APPLIED CRYPTOGRAPHY

Lab 8 : OpenSSL Lab

Name: Siri N Shetty SRN: PES2UG22CS556

Problem 1: OpenSSL from the command line

Step 1 : Run the 4 commands

Expected Deliverables -

i) Output Screenshot (Terminal should have SRN visible) for step 1 showing the 4 cmds

```
[11/19/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl version -a
OpenSSL 1.1.1f  31 Mar 2020
built on: Tue Nov 24 16:18:03 2020 UTC
platform: linux-x86_64
options:  bn(64,64) rc4(8x,int) des(int) idea(int) blowfish(ptr)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -O3 -DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOPENSSL_I
A32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -
DMD5_ASM -DAESNI_ASM -DVPAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM -DNDEBUG
OPENSSLDIR: "/usr/local/ssl"
ENGINESDIR: "/usr/local/lib/engines-1.1"
Seeding source: os-specific
```

```
[11/19/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl help
Standard commands
asn1parse         ca                ciphers           cms
crl               crl2pkcs7         dgst              dhparam
dsa               dsaparam          ec                ecparam
enc               engine            errstr            gendsa
genpkey           genrsa            help              list
nseq              ocsp              passwd            pkcs12
pkcs7             pkcs8             pkey              pkeyparam
pkeyutl           prime             rand              rehash
req               rsa               rsautl            s_client
s_server          s_time            sess_id           smime
speed             spkac             srp               storeutl
ts                verify            version           x509

Message Digest commands (see the `dgst' command for more details)
blake2b512        blake2s256        gost              md4
md5               mdc2              rmd160            sha1
sha224            sha256            sha3-224          sha3-256
sha3-384          sha3-512          sha384            sha512
sha512-224        sha512-256        shake128          shake256
sm3

Cipher commands (see the `enc' command for more details)
aes-128-cbc       aes-128-ecb       aes-192-cbc       aes-192-ecb
aes-256-cbc       aes-256-ecb       aria-128-cbc      aria-128-cfb
aria-128-cfb1     aria-128-cfb8     aria-128-ctr      aria-128-ecb
aria-128-ofb      aria-192-cbc      aria-192-cfb      aria-192-cfb1
aria-192-cfb8     aria-192-ctr      aria-192-ecb      aria-192-ofb
aria-256-cbc      aria-256-cfb      aria-256-cfb1     aria-256-cfb8
```

```
[11/19/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl ciphers -v
TLS_AES_256_GCM_SHA384  TLSv1.3 Kx=any      Au=any  Enc=AESGCM(256) Mac=AEAD
TLS_CHACHA20_POLY1305_SHA256 TLSv1.3 Kx=any      Au=any  Enc=CHACHA20/POLY1305(256) Mac=AEAD
TLS_AES_128_GCM_SHA256  TLSv1.3 Kx=any      Au=any  Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=DH       Au=RSA  Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
ECDHE-RSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH     Au=RSA  Enc=CHACHA20/POLY1305(256) Mac=AEAD
DHE-RSA-CHACHA20-POLY1305 TLSv1.2 Kx=DH       Au=RSA  Enc=CHACHA20/POLY1305(256) Mac=AEAD
ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AESGCM(128) Mac=AEAD
ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AESGCM(128) Mac=AEAD
DHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=DH       Au=RSA  Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AES(256)  Mac=SHA384
ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AES(256)  Mac=SHA384
DHE-RSA-AES256-SHA256   TLSv1.2 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA256
ECDHE-ECDSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AES(128)  Mac=SHA256
ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AES(128)  Mac=SHA256
DHE-RSA-AES128-SHA256   TLSv1.2 Kx=DH       Au=RSA  Enc=AES(128)  Mac=SHA256
ECDHE-ECDSA-AES256-SHA  TLSv1 Kx=ECDH     Au=ECDSA Enc=AES(256)  Mac=SHA1
ECDHE-RSA-AES256-SHA    TLSv1 Kx=ECDH     Au=RSA  Enc=AES(256)  Mac=SHA1
DHE-RSA-AES256-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA1
ECDHE-ECDSA-AES128-SHA  TLSv1 Kx=ECDH     Au=ECDSA Enc=AES(128)  Mac=SHA1
ECDHE-RSA-AES128-SHA    TLSv1 Kx=ECDH     Au=RSA  Enc=AES(128)  Mac=SHA1
DHE-RSA-AES128-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(128)  Mac=SHA1
RSA-PSK-AES256-GCM-SHA384 TLSv1.2 Kx=RSAPSK   Au=RSA  Enc=AESGCM(256) Mac=AEAD
DHE-PSK-AES256-GCM-SHA384 TLSv1.2 Kx=DHEPSK   Au=PSK  Enc=AESGCM(256) Mac=AEAD
RSA-PSK-CHACHA20-POLY1305 TLSv1.2 Kx=RSAPSK   Au=RSA  Enc=CHACHA20/POLY1305(256) Mac=AEAD
DHE-PSK-CHACHA20-POLY1305 TLSv1.2 Kx=DHEPSK   Au=PSK  Enc=CHACHA20/POLY1305(256) Mac=AEAD
ECDHE-PSK-CHACHA20-POLY1305 TLSv1.2 Kx=ECDHEPSK Au=PSK  Enc=CHACHA20/POLY1305(256) Mac=AEAD
```

```
[11/19/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl s_client -connect www.google.com:443 -showcerts
CONNECTED(00000003)
depth=2 C = US, O = Google Trust Services LLC, CN = GTS Root R1
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=1 C = US, O = Google Trust Services, CN = WR2
verify return:1
depth=0 CN = www.google.com
verify return:1
---
Certificate chain
 0 s:CN = www.google.com
   i:C = US, O = Google Trust Services, CN = WR2
-----BEGIN CERTIFICATE-----
MIIEWTCCA0GgAwIBAgIRAJ2cAzVYnqNCEHqwkZPYO2IwDQYJKoZIhvcNAQELBQAw
OzELMAkGA1UEBhMCVVMxHjAcBgNVBAoTFUdvb2dsZSBUcnVzdCBTZXJ2aWNlczEM
MAoGA1UEAxMDV1IyMB4XDTI0MTAyMTA4Mzg0NVoXDTI1MDExMzA4Mzg0NFowGTEX
MBUGA1UEAxMOd3d3Lmdvb2dsZS5jb20wWTATBgcqhkjOPQIBBggqhkjOPQMBBwNC
AAReIlhtyqRSkOSub0oJZ+urCnuQCfHorA21lF6ZwayGaorSZE6eI9ddnffWdi/N
qtni9oRSHO7NiQ1dEQQr8sWMo4ICQzCCAj8wDgYDVR0PAQH/BAQDAgeAMBMGA1Ud
JQQMMAoGCCsGAQUFBwMBMAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFMNA0KyX4PFE
qhcU7asRwZsAZ64lMB8GA1UdIwQYMBaAFN4bHu15FdQ+NyTDIbvsNDltQrIwMFgG
CCsGAQUFBwEBBEwwSjAhBggrBgEFBQcwAYYVaHR0cDovL28ucGtpLmdvb2cvd3Iy
MCUGCCsGAQUFBzAChhlodHRwOi8vaS5wa2kuZ29vZv93ciIuY3J0MBkGA1UdEQOS
TQR86pMWjEJzV3ydaUlaP9jqXhBl84UBXsQZHq9FDJK1wYTuYZczSbX/wkXnuISb
1FySlQZRqZELxk0bFMOQWhn5vfqMxGR8u8i7CAFaQAlh7PRXpemi75Ve71g1SadL
V6ORMouhpkQMpijVLLQekm6wTcL/L2mfamCw93DrvK4GfG0TpnozzWUvmWU45jjY
9++z7n1pKqGkR52TeA==
-----END CERTIFICATE-----
 1 s:C = US, O = Google Trust Services, CN = WR2
   i:C = US, O = Google Trust Services LLC, CN = GTS Root R1
-----BEGIN CERTIFICATE-----
MIIFCzCCAvOgAwIBAgIQf/AFoHxM3tEArZ1mpRB7mDANBgkqhkiG9w0BAQsFADBH
MQswCQYDVQQGEwJVUzEiMCAGA1UEChMZR29vZ2xlIFRydXN0IFNlcnZpY2VzIExM
QzEUMBIGA1UEAxMLR1RTIFJvb3QgUjEwHhcNMjMxMjEzMDkwMDAwWhcNMjkwMjIw
MTQwMDAwWjA7MQswCQYDVQQGEwJVUzEeMBwGA1UEChMVR29vZ2xlIFRydXN0IFNl
cnZpY2VzMQwwCgYDVQQDEwNXUjIwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQCp/5x/RR5wqFOfytnlDd5GV1d9vI+aWqxG8YSau5HbyfsvAfuSCQAWXqAc
+MGr+XgvSszYhaLYWTwO8xj7sfUkDSbutltkdnwUxy96zqhMt/TZCPzfhyM1IKji
aeKMTi+xWfpqoh6zySBTGYLKNlNtYE3pAJH8do1cCA8Kwtzxc2vFE24KT3rC8qIc
```

```
qDTBU39CluVIQeuQRgwG3MuSxl7zRERDRilGoKb8uY45JzmxWuKxrfwT/478JuHU
/oTxUFqOl2stKnn7QGTq8z29W+GgBLCXSBxC9epaHM0myFH/FJlniXJfHeytWt0=
-----END CERTIFICATE-----
 2 s:C = US, O = Google Trust Services LLC, CN = GTS Root R1
   i:C = BE, O = GlobalSign nv-sa, OU = Root CA, CN = GlobalSign Root CA
-----BEGIN CERTIFICATE-----
MIIFYjCCBEqgAwIBAgIQd70NbNs2+RrqIQ/E8FjTDTANBgkqhkiG9w0BAQsFADBX
MQswCQYDVQQGEwJCRTEZMBcGA1UEChMQR2xvYmFsU2lnbiBudi1zYTEQMA4GA1UE
CxMHUm9vdCBDQTEbMBkGA1UEAxMSR2xvYmFsU2lnbiBSb290IENBMB4XDTIwMDYx
OTAwMDA0MloXDTI4MDEyODAwMDA0MlowRzELMAkGA1UEBhMCVVMxIjAgBgNVBAoT
GUdvb2dsZSBUcnVzdCBTZXJ2aWNlcyBMTEMxFDASBgNVBAMTC0dUUyBSb290IFIx
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAthECix7joXebO9y/lD63
ladAPKH9gvl9MgaCcfb2jH/76Nu8ai6Xl6OMS/kr9rH5zoQdsfnFl97vufKj6bwS
iV6nqlKr+CMny6SxnGPb15l+8Ape62im9MZaRw1NEDPjTrETo8gYbEvs/AmQ351k
KSUjB6G00j0uYODP0gmHu81I8E3CwnqIiru6z1kZ1q+PsAewnjHxgsHA3y6mbWwZ
DrXYfiYaRQM9sHmklCitD38m5agI/pboPGiUU+6DOogrFZYJsuB6jC511pzrp1Zk
```

```
Server certificate
subject=CN = www.google.com

issuer=C = US, O = Google Trust Services, CN = WR2

---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: ECDSA
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 4109 bytes and written 396 bytes
Verification error: unable to get local issuer certificate
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 256 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 20 (unable to get local issuer certificate)
---
read:errno=0
```

Problem 2 : Generating Keys

Step 1 : Run the cmd to generate a 2048-bit RSA key

Step 2 : Run the cmd to generate a 256-bit ECC key

Expected Deliverables -

i) Output Screenshot (Terminal should have SRN visible) for step 1

```
[11/19/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl genrsa -out rsa_key.pem 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................................................
.....................+++++
e is 65537 (0x010001)
```

ii) Output Screenshot (Terminal should have SRN visible) for step 2

```
[11/19/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl ecparam -genkey -name prime256v1 -out ecc_key.pem
[11/19/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl ec -in ecc_key.pem -text -noout
read EC key
Private-Key: (256 bit)
priv:
    04:09:d9:40:8f:5c:e8:f1:de:f1:27:7f:4b:46:51:
    5b:e2:b9:97:56:8c:cf:b1:13:83:92:86:ff:e4:46:
    43:0b
pub:
    04:e4:55:de:66:02:14:ba:f3:2a:ab:c8:13:32:4d:
    05:01:d4:29:fc:db:7e:19:60:dc:e8:92:d4:79:c0:
    96:43:1d:1f:27:79:3d:5c:62:02:43:fd:d8:9d:a0:
    20:f2:cf:aa:7a:da:6a:58:c8:60:06:44:c7:02:e8:
    c7:cd:38:d3:5e
ASN1 OID: prime256v1
NIST CURVE: P-256
```

Problem 3 : Encrypting and Decrypting Data

Step 1 : encrypt data using aes-256-cbc
Step 2 : decrypt data using aes-256-cbc
Step 3 : perform encryption and decryption using diff algorithm
Step 4 : implement password prompt
Step 5 : batch encryption

Expected Deliverables -

i) Output Screenshot (Terminal should have SRN visible) for step 1

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl enc -aes-256-cbc -in plaintext.txt -out ciphertext.enc -k helloworld
*** WARNING : deprecated key derivation used.
```

ii) Output Screenshot (Terminal should have SRN visible) for step 2.

```
11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl enc -aes-256-cbc -d -in ciphertext.enc -out plaintext_decrypted.txt -k helloworld
** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

iii) Output Screenshot (Terminal should have SRN visible) for step 3

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl enc -aes-128-cbc -in plaintext.txt -out ciphertext.enc -k helloworld
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl enc -aes-128-cbc -d -in ciphertext.enc -out plaintext_decrypted.txt -k helloworld
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

iv) Compare the results and understand the differences in security and performance.

Security:

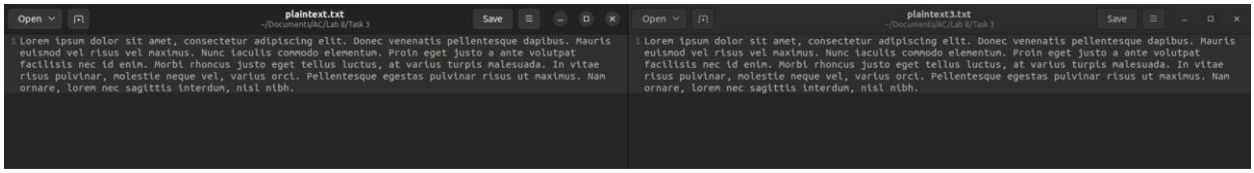
1. AES-256-CBC uses a 256-bit key length, offering stronger theoretical security
2. AES-128-CBC uses a 128-bit key length, which is considered cryptographically secure for most applications
3. The 256-bit version has a higher security margin against future quantum computing attacks.

Performance:

1. AES-128-CBC generally performs faster as it uses fewer rounds (10 rounds vs 14 rounds in AES-256)
2. Lower computational overhead with 128-bit, making it more efficient for resource-constrained systems
3. Memory usage is slightly lower with 128-bit encryption

v) Output Screenshot (Terminal should have SRN visible) for step 4

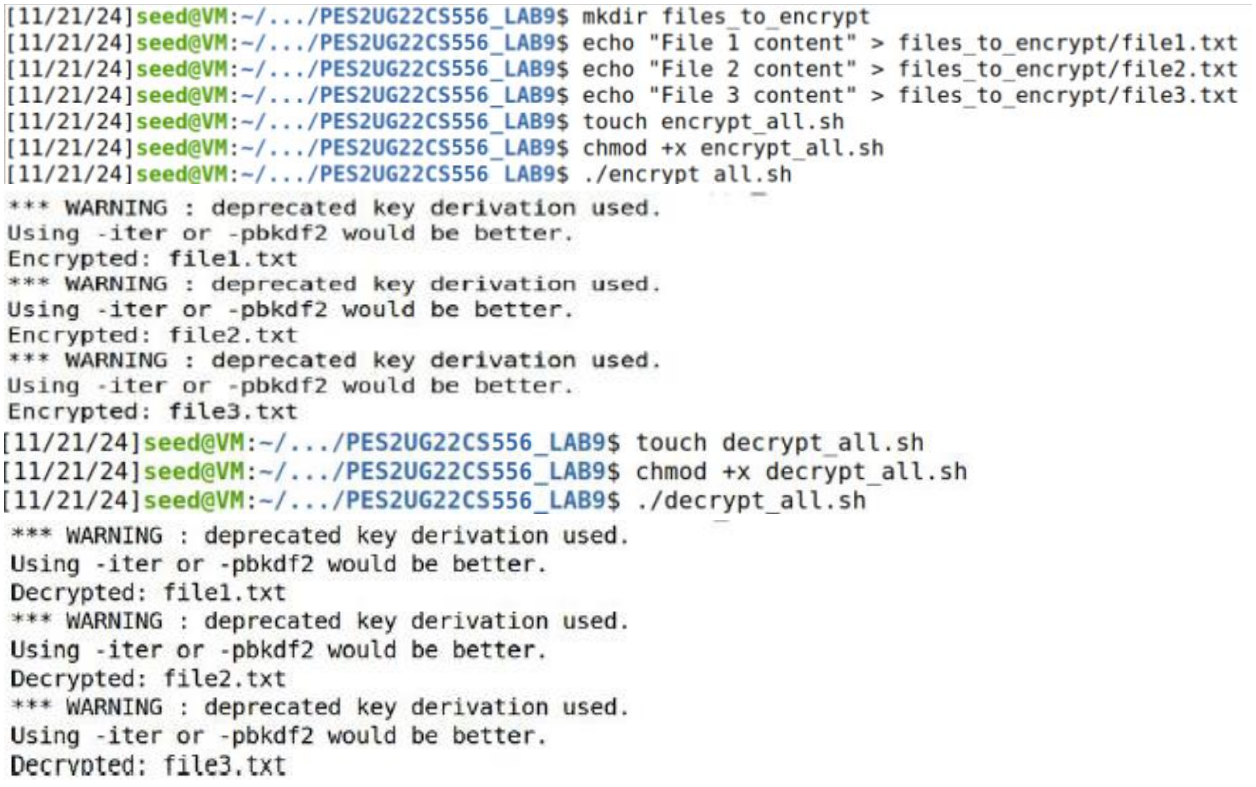
```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl enc -aes-256-cbc -in plaintext.txt -out ciphertext.enc -k mysecret
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl enc -aes-256-cbc -d -in ciphertext.enc -out plaintext_decrypted.txt -k mysecret
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$
```



vi) Code for step 5

```
#!/bin/bash
# Directory containing the files to encrypt
SOURCE_DIR="files_to_encrypt"
DEST_DIR="encrypted_files"
 # Password for encryption
PASSWORD="mysecret"
# Create destination directory if it doesn't exist
mkdir -p "$DEST_DIR"
# Encrypt each file
for FILE in "$SOURCE_DIR"/*; do
FILENAME=$(basename "$FILE")
openssl enc -aes-256-cbc -in "
$FILE" -out "$DEST_DIR/$FILENAME.enc" -k "$PASSWORD"
echo "Encrypted: $FILENAME"
done
```

vii) Output Screenshot (Terminal should have SRN visible) for step 5



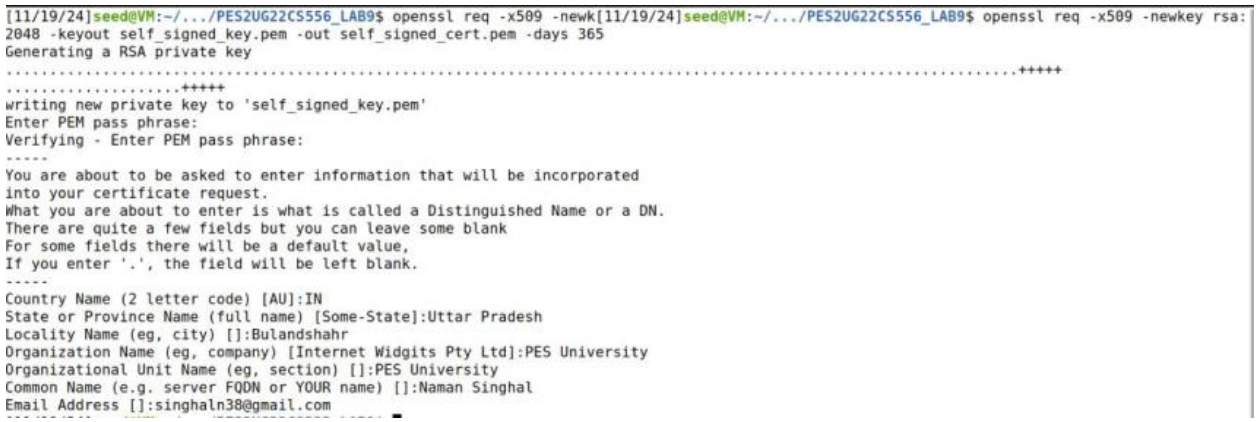
Problem 4 : Generating Certificates

Step 1 : Run the cmd to create a self-signed certificate

Step 2 : Run the cmd to generate a CSR

Expected Deliverables -

i) Output Screenshot (Terminal should have SRN visible) for step 1

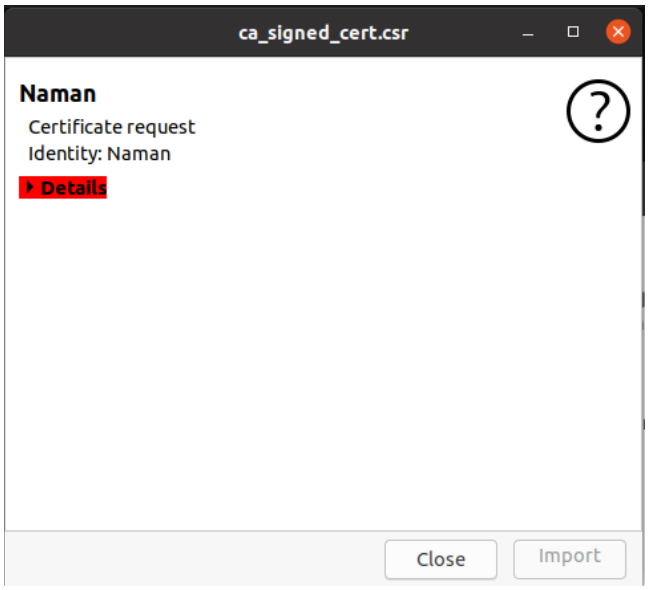


```
[11/19/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl req -x509 -newk[11/19/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl req -x509 -newkey rsa:
2048 -keyout self_signed_key.pem -out self_signed_cert.pem -days 365
Generating a RSA private key
.....................................................................................................+++++
.....................+++++
writing new private key to 'self_signed_key.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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]:IN
State or Province Name (full name) [Some-State]:Uttar Pradesh
Locality Name (eg, city) []:Bulandshahr
Organization Name (eg, company) [Internet Widgits Pty Ltd]:PES University
Organizational Unit Name (eg, section) []:PES University
Common Name (e.g. server FQDN or YOUR name) []:Naman Singhal
Email Address []:singhaln38@gmail.com
```

ii) Output Screenshot (Terminal should have SRN visible) for step 2

```
[11/19/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl req -newkey rsa:2048 -keyout ca_signed_key.pem -out ca_signed_cert.csr
Generating a RSA private key
............+++++
.+++++
writing new private key to 'ca_signed_key.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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]:IN
State or Province Name (full name) [Some-State]:Uttar Pradesh
Locality Name (eg, city) []:Bulandshahr
Organization Name (eg, company) [Internet Widgits Pty Ltd]:PESU
Organizational Unit Name (eg, section) []:PESU
Common Name (e.g. server FQDN or YOUR name) []:Naman
Email Address []:singhaln38@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:helloworld
An optional company name []:Vartasar
```



Problem 5 : OpenSSL create self-signed certificate

Step 1a : Create encrypted password file
Step 1b : Show decryption of password file
Step 2a : Create new directory to store certificate
Step 2b : Copy the encrypted password file to /root/certs
Step 3 : Generate private key
Step 4 : Create CSR
Step 5 : Create self-signed certificate using openssl x509
Step 6 : Show the contents of the files

Expected Deliverables -

i) Output Screenshot (Terminal should have SRN visible) for step 1

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ echo "secret" > mypass
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl enc -aes256 -pbkdf2 -salt -in mypass -out mypass.enc
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
```

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl enc -aes256 -pbkdf2 -salt -d -in mypass.enc
enter aes-256-cbc decryption password:
secret
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$
```

ii) Output Screenshot (Terminal should have SRN visible) for step 2

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ sudo mkdir /root/certs
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ sudo su
root@VM:/home/seed/Desktop/PES2UG22CS556_LAB9# cd /root/certs
root@VM:~/certs#
```

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ sudo cp mypass.enc /root/certs
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$
```

iii) Output Screenshot (Terminal should have SRN visible) for step 3

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl genrsa -des3 -passout file:mypass.enc -out server.key
Generating RSA private key, 2048 bit long modulus (2 primes)
...................+++++
....+++++
e is 65537 (0x010001)
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$
```

iv) Output Screenshot (Terminal should have SRN visible) for step 4

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl req -new -key server.key -out server.csr -passin file:mypass.enc
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]:IN
State or Province Name (full name) [Some-State]:UP
Locality Name (eg, city) []:BSR
Organization Name (eg, company) [Internet Widgits Pty Ltd]:PESU
Organizational Unit Name (eg, section) []:AC Lab
Common Name (e.g. server FQDN or YOUR name) []:Naman
Email Address []:singhaln38@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:helloworld
An optional company name []:Varta
```

v) Read and explain the use of -passout and -passin option.

1. `-passout` (Password Output):
    a. Used when creating/encrypting files that need a password
    b. Specifies how to generate or obtain the password for encryption
2. `-passin` (Password Input):
    a. Used when reading/decrypting password-protected files
    b. Specifies how to obtain the password for decryption

vi) Output Screenshot (Terminal should have SRN visible) for step 5

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt -passin file:mypass
s.enc
Signature ok
subject=C = IN, ST = KA, L = BLR, O = PESU, OU = AC Lab, CN = Siri, emailAddress = sirishetty.narendra@gmail.com
Getting Private key
```

vii) Output Screenshot (Terminal should have SRN visible) for step 6

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl rsa -noout -text -in server.key -passin file:mypass.enc
RSA Private-Key: (2048 bit, 2 primes)
modulus:
    00:c5:5a:10:7b:5c:69:ae:c4:c0:30:cf:89:be:61:
    bd:f2:36:9d:da:82:e7:88:69:3f:fc:d8:77:5c:dc:
    e8:3f:d3:a4:29:f5:9c:3c:7f:b5:fa:f7:e2:ee:68:
    86:01:97:55:a4:eb:9f:56:92:25:e8:68:d1:24:87:
    9b:51:77:44:c7:41:3a:f8:90:f3:4f:52:0e:fc:93:
    86:aa:5d:d4:0d:d3:0b:ad:5d:79:a4:71:6b:12:70:
    93:1f:42:19:e2:14:68:af:b3:bb:15:f0:75:6a:b1:
    50:9e:ee:ae:29:2e:53:64:ff:22:3a:a8:a3:6d:73:
    62:b1:11:95:7a:2a:94:d7:7a:9f:53:ab:b1:81:47:
    32:91:4d:06:26:a3:1d:a0:dd:68:0b:d0:44:38:ee:
    0b:54:cc:03:1a:1e:e8:9f:f5:16:41:7e:c8:e3:71:
    a7:90:68:a1:b5:6d:76:27:f0:5e:39:a8:ef:cc:33:
    d1:3a:5c:49:c6:aa:ef:3e:26:a0:c2:29:2a:5c:9e:
    e4:bc:04:69:27:0c:3d:b4:24:74:a6:bd:46:bb:03:
    7f:a3:23:b6:50:eb:ce:6a:f0:38:4f:9f:16:81:58:
    bf:ff:20:62:6e:b7:1e:f7:24:b2:d9:e5:f5:40:30:
    ff:ca:16:7f:92:fc:68:e0:06:15:13:a0:0d:c9:51:
    92:3b
publicExponent: 65537 (0x10001)
privateExponent:
    00:87:e7:fa:29:b9:fe:5d:88:c9:01:d4:2a:7b:9d:
    3b:fd:ad:77:0f:9f:ce:6a:b6:70:86:63:5b:ef:eb:
    81:55:53:1e:5e:76:f1:dd:07:e5:fe:aa:ee:f0:57:
    b2:d1:2d:b2:a1:1c:52:62:7f:ca:f3:3e:1b:a9:18:
    69:f1:b4:3c:fd:2b:02:bd:62:b4:ec:0f:0a:9b:0d:
    cd:53:4d:c2:56:b2:db:fb:cb:bf:95:6b:35:dd:41:
    01:50:29:69:41:b3:e4:53:fb:65:ff:39:d4:e2:60:
    b1:b0:81:96:16:6d:fc:a8:34:bb:11:c2:48:a9:7b:
    28:9f:e0:08:1c:55:6e:a2:e4:6b:13:e7:a3:62:29:
    4a:d5:aa:e1:0f:88:56:ae:45:d7:cc:7a:9d:7b:77:
```

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl req -noout -text -in server.csr -passin file:mypass.enc
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = KA, L = BLR, O = PESU, OU = AC Lab, CN = Siri, emailAddress = sirishetty.narendra@gmail.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c5:5a:10:7b:5c:69:ae:c4:c0:30:cf:89:be:61:
                    bd:f2:36:9d:da:82:e7:88:69:3f:fc:d8:77:5c:dc:
                    e8:3f:d3:a4:29:f5:9c:3c:7f:b5:fa:f7:e2:ee:68:
                    86:01:97:55:a4:eb:9f:56:92:25:e8:68:d1:24:87:
                    9b:51:77:44:c7:41:3a:f8:90:f3:4f:52:0e:fc:93:
                    86:aa:5d:d4:0d:d3:0b:ad:5d:79:a4:71:6b:12:70:
                    93:1f:42:19:e2:14:68:af:b3:bb:15:f0:75:6a:b1:
                    50:9e:ee:ae:29:2e:53:64:ff:22:3a:a8:a3:6d:73:
                    62:b1:11:95:7a:2a:94:d7:7a:9f:53:ab:b1:81:47:
                    32:91:4d:06:26:a3:1d:a0:dd:68:0b:d0:44:38:ee:
                    0b:54:cc:03:1a:1e:e8:9f:f5:16:41:7e:c8:e3:71:
                    a7:90:68:a1:b5:6d:76:27:f0:5e:39:a8:ef:cc:33:
                    d1:3a:5c:49:c6:aa:ef:3e:26:a0:c2:29:2a:5c:9e:
                    e4:bc:04:69:27:0c:3d:b4:24:74:a6:bd:46:bb:03:
                    7f:a3:23:b6:50:eb:ce:6a:f0:38:4f:9f:16:81:58:
                    bf:ff:20:62:6e:b7:1e:f7:24:b2:d9:e5:f5:40:30:
                    ff:ca:16:7f:92:fc:68:e0:06:15:13:a0:0d:c9:51:
                    92:3b
                Exponent: 65537 (0x10001)
        Attributes:
            unstructuredName         :Varta
            challengePassword        :helloworld
    Signature Algorithm: sha256WithRSAEncryption
         4a:d7:f7:c3:8f:d7:97:f8:a9:e7:9b:1e:69:8c:cd:53:98:46:
         6a:c5:f4:44:02:24:e6:b5:64:f0:43:2e:58:42:85:31:c7:9f:
```

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl x509 -noout -text -in server.crt -passin file:mypass.enc
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            69:64:01:a0:42:59:e7:50:02:f2:53:ea:23:cb:43:b3:65:53:4e:7f
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IN, ST = KA, L = BLR, O = PESU, OU = AC Lab, CN = Siri, emailAddress = sirishetty.narendra@gmail.com
        Validity
            Not Before: Nov 21 20:26:26 2024 GMT
            Not After : Nov 21 20:26:26 2025 GMT
        Subject: C = IN, ST = KA, L = BLR, O = PESU, OU = AC Lab, CN = Siri, emailAddress = sirishetty.narendra@gmail.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c5:5a:10:7b:5c:69:ae:c4:c0:30:cf:89:be:61:
                    bd:f2:36:9d:da:82:e7:88:69:3f:fc:d8:77:5c:dc:
                    e8:3f:d3:a4:29:f5:9c:3c:7f:b5:fa:f7:e2:ee:68:
                    86:01:97:55:a4:eb:9f:56:92:25:e8:68:d1:24:87:
                    9b:51:77:44:c7:41:3a:f8:90:f3:4f:52:0e:fc:93:
                    86:aa:5d:d4:0d:d3:0b:ad:5d:79:a4:71:6b:12:70:
                    93:1f:42:19:e2:14:68:af:b3:bb:15:f0:75:6a:b1:
                    50:9e:ee:ae:29:2e:53:64:ff:22:3a:a8:a3:6d:73:
                    62:b1:11:95:7a:2a:94:d7:7a:9f:53:ab:b1:81:47:
                    32:91:4d:06:26:a3:1d:a0:dd:68:0b:d0:44:38:ee:
                    0b:54:cc:03:1a:1e:e8:9f:f5:16:41:7e:c8:e3:71:
                    a7:90:68:a1:b5:6d:76:27:f0:5e:39:a8:ef:cc:33:
                    d1:3a:5c:49:c6:aa:ef:3e:26:a0:c2:29:2a:5c:9e:
                    e4:bc:04:69:27:0c:3d:b4:24:74:a6:bd:46:bb:03:
                    7f:a3:23:b6:50:eb:ce:6a:f0:38:4f:9f:16:81:58:
                    bf:ff:20:62:6e:b7:1e:f7:24:b2:d9:e5:f5:40:30:
                    ff:ca:16:7f:92:fc:68:e0:06:15:13:a0:0d:c9:51:
                    92:3b
```

Problem 6 : Certificate Validity Check Using OpenSSL

Step 1 : perform prerequisites and run the cmd to verify validity

Expected Deliverables -

i) Output Screenshot (Terminal should have SRN visible) for step 1

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl verify -CAfile server.crt server.crt
server.crt: OK
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$
```

ii) Why is it important to verify the validity of a certificate before trusting it in a secure communication setup?

Verifying a certificate is crucial because it:

1. Authenticates Identity:
   Confirms the entity you're communicating with is genuine.
2. Ensures Data Integrity:
   Prevents tampered or fake certificates from being used.
3. Enables Secure Encryption:
   Validates strong encryption for safe communication.
4. Checks Revocation/Expiration:
   Ensures the certificate is still valid and not revoked.
5. Prevents Attacks:
   Protects against Man-in-the-Middle attacks and unauthorized access.

Failure to verify can lead to data breaches, impersonation, and loss of trust in secure communications.

iii) What are some common reasons for certificate verification failures, and how can they be addressed?

Certificate verification failures occur due to expiration, revocation, mismatched CN, untrusted CAs, or incomplete chains. Solutions include renewing certificates, configuring chains correctly, and ensuring secure algorithms and accurate system time.

iv) How can you obtain a trusted CA certificate or bundle in real-world scenarios?

To obtain a trusted CA certificate or bundle:

1. Certificate Authority (CA): Purchase or get a free certificate from trusted CAs like DigiCert, Let's Encrypt, or GlobalSign.
2. Operating System or Browser Trust Stores: Use pre-installed CA certificates in your system or browser.
3. Let's Encrypt: Obtain free certificates using tools like Certbot.
4. Public Repositories: Download trusted CA certificates from repositories like Mozilla.
5. Self-Signed for Internal Use: Create your own CA for internal certificates.
6. CA Bundles: CAs often provide a bundle of root and intermediate certificates for full trust validation.

Problem 7 : Testing TLS Connections

Step 1 : test a TLS connection

Step 2 : Modify the openssl s_client command to check if the server's certificate is valid

Step 3 : Use OpenSSL to list the supported cipher suites by the server.

Step 4 : Try connecting to the server using different TLS/SSL protocols, such as TLS 1.2 and TLS 1.3.

Expected Deliverables -

i) Output Screenshot (Terminal should have SRN visible) for step 1

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl s_client -connect example.com:443
CONNECTED(00000003)
depth=1 C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = US, ST = California, L = Los Angeles, O = Internet\C2\A0Corporation\C2\A0for\C2\A0Assigned\C2\A0Names\C2\A0and\C2\A0Numbers, CN =
 www.example.org
verify return:1
---
Certificate chain
 0 s:C = US, ST = California, L = Los Angeles, O = Internet\C2\A0Corporation\C2\A0for\C2\A0Assigned\C2\A0Names\C2\A0and\C2\A0Numbers, CN = ww
w.example.org
   i:C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
 1 s:C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
   i:C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root G2
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIHbjCCBlagAwIBAgIQB1vO8waJyK3fE+Ua9K/hhzANBgkqhkiG9w0BAQsFADBZ
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMTMwMQYDVQQDEypE
aWdpQ2VydCBHbG9iYWwgRzIgVExTIFJTQSBTSEEyNTYgMjAyMCBDQTEwHhcNMjQw
MTMwMDAwMDAwWhcNMjUwMzAxMjM1OTU5WjCBljELMAkGA1UEBhMCVVMxEzARBgNV
BAgTCkNhbGlmb3JuaWExFDASBgNVBAcTC0xvcyBBbmdlbGVzMUIwQAYDVQQKDDlJ
bnRlcm5ldMKgQ29ycG9yYXRpb27CoGZvcsKgQXNzaWduZWTCoE5hbWVzwqBhbmTC
oE51bWJlcnMxGDAWBgNVBAMTD3d3dy5leGFtcGxlLm9yZzCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAIaFD7sO+cpf2fXgCjIsM9mqDgcpqC8IrXi9wga/
9y0rpqcnPVOmTMNLsid3INbBVEm4CNr5cKlh9rJJnWlX2vttJDRyLkfwBD+dsVvi
vGYxWTLmqX6/1LDUZPVrynv/cltemtg/1Aay88jcj2ZaRoRmqBgVeacIzgU8+zmJ
7236TnFSe7fkoKSclsBhPaQKcE3DjsluszJs8sdECQTdoFX9I6UgeLKFXtg7rRf/
hcW5dI0zubhXbrW8aWXbCzySVZn0c7RkJMpnTCiZzNxnPXnHFpwr5quqqjVyN/aB
KkjoP04Zmr+eRqoyk/+lslq0sS8eaYSSHbC5ja/yMWyVhvMCAwEAAaOCA/IwggPu
MB8GA1UdIwQYMBaAFHSFgMBmx9833s+9KTeqAx2+7c0XMB0GA1UdDgQWBBRM/tAS
TS4hz2v68vK4TEkCHTGRijCBgQYDVR0RBHoweIIPd3d3LmV4YW1wbGUub3Jnggtl
eGFtcGxlLm9yZ4IPZXhhbXBsZS5lZHWCC2V4YW1wbGUuY29tggtleGFtcGxlLm9y
```

ii) Output Screenshot (Terminal should have SRN visible) for step 2

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl s_client -connect example.com:443 -CAfile ./server.crt
CONNECTED(00000003)
depth=1 C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = US, ST = California, L = Los Angeles, O = Internet\C2\A0Corporation\C2\A0for\C2\A0Assigned\C2\A0Names\C2\A0and\C2\A0Numbers, CN =
 www.example.org
verify return:1
---
Certificate chain
 0 s:C = US, ST = California, L = Los Angeles, O = Internet\C2\A0Corporation\C2\A0for\C2\A0Assigned\C2\A0Names\C2\A0and\C2\A0Numbers, CN = ww
w.example.org
   i:C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
 1 s:C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
   i:C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root G2
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIHbjCCBlagAwIBAgIQB1vO8waJyK3fE+Ua9K/hhzANBgkqhkiG9w0BAQsFADBZ
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMTMwMQYDVQQDEypE
aWdpQ2VydCBHbG9iYWwgRzIgVExTIFJTQSBTSEEyNTYgMjAyMCBDQTEwHhcNMjQw
MTMwMDAwMDAwWhcNMjUwMzAxMjM1OTU5WjCBljELMAkGA1UEBhMCVVMxEzARBgNV
BAgTCkNhbGlmb3JuaWExFDASBgNVBAcTC0xvcyBBbmdlbGVzMUIwQAYDVQQKDDlJ
bnRlcm5ldMKgQ29ycG9yYXRpb27CoGZvcsKgQXNzaWduZWTCoE5hbWVzwqBhbmTC
oE51bWJlcnMxGDAWBgNVBAMTD3d3dy5leGFtcGxlLm9yZzCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAIaFD7sO+cpf2fXgCjIsM9mqDgcpqC8IrXi9wga/
9y0rpqcnPVOmTMNLsid3INbBVEm4CNr5cKlh9rJJnWlX2vttJDRyLkfwBD+dsVvi
vGYxWTLmqX6/1LDUZPVrynv/cltemtg/1Aay88jcj2ZaRoRmqBgVeacIzgU8+zmJ
7236TnFSe7fkoKSclsBhPaQKcE3DjsluszJs8sdECQTdoFX9I6UgeLKFXtg7rRf/
hcW5dI0zubhXbrW8aWXbCzySVZn0c7RkJMpnTCiZzNxnPXnHFpwr5quqqjVyN/aB
KkjoP04Zmr+eRqoyk/+lslq0sS8eaYSSHbC5ja/yMWyVhvMCAwEAAaOCA/IwggPu
MB8GA1UdIwQYMBaAFHSFgMBmx9833s+9KTeqAx2+7c0XMB0GA1UdDgQWBBRM/tAS
TS4hz2v68vK4TEkCHTGRijCBgQYDVR0RBHoweIIPd3d3LmV4YW1wbGUub3Jnggtl
eGFtcGxlLm9yZ4IPZXhhbXBsZS5lZHWCC2V4YW1wbGUuY29tggtleGFtcGxlLm9y
```

iii) Explain the significance of verifying the certificate's authenticity.

Verifying a certificate's authenticity ensures that you are communicating with the legitimate server and not an imposter. It prevents Man-in-the-Middle (MITM) attacks, ensures data integrity, and authenticates the server's identity. A valid certificate builds trust, helps meet security compliance standards, and protects against phishing and fraud. It guarantees that the data exchanged is secure and not tampered with

iv) Output Screenshot (Terminal should have SRN visible) for step 3

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl ciphers -v
TLS_AES_256_GCM_SHA384  TLSv1.3 Kx=any      Au=any  Enc=AESGCM(256) Mac=AEAD
TLS_CHACHA20_POLY1305_SHA256 TLSv1.3 Kx=any      Au=any  Enc=CHACHA20/POLY1305(256) Mac=AEAD
TLS_AES_128_GCM_SHA256  TLSv1.3 Kx=any      Au=any  Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=DH       Au=RSA  Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
ECDHE-RSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH     Au=RSA  Enc=CHACHA20/POLY1305(256) Mac=AEAD
DHE-RSA-CHACHA20-POLY1305 TLSv1.2 Kx=DH       Au=RSA  Enc=CHACHA20/POLY1305(256) Mac=AEAD
ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AESGCM(128) Mac=AEAD
ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AESGCM(128) Mac=AEAD
DHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=DH       Au=RSA  Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AES(256)  Mac=SHA384
ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AES(256)  Mac=SHA384
DHE-RSA-AES256-SHA256   TLSv1.2 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA256
ECDHE-ECDSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AES(128)  Mac=SHA256
ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AES(128)  Mac=SHA256
DHE-RSA-AES128-SHA256   TLSv1.2 Kx=DH       Au=RSA  Enc=AES(128)  Mac=SHA256
ECDHE-ECDSA-AES256-SHA  TLSv1 Kx=ECDH     Au=ECDSA Enc=AES(256)  Mac=SHA1
ECDHE-RSA-AES256-SHA    TLSv1 Kx=ECDH     Au=RSA  Enc=AES(256)  Mac=SHA1
DHE-RSA-AES256-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA1
ECDHE-ECDSA-AES128-SHA  TLSv1 Kx=ECDH     Au=ECDSA Enc=AES(128)  Mac=SHA1
ECDHE-RSA-AES128-SHA    TLSv1 Kx=ECDH     Au=RSA  Enc=AES(128)  Mac=SHA1
DHE-RSA-AES128-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(128)  Mac=SHA1
RSA-PSK-AES256-GCM-SHA384 TLSv1.2 Kx=RSAPSK   Au=RSA  Enc=AESGCM(256) Mac=AEAD
DHE-PSK-AES256-GCM-SHA384 TLSv1.2 Kx=DHEPSK   Au=PSK  Enc=AESGCM(256) Mac=AEAD
RSA-PSK-CHACHA20-POLY1305 TLSv1.2 Kx=RSAPSK   Au=RSA  Enc=CHACHA20/POLY1305(256) Mac=AEAD
DHE-PSK-CHACHA20-POLY1305 TLSv1.2 Kx=DHEPSK   Au=PSK  Enc=CHACHA20/POLY1305(256) Mac=AEAD
ECDHE-PSK-CHACHA20-POLY1305 TLSv1.2 Kx=ECDHEPSK Au=PSK  Enc=CHACHA20/POLY1305(256) Mac=AEAD
AES256-GCM-SHA384       TLSv1.2 Kx=RSA      Au=RSA  Enc=AESGCM(256) Mac=AEAD
PSK-AES256-GCM-SHA384   TLSv1.2 Kx=PSK      Au=PSK  Enc=AESGCM(256) Mac=AEAD
PSK-CHACHA20-POLY1305   TLSv1.2 Kx=PSK      Au=PSK  Enc=CHACHA20/POLY1305(256) Mac=AEAD
RSA-PSK-AES128-GCM-SHA256 TLSv1.2 Kx=RSAPSK   Au=RSA  Enc=AESGCM(128) Mac=AEAD
```

v) Discuss the importance of selecting strong cipher suites for security.

Selecting strong cipher suites is essential for ensuring secure encryption, data integrity, and authentication. Strong suites protect against modern attacks, such as brute force and MITM attacks, and ensure compliance with security standards. They also future-proof your systems by using cryptographically secure algorithms, preventing vulnerabilities from outdated encryption methods.

vi) Output Screenshot (Terminal should have SRN visible) for step 4

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl s_client -connect example.com:443 -tls1_2
CONNECTED(00000003)
depth=1 C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = US, ST = California, L = Los Angeles, O = Internet\C2\A0Corporation\C2\A0for\C2\A0Assigned\C2\A0Names\C2\A0and\C2\A0Numbers, CN =
 www.example.org
verify return:1
---
Certificate chain
 0 s:C = US, ST = California, L = Los Angeles, O = Internet\C2\A0Corporation\C2\A0for\C2\A0Assigned\C2\A0Names\C2\A0and\C2\A0Numbers, CN = ww
w.example.org
   i:C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
 1 s:C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
   i:C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root G2
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIHbjCCBlagAwIBAgIQB1vO8waJyK3fE+Ua9K/hhzANBgkqhkiG9w0BAQsFADBZ
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMTMwMQYDVQQDEypE
aWdpQ2VydCBHbG9iYWwgRzIgVExTIFJTQSBTSEEyNTYgMjAyMCBDQTEwHhcNMjQw
MTMwMDAwMDAwWhcNMjUwMzAxMjM1OTU5WjCBljELMAkGA1UEBhMCVVMxEzARBgNV
BAgTCkNhbGlmb3JuaWExFDASBgNVBAcTC0xvcyBBbmdlbGVzMUIwQAYDVQQKDDlJ
bnRlcm5ldMKgQ29yYXRpb27CoGZvcsKgQXNzaWduZWTCoE5hbWVzwqBhbmTC
oE51bWJlcnMxGDAWBgNVBAMTD3d3dy5leGFtcGxlLm9yZzCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAIaFD7sO+cpf2fXgCjIsM9mqDgcpqC8IrXi9wga/
9y0rpqcnPVOmTMNLsid3INbBVEm4CNr5cKlh9rJJnWlX2vttJDRyLkfwBD+dsVvi
vGYxWTLmqX6/1LDUZPVrynv/cltemtg/1Aay88jcj2ZaRoRmqBgVeacIzgU8+zmJ
7236TnFSe7fkoKSclsBhPaQKcE3Djs1uszJs8sdECQTdoFX9I6UgeLKFXtg7rRf/
hcW5dI0zubhXbrW8aWXbCzySVZn0c7RkJMpnTCiZzNxnPXnHFpwr5quqqjVyN/aB
KkjoP04Zmr+eRqoyk/+lslq0sS8eaYSSHbC5ja/yMWyVhvMCAwEAAaOCA/IwggPu
MB8GA1UdIwQYMBaAFHSFgMBmx9833s+9KTeqAx2+7c0XMB0GA1UdDgQWBBRM/tAS
```

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl s_client -connect example.com:443 -tls1_3
CONNECTED(00000003)
depth=1 C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = US, ST = California, L = Los Angeles, O = Internet\C2\A0Corporation\C2\A0for\C2\A0Assigned\C2\A0Names\C2\A0and\C2\A0Numbers, CN =
 www.example.org
verify return:1
---
Certificate chain
 0 s:C = US, ST = California, L = Los Angeles, O = Internet\C2\A0Corporation\C2\A0for\C2\A0Assigned\C2\A0Names\C2\A0and\C2\A0Numbers, CN = ww
w.example.org
   i:C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
 1 s:C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
   i:C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root G2
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIHbjCCBlagAwIBAgIQB1vO8waJyK3fE+Ua9K/hhzANBgkqhkiG9w0BAQsFADBZ
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMTMwMQYDVQQDEypE
aWdpQ2VydCBHbG9iYWwgRzIgVExTIFJTQSBTSEEyNTYgMjAyMCBDQTEwHhcNMjQw
MTMwMDAwMDAwWhcNMjUwMzAxMjM1OTU5WjCBljELMAkGA1UEBhMCVVMxEzARBgNV
BAgTCkNhbGlmb3JuaWExFDASBgNVBAcTC0xvcyBBbmdlbGVzMUIwQAYDVQQKDDlJ
bnRlcm5ldMKgQ29ycG9yYXRpb27CoGZvcsKgQXNzaWduZWTCoE5hbWVzwqBhbmTC
oE51bWJlcnMxGDAWBgNVBAMTD3d3dy5leGFtcGxlLm9yZzCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAIaFD7sO+cpf2fXgCjIsM9mqDgcpqC8IrXi9wga/
9y0rpqcnPVOmTMNLsid3INbBVEm4CNr5cKlh9rJJnWlX2vttJDRyLkfwBD+dsVvi
vGYxWTLmqX6/1LDUZPVrynv/cltemtg/1Aay88jcj2ZaRoRmqBgVeacIzgU8+zmJ
7236TnFSe7fkoKSclsBhPaQKcE3Djs1uszJs8sdECQTdoFX9I6UgeLKFXtg7rRf/
hcW5dI0zubhXbrW8aWXbCzySVZn0c7RkJMpnTCiZzNxnPXnHFpwr5quqqjVyN/aB
KkjoP04Zmr+eRqoyk/+lslq0sS8eaYSSHbC5ja/yMWyVhvMCAwEAAaOCA/IwggPu
MB8GA1UdIwQYMBaAFHSFgMBmx9833s+9KTeqAx2+7c0XMB0GA1UdDgQWBBRM/tAS
TS4hz2v68vK4TEkCHTGRijCBgQYDVR0RBHoweIIPd3d3LmV4YW1wbGUub3Jnggtl
```

vii) Compare the results and identify the protocol versions supported by the server.

1. If the server supports TLS 1.2, you will see "TLSv1.2" in the protocol line of the output.
2. If the server supports TLS 1.3, you will see "TLSv1.3" in the protocol line of the output.
3. If the server does not support a version, the command will either fail to connect or show an "unknown protocol" error.

Conclusion:
TLS 1.3 is the more modern and secure version, and it should be prioritized over TLS 1.2. Servers that support both versions are more flexible and secure, but using TLS 1.3 is recommended for better encryption and performance.

Problem 8 : Troubleshooting Security Problems

Step 1 : Collect info about connection debug cmd

Expected Deliverables -

i) Output Screenshot (Terminal should have SRN visible) for step 1

```
[11/21/24]seed@VM:~/.../PES2UG22CS556_LAB9$ openssl s_client -connect example.com:443 -showcerts -debug
CONNECTED(00000003)
write to 0x555bb1c35970 [0x555bb1c48fc0] (313 bytes => 313 (0x139))
0000 - 16 03 01 01 34 01 00 01-30 03 03 87 d2 51 fd 3a   ....4...0....Q.:
0010 - f7 9b a0 93 9d 1a 36 0b-34 c1 0a ea 38 e0 cf 4c   ......6.4...8..L
0020 - a6 91 a6 aa a9 06 78 93-96 7f 02 20 20 1b 29 b4   ......x....  .).
0030 - 66 5b 3a b3 11 73 9a cf-a4 5f 4c 8d fa db 4e b4   f[:..s..._L...N.
0040 - 8d 88 4e f5 bf 21 f9 93-a6 8f 60 50 00 3e 13 02   ..N..!....`P.>..
0050 - 13 03 13 01 c0 2c c0 30-00 9f cc a9 cc a8 cc aa   .....,.0........
0060 - c0 2b c0 2f 00 9e c0 24-c0 28 00 6b c0 23 c0 27   .+./...$.(.k.#.'
0070 - 00 67 c0 0a c0 14 00 39-c0 09 c0 13 00 33 00 9d   .g.....9.....3..
0080 - 00 9c 00 3d 00 3c 00 35-00 2f 00 ff 01 00 00 a9   ...=.<.5./......
0090 - 00 00 00 10 00 0e 00 00-0b 65 78 61 6d 70 6c 65   .........example
00a0 - 2e 63 6f 6d 00 0b 00 04-03 00 01 02 00 0a 00 0c   .com............
00b0 - 00 0a 00 1d 00 17 00 1e-00 19 00 18 00 23 00 00   .............#..
00c0 - 00 16 00 00 00 17 00 00-00 0d 00 30 00 2e 04 03   ...........0....
00d0 - 05 03 06 03 08 07 08 08-08 09 08 0a 08 0b 08 04   ................
00e0 - 08 05 08 06 04 01 05 01-06 01 03 03 02 03 03 01   ................
00f0 - 02 01 03 02 02 02 04 02-05 02 06 02 00 2b 00 09   .............+..
0100 - 08 03 04 03 03 03 02 03-01 00 2d 00 02 01 01 00   ..........-.....
0110 - 33 00 26 00 24 00 1d 00-20 ce 53 ef c8 bc 5c 9e   3.&.$... .S...\.
0120 - dd 7f cd 34 53 84 9e 4d-4a 69 42 ec 1c 73 72 6d   ...4S..MJiB..srm
0130 - 47 e6 73 01 61 d5 26 30-49                        G.s.a.&0I
read from 0x555bb1c35970 [0x555bb1c3fcb3] (5 bytes => 5 (0x5))
0000 - 16 03 03 00 58                                    ....X
read from 0x555bb1c35970 [0x555bb1c3fcb8] (88 bytes => 88 (0x58))
0000 - 02 00 00 54 03 03 cf 21-ad 74 e5 9a 61 11 be 1d   ...T...!.t..a...
0010 - 8c 02 1e 65 b8 91 c2 a2-11 16 7a bb 8c 5e 07 9e   ...e......z..^..
0020 - 09 e2 c8 a8 33 9c 20 20-1b 29 b4 66 5b 3a b3 11   ....3.  .).f[:..
0030 - 73 9a cf a4 5f 4c 8d fa-db 4e b4 8d 88 4e f5 bf   s..._L...N...N..
0040 - 21 f9 93 a6 8f 60 50 13-02 00 00 0c 00 2b 00 02   !....`P......+..
0050 - 03 04 00 33 00 02 00 17-                          ...3....
write to 0x555bb1c35970 [0x555bb1c48fc0] (352 bytes => 352 (0x160))
0000 - 14 03 03 00 01 01 16 03-03 01 55 01 00 01 51 03   ..........U...Q.
```

ii) Explain your observation

The OpenSSL s_client -connect hostname:port -showcerts -debug command helps troubleshoot TLS connection issues by providing detailed information:

1. Server Certificate: Displays the server's certificate, helping verify its validity, expiration, and trustworthiness (e.g., if the CA is trusted).

2. Cipher Suite: Shows the encryption algorithms used. Weak cipher suites can indicate vulnerabilities, so stronger suites like AES-256 should be used.
3. TLS Handshake: Logs details of the handshake, showing if there are version mismatches or unsupported cipher suites that could cause connection issues.
4. Connection Issues: If the connection fails, it will show errors like "connection refused" or "handshake failure," helping identify server unavailability or configuration issues.
5. Certificate Validation Failures: If the certificate has problems (expired, self-signed, or hostname mismatch), the output will indicate these issues, pointing to what needs fixing. In short, this command helps diagnose certificate, cipher suite, and handshake issues, ensuring a secure and functional TLS connection

Submission Format : SRN_Name_Lab8_AC.pdf