ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 11, Issue 2, February 2024

Self-Signed Certificate for Calibration Software

[1] Prathmesh Nitin Ghodake, [2] Nitin Gavankar, [3] Shailender Shekhawat

[1] [2] Computer Science and Engineering, Walchand College of Engineering, Sangli, Maharastra, India
[3] Calibration Company, India
Corresponding Author Email: [1] prathmeshghodke3545@gmail.com, [2] nitin.gavankar@walchandsangli.ac.in, [3] shailender.shekhawat@gmail.com

Abstract*— SSL/TLS and certificate authorities play a major role in the current web infrastructure. This article provides an overview of how to generate a self-signed SSL/TLS certificate. Digital certificates that are not produced by a reputable external certificate issuer and are self-signed by the entity whose identity they certify are known as self-signed certificates. Self-signed certificates are commonly used for testing purposes or for small internal networks where the price and complexity of acquiring a trusted certificate are unjustifiable. The article explains the steps involved in creating a self-signed certificate, including creating a private key, creating a CSR (certificate signing request), and finally creating the self-signed certificate using the private key and CSR. This article gives an overview of how to create a self-signed certificate.

Keywords: TLS/SSL, HTTP, HTTPS, OpenSSL, Certificate, man-in-the-middle attack.

I. INTRODUCTION

SSL allows clients to authenticate the identity of servers by verifying their X.509 [1] digital certificates and reject connections if the server's certificate is not issued by a trusted certificate authority (CA). A self-signed certificate is a digital certificate created and approved by the corresponding object it depicts, instead of a trusted third-party Certificate Authority (CA). Unlike certificates signed by a trusted CA, self-signed certificates are not validated by a trusted third party, which means they are not inherently trusted by default. However, they can be useful in a variety of situations, such as for testing or development environments, or for secure communication between devices that trust each other. Self-signed certificates contain a public key that can be used for encryption and authentication, as well as other information about the entity it represents, such as its name, contact information, and expiration date. They are often used as a cost-effective way to secure internal communication channels or for quick testing of secure communication protocols without the need for a trusted CA. However, because anyone can generate a self-signed certificate and deploy it to impersonate a trusted website or service, self-signed certificates do not provide the same level of security as certificates signed by a trusted third-party CA. As a result, it is important to only use self-signed certificates when the risk of impersonation is low and to always verify the certificate's authenticity before trusting it. Constructing a compelling security warning dialog has been a demanding mission for web browser vendors. There is Numerous research [5], [6], [7], [8] have displayed that numerous users in fact disregard warnings of SSL certificates. Notice that users who carelessly disregard certificate warnings would be undefended to the simplest SSL attacks. Many institutions are choosing to manage their own Certificate Authority instead of outsourcing to a third party [9].

The authentication system is the first line to prevent unauthorized users from gaining access to the system [10]. Authentication protocols are capable of simply authenticating the connecting party as well as authenticating itself to the connecting party [11]. Authentication can be based on different types of authentication tokens. Authentication tokens are normally categorized into categories: something known and something possessed [12].

II. BACKGROUND

A. TLS / SSL Protocol

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are cryptographic protocols utilized to establish secure communication over a network, ensuring information confidentiality and integrity. It includes HTTPS, IMAPS, SMTP, and XMPP [2]. The primary purpose of these protocols is to facilitate the establishment of a secure connection between the web server and a client, such as a web browser. TLS is the next in line to SSL, and it provides better security and improved performance. Secure Sockets Layer (SSL) lets clients validate the individuality of servers by verifying their X.509 [3]. SSL/TLS protocols work by encrypting the data transmitted between the server and the client, which makes it difficult for an attacker to intercept and read the data. The utilization of digital certificates within the protocols serves the purpose of verifying both the server and client identities, effectively mitigating the risk of man-in-the-middle attacks. SSL/TLS uses a combination of symmetric and asymmetric encryption to secure the communication between the server and the client. Symmetric encryption is used to encrypt the data, and asymmetric encryption is used to establish a secure connection and exchange the keys used for symmetric encryption. SSL/TLS protocols have become the standard for secure communication on the internet and are used by numerous

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 11, Issue 2, February 2024

applications, such as web browsers, email clients, and messaging apps.

TLS/SSL protocols provide several key benefits for securing internet communication, including:

1. Confidentiality: TLS/SSL protocols encrypt data transmitted between clients and servers, protecting it from interception and eavesdropping.
2. Integrity: TLS/SSL protocols ensure that data transmitted between clients and servers is not tampered with or modified in transit, protecting it from unauthorized changes.

Authentication: TLS/SSL protocols use digital certificates to verify the identity of the server and protect against man-in-the-middle attacks.

B. Certificate Authorities

Certificate Authorities (CAs) are organizations that issue digital certificates and are the entities responsible for their issuance used to verify the identity of a website or other entity over the Internet. CAs plays a crucial role in authenticating the identity of the requester of a certificate and ensuring that the certificate contains accurate and reliable information about the certified entity. In the context of self-signed certificates, the term "CA" is used to refer to the entity that issues the certificate to itself, rather than to an external third-party organization. In a self-signed certificate, the entity that generates the certificate acts as its own CA. This means that the entity creates its public key infrastructure and uses it to issue digital certificates that are signed using its private key. Because self-signed certificates are authenticated by a reliable intermediary CA, they are often not trusted by default by web browsers and other software that rely on digital certificates to verify the identity of a website or other entity. Two types of certificates are domain validation (DV) and extended validation (EV). Many Certificate Authorities allow verification by putting a file in a certain URL or by modifying the DNS record [4]. An EV needs more steps as the certificate also confirms that the expectation is arriving at the meant domain but additionally the individuality of the organization or person attached to it. This procedure may incorporate phone calls, individuality documentation, and even in-person interviews correspondingly, EV certificates are harder to inherit and commonly more costly. While self-signed certificates can be a useful way for small organizations or individuals to secure their web servers or other internet-facing services, they are generally not recommended for use on public-facing websites or other services that are accessed by a wide range of users. This is because users may not trust the self-signed certificate and may be discouraged from using the service or may be vulnerable to man-in-the-middle attacks if they do choose to trust the certificate.

C. Root Store

Root stores are repositories of trusted root certificates that are pre-installed on operating systems, web browsers, and other software. Root stores are maintained by various organizations, such as Microsoft, Mozilla, and Google. When a client (such as a web browser) encounters a self-signed certificate, it checks to see if the certificate's issuer (the entity that signed the certificate) is in its root store. If the issuer is not in the root store, the client will display a warning or error message indicating that the certificate is not trusted. To avoid this problem, some organizations create their root certificates and distribute them to clients in advance. This is often done in enterprise settings where the clients are controlled by a central IT department. By distributing their root certificate, the organization can issue self-signed certificates that are trusted by their clients.

III. METHODOLOGY

A. Generate a PEM file

A PEM file is a format for storing SSL/TLS certificates and private keys. It is a Base64-encoded format that contains either a public or a private key. A PEM file is a format for storing SSL/TLS certificates and private keys. It is a Base64-encoded format that contains either a public or a private key.

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -
out cert.pem -days 365
```

Figure 1: Generate PEM file

Here's what each part of the command does:

- **Openssl**: This is the command for the OpenSSL toolkit.
- **req**: This will command generate a new CSR and a new private key.
- **-x509**: This option tells OpenSSL to create a self-signed certificate instead of a CSR.
- **-newkey rsa:4096**: This option tells OpenSSL to generate a new 4096-bit RSA key pair.
- **-keyout key.pem**: This determines the filename to which the private key should be saved (in PEM format).
- **-out cert.pem**: This determines the filename to which the certificate should be saved (in PEM format).
- **-days 365**: This determines the number of days the certificate is valid.

Once the command is executed, OpenSSL will generate a new RSA key pair, create a self-signed certificate with the specified validity period, and save the private key to key.pem and the certificate to cert. pem.

B. Generate Certificate Signing Request (CSR)

A self-signed certificate is a type of digital certificate that carries its own signature, instead of being signed by a trusted third-party entity. When creating a self-signed certificate, you need to provide certain parameters that define the

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 11, Issue 2, February 2024

properties of the certificate. Here are the most common parameters that are included in a self-signed certificate:

1. Common Name (CN): This is the primary identifier for the certificate, and the server's complete domain name, with all necessary qualifications (FQDN), where the certificate will be utilized on.
2. Organization (O): This field specifies the legal name of the organization that is responsible for the certificate.
3. Organizational Unit (OU): This field specifies the department or division within the organization that is responsible for the certificate.
4. Country (C): This field specifies the two-letter ISO code for the country where the organization is located.
5. State or Province (ST): This field specifies the name of the state or province where the organization is located.
6. Locality or City (L): This field specifies the full name of the city where the organization is located.
7. Email Address: This field specifies the email address of the person responsible for the certificate.
8. Public Key: This cryptographic key serves the purpose of decrypting data and validate signatures. The public key is typically included in the certificate.
9. Private Key: This cryptographic key serves the purpose of decrypting data and signing messages. The private key is typically kept confidential and is not included in the certificate.

Validity Period: This determines the time that the certificate is valid, typically measured in years.

```
openssl req -new -key <your_private_key>.key -out <your_csr>.csr
```

Figure 2: Generate CSR file

C. Generate RootCA

A root CA (Certificate Authority) is an entity with a reliable reputation is responsible for issuing digital certificates that are used to verify the identity of websites and other digital assets.

Create a root certificate using the private key with the following command:

```
openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 365 -out rootCA.crt
```

Figure 3: Generate your own Root CA

D. Sign CSR with Root CA

We sign CSR with our own Root CA that we created above

```
openssl x509 -req -in <your_csr>.csr -CA <your_root_ca_certificate>.crt -CAkey <your_root_ca_private_key>.key -CAcreateserial -out <your_signed_certificate>.crt -days <number_of_days>
```

Figure 4: Sign CSR with our Root CA

<your_csr>.csr with the filename of your CSR.

<your_root_ca_certificate>.crt with the filename of your self-signed root CA certificate,

<your_root_ca_private_key>.key with the filename of your self-signed root CA private key,

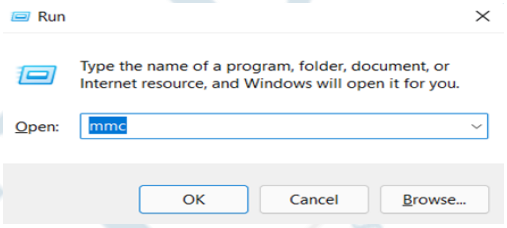
<your_signed_certificate>.crt with the filename you want to give your signed certificate,

<number_of_days> with the number of days that the signed certificate should be valid.

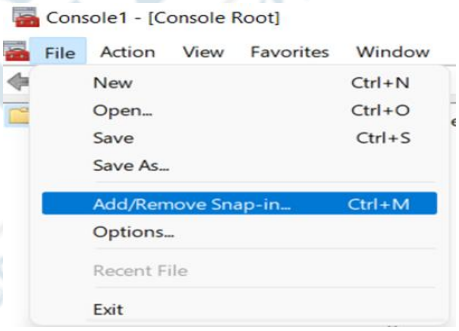
As a result, the CA-signed certificate will be in the .crt file.

E. Install Certificate in Computer

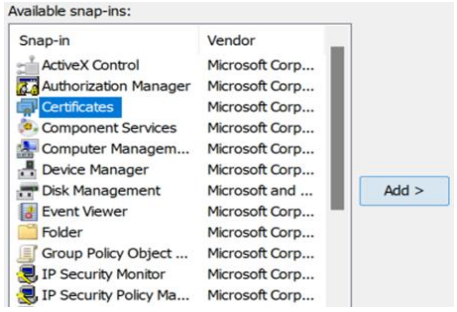
1. Press Win + R → type "MMC" → Click OK.



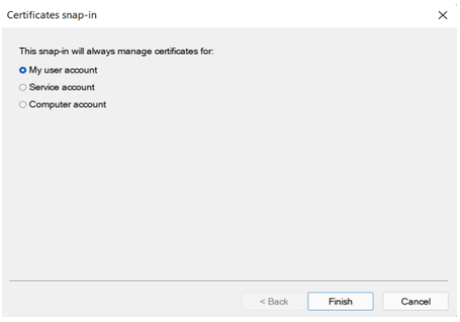
2. Go to 'File' and click on Add/Remove Snap-in.



3. Click on 'Certificate' -> 'Add'.



4. Select 'User Account' and click on 'Finish.

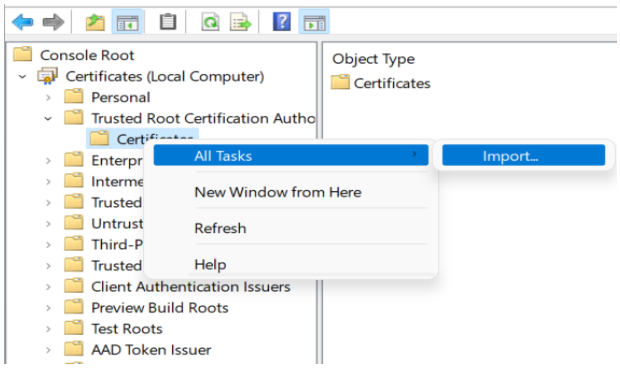


ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 11, Issue 2, February 2024

5. Now, expand ‘Certificates’ -> ‘Trusted Root Certification Authority’ -> ‘Certificate’, Right click on it -> ‘All task’ -> ‘Import’.



IV. FLOW DIAGRAM

Figure 5: Flow diagram to generate a self-signed certificate.

V. BENEFITS OF SELF-SIGNED CERTIFICATE

Cost: It’s available at free of cost. There is no need to purchase a certificate from a trusted third-party CA.

Control: With a self-signed certificate, the organization has complete control over creating and distributing the certificate. The organization can create and revoke certificates as per their requirements.

Flexibility: Self-signed certificates are more flexible than other types of certificates. They can be used when a trusted third-party CA is unavailable, such as in testing or development environments.

VI. LIMITATION OF SELF-SIGNED CERTIFICATE

Trust: The most significant limitation of self-signed certificates is the lack of trust. Because the certificate lacks the endorsement of a trusted third-party certification authority, there is no way to establish the identity of the organization.

Compatibility: Another significant disadvantage of self-signed certificates is that they may not be compatible with some older browsers and operating systems. This can lead to issues when connecting to secure websites.

REFERENCES

[1] Cooper, David, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and William Polk. *Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile*. No. rfc5280. 2008.

[2] Dierks, Tim, and Eric Rescorla. *The transport layer security (TLS) protocol version 1.2*. No. rfc5246. 2008.

[3] CA-Browser-Forum-BR-1.4.1.pdf (2016, Sept. 7) Baseline Requirements Certificate Policy for the Issuance and Management of Publicly Trusted Certificates. [Online] Available: https://cabforum.org/baseline-requirements-documents/

[4] CA/Browser Forum EVV1_6_0..pdf (2016, Jul. 1) Guidelines for the Issuance and Management of Extended Validation Certificates.[Online] Available: https://cabforum.org/ extended-validation/

[5] Dhamija, Stuart E. Schechter Rachna, Andy Ozment, and Ian Fischer. "The Emperor’s New Security Indicators." (2007).

[6] Sunshine, Joshua, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. "Crying wolf: An empirical study of ssl warning effectiveness." In *USENIX security symposium*, pp. 399-416. 2009.

[7] Akhawe, Devdatta, and Adrienne Porter Felt. "Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness." In *USENIX security symposium*, vol. 13, pp. 257-272. 2013

[8] Felt, Adrienne Porter, Robert W. Reeder, Hazim Almuhimedi, and Sunny Consolvo. "Experimenting at scale with google chrome's SSL warning." In *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 2667-2670. 2014.

[9] Patriciu, Victor V., and Aurel Serb. *Design aspects in a public key infrastructure for network applications security*. MILITARY TECHNICAL ACADEMY BUCHAREST (ROMANIA) COMPUTER ENGINEERING DEPT, 2000.

[10] Wang, Haiyuan. "Security Architecture for the TEAMDEC System." PhD diss., Virginia Tech, 2000.
[11] Duncan, Richard. "An overview of different authentication methods and protocols." *SANS Institute* (2001).
[12] Klemetti, Kari. "Authentication in Extranets." PhD diss., HELSINKI UNIVERSITY OF TECHNOLOGY, 2001.
[13] Web reference: https://aboutssl.org/installing-self-signed-ca-certificate-in-window.
[14] Web reference: https://www.baeldung.com/openssl-self-signed-cert.