Crypto Programming with OpenSSL

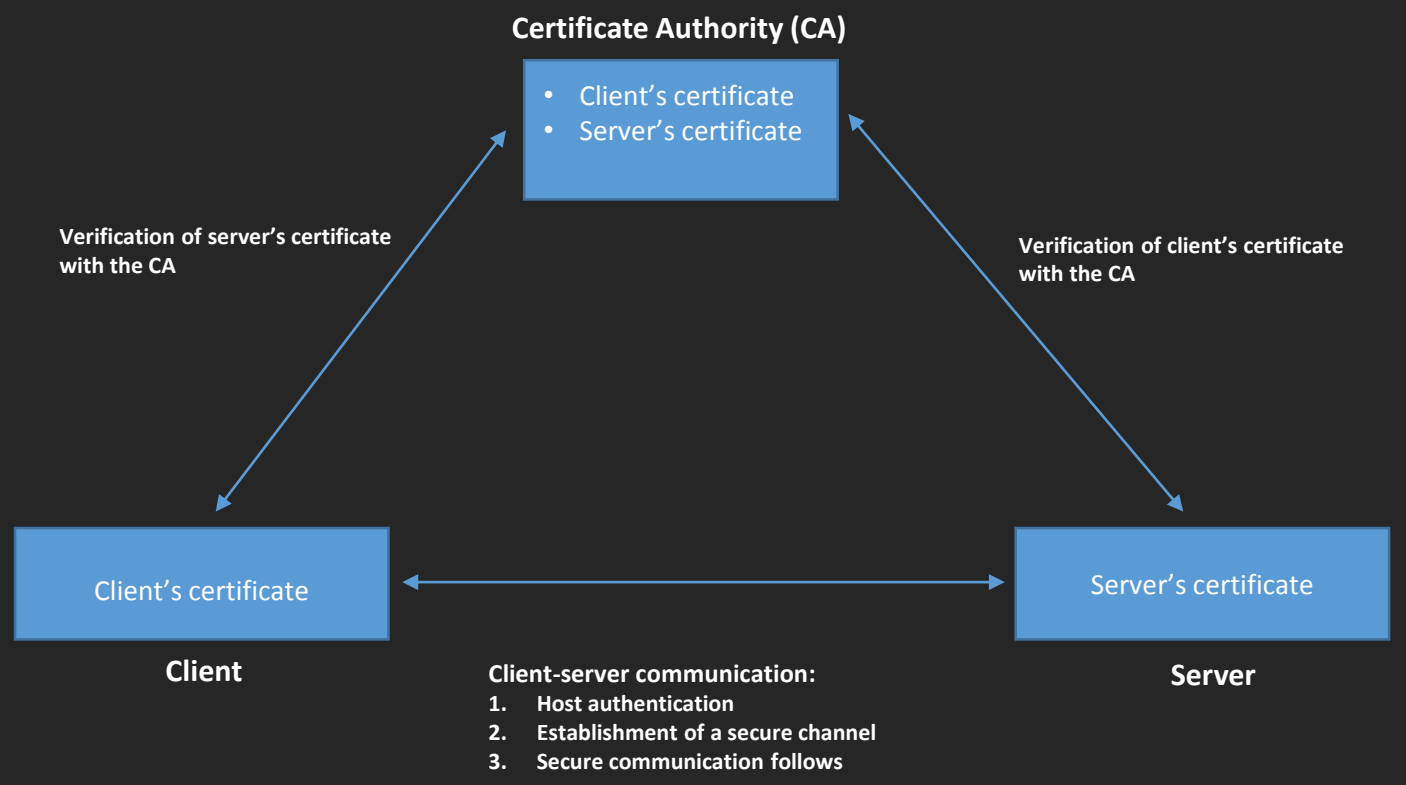
(Creating Certificates)

Secure Host-to-Host Communication

- Secure communication between hosts is necessary to prevent successful MITM attacks
- The communication channel is encrypted via the SSL (Secure Sockets Layer) protocol
- Communicating hosts (e.g., clients and servers) maintain public identities (or digital certificates) that are verified with a trusted third party
- The authenticity of communicating hosts is established through the third-party-dependent verification process
- The third-party verifies that a digital certificate presented by a host actually belongs to the host. (This is an important basis for establishing trust in the communication process)
- OpenSSL: open-source implementation of SSL. Crypto library available for programmatically implementing secure host-to-host communication

Host Authentication: The Handshake Process

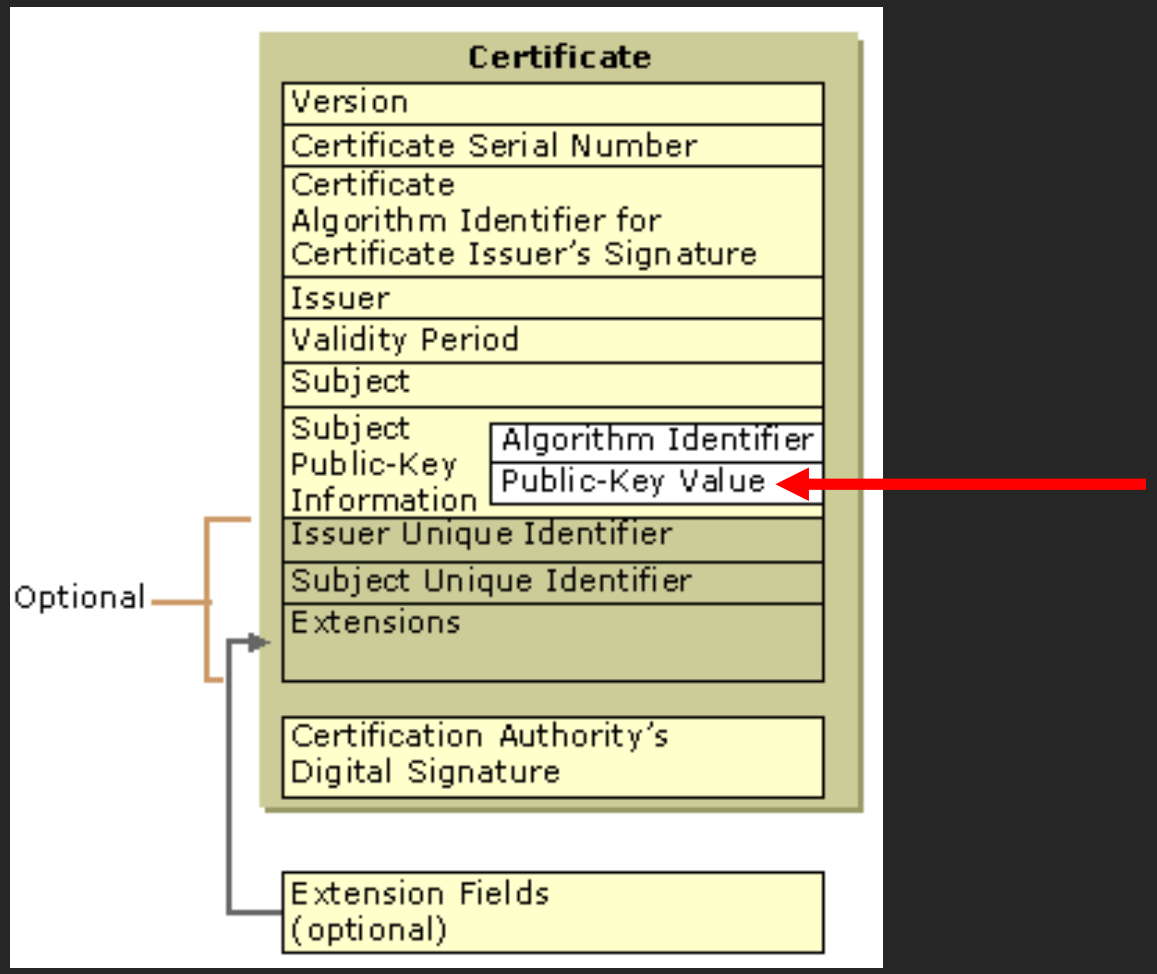
- Server → Client (Server's certificate is verified and Server is authenticated)
- Client → Server (Client's certificate is verified and Client is authenticated)
- Client-Server secure communication channel is established



More specifically…

- Client requests resource from server
- Server sends Client its certificate
- Client verifies Server's authenticity on the basis of the received certificate
- In turn, Client sends its certificate to Server
- Server verifies Client's authenticity on the basis of the received certificate
- Once mutual authentication of Client and Server has successfully taken place, a secure communication channel is established

The (Digital) Certificate



Creating Digital Certificates

- Entails generating private/public key pairs
- A trusted third party (CA) signs a host's certificate with its (CA's) public key, for future verification and host authentication purposes
- You can act as your own CA, where you self-sign the certificate, or you can forward the certificate to a well-known commercial CA (e.g., Go Daddy, VeriSign, DigiCert, etc.) for signing
- For our purposes, we self-sign our certificates
- Our certificate-creating tool: OpenSSL

Let’s create and sign our certificate, acting as our own certificate authority

Creating Certificates with OpenSSL: CA setup

- First, set up the certificate authority (CA):
  - Create a working environment in your home directory (or any directory of choice) with the command:
    cd && mkdir -p ourCA/signedcerts && mkdir ourCA/private && cd ourCA
    (we indicate "ourCA" as the name of our working directory; you can pick any name of your choice.)
  - Three initially empty sub-directories are created: ~/ourCA, ~/ourCA/private, and ~/ourCA/signedcerts
  - Directory contents: ourCA: holds the certificates database, requests, CA certificate, generated certificates, and keys
    ourCA/private: holds the CA's private key
    ourCA/signedcerts: holds copies of signed certificates (with serial naming)

- Next, create a certificate database within the ~/ourCA/ directory with the following command :

  echo '01' > serial && touch index.txt

  - The files, "serial" and "index.txt", which are now visible in the current directory (~/ourCA), serve for the sequential naming of copies of signed certificates, later on in the certificate creating process

Creating Certificates with OpenSSL – CA setup contd.

- Create a CA config file (caconfig.cnf), to used in the creation of CA certificates:
  - Using a suitable editor, e.g:
    gedit caconfig.cnf
  - Then, insert the contents of the file, caconfig.txt into this config file.
  - Customize the following fields to suite your environment:
    1. /home/<username>/ (under "**local_ca**" and "**req**".), with your own username (type **whoami**, at the command prompt)
    2. commonName, stateOrProvinceName countryName, etc (under "**root_ca_distinguished_name**")
  - Save as caconfig.cnf and close

Creating Certificates with OpenSSL – CA setup contd.

- Generate the CA Root Certificate and Key, as follows:
  - Create an environment variable (OPENSSL_CONF) that enables OpenSSL to look for a config file in the location, ~/ourCA/:

    export OPENSSL_CONF=~/ourCA/caconfig.cnf
  - Generate the CA certificate and key:

    openssl req -x509 -newkey rsa:2048 -out cacert.pem -outform PEM -days 1825
  - Enter an easy-to-remember passphrase
  - The CA public certificate (cacert.pem) and the CA private key (cakey.pem) can now be respectively seen in the following directories: /myCA/ and /myCA/private/

Creating and Signing a Certificate

- Create a config file (mycertconfig.cnf):
  - Using a suitable editor, e.g:
    gedit mycertconfig.cnf
  - Insert the contents of the file, mycertconfig.txt into this config file.
  - Customize the following fields to suite your environment:
    1. commonName, etc (under "**server_distinguished_name**".)
    2. commonName must match the host name (type **hostname**, at the command line) for the host for which the key will be used
  - Save the file as mycertconfig.cnf and close

Creating and Signing a Certificate – contd.

- Generate the host certificate and Key, as follows:
  - Create an environment variable (OPENSSL_CONF) that would enable OpenSSL to look for a config. file in the location, ~/ourCA/:
    export OPENSSL_CONF=~/ourCA/mycertconfig.cnf
  - Generate the certificate and key with the command:
    openssl req -newkey rsa:1024 -keyout tempkey.pem -keyform PEM -out tempreq.pem -outform PEM
    (tempkey.pem is the private key, while tempreq.pem is the ready-to-be-signed public key/certificate)
  - Enter an easy-to-remember passphrase
  - You can decrypt the temporary private key (tempkey.pem) into a new key (myprivkey.pem), as follows:
    openssl rsa < tempkey.pem > myprivkey.pem
  - (Leaving the private key encrypted will require entering the passphrase each time the system using the encrypted key is started, which can be inconveniencing. Obvious trade-off between better security and convenience!)
  - The decrypted private key (myprivkey.pem) can be seen in the directory, ~/ourCA/

Creating and Signing a Certificate – contd.

- Now, sign the certificate (tempreq.pem) as follows:
  - With the following command, modify the environment variable (OPENSSL_CONF) that would enable OpenSSL to look for a config. file in the location, ~/ourCA/ – OpenSSL switches back to the CA configuration:
    export OPENSSL_CONF=~/ourCA/caconfig.cnf
  - Sign the certificate with the command:
    openssl ca -in tempreq.pem -out mycert.pem
    (tempreq.pem is the certificate to be signed, while mycert.pem is the eventual name of the signed certificate)
  - Enter the CA's passphrase and follow the subsequent prompts to complete the signing process
  - The signed certificate (mycert.pem) and the private key (myprivkey.pem) can be seen in the current directory
  - cd to the directory, ~/ourCA/signedcerts/ to find a copy of the signed certificate, named in the serial format, "01.pem",… This way, the CA maintains a list of all the certificates it has signed.

Further Activity

Repeat the CA setup and certificate creation and signing procedure on a different machine (e.g., on a different guest VM), where you may then later apply the generated certificates in a crypto program, to implement secure communication between your machines (as “client” and “server”.)

OpenSSL Crypto Programming – Useful Links

Explore the following links, which provide important information on crypto programming steps, as well as specific APIs that can be applied:

http://h71000.www7.hp.com/doc/83final/ba554_90007/ch04s03.html

http://www.linuxjournal.com/article/4822

http://www.linuxjournal.com/article/5487

http://www.ibm.com/developerworks/library/l-openssl/