SSL/TLS Certificates and OpenSSL Cheat sheet

saumgm.com/p/openssl-cheat-sheet.html

SauMGM J. Thomas Stokkeland Information Technology Blog

Started this for myself, just to remember the syntax of various things - over the years others have asked me to add details and explanations as well. (Work in progress, migrating items from my old google-site)

NOTE: This article was written some years ago and only considered RSA keys, one of these days i will update it to talk about ECDSA keys as well, as I am guessing RSA keys will eventually be obsolete.

What is an SSL (TLS) cert?

There is a fair amount of misunderstanding out there about what an "SSL Certificate" does and how to use them. Technically speaking no-one uses SSL anymore, the network encryption protocol in use is now TLS, regardless, it is usually referred to as a "SSL Certificate". Technically, it is a x509 certificate with a RSA key - therefore we work with x509 and rsa when dealing with openssl and web server certs.
A Certificate uses a "Common name" field (and other things) to identify what it is for, e,g. www.saumgm.com - this way when your browser receives a certificate, it compares that to the name of the website it got it from, that along with the expiration date and an electronic signature makes it "valid".

Certificate Authority (CA)

A "publicly valid" certificate has been signed by an authorized Certificate Authority (CA), the CA has done at least some basic validation to make sure you own what you claimed as your common name, some types of certificates do extended validation (ev-certs), all financials institutions and many others use these certs.

Most people don't need to, but if you are looking to set up a local CA, look at this article.

Public + Private Key encryption

x509 public certificates are issued to match the RSA Key. The public certificate is used to encrypt the data sent, the private key is used to decrypt the data - therefore you give the public certificate to anyone you want to send you data, you never ever share the private key. This is what the web server does, it sends the public certificate to anyone connecting, the client uses it to encrypt, the webserver uses the private key to decrypt the data. What varies in a connection is the protocol and cipher used, e,g, TLS v1.1 and AES-128 (and hundreds of variations/combinations).
It can be used for many more things than just web servers, anything basically - most

services utilize encryption in some way.
Never forget: Keep your private key safe

The Process of getting a (valid) cert

Here are the typical steps of getting a simple web site cert - using openssl

- Create your private rsa key (2048 bit)
  - openssl genrsa -des3 -out mydomain.key 2048
  - The password is to protect the key, if you need one that is unprotected skip the -des3
  - Make sure you keep this file safe
- Create a Certificate Signing Request (CSR)
  - openssl req -new -key mydomain.key -out mydomain.csr
  - Follow the prompts, remember the information you use will be visible in your public cert
  - Common name should be your web site domain, e.g. www.saumgm.com
  - Multi-domain certs are a bit more complicated, another section for that on openssl on this page someday - for now, be aware that some CA's may "update" your cert to include your bare domain and with the www. - but just in case they don't, choose the one you want your cert to match.
  - If buying a wild card cert, common name should be: *.domain.net
    Most CA's will here update your cert to include both bare domain, and the wildcard.
  - No need to input email or what is sometimes referred to as Extra information
- Submit the CSR to your CA (Order cert, follow their process, submit or paste the CSR in)
- The CA will ask you to verify your domain, typically by email
- Once verified, the CA will send you your signed certificate

Process of creating a self signed cert

Very similar to the process above for a valid cert - you just sign it yourself instead of paying the CA

- Create your private rsa key (2048 bit)
  openssl rsa -des3 -out mydomain.key 2048
- Create a Certificate Signing Request (CSR)
  openssl req -new -key mydomain.key -out mydomain.csr
- Sign the certificate yourself
  openssl x509 -req -days 720 -in mydomain.csr -signkey mydomain.key -out mydomain.crt

Valid Signed vs Self-signed certificates

There are two main reasons we use certificates:

1. Encrypt the data in motion - protect against anyone sniffing/reading it
2. Validate that the certificate provider is who it says it is - protect against someone using the wrong certificate to trick you

#1 is automatic - if you have a cert+key you can encrypt and decrypt.
#2 is done by certificate signatures - crypto-technology that an authority uses their private key to sign the certificate for the end user. The certificate always includes a Common Name (CN) which is used to verify that the usage matches the signed certificate, typically a hostname/domain name.
There is a limited amount of commonly recognized certificate authorities (CA) out there, one typically has to purchase a certificate, validate that you own the resource (such as a domain name), before they issue a signed certificate.
For systems where globally recognized signatures are not required, one can use a self signed or internal organization (Self-CA) signed certificate - for any public website you will probably need a valid one.

The chain / intermediaries

Due to the explosion in use of certificates for http servers, there are a lot of authorities out there now, so many of them are considered 2nd tier authorities, meaning they are not part of the few root-ones that all clients/browsers know to recognize, so this means that for a 2nd tier authority to be allowed to issue certificates, this authority must be verified and signed by a first tier authority - this creates the need for a certificate chain. Where root-companyA has signed for second-companyB, companyB can now sign, but must provide an intermediate certificate to validate their signature.
This is very common, and often the headache of a certificate manager, some modern browsers will have all the new second tier ones already, while some older devices are not, so it is a common issue that an iPhone or Android may not accept a certificate while Internet Explorer or Chrome on a PC does. The process of checking a certificate chain can be cumbersome, if it is a web server and it is accessible from the internet, i recommend https://www.sslshopper.com/ssl-checker.html

TLS

TLS can be initiated off the bat, like when you connect it expects you to "negotiate crypto" without any preliminaries, this is common with HTTP over TLS (Commonly referred to as HTTPS). In other protocols it may start in plain text, and exchange some information, then initiate TLS before any real data starts to flow, typically with a STARTTLS command - this is common in many SMTP implementations.

There is a lot more to this protocol than described here, google for SSLv3 Hello, TLS 1.0/1.1/1.2 issues with improper implementations, it is a complex topic.

HTTP specifics

Since HTTP is the most common one used with valid signed, globally recognized certificates - here are the specifics on how it works.

- Certificate common name must match domain name, e.g. www.saumgm.com or a wildcard *.mydomain.org
- Certificate must be valid, signed by something the client/browser accepts as an authority
- Certificate must not be expired
- A browser/http client will connect to the web server, conceptually something like this:
    - Connect TCP (typically on port 443)
    - Negotiate and set up TLS, this process includes
        - ciphers/certificates etc
        - compare common name (or multi name) to connected hostname
        - compare signature (and chain) to valid trusted authorities
        - compare cert expiration date to current date
        - Many browsers/systems will also check public revocation lists
    - Start encrypted HTTP session
- HTTP 1.0/1.1
    - This is the most commonly used protocol
    - Once the session is done, it is torn down, certain pieces are cached so that a new session in short time will be quicker to come up/less work
- HTTP 2.0
    - This is not widely supported/deployed yet (Late 2018) - the most common deployments are with F5's or other separation devices - only supported in modern browsers
    - A session can be kept open for multiple requests without the need to set up, it also supports parallel requests speeding up load times and reducing web server overhead.
- As mentioned before - the most common issue with cert deployment is the certificate chain, not all clients/browser have the same root trusts, so it is very important to always include a full chain in any web server setup.

Certificate Formats

PEM

Most common, and my personal preference of the format i want them given to me is PEM, this is (usually/mostly) a Base64 encoded certificate in a text file, meaning the files are "human readable", can copy/paste etc.
File may contain certificate(s) and key
Typical file extensions: .cer .crt .cert .pem .key
Many web servers can take these as is, including Apache httpd and F5/LTM

NOTE: The Default PEM format for a private key make with OpenSSL is PKCS1, some systems, such as Azure Keyvaults require use of PKCS8 for the private key format, see conversion below on how to convert. An RSA key in PKCS1 starts with "-----BEGIN RSA PRIVATE KEY-----" while a PKCS8 encoded RSA key starts with "-----BEGIN PRIVATE KEY-----". Everything in this article assumes PKCS1 unless specified otherwise.

DER

This is a simple binary format, so these are not "human readable", only certificate tools (like OpenSSL) can read them.
File may contain certificate(s) and key
Typical file extensions: .der .cer
Many Java platforms will use these (E.g. Citrix NetScaler)

P7B/PKCS7

I don't see these very often anymore - it is a Base64 encoded format. - mostly on Windows systems as far as I can recall.
It can only contain Certificate(s), no key
Typical file extension: .p7b .p7c

P12/PFX/PKCS12

This is a binary format, and the file is encrypt able in itself.
File may contain certificate(s) and key
Typical file extensions: .p12 .pfx
Used by a lot of different systems, Windows systems preferred format (And I find that when i need to create Java Keystores, this is the best format to import from).

Java Keystore

This is not a standard format per say - but it is what many Java tools (Like some Tomcat Libraries) use to store certificates, the keystore itself will have a password, as well as the private key, and a lot of software (Tomcat included I believe) require those passwords to be the same. You create these keystores using the JRE keytool command - I prefer to use openssl for all creation, convert to P12, then import to Keystore.

Modulus comparison of key and cert

The output of these commands should match

- openssl x509 -noout -modulus -in certfile | openssl md5
- openssl rsa -noout -modulus -in keyfile | openssl md5

If it does not, the cert does not belong to the key.

Remove Encryption from a key

```
openssl rsa -in keyfile -out unprotected_keyfile
```

Certificate and Key Conversions

I always recommend converting from whatever you have to PEM, then make whatever you need from that format. In a PEM you can easily spot if you have they key or not.

NOTE: The Default PEM format for a private key make with OpenSSL is PKCS1, some systems, such as Azure Keyvaults require use of PKCS8 for the private key format.

- DER to PEM
  - openssl x509 -inform der -in cert.der -out cert.pem
  - openssl rsa -inform der -in key.der -out key.pem
- P7 to PEM
  openssl pkcs7 -print_certs -in certs.p7b -out certs.pem
- P12 to PEM
  - openssl pkcs12 -in bundle.p12 -out cert.pem -clcerts -nokeys
  - openssl pkcs12 -in bundle.p12 -out key.pem -nocerts -nodes
- From PEM to DER:
  - openssl x509 -outform der -in cert.pem -out cert.der
  - openssl rsa -outform der -in key.pem -out key.der
- From PEM to P12:
  openssl pkcs12 -export -in certs.pem -inkey key.pem -out bundle.p12
- From Unencrypted PEM (PKCS1) Key to Unencrypted PEM PKCS8 Key Format
  openssl pkcs8 -topk8 -inform PEM -outform PEM -nocrypt -in key.pem -out key.pem-pkcs8

An example of what you may need in a p12 is your cert, your key, and your chain certs from CA, copy your cert and chain certs to the certs.pem, then create your p12.
For some systems, like Citrix NetScaler, you probably need the CA chain/cert in one DER, and your own in a separate one. F5's are easy peasy, just copy/paste the PEM.

Get Information

- PEM cert get details:
  openssl x509 -in cert.pem -text -noout
- PEM key check
  openssl rsa -in key.pem -check -noout
- CSR get details
  openssl req -text -noout -in myrequest.csr
- P12 get details
  openssl pkcs12 -info -in mybundle.p12
  (This outputs the certs in PEM, you can add -noout, but that also supresses the issue details)

Linux/Unix - Create a local Certificate Authority (CA)

I get these questions all the time - people know i have some runtime with certificates and such - one question is "Can't i just issue my own certs?" - and the answer of course is yes - but I always make sure to add that it won't be any use on a public web site since no-one will trust it. So setting up your own CA is not "generally useful", it is more if you need some specific things, like issuing certificates with a single signing source for client logins

or similar. Most business will have a couple of Windows Domain controllers, if you need to sign certs for a limited set of users, what you should do is make sure some system in your windows domain runs Certificate Services, then issue certs from there, make sure any non-domain-members has a trust for that CA. If you actually do need to set up you own CA, here is one way to do it Procedure to set up your own local CA The common name for the CA cert must NOT be the same as a domain name or anything e

Cisco UCS Mini - Add Extender Chassis

If you happen to own a UCS Mini Setup, a 5108 Chassis with two Fi 6324 or similar, and you are looking for documentation on how to add another 5108 Chassis with fabric extenders (2204XP in my case), then Cisco really does not have much out there, nor is there a lot of googlable information either (Everything you find is related to standalone Fabric Interconnects and "standard" UCS). Even after calling TAC, it took a while to get something, and what they told us was not even accurate. So here is how we did it, and it worked, came up without any interruption to current chassis, network, or running profiles. Equipment Of course we used our Cisco vendor to spec the equipment, but just for reference here is the list of what we had and what we added: Original Setup 5108 Chassis Fi 6324 (Qty 2) Ports 1-2 for Fibre Channel, and 3-4 for Ethernet (MMF) Connected to a stack of switches and pair of FC switches/SAN Running UCS version 4.0.1 (Fairly recently upgraded as of M

Active Directory Account Lockout - Narrowing Down the source

If you are in a all-windows shop where everything is nice and neat, everybody has a proper domain membership and all authentication is SSO or Windows Integrated, then you probably do not have much of a problem with repeated account lockouts. On the other hand, if you are in a mixed environment, lots of :Linux, Mac, and unmanaged Wintendo, then you probably run into some users that manage to Lock themselves out frequently - typically for several days in a row after the account password had been changed. Reasons can be plenty fold - typically saved credentials somewhere, like a git client, sql-server client, email client, rdp-manager, smbfs-automount, or anything that tries a bunch of logins when you start it up, or keeps trying in the background. As a sysadmin, you don't have time to narrow it down for the end user - but they will be adamant it is not their fault, so you probably need to prove that "Yes it is" - so I use powershell to grab 4740 events from Domain Con