Asg 4: ABCs of Digital Certificates – CS23MTECH11026 - REPORT

PART-A: Comparison of Digital Certificates in the chain of trust of a popular website

- Website: https://spboss.blog/

Field Name	**Subject (CN) of certificate holder (website)** *spboss.blog*	**Subject (CN) of certificate holder (intermediate)** *GTS CA 1P5*	**Subject (CN) of certificate holder (root)** *GTS Root R1*	**Remarks/observations**
Issuer	GTS CA 1P5	GTS Root R1	GTS Root R1	Intermediate and Root CA are same
Version No.	Version 3	Version 3	Version 3	All certificates are of same version
Signature Algo	PKCS #1 SHA-256 With RSA Encryption	PKCS #1 SHA-256 With RSA Encryption	PKCS #1 SHA-384 With RSA Encryption	All certificate's encryption and signing Algorithm are same but the Root CA's uses SHA-384 instead of SHA-256
Size of digest that is signed to generate Cert Sign	256 bits	256 bits	384 bits	Size of Digest generate at root side is different then intermediate and end domain
Size of Cert Signature	2048 bits	4096 bits	4096 bits	By pasting the .crt file in this website https://lapo.it/asn1js/# and checking the signature file, which shows the size of cert signature
Validity period	11-01-2024 to 10-04-2024	13-08-2020 to 30-09-2027	22-06-2016 to 22-06-2036	Validity is increasing form end domain to Root CA
Is Subject field (CN),	Yes	No	No	Only End domain has the Fully

FQDN?				qualified domain Name as we can see in the subject alternative field
Certificate type: DV, IV, OV or EV? Tell also how you are able to determine the type!	DV	OV	OV	Just by looking at the subject field only the end entity contains the DNS name. Also, the Intermediate and Root are able to sign other DNS.
Subject Alternative Name(s) (SAN/UCC), if any	DNS Name: spboss.blog DNS Name: *.spboss.blog	NA	NA	There are only 2 DNS in the subject alternative field. Intermediate and Root don't have any
Certificate category: Single domain, wildcard, Multi-domain SAN/UCC cert?	Wildcard	Single Domain	Single Domain	To check this I have look for (*.) in the subject alternative field. But my intermediate and root don't have any alternative due to that they both are Single Domain.
Public Key Info like key algo, key length, public exponent (e) in case of RSA	Key algo - PKCS #1 RSA Encryption Key length - 2048 bits e - 65537	Key algo - PKCS #1 RSA Encryption Key length – 2048 bits e - 65537	Key algo - PKCS #1 RSA Encryption Key length – 4096 bits e - 65537	Except Root the intermediate and end entity have the same algo and key length. The Root has the largest key length which are mentioned below.
Public key or modulus (n) in case of RSA	B4 BC 33 11 11 A6 5D B3 4A CB 2C 49 2E FE FB 31 4A E7 9E AB 6F 7A 35 3D 11 8E C1 70 1E F4 18 06 B0 10 CC 4D F2 CB 46 11 EB 60 89 7A EF	B3 82 F0 24 8C BF 2D 87 AF B2 D9 A7 AE FA CA BA 44 D6 5B 3E FE B2 F7 B2 65 16 DC DE 10 E8 4F 2D 10 58 5A 28 86 87 A1 EE 6A B3 A0 D9	B6 11 02 8B 1E E3 A1 77 9B 3B DC BF 94 3E B7 95 A7 40 3C A1 FD 82 F9 7D 32 06 82 71 F6 F6 8C 7F FB E8 DB BC 6A 2E 97 97 A3 8C 4B F9 2B	NA

	D7 3B 9B 4B 0B 7E 45 70 9F 93 91 34 0A 2E 85 E2 0F 50 4A 33 F4 CD 4B B3 5F 9A E1 C5 D8 52 1E 72 92 93 20 D6 B9 FE 3F F3 B9 69 78 46 74 4C 17 01 46 6B 55 CD B2 C1 B7 1D AB 73 70 7C EB 12 A0 B6 7D 45 F0 CA 3F 97 C7 5F EB B2 E5 43 18 80 62 D0 7E 45 FD 38 FF 19 FB D0 C1 B0 59 15 15 A2 61 26 96 8E E7 DA B3 4F E3 0B 23 A9 9F 47 C5 50 77 30 BA 0F BB D3 F5 FF CB FE A8 D7 A7 F1 4F E2 C8 16 3A 66 DA D4 13 74 5A 5F 39 99 69 AD DD 43 3F AE 7D 75 F2 D5 7D 21 4F 45 1A 13 D9 D2 F4 9B 0D 64 88 64 23 11 AE 23 A6 AB 5D 12 C0 77 D1 16 E2 AF 6C A3 98 A3 C0 73 47 D8 BE FC A2 0B AC EA DB 8B 2F 03 60 40 AD 32 22 89 39 46 53 9F AC 61 91 E6 02 2D 87	75 4F 7F A1 52 01 8B 55 A8 4A 5B 06 48 C8 36 12 25 AB 89 F9 F2 23 5F 9D 60 65 F9 5C DA BE 3A E8 5C 6D 7D 9C D0 84 18 85 30 CD 4E 9B EC 3C D8 B3 E1 96 D4 F3 C5 0B 65 DB 8F B0 74 CB F6 1E F3 78 F1 AC 95 C5 DD 73 C3 31 88 81 AF 74 AA 6F FD 0C E3 05 95 F0 C5 10 4F 65 63 FA A0 AF C6 18 3D C5 A1 DF 97 79 D7 05 89 B3 30 B0 74 AE 3D 92 10 6B 8C 15 77 DD 0B 04 57 FB 81 03 DD EA 22 34 D5 E5 56 B2 F0 C4 8D 41 B1 C3 02 DB 62 EC 80 D0 FF 76 D4 86 E4 04 1A B6 B6 0C 2B 62 71 7D D9 AF D9 F1 5E FA C0 1E CA A0 19 5C 55 F0 80 D1 2A 0C 07 86 90 9F 35 E3 28 2B 5B EF 23 C8 A3 1D A4 A3 3A EE FE 83 DC 82 4C 25 B0 4D C5 51 AD 9E 9B D3 5B 84 C2 1A 5A E9	F6 B1 F9 CE 84 1D B1 F9 C5 97 DE EF B9 F2 A3 E9 BC 12 89 5E A7 AA 52 AB F8 23 27 CB A4 B1 9C 63 DB D7 99 7E F0 0A 5E EB 68 A6 F4 C6 5A 47 0D 4D 10 33 E3 4E B1 13 A3 C8 18 6C 4B EC FC 09 90 DF 9D 64 29 25 23 07 A1 B4 D2 3D 2E 60 E0 CF D2 09 87 BB CD 48 F0 4D C2 C2 7A 88 8A BB BA CF 59 19 D6 AF 8F B0 07 B0 9E 31 F1 82 C1 C0 DF 2E A6 6D 6C 19 0E B5 D8 7E 26 1A 45 03 3D B0 79 A4 94 28 AD 0F 7F 26 E5 A8 08 FE 96 E8 3C 68 94 53 EE 83 3A 88 2B 15 96 09 B2 E0 7A 8C 2E 75 D6 9C EB A7 56 64 8F 96 4F 68 AE 3D 97 C2 84 8F C0 BC 40 C0 0B 5C BD F6 87 B3 35 6C AC 18 50 7F 84 E0 4C CD 92 D3 20 E9 33 BC 52 99 AF 32 B5 29 B3 25 2A B4 48 F9 72 E1 CA 64 F7 E6 82 10 8D E8 9D C2 8A 88 FA 38 66 8A FC 63 F9 01 F9 78 FD 7B 5C 77 FA 76 87 FA EC DF B1 0E 79 95 57 B4	

			BD 26 EF D6 01 D1 EB 16 0A BB 8E 0B B5 C5 C5 8A 55 AB D3 AC EA 91 4B 29 CC 19 A4 32 25 4E 2A F1 65 44 D0 02 CE AA CE 49 B4 EA 9F 7C 83 B0 40 7B E7 43 AB A7 6C A3 8F 7D 89 81 FA 4C A5 FF D5 8E C3 CE 4B E0 B5 D8 B3 8E 45 CF 76 C0 ED 40 2B FD 53 0F B0 A7 D5 3B 0D B1 8A A2 03 DE 31 AD CC 77 EA 6F 7B 3E D6 DF 91 22 12 E6 BE FA D8 32 FC 10 63 14 51 72 DE 5D D6 16 93 BD 29 68 33 EF 3A 66 EC 07 8A 26 DF 13 D7 57 65 78 27 DE 5E 49 14 00 A2 00 7F 9A A8 21 B6 A9 B1 95 B0 A5 B9 0D 16 11 DA C7 6C 48 3C 40 E0 7E 0D 5A CD 56 3C D1 97 05 B9 CB 4B ED 39 4B 9C C4 3F D2 55 13 6E 24 B0 D6 71 FA F4 C1 BA CC ED 1B F5 FE 81 41 D8 00 98 3D 3A C8 AE 7A 98 37 18 05 95	
Key usages; how do they vary in the chain, mention in	Signing Key Encipherment	Signing Certificate Signer CRL Signer	Signing Certificate Signer CRL Signer	As the trusting functionality should be given to the provider of the certificate like

the remarks?				intermediate and root. But not given to end entity because of security reasons.
Basic constraints, how do they vary in the chain?	Is not a Certification Authority	Is a Certification Authority Maximum number of intermediate CAs: 0	Is a Certification Authority Maximum number of intermediate CAs: unlimited	This field determines that it is a Certificate Authority and how many valid certificate it can provide
Name constraints (if any), how are these useful?	NA	NA	NA	NA
Size of the certificate	1.91 KB	1.96 KB	1.89 KB	Even thought the Root CA uses 4096 bits for key length its size is less than the intermediate and end entity CAs.
URI of CRL	URI: http://crls.pki.goog/gts1p5/et-Lsmj_QTM.crl	URI: http://crl.pki.goog/gtsr1/gtsr1.crl	NA	This field specifies where is the list of revoked certificates and those are signed by private key.
URI of OCSP Responder	OCSP Responder: URI: http://ocsp.pki.goog/s/gts1p5/wlTHQTjR2cg	OCSP Responder: URI: http://ocsp.pki.goog/gtsr1	NA	Server that maintains the recent status of digital certificates.
Any other parameters that you found interesting?	NA	NA	NA	NA

Answer the following queries after filling out the above table:

1. Which certificate type (DV/OV/IV/EV) is more trustable, secure, and expensive?

- Extended Validation (EV) is most trustable and due to complex validation process, it is most secure and it is expensive because it is reserved for high security applications

2. What is the role of the Subject Alternative Name (SAN) field in X.509 certificates?

- The SAN field in X.509 certificates it determines that for which additional domain names this certificate is valid. Due to this functionality, it offers the flexibility and scalability of the certificates.

3. Why are key usages and basic constraints different for root, intermediate and end certificates? What could go wrong if all of them have the same values?

- Key usages and basic constraints give information about how the certificates should be used with Public Key Infrastructure (PKI). The root and intermediate certificates are allowed to sign other certificates by the end user don't have access to sign other certificates. Also, the intermediate certificate is able to sign limited number of certificates. If those values are kept same then the end entity may give unauthorized CA certificates which compromise the trust and security of PKI system.

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

- Signature is encrypted file of certificate content and Fingerprint / Thumbprint is hash of certificate content.

5. Why do RSA key lengths increase over the years? Why is ECDSA being preferred over RSA now-a-days?

- Over the years the advancements in computing power have increased which leads to faster decryption even for brute force approach. Hence, due to that reason the RSA key lengths are increasing to make it harder to decrypt. Now a days ECDSA is preferred over RSA because ECDSA provides smaller key length with same level of security as RSA. Also, ECDSA is computationally faster than RSA.

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

- **PROS: 1.)** The client is satisfied with the trust build by the browser and the CA's. **2.)** As the certificates are preloaded the verification becomes easier.
- **CONS: 1.)** Client has to depend on the browser updates to get new CAs in the list. **2.)** What if client don't want some of CA's which client may find untrustworthy.

7. Why are root CAs kept offline? How do they issue certificates to intermediate CAs?

- Root CAs is kept offline due to security reason. As intermediate certificates are signed by root and if the root CA is compromised then all the other confidential information is compromised. From intermediate CAs a Certificate Signing Request (CSR) is sent to root CA. Then root CA sent it back to the intermediate with its sign on it.

8. Why are root and intermediate certificates of new CAs cross-signed by the legacy CAs? Answer this by taking Let's Encrypt and its parent organization as root and intermediate CAs.

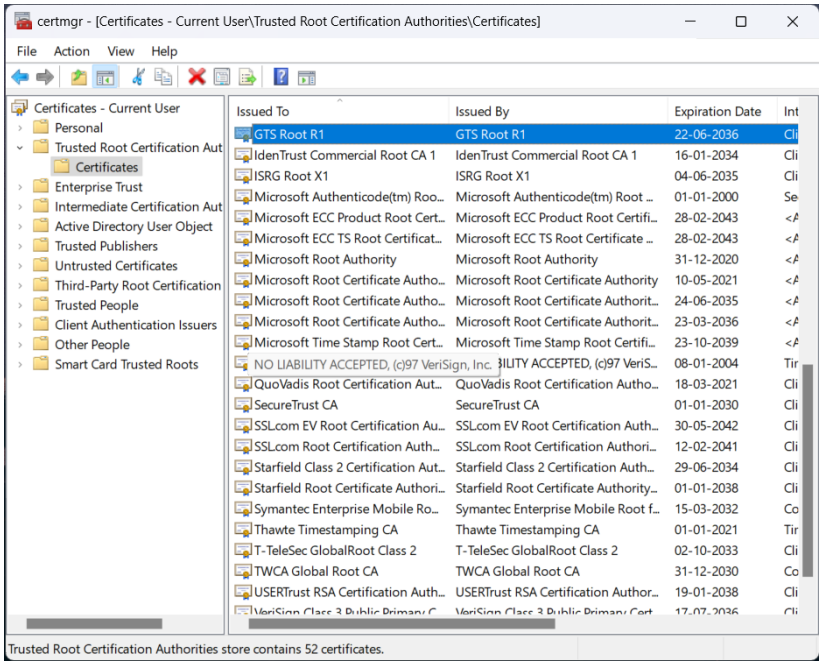
- To maintain the integrity the new CAs should be cross-signed by the legacy CAs because the legacy CAs assure the Trust. Then after getting signed by the legacy CAs we load the new CAs to our browser.

9. What challenge is posed to the certificate seekers (Alice) by Let's Encrypt CA before issuing wildcard certificates? How does Alice respond to it so that she passes it?

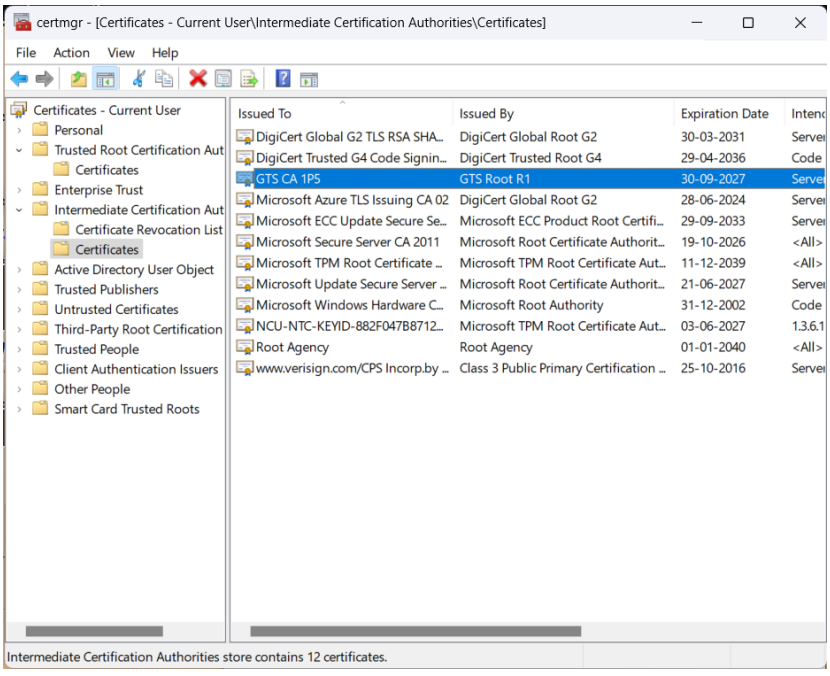
- DNS-01 challenge should be passed by certificate seekers (Alice) in order to get issued a wildcard from Let's Encrypt CA. If Alice is handling its Domain Name than Alice should be able to add the TXT record in order to pass this test. After the Let's encrypt CA will verify by checking that TXT record added by Alice.

10. List out names of OS/Browser/Company whose root stores were pre-populated with Root and Intermediate CA certificates of the website spboss.blog ?

Windows 11 – version 23H2

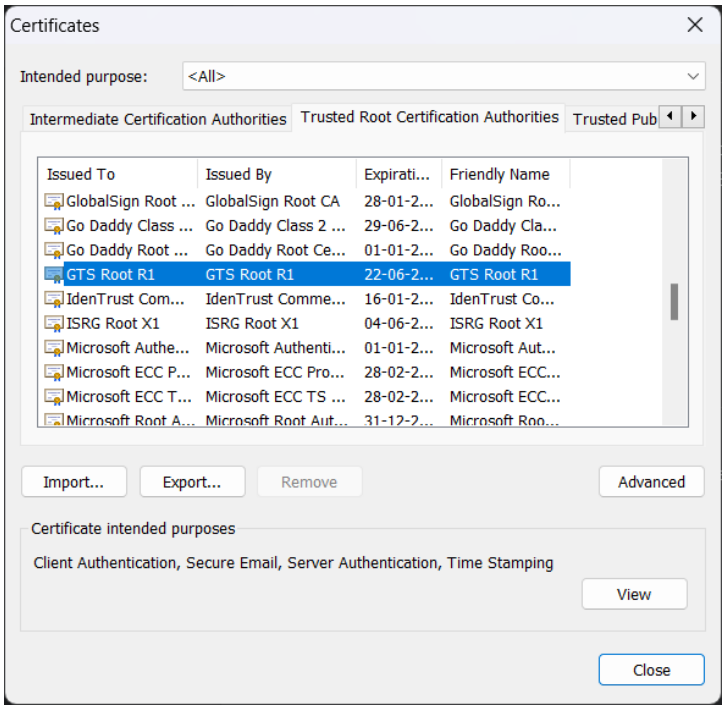


Root CA

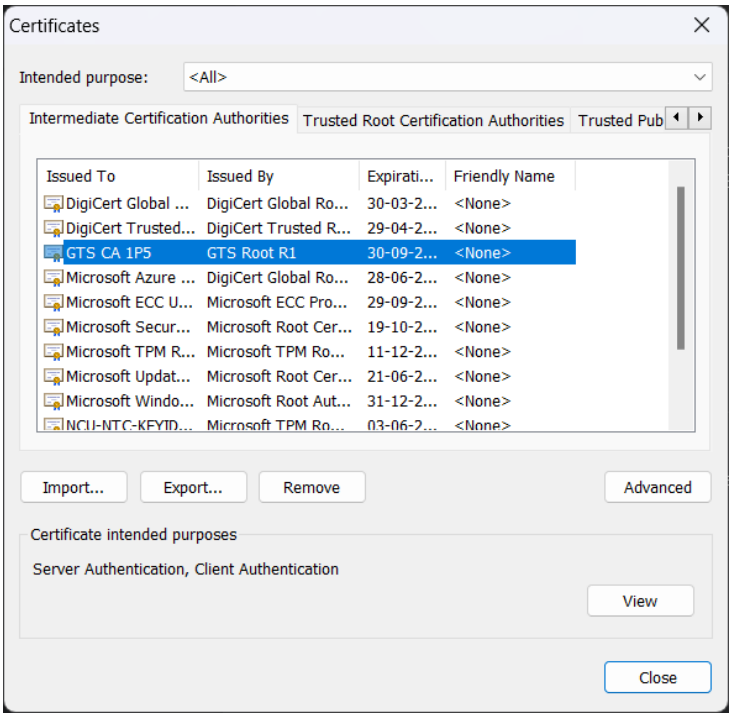


Intermediate CA

Google Chrome – Microsoft Edge – Brave Browser : These all browser have both Root CA and Intermediate CA.



Root CA



Intermediate CA

PART-B

1. A browser X has received the digital certificate of the website spboss.blog over a TLS connection. How does it verify whether the certificate is valid? Write a psuedo-code of browser X's verifier function named myCertVerifier( ) and explain how it works by picking the entire chain of trust of an end-user cert (of the website spboss.blog) in PART-A of this assignment.
   Pseudocode for myCertVerifier() [to check for correct hierarchy we can check issuer of the preceding provider name.]

```
- def myCertVerifier():
    # loop to check intermediate as well as root certificate
    for certi in range(len(chain of trust)):
    {
        # Check validity by Not before and Not after field
        if (today's date > Not after):
            print("Certificate Not Valid")
            return
        else:
            print("Certificate Valid")

        # Verifying digital signature
        msg = encrypted_digest_using_server_secret_key
        hash_got = decrypt_using_server_public(msg)
        if  ( Hash(original_long_msg) == hash_got ):
            print("Signature Verified")
        else:
            print("Signature not verified")
            return

        check for the key usage and if the certificate is revoked or not
        if above check valid then:
            print("Great. Checking further")
        else:
            print("Key usage field not valid")
            return

        check the local DNS query and CN / SAN field if there is domain name what we
        have asked for
        if above check verifies then:
            print("This certificate completely valid and proceed for the above hierarchy")
        else:
            print("Connection with trudy")
            return
    }
```

2. Consider the scenario in which evil Trudy has used the domain validated (DV) digital certificate of the website (Bob) named Bob.com to launch her own web server with the domain name, **xyz.com**. Does your function myCertVerify( ) returns valid or invalid for this when someone like Alice (browser) tries to access Trudy's website xyz.com?

- My function myCertVerify() will return invalid for this kind of attack. Because the Alice (browser) also check for the domain name asked for. Here Alice ask for bob.com by in the certificate it gets the xyz.com form Trudy.

3. Consider another scenario in which evil Trudy has used the digital certificate of Bob's website Bob.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 by sending a DNS query, Trudy responds with her IP address by DNS cache poisoning (What is DNS cache poisoning? | DNS spoofing | Cloudflare) Does your function myCertVerifier( ) returns valid or invalid for this and what are the consequences? What kind of attacks can Trudy launch in this scenario?

- myCertVerifier() returns invalid. Same reason for this scenario because even if the Trudy launches the DNS spoofing the DNS name sent by the Trudy will not match with what Alice have asked for, just by comparing the local DNS query and the response which it came from.
- Trudy can launch attack like Man in the middle (MITM), Session hijacking, Phishing etc.

7-zip:

- AES-256 algorithm for encryption. Cipher key length used is 256 bits. The hashing algorithm used is SHA-256. Default method used for compression is LZMA.
- For increasing the cost of exhaustive search for passwords 7-Zip uses big number of iterations to produce cipher key from text password.

References:

1. https://crt.sh/
2. https://ahrefs.com/blog/most-visited-websites/
3. http://lapo.it/asn1js/#
4. http://phpseclib.sourceforge.net/x509/decoder.php
5. https://www.ssl.com/article/dv-ov-and-ev-certificates/
6. https://www.ccadb.org/
7. DV, OV, IV, and EV Certificates - SSL.com
8. 7-Zip (7-zip.org)

PLAGIARISM STATEMENT <Include it in your report>

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 plagiarized 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: Bhargav Patel
Date: 06-02-2024
Signature: B.P