**Ministerul Educației, Culturii și Cercetării**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**Departamentul Ingineria Software și Automatică**

**Raport**

Lucrarea de laborator nr.2

Disciplina: Securitatea Informațională

Tema: Algoritmul RSA

**Efectuat**: st.gr. TI-202 Bunescu Gabriel

**Verificat**: asist. univ. Todos Alexandru

Chișinău 2023

**Scopul lucrării:**

Elaborarea unui algorit de criptare și decriptare.

**Mersul lucrării:**

1. Studierea Algoritmului RSA
2. Calculul cheilor
3. Diferenţele între algoritmi simetrici şi asimetrici
4. Realizarea unei aplicaţi ce utilizează RSA ca metodă de criptare
5. **Studierea Algoritmului RSA**

Algoritmul RSA (Rivest-Shamir-Adleman) este un algoritm de criptare cu cheie publică, folosit pentru a securiza comunicarea și pentru a proteja datele. A fost dezvoltat de către Ron Rivest, Adi Shamir și Leonard Adleman în 1977 și rămâne unul dintre cele mai utilizate algoritme criptografice.

Iată cum funcționează algoritmul RSA într-o descriere generală:

1. Generarea Cheilor:

* Un utilizator generează o pereche de chei RSA, constând dintr-o cheie publică și o cheie privată.
* Cheia publică poate fi distribuită oricui, în timp ce cheia privată trebuie păstrată secretă.

1. Criptarea:

* persoană dorește să trimită un mesaj securizat unei alte persoane.
* Ea folosește cheia publică a destinatarului pentru a cripta mesajul.
* Mesajul criptat este trimis destinatarului.

1. Decriptarea:

* Destinatarul primește mesajul criptat și folosește cheia sa privată pentru a-l decripta.
* Numai destinatarul, care deține cheia privată corespunzătoare, poate decripta mesajul.

1. Securitate:

* Securitatea algoritmului RSA se bazează pe dificultatea factorizării unui număr mare în factori primi.
* Cu cât cheia este mai lungă (mai mulți biți), cu atât este mai dificil să se efectueze factorizarea și, prin urmare, algoritmul este mai sigur.

Principalele componente ale algoritmului RSA sunt următoarele:

* Generarea Cheilor: Aceasta implică alegerea a două numere prime mari (p și q) și calcularea altor valori, cum ar fi n, e (cheia publică) și d (cheia privată).
* Criptarea: Mesajul este convertit într-un număr întreg și este ridicat la puterea e modulo n pentru a obține mesajul criptat.
* Decriptarea: Mesajul criptat este ridicat la puterea d modulo n pentru a obține mesajul original.
* Securitate: Securitatea algoritmului RSA se bazează pe dificultatea găsirii factorilor primi p și q ai produsului n. Dacă cineva ar reuși să factorizeze n în p și q, atunci ar putea calcula cheia privată și ar putea decripta mesajele.

Pentru a utiliza RSA în practică, trebuie să utilizați biblioteci de criptografie sau software specializat care să implementeze algoritmul, deoarece implementarea corectă a acestuia poate fi complexă din punct de vedere matematic.

Este important să menționăm că, în timp ce RSA este un algoritm robust, trebuie să fie utilizat în mod corespunzător și să fie gestionate cheile cu grijă pentru a asigura securitatea comunicațiilor și a datelor.

1. **Pentru a calcula cheile RSA, trebuie să urmați următorii pași**
2. Generarea numerelor prime p și q:

* Alegeți două numere prime mari, p și q. Acestea trebuie să fie numere prime distincte și trebuie să fie știute doar de către cel care generează cheile.

1. Calculul produsului n:

* Calculați produsul n = p \* q. Acesta este un număr întreg mare și este parte integrantă a cheilor RSA.

1. Calculul funcției phi (φ) a lui Euler pentru n:

* Calculează φ(n) = (p-1) \* (q-1), unde φ este funcția lui Euler (numărul de numere întregi mai mici sau egale cu n care sunt prime între ele cu n).

1. Alegerea unui exponent public e:

* Alegeți un exponent public e. În mod obișnuit, valoarea comună pentru e este 65537 (2^16+1), deoarece are proprietăți matematice favorabile și asigură o criptare eficientă.

1. Calculul exponentului privat d:

* Calculați exponentul privat d astfel încât să satisfacă următoarea relație: d \* e ≡ 1 (mod φ(n))
* Acest lucru poate fi calculat folosind un algoritm numit "algoritmul extins Euclidian."

1. Cheia publică și cheia privată:

* Cheia publică este formată din p și e. Aceasta este distribuită celor care vor să vă trimită mesaje criptate.
* Cheia privată este formată din p și d. Aceasta trebuie păstrată în siguranță și nu trebuie să fie divulgată.

După ce ați calculat aceste valori, aveți o pereche de chei RSA:

* Cheia Publică: (n, e)
* Cheia Privată: (n, d)

Aceste chei sunt esențiale pentru criptarea și decriptarea mesajelor utilizând algoritmul RSA. Este important să protejați cheia privată și să o păstrați în siguranță, deoarece aceasta este cea care permite decriptarea mesajelor criptate cu cheia publică corespunzătoare.

1. **Diferenţele între algoritmi simetrici şi asimetrici**

Algoritmii criptografici pot fi împărțiți în două categorii principale: algoritmi simetrici și algoritmi asimetrici. Aceste două categorii diferă semnificativ în modul în care gestionează cheile și în aplicarea lor practică. Iată câteva diferențe cheie între acești doi tipi de algoritmi:

1. Chei:

* Algoritmi Simetrici: Folosesc aceeași cheie pentru atât criptare, cât și decriptare. Această cheie este denumită "cheie de criptare" sau "cheie secretă". Deoarece aceeași cheie este utilizată pentru ambele operațiuni, trebuie păstrată în siguranță și distribuită cu grijă între părțile care doresc să comunice în mod securizat.
* Algoritmi Asimetrici: Folosesc două chei distincte, una pentru criptare și alta pentru decriptare. Cheia de criptare este publică și este cunoscută de toată lumea, în timp ce cheia de decriptare este păstrată secretă și este cunoscută numai de către destinatarul mesajelor criptate.

1. Eficiență:

* Algoritmi Simetrici: Sunt în general mult mai eficienți din punct de vedere al performanței decât algoritmii asimetrici. Criptarea și decriptarea cu chei simetrice sunt rapide și pot procesa cantități mari de date cu ușurință.
* Algoritmi Asimetrici: Sunt mai lenti decât algoritmii simetrici, deoarece operațiile matematice implicate sunt mai complexe. Cu toate acestea, algoritmii asimetrici sunt utilizați în mod obișnuit pentru a schimba chei simetrice (sesiune) în comunicațiile securizate, în loc să cripteze direct datele.

1. Securitate:

* Algoritmi Simetrici: Sunt bine potriviți pentru criptarea datelor, dar pot avea provocări în distribuirea cheilor secrete în mod securizat. Un atacator care obține cheia secretă poate decripta toate mesajele criptate cu acea cheie.
* Algoritmi Asimetrici: Sunt utilizați în primul rând pentru a stabili comunicări sigure și pentru schimbul securizat de chei simetrice. Datorită utilizării a două chei diferite și a matematicii complexe implicate, sunt mai robuste în ceea ce privește securitatea în comparație cu algoritmii simetrici.

1. Utilizări Tipice:

* Algoritmi Simetrici: Sunt utilizați pentru criptarea datelor în tranzit sau în stocare, precum și pentru a proteja confidențialitatea informațiilor. Exemple includ AES, DES, și Triple DES.
* Algoritmi Asimetrici: Sunt utilizați pentru autentificare, semnături digitale, schimbul de chei și pentru a stabili canale de comunicare sigure. Exemple includ RSA, ECC, și algoritmi de schimb de chei, cum ar fi Diffie-Hellman.

1. **Realizarea unei aplicaţi ce utilizează RSA ca metodă de criptare**

Codul dat generează o pereche de chei RSA (o cheie privată și o cheie publică), criptează un mesaj cu cheia publică, apoi decriptează mesajul criptat cu cheia privată. Iată o comentare a fiecărei părți a codului:

**Importarea bibliotecilor necesare:**

**from** cryptography.hazmat.backends **import** default\_backend

**from** cryptography.hazmat.primitives.asymmetric **import** rsa

**from** cryptography.hazmat.primitives **import** serialization

**from** cryptography.hazmat.primitives **import** hashes

**from** cryptography.hazmat.primitives.asymmetric **import** padding

**import** base64

Aceasta este secțiunea de importare, în care sunt importate bibliotecile și modulele necesare pentru a lucra cu criptografia RSA și operații de bază, cum ar fi manipularea șirurilor Base64.

**Generarea unei perechi de chei RSA:**

private\_key = rsa.generate\_private\_key(

public\_exponent=65537,

key\_size=2048,

backend=default\_backend()

)

Aici se generează o pereche de chei RSA, constând dintr-o cheie privată și o cheie publică. Cheia privată este generată utilizând algoritmul RSA, iar parametrii precum dimensiunea cheii și exponentul public sunt specificați.

**Serializarea cheii private și a cheii publice:**

private\_pem = private\_key.private\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PrivateFormat.PKCS8,

encryption\_algorithm=serialization.NoEncryption()

)

public\_key = private\_key.public\_key()

public\_pem = public\_key.public\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PublicFormat.SubjectPublicKeyInfo

)

Aici, cheia privată și cheia publică sunt serializate sub formă de șiruri PEM. Cheia privată este salvată fără nici o encriptare suplimentară (PKCS#8), iar cheia publică este serializată ca informații publice despre cheie.

**Afișarea cheilor:**

**print**("Cheia privată:")

**print**(private\_pem.decode())

**print**("**\n**Cheia publică:")

**print**(public\_pem.decode())

Cheile private și publice sunt afișate în format text.

**Citirea textului pentru criptare:**

plaintext\_message = input("Introduceți textul pe care doriți să-l criptați: ").encode()

Acesta este un mesaj de la utilizator, pe care acesta îl introduce pentru a fi criptat. Mesajul este citit de la tastatură și transformat într-un șir de octeți (bytes) pentru a fi criptat.

**Criptarea mesajului cu cheia publică:**

ciphertext = public\_key.encrypt(

plaintext\_message,

padding.OAEP(

mgf=padding.MGF1(algorithm=hashes.SHA1()), *# Folosim SHA-1 aici pentru MGF1*

algorithm=hashes.SHA1(), *# Folosim SHA-1 pentru padding*

label=None

)

)

Mesajul este criptat folosind cheia publică cu ajutorul padding-ului OAEP (Optimal Asymmetric Encryption Padding), care adaugă securitate suplimentară la criptare.

**Convertirea mesajului criptat în șir Base64:**

ciphertext\_base64 = base64.b64encode(ciphertext).decode()

Mesajul criptat este codificat în format Base64 pentru a putea fi ușor afișat și partajat.

**Afișarea mesajului criptat:**

**print**("**\n**Mesaj criptat (Base64):")

**print**(ciphertext\_base64)

Mesajul criptat în format Base64 este afișat la consolă.

**Decriptarea mesajului cu cheia privată:**

decoded\_ciphertext = base64.b64decode(ciphertext\_base64)

decrypted\_message = private\_key.decrypt(

decoded\_ciphertext,

padding.OAEP(

mgf=padding.MGF1(algorithm=hashes.SHA1()), *# Folosim SHA-1 aici pentru MGF1*

algorithm=hashes.SHA1(), *# Folosim SHA-1 pentru padding*

label=None

)

)

Mesajul criptat este mai întâi decodificat din Base64, apoi este decriptat folosind cheia privată și aceeași configurație de padding OAEP ca la criptare.

**Afișarea mesajului decriptat:**

**print**("**\n**Mesaj decriptat:")

**print**(decrypted\_message.decode())

Mesajul decriptat este afișat la consolă.

Acest este o implementare simplu de utilizare a criptografiei RSA pentru a cripta și decripta mesaje utilizând cheile publice și private corespunzătoare.



Figura 1 – Afișarea cheii private.

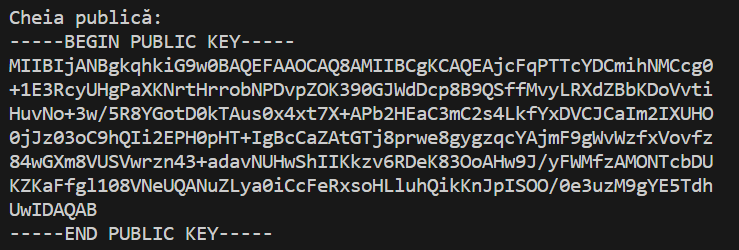


Figura 2 – Afișarea cheii publice.

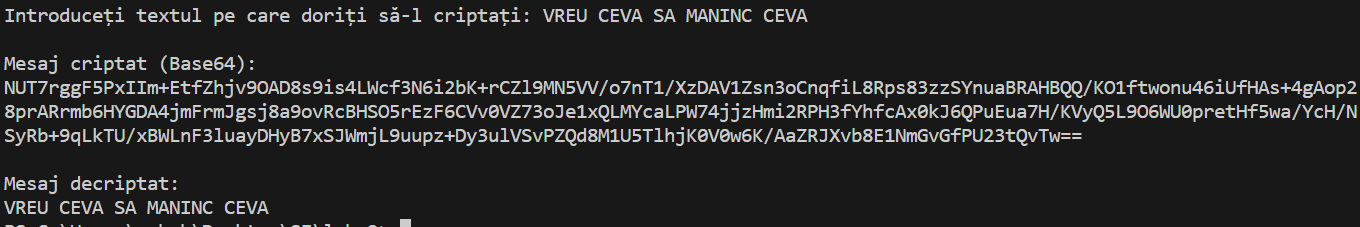


Figura 3 – Afișarea mesajului criptat și mesajul decriptat.

**Concluzie:**

În cadrul acestui proiect, am elaborat un algoritm simplu de criptare și decriptare folosind algoritmul RSA (Rivest-Shamir-Adleman). RSA este un algoritm de criptare cu cheie publică și este utilizat pentru securizarea comunicării și protejarea datelor.

Am creat o aplicație simplă care utilizează algoritmul RSA pentru a cripta și decripta mesaje. Am generat cheile RSA, am citit un mesaj de la utilizator, l-am criptat cu cheia publică, am afișat mesajul criptat în format Base64, apoi l-am decriptat cu cheia privată și am afișat mesajul decriptat.

În final, acest proiect ne-a oferit o înțelegere de bază a algoritmului RSA și a modului în care pot fi generate și utilizate cheile RSA pentru a asigura securitatea comunicațiilor și a datelor. Algoritmul RSA rămâne un instrument esențial în domeniul criptografiei și este folosit într-o varietate de aplicații pentru protejarea informațiilor sensibile.

**Bibliografie:**

Codul sursă [17.09.2023] Disponibil:

https://www.devglan.com/online-tools/rsa-encryption-decryption

**Anexa:**

**from** cryptography.hazmat.backends **import** default\_backend

**from** cryptography.hazmat.primitives.asymmetric **import** rsa

**from** cryptography.hazmat.primitives **import** serialization

**from** cryptography.hazmat.primitives **import** hashes

**from** cryptography.hazmat.primitives.asymmetric **import** padding

**import** base64

*# Generăm o pereche de chei RSA*

private\_key = rsa.generate\_private\_key(

public\_exponent=65537,

key\_size=2048,

backend=default\_backend()

)

*# Serializăm cheia privată pentru a o putea stoca sau partaja*

private\_pem = private\_key.private\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PrivateFormat.PKCS8,

encryption\_algorithm=serialization.NoEncryption()

)

*# Obținem și cheia publică asociată cu cheia privată*

public\_key = private\_key.public\_key()

*# Serializăm cheia publică pentru a o putea partaja*

public\_pem = public\_key.public\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PublicFormat.SubjectPublicKeyInfo

)

*# Afișăm cheile*

**print**("Cheia privată:")

**print**(private\_pem.decode())

**print**("**\n**Cheia publică:")

**print**(public\_pem.decode())

*# Citim textul de la tastatură pentru criptare*

plaintext\_message = input("Introduceți textul pe care doriți să-l criptați: ").encode()

*# Criptăm mesajul folosind cheia publică*

ciphertext = public\_key.encrypt(

plaintext\_message,

padding.OAEP(

mgf=padding.MGF1(algorithm=hashes.SHA1()), *# Folosim SHA-1 aici pentru MGF1*

algorithm=hashes.SHA1(), *# Folosim SHA-1 pentru padding*

label=None

)

)

*# Convertim mesajul criptat în șir Base64*

ciphertext\_base64 = base64.b64encode(ciphertext).decode()

*# Afișăm mesajul criptat în format Base64*

**print**("**\n**Mesaj criptat (Base64):")

**print**(ciphertext\_base64)

*# Decriptăm mesajul folosind cheia privată*

decoded\_ciphertext = base64.b64decode(ciphertext\_base64)

decrypted\_message = private\_key.decrypt(

decoded\_ciphertext,

padding.OAEP(

mgf=padding.MGF1(algorithm=hashes.SHA1()), *# Folosim SHA-1 aici pentru MGF1*

algorithm=hashes.SHA1(), *# Folosim SHA-1 pentru padding*

label=None

)

)

*# Afișăm mesajul decriptat*

**print**("**\n**Mesaj decriptat:")

**print**(decrypted\_message.decode())