Elektrotehnički fakultet u Beogradu

Zaštita podataka

Projekat PGP

Izveštaj

Slobodan Jevtić 17/0758

Mateja Samuilović 17/0726

Sadržaj:

1. Uvod
2. Kratak opis projekta
3. Realizacija
   3.1. Osnovna ideja
   3.2. Rešenje algoritama
   3.3. GUI
   3.4. Dodatne funkcionalnosti
4. Zaključak

1.Uvod

U projektnom zadatku iz Zaštite podataka je ralizovan OpenPGP (Pretty Good Privacy).

2.Kratak opis zadatka

Zadatak se zasnivao na upotrebi DSA , ElGamal i 3DES, kao i CAST5 algoritma sve u cilju realizovana funkcionalnosit generisnanja , brisanja ključeva, uvoza i izvoz ključeva. Primanje i slanje poruke je takođe potrebno realizovati.

Takođe je očekivano da se realizuje i Grafički korinički interfejsa, kao i neophodnih pomoćnih funkcija.

3.Način realizacije rešenja

3.1 Osnovna ideja

Samo rešenje zadatka je odvojeno je u osnovi u dva dela, tako je raspoređeno i po paketima pri čemu controller paket se koristi za realizaciju algoritama, a view za grafički korisnicki interfejs.U nastavku ćemo dalje diskutovati sadržaje paketa. Cilj je bio da korisniku što bude lakše korišćenje i uviđanje funkcionalnosti.

3.2Rešenje algoritama

Deo za rad sa algoritmima za **OpenPGP** protokol je odvojen u pet klasa u podpaketu etf.openpgp.js170758dsm170726d.controller. U nastavku diskutujemo clase i interfejse ovog paketa.

Interface Pgp je interfejs sa potpisima funkcija za generisanje, birasanje importovanja i eksportovanje kljuceva. Isto je i za javni i za privatni ključ. Postoje i potpus za prijem i slanje funkcije.

Prva implementacija interfejsa je u **PgpAbstract** klasi. Ona sadrži u sebi niz abstraktnih metoda koje su za verifikaciju, enkripciju i dekripciju . Funkcije koje su ralizovane su funkcije za proveru dali je funkcija primljena ili poslata enkriptovana ,verifikovana ili ne.

```
@Override
public void receiveMessage(String fileName) {
    try {
        decryptAndVerify(fileName, fileName + ".out");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
@Override
public KeyData importKey(String fileName) {
    KeyData secretKeyData = importSecretKey(fileName);
    KeyData publicKeyData = importPublicKey(fileName);
    if(secretKeyData == null) {
        return publicKeyData;
    }
    return secretKeyData;
}
```

PgpProtocol klasa je praktično osnovan klasa rada sa celog zadatka. Ona je izvedena iz PgpAbstrakt klase. U toj klasi ima funkcije za čitanje i upis u fajlove. Generisanje , izvoženje , uvoženje i čuvanje ključa, bilo privatnog ili javnog. Takodje postoje funkcije za verifikaciju potpisa, potpisivanja i enkripcije . Niz metoda koji se zasnivanju na upotrebi klasa iz bouncycastle biblioteke.

Prvobitno je uzet Prsten kolekcije ili ti Ring.

```
public class PgpProtocol extends PgpAbstract {
    private PGPPublicKeyRingCollection publicKeyRingCollection;
    private PGPSecretKeyRingCollection secretKeyRingCollection;
    public PgpProtocol() {
        super();
        readFiles();
    }
```

Dalje tu je pomoćna funkcija za listu objekata KeyData (u nastavku dalje o njoj). Ovde je u upotrebi za čuvanje niza ključeva . Na narednoj slici je prikazan slučaj za getSecretKey() , upotreba interator za prolazak kroz listu i za dalje sortiranje. Na potpuno isti način je rešene i operacije za različite ključeve.

```
public List<KeyData> getSecretKeys() {
    List<KeyData> keyList = new LinkedList<KeyData>();
    Iterator<PGPSecretKeyRing> iter = secretKeyRingCollection.getKeyRings();
    while (iter.hasNext()) {
        PGPSecretKeyRing secretKeyRing = iter.next();
        PGPSecretKey secretKey = secretKeyRing.getSecretKey();
        String keyID = Long.toHexString(secretKey.getKeyID());
        String keyUser = secretKey.getUserIDs().next();
        KeyData data = new KeyData(keyUser, keyID, true);
        keyList.add(data);
    }

    return keyList;
}
```

U istoj klasi se generiše i par ključeva upotrebljeno klasom KeyPair.

```
public KeyData generateKeyPair(String name, String email, String password, int dsaBitLength, int elGamalBitLength) {
    KeyPair dsaKeyPair = getKeyPair("DSA", dsaBitLength);
    KeyPair elGamalKeyPair = getKeyPair("ELGAMAL", elGamalBitLength);

    try {
        PGPKeyPair pgpDsaKeyPair = new JcaPGPKeyPair(PGPPublicKey.DSA, dsaKeyPair, new Date());
        PGPKeyPair pgpElGamalKeyPair = new JcaPGPKeyPair(PGPPublicKey.ELGAMAL_ENCRYPT, elGamalKeyPair, new Date());

        PGPDigestCalculator sha1DigestCalculator = new JcaPGPDigestCalculatorProviderBuilder().build()
                .get(HashAlgorithmTags.SHA1);

        String identity = name + "<" + email + ">";
        PGPKeyRingGenerator keyRingGenerator = new PGPKeyRingGenerator(PGPSignature.POSITIVE_CERTIFICATION,
                pgpDsaKeyPair, identity, sha1DigestCalculator, null, null,
                new JcaPGPContentSignerBuilder(pgpDsaKeyPair.getPublicKey().getAlgorithm(), HashAlgorithmTags.SHA1),
                new JcePBESecretKeyEncryptorBuilder(PGPEncryptedData.CAST5, sha1DigestCalculator).setProvider("BC")
                        .build(password.toCharArray()));

        keyRingGenerator.addSubKey(pgpElGamalKeyPair);

        secretKeyRingCollection = PGPSecretKeyRingCollection.addSecretKeyRing(secretKeyRingCollection,
                keyRingGenerator.generateSecretKeyRing());
        publicKeyRingCollection = PGPPublicKeyRingCollection.addPublicKeyRing(publicKeyRingCollection,
                keyRingGenerator.generatePublicKeyRing());

        savePublicKeys();
        saveSecretKeys();

        return new KeyData(name, email, Long.toHexString(pgpDsaKeyPair.getKeyID()), true);

    } catch (PGPException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        return null;
    }

}
```

Imamo i realizaciju naredne tri metode , koje su identične i za sve bilo javni ili privatni ključ. To predstavlja čuvanje , brisanje i izvoz ključ, kao i njegov uvoz. Kasto je ranije rečeno upotrebljen je Ring za čuvanje ključeva.

```
private void saveSecretKeys() {
    try {
        OutputStream secretOut = new FileOutputStream(SECRET_KEYS_FILE);
        secretOut = new ArmoredOutputStream(secretOut);

        secretKeyRingCollection.encode(secretOut);

        secretOut.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
@Override
public void deleteKeyPair(long keyID) {
    // TODO Auto-generated method stub
    try {
        if (publicKeyRingCollection.contains(keyID)) {
            PGPPublicKeyRing keyRing = publicKeyRingCollection.getPublicKeyRing(keyID);
            publicKeyRingCollection = PGPPublicKeyRingCollection.removePublicKeyRing(publicKeyRingCollection,
                    keyRing);
            savePublicKeys();
        }
        if (secretKeyRingCollection.contains(keyID)) {
            PGPSecretKeyRing keyRing = secretKeyRingCollection.getSecretKeyRing(keyID);
            secretKeyRingCollection = PGPSecretKeyRingCollection.removeSecretKeyRing(secretKeyRingCollection,
                    keyRing);
            saveSecretKeys();
        }
    } catch (PGPException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

}
@Override
public void exportPublicKey(String fileName, long keyID) {
    // TODO Auto-generated method stub
    OutputStream publicOut;
    try {
        publicOut = new FileOutputStream(fileName + PUBLIC_KEY_APPEND);
        publicOut = new ArmoredOutputStream(publicOut);
        publicKeyRingCollection.getPublicKeyRing(keyID).encode(publicOut);
        publicOut.close();
    } catch (PGPException | IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } finally {

    }
}
```

KeyData klasa je klasa čija je uloga praktično samo za čuvanje podataka o korisniku tačnije geteri i seteri tih podataka sa dva konstruktora koji se razlikuju po raličitim argumentima.

Algoritmi koji su koriscenji

Projektnim zadtakom smo dobili algoritme za grupu 3.Među njima je dat zahtev za Algoritmima za asimetricne kljuceve DSA I ElGamal.

3DSA

U samom zadatku je dato da imamo dve varijante duzine kljuca.

Prvobitan deo algoritma se zasniva na formiranju kljuca. Prvo se formira prost broj q , koji je duzine od 160 bita, sto bi znacilo da je samim time vrednost izmedju 2159 i 2160. Nakokon toga formiramo prost broj duzine od 1024 ili 2048 zavisi od izbora korisnika

Bitan je da je q delilac p-1, dalje generisemo broj alfa. One je generisan iz Prstena u klasi PgpProtocol.

```
public class PgpProtocol extends PgpAbstract {
    private PGPPublicKeyRingCollection publicKeyRingCollection;
    private PGPSecretKeyRingCollection secretKeyRingCollection;
    public PgpProtocol() {
        super();
        readFiles();
    }
```

Svi ovi ključevi se uzimaju isli stavljaju pomoću interatora

```
List<KeyData> keyList = new LinkedList<KeyData>();
Iterator<PGPSecretKeyRing> iter = secretKeyRingCollection.getKeyRings();
while (iter.hasNext()) {
```

Neophodno je formirati random broj d koji je iz opsega 1<d<g-1, koji sae dalje koristi kao eksponent broja alfa koji se dalje koritsti u formiranju broja B.

Posle ovog postupka formiramo javni kljuc od p,q,alfa i b ; dok kao privatan kljuc ostaje broj d. Ovo je prakticno prikazano samo na strani jednog aktera.

ElGamal

Ovaj algoritam se zasniva na Diffie Helman razmeni kljuceva. U ovomo zadatku je dato da imamo izbor tri velicine kljuca. Sto se bira u :

Dalje se nakon izbora velicine kluca iamo deo formiranja kljuca y , koji se dalje razvija u uz pomoznog i privatnog dela pomocu formule $y=g^x \bmod p$, koji predstavlja javni kljuc. Vrsenje enkripcije se prsi pomocu tog dobijenog y i pomocu k-a. Enkripcija ima dva dela r i c. R dobijamo formulom $r=g^k \bmod p$, i od plaintexta m koji se dalje koristi u dobijanju dela c. Ono se dobija $c=m*y^k \bmod p$.

Potpuno jednako postupak u suprotnom smeru je i za dekriptovanje $D=C(r^{p-1-x} \bmod p)$.

Na narednoj slici je prikazano kako se uz pomoć klasa KeyPair inicijalizije koji se algorima koristi u klasi PgpProtocol.

```
public KeyData generateKeyPair(String name, String email, String password, int dsaBitLength, int elGamalBitLength) {
    KeyPair dsaKeyPair = getKeyPair("DSA", dsaBitLength);
    KeyPair elGamalKeyPair = getKeyPair("ELGAMAL", elGamalBitLength);

    try {
        PGPKeyPair pgpDsaKeyPair = new JcaPGPKeyPair(PGPPublicKey.DSA, dsaKeyPair, new Date());
        PGPKeyPair pgpElGamalKeyPair = new JcaPGPKeyPair(PGPPublicKey.ELGAMAL_ENCRYPT, elGamalKeyPair, new Date());
```

U nastavku je deo za algoritme za simetricne ključeve

DES

Ovde je upotrebljena verzija 3DES-a sa EDE om. U nastavku diskutujemo DES, zasnivanje EDE je prakticno enkripcija , dekripcija i opet enkripcija

Algoratim za enkriptovanje se odvija u dva dela, formiranje kljuceva i samo enkriptovanje poruke.

U samom delu gde se formiraju kljuceve imamo jasan postupak od koji se prakticno ponavlja u 16 ponavljanja. Prvobitno se uzima niz simbola koji predstavljaju pocetni kljuc. Njega dalje redom pretvaramo ASCII brojeve koje dalje koristimo u binarnom obliku. Nakon dobija niza binarnih brojeva vrsimo i permutaciju po tabeli. Naredni korak je odvajanje u levi i desni deo. Oba dela vrsimo šiftovanje u levo i to radimo 16 puta kako bi dobili 16 kljuceva. Za svako od kljuceba spajamo ih po odredjenom nizu kako bi i izvrsili dodatnu permutaciju i dodatno prosirimo na duzinu svakok od 48 bita. Samim time smo dobili 16 razlicitih sub kljuceva duzine od 48 bita.

Naredni deo DSA se zasniva na samoj enkripciji poruke.Ovo je blokovski algoritam tako da ce se podeliti pocetna poruka na blokove od 64 bita Prvobitno se radi pocetna permutacija po tablici. Dalje se odvaja leva i desna polovina. Poptuno se razlicito odvijaju operacije and desnoj i levoj strain.

Desna se od 32 bita razvija na 48 bita. Ona se dalje XORuje sa prvim sub ključem iz prethodno napravljenog niza. Dalje se 48 bita dele na blokove od 6 bita koji se dalje kokristi za odvajanje bitova po kojima se dalje locira odredjeni bit u tablici za svaku dalje permutaciju. Nakon čega se dalje novodobijena desna strana spaja sa pocetnom levo starnom, pri čemu početna leva strana postaje nova desna. Ovo se radi sve dok se nepromene svi ključevi od razvijenih početnih 16 subključeva. Ceo ovaj postupak je implementiran i samom projektu klasi PgpProtocol

Same dalje funkcionalnost i koji algoritam koristi bira u GUIju.

U samom zadatku su date dve varijante koje dalje za kljuceve veličine od 1024 i 2048 bita. Korisnik moze izabrati koju verziju

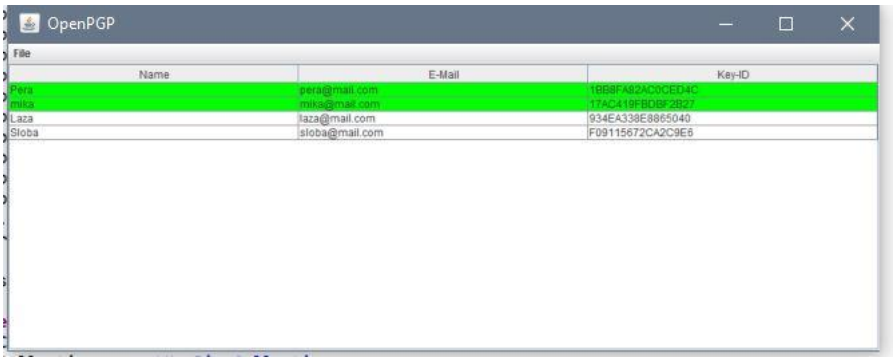
CAST 5

CAST aloritam prestavalj gurpu algoritama CAST , simetricnog plokovskog šifrovanja. Zasnovan je na istom principu koriscenjem Feistel structure koja se pojavaljuje i u drugim koderima. Izvršava se u 12 ili 16 rundki . Ovde je veryija sa 12 rudndi u kojima se poredi sa funkcijom pronalaženja narednog elementa pomoću Sboxa veličine 8 puta 32bitna sekvenca.

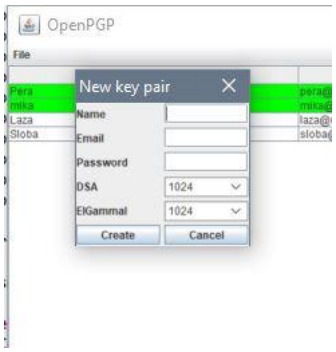
Takođe kao i DEA se koristi Addition , subtraction I Xor , slično kao u feisetl funkciji . Koristi se kao dodatak takozvana bent funkcija.

3.3 GUI

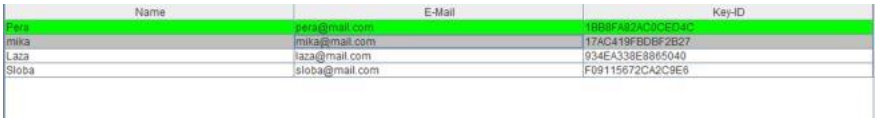
Izgled Grafičkog korisničkog interfejsa je stavljeno u pake etf.openpgp.js170758dsm170726d.view. Ovaj paket ima pet klasa MainWindow, NewKeyPair, PgpDialog, RowRederer, SendMessageDialog…



Pocetna klasa MainWindow u kojoj se prikacuje MenuBar i prikaz Table sa podacima o korisnika. Security key je obojen zelenom bojom ako je ispravan.



Takodje se odavde može pristupiti PopUp meniju desnim klikom na element tabele. Element tabele se prikazuje klasom RowRender koja prikazuje jedan redu tabeli. Klasa RowRender ima reference kao i MainWindow , reference ka PgpProtocol klasi.



Naredna su tri klase za dijalog NewKeyPariDialog,PgpDialog i SendMessageDialog . Sve tri klase se kroste za prozor koji se koristi kao pomocni prozori Parent prozora MainWindow klase. Isključivo se koriste za komunikaciju sa korisnikom.

O Grafickom interfejsu ne bi bilo potrebe dalje diskutovati jedina njegova uloga je komunikacija sa korisnikom.

3.4 Dodatne funkcionalnosti

Radi ralizacije postoje dodatne realizovane funkcionalnosti kao sto su potpisvanje poruke pomocu SHA-1 , kompresija poruke ZIPom ili konverzija podataka u pomocu radix-64.

Algoritam SHA1 koji se ovde koriste za potpis poruke. Provere za ZIP , i za radix-64 formu prikazanu u nastavku. Ovo se sve provera i realizuje pozivom funkcija iz PgpProtocola.

```
PGPLiteralDataGenerator literalDataGenerator = new PGPLiteralDataGenerator();
File actualFile = new File(outputFileName);
OutputStream literalOut = literalDataGenerator.open(compressedOut, PGPLiteralData.B
        new Date(actualFile.lastModified()), new byte[BUFFER_SIZE]);

byte[] buf = new byte[BUFFER_SIZE];
int len;
while ((len = inputStream.read(buf, 0, buf.length)) > 0) {
    literalOut.write(buf, 0, len);
    if (sign) {
        signatureGenerator.update(buf, 0, len);
    }
}
literalOut.close();
literalDataGenerator.close();
if (sign) {
    signatureGenerator.generate().encode(compressedOut);
}

compressedOut.close();
compressedDataGenerator.close();
if (encrypt) {
    encryptedOut.close();
    encryptedDataGenerator.close();
}

if (convertToRadix64)
    outputStream.close();
```

Klasa FileUtil sadrži metode za obradu fajlova. Metotode nisu vezane za samo izvšavanje algoritama. Citanje cele linije, računanje dužine linije.

Dodatak je funkcija bytesToHex koja pretvara bite u heksadecimalnu vrednost.U kojoj je upotrebljen bitviser operator I koji prolazi kroz ceo nizi dalje vraća niz heksadecemalnih vrednosti.

```
public static String bytesToHex(byte[] bytes) {
    char[] hexChars = new char[bytes.length * 2];
    for ( int j = 0; j < bytes.length; j++ ) {
        int v = bytes[j] & 0xFF;
        hexChars[j * 2] = hexArray[v >>> 4];
        hexChars[j * 2 + 1] = hexArray[v & 0x0F];
    }
    return new String(hexChars);
}
```

Kao I funckiju za kompresovanje fajlova compressFile .Korisi se klasa PGPCompressedDataGenerator kao dalje krositimo za kompresiju.

```
public static byte[] compressFile(String fileName, int algorithm) throws IOException {
    ByteArrayOutputStream bOut = new ByteArrayOutputStream();
    PGPCompressedDataGenerator comData = new PGPCompressedDataGenerator(algorithm);
    PGPUtil.writeFileToLiteralData(comData.open(bOut), PGPLiteralData.BINARY, new File(fileName));
    comData.close();
    return bOut.toByteArray();
}
```

Svi izetci su prikazani korisniku pomoću Alertboxa.

4.Zaključak

Ceo projektni zadatak je rešen na efikasan način I po uzouru na program Kleopatra. Pretpostavlja se da je ovo rešenje vrlo intuitivno za upotrebu.