

Проект по предметот

Информациска безбедност



Timestamping server (апликација за архивирање документи)

Професор:

Др. Христина Михајлоска

Студенти:

Зорица Коцева, 185043 КИ

Даниел Бојчев, 181578 SEIS

Прохор Чамуровски, 181539 SEIS

Содржина

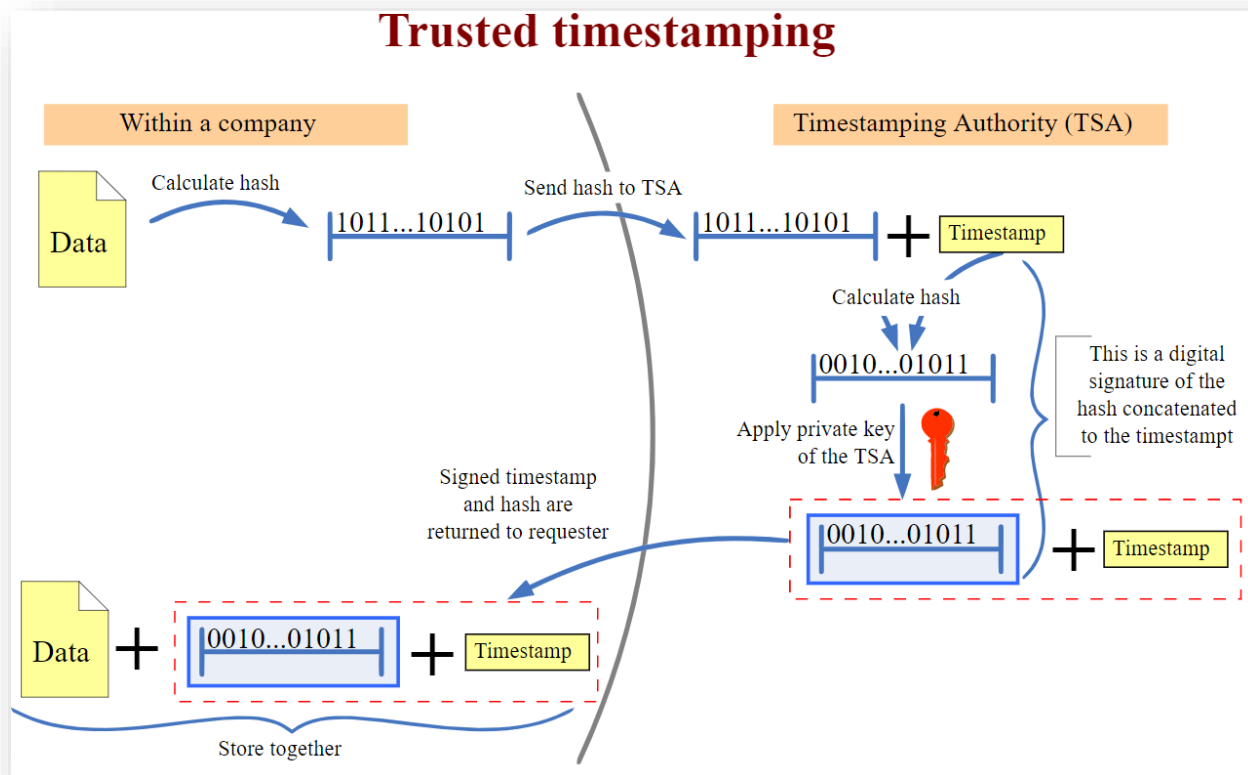
Околу нашиот проект	3
Постапно имплементирање на Timestamping server	3
Имплементиран Trusted Timestamping преку Java код.....	4
Постапно валидирање на работата на trusted Timestamping server.....	7
Имплементирана валидација на работата на Trusted Timestamping преку Java код.....	8



Околу нашиот проект

Проектот на кој избравме да работи нашиот тим е симулација за програма, односно апликација која архивира документи. За самата имплементација на апликацијата имаме направено четири класи во кои соодветно имаме јасна и прецизна разработка на кодот за сето тоа како функционира.

Постапно имплементирање на Timestamping server



Слика 1. Чекори во имплементација на процесот на Timestamping

Со тоа што го имаме текстот на почеток како влезен податок, правиме пресметка со хеширање и добиениот хеш го препраќаме до TSA (Timestamping Authority) кој овде додава Timestamp. Сега, на овој Timestamp и претходниот хеш кој беше испратен до TSA, ќе им биде пресметан друг хеш каде што на излез ќе го имаме хешот добиен од Timestamp-от и хешот добиен од текстот(data-та) испратен до TSA. Сегашниот хеш се енкриптира со клуч, и да забележиме добро дека на овој добиен хеш, се додава Timestamp и сега моментално потпишаните Timestamp и хешот заедно со почетниот текст како Data се зачувуваат заедно се пакуваат на некој начин.

Сето ова имплементирано во нашиот код е прикажано во класите кои ќе се објаснуваат подетално.

Имплементиран Trusted Timestamping преку Java код

За да ја имаме Data како податок која се предава, тоа го имаме направено во класата TSA.java, односно декларираме променлива text која ни претставува Data string од корисникот.

```

10 public class TSA {
11     //data string од корисникот
12     private String text;
13     //hash-upon string
14     private String hashedData;
15     //timestamp
16     private LocalDateTime timestamp;
17
18     public TSA(String hashedData) {
19         text = "[" + hashedData + "];";
20         this.hashedData = Hashing.getCryptoHash(hashedData, "SHA-256");
21         timestamp = generateTimestamp();
22     }
23
24     //генерирање на timestamp со помош на LocalDateTime(ја дава сегашната година, месец, час, мин...)
25     private LocalDateTime generateTimestamp() { return LocalDateTime.now(); }

```

Дополнително се декларираат и hashedData – string кој го претстацува хешираниот податок од влез во кој ќе биде зачуван текстот кога ќе ја хешираме и користејќи LocalDateTime креираме timestamp.

За да може да се постават во конструктор се предава како аргумент string-от hashedData и се предава во променливата tekst, сврзувајќи ја форматирано. За самата променлива погоре дефинирана на ниво на класата, користиме getCryptoHash метод од нашата следна Hashing класа која овозможува освен примајќи го стрингот hashedData, дефинираме дека како алгоритам користиме SHA-256.

Во конструкторот за да може да ја иницијализираме променливата timestamp го имаме методот generateTimestamp кој ги враќа моменталната година, месец, ден, минута со користејќи LocalDateTime.now().

Во наредниот чекор за да можеме да го пресметаме хешот, од методот getCryptoHash на нашата Hashing класа, да направиме подетален поглед префрлувајќи се на таа класа која ја има во прилог:

```

1 package InformationSecurity;
2
3 import java.math.BigInteger;
4 import java.security.MessageDigest;
5 import java.security.NoSuchAlgorithmException;
6
7 //класа што овозможува хеширање на даден String и соодветно тип на алгоритам
8 public class Hashing {
9
10     public static String getCryptoHash(String input, String algorithm) {
11         try {
12             MessageDigest msgDigest = MessageDigest.getInstance(algorithm);
13             byte[] inputDigest = msgDigest.digest(input.getBytes());
14             BigInteger inputDigestBigInt = new BigInteger(1, inputDigest);
15             String hashtext = inputDigestBigInt.toString(16);
16
17             while (hashtext.length() < 32) {
18                 hashtext = "0" + hashtext;
19             }
20             return hashtext;
21         } catch (NoSuchAlgorithmException e) {
22             throw new RuntimeException(e);
23         }
24     }
25 }

```

За методот во класата како два аргументи се предаваат текстот-влезот од data и алгоритмот. Оваа класа овозможува стрингот кој се предава да е хеширан со соодветно дефинираниот предаден тип на алгоритам, во овој случај ние предадовме да се користи SHA-256. Алгоритмот се предава на

getInstance методот кој се зачувува во msgDigset користејќи ја MessageDigset, а input текстот во бајти се поставува во низа од бајти како inputDigset. Декларираниот hashtext во стринг се претвара и се додека не достигнеме должината да е помала од 32, ќе се додава и поставува во hashtext кој подоцна се враќа.

Доколку се вратиме пак во TSA класата, за да можеме да покажеме имплементација на слика 1, би објасниле за методот digitalTSATimestamp() кој соодветно фрла исклучоци.

```

10
11 public class TSA {
12     //data string од корисникот
13     private String text;
14     //hash-upon string
15     private String hashedData;
16     //timestamp
17     private LocalDateTime timestamp;
18
19     public TSA(String hashedData) {
20         text = "[" + hashedData + "]";
21         this.hashedData = Hashing.getCryptoHash(hashedData, "SHA-256");
22         timestamp = generateTimestamp();
23     }
24
25     //генерирање на timestamp со помош на LocalDateTime (ја зема сегашната година, месец, час, мин..)
26     private LocalDateTime generateTimestamp() { return LocalDateTime.now(); }
27
28     //имплементација на слика1
29
30     public String digitalTSATimestamp() throws IllegalBlockSizeException, InvalidKeyException, NoSuchPaddingException, NoSuchAlgorithmException, BadPaddingException {
31         String cryptoHash = Hashing.getCryptoHash(input: hashedData + timestamp, algorithm: "SHA-256");
32         String encryptedString = Base64.getEncoder().encodeToString(RSAUtil.encrypt(cryptoHash, RSAUtil.getPublicKey()));
33         return text + " " + encryptedString + " " + timestamp;
34     }

```

Со декларирање на cryptoHash String и негово иницијализирање користејќи ја класата Hashing со што го користиме методот getCryptoHash(), како аргументи ни се потребни хешираната дата и timestamp според сликата 1, а за алгоритам повторно го предаваме и работиме со SHA-256. Откако ќе го имаме криптираниот хеш, да забележиме дека се енкриптира стринг и со ова, ја искористуваме RSAUtil класата каде со Base64 се искористува енкодер и притоа се енкриптира cryptoHash како прв предаден аргумент и јавниот клуч од RSAUtil класата во методот encrypt().

Класата RSAUtil ја гледаме во продолжение каде што накратко ќе се објасни кодот во методите што ги содржи самата класа.

```

1 package InformationSecurity;
2
3 import javax.crypto.BadPaddingException;
4 import javax.crypto.Cipher;
5 import javax.crypto.IllegalBlockSizeException;
6 import javax.crypto.NoSuchPaddingException;
7 import java.security.*;
8 import java.security.spec.InvalidKeySpecException;
9 import java.security.spec.PKCS8EncodedKeySpec;
10 import java.security.spec.X509EncodedKeySpec;
11 import java.util.Base64;
12
13 public class RSAUtil {
14
15     private static String publicKey = "MIGfMA0GCQsGSIb3DQEBAQUAA4GNADCBiQKBgQCfGVfrY4jQ8oZQW0ygzB3roKXhD4YeT2x2p41d6kP1xe73rT21h04g1agN2vgoZuHuOPqa5and6kAmk2UjncHu6D1au";
16     private static String privateKey = "MIICdQIBADANBgkqhkiG9w0BAQEFAASCA1BgggggggAgEAAoGBAKAUZV+tj1NBKh1B2bKBNzeuggdyPhh5PbHanjV0Bq+LF7vetPVhbT1cVqA3a+Chmg44+pr1qd3qQCv";
17
18     public static PublicKey getPublicKey(String base64PublicKey){
19         PublicKey publicKey = null;
20         try{
21             X509EncodedKeySpec keySpec = new X509EncodedKeySpec(Base64.getDecoder().decode(base64PublicKey.getBytes()));
22             KeyFactory keyFactory = KeyFactory.getInstance("RSA");
23             publicKey = keyFactory.generatePublic(keySpec);
24             return publicKey;
25         } catch (NoSuchAlgorithmException e) {
26             e.printStackTrace();
27         } catch (InvalidKeySpecException e) {
28             e.printStackTrace();
29         }
30         return publicKey;
31     }
32
33     public static PrivateKey getPrivateKey(String base64PrivateKey){
34         PrivateKey privateKey = null;

```

Најпрво имаме декларирано стрингови кои ни се иницијализирани и тоа јавен клуч (publicKey) и приватен, таен клуч (privateKey).

За јавниот и тајниот клуч имаме посебно методи каде се декодираат, добиваат инстанци од RSA и се генерираат самите клучеви.



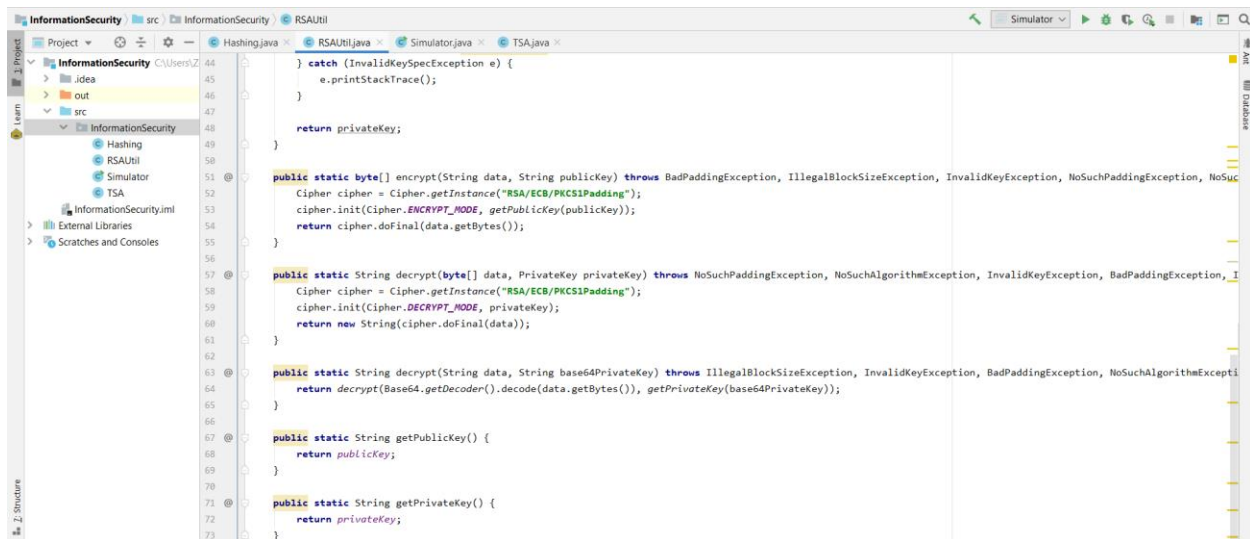
```

17
18 public static PublicKey getPublicKey(String base64PublicKey){
19     PublicKey publicKey = null;
20     try{
21         X509EncodedKeySpec keySpec = new X509EncodedKeySpec(Base64.getDecoder().decode(base64PublicKey.getBytes()));
22         KeyFactory keyFactory = KeyFactory.getInstance("RSA");
23         publicKey = keyFactory.generatePublic(keySpec);
24         return publicKey;
25     } catch (NoSuchAlgorithmException e) {
26         e.printStackTrace();
27     } catch (InvalidKeySpecException e) {
28         e.printStackTrace();
29     }
30     return publicKey;
31 }
32
33 public static PrivateKey getPrivateKey(String base64PrivateKey){
34     PrivateKey privateKey = null;
35     PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(Base64.getDecoder().decode(base64PrivateKey.getBytes()));
36     KeyFactory keyFactory = null;
37     try {
38         keyFactory = KeyFactory.getInstance("RSA");
39     } catch (NoSuchAlgorithmException e) {
40         e.printStackTrace();
41     }
42     try {
43         privateKey = keyFactory.generatePrivate(keySpec);
44     } catch (InvalidKeySpecException e) {
45         e.printStackTrace();
46     }
47     return privateKey;
48 }
49
50

```

Дополнително во самата класа ги имаме и get и set меетодите за јавниот и приватниот клуч, но и методите кои ни служат за енкрипција и декрипција на пораките, односно текстот кој го имаме. Овде искористена ни е готовата имплементација со импортирање на Cipher.

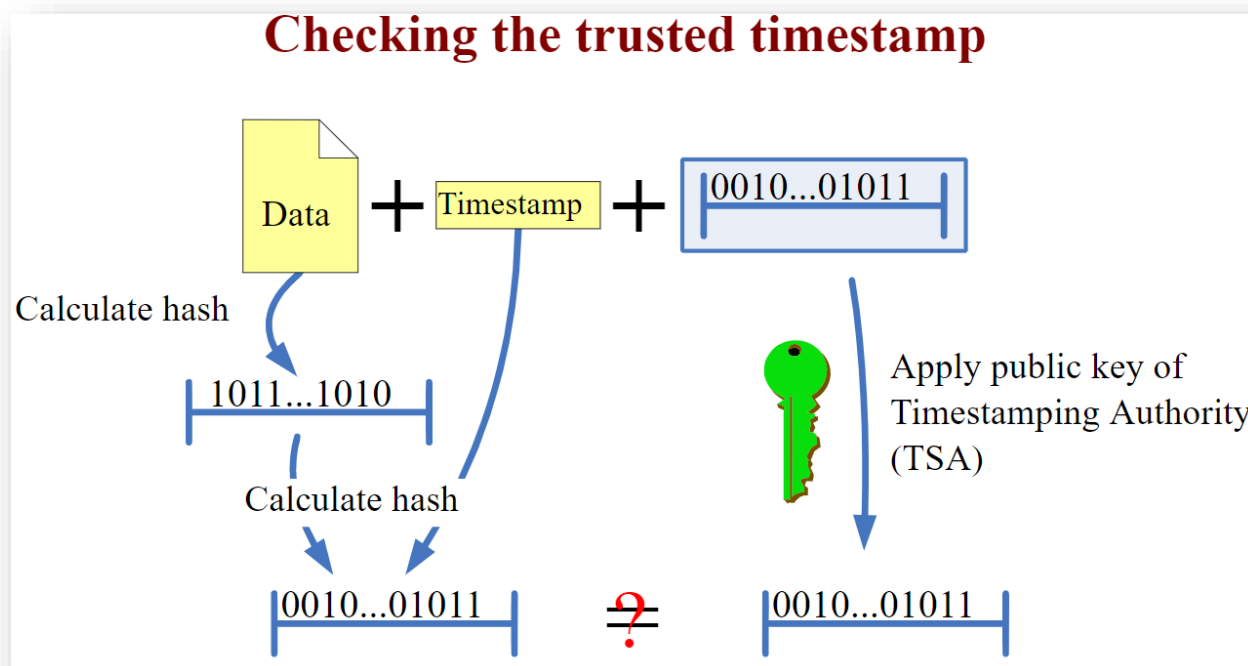
Соодветно во методот за енкриптирање се искористува со Cipher ENCRYPT_MODE, а за декрипција DECRYPT_MODE.



```

44     } catch (InvalidKeySpecException e) {
45         e.printStackTrace();
46     }
47     return privateKey;
48 }
49
50 public static byte[] encrypt(String data, String publicKey) throws BadPaddingException, IllegalBlockSizeException, InvalidKeyException, NoSuchPaddingException, NoSuchAlgorithmException {
51     Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
52     cipher.init(Cipher.ENCRYPT_MODE, getPublicKey(publicKey));
53     return cipher.doFinal(data.getBytes());
54 }
55
56 public static String decrypt(byte[] data, PrivateKey privateKey) throws NoSuchPaddingException, NoSuchAlgorithmException, InvalidKeyException, BadPaddingException, IllegalBlockSizeException {
57     Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
58     cipher.init(Cipher.DECRYPT_MODE, privateKey);
59     return new String(cipher.doFinal(data));
60 }
61
62 public static String decrypt(String data, String base64PrivateKey) throws IllegalBlockSizeException, InvalidKeyException, BadPaddingException, NoSuchAlgorithmException {
63     return decrypt(Base64.getDecoder().decode(data.getBytes()), getPrivateKey(base64PrivateKey));
64 }
65
66 public static String getPublicKey() {
67     return publicKey;
68 }
69
70 public static String getPrivateKey() {
71     return privateKey;
72 }
73

```



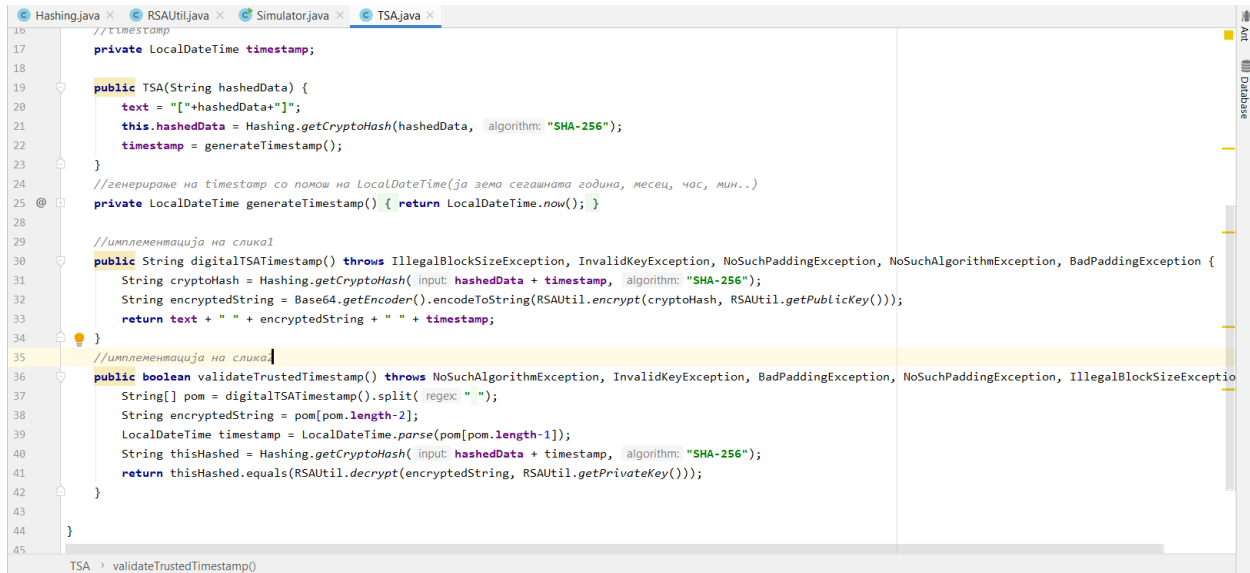
Слика 2. Чекори во имплементација на процесот на Timestamping

За да можеме да го провериме односно валидираме текстот на почеток како влезен податок, веднаш го земаме и правиме пресметка со хеширање и на добиениот хеш уште еднаш правиме пресметка со хеш само што сега вториот пат тоа е заедно со Timestamp-от. Хешот добиен на овој начин ќе се зачува за подоцна. Сега, она што следува е да провериме, односно да споредиме дали хеш кој го имаме со јавен клуч е ист со тој што го пресметувавме според Data-та и го зачувавме. Доколку се исти после споредувањето, значи дека се е во ред со валидацијата на нашиот Timestamping server, доколку се случи да бидат различни значи дека имаме некоја грешка, што не сакаме тоа да биде во нашиот случај.

Сето ова имплементирано во нашиот код е прикажано во класите кои ќе се објаснуваат подетално.

Имплементирана валидација на работата на Trusted Timestamping преку Java код

За втората слика од класата TSA за имплементација на валидацијата на нашиот Trusted Timestamping сервер, го имаме следниот код со методот `validateTrustedTimestamp()`, кој ни враќа `boolean`, односно `true` доколку се е валидно во врска со имплементацијата на нашиот код од програмата.



```

16 //timestamp
17 private LocalDateTime timestamp;
18
19 public TSA(String hashedData) {
20     text = "["+hashedData+"]";
21     this.hashedData = Hashing.getCryptoHash(hashedData, algorithm: "SHA-256");
22     timestamp = generateTimestamp();
23 }
24 //генерирање на timestamp со помош на LocalDateTime(ја зема сегашната година, месец, час, мин..)
25 private LocalDateTime generateTimestamp() { return LocalDateTime.now(); }
26
27 //имплементација на слика1
28
29 public String digitalTSATimestamp() throws IllegalBlockSizeException, InvalidKeyException, NoSuchPaddingException, NoSuchAlgorithmException, BadPaddingException {
30     String cryptoHash = Hashing.getCryptoHash(input: hashedData + timestamp, algorithm: "SHA-256");
31     String encryptedString = Base64.getEncoder().encodeToString(RSAUtil.encrypt(cryptoHash, RSAUtil.getPublicKey()));
32     return text + " " + encryptedString + " " + timestamp;
33 }
34
35 //имплементација на слика2
36 public boolean validateTrustedTimestamp() throws NoSuchAlgorithmException, InvalidKeyException, BadPaddingException, NoSuchPaddingException, IllegalBlockSizeException {
37     String[] pom = digitalTSATimestamp().split(" ");
38     String encryptedString = pom[pom.length-2];
39     LocalDateTime timestamp = LocalDateTime.parse(pom[pom.length-1]);
40     String thisHashed = Hashing.getCryptoHash(input: hashedData + timestamp, algorithm: "SHA-256");
41     return thisHashed.equals(RSAUtil.decrypt(encryptedString, RSAUtil.getPrivateKey()));
42 }
43
44 }
45
TSA > validateTrustedTimestamp()

```

За оваа имплементација во методот, имаме низа од `String` каде се зема `digitalTSATimestamp()` нашиот израз кој го имаме како текст, енкриптиран стринг и timestamp-от и се разделува.

Во енкриптираниот стринг се зема претпоследниот од крајот и се поставува во `encryptedString`. Всушност timestamp-от ни е последниот, а хешот соодветно си го имаме во `thisHashed`. На крај само ќе провериме дали `thisHashed` е еднаков со оној стринг кој ќе го добиеме искористувајќи го методот за декриптиравајќи го методот за декрипција од `RSAUtil` класата предавајќи ги `encryptedString` и од самата класа декларираниот и соодветно иницијализиран приватен клуч. Доколку се исти и се совпаѓаат тоа значи дека успешно е сработено и соодветно валидирано барањето.

Во Simulator класата, главна за извршување на нашиот Timestamping server, креираме инстанца од TSA каде предаваме хеширана Data, стринг кој ќе ни биде за тестирање.

```

14 public class Simulator {
15     public static void main(String[] args) throws IOException, InvalidKeyException, NoSuchPaddingException, BadPaddingException {
16         try {
17             Scanner sc = new Scanner(System.in);
18             String fileName = sc.next();
19             File file = new File(fileName);
20             if(file.createNewFile()){
21                 //Креираме фајл и запишуваме во него
22                 FileWriter fileWriter = new FileWriter(fileName);
23                 String str="";
24                 while((str = sc.next()).equals("")){
25                     fileWriter.append(str).append("\n");
26                 }
27                 fileWriter.close();
28             }
29             StringBuilder sb = new StringBuilder();
30             Scanner myReader = new Scanner(file);
31             while (myReader.hasNextLine()){
32                 sb.append(myReader.nextLine());
33             }
34             myReader.close();
35             //Креираме инстанца од TimestampingAuthority(TSA)
36             //TODO: Имплементација на фајл наместо стринг
37             TSA tsa = new TSA(sb.toString());
38             //Повикување на digitalTSATimestamp методот од класата TSA
39             //Како на слика 1 соодветно го враќа текстот + дигиталниот timestamp и самиот timestamp
40             System.out.println(tsa.digitalTSATimestamp());
41             //Валидација на TSA
42             System.out.println(tsa.validateTrustedTimestamp());
43         } catch (NoSuchAlgorithmException | IOException e) {
44             System.err.println(e.getMessage());
45         }
46     }
47 }

```

Користиме file за да може да се земе пораката и на два начини можеме да го направиме тоа: доколку ја стартуваме апликацијата и од конзола внесеме некој текст, потребно е за крај да го поставиме знакот „!“ за да може да се прекине вчитувањето и автоматски да се креира датотека со таа содржина и на излез да се прикаже соодветна порака како на следниот начин:

```

17 Scanner sc = new Scanner(System.in);
18 String fileName = sc.next();
19 File file = new File(fileName);
20 if(file.createNewFile()){
21     //Креираме фајл и запишуваме во него
22     FileWriter fileWriter = new FileWriter(fileName);
23     String str="";
24     while((str = sc.next()).equals("")){
25         fileWriter.append(str).append("\n");
26     }
27     fileWriter.close();
28 }
29 StringBuilder sb = new StringBuilder();
30 Scanner myReader = new Scanner(file);
31 while (myReader.hasNextLine()){
32     sb.append(myReader.nextLine());
33 }
34 myReader.close();
35 //Креираме инстанца од TimestampingAuthority(TSA)
36 //TODO: Имплементација на фајл наместо стринг
37 TSA tsa = new TSA(sb.toString());
38 //Повикување на digitalTSATimestamp методот од класата TSA
39 //Како на слика 1 соодветно го враќа текстот + дигиталниот timestamp и самиот timestamp
40 System.out.println(tsa.digitalTSATimestamp());
41 //Валидација на TSA
42 System.out.println(tsa.validateTrustedTimestamp());
43 } catch (NoSuchAlgorithmException | IOException e) {
44     System.err.println(e.getMessage());
45 }

```

```

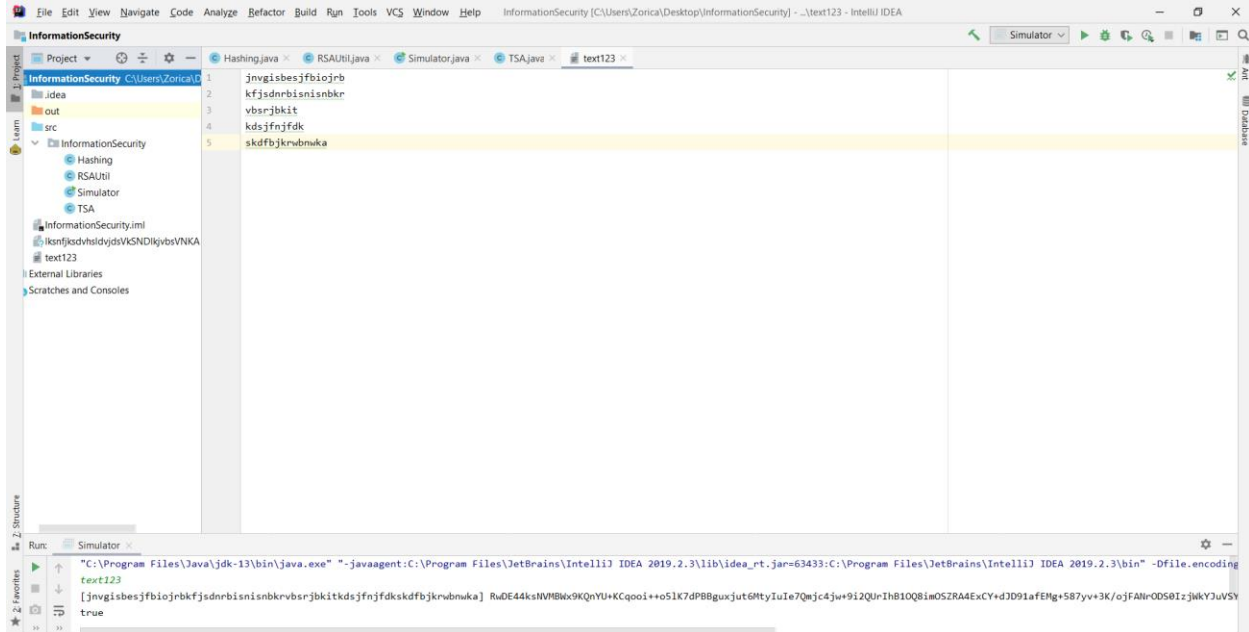
Run: Simulator
ksdjvbjkdsavknv!aklnvd
[ksdjvbjkdsavknv!aklnvd] KcQ7btXu3ux0KisYA1RUW5ux0C101BZ10a3UrFhc75vLrhis1NdZM33bVnPMZbnkAv1Y1NC3oUFN89c0DXJ06uE++ve6XtD8uu+Rao8B5IAQsF17LMygVohh4uq7deubPvE1E1TcT4R7LY27wT31WJ7Yzbc2AVG60a
true

```

За излез го принтаме со повикување на digitalTSATimestamp методот од класата TSA текстот + дигиталниот timestamp и самиот timestamp, а за проверка односно за валидација се повикува исто од tsa инстанцата validateTrustedTimestamp методот.

Забележуваме дека на излез со извршување на кодот ја добиваме пораката и се прикажува дека успешно е извршена валидација.

Вториот начин е ние да креираме датотека во самиот почеток на апликацијата за да може да го земе за читање како што е во нашиов случај `text123` и од таму содржината да ја прочита и стандардно на ист начин да го прикаже излезот.



Со внес на името на датотеката во која се наоѓа текстот врз кој треба да го извршиме кодот, ја отвора и зема содржината и соодветно ни прикажува со форматиран излез и `true` дека е валидно.