howest
hogeschool

Demystifying keys exchange Encryption in actions (ssh and https)

Chris Roets
Guy Van Eeckhout

What we learned in Cybersecurity Essentials

- Symmetric keys
  - Encryption and decryption is done using the same key.
  - Fast, but anyone having the key can encrypt and decrypt.
- Asymmetric keys
  - Private key
  - Public key
  - Slow

Different key usage

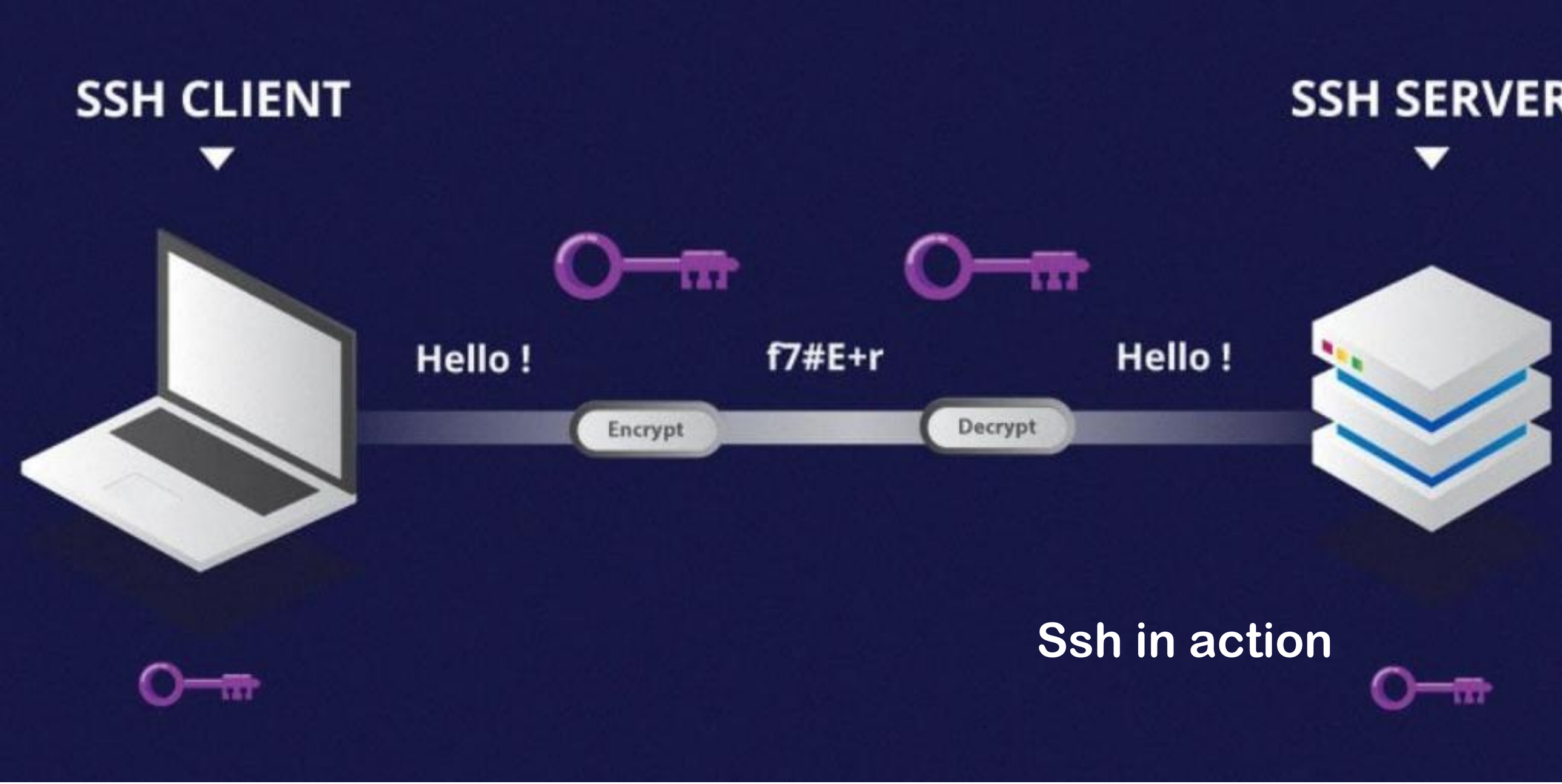
- Confidentiality : how can I make sure that my message can only be read by the one I want
    - Encrypt using host public key
    - Decrypt using his private key
  - Used in
    - Secure communication (https)
    - Sending encrypted messages to a specific recipient

Different key usage

- Integrity and Authenticity : How can I make sure that the message I receive comes from the one he claims to be and is not changed
    - Encrypt using his private key
    - Decrypt using his public key
  - Used in
    - Code signing
    - HTTPS certificates
    - Blockchain transactions

Timeline

- SSH
  - Year: 1995
  - Author: Tatu Ylönen, a researcher at Helsinki University of Technology
  - Context: Developed in response to password-sniffing attacks on the university network.
- OpenSSL
  - Origin: Derived from the earlier SSLeay library (created by Eric Young and Tim Hudson from 1995–1998).
  - OpenSSL project created: December 1998
  - First public release: OpenSSL 0.9.1 in December 1998, shortly after the SSLeay project ended.


SSH CLIENT
SSH SERVER
Hello !
f7#E+r
Hello !
Encrypt
Decrypt
Ssh in action

Ssh on the move (simple)

Session key

Session key(host public key)

Known hosts

Host authentication

Host public key(Session key)

Challenge==random number

Client private key(Challeng)

user authentication

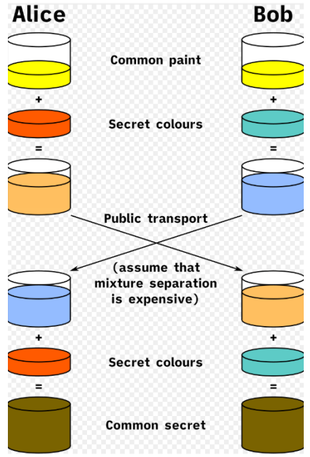
Authorized_keys(
client response

- Weak points :
  - Initial session key
  - Transport of public keys

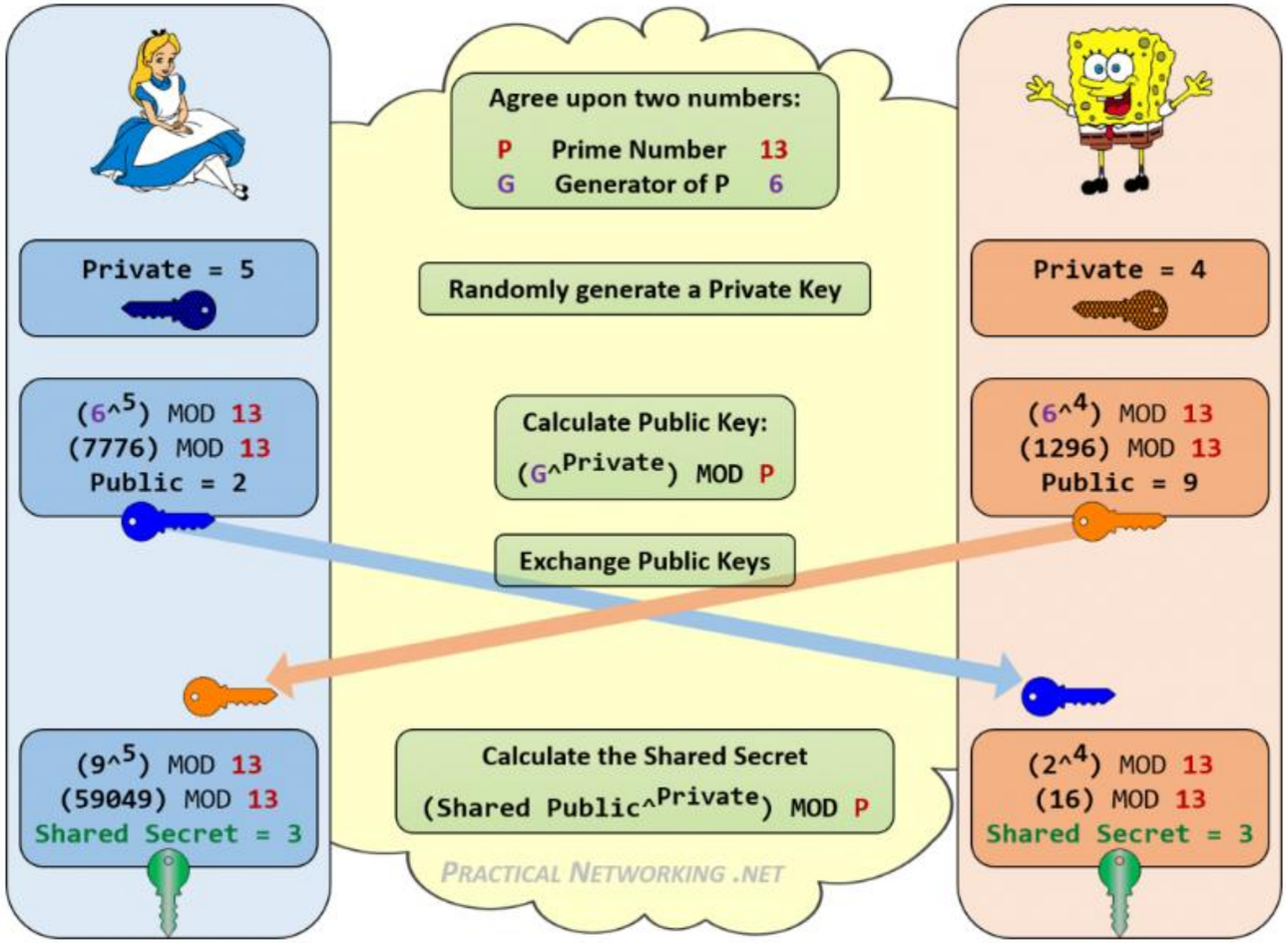
howest hogeschool

Ssh on the move (diffie-hellman key exchange)

- Agree on a temporary session symmetric key for key exchange
- It does NOT guarantee that the party you shared it with is the one you *intended*.



[Diffie–Hellman key exchange - Wikipedia]



howest hogeschool

Two factor authentication

SSH authentication with Microsoft Entra ID - Microsoft Entra | Microsoft Learn

Configure SSH to use two-factor authentication | Ubuntu

howest hogeschool

Local vs. Remote port forwarding

- Local forwarding
  - **ssh -L localsocket:listeningaddress:remotesocket user@gateway**

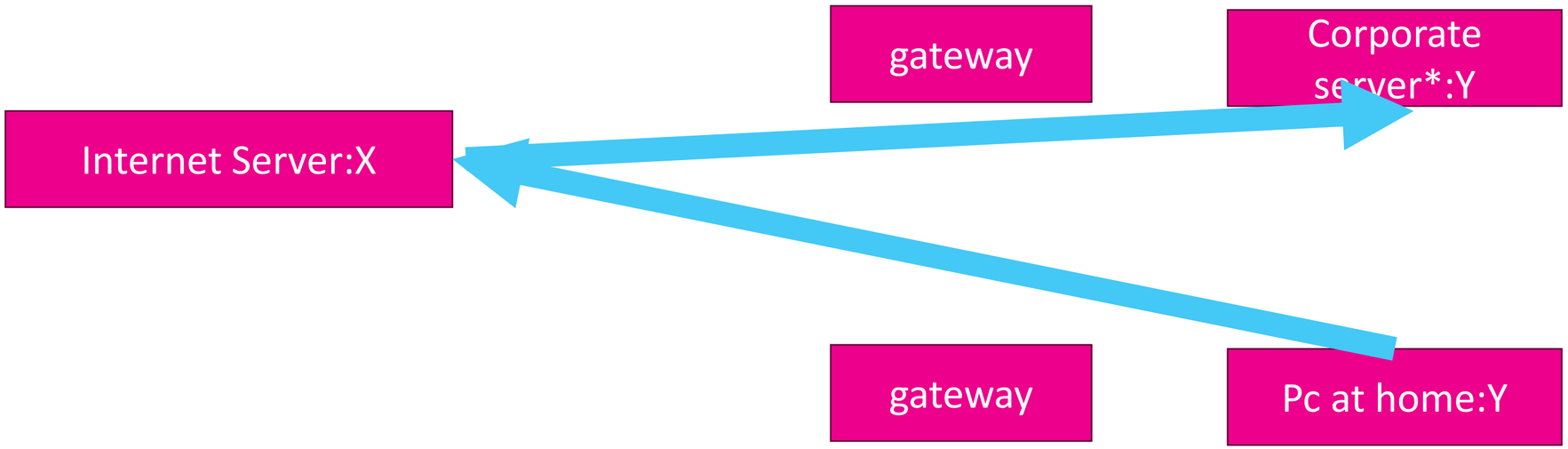
- Remote forwarding
  - **ssh -R remotesocket: listeningaddress:localsocket user@gateway**

Be aware privileged ports

howest hogeschool

Port forwarding example

ssh -R X:server:Y user@Internetserver



ssh -L Y:server:X user@Internetserver

howest hogeschool

Local vs. Reverse port forwarding

Local

Note:

- Works out of the box
- Remotehost and IP can both be external (and even the same)
- We forward localport to remotehost:remoteport

ssh -L localport:remotehost:remoteport user@IP

ssh -L 8080:reddit.com:80 sshuser@172.21.18.53

Local vs. Reverse port forwarding

Reverse

Downside:

- Need to edit /etc/ssh/sshd_config
  - “GatewayPorts yes”
- Restart ssh-daemon: systemctl restart sshd

ssh -R remoteport:localhost:localport user@IP

ssh -R 8888:localhost:8000 sshuser@172.21.18.53

Local vs. Reverse port forwarding

Use cases – Local or Reverse port forwarding?

Imagine a database server. The server itself has an SSH-server and runs the PostgreSQL service on port 5432. The DBA configured the database server to only allow local connections for security reasons. You however want to connect to the database console to check something out. You have an external IP (with SSH available) at your disposal. What is the easiest way to solve this issue? Where do I need to browse/connect to?

ssh -L 9000:localhost:5432 user@IP

→ Connect to localhost:9000

Local vs. Reverse port forwarding

Use cases – Local or Reverse port forwarding?

Imagine you were working on the Tetris-project last year. You are working at home behind NAT from your ISP. In your IDE you're hosting the application on localhost on port 8000. 2 colleagues want to test your newly added feature but they too are at home behind NAT. You have an external IP (with SSH-server available) at your disposal. What is the easiest way to solve this issue? Where do they need to browse/connect to?

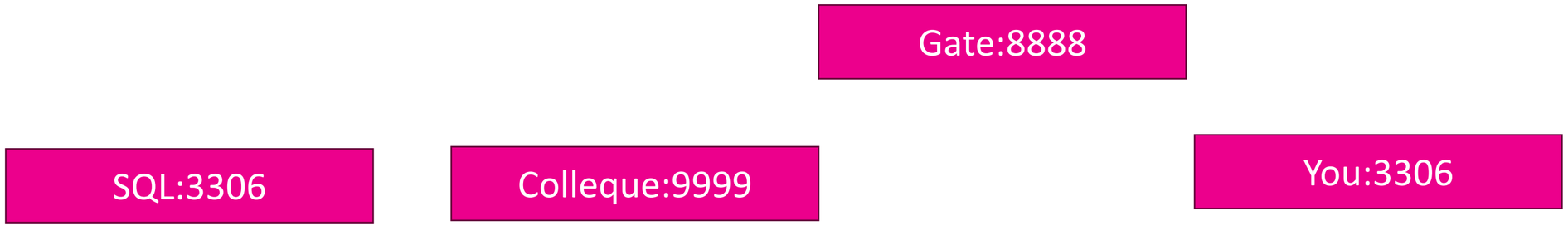
ssh -R 8888:localhost:8000 user@IP

→ Browse to IP:8888

Local vs. Reverse port forwarding

Use cases – Local or Reverse port forwarding?

Imagine you working for a company. You are working from a Starbucks coffee shop. The VPN to your internal network broke down but a colleague needs your help with something on the MySQL server (port 3306) This server has no SSH! Because it is a production server the data is highly sensitive and you want encrypted traffic everywhere. You have an external IP (with SSH-server available) at your disposal. What is the easiest way to solve this issue? Where do I need to browse/connect to?



howest hogeschool

Local vs. Reverse port forwarding

Use cases – Local or Reverse port forwarding?

Imagine you working for a company. You are working from a Starbucks coffee shop. The VPN to your internal network broke down but a colleague needs your help with something on the MySQL server (port 3306) This server has no SSH! Because it is a production server the data is highly sensitive and you want encrypted traffic everywhere. You have an external IP (with SSH-server available) at your disposal. What is the easiest way to solve this issue? Where do I need to browse/connect to?

Colleague: ssh -L 9999:localhost:3306 user@sql-server

Colleague: ssh -R 8888:localhost:9999 user@gateway

You ssh -L 3306:localhost:8888 user@gateway

→ You: Connect to localhost:3306

Overview

- What is OpenSSL
- SSL Protocol
- Command-Line Interface
- Application Programming Interface
- Problems with OpenSSL
- Summary

OpenSSL

What is OpenSSL

The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, fully featured, and Open Source toolkit implementing the SSL_v2/v3 and TLS_v1 protocols as well as a full-strength general purpose cryptography library.

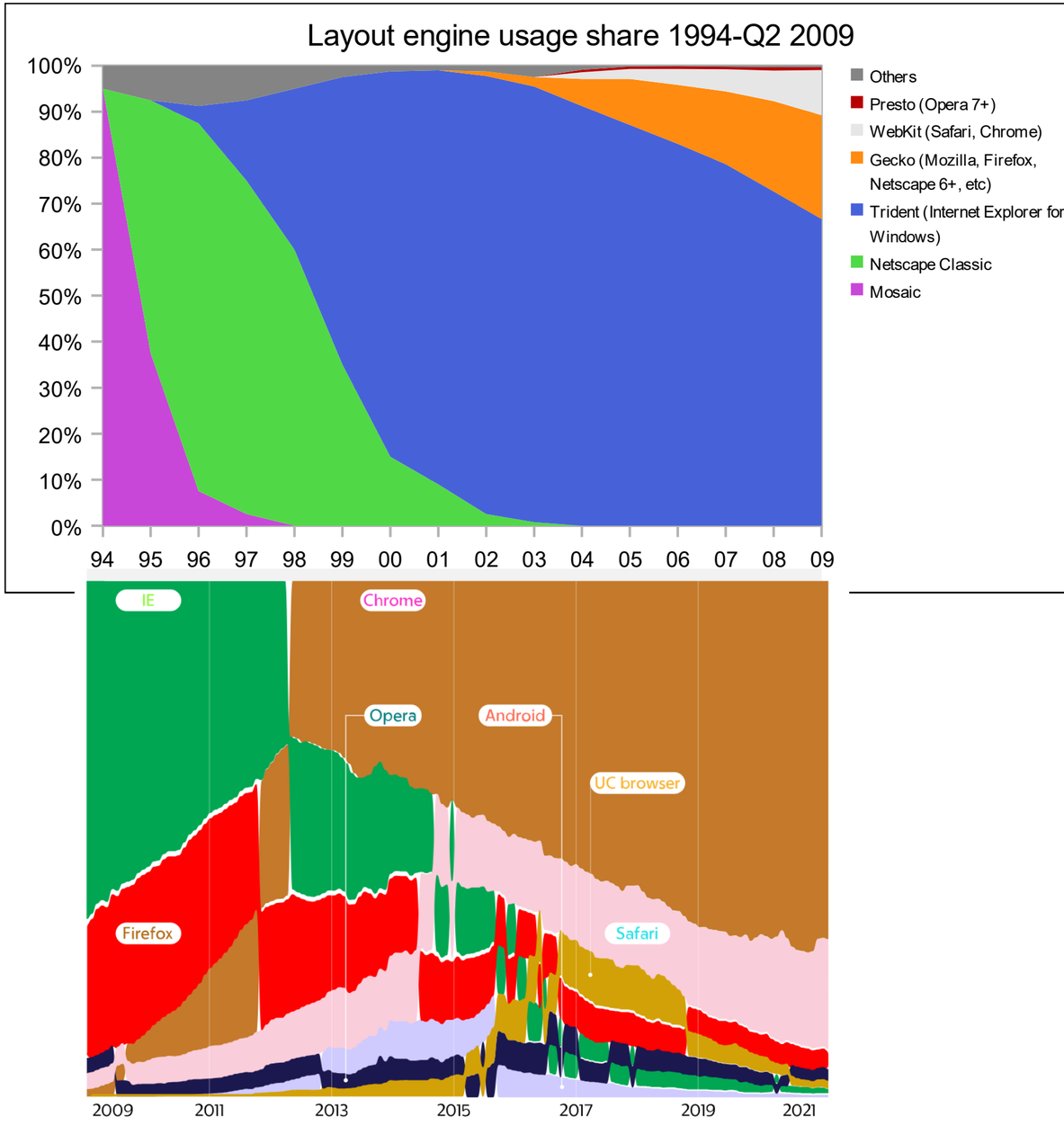
SSL : secure socket Layer

TLS : Transport Layer Security

The OpenSSL Project is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the toolkit and its related documentation.

OpenSSL is based on the excellent SSLeay library developed by Eric A. Young and Tim J. Hudson.

SSLeay is a freely available implementation of the Netscape 3.0 SSL protocol.



howest hogeschool

What is OpenSSL – Cont.

Features:

- Fully Functional Implementation
- Cross-Platform (Unix & Windows)
- Command-Line Interface (openssl command)
- Application Programming Interface (C/C++, Perl, PHP & Python)

howest hogeschool

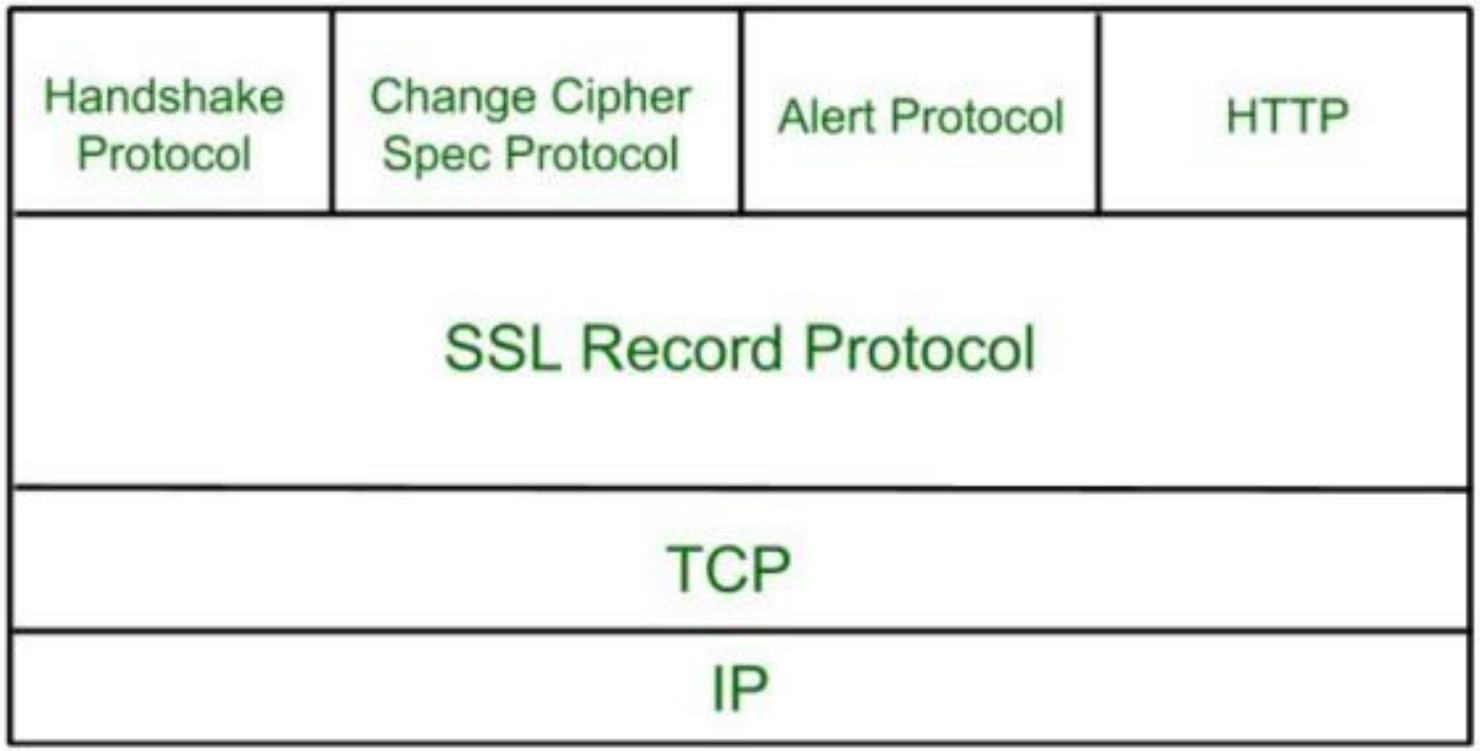
SSL Protocol

- The primary goal of the SSL (Secure Sockets Layer) Protocol and its successor - TLS (Transport Layer Security) Protocol is to provide privacy and reliability between two communicating applications.
- It is composed of four layers:
  - SSL Record Protocol
    - It is used for the transmission of (bulk) data.
  - SSL Handshake Protocol
    - It is used to establish the secure connection for data transfer.
  - Change-cipher spec protocol
    - SSL connection state (pending/active)
  - Alert protocol
    - Error handling

Our focus

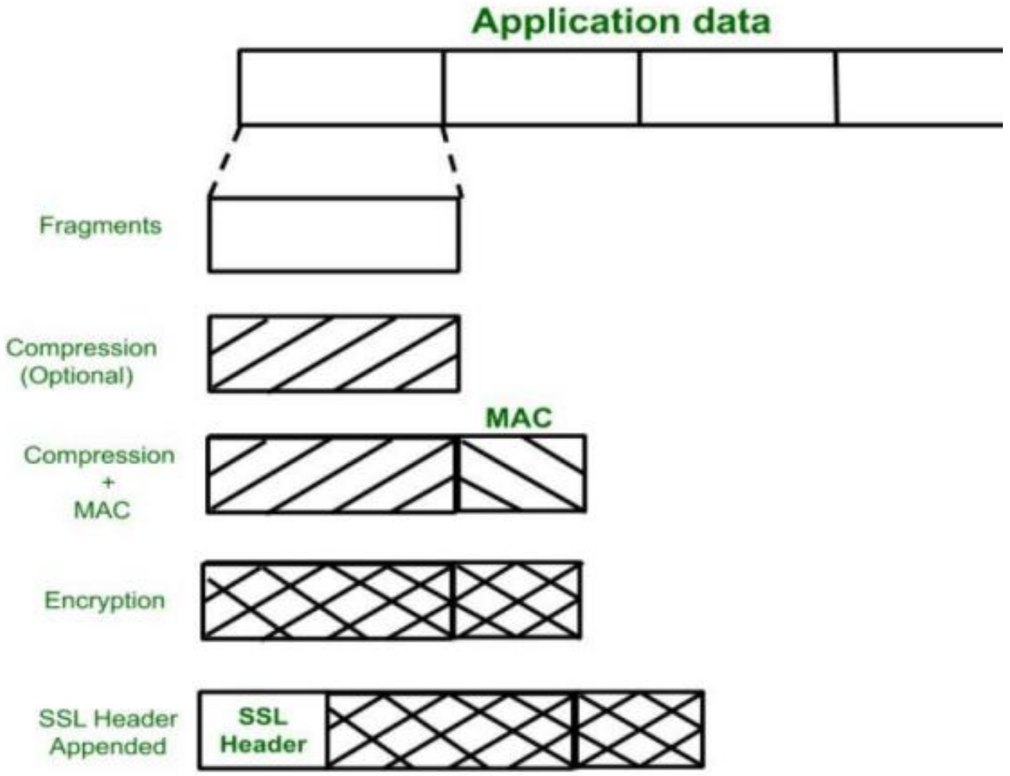
SSL Protocol Stack

SSL Protocol Stack:



SSL Record Protocol:

- SSL Record provides two services to SSL connection.
  - Confidentiality
  - Integrity and authenticity
- In the SSL Record Protocol application data is divided into fragments. The fragment is compressed and then encrypted MAC (Message Authentication Code) generated by algorithms like SHA (Secure Hash Protocol) and MD5 (Message Digest) is appended. After that encryption of the data is done and in last SSL header is appended to the data



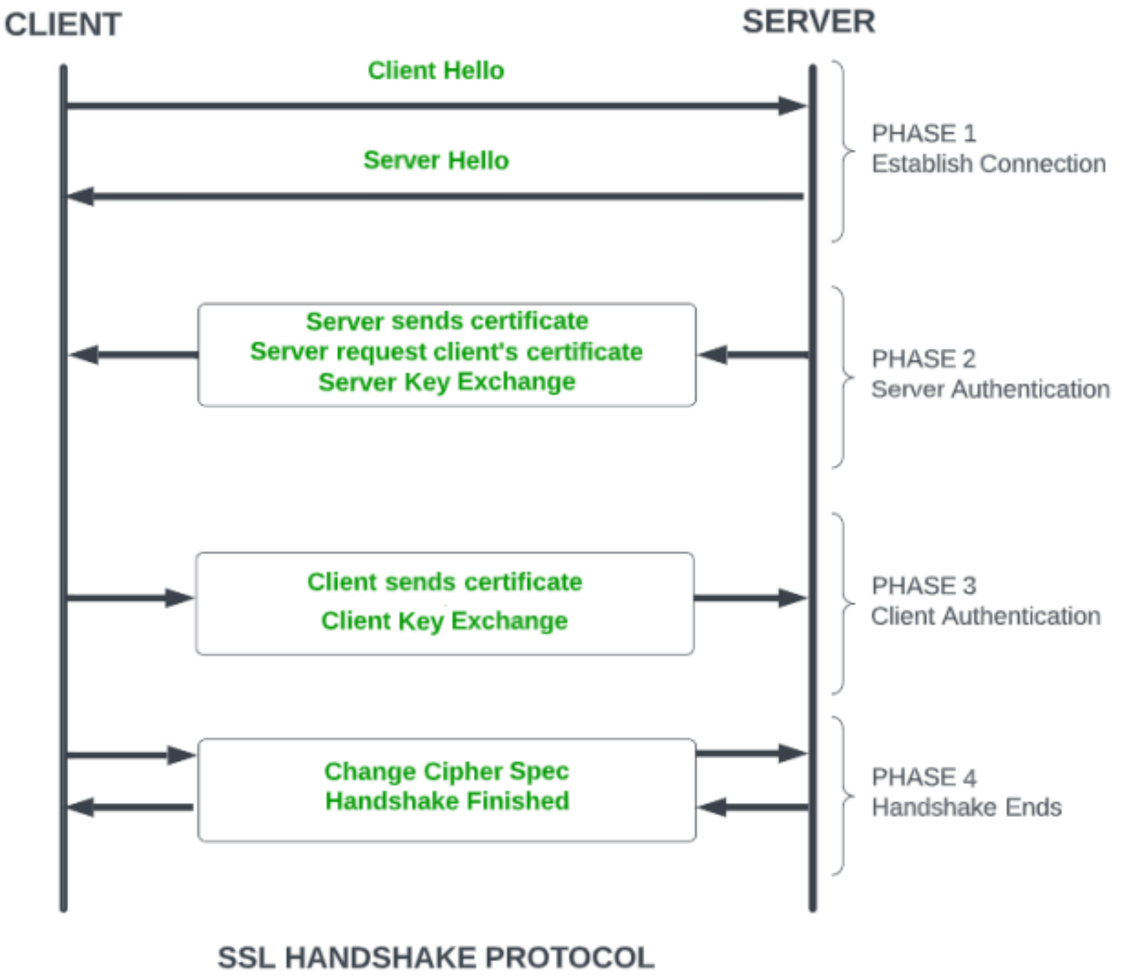
Handshake Protocol

Phase-1: In Phase-1 both Client and Server send hello-packets to each other. In this IP session, cipher suite and protocol version are exchanged for security purposes.

Phase-2: Server sends his certificate and Server-key-exchange. The server end phase-2 by sending the Server-hello-end packet.

Phase-3: In this phase, Client replies to the server by sending his certificate and Client-exchange-key.

Phase-4: In Phase-4 Change-cipher suite occurred and after this Handshake Protocol ends



howest hogeschool

Change-cipher Protocol

- This protocol uses the SSL record protocol. Unless Handshake Protocol is completed, the SSL record Output will be in a pending state. After the handshake protocol, the Pending state is converted into the current state. Change-cipher protocol consists of a single message which is 1 byte in length and can have only one value. This protocol's purpose is to cause the pending state to be copied into the current state

Alert Protocol

- This protocol is used to convey SSL-related alerts to the peer entity. Each message in this protocol contains 2 bytes. The level is further classified into two parts:
  - Warning (level = 1): This Alert has no impact on the connection between sender and receiver. Some of them are:
    - Bad certificate: When the received certificate is corrupt.
    - No certificate: When an appropriate certificate is not available.
    - Certificate expired: When a certificate has expired.
    - Certificate unknown: When some other unspecified issue arose in processing the certificate, rendering it unacceptable.
    - Close notify: It notifies that the sender will no longer send any messages in the connection.
  - Fatal Error (level = 2): This Alert breaks the connection between sender and receiver. The connection will be stopped, cannot be resumed but can be restarted. Some of them are:
    - Handshake failure: When the sender is unable to negotiate an acceptable set of security parameters given the options available.
    - Decompression failure: When the decompression function receives improper input.
    - Illegal parameters: When a field is out of range or inconsistent with other fields. Bad record MAC: When an incorrect MAC was received.
    - Unexpected message: When an inappropriate message is received. The second byte in the Alert protocol describes the error.

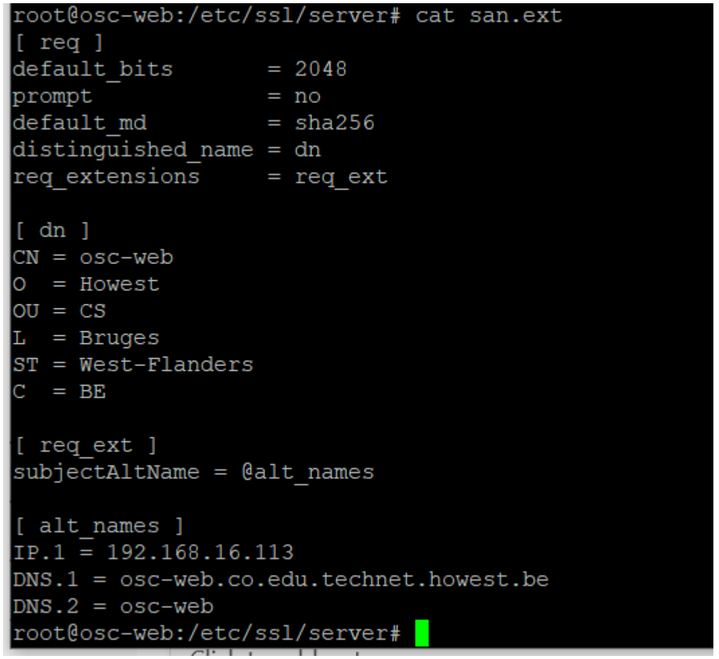
Mathematical assumptions

- We assume this as explaining this would take us to much time
    - CFR : cybersecurity essentials
- A text encoded with the private key can only be decoded by the public key :
    - If you encrypt something with a private key, it can only be read by the one who has the public key
    - If you encrypt something with a public key, it can only be read by the one who has the private key
- BUT
    - How can you be sure that the public key really comes from the one who he claims to be
    - You need a third party that can confirm this : Certificate Authority.


HTTP
HTTPS

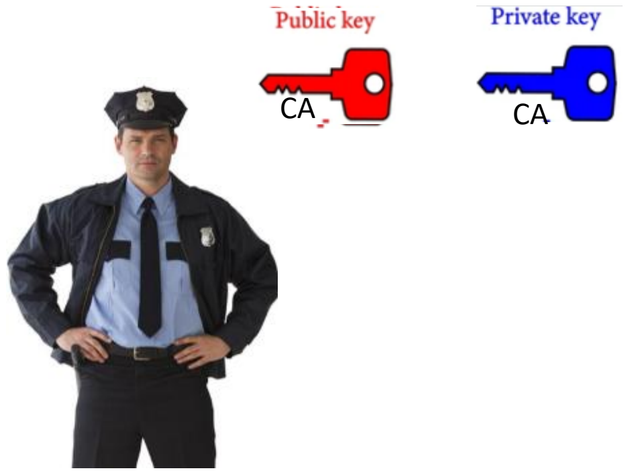
What is a certificate

- A certificate contains
  - To whom issued
  - By whom issued
  - Server information
  - Server public key
  - Signature : Server public key encrypted with the root-CA private key
    - NOT the CA key

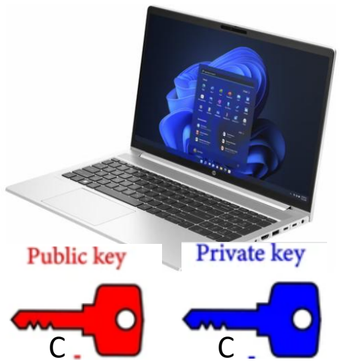


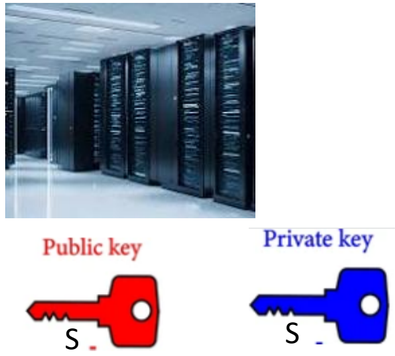
```
root@osc-web:/etc/ssl/server# cat san.ext
[ req ]
default_bits       = 2048
prompt             = no
default_md         = sha256
distinguished_name = dn
req_extensions     = req_ext

[ dn ]
CN = osc-web
O  = Howest
OU = CS
L  = Bruges
ST = West-Flanders
C  = BE

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
IP.1 = 192.168.16.113
DNS.1 = osc-web.co.edu.technet.howest.be
DNS.2 = osc-web
root@osc-web:/etc/ssl/server#
```

SSL Protocol – Setup.



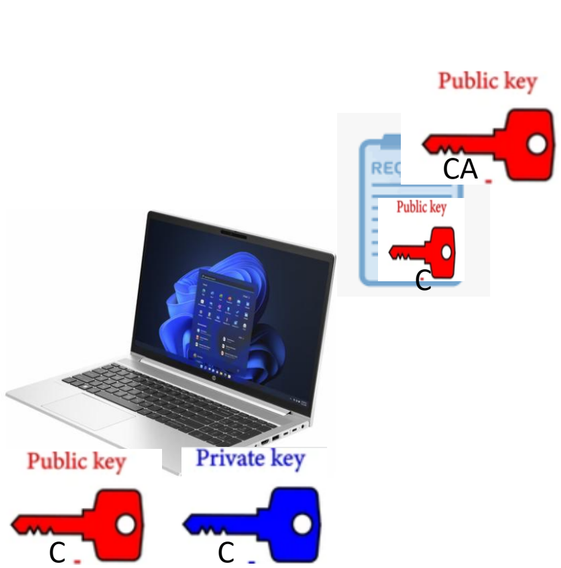
- Create private public key pair
  - openssl genrsa -des3 -passout pass:x -out mysite.pass.key 2048
  - openssl rsa -passin pass:x -in mysite.pass.key -out mysite.key
- Create or use a Certification Authority (CA)
  - Create example :
    - openssl req -x509 -new -nodes -key MyOrg-RootCA.key -sha256 -days 1826 -out MyOrg-RootCA.crt -subj '/CN=MyOrg Root CA/C=BE/ST=West-vlaanderen/L=Brugge/O=Howest'
  - Use example :Let's Encrypt
- Add new CA to Certification chain
  - SSLCertificateChainFile /etc/ssl/MyOrg-RootCA.crt, etc…
  - Certmgr. Etc….



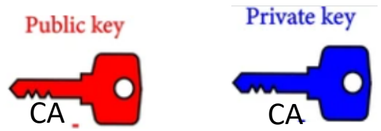


SSL Protocol – Register yourself.

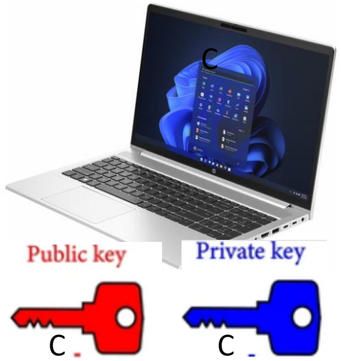
- Create a certificate request (csr)
  - Csr contains public key
- Register your certificate towards the CA
- Sign the certificate @ the CA

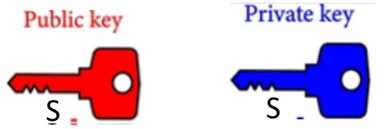


SSL Protocol – Client Server Authenticatio



- Server sends an initial request with its certificate
- Client decrypts the signature using the ROOT-CA public key
- If the same : ok
- Client send a response with its certificate
- Server decrypts the signature using the ROOT-CA public key
- If the same : ok
- Now they negotiate a session key





howest hogeschool

Certificate authority chaining

- Solves the problem of CA public key exchange
  - Initial once are on you distribution medium : ROOT-CA’s
  - When you create a CA it needs to be verified by a Parent CA
  - Same mechanism applies for you to add the CA-Authority

SSL-vertrouwensketen | Hoe SSL-keten werkt | Rootcertificaat, Intermediair certificaat, Servercer...

Possible problem scenarios

- Certificate received signed by an authority that is not known
  - Invalid CA error
  - Register the CA
- Decrypted server signature is not equal to the public key of the server
  - Somebody may be trying to steal your data
- Decrypted client signature is not equal to the public key of the server
  - Permission denied
- App kind of permissions
  - $SSL_CLIENT_S_DN_CN

Command-Line Interface

- Functionality
  - Creation of RSA, DSA & DH key pairs
  - Creation of X509 Certificates, CSRs & CRLs
  - Calculation of Message Digests
  - Encryption & Decryption with Ciphers
  - SSL/TLS Client & Server Tests
  - Handling of S/MIME signed and/or encrypted mails

Command-Line Interface – Cont.

- Example 1 – Secure Apache Web Server with mod_ssl & OpenSSL
- Example 2 – S/MIME

Secure Apache Web Server with mod_ssl & OpenSSL

- Generate the Root Certificate
- Generate the CSR (Certificate Signing Request)
- Sign the CSR
- Generate the PKCS12
- Modify the Apache Configuration File

Generate The Root Certificate

- openssl req -x509 -days 2922 -newkey rsa:1024 -md5 -out ca.crt -keyout ca.key -config .\openssl.cnf

Generate The CSR

- openssl req -newkey rsa:1024 -out mec.csr -keyout mec.key -config .\openssl.cnf -reqexts v3_req

Sign The CSR

- openssl x509 -req -in mec.csr -extfile .\openssl.cnf -extensions usr_cert -CA ca.crt -CAkey ca.key -CAcreateserial -sha1 -days 1461 -out mec.crt

Generate The PKCS12

- openssl pkcs12 -export -out mec.p12 -in mec.crt -inkey mec.key -certfile ca.crt

Modify The Apache Configuration File

The Apache Configuration File – httpd.conf

```
LoadModule ssl_module modules/libssl.so
AddModule mod_ssl.c

SSLEngine off
SSLSessionCache dbm:logs/ssl_cache
SSLSessionCacheTimeout 300

Listen 80
Listen 443
```

Modify The Apache Configuration File – Cont.

```
<VirtualHost _default_:80>
        <Location /admin>
                Deny from all
        </Location>
</VirtualHost>

<VirtualHost _default_:443>
        SSLEngine on
        SSLCertificateFile “conf/ssl.crt/mec.crt”
        SSLCertificateKeyFile “conf/ssl.key/mec.key”
        SSLCACertificateFile “conf/ssl.crt/ca.crt”
```

Modify The Apache Configuration File – Cont.

```
        <Location /admin>
                SSLVerifyClient require
                SSLRequire %{SSL_CLIENT_S_DN_CN} eq                    “Administrator”
        </Location>
</VirtualHost>
```

S/MIME

- Sign
  - openssl smime -sign -in m.txt -out sign_clear.eml -signer jingli.pem
- Verify
  - openssl smime -verify -in sign_clear.eml -signer jingli.pem -CAfile ca.crt

S/MIME – Cont.

- Encrypt
  - openssl smime -encrypt -des3 -in m.txt -out encrypt.eml jingli.crt
- Decrypt
  - openssl smime -decrypt -in encrypt.eml -recip jingli.pem
- Sign & Encrypt
  - openssl smime -sign -in m.txt -text -signer jingli.pem | openssl smime -encrypt -des3 -out sign_encrypt.eml jingli.pem

Application Programming Interface

- libssl.a or libssl.so
  - Implementation of SSL_v2/3 & TLS_v1
- libcrypto.a or libcrypto.so
  - Ciphers (AES, DES, RC2/4, Blowfish, IDEA)
  - Digests (MD5, SHA-1, MDC2)
  - Public Keys (RSA, DSA, DH)
  - X509s (ASN.1 DER & PEM)
  - Others (BIO, BASE64)

Application Programming Interface – Cont.

- OpenSSL's libraries are also used by other tools, such as OpenCA, OpenSSH, to implement secure transmission of data
- Using SSL Proxy, arbitrary socket connections can be secured by SSL

Problems with OpenSSL

- It is powerful, but is not easy for use
- Non object-oriented
- Be lack of documents, especially for APIs
- Problems with shared libraries in some platforms

Summary

- OpenSSL is an Open Source toolkit implementing SSL/TLS & Cryptography
- It has a command-line interface & an application programming interface
- There are a lot of tools using OpenSSL’s libraries to secure data or establish secure connections

Different key types

- *.p8, *.pkcs8,*.key, *.ppk : private keys
- *.pub : public key
- *.csr or *.req or sometimes *.p10 : Certificate Signing Request
    - Public key
    - CN, OU,etc…
    - SAN-ext
- *.crt or *.cer *.pem *.der : certificate
    - Public key
    - Information of root authority + signature with CA public key
- .p12 or .pfx : password protected certificate

Questions for other apps

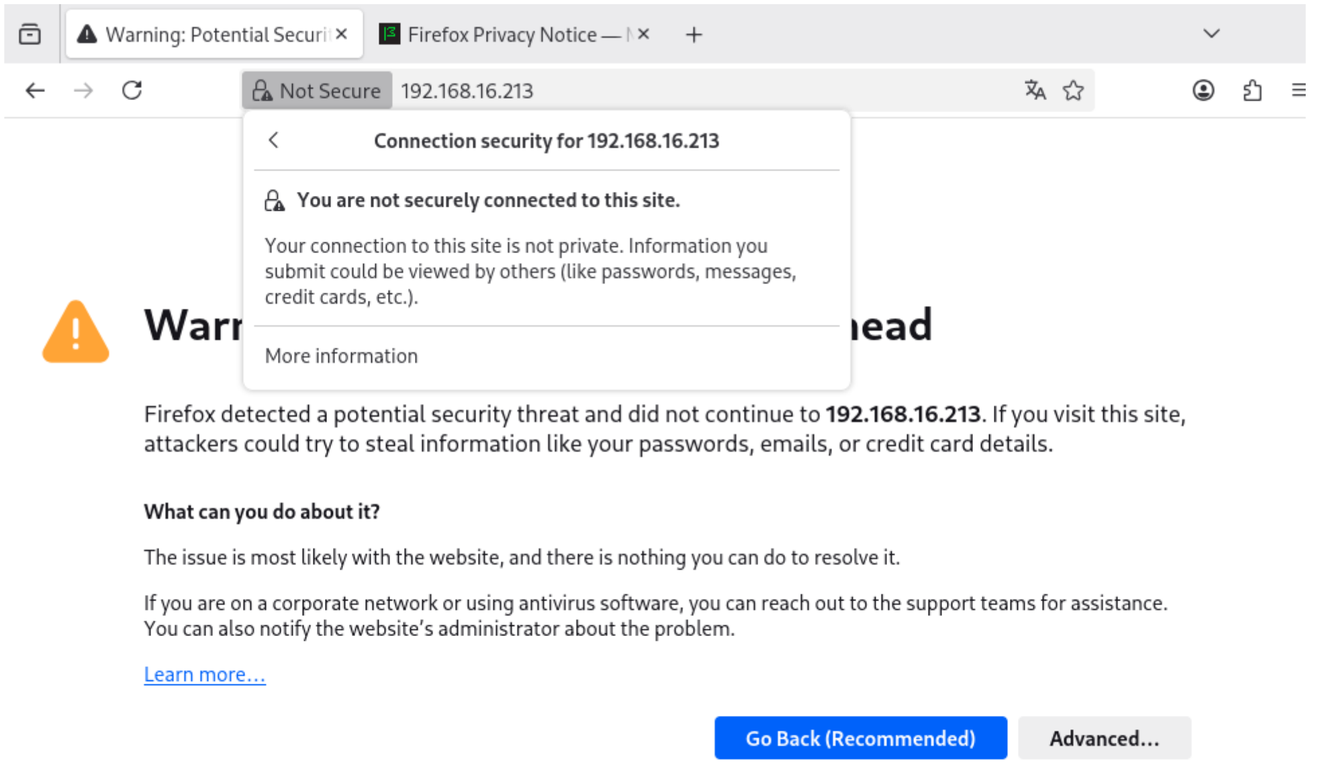
- Does the server know that its really the client
- Does the client know that it really the server

- Are most web-site protected like this, or are there other ways to authenticate
  - What happens if you login from another web browser ?
  - What happens if you forgot your password ?
- Does ssh has the same level of security, higher or lower ?
- Does itsme has the same level of security, higher or lower ?
- How about facebook/whatapp ?
- How about your public available home server ?

References

- *OpenSSL* – http://www.openssl.org
- *SSL* – http://www.netscape.com/eng/ssl3/draft302.txt -> obsolete
- *TLS* – http://www.ietf.org/rfc/rfc2246.txt
- *Apache* – http://www.apache.org
- *mod_ssl* – http://www.modssl.org
- *Network Security with OpenSSL* by Pravir Chandra, Matt Messier & John Viega
- *Applied Cryptography* by Bruce Schneier

Initial message



After root authority is install