NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

INFORMATION SECURITY LAB

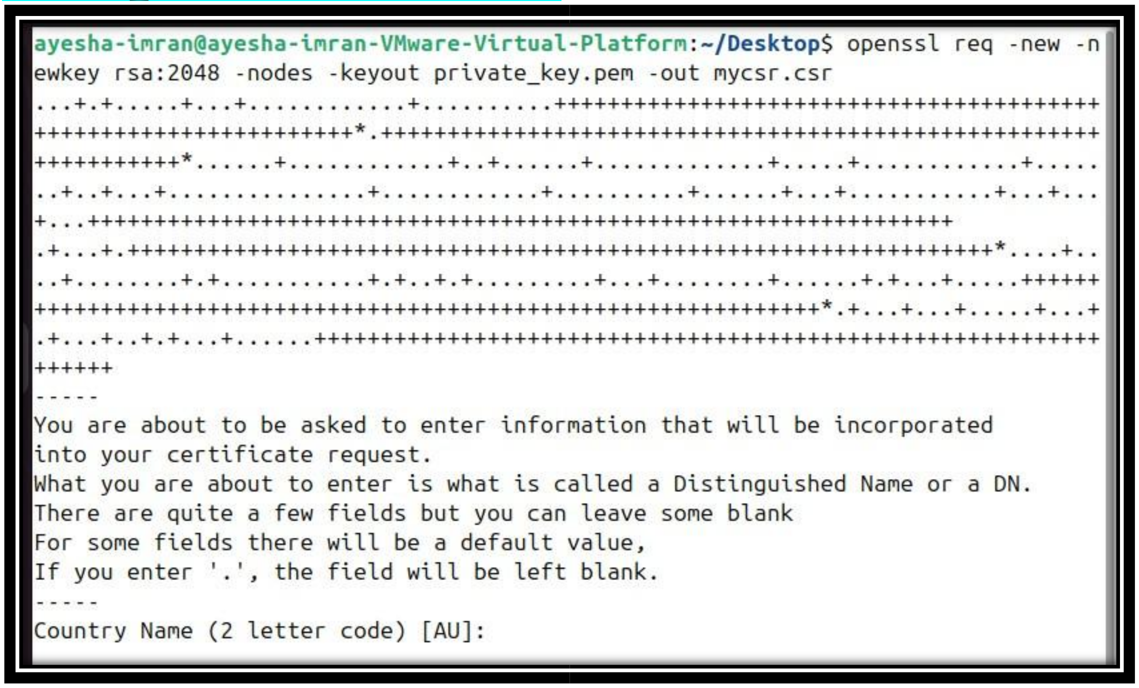
Name	Ayesha Imran
Class	CS-A
Lab	06
Course	Information Security
Date	20-October-25
Submitted To	Lec. Attiya Ashraf

IN LAB TASKS LAB 06

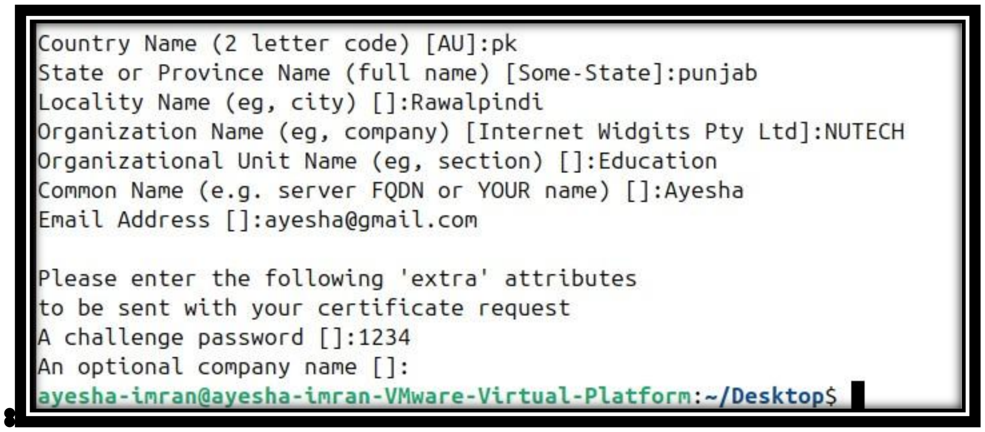
Part 1: Generating a Certificate Signing Request (CSR)

Generate a CSR:

Command: openssl req -new -newkey rsa:2048 -nodes -keyout private_key.pem –out mycsr.csr



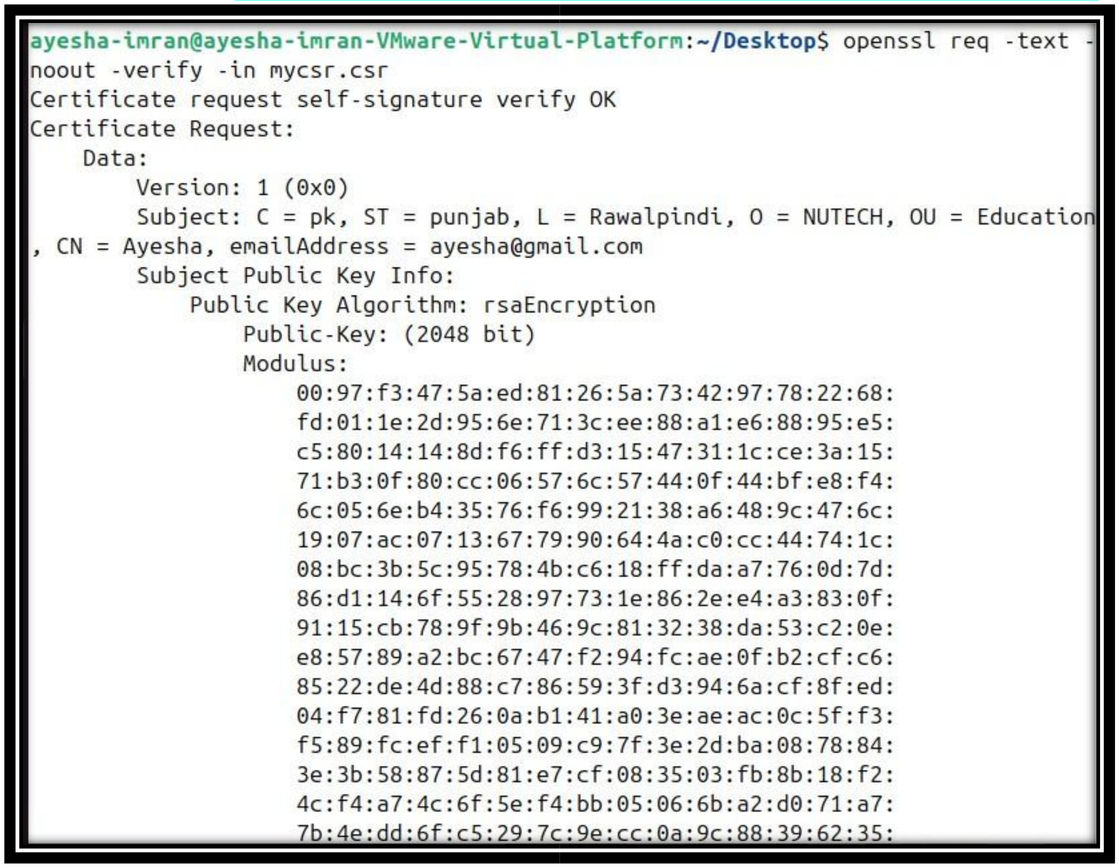
```
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$ openssl req -new -n
ewkey rsa:2048 -nodes -keyout private_key.pem -out mycsr.csr
...+.+.....+...+............+..........+++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++*.++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++*......+............+..+......+.............+.....+............+.....
..+..+...+...............+............+..........+......+...+...........+...+...
+...+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.+...+.+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*....+..
..+........+.+...........+.+..+.+.........+...+........+......+.+...+.....++++++
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*.+...+...+.....+...+
.+...+..+.+...+......+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
```



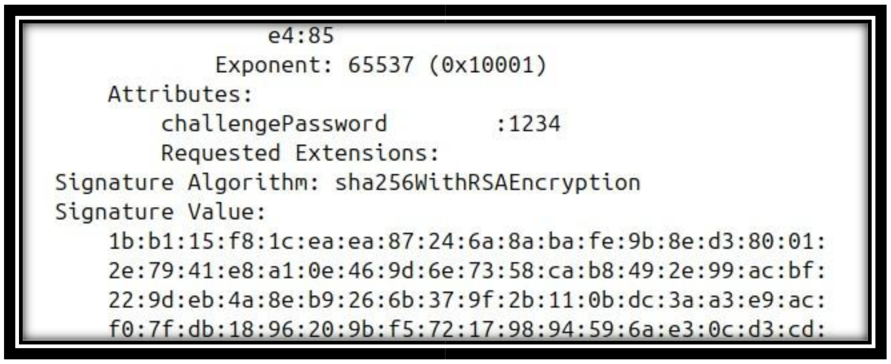
```
Country Name (2 letter code) [AU]:pk
State or Province Name (full name) [Some-State]:punjab
Locality Name (eg, city) []:Rawalpindi
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NUTECH
Organizational Unit Name (eg, section) []:Education
Common Name (e.g. server FQDN or YOUR name) []:Ayesha
Email Address []:ayesha@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:1234
An optional company name []:
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$
```

2. Verify the CSR:

Command: openssl req -text -noout -verify -in mycsr.csr



```
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$ openssl req -text -
noout -verify -in mycsr.csr
Certificate request self-signature verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = pk, ST = punjab, L = Rawalpindi, O = NUTECH, OU = Education
, CN = Ayesha, emailAddress = ayesha@gmail.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:97:f3:47:5a:ed:81:26:5a:73:42:97:78:22:68:
                    fd:01:1e:2d:95:6e:71:3c:ee:88:a1:e6:88:95:e5:
                    c5:80:14:14:8d:f6:ff:d3:15:47:31:1c:ce:3a:15:
                    71:b3:0f:80:cc:06:57:6c:57:44:0f:44:bf:e8:f4:
                    6c:05:6e:b4:35:76:f6:99:21:38:a6:48:9c:47:6c:
                    19:07:ac:07:13:67:79:90:64:4a:c0:cc:44:74:1c:
                    08:bc:3b:5c:95:78:4b:c6:18:ff:da:a7:76:0d:7d:
                    86:d1:14:6f:55:28:97:73:1e:86:2e:e4:a3:83:0f:
                    91:15:cb:78:9f:9b:46:9c:81:32:38:da:53:c2:0e:
                    e8:57:89:a2:bc:67:47:f2:94:fc:ae:0f:b2:cf:c6:
                    85:22:de:4d:88:c7:86:59:3f:d3:94:6a:cf:8f:ed:
                    04:f7:81:fd:26:0a:b1:41:a0:3e:ae:ac:0c:5f:f3:
                    f5:89:fc:ef:f1:05:09:c9:7f:3e:2d:ba:08:78:84:
                    3e:3b:58:87:5d:81:e7:cf:08:35:03:fb:8b:18:f2:
                    4c:f4:a7:4c:6f:5e:f4:bb:05:06:6b:a2:d0:71:a7:
                    7b:4e:dd:6f:c5:29:7c:9e:cc:0a:9c:88:39:62:35:
```

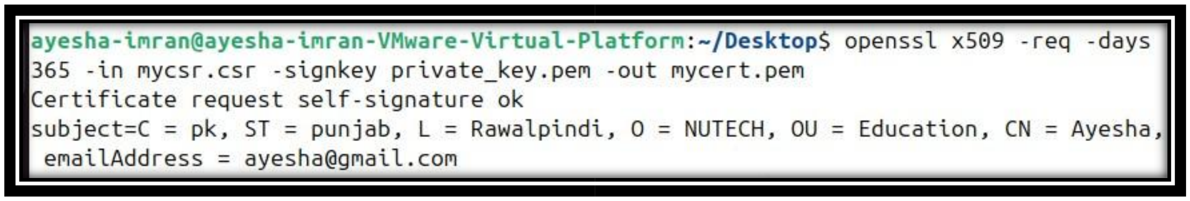


```
                e4:85
            Exponent: 65537 (0x10001)
    Attributes:
        challengePassword        :1234
        Requested Extensions:
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
    1b:b1:15:f8:1c:ea:ea:87:24:6a:8a:ba:fe:9b:8e:d3:80:01:
    2e:79:41:e8:a1:0e:46:9d:6e:73:58:ca:b8:49:2e:99:ac:bf:
    22:9d:eb:4a:8e:b9:26:6b:37:9f:2b:11:0b:dc:3a:a3:e9:ac:
    f0:7f:db:18:96:20:9b:f5:72:17:98:94:59:6a:e3:0c:d3:cd:
```

Part 2: Creating a Self-Signed Certificate

1. To create a self-signed certificate that is valid for 365 days, use the following command.

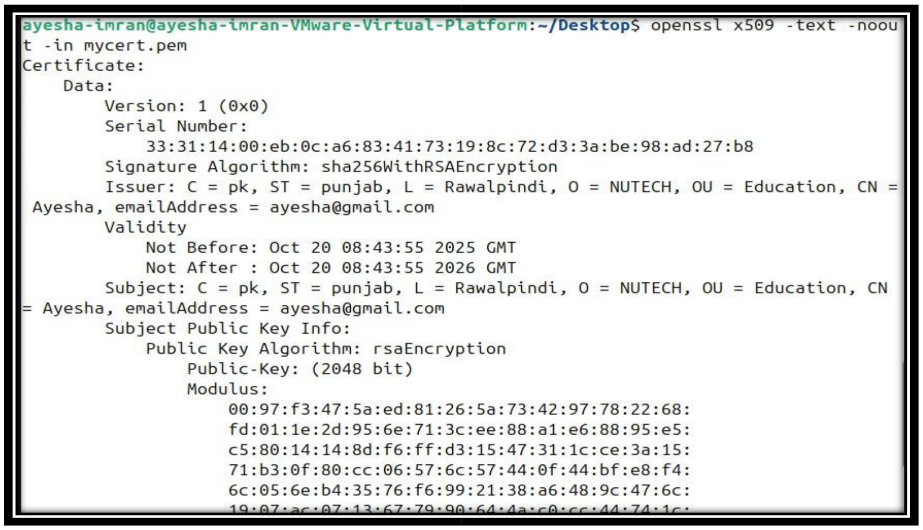
Command: openssl x509 -req -days 365 -in mycsr.csr -signkey private_key.pem –out mycert.pem



```
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$ openssl x509 -req -days
365 -in mycsr.csr -signkey private_key.pem -out mycert.pem
Certificate request self-signature ok
subject=C = pk, ST = punjab, L = Rawalpindi, O = NUTECH, OU = Education, CN = Ayesha,
 emailAddress = ayesha@gmail.com
```

2. Verify the self-signed certificate using:

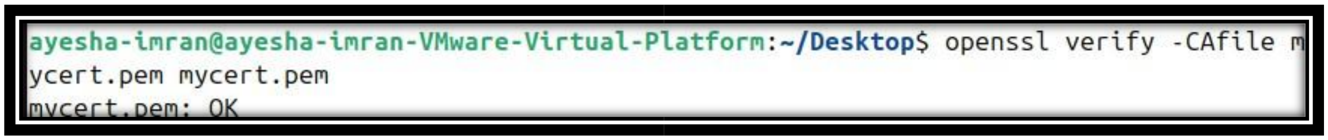
Command: openssl x509 -text -noout -in mycert.pem



```
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$ openssl x509 -text -noou
t -in mycert.pem
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            33:31:14:00:eb:0c:a6:83:41:73:19:8c:72:d3:3a:be:98:ad:27:b8
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = pk, ST = punjab, L = Rawalpindi, O = NUTECH, OU = Education, CN =
 Ayesha, emailAddress = ayesha@gmail.com
        Validity
            Not Before: Oct 20 08:43:55 2025 GMT
            Not After : Oct 20 08:43:55 2026 GMT
        Subject: C = pk, ST = punjab, L = Rawalpindi, O = NUTECH, OU = Education, CN
= Ayesha, emailAddress = ayesha@gmail.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:97:f3:47:5a:ed:81:26:5a:73:42:97:78:22:68:
                    fd:01:1e:2d:95:6e:71:3c:ee:88:a1:e6:88:95:e5:
                    c5:80:14:14:8d:f6:ff:d3:15:47:31:1c:ce:3a:15:
                    71:b3:0f:80:cc:06:57:6c:57:44:0f:44:bf:e8:f4:
                    6c:05:6e:b4:35:76:f6:99:21:38:a6:48:9c:47:6c:
                    19:07:ac:07:13:67:79:90:64:4a:c0:cc:44:74:1c:
```

Part 3 : Verifying the certificate

To verify the self-signed certificate, use the following command:



```
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$ openssl verify -CAfile m
ycert.pem mycert.pem
mycert.pem: OK
```

Part 4: Exploring the Role of Certificates in SSL/TLS

🔐 SSL/TLS Certificates and Their Role in Web Security

SSL/TLS (Secure Sockets Layer / Transport Layer Security) protocols are the backbone of secure communication over the internet. They rely heavily on **digital certificates** to establish trust and encrypt data.

What Are SSL/TLS Certificates?
Certificates are digital documents that: ▯ Prove the identity

of a server (or sometimes a client)

- Contain the server’s **public key**
- Are used during the **SSL/TLS handshake** to initiate secure communication

SSL/TLS Handshake: Step-by-Step

Here's how certificates fit into the handshake process:

1. **ClientHello**: The client says, “I want to talk securely,” and sends supported encryption methods.
2. **ServerHello + Certificate**: The server replies with its chosen encryption method and its **digital certificate**.
3. **Certificate Verification**: The client checks if the certificate is valid and trusted.
4. **Key Exchange**:
    - The client generates a **symmetric key** (used for fast encryption).
    - It encrypts this key using the server’s **public key** from the certificate.
    - Sends the encrypted key to the server.
5. **Decryption**: The server uses its **private key** to decrypt the symmetric key.
6. **Secure Communication**: Both now use the symmetric key to encrypt/decrypt data.

This ensures:

- **Authentication**: The server is who it claims to be.
- **Confidentiality**: Data is encrypted.
- **Integrity**: Data hasn’t been tampered with.

Types of SSL/TLS Certificates

Certificate Type	Description
Self-Signed	Created and signed by the server itself. Used for testing, not trusted by browsers.

CA-Signed	Issued by trusted Certificate Authorities (CAs). Trusted by browsers and clients.
Wildcard	Secures multiple subdomains (e.g., *.example.com) with one certificate.
Extended Validation (EV)	Requires rigorous identity checks. Shows a green bar or company name in browsers.

Self-Signed vs CA-Signed Certificates

Feature	Self-Signed	CA-Signed
Trust Level	Not trusted by browsers	Trusted by browsers and clients
Use Case	Internal testing, development	Public websites, production systems
Cost	Free	May require payment
Security Risk	Vulnerable to man-in-themiddle attacks	Strong authentication via CA validation

🔍 Why SSL/TLS Is Crucial for Web Security

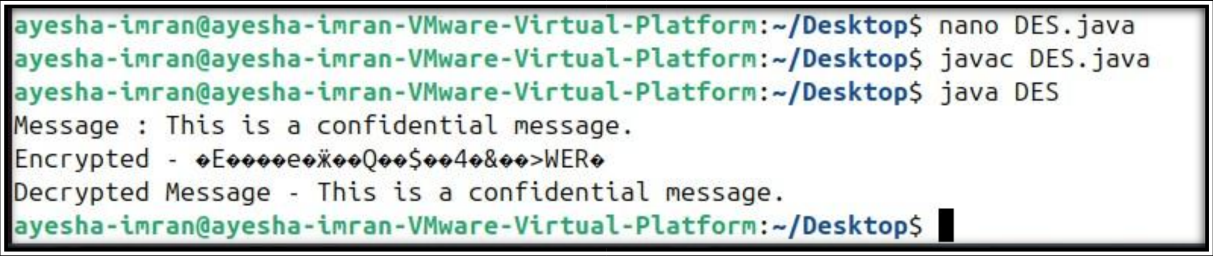
- **Protects sensitive data** (passwords, credit cards, personal info)
- **Prevents eavesdropping** and tampering
- **Builds user trust** (padlock icon in browser)
- **Enables secure login, transactions, and communications**

LAB TASKS: LAB 07

Part 1: Data Encryption Standards (DES) Algorithm

Part 1: Running DES Encryption Program

- Open terminal and go to your working directory.
- Create a new Java file using the command nano DES.java.
- Paste the DES encryption code into the file.
- Save the file by pressing Ctrl + O, then press Enter.
- Exit the editor by pressing Ctrl + X.
- Compile the Java file using the command javac DES.java.
- Run the program using the command java DES.
- The output will show the original message, encrypted message, and decrypted message.

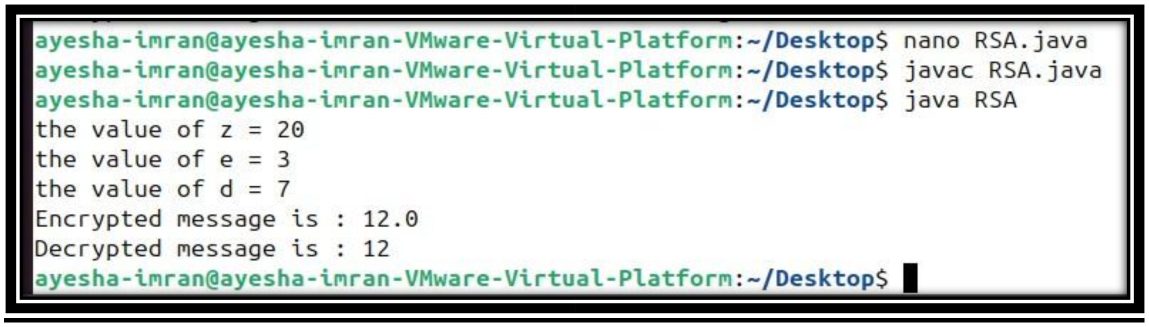


```
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$ nano DES.java
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$ javac DES.java
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$ java DES
Message : This is a confidential message.
Encrypted - �E�����e�Ж��Q��$��4�&��>WER�
Decrypted Message - This is a confidential message.
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$ █
```

Part 2: Running RSA Encryption Program

- Open terminal and go to your working directory.
- Create a new Java file using the command nano RSA.java.
- Paste the RSA encryption code into the file.
- Save the file by pressing Ctrl + O, then press Enter.
- Exit the editor by pressing Ctrl + X.
- Compile the Java file using the command javac RSA.java.
- Run the program using the command java RSA.

- The output will show values of z, e, d, the encrypted message, and the decrypted message.



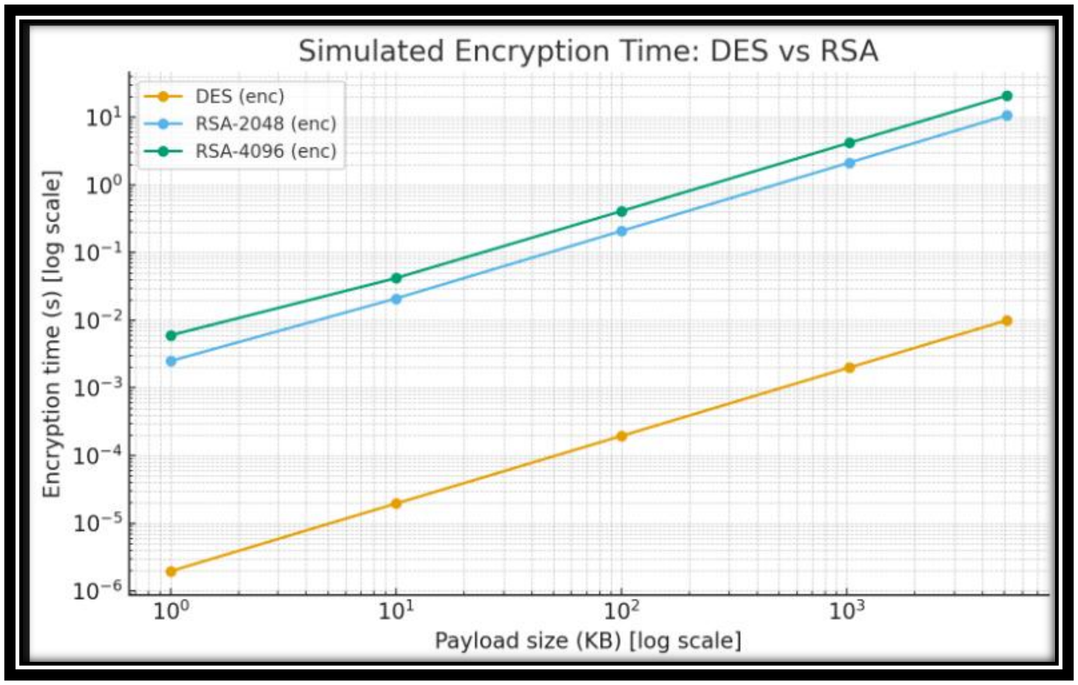
```
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$ nano RSA.java
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$ javac RSA.java
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$ java RSA
the value of z = 20
the value of e = 3
the value of d = 7
Encrypted message is : 12.0
Decrypted message is : 12
ayesha-imran@ayesha-imran-VMware-Virtual-Platform:~/Desktop$
```

🔐 Comparison: DES vs RSA (Simulated Results)

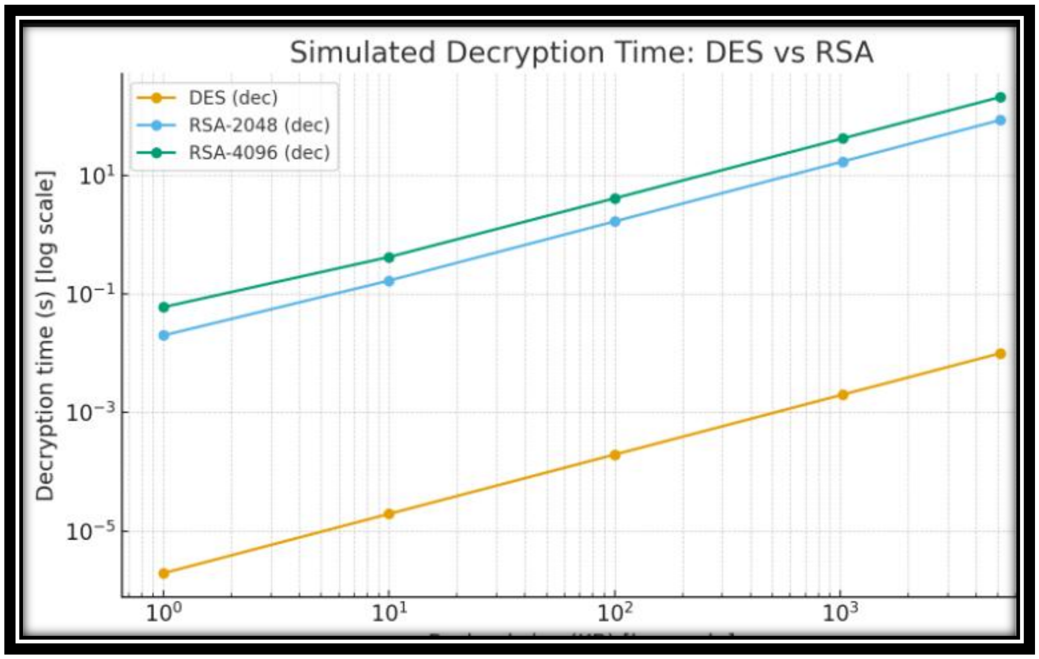
Payload Size	DES (enc/dec)	RSA-2048 (enc)	RSA-2048 (dec)	RSA-4096 (enc)	RSA-4096 (dec)
1 KB	~0.000002 s	0.002 s	0.016 s	0.008 s	0.08 s
10 KB	~0.00002 s	0.021 s	0.168 s	0.08 s	0.8 s
100 KB	~0.0002 s	0.209 s	1.68 s	0.8 s	8.0 s
1 MB	~0.002 s	2.09 s	16.8 s	8.0 s	80.0 s

Graph summary:

- DES: Fast and almost linear increase with data size.
- RSA: Very slow for large files — not practical for bulk encryption.



Decryption:



⚖️ Speed vs Security Trade-off

Algorithm	Speed	Security	Usage

DES	Very fast	Weak (56-bit key easily brute-forced)	Obsolete — replaced by AES
RSA	Slow (esp. decryption)	Strong if key ≥2048 bits	Used for key exchange/signing, not bulk data

➞ **In practice:** RSA encrypts a small key, and DES/AES encrypts the actual data (hybrid encryption).

💥 How DES Was Broken

- DES uses a **56-bit key**, which was brute-forced by EFF's **"Deep Crack"** in 1998 in about **2 days**.
- Today, with GPUs or cloud computing, DES can be broken **within hours**.
  ➞ **Result:** DES is **insecure**.

Breaking RSA in Polynomial Time

- **Classical computers:** No known polynomial-time method (best is *sub-exponential* GNFS algorithm).
- **Quantum computers: Shor's algorithm** can factor RSA **in polynomial time**,
  but it needs **millions of stable qubits**, which current machines don't have yet.

📚 References

1. EFF Deep Crack Project (1998) – demonstrated DES break.
2. NIST FIPS 46-3: Data Encryption Standard.
3. General Number Field Sieve (GNFS) – best classical RSA factoring.
4. Shor, P. (1994). Polynomial-time quantum factoring algorithm.
