```
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
```