UNIVERZITET U BEOGRADU

ELEKTROTEHNIČKI FAKULTET

Logo

Description automatically generated

Projekat iz predmeta Zaštita podataka

PGP protokol

Studenti:

Lazar Ristić 2015/0658

Milica Šoljaga 2013/0656

Avgust 2021.

# Uvod

U daljem odeljku biće prikazani i objašnjeni osnovni algoritmi koji su korišćeni u projektu koji simulira proces generisanja i brisanja ključeva kao i proces enkripcije i dekripcije u okviru PGP protokola. Pri izradi zadatka korišćena je Bouncycastle biblioteka.

# Generisanje ključeva

U ovoj metodi se generišu DSA i El Gamal parovi ključeva, zadate dužine, (DSA:1024/2048, El Gamal: 1024/2048/4096) kao i PGPPublicKeyRing i PGPSecretKeyRing prema specifikaciji projektnog zadatka. DSA će biti korišćen za potpisivanje a El Gamal za enkripciju.

public void ***generateDSAELGamalKeyRing***(Integer dsaKeySize, Integer elGamalKeySize, String name, String email, String passPhrase)  
 throws NoSuchProviderException, NoSuchAlgorithmException, InvalidAlgorithmParameterException, IOException, PGPException {  
 KeyPairGenerator dsaKpg = KeyPairGenerator.*getInstance*("DSA", "BC");  
 dsaKpg.initialize(dsaKeySize);  
 KeyPair dsaKp = dsaKpg.generateKeyPair();  
  
 KeyPair elgKp = generateElGamalKeyPair(elGamalKeySize);  
 PGPKeyPair dsaKeyPair = new JcaPGPKeyPair(PGPPublicKey.*DSA*, dsaKp, new Date());  
 PGPKeyPair elgKeyPair = new JcaPGPKeyPair(PGPPublicKey.*ELGAMAL\_ENCRYPT*, elgKp, new Date());  
 PGPDigestCalculator sha1Calc = new JcaPGPDigestCalculatorProviderBuilder().build().get(HashAlgorithmTags.*SHA1*);  
 PGPKeyRingGenerator keyRingGen = new PGPKeyRingGenerator(PGPSignature.*POSITIVE\_CERTIFICATION*, dsaKeyPair,  
 Utils.*getInstance*().formatNameAndEmail(name, email), sha1Calc, null, null, new JcaPGPContentSignerBuilder(dsaKeyPair.getPublicKey().getAlgorithm(), HashAlgorithmTags.*SHA1*),  
 new JcePBESecretKeyEncryptorBuilder(PGPEncryptedData.*AES\_256*, sha1Calc).setProvider("BC").build(passPhrase.toCharArray()));  
 keyRingGen.addSubKey(elgKeyPair);  
 PGPPublicKeyRing pkr = keyRingGen.generatePublicKeyRing();  
 PGPSecretKeyRing skr = keyRingGen.generateSecretKeyRing();  
 User u = new User(Utils.*getInstance*().formatNameAndEmail(name, email));  
 u.setPassword(passPhrase);  
 u.setPubKeyRing(pkr);  
 u.setSecKeyRing(skr);  
  
 KeyRingHelper.*getInstance*().savePublicKeyRing(pkr);  
 KeyRingHelper.*getInstance*().saveSecretKeyRing(skr);  
}

# Potpisivanje poruke

U nastavku je prikazan algoritam za potpisivanje poruke, koji od ulaznih parametara ima poruku koju treba da potpiše, privatni ključ onoga koji potpisuje poruku i lozinku kao dodatnu zaštitu prilikom potpisivanja. Povratna vrednost metode je potpis poruke tipa String.

private static String ***signMessageByteArray***(String message, PGPSecretKey secretKey,  
 String passPhrase) throws IOException, PGPException {  
  
 byte[] messageCharArray = message.getBytes();  
 PGPPrivateKey privateKey = secretKey.extractPrivateKey(new JcePBESecretKeyDecryptorBuilder().setProvider("BC").build(passPhrase.toCharArray()));  
 PGPSignatureGenerator signatureGenerator = new PGPSignatureGenerator(new JcaPGPContentSignerBuilder(secretKey.getPublicKey().getAlgorithm(), HashAlgorithmTags.*SHA1*).setProvider("BC"));  
 signatureGenerator.init(PGPSignature.*BINARY\_DOCUMENT*, privateKey);  
  
 ByteArrayOutputStream encOut = new ByteArrayOutputStream();  
 OutputStream out = encOut;  
 out = new ArmoredOutputStream(out);  
  
 Iterator it = secretKey.getPublicKey().getUserIDs();  
 if (it.hasNext()) {  
 PGPSignatureSubpacketGenerator spGen = new PGPSignatureSubpacketGenerator();  
 spGen.addSignerUserID(false, (String) it.next());  
 signatureGenerator.setHashedSubpackets(spGen.generate());  
 }  
 BCPGOutputStream bOut = new BCPGOutputStream(out);  
 signatureGenerator.generateOnePassVersion(false).encode(bOut);  
  
 PGPLiteralDataGenerator lGen = new PGPLiteralDataGenerator();  
 OutputStream lOut = lGen.open(bOut, PGPLiteralData.*BINARY*,  
 PGPLiteralData.*CONSOLE*, messageCharArray.length, new Date());  
 for (byte c : messageCharArray) {  
 lOut.write(c);  
 signatureGenerator.update(c);  
 }  
  
 lGen.close();  
 signatureGenerator.generate().encode(bOut);  
 out.close();  
  
 return encOut.toString();  
}

# Enkripcija

U nastavku je prikazana metoda za enkriptovanje poruke, koja kao argumente prima poruku koja se šifruje, listu javnih ključeva svih odabranih primaoca kojima se poruka šifruje, algoritam kojim se poruka šifruje (3DES/IDEA) i putanja do datoteke gde će enkriptovana poruka biti sačuvana.

public byte[] ***encryptMessageUsingSessionKey***(String message, List<PGPPublicKey> pgpPublicKeyList, int symetricAlgoritmCode, String filepath) {  
 try {  
 OutputStream out = new ArmoredOutputStream(new BufferedOutputStream(new FileOutputStream(filepath)));  
 byte[] bytes = compress(message);  
  
 PGPEncryptedDataGenerator encGen = new PGPEncryptedDataGenerator(  
 new JcePGPDataEncryptorBuilder(symetricAlgoritmCode).setWithIntegrityPacket(true).setSecureRandom(new SecureRandom()).setProvider("BC"));  
 pgpPublicKeyList.forEach(pgpPublicKey -> {  
 encGen.addMethod(new JcePublicKeyKeyEncryptionMethodGenerator(pgpPublicKey).setProvider("BC"));  
 });  
 OutputStream cOut = encGen.open(out, bytes.length);  
 cOut.write(bytes);  
 cOut.close();  
 out.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (PGPException e) {  
 e.printStackTrace();  
 }  
 return null;  
}

# Kompresija

Algoritam korišćen za kompresiju je ZIP algoritam, koji kako argument prima poruku koju kompresuje a povratna vrednost je niz bajtova kompresovane poruke.

public byte[] ***compress***(String message) throws IOException {  
 byte[] data = message.getBytes();  
 PGPCompressedDataGenerator compressGen = new PGPCompressedDataGenerator(PGPCompressedData.*ZIP*);  
 ByteArrayOutputStream bos = new ByteArrayOutputStream();  
 OutputStream compressOut = compressGen.open(bos);  
 OutputStream os =  
 new PGPLiteralDataGenerator().open(compressOut, PGPLiteralData.*BINARY*, "", data.length, new Date());  
 os.write(data);  
 os.close();  
 compressGen.close();  
 return bos.toByteArray();  
}

# Konverzija

Algoritam korišćen za konverziju je Radix64 koji čita iz fajla poruku, konvertuje je vraća u fajl već konvertovanu poruku.

public void ***conversion()*** throws IOException {  
 if(!authentication && !privacy && !compression){  
 *writeToFile*(filepath,message);  
 }  
 Path path = Paths.*get*(filepath);  
 String msg = *readFromFileIntoString*(filepath);  
 Files.*write*(path, Base64.*getEncoder*().encode(Strings.*toByteArray*(msg)));  
}

# Dekripcija

Algoritam za dekripciju čita iz datoteke poruku, tako što obrađuje deo po deo poruke, zatim pronalazi identifikator ključa ili više njih kojim je poruka šifrovana i dešifruje je privatnim ključem, uz dodatni zahtev za lozinkom ukoliko je privatni ključ zaštićen lozinkom. Uz dekripciju je urađena verifikacija i dekompresija. Rezultat metode je novi fajl sa originalnom porukom.

public static void ***decrypt***(InputStream in, JPanel mainPanel) throws IOException {  
 System.*out*.println("USAO U DECRYPT");  
 String dialogMessage = new String();  
 in = PGPUtil.*getDecoderStream*(in);  
 try {  
 JcaPGPObjectFactory pgpF = new JcaPGPObjectFactory(in);  
 PGPEncryptedDataList enc;  
 Object o = pgpF.nextObject();  
 if (o instanceof PGPEncryptedDataList) {  
 enc = (PGPEncryptedDataList) o;  
 } else {  
 enc = (PGPEncryptedDataList) pgpF.nextObject();  
 }  
 Iterator it = enc.getEncryptedDataObjects();  
 PGPPrivateKey sKey = null;  
 PGPPublicKeyEncryptedData pbe = null;  
  
 while (sKey == null && it.hasNext()) {  
 pbe = (PGPPublicKeyEncryptedData) it.next();  
 char[] myChars = new char[1];  
 myChars[0] = '"';  
 sKey = KeyRingHelper.*getInstance*().getPrivateKey(pbe.getKeyID(), myChars);  
 }  
 if (sKey == null) {  
 JOptionPane.*showMessageDialog*(null, "There is no private key for decryption");  
 return;  
 }  
  
 InputStream clear = pbe.getDataStream(new JcePublicKeyDataDecryptorFactoryBuilder().setProvider("BC").build(sKey));  
 JcaPGPObjectFactory plainFact = new JcaPGPObjectFactory(clear);  
 Object message = plainFact.nextObject();  
 JcaPGPObjectFactory pgpFact = new JcaPGPObjectFactory(in);  
 ;  
 if (message instanceof PGPCompressedData) {  
 PGPCompressedData cData = (PGPCompressedData) message;  
 pgpFact = new JcaPGPObjectFactory(cData.getDataStream());  
  
 message = pgpFact.nextObject();  
 }  
 String outFileName;  
 if (message instanceof PGPLiteralData) {  
 PGPLiteralData ld = (PGPLiteralData) message;  
 JFileChooser fileChooser = new JFileChooser();  
 fileChooser.setCurrentDirectory(new File(System.*getProperty*("user.home")));  
 fileChooser.setFileFilter(new FileNameExtensionFilter("\*.txt", "txt"));  
 int result = fileChooser.showOpenDialog(mainPanel);  
 if (result == JFileChooser.*APPROVE\_OPTION*) {  
 File selectedFile = fileChooser.getSelectedFile();  
 outFileName = selectedFile.getAbsolutePath();  
 if (!outFileName.substring(outFileName.lastIndexOf(".") + 1).equals("txt"))  
 outFileName += ".txt";  
 } else {  
 outFileName = ld.getFileName();  
 if (outFileName.length() == 0) {  
 outFileName = "defaultImeFajla";  
 }  
 }  
 InputStream unc = ld.getInputStream();  
 OutputStream fOut = new FileOutputStream(outFileName);  
  
 Streams.*pipeAll*(unc, fOut);  
  
 fOut.close();  
 } else if (message instanceof PGPOnePassSignatureList) {  
 PGPOnePassSignatureList p1 = (PGPOnePassSignatureList) message;  
  
 PGPOnePassSignature ops = p1.get(0);  
  
 PGPLiteralData p2 = (PGPLiteralData) pgpFact.nextObject();  
  
 InputStream dIn = p2.getInputStream();  
 int ch;  
 PGPPublicKey key = KeyRingHelper.*getInstance*().getPublicKey(ops.getKeyID());  
 if (key == null) {  
 JOptionPane.*showMessageDialog*(null, "Message Could not be verified, make sure you have the public key.");  
 return;  
 }  
 SimpleDateFormat sdf = new SimpleDateFormat("dd.MM.yyyy hh:mm");  
 dialogMessage = "Message signed by : " + dialogMessage + key.getUserIDs().next() + "\n Signature Created at : " + sdf.format(key.getCreationTime());  
 PGPSignatureList p3 = (PGPSignatureList) pgpFact.nextObject();  
  
 if (ops.verify(p3.get(0))) {  
 dialogMessage = dialogMessage + "\n Signature verified";  
 System.*out*.println("signature verified.");  
 } else {  
 dialogMessage = dialogMessage + "\n Signature verification failed";  
 }  
 } else {  
 throw new PGPException("message is not a simple encrypted file - type unknown.");  
 }  
 if (pbe.isIntegrityProtected()) {  
 if (!pbe.verify()) {  
 dialogMessage = dialogMessage + "\n Message failed integrity check";  
 } else {  
 dialogMessage = dialogMessage + "\n Message integrity check passed.";  
 }  
 } else {  
 dialogMessage = dialogMessage + "\n No message integrity check.";  
 }  
 JOptionPane.*showMessageDialog*(null, dialogMessage);  
 } catch (PGPException e) {  
 System.*err*.println(e);  
 if (e.getUnderlyingException() != null) {  
 e.getUnderlyingException().printStackTrace();  
 }  
 }  
}

# Realizovane klase