instalacije, on je integrisan u Windows Explorer i proces započinje desnim klikom na fajl koji je potrebno zaštititi i odabirom opcije „AES Encrypt“ u padajućem meniju (Slika 2.12). Nakon toga, otvara se prozoru u koji je neophodno ukucati tajnu lozinku, koju znaju samo pošiljalac i primalac šifrata, i na osnovu koje će biti generisan tajni ključ (Slika 2.13).



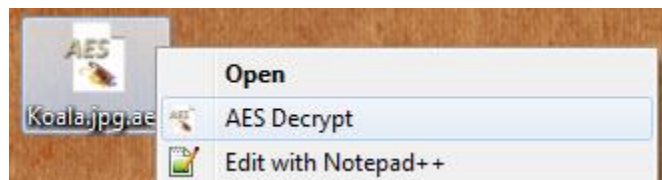
2.12 Započinjanje procesa šifrovanja koristeći AES Crypt



2.13 Unošenje lozinke

Ovaj program koristi AES-256 algoritam, što znači da se podaci šifruju u blokovima od po 128 bita u 14 rundi, koristeći ključ od 256 bita.^[4] Time se proces šifrovanja završava, i na istoj lokaciji na kojoj se nalazio fajl koji je trebalo zaštititi, pojavljuje se šifrat koji nosi naziv originalnog fajla, njegove ekstenzije i na kraju naziv njegove trenutne ekstenzije „.aes“, specifične za šifrat dobijen korišćenjem AES Crypt aplikacije. Dobro je napomenuti da ovaj program takođe nudi i istovremeno šifrovanje više fajlova - time što se na samom početku selektuje više fajlova i tek onda izvrši desni klik i ostatak procesa.

Nakon što primalac dobije šifrat, može ga dešifrovati prolaskom kroz sličan proces. Desnim klikom na šifrat, odabirom opcije „AES Decrypt“ (Slika 2.14) i unošenjem ispravne tajne lozinke, na istoj lokaciji pojavljuje se otvoreni fajl kome se može neometano pristupiti.

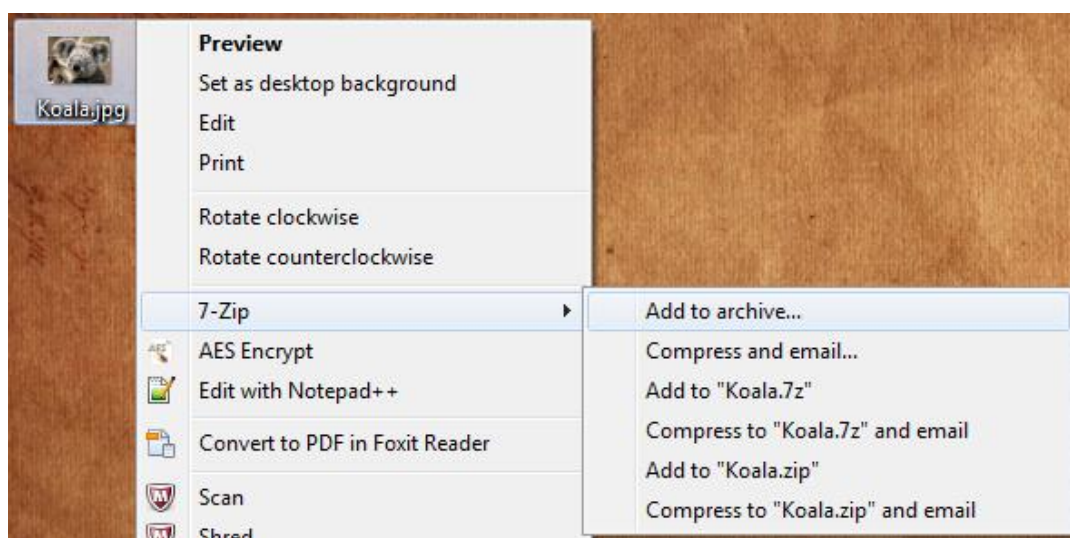


2.14 Započinjanje procesa dešifrovanja koristeći AES Crypt

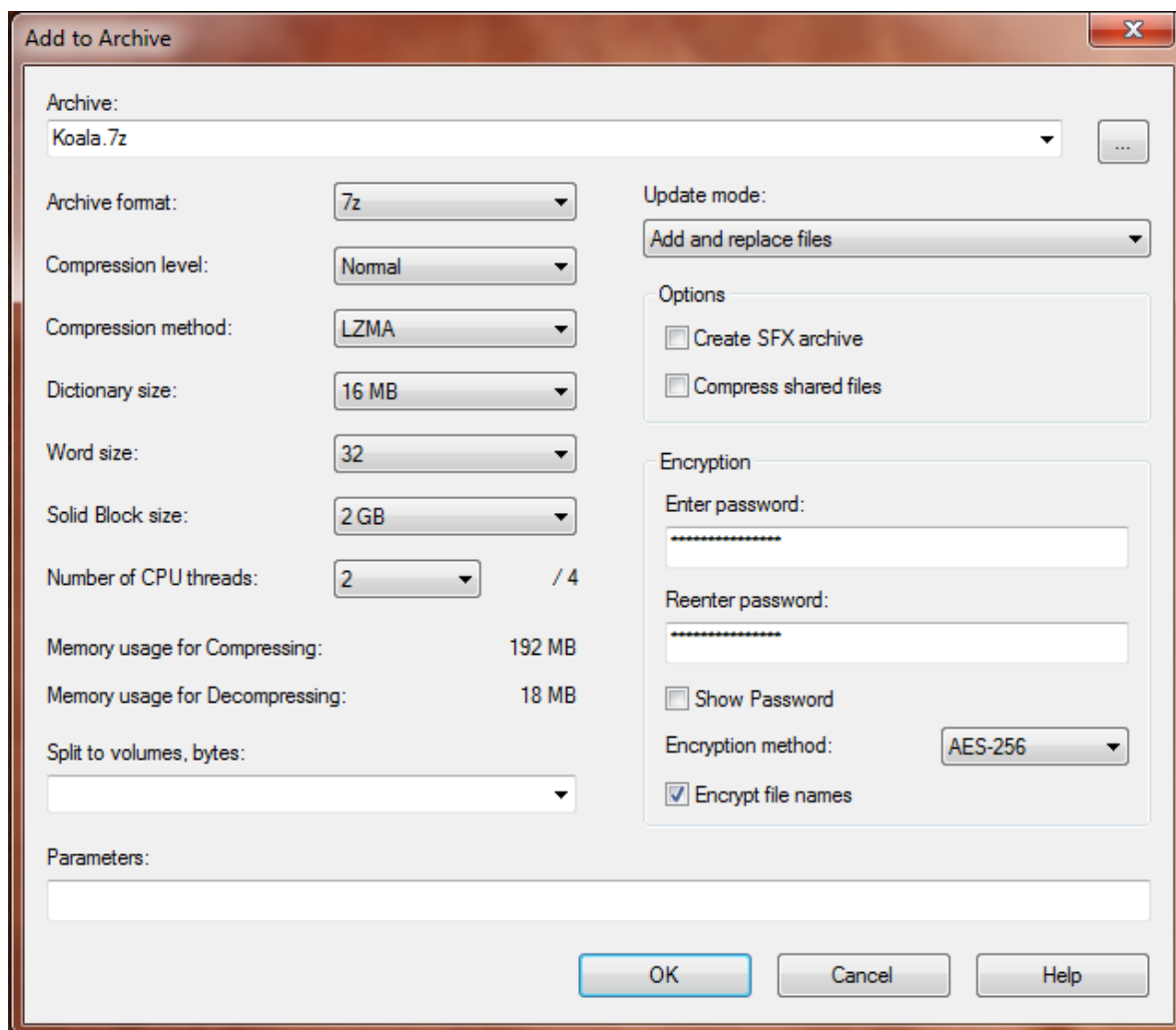
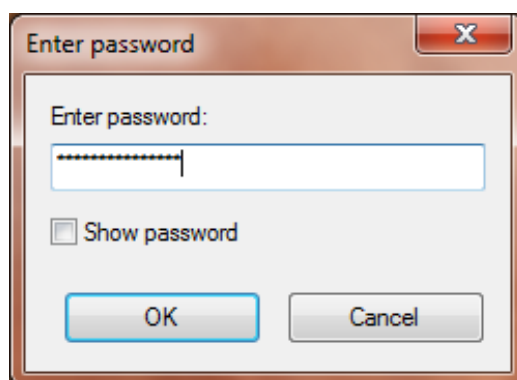
2.4 7-zip

7-zip predstavlja open-sorce softver koji primarno služi za kompresovanje fajlova ali poseduje i ugrađenu funkciju koja daje mogućnost šifrovanja fajlova. Iako zaštita fajlova nije glavna funkcija ovog program, on se sve više koristi u ovu svrhu, pa ipak spada u jedan od najzastupljenijih softvera za koji se korisnici odlučuju kada žele da zaštite svoje podatke. Većina izvornog koda je pod GNU LGPL licencom (GNU Lesser General Public License). Program je zvanično dostupan u obliku konzolne i grafičke aplikacije na 32-bitnom i 64-bitnom Windows operativnom sistemu, ali postoje i nezvaničene verzije pogodne za Linux i Mac OS operativne sisteme.^[1]

Nakon instalacije, 7-zip je dostupan kroz Windows Explorer i pokreće se odabirom fajlova koji se štite i desnim klikom miša (Slika 2.15). U padajućem meniju odabira se opcija "7-zip->Add to archive.." nakon koje se pojavljuje prozor za podešavanje opcija šifrovanja (Slika 2.16). Iako korisnik ima mogućnost da izabere jedan od ponuđenih formata arhive koja se dobija na kraju procesa, jedino "zip" i "7z" formati nude opciju šifrovanja podataka. Oba formata nude korišćenje AES-256 algoritma, dok "zip" nudi i stariju metodu "ZipCrypto" koja se smatra slabijom zaštitom pa se zato neće razmatrati u ovom radu. Kao što je već ranije pomenuto, koristeći AES-256 algoritam, podaci se šifruju u blokovima od po 128 bita u 14 rundi, dok je ključ veličine 256 bita.^[5] Ključ se generiše na osnovu unete tajne lozinke, koja se takođe zahteva i pri pokušaju dešifrovanja datoteke (Slika 2.17). Nakon završetka procesa šifrovanja, na istoj lokaciji se pojavljuje datoteka u ranije odabranom formatu. Bitno je napomenuti da bilo koji fajlovi dodati u zaštićenu datoteku nakon procesa šifrovanja, neće biti zaštićeni na isti način, pa se u tom slučaju savetuje ponavljanje postupka šifrovanja sa prethodim i dodatnim fajlovima koje je potrebno zaštititi. Na ovaj način fajlovi su sigurni i nije im moguće neovlašćeno pristupiti.



2.15 Pokretanje 7-zip aplikacije nad fajlom koji se šifruje

**2.16 Prozor za podešavanje opcija šifrovanja****2.17 Unošenje tajne lozinke**

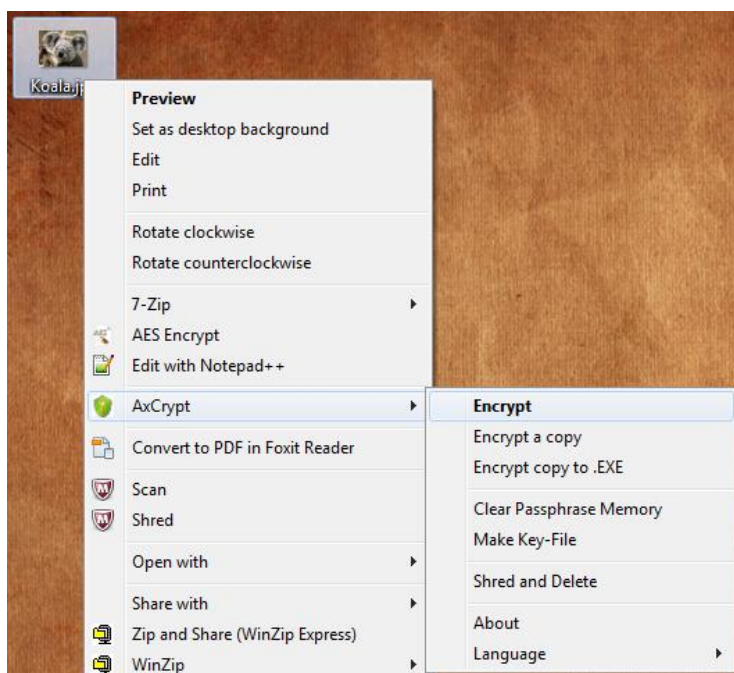
2.5 AxCrypt

AxCrypt je besplatan open-source alat za šifrovanje podataka. Dostupan je na 32-bitnim i 64-bitnim Windows operativnim sistemima na 12 jezika i licenciran je od strane GNU GPL (GNU General Public License). Takođe, javno je dostupan i izvorni kod programa u C++ i C# programskom jeziku kao i specifikacije korišćenog algoritma. Dosta godina u nazad ova aplikacija drži jedno od pet vodećih mesta kada su u pitanju aplikacije za šifrovanje podataka, pa je zato svakako zaslužila svoje mesto i na ovoj listi.^[1]

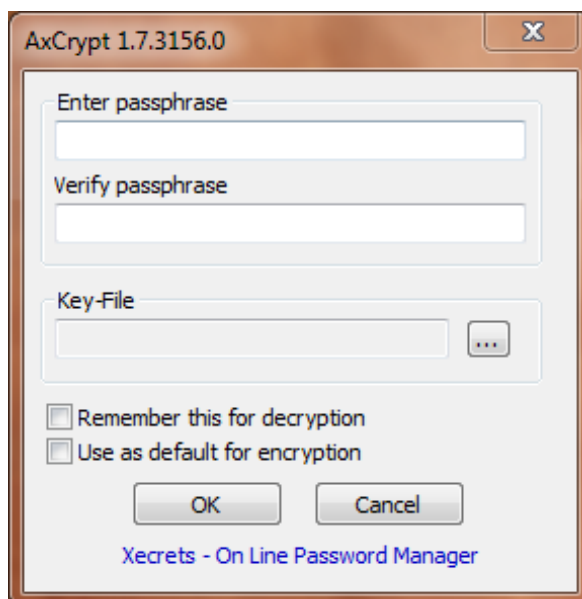
Kao i neke od već pomenutih aplikacija, AxCrypt se takođe nakon instalacije automatski integriše u Windows Explorer i može se pokrenuti desnim klikom na fajl ili folder koji je potrebno zaštititi. Važno je napomenuti da se u slučaju šifrovanja foldera za izlaz procesa ne dobija jedan šifrovan fajl, već se svi fajlovi unutar tog foldera šifruju pojedinačno. Nakon desnog klika i odabira AxCrypt stavke, korisniku je dostupno više opcija : šifrovanje, šifrovanje kopije fajla i šifrovanje kopije u „.exe“ format , dok prve dve opcije za rezultat imaju fajl sa „.axx“ ekstenzijom. U ovom slučaju biće odabrana funkcija Encrypt (Slika 2.18).

U prozoru, koji se otvara korisniku, pruža se izbor šifrovanja lozinkom ili fajl-ključem, ali i kombinacija te dve opcije koja predstavlja jaču i stabilniju zaštitu (Slika 2.19).

U slučaju odabira lozinke preporučuje se unos od minimum 5 karaktera, ali je preporučeno uneti 22 ili više nepovezanih velikih i malih karaktera u kombinaciji sa brojevima kako bi se dostigla maksimalna sigurnost korišćenjem ključa od 128 bita.^[6]



2.18 Pokretanje AxCrypt alata kroz Windows Explorer



2.19 Izbor šifrovanja lozinkom i/ili fajl-ključem

Kada je u pitanju zaštita fajl-ključem u obzir dolazi bilo koji tip fajla, dok je kroz AxCrypt dostupno i generisanje ovakvog tipa ključa. Pri procesu generisanja fajl-ključa koristi se PRNG (The pseudo random number generator) zajedno sa SHA1 kao heš algoritmom. Kolekcija slučajnosti korišćenih za PRNG se generiše na osnovu uobičajenih i manje uobičajenih tehnika, od kojih su neke - unosi sa tastature, pozicija miša ili određenog prozora na desktopu.

AxCrypt koristi AES-128 algoritam za šifrovanje i SHA-1 haš algoritam za sažimanje i verifikaciju podataka. Ključ je dužine 128 bita, a inicijalizacioni vektor predstavlja nasumičan (random) parametar u ovom procesu.

U slučaju odabira kombinacije obe od ove dve opcije, upotrebe lozinke i fajl-ključa, ova dva podatka se hešuju SHA1 algoritmom, i tek nakon toga se koriste za generisanje tajnog ključa koji se koristi za šifrovanje i dešifrovanje.

Proces dešifrovanja zahteva od korisnika lozinku i/ili fajl-ključ, i ispravnim unosom traženih parametara, omogućen je pristup otvorenim fajlovima.^[6]

3. Teorijske osnove istraživanja

U ovom poglavlju, biće predstavljeni najpoznatiji šifarski algoritmi i heš funkcije, sa podebnim osvrtom na Rijndael algoritam i SHA-1 heš funkciju.

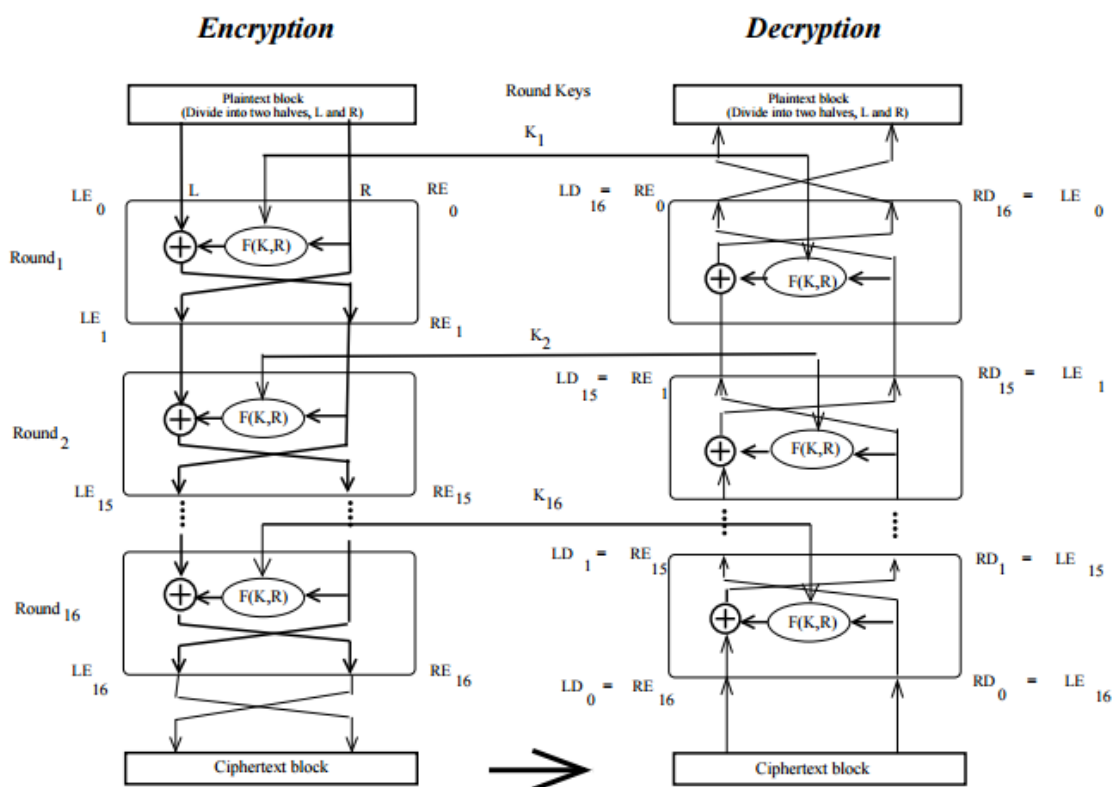
3.1 Pregled simetričnih šifarskih algoritama

Pregled simetričnih šifarskih algoritama predstavlja poglavlje u kome će se navesti i objasniti način funkcionisanja šifarskih sistema DES, Blowfish i AES.

3.1.1 DES

DES (Data Encryption Standard) šifra usvojena je od strane NIST-a (National Institute of Standards and Technology) 1977. godine. DES je baziran na Lucifer šifri koja je ranije razvijena od strane IBM-a. Ovaj algoritam koristi Fejstelovu šifarsku strukturu.

Fajstel struktura koristi isti bazični algoritam za šifrovanje i dešifrovanje (Slika 3.1). Algoritam se sastoji od 16 rundi procesiranja podataka. Svaka runda podrazumeva substituciju praćenu permutacijom. Početni blok, u svakoj rundi, deli se na dve polovine, levu i desnu. U svakoj rundi desna polovina ostaje nepromenjena, dok leva polovina prolazi kroz operaciju zavisnu od desne polovine bloka i ključa. Korak permutacije na kraju svake runde se sastoji od zamene desne i promenjene leve polovine, tako da u narednoj rundi leva polovina postaje desna, dok se desna tretira kao leva i prolazi kroz isti proces kao u prethodnoj rundi. Proces je reverzan, i dešifrovanje se obavlja na isti način sa razlikom da se ključevi specifični za runde koriste obrnutim redosledom.



3.1 Feistelova struktura za simetrične šifre^[7]

Sam ključ je određen sa 8 bajtova u memoriji, međutim, jedan bit u svakom bajtu predstavlja kontrolni bit, što dovodi do ključa sa efektivnom dužinom od 56 bita. Specifični za DES šifru su primena Feistelove strukture i izvođenje ključa za određenu rundu, dužine 48 bita, iz glavnog tajnog ključa.

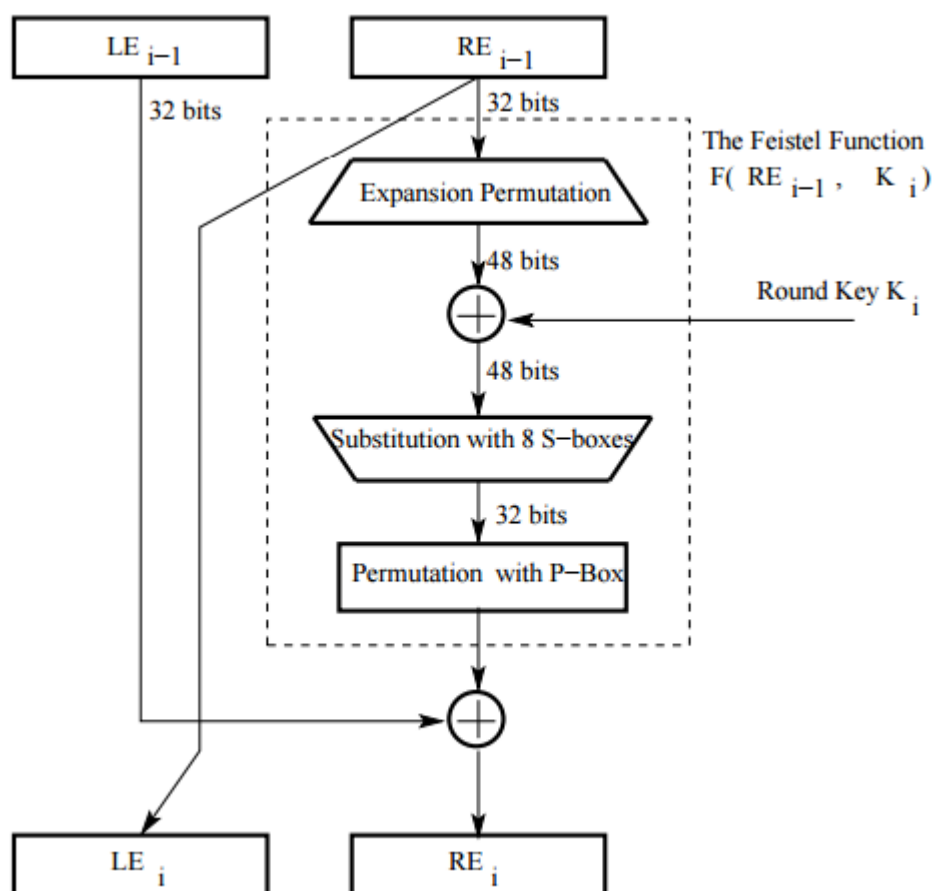
Izvođenje ključa počinje tako što se 56 bita ključa izlažu specifičnoj, inicijalnoj permutaciji. Na početku svake runde ovih 56 bita deli se na dve polovine od 28 bita i kružno se pomeraju na levo, svaka polovina za jedan ili dva bita, u zavisnosti od runde u kojoj se proces trenutno odvija (pomeraj od 2 bita u rundama 1, 2, 9 i 16, dok je u svim ostalim rundama pomeraj jedan bit). Za generisanje ključa specifičnog za određenu rundu, nakon prethodno navedenog procesa, vrši se konkatencija dve polovine. Nakon toga 56 bita ključa podleže specifičnoj vrsti permutacije u kojoj se za izlaz dobija 48 bita koji predstavljaju ključ za određenu rundu. Kružno pomeranje od jednog ili dva bita kao i permutacije osiguravaju da se svaki bit originalnog ključa koristi u otprilike 14 od 16 rundi ovog procesa.

Ulazni blokovi za šifrovanje su veličine 64 bita i dele se na levu i desnu polovinu. Desna polovina se proširuje na blok od 48 bita i to narednim koracima. Prvo se 32-bitni blok deli na osam 4-bitnih reči, zatim se na svaku reč sa leve strane dodaje poslednji bit prethodne reči, a na desni deo početni bit naredne reči (prva i poslednja reč se tretiraju kao susedne). Nakon toga dobijenih 48 bita spaja se sa ključem specifičnim za tu rundu uz pomoć XOR funkcije.

Rezultat prethodne operacije razbija se u osam 6-bitnih reči. Svaka 6-bitna reč prolazi kroz korak substitucije i zamenjena je 4-bitnom rečju (dobijenom na osnovu početne reči). Dobijena 32 bita prolaze kroz korak specifične permutacije u kojoj svaki bit menja svoje mesto na način definisan DES algoritmom. Dobijena sekvenca XOR-uje se sa levom polovinom početnog 32-bitnog levog bloka i postaje novi desni blok za narednu rundu. Prethodni početni 32-bitni desni blok, postaje novi levi blok u narednoj rundi. Ceo proces grafički je prikazan na Slici 3.2.

DES šifra je razbijena 1999. godine od strane Electronics Frontiers fondacije (EFF) što je rezultovalo zahtevom NIST-a da se u upotrebu uvede trostruki DES (Triple DES) koji predstavlja trostruku primenu originalnog DES algoritma.^[7]

Triple DES koristi ključ dužine 192 bita, koji se kasnije razbija u 3 ključa od po 64 bita koja će se zapravo koristiti u procesu šifrovanja. Proces se odvija identično kao i u DES algoritmu samo se ponavlja tri puta, u skladu sa nazivom algoritma. Podaci se šifruju prvim ključem, zatim dešifruju drugim i na kraju opet šifruju trećim ključem. Proces dešifrovanja je reverzan u odnosu na proces šifrovanja. Ovaj algoritam je tri puta sporiji od DES šifre ali je takođe i tri puta sigurniji.^[8]



3.2 Grafički prikaz jedne runde DES algoritma^[7]

3.1.2 Blowfish

Blowfish je simetrični, blokovski šifarski algoritam. Dizajniran je od strane američkog kriptografa Brusa Šnera (Bruce Schneier-a) 1993. godine kao brza i besplatna alternativa za već postojeće šifarske algoritme, i često se koristi kao zamena za DES algoritam. Od samog nastanka do sada ovaj algoritam se često analizirao i vremenom zaslužio poverenje, kao jedan od veoma jakih metoda algoritama za šifrovanje podataka. Ovaj algoritam je dostupan besplatno, za javnu i ličnu upotrebu, i njegovo korišćenje ne zahteva licencu.^[9]

Blowfish algoritam šifruje podatke u blokovima od po 64 bita u 16 rundi i dužina ključa može da varira od 32 do 448 bita. Za šifrovanje koristi Fejstelovu strukturu (objašnjenje u poglavlju 3.1.) i velike podključeve izvedene iz osnovnog tajnog ključa.^[10]

Generisanje ključa se odvija na početku svakog šifrovanja i dešifrovanja i tu se najviše 448 bita pretvara u ukupno 4168 bita. Prolaskom kroz 521 iteraciju generišu se pod ključevi za Blowfish algoritam.^[11]

Jedan je od najbržih blokovskih algoritama ukoliko su ključevi koje se koriste konstantni. Pogodan je za aplikacije u kojima se ključ ne menja često, jer se svaki novi ključ obrađuje pre upotrebe, i po kompleksnosti to je ekvivalentno šifrovanju 4KB teksta, što je veoma sporo u odnosu na druge blokovske šifre.^[10]

3.1.3 AES

AES (Advanced Encryption Standard) je simetrični blokovski algoritam koji služi za zaštitu podataka i koristi se u Američkoj vladi za zaštitu poverljivih informacija.

1997. godine NIST je objavio da je potrebna zamena za DES (Data Encryption Standard) algoritam koji se koristio dugo vremena, i tada već postajao ranjiv na brute-force napade. Jedan od predloženih algoritama bio je Rijndael algoritam koji je kasnije korišćen kao osnova za AES algoritam objavljen u oktobru 2000. godine.^[12] Rijndael algoritam biće detaljnije objašnjen u narednom poglavlju.

Postoje tri verzije ovog algoritma i to su: AES-128, AES-192 i AES-256. Svaka verzija šifruje i dešifruje podatke u blokovima od 128 bita koristeći dužinu ključa koja je navedena u samom nazivu algoritma, dakle redom 128, 192 i 256 bita. Ukoliko se koristi ključ dužine 128 bita, šifrovanje se odvija u 10 rundi, 192 bita - 12 rundi i 256 bita - 14 rundi. Izuzevši poslednju rundu, sve runde su identične.

Pre samog početka šifrovanja, pravi se jedna matrica podataka, zasnovana na dobijenom ključu i ona se kasnije koristi kao izvor podataka kada je potrebno iskoristiti deo ključa. Nakon toga se izvršava XOR funkcija nad nizom otvorenog teksta i prve četiri reči iz matrice ključa. Tek tada sam proces šifrovanja može da

počne. (Važno je napomenuti da se početak procesa dešifrovanja razlikuje, i da se otvoren tekst XOR-uje sa poslednje, umesto prve, četiri reči ključa.)

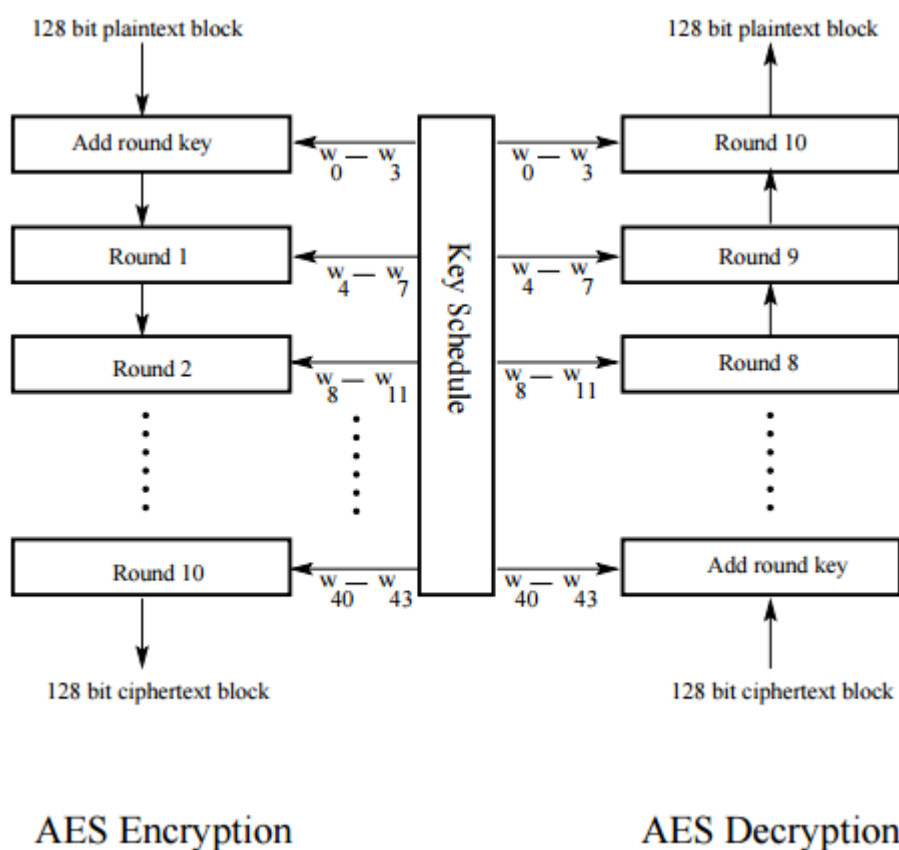
Tekst koji se šifruje ili dešifruje pretvara se u matricu podataka, i nad tom matricom odvijaju se sve runde procesa.

Svaka runda je bazirana na istim koracima: substitucija bajtova, permutacija redova, mešanje kolona i dodavanje ključa. Poslednji korak se, u procesu šifrovanja, sastoji od izvršavanja XOR funkcije nad rezultatom prethodna tri koraka i četiri reči iz matrice ključa.

Dešifrovanje se sastoji od istih koraka, ali se izvršavaju različitim redosledom: obrnuto pomeranje redova, obrnuta zamena bajtova, dodavanje ključa – XOR funkcija nad rezultatom prethodna dva koraka i četiri reči matrice ključa, i na kraju obrnuto mešanje kolona.

U poslednjoj rundi procesa šifrovanja izostavlja se korak mešanja kolona, dok se u poslednjoj rundi dešifrovanja izostavlja korak obrnutog mešanja kolona.

Priroda substitucije i permutacije u AES-u omogućava brzu softversku implementaciju algoritma.^[13]



3.3 Grafički prikaz strukture AES algoritma u slučaju korišćenja 128-bitnog ključa^[13]

3.2 Rijndael algoritam

Rijndael algoritam napravljen je od strane dva belgijska kriptologa Vincenta Rejmena i Džona Demena (eng. Vincent Rijmen, Joan Daemen). Kako je već ranije pomenuto, ovaj algoritam bio je osnova za konstruisanje AES šifre i predložen je NIST-u 1997. godine kao jedno od rešenja koje bi zamenilo DES algoritam. Nešto kasnije, odabran je kao najbolje predloženo rešenje, ali sa određenom promenom. Ova simetrična, blokovska šifra se od AES algoritma razlikuje samo po jednoj stavci, a to je opseg podržanih vrednosti za dužinu bloka i dužinu ključa. Kod Rijndael algoritma, dužina bloka i dužina ključa mogu imati nezavisne dužine sve dok ta vrednost predstavlja umnožak 32 bita, nije kraća od 128 bita i duža od 256 bita. Međutim, ovaj oblik algoritma nije vrednovan u procesu odabira osnove za AES šifru, takođe nije ni u usvojen u sadašnjem FIPS standardu. Veličina bloka je određena na 128 bita, dok veličina ključa može biti 128, 192 i 256 bita. Razvoj drugačijeg rešenja sa prvobitnom i fleksibilnijom verzijom Rijndael algoritma je moguć, ali se do sada nije pokazala potreba za takvim korakom.

Ulazne i izlazne vrednosti procesa šifrovanja i dešifrovanja su jednodimenzionalni nizovi bajtova od po 8 bita. Ulazni parametri za šifrovanje su otvoreni tekst i tajni ključ, dok se na izlazu dobija šifrat. U procesu dešifrovanja ulaz predstavljaju šifrat i tajni ključ, dok se na izlazu dobija otvoreni tekst.^[14]

Rijndael je iterativna blokovska šifra, i blokovi podataka se obrađuju u rundama posebnih transformacija. Stanje blokova podataka koji se šifruju, predstavljeni su matricom stanja, nad kojom se izvršavaju runde transformacije.

Glavni tajni ključ se proširuje specifičnim Rijndael metodama, a njegovi delovi predstavljaju potključeve koji se koriste u rundama obrade podataka. Potključ mora biti dužine jednake dužini bloka podataka pomnoženoj sa brojem rundi plus 1. Broj rundi, ne računajući poslednju dodatnu rundu, varira u zavisnosti od dužine bloka i ključa. Ukoliko su i blok podataka i ključ dužine 128 bita, obrada se odvija u 9 rundi. Ako su blok ili ključ dužine 192 bita i ni jedan element nije duži od toga, 11 rundi. I na kraju 13 rundi, ako su blok ili ključ dugački 256 bita. Proširenje originalnog ključa i korišćenje njegovih delova u procesu šifrovanja, osigurava da ključ nikada nije prikazan u celosti, čime se čuva njegova poverljivost i izbegava se mogućnost kriptanalitičkog napada na metode generisanja ključa.

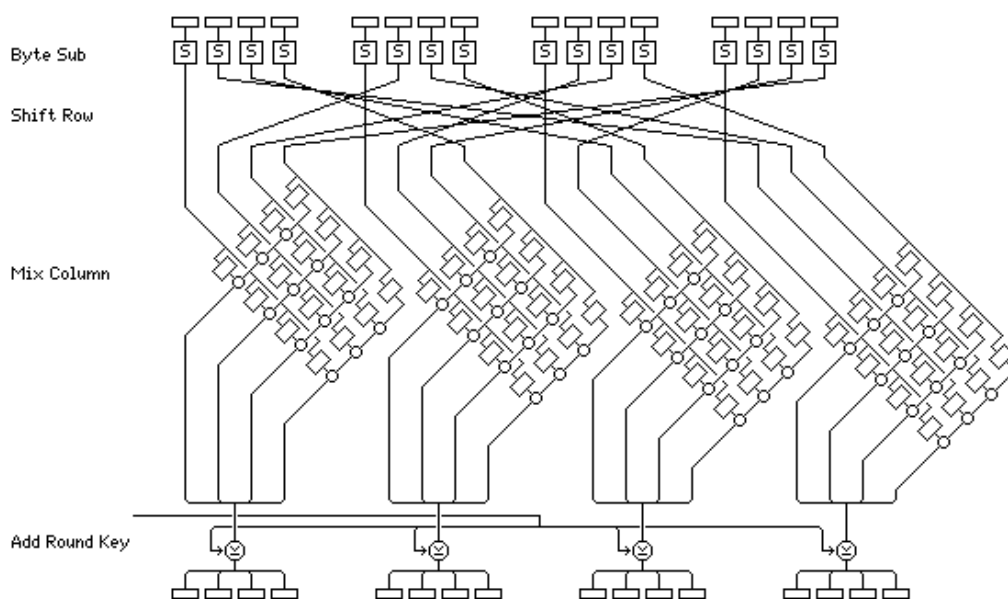
U ovom algoritmu, bajtovi nad kojima se izvršavaju matematičke operacije, tretiraju se kao polinomi a ne kao brojevi. To omogućava drugačiju i nekada lakšu implementaciju.

Šifrovanje i dešifrovanje kod Rijndael algoritma, počinje i završava sa korakom mešanja bloka podataka i potključa. Mešanje podrazumeva izvršavanje operacije XOR (eksluzivno ILI) nad ove dve vrednosti. Nakon ovog koraka, sledi izvršavanje rundi transformacije, dok se u poslednjoj rundi izostavlja korak mešanja kolona.

Svaka runda sastoji se od četiri transformacije koje se izvršavaju nad blokom podataka.

1. ByteSub funkcija je nelinearna funkcija substitucije bajtova. Substitucija se odvija na osnovu substitucijske tabele koja se drugačije naziva S kutija. Svaka promena koja se odvija nad bajtovima mapirana je unutar S kutije. Difuzija, od koje zavisi kompleksnost kriptografskog napada na algoritam, zavisi od broja aktivnih S kutija. Takođe, kompleksnost napada zavisi i od veze između ulaza i izlaza pojedinačne S kutije.
2. Shift Row funkcija pomera (rotira ulevo) bajtove u redovima matrice stanja i to svaki put sa drugačijim pomerajem. Prvi red se ne pomera, a pomeraj u ostalim redovima zavisi od dužine bloka podataka.
3. MixColumn funkcija omogućava interakciju bajtova u matrici stanja. Svaki bajt u koloni utiče na svaki drugi bajt, i to upotrebom različitih matematičkih operacija gde se bajtovi posmatraju kao polinomi. Funkcije ShiftRow i MixColumn su koraci koji osiguravaju da svi delovi bloka podataka utiču jedni na druge.
4. AddRoundKey funkcija generiše novi potključ i dodaje ga matrici stanja koristeći funkciju XOR. Prve četiri reči proširenog ključa predstavljaju potključ koji će se koristiti prvoj rundi obrade, dok se naredni potključ dobija iz naredne četiri reči proširenog ključa.

Dešifrovanje ovim algoritmom se obavlja na isti način, s tim što se u rundama transformacije koriste funkcije inverzne funkcijama ByteSub, Shift Row i Mix Column, dok funkcija AddRoundKey ostaje nepromenjena.



3.4 Grafički prikaz Rijndael algoritma^[15]

Rijndael šifra je teorijski slaba na „Square attack“, ali u praksi ovaj napad nije dovoljan da bi ugrozio sigurnost algoritma. Takođe, ovaj algoritam ograničen je dešifrovanjem koje traje duže od procesa šifrovanja i zahteva više koda i ciklusa obrade, što nije pogodno za implementacije kod, na primer, smart kartica.

Uprkos bogatoj i kompleksnoj algebarskoj strukturi koju poseduje Rijndael algoritam, njegovo izvođenje zahteva vrlo malo memorije i lako se može implementirati u različite kompjuterske sisteme, pa čak i one koji su memorijski ograničeni.

Što se tiče otpornosti ovog algoritma na brute-force napad, ta osobina opisana je u jednom citatu: „Pretpostavljajući da neko može da napravi mašinu koja je sposobna da nađe DES ključ u sekundi (tj. da isproba 255 ključeva po sekundi), onda bi takvoj mašini trebalo približno 149 hiljada milijardi godina da razbije 128-bitni AES ključ. Kako bi se stvorio pojam o toliko dugom periodu, treba navesti činjenicu da se veruje da je univerzum star manje od 20 milijardi godina“.^[15]

3.3 Heš funkcije

Heš funkcije su funkcije koje za ulaz uzimaju poruku proizvoljne dužine i na izlazu generišu niz fiksne dužine koji se najčešće naziva heš vrednost, heš kod ili otisak. Dužina izlaza, kod najčešće upotrebljivanih heš funkcija, varira od 160 do 512 bita, u zavisnosti od korišćenog algoritma.

Kako izlaz ovakvih funkcija zavisi od svakog bita u prosleđenoj poruci na ulazu, bilo kakva promena ulaza rezultuje potpunom promenom izlaza. Iz razloga ove funkcije su našle svoju primenu u procesu provere autentičnosti poruka iliti primljenih podataka.

Poruka kao i njena heš vrednost moraju biti zaštićeni na neki način prilikom slanja, kako im ne bi bilo moguće pristupiti ili ih eventualno izmeniti. Ukoliko se poruka i njena heš vrednost šalju potpuno nezaštićeni, na prijemnoj strani ne postoji mogućnost za autentifikaciju. Postoje različiti načini za zaštitu heš vrednosti poruke i navedeni su u nastavku teksta.

- Kada su u pitanju simetrični šifarski sistemi, jedna od opcija je konkatencija poruke i njene heš vrednosti i tek nakon toga izvršenje procesa šifrovanja. Time su oba podatka zaštićena. Primalac šifrata ima mogućnost da proveri autentičnost poruke nakon dešifrovanja, izračunavanjem heš vrednosti otvorene poruke i upoređivanjem te vrednosti sa dobijenim hešom. Heš kod obezbeđuje autentifikaciju, dok šifrovanje obezbeđuje poverljivost.
- Ukoliko poverljivost nije problem, već samo autentifikacija, postoji mogućnost šifrovanja simetričnom šifrom same heš vrednosti, dok se poruka šalje u otvorenom obliku. U tom slučaju samo onaj koji poseduje tajni ključ može saznati heš vrednost originalne poruke. Na taj način može lako saznati da li je poruka koju je dobio ista ona koja je inicijalno i poslata.
- Postoji mogućnost varijacije na prethodni način slanja heš vrednosti. Ona podrazumeva digitalno potpisivanje originalne heš vrednost privatnim ključem, kako bi primalac mogao jednostavnim dešifrovanjem javnim ključem da dođe do tražene vrednosti. Takođe u ovoj verziji slanja, poverljivost nije problem, već isključivo verodostojnost.
- Kada problem predstavljaju i poverljivost i verodostojnost, moguće je prethodni način dopuniti dodatnim šifrovanjem pre slanja i to upotrebom nekog od simetričnih šifarskih algoritama. Dakle, pošiljalac računa heš vrednost poruke i digitalno je potpisuje svojim privatnim ključem, zatim se vrši konkatencija poruke i digitalno potpisane heš vrednosti i na samom kraju, pre slanja, viši se šifrovanje tajnim ključem. Primalac mora posedovati prvo tajni ključ kako bi došao do poruke, a zatim i javni ključ pošiljaoca kako bi utvrdio njenu verodostojnost.

- Jedan od drugačijih pristupa autentifikaciji (kada poverljivost nije problem) predstavlja slanje poruke i heš koda u otvorenoj formi. U ovom slučaj pošiljalac na otvorenu poruku dodaje tajni string S i tek nakon toga računa njenu heš vrednost. Putem mreže se šalje poruka (bez stringa S) i generisani heš kod. Na ovaj način osigurano je da primalac mora posedovati tajni string S kako bi došao do iste heš vrednosti i na taj način utvrdio autentičnost dobijene poruke.
- Još jedan način koji objedinjuje rešavanje problema verodostojnosti i poverljivosti je modifikovanje prethodnog načina pripreme podataka za slanje. Ovaj način predstavlja konkatenciju poruke i generisane heš vrednosti, i njihovo zajedničko šifrovanje tajnim simetričnim ključem. Samo ona osoba koja poseduje tajni ključ i tajni string S biće u mogućnosti da pristupi podacima i utvrdi njihovu autentičnost.

Heš funkcija je sigurna za upotrebu ukoliko su ispunjena dva uslova. Prvi je da heš funkcija nije reverzna, što znači da ni na koji način nije moguće dobiti poruku na samo osnovu njene heš vrednosti. A drugi je da nije moguće naći dve poruke koje bi prolaskom kroz heš funkciju generisale istu heš vrednost. Heš vrednost sama po sebi predstavlja jedinstveni otisak.^[16]

U ovom poglavlju biće navedene i objašnjene neke od najpoznatijih i najrasprostranjenijih heš funkcija.

3.3.1 Merklova struktura

Merklova struktura je struktura koja se koristi kod heš algoritama i predložena je od strane Ralf Merkla (eng. Ralph Merkle) 1979. godine. Ova struktura predstavlja osnovu za neke od najsigurnijih heš funkcija, a to su MD5, Whirlpool i SHA serija funkcija. U nastavku teksta biće objašnjeno kako ona funkcioniše i šta podrazumeva.

Ulazna poruka se deli na N broj blokova gde je svaki blok jednake veličine i sadrži m bita. Dešava se da poslednji blok bude različite veličine od prethodnih, i u tom slučaju on se dopunjuje posebnim metodama kako bi imao istu dužinu kao i prethodni blokovi. Poslednji blok takođe podrazumeva informaciju o ukupnoj dužini poruke čija se heš vrednost računa.

Merklova struktura se sastoji od N faza, pri čemu se u svakoj fazi obrađuje m bita ulazne poruke. Svaka faza za ulaz ima 2 parametra: blok od m bita namenjen toj fazi heš funkcije i k-bitni izlaz iz prethodne faze. Kako prvoj fazi ne prethodi ni jedna druga, ona umesto k-bitnog izlaza kao ulaznog parametra dobija poseban niz od k bita koji se naziva inicijalizacioni vektor.

Funkcija f koja služi za procesiranje ulaznih podataka i generisanje izlaznog niza od k bita naziva se funkcija kompresije. Taj naziv je dobila iz razloga što je ulazni blok podataka po veličini uglavnom veći od niza na izlazu. Ova funkcija može

podrazumevati više rundi procesiranja, i to u zavisnosti od heš algoritma u kome se koristi.^[16]

3.3.2 MD5

MD5 (Message-Digest algorithm 5) je heš funkcija koja predstavlja proširenje svog prethodnika MD4 i dizajnirana je od strane Ronald-a Riverst-a 1991. godine. MD4 je dizajniran da bude brz, ali to je njegova jedina prednost u odnosu na MD5 koji se smatra znatno sigurnijim. U daljem tekstu biće objašnjen način na koji funkcioniše MD5 algoritam.

Poruka koja se prosleđuje ovoj funkciji može biti proizvoljne dužine N bita. Njena obrada kako bi se dobio izlaz od fiksnih 128 bita (MD% heš vrednost se tipično predstavlja sa 32 heksadecimalne cifre), obavlja se u 5 koraka.

1. Prosleđena poruka se dopunjuje do dužine deljive sa 512, bez poslednja 64 bita. Dopunjavanje se uvek izvršava, čak iako je dužina poruke već deljiva sa 512, dakle može se dopuniti sa najmanje jednim i najviše 512 bita. Na kraj poruke dodaje se jedan bit „1“, zatim bit „0“ koji se ponavlja sve dok se ne dosegne dužina deljiva sa 512 ali manja za 64 bita.
2. Na kraj poruke, u slobodnih 64 bita, smešta se integer broj koji predstavlja dužinu poruke pre nego što je bila dopunjena. Nakon ovog koraka dužina poruke je jednaka mnoziocu broja 512, što znači da svakih 512 bita u sebi sadrži 16 32-bitnih reči.
3. Treći korak predstavlja inicijalizaciju MD5 bafera (privremeno skladište podataka). On predstavlja mesto za 4 32-bitne reči koje će biti označene sa A, B, C i D. Svaki registar od 32 bita ima naredne početne vrednosti.

Reč A: 67452301

Reč B: EFCDAB89

Reč C: 98BADCFE

Reč D: 10325467

4. U četvrtom koraku, nad svakih 512 bita poruke izvršavaju se četiri runde obrade. Svaka runda ima istu osnovnu strukturu koraka, ali se razlikuju po upotrebi jedne od četiri unapred definisane funkcije kompresije. Kako se blokovi poruke obrađuju, tako se menjaju i vrednosti početne četiri reči A, B, C i D.
5. Izlaz MD5 funkcije predstavljaju promenjene vrednosti prethodno pomenute četiri 32-bitne reči. Sabiranjem njihovih početnih vrednosti sa njihovim trenutnim vrednostima dobija se konačni rezultat, heš vrednost poruke, koji je predstavljen sa 128 bita.

MD5 algoritam je lak za implementaciju i obezbeđuje otisak ulazne poruke.^[17] Dokazano je da nije otporan na kriptografske napade, pa iz tog razloga ne predstavlja najsigurniju opciju kada je u pitanju izbor heš algoritma. Zbog svoje brzine rada i dalje se koristi pri proveru autentičnosti većih fajlova.^[18]

3.3.3 SHA

SHA (Secure Hash Algorithm) odnosi se na naziv grupe kriptografskih heš algoritama koji su odobreni od strane NIST-a. U narednoj tabeli dat je prikaz specifikacija osnovnih algoritama ove grupe.

Algoritam	Veličina poruke (u bitima)	Veličina bloka (u bitima)	Veličina reči (u bitima)	Veličina izlazne heš vrednosti (u bitima)	Sigurnost (idealno) (u bitima)
SHA-1	$< 2^{64}$	512	32	160	80
SHA-256	$< 2^{64}$	512	32	256	128
SHA-384	$< 2^{128}$	1024	64	384	192
SHA-512	$< 2^{128}$	1024	64	512	256

Tabela 3.1 Specifikacije osnovnih SHA algoritama^[16]

- Veličina poruke predstavlja maksimalnu dužinu poruke koju algoritam može da obradi.
- Veličina bloka je veličina blokova u koje se poruka deli kako bi se podaci mogli procesirati.
- Veličina reči je broj bita koji se obrađuje nakon prijema bloka podataka.

Sigurnost se odnosi na to koliko poruka je potrebno generisati kako bi se našla poruka sa ponovljenom heš vrednošću sa verovatnoćom od 50%, kada pretpostavljamo da algoritam nema sigurnosnih propusta. Kod algoritam bez sigurnosnih propusta čija dužina proizvedene heš vrednosti je n bita, u slučaju dobre bezbednosti broj poruka bi morao da bude $2^{n/2}$.^[16]

3.4 SHA-1

Funkcija iz grupe SHA algoritama koja se najviše koristi je ubedljivo SHA-1 funkcija. Iako bi njegova bezbednost teoretski mogla biti veća, on se i dalje koristi u aplikacijama i najpoznatijim protokolima kao što su SSL/TLS, PGP, SSH, S/MIME i IPsec. SHA-1 je naslednik dugo korišćene heš funkcije MD5 koja se i dalje koristi u velikom broju aplikacija kao funkcija za izračunavanje heš vrednosti. Način na koji ovaj algoritam funkcioniše, biće objašnjen u nastavku teksta.

Pre početka izračunavanja heš vrednosti poruke, početna heš vrednost $H^{(0)}$ koja će biti korišćena u daljem procesu setuje se na unapred određene vrednosti. Kako je izlazna heš vrednost dužine 160 bita, ona se u ovom slučaju predstavlja sa pet 32-bitnih reči.

$$H_0^{(0)} = 67452301$$

$$H_1^{(0)} = \text{efcdab89}$$

$$H_2^{(0)} = 98badcfe$$

$$H_3^{(0)} = 10325476$$

$$H_4^{(0)} = \text{c3d2e1f0}$$

Nakon toga, originalna poruka se proširuje dodatnim bitima, na isti način kao što se to čini u MD5 algoritmu. (poglavlje 3.3.2 - koraci 1. i 2.)

SHA-1 algoritam koristi više logičkih funkcija, f_t (pri čemu t predstavlja t -tu 32-bitnu reč iz trenutno obrađivanog bloka). Dakle, u zavisnosti od mesta reči u bloku, na nju se primenjuje unapred definisana funkcija. Svaka funkcija radi sa tri 32-bitne reči, i na izlazu daje jednu 32-bitnu reč. Funkcije su definisane u nastavku.

$$f_t = \begin{cases} Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) & 0 \leq t \leq 19 \\ Parity(x, y, z) = x \oplus y \oplus z & 20 \leq t \leq 39 \\ Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \leq t \leq 59 \\ Parity(x, y, z) = x \oplus y \oplus z & 60 \leq t \leq 79 \end{cases}$$

Operacije predstavljene znakovima funkcionišu na nivou bita.

$\neg$ - komplementna operacija NOT

$\wedge$ - operacija AND

$\oplus$ - operacija XOR

Takođe kao konstante SHA-1 algoritam koristi osamdeset 32-bitnih reči $K_0, K_1, K_2, \dots, K_{79}$ čije vrednosti su date kroz K_t .

$$K_t = \begin{cases} 5a827999 & 0 \leq t \leq 19 \\ 6ed9eba1 & 20 \leq t \leq 39 \\ 8f1bbcdc & 40 \leq t \leq 59 \\ ca62c1d6 & 60 \leq t \leq 79 \end{cases}$$

Kako bi proces bio detaljno objašnjen, potrebno je prvo definisati određene funkcije i simbole.

+ - Sabiranje po modulu 2^{32} .

($x + y$, gde x i y predstavljaju celobrojne vrednosti X i Y , i to u rasponima $0 \leq X < 2^{32}$ i $0 \leq Y < 2^{32}$. Rezultat operacije sabiranja po modulu 2^{32} je „ $Z = (X+Y) \bmod 2^{32}$ “, dakle ostatak pri deljenju zbira X i Y sa 2^{32}). Z celobrojna vrednost se pretvara u reč z , predstavlja rezultat operacije – $z = x + y$)

<< - Operacija pomeranja u levo ($x \ll n$, sa levog kraja x uklanja se n bita, a zatim se na desnu stranu x , dodaje n bita „0“).

>> - Operacija pomeranja u desno ($x \gg n$, sa desnog kraja x uklanja se n bita, a zatim se na levu stranu x , dodaje n bita „0“).

$\vee$ - Operacija OR.

$ROTL^n(x)$ - Operacija rotacije (pomeraja) u levo (x je reč od w bita, a n je celobrojna vrednost u rasponu $0 \leq n < w$, i funkcija je definisana izrazom

$ROTL^n(x) = (x \gg n) \vee (x \gg w - n)$. Ova funkcija je jednaka kružnom pomeraju bita u levo, u reči sa w bita za n pozicija.)

$H^{(i)}$ - i -ta heš vrednost ($H^{(0)}$ je početna, a $H^{(N)}$ je završna heš vrednost).

$H_j^{(i)}$ - j -ta reč i -te heš vrednosti ($H_0^{(i)}$ je prva reč sa levog kraja heš vrednosti i).

M - Poruka čija se heš vrednost računa.

$M^{(i)}$ - i -ti blok poruke, veličine m bita.

$M_j^{(i)}$ - j -ta reč i -tog bloka poruke. ($M_0^{(i)}$ je prva reč sa levog kraja i -tog bloka poruke)

N - Broj blokova proširene poruke.

T - Privremena reč od w bita.

w - Broj bita u reči.

W_t - t -ta reč od w bita

Svaki blok poruke $M^{(1)}, M^{(2)}, \dots, M^{(N)}$ redom prolazi kroz naredne korake obrade.

For $i=1$ to N :

{

1. Priprema rasporeda funkcija koje će se primenjivati na rečima.

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \leq t \leq 79 \end{cases}$$

2. Inicijalizacija pet promenljivih koje će se koristiti u računanju heš vrednosti.

$$a = H_0^{(i-1)}$$

$$b = H_1^{(i-1)}$$

$$c = H_2^{(i-1)}$$

$$d = H_3^{(i-1)}$$

$$e = H_4^{(i-1)}$$

3. For $t=0$ to 79:

{

$$T = ROTL^5(a) + f_t(b, c, d) + e + K_t + W_t$$

$$e = d$$

$$d = c$$

$$c = ROTL^{30}(b)$$

$$b = a$$

$$a = T$$

}

4. Računanje i -te među vrednosti $H^{(i)}$

$$H_0^{(i)} = a + H_0^{(i-1)}$$

$$H_1^{(i)} = a + H_1^{(i-1)}$$

$$H_2^{(i)} = a + H_2^{(i-1)}$$

$$H_3^{(i)} = a + H_3^{(i-1)}$$

$$H_4^{(i)} = a + H_4^{(i-1)}$$

}

Nakon ponavljanja navedena četiri koraka N puta (tj. nakon obrade poslednjeg bloka poruke), 160-bitna heš vrednost se dobija konkatencijom $H^{(N)}$ vrednosti $(H_0^{(N)}, H_1^{(N)}, H_2^{(N)}, H_3^{(N)}, H_4^{(N)})$.^[19]

SHA-1 algoritam razbijen 2005. godine. Tada je dokazano da je moguće naići na dve iste heš vrednosti u prostoru veličine od 2^{69} bita što je mnogo manje u odnosu na ranije naveden nivo sigurnosti od 2^{80} koji je povezan sa ovom heš funkcijom.^[16]

4. Pregled predloženog rešenja

Autorska aplikacija, koja je i tema rada, biće predstavljena u ovom poglavlju. Kao što je već ranije napomenuto, aplikacija je napravljena u softverskom alatu Microsoft Visual Studio, dok je kod napisan u C# programskom jeziku.

Za šifrovanje je korišćena Rijndael šifra sa jednakom dužinom ključa i bloka podataka koji se šifruju, a to je 128 bita. Za izračunavanje heš vrednosti otvorenog fajla, pre šifrovanja i nakon dešifrovanja, korišćen je SHA-1 heš algoritam.

4.1 Uputstvo za upotrebu

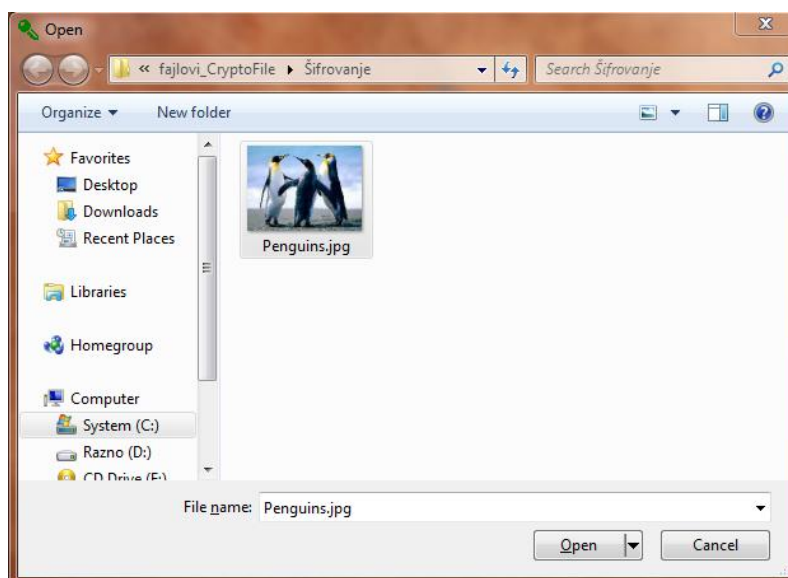
Pre pokretanja programa neophodno je na C disku, napraviti folder sa nazivom „fajlovi_CryptoFile“ i pod-folderima „Šifrovanje“ i „Dešifrovanje“. U folder „Šifrovanje“ smeštaju se podaci koje će program šifrovati, dok se u folder „Dešifrovanje“ smeštaju podaci koje će program dešifrovati. Fajlovi, za neku od ove dve operacije, mogu da se biraju isključivo iz ova dva foldera, i to u zavisnosti od funkcije koja se izvršava. Svrha pravljenja ovih foldera je centralizacija podataka koji će se štititi uz pomoć aplikacije, i njihovo jednostavno pronalaženje ukoliko im je potrebno pristupiti.

Nakon pokretanja aplikacije duplim klikom na „CryptoFile_byKristina.exe“ fajl, otvara se prozor prikazan na Slici 4.1.



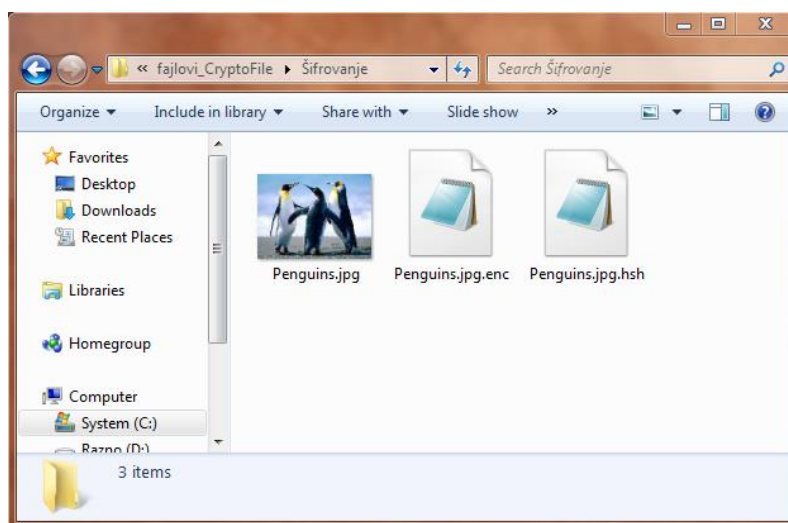
4.1 Izgled aplikacije „CryptoFile“ nakon njenog pokretanja

Pre samog početka procesa šifrovanja, neophodno je uneti ključ koji će se koristiti. Ključ mora biti dužine 8 karaktera – bilo da su u pitanju slova, brojevi ili neki od specijalnih karaktera. Nakon unosa ključa, klikom na dugme „Šifruj“ otvara se prozor u kome je potrebno izabrati fajl koji će se šifrovati (početna i fiksna lokacija sa koje se bira fajl je folder „Šifrovanje“ napravljen na početku). Prozor za odabir fajla prikazan je na Slici 4.2. Nakon odabira fajla korisnik je obavešten o uspešnom završetku procesa šifrovanja.



4.2 Prozor za odabir fajla koji će se šifrovati

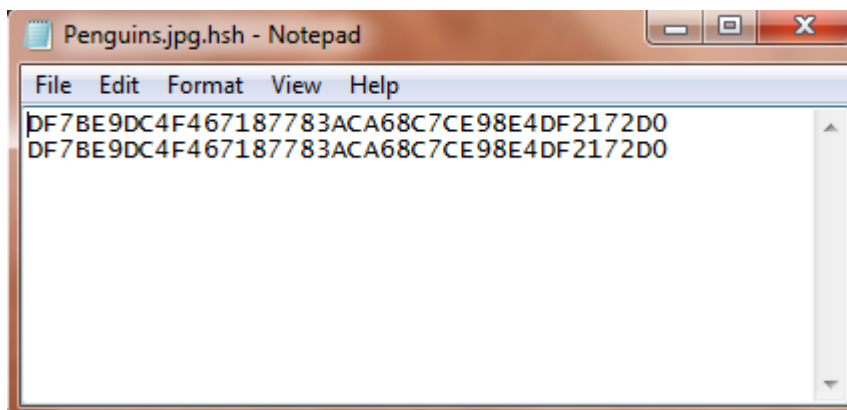
U folderu „Šifrovanje“ pored originalnog fajla, pojavljuju se dva nova fajla - jedan sa ekstenzijom „.enc“, drugi sa ekstenzijom „.hsh“ (Slika 4.3). Fajl „*.enc“ predstavlja šifrat, dok fajl „*.hsh“ predstavlja tekstualni fajl se heš vrednošću originalnog fajla pre šifrovanja.



4.3 Prikaz foldera „Šifrovanje“

U folder „Dešifrovanje“ na ovom ili nekom drugom računaru, smeštaju se zajedno: šifrat i tekstualni-heš fajl. Ponovnim ukucavanjem adekvatnog ključa u aplikaciji, klikom na dugme „Dešifruj“ i odabirom šifrata tj. „*.enc“ fajla uspešno se završava i korak dešifrovanja.

U folderu „Dešifrovanje“ sad se pojavio dešifrovani fajl, ali takođe, heš fajl „*.hsh“ je dopunjen još jednom heš vrednošću koja je izračunata na osnovu fajla dobijenog nakon dešifrovanja (Slika 4.4)



4.4 Izgled heš fajla nakon procesa dešifrovanja

Otvaranjem heš fajla i poređenjem dve ispisane heš vrednosti, vrlo lako se može utvrditi da li je dobijeni fajl isti onaj koji je prvobitno i šifrovan. Identične heš vrednosti potvrđuju autentičnost fajla, dok različite ukazuju na neovlašćen pristup šifratu ili njegovu izmenu.

4.2 Microsoft Visual Studio

Microsoft Visual Studio softverski alat proizveden je od strane kompanije Microsoft i pogodan je za rad na Windows operativnim sistemima. Predstavlja integrisano razvojno okruženje (IDE – integrated development environment) koje sadrži kompletan set alata potreban za razvoj ASP.NET web aplikacija, desktop aplikacija i mobilnih aplikacija. Neki od najpoznatijih podržanih programskih jezika su C, C++, C#, Visual Basic.NET, F#, ali takođe podržava i pisanje XML, HTML, JavaScript i CSS koda. Takođe, poseduje integrisan debager koji olakšava programiranje i uklanjanje grešaka u kodu.^[20]

Pored dosta verzija ovog programa, verzija koja je u ovom radu izabrana za razvoj aplikacije je Microsoft Visual Studio 2010. Alat je preuzet preko portala Dream Spark u sklopu studijskog programa Univerziteta Singidunum koji svim studentima daje mogućnost preuzimanja najnovijih softverskih alata.

4.3 C# programski jezik

C# je savremeni programski jezik koji je objektno orijentisan i široko rasprostranjen kada je u pitanju ovakav način programiranja. Kao posrednik između kompjutera i ovakvog tipa koda, koristi se kompajler, kako bi preveo naredbe pisane u pseudo-prirodnom jeziku, na mašinski jezik, tj. u binarni oblik.^[21]

Programski jezik C# je kompajlerski što znači da su prevođenje i izvršavanje koda dva razdvojena procesa. Specijalno je razvijen sa primarnom namenom za programiranje modularnih softverskih komponenata, pa je s toga bio pogodan kandidat i za razvoj Windows programa tj. aplikacije.^[22]

Kao i svi objektno orijentisani jezici, podržava koncepte enkapsulacije, nasleđivanja, polimorfizma. Sve metode nalaze se unutar definicija klasa. Takođe, podržava rad sa interfejsima i virtuelnim metodama.

C# omogućava razvoj sigurnih i robusnih aplikacija, ali i njihovo pokretanje u .NET Framework razvojnom okruženju. Može se koristiti za izradu Windows aplikacija, XML web servisa, klijent-server aplikacija, aplikacija baza podataka i u mnoge druge svrhe.^[23]

4.4 Pregled funkcija korišćenih u kodu aplikacije

U fajlu „Functions.cs“ koji je sastavni deo same aplikacije, navedene su dve funkcije, i to jedna za šifrovanje, a druga za dešifrovanje. Njihovim pozivanjem, izvršavaju se procesi šifrovanja i dešifrovanja, u zavisnosti od toga kakav je bio zahtev korisnika.

Funkcija za šifrovanje fajlova nosi naziv „EncryptFile“ i kao ulazne parametre prima putanju i naziv fajla koji će se šifrovati, putanju i naziv šifrata koji će se dobiti u ovoj funkciji i lozinku-ključ koji unet od strane korisnika, i koji će biti korišćen prilikom procesa šifrovanja. Ukoliko fajl koji se šifrue ne postoji, korisnik se obaveštava o neuspešnom šifrovanju.

U skladu sa standardima navedenim u udžbeniku doc. dr Saše Adamovića „Zaštita informacionih sistema“, kada se radi o šifrovanju na osnovu lozinke, u aplikaciji je primenjeno pravilo čuvanja lozinke u obliku niza karaktera umesto u standardnom obliku „string“. Razlog ovog poteza je lakše uklanjanje lozinke iz memorije računara. Kada nam lozinka više nije potrebna, jednostavnim postavljanjem svih elemenata niza na „null“ vrednost, lozinka je zauvek izbrisana iz memorije.^[24] Dakle lozinka, koja zapravo predstavlja ključ, transformiše se u niz karaktera, a nakon završetka procesa šifrovanja ili dešifrovanja, bilo da je on uspešan ili ne, ona se trajno uklanja iz memorije računara.

Glavna klasa koja se upotrebljava za proces šifrovanja, smeštena je u paketu „System.Security.Cryptography“ i zove se „RijndaelManaged“. Instanca ove klase, u nastavku programa, ima glavnu ulogu u generisanju šifrata.

Nakon uspešno generisanog šifrata, prelazi se na proces izračunavanja heš vrednosti originalnog fajla. Ova funkcionalnost se nalazi u sklopu „EncryptFile“ funkcije, ali se za njeno izvršavanje, poziva funkcija sa nazivom „GetSHA1Hash“ koja je smeštena u fajlu „Hasher.cs“ koji je takođe sastavni deo aplikacije. Ova funkcija kao glavnu klasu koristi „SHA1CryptoServiceProvider“ iz paketa „System.Security.Cryptography“. Konstruiše se instanca pomenute klase koja generiše heš vrednost uz pomoć funkcije „ComputeHash“. Kada je heš vrednost izračunata i prosleđena funkciji „EncryptFile“ ona se upisuje u tekstualni fajl sa „.hsh“ ekstenzijom, i on se smešta odmah pored generisanog šifrata. Time se proces šifrovanja uspešno završava.

Funkcija za dešifrovanje fajlova nosi naziv „DecryptFile“ i funkcioniše na sličan način kao i funkcija za šifrovanje. Koraci su: provera postojanja fajla (u ovom slučaju šifrata) i obaveštavanje korisnika o neuspešnom procesu dešifrovanja ukoliko fajl ne postoji, konvertovanje ključa u niz karaktera i upotreba instance klase „RijndaelManaged“ u svrhu dešifrovanja fajla. Kako je za proces dešifrovanja neophodno da uz šifrat stoji i „.hsh“ fajl dobijen iz procesa šifrovanja, u bilo kom slučaju neregularnosti, bilo da je to fajl koji ima više redova nego što je to predviđeno ili prosto nije priložen uz šifrat, korisnik će biti obavešten o neuspešnom završetku procesa dešifrovanja. Ukoliko program nije naišao na bilo kakvu neregularnost, nakon izračunavanja heš vrednosti dešifrovanog fajla, ona će se upisati u priložen „.hsh“ fajl, i biće moguće uporediti originalnu i izračunatu vrednost sa krajnjom dobijenom heš vrednosti. Time se proces dešifrovanja uspešno završava.

Kod objašnjenih funkcija nalazi se u dodatku, na kraju rada.

5. Zaključak

U zaključku je dat pregled ostvarenih rezultata i doprinosa razvojem sopstvenog rešenja za zaštitu podataka, kao i predlog za budući rad.

5.1 Rezultati i doprinosi

Konstruisanje, dizajniranje i izrada aplikacije koja bi bila pogodna za zaštitu poverljivih podataka je cilj koji je ovim radom ispunjen. Detaljno istraživanje tržišta, dovelo je do zaključka da zaista postoji veliki broj aplikacija koje služe u svrhu zaštite podataka, ali se uvek može postaviti pitanje „Koliko su takve aplikacije zaista bezbedne?“. Razvojem sopstvenog rešenja, uvek možemo biti sigurni da neovlašćene osobe ne mogu imati pristup procesu zaštite podataka ili na bilo koji način doći do naših poverljivih podataka. Teorijsko istraživanje najboljih simetričnih šifara i heš funkcija, koje podrazumeva njihove specifikacije i način njihovog rada, pomoglo je u procesu odabira šifarskog sistema i heš funkcije koje su najpogodnije za implementaciju. Algoritmi Rijndael i SHA-1 kao jedni od najboljih, izabrani su za osnovne algoritme i predstavljaju sastavni deo aplikacije. Algoritam Rijndael omogućava bezbedno šifrovanje fajlova, dok SHA-1 funkcija izračunavanjem heš vrednosti, pre šifrovanja i nakon dešifrovanja, dozvoljava korisniku da na taj način proverí autentičnost podataka. Bilo da ih je neko izmenio ili im neovlašćeno pristupio, poređenjem heš vrednosti to će se utvrditi.

Izrađena aplikacija, pogodna je za korišćenje u lične ali i poslovne svrhe. Jednostavno i intuitivno okruženje ne zahteva nikakvo predznanje korisnika o kriptologiji, šifarskim sistemima ili heš funkcijama, pa iz tih razloga, ovu aplikaciju može koristiti svako ko ima potrebu da zaštiti svoje podatke.

5.2 Predlog za budući rad

Aplikacija predstavljena u ovom radu je za sada prva i jedina verziju ovog programa. Predlog za budući rad bila bi naredna verzija aplikacije koja bi korisniku omogućila da bira između različitih šifarskih algoritama i heš funkcija. Aplikacija bi imala sopstveni generator ključeva, kako simetričnih tako i asimetričnih, i bila bi uvedena funkcija digitalnog potpisivanja. Aplikacija bi kreirala korisnički nalog za svakog korisnika. Pri ulasku u aplikaciju zahtevalo bi se logovanje na lični nalog pre pristupa funkcijama aplikacije. Svaki nalog bi imao bazu prethodno korišćenih ključeva, kao i mogućnost generisanja ili uvoza novih, koji bi se takođe čuvali u sklopu ličnog naloga. Kako predstavljaju jedan od važnih problema današnjice, razmena i upravljanje ključevima takođe bi bili obuhvaćeni narednim radom.

Literatura

- [1] A. Henry, "LifeHacker," 2015. [Online]. Available: http://lifelhacker.com/five-best-file-encryption-tools-5677725?commerce_insets_disclosure=off&utm_expId=66866090-48.Ej9760cOTJCPS_Bq4mjoww.1.
- [2] "Veracrypt," [Online]. Available: <https://veracrypt.codeplex.com/>.
- [3] "Tutonics," [Online]. Available: <http://www.tutonics.com/2012/11/gpg-encryption-guide-part-1.html>.
- [4] G. C. Kessler, "Aescrypt," 2013. [Online]. Available: <https://www.aescrypt.com/documentation/AES%20Crypt%20User%20Guide.pdf>.
- [5] "Northeastern," [Online]. Available: <http://www.northeastern.edu/securenu/sensitive-information-2/how-to-use-7-zip-to-encrypt-files-and-folders/>.
- [6] "Axantum," [Online]. Available: <http://www.axantum.com/AxCrypt/Security.html>.
- [7] A. Kak, "Engineering Purdue," 2015. [Online]. Available: <https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture3.pdf>.
- [8] "Vocal," [Online]. Available: <http://www.vocal.com/cryptography/tdes/>.
- [9] B. Schneier, "Schneier," [Online]. Available: <https://www.schneier.com/blowfish.html>.
- [10] "Splashdata," [Online]. Available: <http://www.splashdata.com/splashid/blowfish.htm>.
- [11] 2015. [Online]. Available: <http://iitd.vlab.co.in/?sub=66&brch=184&sim=1147&cnt=1>.
- [12] M. Rouse, 2014. [Online]. Available: <http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard>.
- [13] A. Kak, "Engineering Purdue," 2015. [Online]. Available: <https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf>.
- [14] V. R. Joan Daemen, 2001. [Online]. Available: http://jda.noekeon.org/JDA_VRI_Rijndael_2002.pdf.
- [15] 2000. [Online]. Available:

http://www.cs.mcgill.ca/~kaleigh/computers/crypto_rijndael.html.

- [16] A. Kak, "Engineering Purdue," 2015. [Online]. Available: <https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture15.pdf>.
- [17] R. L. Rivest, 1992. [Online]. Available: <https://www.ietf.org/rfc/rfc1321.txt>.
- [18] Prof. dr Mladen Veinović, M.Sc. Saša Adamović, Kriptologija I, Beograd: Univerzitet Singidunum, 2013.
- [19] 2015. [Online]. Available: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.
- [20] "msdn.microsoft," [Online]. Available: [https://msdn.microsoft.com/en-us/library/fx6bk1f4\(v=vs.90\).aspx](https://msdn.microsoft.com/en-us/library/fx6bk1f4(v=vs.90).aspx).
- [21] Prof. dr Milan Milosavljević, Prof. dr Mladen Veinović, Prof. dr Gojko Grubor, Informatika, Beograd: Univerzitet Singidunum, 2013.
- [22] Doc. dr Irina Branović, Objektno orijentisano programiranje: C++, Beograd: Univerzitet Singidunum, 2011.
- [23] "msdn.microsoft," [Online]. Available: <https://msdn.microsoft.com/en-us/en-us/library/z1zx9t92.aspx>.
- [24] Doc. dr Saša Adamović, Zaštita informacionih sistema, Beograd: Univerzitet Singidunum, 2015.

Dodatak 1

Prikaz koda funkcije „EncryptFile“, ranije pominjane u poglavlju 4.4.

```
public static void EncryptFile(string inputFile, string outputFile, string password)
{
    string hash_fajl = "";
    string uspesno = "0";

    string curFile = @inputFile;

    if (File.Exists(curFile) == true)
    {
        try
        {
            UnicodeEncoding UE = new UnicodeEncoding();
            byte[] key = UE.GetBytes(password);

            string cryptFile = outputFile;
            FileStream fsCrypt = new FileStream(cryptFile, FileMode.Create);

            RijndaelManaged RMCrypto = new RijndaelManaged();

            CryptoStream cs = new CryptoStream(fsCrypt,
                RMCrypto.CreateEncryptor(key, key),
                CryptoStreamMode.Write);

            FileStream fsIn = new FileStream(inputFile, FileMode.Open);

            int data;
            while ((data = fsIn.ReadByte()) != -1)
                cs.WriteByte((byte)data);

            fsIn.Close();
            cs.Close();
            fsCrypt.Close();

            uspesno = "1";

            try
            {
                string hash =
CryptoFile_byKristina.Hasher.GetSHA1Hash(inputFile).ToUpper();

                hash_fajl = inputFile + ".hsh";

                using (System.IO.StreamWriter file1 = new
System.IO.StreamWriter(@hash_fajl, true))
                {
                    file1.WriteLine(hash);
                    file1.Close();
                }

                uspesno = "1";
            }
        }
    }
}
```

```
        catch
        {
            uspesno = "0";
        }

        key = null;
    }
    catch
    {
        uspesno = "0";
    }

    int brojac = 0;
    string linija = "";

    try
    {
        using (StreamReader dat_reader = new StreamReader(hash_fajl))
        {
            while ((linija = dat_reader.ReadLine()) != null)
            {
                brojac++;
            }
        }

        if (brojac > 1)
        {
            uspesno = "0";
        }
    }

    catch
    {
    }

}

if (File.Exists(curFile) == false)
{
    uspesno = "0";
}

if (uspesno == "0")
{
    MessageBox.Show("Šifrovanje nije uspešno !",
        "Upozorenje !",
        MessageBoxButtons.OK,
        MessageBoxIcon.Warning);
}

if (uspesno == "1")
{
    MessageBox.Show("Šifrovanje je uspešno !",
        "Informacija !",
        MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
}
```

Dodatak 2

Prikaz koda funkcije „DecryptFile“, ranije pominjane u poglavlju 4.4.

```
public static void DecryptFile(string inputFile, string outputFile, string password)
{
    string hash_fajl = "";
    string uspesno = "0";

    string curFile = @inputFile;

    if (File.Exists(curFile) == true)
    {
        try
        {
            UnicodeEncoding UE = new UnicodeEncoding();
            byte[] key = UE.GetBytes(password);

            FileStream fsCrypt = new FileStream(inputFile, FileMode.Open);

            RijndaelManaged RMCrypto = new RijndaelManaged();

            CryptoStream cs = new CryptoStream(fsCrypt,
                RMCrypto.CreateDecryptor(key, key),
                CryptoStreamMode.Read);

            FileStream fsOut = new FileStream(outputFile, FileMode.Create);

            int data;
            while ((data = cs.ReadByte()) != -1)
                fsOut.WriteByte((byte)data);

            fsOut.Close();
            cs.Close();
            fsCrypt.Close();

            uspesno = "1";

            try
            {
                string hash =
CryptoFile_byKristina.Hasher.GetSHA1Hash(outputFile).ToUpper();

                hash_fajl = outputFile + ".hsh";

                using (System.IO.StreamWriter file1 = new
System.IO.StreamWriter(@hash_fajl, true))
                {
                    file1.WriteLine(hash);
                    file1.Close();
                }

                uspesno = "1";
            }

            catch
            {
```

```
        uspesno = "0";
    }
    key = null;
}

catch
{
    uspesno = "0";
}

int brojac = 0;
string linija = "";
string linija_1 = "";
string linija_2 = "";

try
{
    using (StreamReader dat_reader = new StreamReader(hash_fajl))
    {
        while ((linija = dat_reader.ReadLine()) != null)
        {
            if (brojac == 0)
            {
                linija_1 = linija;
            }

            if (brojac == 1)
            {
                linija_2 = linija;
            }

            brojac++;
        }
    }

    if (brojac > 2)
    {
        uspesno = "0";
    }

    if (brojac < 2)
    {
        uspesno = "0";
    }

    if (linija_1 != linija_2)
    {
        uspesno = "0";
    }
}

catch
{
    uspesno = "0";
}

if (File.Exists(curFile) == false)
{
    uspesno = "0";
}
```

```
    }

    if (uspesno == "0")
    {
        try
        {
            File.Delete(outputFile);
            File.Delete(hash_fajl);

            MessageBox.Show("Dešifrovanje nije uspešno !",
                            "Upozorenje !",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Warning);
        }
        catch
        {
            MessageBox.Show("Dešifrovanje nije uspešno !",
                            "Upozorenje !",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Warning);
        }
    }

    if (uspesno == "1")
    {
        MessageBox.Show("Dešifrovanje je uspešno !",
                        "Upozorenje !",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Warning);
    }
}
```

Dodatak 3

Prikaz koda funkcije „GetSHA1Hash“, ranije pominjane u poglavlju 4.4.

```
public static string GetSHA1Hash(string pathName)
{
    string strResult = "";
    string strHashData = "";

    byte[] arrbytHashValue;
    System.IO.FileStream oFileStream = null;

    System.Security.Cryptography.SHA1CryptoServiceProvider
oSHA1Hasher = new System.Security.Cryptography.SHA1CryptoServiceProvider();

    try
    {
        oFileStream = GetFileStream(pathName);
        arrbytHashValue = oSHA1Hasher.ComputeHash(oFileStream);
        oFileStream.Close();

        strHashData =
System.BitConverter.ToString(arrbytHashValue);
        strHashData = strHashData.Replace("-", "");
        strResult = strHashData;
    }
    catch
    {
        //System.Windows.Forms.MessageBox.Show(ex.Message,
        "Error!", System.Windows.Forms.MessageBoxButtons.OK,
        System.Windows.Forms.MessageBoxIcon.Error,
        System.Windows.Forms.MessageBoxDefaultButton.Button1);
    }

    return(strResult);
}
```