Assignment 1: ABCs of Digital Certificates

Individual Assignment

Kamal Shrestha

CS21MTECH16001

Jan 30, 2022

PART-A **2**

PART-B **16**

7-zip **26**

PART-A

- Visit the website #N in this list of top-100 most visited websites globally where is #N is the last two digits in your roll number and download all the certificates in .CER format in the chain of trust from the Root Certificate, intermediate certificate(s), to the end-user (website) certificate at the leaf in the hierarchy.
- Compare the digital certificates in the chain in terms of various field values by filling this table.

Field Name	**Subject (CN) of the certificate holder (website)**	**Subject (CN) of the certificate holder (intermediate)**	**Subject (CN) of the certificate holder (root)**	**Remarks/observations**
Issuer	C=US, O=DigiCert Inc, CN=DigiCert TLS Hybrid ECC SHA384 2020 CA1	C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert Global Root CA	C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert Global Root CA	Only one Intermediate CA is involved in the chain of trust (validation) and both the CAs are based on DigiCert Inc.
Version No.	Version: 3 (0x2)	Version: 3 (0x2)	Version: 3 (0x2)	All with updated versions of digital certificates.
Signature Algo	ecdsa-with-SHA384	sha384WithRSAEncryption	sha1WithRSAEncryption	Varied Signature Algorithm between the CAs. The root is still using the SHA-1 Hashing Algorithm to get the

				message digest giving 160 bits. The end user certificate signing is also done using ECDSA and not RSA (which is the most common for certificate signing.)
Size of digest	384 bits	384 bits	160 bits	Depends on the hashing algorithm we use.
Signature Value	30:66:02:31:00:cd:aa:e8:16:18:0b:5e:de:24:bc:44:76:f3::e5:1e:a5:03:19:52:7e:fe:57:2c:0b:fe:e2:af:b4:67:3e::82:08:36:ce:01:60:30:8b:5b:a3:4e:50:27:1f:de:02:02::00:8e:fb:20:28:7e:b5:cf:df:1f:90:99:09:83:b0:77:70::e0:94:3f:9d:59:7d:ca:6c:21:69:2e:69:d2:cc:0f:e9:ac::c4:93:c2:9c:d6:83:96:e5:73:39:72:7d:9c	47:59:81:7f:d4:1b:1f:b0:71:f6:98:5d:18:ba:98:47:98:b0::76:2b:ea:ff:1a:8b:ac:26:b3:42:8d:31:e6:4a:e8:19:d0::da:14:e7:d7:14:92:a1:92:f2:a7:2e:2d:af:fb:1d:f6:fb::b0:8a:3f:fc:d8:16:0a:e9:b0:2e:b6:a5:0b:18:90:35:26::da:f6:a8:b7:32:fc:95:23:4b:c6:45:b9:c4:cf:e4:7c:ee::c9:f8:90:bd:72:e3:99:c3:1d:0b:05:7c:6a:97:6d:b2:ab::36:d8:c2:bc:2c:01:92:3f:04:a3:8b:75:11:c7:b9:29:bc::d0:86:ba:92:bc:26:f9:65:c8:37:cd:26:f6:86:13:0c:04::89:e5:78:b1:c1:4e:79:bc:76:a3:0b:51:e4:c5:d0:9e:6a::1a:2c:56:ae:06:36:27:a3:73:1c:08:7d:93:32:d0:c2:44:	cb:9c:37:aa:48:13:12:0a:fa:dd:44:9c:4f:52:b0:f4:df:ae::f5:79:79:08:a3:2418:fc:4b:2b:84:c0:2d:b9:d5:c7:fe::c1:1f:58:cb:b8:6d:9c:7a:74:e7:98:29:ab:11:b5:e3:70::a1:cd:4c:88:99:93:8c:91:70:e2:ab:0f:1c:be:93:a9:ff::d5:e4:07:60:d3:a3:bf:9d:5b:09:f1:d5:8e:e3:53:f4:8e:63:fa:3f:a7:db:b4:66:df:62:66:d6:d1:6e:41:8d:f2:2d:b5::77:4a:9f:9d:58:e2:2b:59:c0:40:23:ed:2d:28:82:45:3e::54:92:26:98:e0:80:48:a8:37:ef:f0:d6:79:60:16:de:ac::0e:cd:6e:ac:44:17:38:2f:49:da:e1:45:3e:2a:b9:36:53::3a:50:06:f7:2e:e8:c4:57:49:6c:61:21:18:d5:04:ad:78::2c:3a:80:6b:a7:eb:af:15:14:e9:d8:89:c1:b9:38:6c:e2::6c:8a:ff:64:b9:7	Even though the size of the diest for end user and intermediate CA is the same, 384 bits, the length/size of the signature value is different because of the difference in the signature algorithm used. End User Certificate (Wikipedia) is using ECDSA which provides the same level of security with much shorter key length compared to RSA, used

		:da:8d:f4:0e:7b:1d:28:03:2b:09:8a:76:ca:77:dc:87:7a::7b:52:26:55:a7:72:0f:9d:d2:88:4f:fe:b1:21:c5:1a:a1::39:f5:56:db:c2:84:c4:35:1f:70:da:bb:46:f0:86:bf:64::c4:3e:f7:9f:46:1b:9d:23:05:b9:7d:b3:4f:0f:a9:45:3a:e3:74:30:98	7:25:57:30:c0:1b:24:a3:e1:dc:e9:df::7c:b5:b4:24:08:05:30:ec:2d:bd:0b:bf:45:bf:50:b9:a9::eb:98:01:12:ad:c8:88:c6:98:34:5f:8d:0a:3c:c6:e9:d5:95:95:6d:de	by intermediate CA.
Validity period	Expires Nov 17 2022 (295 days) Thu, 17 Nov 2022 23:59:59 GMT	Expires Apr 13 2031 (3364 days) Sun, 13 Apr 2031 23:59:59 GMT	Expires Nov 10 2031 (3574 days) Mon, 10 Nov 2031 00:00:00 GMT	Validity for Root Certificates are very high
Is Subject field (CN), FQDN?	FQDN	CN	CN	More alternative names so many fully qualified domain names, if no alternative names then only Single domain with one common name.
Certificate type: DV, IV, OV, or EV? Tell also how you are able to determine the type!	Organization Validation Mentioned in the Certificate	Domain Validation Mentioned in the Certificate	-	Root CAs don't have any validations, Often the end-user web servers have a multi-domain certificate and have a lot of SANs. The type of the

				certificate was mentioned clearly in the certificate also but we could infer the validation type based on the use case of the certificate.
Subject Alternative Name (SAN/UCC), if any	*.wikipedia.org, wikimedia.org, mediawiki.org, wikibooks.org, wikidata.org, wikinews.org, wikiquote.org, wikisource.org, wikiversity.org, wikivoyage.org, wiktionary.org, wikimediafoundation.org, w.wiki, wmfusercontent.org, *.m.wikipedia.org, *.wikimedia.org, *.m.wikimedia.org, *.planet.wikimedia.org, *.mediawiki.org, *.m.mediawiki.org, *.wikibooks.org, *.m.wikibooks.org, *.wikidata.org, *.m.wikidata.org, *.wikinews.org, *.m.wikinews.org, *.wikiquote.org, *.m.wikiquote.org,	-	-	Intermediate and Root CAs dont have multiple access points in the form of multiple alternative names. Also, Wikipedia has a large number of SANs with wildcard entries possible.

	*.wikisource.org, *.m.wikisource.org, *.wikiversity.org, *.m.wikiversity.org, *.wikivoyage.org, *.m.wikivoyage.org, *.wiktionary.org, *.m.wiktionary.org, *.wikimediafoundation.org, *.wmfusercontent.org, wikipedia.org			
Certificate category: Single domain, wildcard or Multi-domain SAN/UCC cert?	Multi-Domain (SAN/UCC)	Single Domain/Explicit Name	Single Domain/Explicit Name	The end-User server needs multiple domains for multiple entries points to the server, so SANs are more there with multi-domain.
Public Key Info like key algo, key length, public exponent (e) in case of RSA	Elliptic Curve Public Key (id-ecPublicKey), 256 bit, 65537 (0x10001)	Elliptic Curve Public Key(id-ecPublicKey), 384 bit, 65537(0x10001)	rsaEncryption, 2048 bits, 65537 (0x10001)	Increasing number in the Public Keys bits size. Even though RSA or ECDSA has been used for certificate signing, ECPK is preferred for Asymmetric Key Cryptography.

Public key or modulus (n) in case of RSA	04:e8:50:2c:d0:d2:4e:a2:b1:92:aa:b6:73:0f:cf::b4:57:e5:c2:c0:7c:ae:6e:55:91:4a:a6:94:67::a5:f8:b0:3f:46:ac:23:52:b4:48:3b:64:64:fb::cd:e9:e4:fb:8f:10:a7:f4:e8:23:ba:95:29:6e: :ca:72:bb:83	04:c1:1b:c6:9a:5b:98:d9:a4:29:a0:e9:d4:04:b5::eb:a6:b2:6c:55:c0:ff:ed:98:c6:49:2f:06:27::cb:bf:70:c1:05:7a:c3:b1:9d:87:89:ba:ad:b4::17:c9:a8:b4:83:c8:b8:90:d1:cc:74:35:36:3c::72:b0:b5:d0:f7:22:69:c8:f1:80:c4:7b:40:8f::68:87:26:5c:39:89:f1:4d:91:4d:da:89:8b:e4:03:c3:43:e5:bf:2f:73	00:e2:3b:e1:11:72:de:a8:a4:d3:a3:57:aa:50:a2::0b:77:90:c9:a2:a5:ee:12:ce:96:5b:01:09:20::01:93:a7:4e:30:b7:53:f7:43:c4:69:00:57:9d::8d:22:dd:87:06:40:00:81:09:ce:ce:1b:83:bf::cd:3b:71:46:e2:d6:66:c7:05:b3:76:27:16:8f::9e:1e:95:7d:ee:b7:48:a3:08:da:d6:af:7a:0c::06:65:7f:4a:5d:1f:bc:17:f8:ab:be:ee:28:d7::7f:7a:78:99:59:85:68:6e:5c:23:32:4b:bf:4e::e8:5a:6d:e3:70:bf:77:10:bf:fc:01:f6:85:d9::44:10:58:32:a9:75:18:d5:d1:a2:be:47:e2:27::f4:9a:33:f8:49:08:60:8b:d4:5f:b4:3a:84:bf::aa:4a:4c:7d:3e:cf:4f:5f:6c:76:5e:a0:4b:37::9e:dc:22:e6:6d:ce:14:1a:8e:6a:cb:fe:cd:b3::64:17:c7:5b:29:9e:32:bf:f2:ee:fa:d3:0b:42::ab:b7:41:32:da:0c:d4:ef:f8:81:d5:bb:8d:58::b5:1b:e8:49:28:a2:70:da:31:04:dd:f7:b2:16::4c:0a:4e:07:a8:ed:4a:3d:5e:b5:7f:a3:90:c3:af:27	We have a varied public key size that is why we have varied key length. 2048 for Root CA that is why the key is having more characters.
Key usages; how do they vary in the chain?	Digital Signature, Server Authentication, Client Authentication	Digital Signature, Certificate Sign, CRL Sign, TLS Web Server, Authentication, TLS Web Client Authentication	Digital Signature, Certificate Sign, CRL Sign	The usages are marked as **critical** which means that the certificates should only be used for the specified purpose and nothing more.

				The end-user can't sign off on CRL with their certificate whereas the CAs can.
Basic constraints, how do they vary in the chain?	Critical	Critical Maximum number of intermediate CAs: 0	Critical Maximum number of intermediate CAs: unlimited	Intermediate CA here cannot certify another CA Whereas the root can certify an unlimited number of intermediate CAs.
Name constraints (if any), how are these useful?	CA: FALSE	CA:TRUE, pathlen:0	CA:TRUE, pathlen:1	End User doesn't have the name field CA: True as it is not a CA. Such will help us in the identification of the end-user subjects. The end-user certificate is missing pathlen, which might be because it is the final issued certificate. Also, the pathlen value is decreasing down the

				trust hierarchy
Size of the certificate	2982 Bytes	1504 Bytes	1360 Bytes	As the end-user certificate needs to incorporate the information of the entire hierarchy, the size of the tbsCertificate is bound to be larger in comparison to intermediate CAs, So the key size for end-user might be more than CAs.
Any other parameters that you found interesting?	CRL Distribution Point http://crl4.digicert.com/DigiCertTLSHybridECCSHA3842020CA1-1.crl Certificate Policies:2.23.140.1.2.2	CRL Distribution Points: URI:http://crl3.digicert.com/DigiCertGlobalRootCA.crl Policy:2.16.840.1.114412.2.1 Policy: 2.23.140.1.1 Policy: 2.23.140.1.2.1 Policy: 2.23.140.1.2.2 Policy: 2.23.140.1.2.3		No CRL Distribution Points or Policies for Root CA which suggests that Root CA does not maintain a CRL list.

Answer the following queries after filling out the above table:

1. **Which certificate type (DV/OV/IV/EV) is more trustable and expensive?**
   EV or Extended Validation Certificate Type is the most trustable and expensive because it involves a complete background check and is mostly relevant for e-commerce and banking companies as it involves a lot of financial transactions and sensitive information. Also, we know that the sequence of expensive certificates is: EV>OV>IV>DV which is completely based on the approach and extent to verification of the individual/organization per its domain/certificate claim.
   Looking at the table above, since Wikipedia is not an organization that works with sensitive information but an organization nevertheless, organizational validation is needed and seems to be enough.

2. **What is the role of the Subject Alternative Name (SAN) field in the X.509 certificate?**
   The role of the Subject Alternative Name (SAN) field in the X.509 certificate is to certify a large number of domains (200-300) (included in the Fully Qualified Domain Name(FQDN) list which are the variants of the Common Name) using a single certificate. Each of these domains in turn can be used to represent a particular access point within the organization.

   Looking at the table above, we can see that Wikipedia has a bunch (39 in total) of SANs (for example:*.wikipedia.org, wikimedia.org, mediawiki.org, wikibooks.org, wikidata.org, wikinews.org, wikiquote.org, wikisource.org, wikiversity.org, wikivoyage.org) that represent different access points (routes), different domains but are verified using a single certificate. Having so many alternative names allows the organization to redirect users to a single entry point in their website with different access points(subdomains/alternative names), a common example would be a typing error in the domain name.

3. **Why are key usages and basic constraints different for root, intermediate, and end certificates?**
   The constraints and the key usages of the digital or SSL certificates are based on the individual/organization/ authority that is either claiming or certifying the certificate (for which the certificate is going to be used).
   For example, end certificate users/subjects like en.wikipedia.org, youtube.com don't need to perform any sort of signature or webs server verifications. Such certificates of end-users are simply to advertise that the communication in their website is secure and certified to any connecting endpoints/subjects. Hence, a simple Client and Server Authentication using the Digital Certificate is enough. Whereas, any intermediate certificate authority or root certificate authority needs to perform the update of CRL entries, Sign off on the digital certificates. Also, the usages of the intermediate and root are pretty much the same as

intermediate CAs are designed to replicate and distribute the functionality of the Root.
The constraints serve a similar purpose. They define limits to which the certificates can be put into use. For an end-user, the constraint specifying whether it is a CA is marked False (Ca: False) but the intermediate subjects have it marked TRUE. The constraint with the maximum number of CAs is included in the certificates for CAs as it allows the subject to issue a certificate to an intermediate CA, which is why Root has the same field set to "*unlimited*". The pathlen also follows the same objective. pathlen=0 means that the certificate cant be used to certify another CA whereas "1" means issuing, renewing is allowed using the certificate.

Constraints extended to the SSL Certificate/ Digital Certificate can be found here: RFC.

4. **What is the difference between the Signature value and the Thumbprint of a digital certificate?**

First and foremost, the signature in the certificate is generated using the private key of the Certificate Authority (CAs) that binds the domain (corresponding to a server) of an end-user to their public key. The *tbsCertificate* part of the actual digital certificate is used to generate the digest using a hash function which in turn is encrypted with an encryption algorithm (with keys of the certificate authority) to generate the signature. The digital certificate for a domain is the mark of encrypted, secure, and trusted communications within that particular domain.
Unlike that, the thumbprint of a digital certificate is generated using the entire certificate as input to a hashing function like SHA-1, MD5, or SHA-256. These hash outputs or fingerprints can be used to identify/index/categorize the actual digital certificate within a certificate authority. So, when a CA wants to get a digital certificate from its certificate database/store, it can use this hash to get it.
A common wordplay would also show the message digest (generated from *tbsCertificate*) in the case of digital certificates, which can also be called a "Thumbprint" or "fingerprint" that identifies the *tbsCertificate*(the bulk of the digital certificate). This suggests that a thumbprint can be quoted as the result of a hash function (also known as a thumbprint algorithm).
Hence, a signature value identifies a domain with its public key generated by the CA using their private key whereas a thumbprint is simply a digest generated using a hashing algorithm.
Looking at the table, the signature value for each of the certificates is significantly larger than the thumbprint, which based on the concept is clear.

5. **Why do RSA key lengths increase over the years? Why is ECDSA being preferred over RSA nowadays?**

RSA algorithm works by taking two large prime numbers (p and q) and generating n ($n = p.q$), e, and d values based on them. The entire security and encryption of the RSA heavily rely on the fact that it is very difficult to factorize n to p and q.

n, e, and the ciphertext ($c = m^e mod\ n$) are public information so it is possible to get the actual message (m) using different techniques but it is believed that it will take trillions of years when tried with the brute force approach as factoring the number, n is a huge computational cost. Taking it one more step, since we are looking at only the prime numbers that make up the factors of n, it won't be a naive brute force algorithm, rather it will be significantly less than trying out all the combinations.
With increasing computational power in recent years, a huge number of calculations and alternative choices can be explored. It is now actually possible to break the RSA encryption algorithm by trying out a brute-force algorithm to get its factors. With 1024 bits in each prime number, we have a maximum of 2048 bits for n, which is less than 10^300 possible combinations. But if we increase the number of bits in prime numbers we would have even more combinations which will require more computational power (objective for it to require such a huge computational power that is not available and will not be available anytime soon).

So, the conclusion we drew from the above discussion is that for an increased level of security we need to increase the key length in RSA.
Now, we have ECDSA, which achieves the same level of security as RSA but with a lot fewer key lengths. For example, if RSA achieves level x security with 1024 bits key length, ECDSA achieves the same level, x with much less key length. (from 160-230). Alternatively, with the same key length for RSA and ECDSA, ECDSA achieves a higher security level (by security level, we mean the time taken for brute force algorithm to find the factors of n). Having a shorter key length requires less encryption time which means an increase in the performance of authentications. So, because of the decreased key lengths and increased performance ECDSA is starting to get preference over RSA.

6. **What are the pros and cons of pre-loading root and intermediate certificates in the root stores of browsers and OSes?**

CAs especially Root CAs play a vital role in the Public Key Infrastructure (PKI) during verification of the certificate and ensuring the connection established is a secure one. A Digital certificate for a subject will contain its public key that the other communicating subject will use to encrypt the messages. But before that, we need to get that public key from within the digital certificate.

A DC will be signed by CA using their private key and to retrieve the public key for the subject, the subject should use the public key of the CA that was preloaded to quickly decrypt the DC and get the public key. This helps in the efficient retrieval of the public keys from the DC. That is one less communication overhead to search the public key of the CA for the subject over the internet before establishing any communication.

One con of pre-loading the root and intermediate certificates would be to update the list every time a new root or new intermediate CAs are added to the list. If such regular update fails there might be a case when an intermediate CA whose certificate was revoked can still verify the subject's certificate and establish an unsecured connection or there might be a new authorized intermediate CA added by a root but due to lack of update not added in the root stores of browsers/OSes, because of it, even a secure connection can be flagged as insecure.

7. **Why are root CAs kept offline?**

Root CAs hold a very important function and the highest level of trust in the entire PKI hierarchy. One of the vital functions of a root CA is to issue certificates to intermediate CAs that in turn work to issue certificates to the subject. Because of the same Root, CAs are a prime target for the hackers, issuing false certificates to subjects, intermediaries will present a huge downside. For example, Imagine three intermediate CAs that issue digital certificates to 10 subjects. This makes certificates to 3X10 = 30 subjects under a single root CA. When that single root gets compromised, hackers can infect the entire 30.

Since this downside is too much and would take a significant amount of time and resources to revert back, root CAs are kept offline and only brought online when needed to deduct the exposure time for outside unauthorised interventions. Offline is purely a precaution for minimal risk management.

8. **List out names of OS/Browser/Company whose root stores are pre-populated with Root and Intermediate CA certificates of the website #N?**

DigiCert TLS Hybrid ECC SHA384 2020 CA1 and DigiCert Global Root CA are the intermediate and root CA respectively. Since they are both based on DigiCert Inc, the question can be formulated as Listing out names of OS/Browser/Company whose root stores are pre-populated with certificates from DigiCert Inc.

The following table contains the list of operating systems and browsers whose stores are pre-populated with certificates issued by DigiCert Inc.

Certificate Authority	**Operating System** *(pre-populated with the corresponding certificate)*	**Browser** *(pre-populated with the corresponding certificate)*
Root Certificate Authority DigiCert Global Root CA Intermediate Certificate Authority DigiCert TLS Hybrid ECC SHA384 2020 CA1	• Access • Android • BlackBerry OS • Brew • Chrome OS • Debian • HP-UX • iOS • Mac OS X • Meego • Palm OS • Palm WebOS • SUSE Linux • Ubuntu • Windows (all versions)	• AOL 5+ • Boxee • Camino 1.0+ • Chrome • Firefox 1.0+ • Grandstream • Internet Explorer 5+ • Konqueror 2.2.1+ • Maxthon • Microsoft Edge • Mozilla 7.0+ • Netscape 4.5+ • Opera 5+ • Safari • Sony Playstation

	• Windows CE • Windows Mobile • Windows Phone 7 • Windows Phone 8 • Windows Server (all versions)	• Nintendo Wii

Reference to the above list: [DigiCert SSL Compatibility]

PART-B

1. You have received the digital certificate of website #N over email. How do you verify whether the certificate is valid without using any online tools or browsers? Write a pseudo-code of your verifier function named myCertChecker( ) and explain how it works by picking the entire chain of trust of an end-user cert (of the website #N) in PART-A of this assignment.

   It is very important to verify the validity of the certificate, first to make sure that the right parties are communicating in between, and secondly to establish secure communication. On receiving a certificate, be it over an email or before the start of a conversation, there are a number of validity checks that the certificate must pass before communication can be started.
   The prerequisites for the function to validate the certificate are
   1. The actual digital certificate
   2. Current Date and Time

   The following functions explain the pseudocode (written in PYTHON) for the function myCertChecker() to verify the certificate:

```
def parse_certificate(certificate, **kwargs):  # kwargs = Keyword Arguments
  """Contains the entire certificate in JSON"""

def get_key_from_store(name_of_CA):
  """Returns Public key based on the name of Certificate Authority"""

def parse_key(key):
  """”Parses p,q,n,e,d based on the values of key"""

def query_CRL(URL, SN):
  """Returns the revocation status by looking at the CRL list based on the SN"""
```

```
def query_OCSP(URL, SN):
"""Returns the revocation status by querying the OCSP Server based on SN"""

def myCertChecker(digital_certificate, today):
    """Function to check the validity of the digital certificate
    1. Integrity Check: Verification of the Signature
    2. Validity of the Certificate
    3. Revocation Status Check """

    """Generate message digest using tbs"""

    # Get the tbsCertificate portion from the digital_certificate
    tbsCertificate = parse_certificate(digital_certificate, tbs=True)

    # Get the Signature Hash and Signing Algorithm
    hash, encryption = parse_certificate(digital_certificate, sign_algorithm=True)

    # Use the information of hash and tbs to get the message digest
    message_digest_from_tbs = hash(tbsCertificate)

    """Generate Message Digest from Signature Value"""

    # Get Signature Value from digital_certificate
    sign_value = parse_certificate(digital_certificate, sign=True)

    # Get the public key of the Certificate Authority from root stores
    pub_key = get_key_from_store(parse_certificate(digital_certificate, name_CA=True))
```

```
    # Parse the values of (n,e) from public key
    n, e = parse_key(pub_key)

    # Generate the message digest using the signature value from certificate
    message_digest_from_sign = (sign_value**e) % n

    """ Verify the integrity of the certificate: No Certificate Tampering"""
    if message_digest_from_tbs == message_digest_from_sign:
        valid_from, valid_to = parse_certificate(digital_certificate, validity=True)

        """ Checking the Validity of the certificate"""
        if (valid_from < today and valid_to >= today):
            serial_no = parse_certificate(digital_certificate, sn=True)

            # Querying CRL Distribution Points to check the revocation status
            crl_path = parse_certificate(digital_certificate, crl_path=True)
            CRL_revocation_status = query_CRL(crl_path, serial_no)

            # Querying OCSP Server to check the revocation status
            ocsp_server = parse_certificate(digital_certificate, ocsp_server=True)
            OCSP_revocation_status = query_OCSP(ocsp_server, serial_no)

            """ Checking the Revocation Status"""
            if CRL_revocation_status and OCSP_revocation_status:
                # Certificate is VALID
    return True
else:
    # Certificate has been revoked.
```

```
            return False
        else:
            # Certificate has expired.
            return False
    else:
        # Certificate has been compromised.
        return False

# MAIN FUNCTION
myCertChecker(_.wikipedia.org, "2022-01-30 06:42:55.581381")
```

The above pseudo-code checks for the following:

1. Verification of the Signature
2. Validity of the Certificate
3. Revocation Status of the Certificate

To implement the same checker to verify the certificate of en.wikipedia.org, we first download/export the digital certificate in CER or PEM format [X.509 Certificate in PEM format (application/pkix-cert+pem)]. We use a parser for the certificate to get each and every detail in the form of keywords and values so that we can get a particular value when needed.
The following figure shows the same:

Certificate Properties

Subject	C=US, ST=California, L=San Francisco, O=Wikimedia Foundation, Inc., CN=*.wikipedia.org
Issuer	C=US, O=DigiCert Inc, CN=DigiCert TLS Hybrid ECC SHA384 2020 CA1
Valid From	Oct. 19, 2021, midnight
Valid To	Nov. 17, 2022, 11:59 p.m.
Key Size	256 bits
Key Algorithm	id-ecPublicKey
Sig. Algorithm	ecdsa-with-SHA384
SANs	*.wikipedia.org, wikimedia.org, mediawiki.org, wikibooks.org, wikidata.org, wikinews.org, wikiquote.org, wikisource.org, wikiversity.org, wikivoyage.org, wiktionary.org, wikimediafoundation.org, w.wiki, wmfusercontent.org, *.m.wikipedia.org, *.wikimedia.org, *.m.wikimedia.org, *.planet.wikimedia.org, *.mediawiki.org, *.m.mediawiki.org, *.wikibooks.org, *.m.wikibooks.org, *.wikidata.org, *.m.wikidata.org, *.wikinews.org, *.m.wikinews.org, *.wikiquote.org, *.m.wikiquote.org, *.wikisource.org, *.m.wikisource.org, *.wikiversity.org, *.m.wikiversity.org, *.wikivoyage.org, *.m.wikivoyage.org, *.wiktionary.org, *.m.wiktionary.org, *.wikimediafoundation.org, *.wmfusercontent.org, wikipedia.org
Serial Number	02:7D:94:1B:29:2C:DB:2E:DA:F9:93:11:18:53:74:3E (331049704741152673306075719603555231 8)
SHA256 Fingerprint	57:0A:56:29:10:3E:74:95:96:33:6C:7B:A7:50:D3:85:ED:12:4D:B5:DD:BD:D6:46:15:46:50:0F:E8:16:53:F2
SHA1 Fingerprint	D6:06:82:CE:7D:BA:8A:1A:BD:8E:83:D2:38:D5:44:23:D9:D5:54:ED
MD5 Fingerprint	C1:4A:94:4A:36:D6:25:42:D7:18:12:E5:B6:BB:21:42
Validation Type	Organization Validation

We start by validating the **integrity** of the certificate. We take the tbsCertificate portion of the Digital Certificate that is in plain text format to generate the message digest using the signing (hashing and encrypting) algorithm: ecdsa-with-SHA384. Here we use the properties shown in the above figure for Wikipedia to generate the digest.

Similarly, we retrieve the message digest, from the signature value, that was signed by the CA, DigiCert TLS Hybrid ECC SHA384 2020 CA1, signed using their private key, and retrieved using the public key of CA (Digicert). Comparing these two digests should give us the idea of whether the certificate while being sent via Email has been tampered with or not. Here the signature value (384 bits) was the following:

30:66:02:31:00:cd:aa:e8:16:18:0b:5e:de:24:bc:44:76:f3:a3:e5:1e:a5:03:19:52:7e:fe:57:2c:0b:fe:e2:af:b4:67:3e:6a:82:08:36:ce:01:60:30:8b:5b:a3:4e:50:27:1f:de:02:02:31:00:8e:fb:20:28:7e:b5:cf:df:1f:90:99:09:83:b0:77:70:6f:e0:94:3f:9d:59:7d:ca:6c:21:69:2e:69:d2:cc:0f:e9:ac:53:c4:93:c2:9c:d6:83:96:e5:73:39:72:7d:9c

If not, we confirm that the certificate is not tampered with and we can move further with our checks.
We now check the **validity** of the certificate by looking at the valid till date (validity depends on how long has the certificate been issued). If validity fails, we can no longer guarantee secure communication. Here as we can see in the figure above, the validity of the certificate is till Nov. 17, 2022, 11:59 p.m. So, the certificate successfully passes the validity check.

After a validity check, we need to see if the certificate has been **revoked** or not? For that, we need to query the OCSP server for the revocation status or check the Certificate Revocation List (CRL) if the certificate has been included in the CRL list that marks the revocation of certification status. The thumbprint or a serial number of the certificate is used for the same purpose.

```
X509v3 CRL Distribution Points:

    Full Name:
      URI:http://crl3.digicert.com/DigiCertTLSHybridECCSHA3842020CA1-1.crl

    Full Name:
      URI:http://crl4.digicert.com/DigiCertTLSHybridECCSHA3842020CA1-1.crl

X509v3 Certificate Policies:
    Policy: 2.23.140.1.2.2
      CPS: http://www.digicert.com/CPS

Authority Information Access:
    OCSP - URI:http://ocsp.digicert.com
    CA Issuers - URI:http://cacerts.digicert.com/DigiCertTLSHybridECCSHA3842020CA1-1.crt
```

The above figure shows the URLs for the CRL list and the OCSP server. Here, the CRL List and the OCSP server are maintained by the Digicert, Intermediate CA. The pseudo-code looks for the certificate of Wikipedia using the serial no: 02:7D:94:1B:29:2C:DB:2E:DA:F9:93:11:18:53:74:3E (331049704741152673306075719603555231 8) in the CRL list and queries the OCSP server using the same. If the certificate is found in either of them, then the certificate is found to be revoked and the communication might not be secure. The process can be more streamlined using OCSP Stapling when websites like Wikipedia attach the OCSP Revocation Status with DIgital Certificate while trying to establish connection. Since, we are only referring to the certificate and its checks we do not worry about the stapling of OCSP status.

Now, if we have to look at the validity of the certificate for Digicert, which is an Intermediate CA here, we have to perform the same checks taking DigiCert TLS Hybrid ECC SHA384 2020 CA1 as subject and the root as DigiCert Global Root CA.

With all the three checks completed, we can say that the certificate for the subject, Wikipedia is VALID.

The full pseudo-code can be viewed here. *Pseudo-Code for Certificate Checker*

Apart from the above checks, we can also implement a permitted subtrees check based on the named constraint which indicates that if a subject contains the common name or any name in the subject's alternative name that is not based on the permitted subtrees, that particular certificate can be declared as invalid.

2. Consider the scenario in which evil Trudy has used the digital certificate of the website (Bob) named abc.com to launch her own web server with the domain name, **xyz.com**. Does your function myCertChecker( ) return valid or invalid for this when someone like Alice tries to access Trudy's website xyz.com from a browser like Chrome/Edge/Firefox?

   Currently, if Trudy duplicates the digital certificate of Bob (**abc.com**) and launches her new website **xyz.com**, the function myCertChecker() returns valid (saying that the certificate is authentic) which is because of the fact that it doesn't consider the verification of the common name or subject's alternative name. **An integral part of the verification of the digital certificate of any website/subject is the verification of the common name and the subject's alternative name**. What this verification does is it matches the digital certificate with the website that is being accessed. Failing this verification means that a third party has used the cloned/downloaded/stolen digital certificate for a cloned website from a different web server.
   Hence, In addition to the above pseudo-code, we need to add the following snippet to verify the sans and common name:

```
""" Checking the Revocation Status"""
if CRL_revocation_status and OCSP_revocation_status:
    cn, sans = parse_certificate(digital_certificate, cn_sans=True)

    """Verification of Common Name and Subject's ALternative Name"""
    if domain_name == cn or domain_name in sans:   # domain_name = Domain Name of the website being accessed (xyz.com)
        # Certificate is VALID
        return True
    else:
        # Certificate doesn't have valid CN/SANs.
        return False
else:
    # Certificate has been revoked.
    return False
```

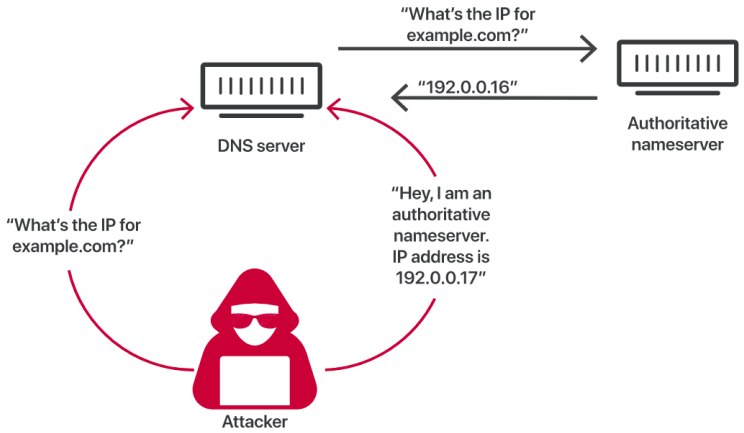
   The full pseudo-code can be viewed here. [*Pseudo-Code for Certificate Checker*]

3. Consider the scenario in which evil Trudy has used the digital certificate of Bob's website abc.com to launch her own web server with the domain name, **xyz.com**. When a web client (Alice) tries to connect with Bob's website abc.com by sending a DNS query, Trudy responds with her IP address by launching a MITM attack (What is DNS cache poisoning? | DNS spoofing | Cloudflare). Does your function myCertChecker( ) returns valid or invalid for this and what are the consequences? What kind of attacks can Trudy launch in this scenario?

   Our scenario is, Trudy has used the digital certificate of Bob's website abc.com to launch her own web server with the domain name, **xyz.com**. When a web client (Alice) tries to connect with Bob's website abc.com by sending a DNS query, Trudy responds with her IP address by launching a MITM attack.
   Here, Trudy can launch a MITM attack in two ways:

   1. She can intercept the DNS request from Alice, and as a response to that UDP request, she can send her own cloned website's IP address to Alice. Alice will still think that she is talking to Bob's website abc.com but she would actually be talking to Trudy's Website (xyz.com).

   2. Or Trudy can poison the DNS Cache that stores the IP of websites for quick access and instead of redirecting the abc.com query to Bob's web server, Trudy redirects the request to her own web server.



Reference: What is DNS cache poisoning? | DNS spoofing | Cloudflare

The function myCertCheckert() is designed to validate the authenticity and integrity of the digital certificate. Since Trudy has used the digital certificate of Bob's website abc.com with her own web server xyz.com, the certification test will return **INVALID** by failing the common name and SANs verification check.

But this verification check is only possible after the request for DNS of abc.com by Alice has been compromised by Trudy and redirected to xyz.com. (Case of DNS cache poisoning). So the steps involved are

1. Get the DNS of **abc.com**
2. Validate the Certificate
3. Establish a secure connection

The function myCertChecker() is used to validate the second step but Alice's request has already been compromised in the first STEP. Now, to make sure that the IP returned as a request to the DNS query (UDP) is the actual IP of **abc.com** not of a fake website like **xyz.com**, we need to make sure that DNS request/cache hasn't been compromised.

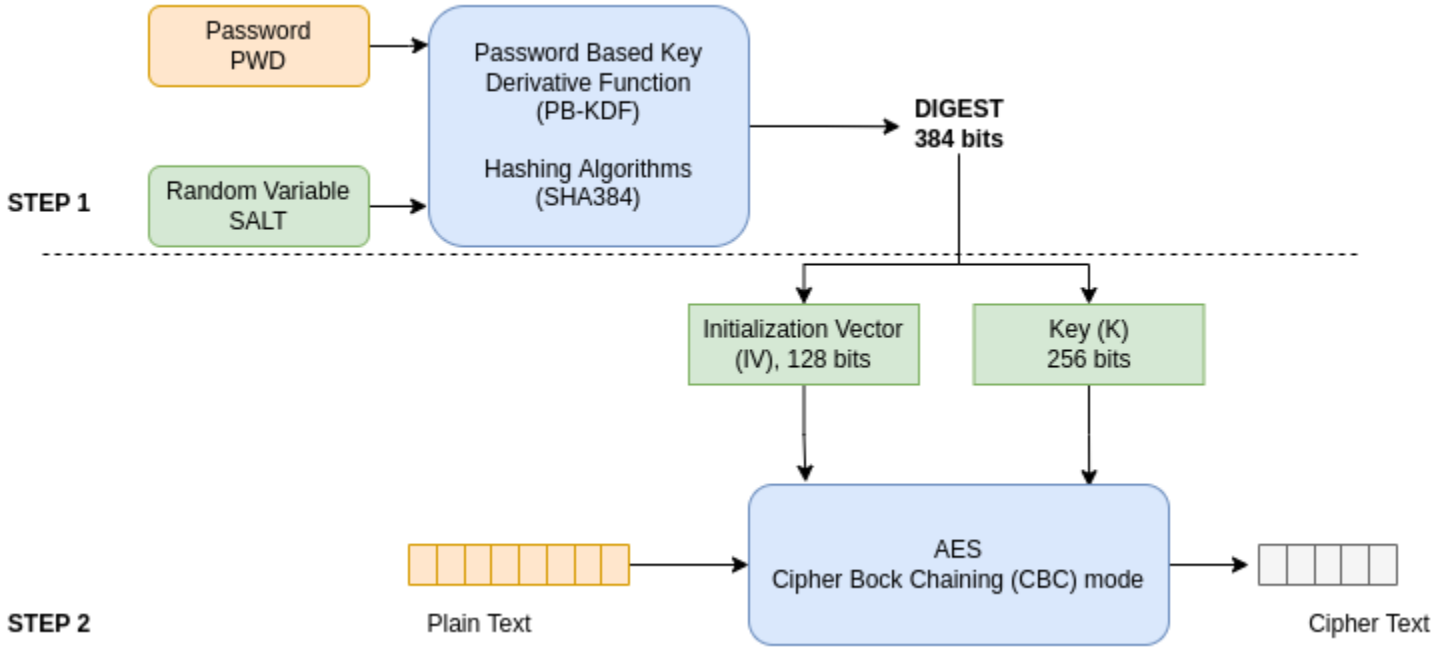
Let's say we don't perform common name or SANs validation for **xyz.com** (impersonating as **abc.com**), then Trudy can launch all sorts of MITM or WITM attacks like Impersonation, Denial of Service Attacks, Hijacking. Trudy won't be able to affect the integrity of the messages as Alice will be encrypting all the messages using Bob's Public key (requiring Bob's private key to decrypt), even though all other attacks are possible. As a consequence, the availability of the website **abc.com** will be compromised.

One way of verifying that the response hasn't been compromised and is sent from an actual authoritative name server for **abc.com**, we need DNSSEC (DNS Security) which makes the use of PKI to sign the IP response to every DNS query and validates the integrity of response and is not tampered with in between by anyone like Trudy.

7-zip

Briefly explain how 7-zip uses the password to encrypt compressed files using secure hash and symmetric algorithms. What role does the password length play in brute force attacks to decrypt the encrypted files?

Open-source software like 7-zip are examples of AES encryption with CBC mode in practice. Such software uses a function called Password-Based Key Derivative Function (PB-KDF) [Version 2] using two input parameters; PWD (password for encryption) and a random variable, SALT.
Such functions are nothing but secure hash algorithms that generate the digest using password and SALT as shown in STEP 1 of the figure below.



So, The output of such a function is also known as Digest which is now used as an input to symmetric algorithms like AES in CBC mode.
Now, we have the digest which is separated into the Initialization Vector(IV) and Key(K) which in turn are the inputs to the symmetric encryption algorithm, here we have AES in Cipher Block Chaining Mode (CBC) that converts the plain text or input file to an encrypted text/cipher text or encrypted output file.

Let's say a brute force attack is intended to be launched to decrypt the ciphertext generated using the above process. The attackers have the SALT values, the DIGEST, and the Cipher Text. All they need now is the PWD to break the encryption and reveal the actual plain text. If they have the PWD value they can generate the digest using SALT and PWD by feeding it to the Hashing algorithm. With the generated digest they can compare with the original digest to check if the password is right or not, giving them a feedback brute force loop to crack the passwords. The attackers need to try out the entire combination of PWDs to figure out the actual PWD, hence the brute force approach.

So, let us have L = Length of password, and characters of PWD are ASCII characters (128 in number).

Now, the possible combination of PWDs that can be formulated with given assumptions is $No\ of\ Characters^{Length\ of\ Passwords} = 128^{L}$.

So, increasing the length of the password, the number of possible combinations increases exponentially, as does the time to break it.

If L = 25, for the same conditions, we have $128^{25} = 4.7890486e + 52$ possible combinations of passwords. Saying it takes 2 seconds to try out one combination (Depends on the actual hashing algorithm we use to generate the digest), it will take around

$4.7890486e + 52 / 2 = 2.3945243e + 52\ sec = 7.5929867e + 44\ years$ which is very very difficult to execute with current computing resources.

Hence, the length of PWD passwords plays a vital role in determining whether the encrypted file can be decrypted by attackers or not using Brute Force Approach.

PLAGIARISM STATEMENT

I certify that this assignment/report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment/report has not previously been submitted for assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, I understand my responsibility to report honor violations by other students if I become aware of it.

Name: Kamal Shrestha
Date: Jan 30, 2022
Signature: K.S.