

**UNIVERSITETI I PRISHTINËS**  
**FAKULTETI I SHKENCAVE MATEMATIKO-NATYRORE**  
**Programi: Shkenca Kompjuterike**



**Punim Seminarik - Implementimi i kriptosistemit RSA në Java**  
**Lënda - Siguria e të Dhënave**

**Studentët :**

Egzonit Demhasaj

Gabriel Kolaj

**Profesor :**

Artan Berisha

Besnik Duriqi

**Maj 2023, Prishtinë**

## **Përmbajtja**

1. Një prezantim i shkurtër i projektit ;
2. Arkitektura e Programit ;
3. Dokumentimi Teknik (sqarimi i kodit burimor) ;
4. Testimi i programit ;
5. Përmbledhje .

## 1. Një prezantim i shkurtër i projektit

Në një botë gjithnjë e më të lidhur, nevoja për komunikim të sigurt dhe mbrojtje të të dhënave është bërë parësore. Kriptografia luan një rol jetik në sigurimin e konfidencialitetit dhe integritetit të informacionit të ndjeshëm. Një nga sistemet kriptografike më të përdorura dhe më të fuqishme është algoritmi **RSA** (Rivest-Shamir-Adleman). Në këtë projekt, ne do të gërmojmë në zbatimin e kriptosistemit RSA në Java, një gjuhë programimi e orientuar drejt objekteve e njohur për shkathtësinë e saj dhe bibliotekat e gjera.

Algoritmi RSA, i quajtur sipas shpikësve të tij, **Ronald Rivest**, **Adi Shamir** dhe **Leonard Adleman**, është një sistem enkriptimi me çelës publik. Ndryshe nga algoritmet me çelës simetrik, RSA përdor dy çelësa të veçantë: **një çelës publik** që përdoret për enkriptim dhe **një çelës privat** që përdoret për deshifrim. Kjo karakteristikë unike e bën RSA një zgjedhje ideale për komunikim të sigurt në skenarë ku palët e përfshira nuk kanë ndarë më parë një çelës sekret.

Objektivi kryesor i këtij projekti është të zhvillojë një program Java që mund të gjenerojë çifte çelësash RSA, të kodojë dhe deshifrojë mesazhet duke përdorur algoritmin RSA. Ne do të eksplorojmë parimet matematikore që përmbanë në vete RSA, duke përfshirë aritmetikën modulare, gjenerimin e numrave të thjeshtë dhe Teoremën Kineze mbi mbetjen. Duke kuptuar këto koncepte themelore, ne do të jemi në gjendje të zbatojmë në mënyrë efektive procedurat e enkriptimit dhe deshifrimit RSA.

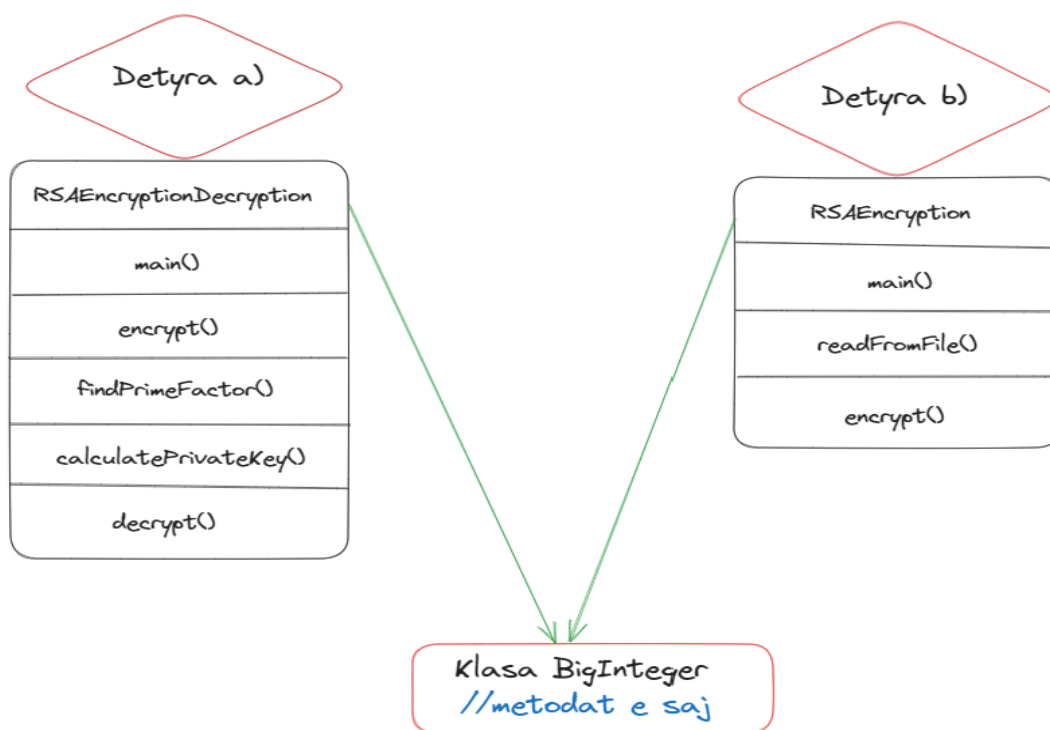
Gjatë gjithë projektit, ne do të shfrytëzojmë gjuhën e programimit Java për të krijuar një program intuitivë dhe miqësorë për përdoruesit për gjenerimin e çelësave, enkriptimin e mesazheve dhe deshifrimit. Platforma Java ofron biblioteka të fuqishme për operacione kriptografike, duke e bërë atë një zgjedhje ideale për këtë zbatim. Ne do t'i përdorim këto biblioteka për të trajtuar llogaritjet komplekse matematikore të përfshira në enkriptimin dhe deshifrimit RSA.

Për më tepër, ne do të diskutojmë gjithashtu pikat e forta dhe kufizimet e algoritmit RSA, duke përfshirë konsideratat e madhësisë kryesore dhe kompleksitetin llogaritës. Kuptimi i këtyre aspekteve do të na lejojë të marrim vendime të informuara në lidhje me përdorimin praktik të RSA në skenarë të ndryshëm.

Deri në fund të këtij projekti, ne synojmë të kemi një implementim plotësisht funksional të RSA në Java që mund të enkriptojë dhe deshifrojë në mënyrë të sigurt mesazhet, duke demonstruar fuqinë dhe efektivitetin e kriptosistemit RSA. Ky projekt jo vetëm që do të thellojë të kuptuarit tonë të kriptografisë, por gjithashtu do të përmirësojë aftësitë tona programuese, veçanërisht në fushën e komunikimit të sigurt dhe mbrojtjes së të dhënave.

## 2. Arkitektura e programit

Fillimisht duhet të cekim se në punimin e projektit tonë e kemi përdorur arkitekturën e programimit të cilën e kemi përdorur deri më tani edhe në ushtrimet tona, pra e kemi përdorur **MVC (Model View Controller)** arkitekturën. Tani projekti jonë ka qenë i ndarë në 2 pjesë, më konkretisht kemi pasur për të zgjidhur dy detyra duke përdorur kriptosistemin RSA me elementet e tij, andaj edhe kemi krijuar 2 programe duke shfrytëzuar 2 klasë të pavararura nga njëra-tjetra. Arkitektura e 2 programeve tona është dhënë në figurën e mëposhtme :



**Fig.1.** Arkitektura e të dy klasëve që përmbajnë në vete kodin për zgjidhjen e të dy detyrave, **a)** dhe **b)**

Vlenë të theksojmë se të dy këto klasa kanë përdorur metodat e klasës së gatshme të java, **BigInteger**, me qëllim të saktësisë më të lartë në llogaritje, manipulimit me numra më të mëdhenjë, si dhe llogaritjeve të tjera që kryejnë metodat e këtyre klasëve.

### 3. Dokumentimi Teknik (sqarimi i kodit burimor)

Para se të fillojmë me sqarimin e kodit burimor, do të ishte më mirë që të ndaleshim edhe pak tek algoritmi RSA, që të sqarojmë shkurtimisht hapat e realizimit të enkriptimit/dekriptimit të mesazheve me anë të këtij kriptosistemi, ashtu që ta kemi sa më të qartë më pas sqarimin e kodit burimor. Më poshtë do të japim hapat kryesorë të këtij kriptosistemi :

1. Konsideroni dy numra të thjeshtë  $p$  dhe  $q$ ;
2. Llogaritni  $n = p * q$ ;
3. Llogarit  $\phi(n) = (p - 1) * (q - 1)$ ;
4. Zgjidhni një  $\gcd(e, \phi(n)) = 1$ ;
5. Llogaritni  $d$  të tillë  $e * d \bmod \phi(n) = 1$ ;
6. Çelësi publik  $\{e, n\}$  Çelësi privat  $\{d, n\}$ ;
7. Teksti shifror  $C = P^e \bmod n$  ku  $P$  = tekst i thjeshtë;
8. Për Dëshifrimin  $D = C^d \bmod n$  ku  $D$  do të rimbursojë tekstin e thjeshtë.

Tani pasi i kemi pak a shumë të qartë hapat se si funksionon kriptosistemi RSA, mund të vazhdojmë me sqarimin e kodit burimor. Ashtu edhe siç e cekëm edhe pak më herët tek arkitektura e programit, projekti jonë ka përmbajtur në vete zgjidhjen e dy detyrave duke përdorur kriptosistemin RSA. Më saktësisht, dy detyrat tona kanë qenë këto:

**a)** Le të jenë  $n=253$  dhe  $e=31$  çelës publik. Enkripto  $m=10$ , dhe gjejë  $p$ ,  $q$  dhe  $d$ , pastaj dekripto  $c=35$  ;

**b)** Le të jenë  $p=67, q=73$  dhe  $e=89$ . Enkripto tekstin “**SIGURI**” me gjatësi të bllokut 2.

Për zgjidhjen e këtyre dy detyrave, ne kemi krijuar dy klasë të veçanta në Java, ashtu siç edhe shihet nga arkitektura e programit, ku të dy klasat janë të pavarura nga njëra tjetra dhe secila i ka metodat e veta, rolin e të cilave do ta tregojmë në vazhdim.

Detyra **a)** :

Për zgjidhjen e problemit që përmbanë në vete detyra a), ne kemi krijuar një klasë në Java, të cilën e kemi emëruar **RSAEncryptionDecryption**. Kjo klasë përmbanë në vete edhe 5 metoda të tjera të cilat mundësojnë zgjidhjen e problemit që paraqet detyra a), si enkriptimi i mesazhit  $m=10$ , gjetja e faktorëve të thjeshtë  $p$  dhe  $q$ , pastaj gjenerimi i çelësit privat  $d$  dhe dekriptimi i mesazhit të shifruar  $c=35$ . Ndër këto metoda, gjendet edhe metoda **main** për testimin e programit. Në vijim do të japim një përshkrimi më të detajizuar të këtyre metodave :

- **encrypt(BigInteger m, BigInteger n, BigInteger e)** : Kjo metodë përdoret për të kryer enkriptimin duke përdorur algoritmin RSA. Duhet tre parametra, **m** që përfaqëson mesazhin ose tekstin e thjeshtë që duhet të kodohet, **n** që përfaqëson modulën e çiftit të çelësve RSA e që është pjesë e çelësit publik, si dhe parametri **e** që përfaqëson eksponentin publik të çiftit të çelësve RSA. Është gjithashtu një pjesë e çelësit publik.
- **findPrimeFactor(BigInteger n)** : Kjo metodë përdoret për të gjetur një nga faktorët e thjeshtë të një numri të caktuar **n**. Në kontekstin e algoritmit RSA, kjo metodë mund të përdoret për të gjetur një nga faktorët e thjeshtë (qoftë **p** ose **q**) të modulit **n**, i cili është një hap vendimtar në procesin e gjenerimit të çelësve. Metoda merr një parametër **n** të tipit **BigInteger**. Ky përfaqëson numrin për të cilin duam të gjejmë një nga faktorët e thjeshtë të tij. Metoda fillon duke inicializuar një variabël **BigInteger i** me vlerën 2. Më pas hyn në një unazë **while** që vazhdon derisa moduli **n** me **i** të jetë i barabartë me zero, d.m.th., derisa **n** të pjesëtohet me **i** pa mbetje. Brenda ciklit **while**, vlera e **i** rritet me një duke përdorur metodën **add** të klasës **BigInteger**. Kjo lejon që metoda të përsëritet përmes numrave të njëpasnjëshëm dhe të kontrollojë nëse janë faktorë të **n**. Pasi cikli përfundon, vlera e **i** do të jetë një faktor i thjeshtë prej **n**. Ky faktor i thjeshtë kthehet si rezultat nga metoda. Është e rëndësishme të theksohet se kjo metodë supozon se **n** ka të paktën një faktor të thjeshtë. Në kontekstin e

RSA,  $n$  është zakonisht produkt i dy numrave të mëdhenjë të thjeshtë, duke e bërë atë me shumë gjasa që të ketë faktorë të thjeshtë.

➤ **calculatePrivateKey(BigInteger e, BigInteger p, BigInteger q)** : Kjo metodë përdoret për të llogaritur çelësin privat  $d$  në algoritmin RSA, i cili është i nevojshëm për dekriptim të mesazhit. Metoda merr tre parametra,  $e$  të tipit BigInteger, i cili përfaqëson eksponentin publik të çiftit të çelësive RSA,  $p$  të tipit BigInteger, i cili përfaqëson një nga faktorët e thjeshtë të modullit  $n$ , si dhe  $q$  të tipit BigInteger, i cili përfaqëson faktorin tjetër të thjeshtë të modullit  $n$ . Metoda fillon duke llogaritur vlerën e  $fiN$ , i cili reprezenton **funksionin e Eulerit** për numrin  $n$ . Në RSA,  $fiN$  llogaritet si  $(p - 1) * (q - 1)$ , ku  $p$  dhe  $q$  janë faktorët e thjeshtë të  $n$ . Funksioni i Eulerit llogarit numrin e numrave më të vegjël se  $n$  e që nuk kanë faktorë të përbashkët me  $n$  përveç 1. Më pas, metoda përdor funksionin **modInverse** të klasës **BigInteger** për të llogaritur inversin modular të  $e$  në lidhje me  $fiN$ . Inversi modular i  $e$  është një vlerë  $d$  e tillë që  $(e * d) \bmod fiN = 1$ . Me fjalë të tjera,  $d$  është inversi shumëzues i  $e$  **modulo**  $fiN$ . Vlera e llogaritur e  $d$  është çelësi privat që përdoret për deshifrim në algoritmin RSA. Ai i lejon marrësit të deshifrojë tekstet e shifruara duke përdorur operacionin e fuqizimit modular, duke aplikuar  $(c^d) \bmod n$ . Kjo metodë është një hap thelbësor në gjenerimin e çelësit RSA, pasi përcakton çelësin privat që korrespondon me eksponentin e dhënë të çelësit publik  $e$  dhe faktorët e thjeshtë  $p$  dhe  $q$ . Ai luan një rol vendimtar në sigurimin e komunikimit të sigurt duke lejuar vetëm marrësit e autorizuar të deshifrojnë mesazhet e koduara.

➤ **decrypt(BigInteger c, BigInteger n, BigInteger d)** : Kjo metodë përdoret për të kryer deshifrimin duke përdorur algoritmin RSA. Duhet tre parametra,  $c$  i tipit BigInteger, i cili përfaqëson tekstin e shifruar që duhet të deshifrohet,  $n$  i tipit BigInteger, i cili përfaqëson modulin e çiftit të çelësive RSA, si dhe  $d$  po ashtu i tipit BigInteger, i cili përfaqëson eksponentin privat të çiftit të çelësive RSA. Metoda përdor funksionin **modPow**, i cili është një metodë e integruar në klasën **BigInteger** në Java. Funksioni modPow njehson fuqinë modulare të  $c$  të ngritur në fuqinë e  $d$  **modulo**  $n$ . Me fjalë të tjera, ai llogarit  $(c^d) \% n$ . Rezultati i kësaj llogaritjeje është teksti i thjeshtë, i cili përfaqëson versionin e deshifruar të tekstit të koduar  $c$ . Procesi i deshifrimit mbështetet



në marrëdhënien matematikore ndërmjet eksponentit privat **d** dhe eksponentit publik **e**. Ndërsa procesi i enkriptimit përdor çelësin publik (**e**, **n**), procesi i deshifrimit përdor çelësin privat (**d**, **n**). Duke aplikuar operacionin e fuqizimit modular, metoda e dekriptimit siguron që procesi i deshifrimit të mbetet efikas dhe i realizueshëm nga ana llogaritëse, edhe për vlera të mëdha të **c**, **d** dhe **n**. Është e rëndësishme të theksohet se metoda e dekriptimit supozon se çelësi privat **d** i ofruar është çelësi privat përkatës i saktë me çelësin publik të përdorur për enkriptim. Nëse përdoret një çelës privat i pasaktë, rezultati i deshifruar mund të jetë i pasaktë ose i pakuptimtë. Metoda e dekriptimit luan një rol jetik në algoritmin RSA, duke lejuar marrësit e autorizuar me çelësin e duhur privat të marrin tekstin origjinal origjinal nga teksti i koduar i koduar, duke siguruar komunikim të sigurt dhe konfidencialitet të të dhënave.

Detyra **b)** :

Për zgjidhjen e problemit që mbanë në vete Detyra b) ne kemi krijuar një klasë në Java me emrin **RSAEncryption**, e cila në vete përmbanë 3 metoda, ndër të cilat edhe metoda **main** për testimin e programit. Dy metodat e tjera të kësaj klase janë edhe metoda **readFromFile** dhe metoda **encrypt**, të cilat përdoren për leximin e përmbajtjes së file-it dhe enkriptimin e asaj përmbajtje duke përdorur algoritmin RSA konkretisht. Siç shihet më lartë nga definimi i problemit të kësaj detyre, ne thjeshtë kemi pasur për detyrë ta enkriptojmë mesazhin “**SIGURI**” duke marrë të gatshme elementet e nevojshme për enkriptim me anë të algortimit RSA, e gjithashtu duke e marrë edhe gjatësinë e bllokut 2. Megjithatë, me qëllim që ta modifikojmë më shumë programin dhe ta bëjmë më konsistent ndaj shfrytëzuesit e më gjeneralizues, ne e kemi shkruar klasën **RSAEncryption** ashtu që mesazhin që ka për ta enkriptuar e merr dhe e lexon nga një text file, të cilin e kemi emëruar si **FileMessage.txt**, e na shfaq më pas mesazhin e shifruar. Për më shumë detaje rreth funksionimit të programit do të keni mundësinë të dini më vonë, gjatë **Testimit të Programit**, ndërsa për të kuptuar më shumë rreth metodave të kësaj klase keni mundësinë në vijim :

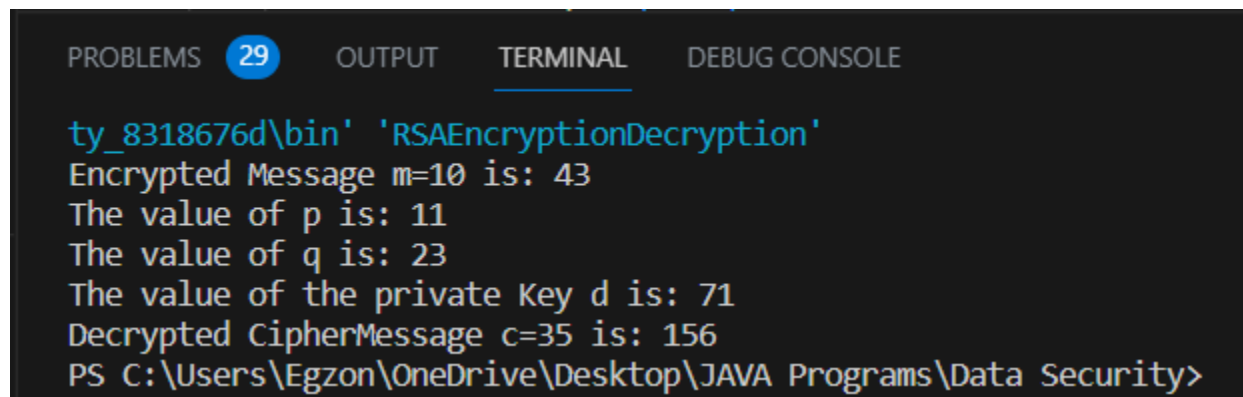
- **encrypt(String message, BigInteger n, BigInteger e, int blockSize):** Kjo metodë përdoret për të enkriptuar një mesazh të caktuar duke përdorur algoritmin e enkriptimit RSA. Metoda merr katër parametra, **mesazhin që do të enkriptohet (message)**, **moduli (n)**, **eksponenti publik (e)** dhe **madhësia e bllokut (blockSize)** që llogarit numrin e blloqeve të nevojshme për të përfaqësuar mesazhin bazuar në madhësinë e bllokut të dhënë. Ky parametrë inicializon një varg (**encryptedBlocks**) për të ruajtur blloqet e koduara të mesazhit. Metoda përsëritet mbi çdo bllok. Për çdo bllok, ai krijon një StringBuilder (**blockString**) për të ndërtuar përfaqësimin e vargut të bllokut. Brenda ciklit të brendshëm, ai shton vlerën **ASCII** të çdo karakteri në mesazh në **blockString**. Nëse gjatësia e mesazhit shterohet përpara se të arrijë madhësinë e bllokut, ajo shton hapësira për të mbushur bllokun. **BlockString** më pas shkurtohet për të hequr çdo hapësirë pasuese dhe konvertohet në një **BigInteger** (bllok). Së fundi, metoda aplikon algoritmin e enkriptimit RSA duke llogaritur fuqinë modulare të bllokut (**block.modPow(e, n)**) dhe ruan rezultatin në vargun e **encryptedBlocks**. Pasi të gjitha blloqet të jenë të koduara, metoda kthen si rezultat vargun e **encryptedBlocks** që përmban mesazhin e koduar. Është e rëndësishme të theksohet se zbatimi i ofruar supozon se mesazhi përbëhet nga karaktere të përfaqësuar nga vlerat e tyre ASCII dhe klasa **BigInteger** përdoret për të trajtuar numra të mëdhenj në procesin e enkriptimit RSA.
- **readFromFile(String filePath):** Kjo metodë përdoret për të lexuar përmbajtjen e një skedari të specifikuar nga parametri **filePath** dhe për të kthyer rezultatin si një varg. Metoda merr një parametër të vetëm, **filePath**, i cili përfaqëson rrugën drejt file-it që duhet lexuar. Ai inicializon një **StringBuilder** (**messageBuilder**) për të ruajtur përmbajtjen e file-it. Brenda një blloku prove, ai krijon një objekt **BufferedReader**, duke i kaluar atij një **FileReader** që përfaqëson file-in e specifikuar nga **filePath**. Kjo ju lejon të lexoni përmbajtjen e file-it. Metoda më pas hyn në një unazë ku lexon çdo rresht të file-it duke përdorur metodën **readLine()** të **BufferedReader**. Për sa kohë që ka më shumë rreshta për të lexuar (**line != null**), cikli vazhdon. Për çdo rresht të lexuar, metoda e shton atë te **messageBuilder** duke përdorur

metodën **append()**, që gjithashtu shton **System.lineSeparator()** për të ruajtur ndërprerjet origjinale të linjës në file. Pas leximit të të gjitha rreshtave, `BufferedReader` mbyllet automatikisht për shkak të përdorimit të deklaratës apo bllokut **try**. Kjo siguron trajtimin e duhur të burimeve dhe shmang rrjedhjet e burimeve. Nëse ndodh një përjashtim gjatë procesit të leximit të file-it, si p.sh. një **IOException**, përjashtimi kapet nga blloku **catch** dhe gjurmët e tij printohen. Së fundi, metoda e kthen përmbajtjen e file-it si një varg duke thirrur metodën **toString()** në `messageBuilder`. Në përmbledhje, kjo metodë ofron një mënyrë të përshtatshme për të lexuar përmbajtjen e një file-i dhe për t'i rikthyer ato si një varg.

Kjo ishte e tëra sa i përket sqarimit të kodit burimor, kurse për të kuptuar më shumë rreth funksionimit të programit, do të keni mundësinë në vijim gjatë **Testimit të programit**.

## 4. Testimi i programit

Ashtu siç treguam më herët, tani do të shfaqim disa figura që na tregojnë rezultatet e programit pasi ekzekutimit të tij dhe do të konsatojmë se a janë rezultatet e njëjta me ato që pretenduam më herët. Fillimisht do të testojmë klasën **RSAEncryptionDecryption**, e cila përmbanë zgjidhjen e Detyrës a), e më pas do të testojmë klasën **RSAEncryption**, e cila përmbanë zgjidhjen e Detyrës b). Kalojmë tek klasa **RSAEncryptionDecryption** dhe me ekzekutimin e kodit, na shfaqet rezultati i kërkuar nga kjo detyrë :

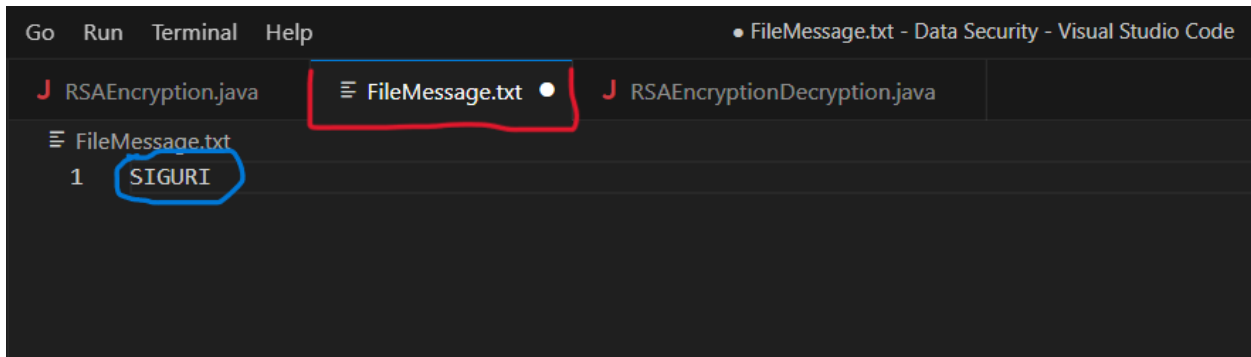


```
PROBLEMS 29 OUTPUT TERMINAL DEBUG CONSOLE
ty_8318676d\bin' 'RSAEncryptionDecryption'
Encrypted Message m=10 is: 43
The value of p is: 11
The value of q is: 23
The value of the private Key d is: 71
Decrypted CipherMessage c=35 is: 156
PS C:\Users\Egzon\OneDrive\Desktop\JAVA Programs\Data Security>
```

**Fig.1.** Rezultati i klasës **RSAEncryptionDecryption** që zgjidh Detyrën a)

Shohim se pas ekzekutimit të kësaj klase, na janë shfaqur rezultatet e duhura, pra fillimisht na është shfaqur mesazhi i enkriptuar **m=10**, ku rezultati i enkriptimit ka dalur **43**, pastaj programi ka llogaritur vlerën e **p** dhe **q**, ku fillimisht e kemi ditur vlerën e **n=253**, pastaj kemi llogaritur çelësin privat **d=71**, dhe në fund duke shfrytëzuar çelësin privat të llogaritur e kemi dekriptuar mesazhin e shifruar **c=35**, me ç'rast kemi fituar rezultatin **156**.

Tani kalojmë tek testimi i klasës **RSAEncryption**. Para se ta ekzekutojmë kodin dhe ta paraqesim rezultatin e fituar, fillimisht le të shohim përmbajtjen e file-it **FileMessage.txt**, përmbajtjen e të cilit do ta enkriptojmë me anë të kriptosistemit **RSA** të cilin e kemi implementuar.



**Fig.2.** Përmbajtja e file-it FileMessage.txt, të cilin kemi për ta enkriptuar

Shohim se file-i përmbanë pikërisht mesazhin që ka kërkuar Detyra b), pra mesazhin **“SIGURI”**, të cilin klasa jonë do ta lexojë dhe do ta enkriptoje me anë të metodave të cilat i kemi përmendur më herët.

Tani le të shohim rezultatin pas ekzekutimit të klasës RSAEncryption :

```

13
14 String filePath = "FileMessage.txt";
15
16 String message = readFromFile(filePath);
17
18 BigInteger n = p.multiply(q);
19 BigInteger[] encryptedMessage = encrypt(message, n, e, blockSize);
20
21 System.out.println("Encrypted Message:");
22 for (BigInteger block : encryptedMessage) {
23     System.out.println(block);
24 }
25
26
27 public static String readFromFile(String filePath) {
28     /*This method is used to read the contents of the file specified by the filePath

```

```

ty 8318676d\bin\ 'RSAEncryption'
Encrypted Message:
2606
1664
660
967
PS C:\Users\Egzon\OneDrive\Desktop\JAVA Programs\Data Security>

```

**Fig.3.** Rezultati i klasës RSAEncryption që zgjidh Detyrën b)

Nga figura shohim (aty ku kemi nënvizuar me të kaltërt) se kemi lexuar përmbajtjen nga file-i FileMessage.txt me anë të metodës përkatëse, e më pas e kemi enkriptuar përmbajtjen e atij file-i, po ashtu me anë të metodës përkatëse, ku rezultati i enkriptimit është shfaqur është shfaqur në rreshtin e konsolës (aty ku kemi nënvizuar me të kuqe).

## 5. Përmbledhje

Dhe kështu erdhëm në përfundim të dokumentimit të projektit tonë. Projektin tonë jemi munduar që ta marrim me seriozitet të plotë, ashtu siç edhe e kërkon fakulteti, jemi munduar që ta punojmë jashtëzakonisht shumë mirë, pa gabime, me shumë modifikime që e bëjnë programin më efektivë ndaj shfrytëzuesit, dhe jemi munduar që të përdorim gjëra që kemi mësuar nga të dy profesorët e nderuar gjatë këtij semestri në lëndën **Siguria e të Dhënave**.

Edhe Dokumentimin e Projektit jemi munduar ta bëjmë sa më mirë, pasi që një projekt sado i mirë që të jetë, nëse nuk ka dokumentim, atëherë ai është një “hiç”. Jemi munduar që ta spjegojmë shumë mirë projektin tonë, duke përfshirë edhe figura ashtu që të jemi sa më të qartë me ju profesorë, por edhe me ndonjë shfrytëzues tjetër. Të gjitha veçoritë e projektit tonë, të cilat i kemi treguar tek **Dokumentimi Teknik**, jemi munduar t’i vërtetojmë përmes **Testimit të Aplikacionit** dhe të jemi sa më të sigurtë.

Ne e kemi dhënë më të mirën tonë, andaj shpresojmë që të ju pëlqejë projekti jonë. Natyrisht se ka vend për shumë përmirësime, fundja ky është qëllimi ynë që të marrim kritika nga profesorët tonë dhe të punojmë në përmirësimin e tyre. Ne ju falënderojmë shumë për përkrahjen dhe shpjegimin e qartë gjatë këtij semestri, ishte shumë kënaqësi që të ishim student i juaji.

**Me shumë respekt, Egzonit Demhasaj & Gabriel Kolaj**  
**Maj 2023, Prishtinë**