Exercise Cybersecurity : PKI
Q1 : From the two given openssl commands, what is the difference?
Ans : No difference at all, because I using MacOS

Q2 : What does the error (verify error) in the first command mean? Please explain.
Ans : No Error due to the command, because I using MacOS.

Q3 : Briefly explain what is stored in an X.509 certificate (i.e. data in each field).
Ans :
Version : 3 (0x2) : X.509 V3 certificate
Serial Number : Just a unique identifier assigned by issuer to this cert
Signature Algorithm : Algorithm for hashing here using ecdsa-with-SHA384
Issuer : The CA that signed the cert , C=country=US : America , O=organization=Let's Encrypt, CN=E6=Common Name (this CA's intermediate)
Validity : Valid time, Cert is valid only in this 90-day window.
Subject : CN=twitter.com : identity being certified, here domain twitter.com
Subject Public Key Info : Public key information said : Twitter's public key, 256-bit ECC on curve P-256
Extension : some properties
Signed Certificate Timestamps (SCTs) : Proofs from certificate transparency logs
Signature : Clients use this to verify the issuer's public key.

Q4 : From the information in exercise 3, is there an intermediate certificate? If yes, what purpose does it serve?
Ans : Yes, because issuer is E6 which is intermediate not root CA, it serves as web TLS, as a bridge between the root CA and leaf like twitter.

Q5 : Is there an intermediate CA, i.e. is there more than one organization involved in the certification? Say why you think so.
Ans : Yes, because we have intermediate CA and root CA for our certificate. We called it the certificate chain of trust. In twitter case which is leaf certificate we got intermediate certificate (E6 which serve for Web TLS) and Root certificate(ISRG Root X1 which is directly trusted by OS/browser root store), The reason why we have to have chain of trust is to keep root key offline for security, risk containment E6 got hack will not effect root CA.

Q6 : What is the role of ca-certificates.crt?
Ans : It's used to store trusted root CA certificates. Told my client like openssl which root CA certificate I trust because many tools don't query the OS trust store directly.

Q7 : Explore the ca-certificates.crt. How many certificates are in there? Give the command/method you have used to count.
Ans : 154 certificates, ca-certificates.crt is a PEM bundle -> just many —-BEGIN CERTIFICATE—- ~~ —-END CERTIFICATE—- block concatenated. so just using grep -c is enough
command `grep -c "BEGIN CERTIFICATE" ca-certificates.crt`

Q8 : Extract a root certificate from ca-certificates.crt. Use the openssl command to explore the details. Do you see any Issuer information? Please compare it to the details of twitter's certificate and the details of the intermediate certificate.
Ans : Issuer = Subject, while both E6 and twitter are different, so we know that this is a root CA certificate.
Twitter

```
ak1ra@Achiras-MacBook-Air Downloads % openssl x509 -in twitter_com.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            06:53:96:80:57:c6:c6:dc:f5:31:36:db:c5:c4:9f:bc:9e:a8
        Signature Algorithm: ecdsa-with-SHA384
        Issuer: C=US, O=Let's Encrypt, CN=E6
        Validity
            Not Before: Aug 19 20:42:10 2025 GMT
            Not After : Nov 17 20:42:09 2025 GMT
        Subject: CN=twitter.com
```

E6

```
ak1ra@Achiras-MacBook-Air Downloads % openssl x509 -in intermediate.cer -inform DER -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            b0:57:3e:91:73:97:27:70:db:b4:87:cb:3a:45:2b:38
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=Internet Security Research Group, CN=ISRG Root X1
        Validity
            Not Before: Mar 13 00:00:00 2024 GMT
            Not After : Mar 12 23:59:59 2027 GMT
        Subject: C=US, O=Let's Encrypt, CN=E6
```

Root CA

```
ak1ra@Achiras-MacBook-Air Downloads % openssl x509 -in root1.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 0 (0x0)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=Arizona, L=Scottsdale, O=GoDaddy.com, Inc., CN=Go Daddy Root Certificate Auth
ority - G2
        Validity
            Not Before: Sep  1 00:00:00 2009 GMT
            Not After : Dec 31 23:59:59 2037 GMT
        Subject: C=US, ST=Arizona, L=Scottsdale, O=GoDaddy.com, Inc., CN=Go Daddy Root Certificate Aut
hority - G2
```

Q9 : Yep, do it

Q10 : I will append my code&result at the end of this doc

Q11 :
class1 have class3 dont : Not really, class1 is easier/faster/cheaper
class3 have class1 dont : sign certificate for e-commerce, code-signing and high-assurance TLS where users expect verified organizational identity.

Q12 : Assuming that a Root CA in your root store is hacked and under the control of an attacker, and this is not noticed by anyone for months.
a. What further attacks can the attacker stage? Draw a possible attack setup.

If a **root CA** is hacked, the attacker has the private key. They can issue valid certificates for any website, company, or software. That lets them:

- Set up fake `bank.com` or `gmail.com` and run MITM attacks.
- Sign malware so OSes trust it as "legit software."
- Create new intermediates to issue many leaf certs quietly.

b. In the attack you have described above, can we rely on CRLs or OCSP for protection? Please explain

No.

- CRL/OCSP responses are signed by the CA. If the attacker has the root key, they can forge "good" revocation responses.

```
from OpenSSL import crypto
import pem
```

```
def verify(leaf_cert,inter_cert):
    with open(leaf_cert, 'r') as cert_file:
        cert = cert_file.read()

    with open(inter_cert, 'r') as int_cert_file:
        int_cert = int_cert_file.read()

    pems=pem.parse_file('./ca-certificates.crt');
    trusted_certs = []
    for mypem in pems:
        trusted_certs.append(str(mypem));

    trusted_certs.append(int_cert);

    verified = verify_chain_of_trust(cert, trusted_certs)

    if verified:
        print('Certificate verified')


def verify_chain_of_trust(cert_pem, trusted_cert_pems):

    certificate = crypto.load_certificate(crypto.FILETYPE_PEM, cert_pem)

    # Create and fill a X509Store with trusted certs
    store = crypto.X509Store()
    for trusted_cert_pem in trusted_cert_pems:
        trusted_cert = crypto.load_certificate(crypto.FILETYPE_PEM,
trusted_cert_pem)
        store.add_cert(trusted_cert)

    # Create a X590StoreContext with the cert and trusted certs
    # and verify the the chain of trust
    store_ctx = crypto.X509StoreContext(store, certificate)
    # Returns None if certificate can be validated
    result = store_ctx.verify_certificate()

    if result is None:
        return True
    else:
        return False
```

```
import sys, subprocess, re, os, tempfile
```

```
# Regular Expression
PEM_RE = re.compile(r"-----BEGIN CERTIFICATE-----.*?-----END CERTIFICATE-----", re.S)

# Get
def fetch_chain(domain: str) -> list[str]:
```

```
    r = subprocess.run(
        ["openssl","s_client","-connect",f"{domain}:443","-servername",domain,"-showcerts"],
        stdin=subprocess.DEVNULL, # No input, dont wait for it
        capture_output=True,
        text=True, # stdout/stderr as str
        timeout=20
    )
    blocks = PEM_RE.findall(r.stdout)
    if not blocks:
        raise RuntimeError(f"No PEMs found for {domain}. stderr: {r.stderr[:200]}")
    return blocks

def pick_intermediate_from_blocks(blocks):
    # blocks[0] = leaf
    for b in blocks[1:]:
        x = crypto.load_certificate(crypto.FILETYPE_PEM, b)

        is_ca = False
        for i in range(x.get_extension_count()):
            ext = x.get_extension(i)
            if ext.get_short_name() == b"basicConstraints":
                # e.g. "CA:TRUE, pathlen:0"
                if "CA:TRUE" in str(ext):
                    is_ca = True
                    break

        # intermediate = CA:TRUE and not self-signed
        if is_ca and x.get_subject().der() != x.get_issuer().der():
            return b

    # fallback: take second cert if present
    if len(blocks) >= 2:
        return blocks[1]
    raise RuntimeError("No intermediate certificate found")

def save_leaf_and_intermediate(domain: str) -> tuple[str,str]:
    blocks = fetch_chain(domain)
    leaf_pem = blocks[0]
    interm_pem = pick_intermediate_from_blocks(blocks)
    leaf_path = f"{domain}.leaf.pem"
    inter_path = f"{domain}.intermediate.pem"
    with open(leaf_path,"w") as f: f.write(leaf_pem)
    with open(inter_path,"w") as f: f.write(interm_pem)
    return leaf_path, inter_path
```

```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
domains = ["twitter.com","google.com","www.chula.ac.th","classdeedee.cloud.cp.eng.chula.ac.th"]

for d in domains:
        leaf, inter = save_leaf_and_intermediate(d)
```

```
        print(d)
        verify(leaf, inter)  # calls YOUR function, unchanged
```

```
twitter.com
Certificate verified
google.com
Certificate verified
www.chula.ac.th
Certificate verified
classdeedee.cloud.cp.eng.chula.ac.th
Certificate verified
```