Indice

Crittografia e Hash

Crittografia simmetrica

Supponiamo che Alice e Bob vogliano scambiarsi messaggi cifrati, apriamo due terminali nella stessa directory che simulano rispettivamente Alice e Bob.

Creare un file `plaintext-file` con un testo a piacere:

```
$ echo "Ciao sono Alice, questo è il messaggio in chiaro per bob" > plaintext-file
```

Per cifrare il file eseguire il comando seguente ed inserire una password (verrà poi richiesta per decifrare):

```
$ openssl enc -aes-256-cbc -pbkdf2 -in plaintext-file -out cyphertext-file
```

Verificare che il testo cifrato sia ora incomprensibile:

```
$ cat cyphertext-file
```

Per decifrare:

```
$ openssl enc -d -aes-256-cbc -pbkdf2 -in cyphertext-file -out unencrypted-file
```

Verificare che il testo decifrato corrisponde al testo in chiaro:

```
$ sha256sum plaintext-file
$ sha256sum unencrypted-file
```

Nota: *Nella cifratura simmetrica occorre che il mittente e il destinatario del messaggio (Alice e Bob) condividano la stessa password* → **encryption_key == decryption_key**.

Funzioni Hash e Message Authentication Codes

Hash

Le funzioni hash sono funzioni non invertibili utili per verificare l'integrità e rilevare possibili modifiche dei dati.

Hanno due scopi principali:

- Rilevare modifiche **accidentali** dei dati trasmessi (errori di trasmissione e propagazione)

- Rilevare modifiche **intenzionali** dei dati da parte di attaccanti che intercettano la comunicazione

Creare un file a piacere:

```
$ echo "This is file1" > file1
```

Calcolare l'hash utilizzando la funzione sha256 del file1 utilizzando il comando `sha256sum` o `openssl dgst --sha256`:

```
$ sha256sum file1
```

Modificare di un solo byte il file creato (*es: "This is file1" → "This is file2"*) e ricalcolare la funzione di hash:

```
$ sha256sum file1
```

Nota: *La modifica di un singolo byte del testo produce un hash completamente diverso.*

MAC

Le funzioni **MAC** (Message Authentication Code) utilizzano una chiave segreta per calcolare un digest del messaggio. Esistono diverse tipologie di funzioni MAC:

- HMAC → basate su funzioni hash
- CMAC → basate su cifratura a blocchi

Facendo riferimento al `file1` creato precedentemente:

```
$ openssl dgst --sha256 --hmac sharedkey file1
```

Questa funzione crea un digest utilizzando la funzione *sha256* e la chiave `sharedkey`. Con un digest è quindi possibile verificare integrità e autenticità, ma non si può garantire la riservatezza delle comunicazioni.

Avendo a disposizione la pre-shared key `sharedkey` il destinatario che riceve *file1* e il rispettivo digest riesce, ricalcolando il digest con il comando `openssl dgst --sha256 --hmac sharedkey file1`, a verificare (se i digest corrispondono) che il messaggio contentuo in *file1* **non** è stato modificato ed è stato generato da un utente in possesso della chiave `sharedkey`.

ECB Penguin

Supponiamo di avere a disposizione una chiave di cifratura di lunghezza pari a 16 bytes. Vogliamo cifrare dei dati che sono più di 16 bytes con l'algoritmo ECB. ECB è una tecnica utilizzata per l'applicazione di schemi di crittografia simmetrica che operano su blocchi di dati di lunghezza fissa B (16 bytes nel nostro caso).

Visualizziamo ora graficamente il risultato della cifratura utilizzando l'algoritmo ECB.

Scarichiamo e visualizziamo l'immagine in formato PPM (è un formato composto da un header in formato ASCII seguito da una sequenza di 3-bytes RGB che rappresentano i pixels).

```
$ wget https://raw.githubusercontent.com/robertdavidgraham/ecb-penguin/master/Tux.ppm \
-O tux.ppm

$ ristretto tux.ppm
```

Ora estraiamo l'header e il body dell'immagine:

```
$ head -n 3 tux.ppm > header.txt

$ tail -n +4 tux.ppm > body.bin
```

Applichiamo l'algoritmo di cifratura ECB al body dell'immagine (provate con password diverse):

```
$ openssl enc -aes-128-ecb -nosalt -pass pass:"password" -in body.bin -out body.ecb.bin
```

Ricostruiamo l'immagine aggiungendo l'header al body cifrato:

```
$ cat header.txt body.ecb.bin > tux.ecb.ppm
```

Visualizziamo l'immagine:

```
$ ristretto tux.ecb.ppm
```

Nota: Cosa notate?

Crittografia asimmetrica

A differenza della crittografia simmetrica Alice e Bob utilizzando chiavi diverse **encryption_key != decryption_key** per cifrare le comunicazioni, dove la **encryption_key** è pubblica, mentre la **decryption_key** è segreta.

Aprire la rete marionnet `alice_bob.mar` presente nell'archivio ed eseguire i seguenti comandi.

Creare la chiave privata per **Alice** in formato pem:

```
root@alice:$ openssl genrsa -aes128 -out alice_private.pem 1024
```

Creare la corrispettiva chiave pubblica di **Alice**:

```
root@alice:$ openssl rsa -in alice_private.pem -pubout > alice_public.pem
```

Condividere la chiave pubblica di **Alice** con **Bob**:

```
root@bob:$ nc -l -p 8080 > alice_public.pem
```

```
root@alice:$ nc bob 8080 < alice_public.pem -q1
```

Bob crea un file segreto e lo cifra con la chiave pubblica di **Alice**:

```
root@bob:$ echo "Testo segreto" > top_secret.txt
root@bob:$ openssl rsautl -encrypt -inkey alice_public.pem -pubin -in top_secret.txt \
-out top_secret.enc
```

Inviare il file cifrato ad **Alice**:

```
root@alice:$ nc -l -p 8080 > top_secret.enc
```

```
root@bob:$ nc alice 8080 < top_secret.enc -q1
```

Decifrare il file appena creato con la chiave privata di **Alice** e controllare il contenuto:

```
root@alice:$ openssl rsautl -decrypt -inkey alice_private.pem -in top_secret.enc > \
top_secret.txt
root@alice:$ cat top_secret.txt
```

Firma file

Firmare il file `top_secret.txt` su **Alice**:

```
root@alice:$ openssl dgst -sha256 -sign alice_private.pem -out signature top_secret.txt
```

Verifica della firma del file:

```
root@alice:$ openssl dgst -sha256 -verify alice_public.pem -signature signature \
top_secret.txt
```

Certification Authority e Certificati

Una Certification Authority (CA) è un'entità che firma certificati digitali (notaio del web). La maggioranza dei siti Web attuali rassicura i propri utenti utilizzando una connessione sicura per lo scambio dei dati tra client e server (protocollo HTTPS). Per garantire: **autenticità**, **riservatezza** e **integrità** dei dati scambiati si utilizza il protocollo TLS e certificati digitali.

La rete marionnet `ca_certificate.mar` presente nell'archivio mostra un'esempio di infrastruttura necessaria ad implementare una piccola catena di certificati. All'avvio della rete vengono avviati 4 terminali per i seguenti hosts: **caroot**, **caintermediate**, **server**, **client**.

In particolare:

- **caroot**: implementa la Certificate Authority (CA).
- **caintermediate**: implementa la Intermediate Certificate Authority.
- **server**: implementa un server Web con protocollo HTTPS per l'azienda *www.fake.com*.
- **client**: implementa un client che vuole interagire con il server *www.fake.com*.

CA root

Si parte con la creazione delle strutture dati necessarie per la CA (host *root*).

```
root@root:$ mkdir /root/ca
root@root:$ cd /root/ca
root@root:$ mkdir certs crl newcerts private
root@root:$ chmod 700 private
root@root:$ touch index.txt
root@root:$ echo 1000 > serial
```

I files `index.txt` e `serial` agiscono come database dove vengono salvati dei certificati firmati.

Nella root directory dovrebbe essere già presente il file di configurazione `openssl.cnf` necessario per le fasi successive, prima di proseguire spostarlo nella directory `/root/ca` appena creata.

```
root@intermediate:$ mv /openssl.cnf /root/ca/openssl.cnf
```

In alternativa, tramite firefox raggiungere il seguente link https://pastebin.com/raw/rrF7Qb06 e copiare il contenuto del file di sul file `/root/ca/openssl.cnf`.

Creazione della chiave di root per la CA:

```
root@root:$ cd /root/ca
root@root:$ openssl genrsa -aes256 -out private/ca.key.pem 4096

> Enter pass phrase for ca.key.pem: rootpassword
> Verifying - Enter pass phrase for ca.key.pem: rootpassword

root@root:$ chmod 400 private/ca.key.pem
```

Creazione del certificato per la root CA:

```
root@root:$ cd /root/ca
root@root:$ openssl req -config openssl.cnf -key private/ca.key.pem -new -x509 \
-days 7300 -extensions v3_ca -out certs/ca.cert.pem


> Enter pass phrase for ca.key.pem: rootpassword
> You are about to be asked to enter information that will be incorporated
```

```
> into your certificate request.
> -----
> Country Name (2 letter code) []:
> State or Province Name []:
> Locality Name []:
> Organization Name []:
> Organizational Unit Name []:Certificate Authority
> Common Name []:Root CA
> Email Address []:

root@root:$ chmod 444 certs/ca.cert.pem
```

Verifica del certificato:

```
root@root:$ openssl x509 -noout -text -in certs/ca.cert.pem
```

CA intermediate

Creazione dell'ambiente necessario al per l'host *intermediate*:

```
root@intermediate:$ mkdir /root/intermediate
root@intermediate:$ cd /root/intermediate
root@intermediate:$ mkdir certs crl csr newcerts private
root@intermediate:$ chmod 700 private
root@intermediate:$ touch index.txt
root@intermediate:$ echo 1000 > serial
```

Nella root directory dovrebbe essere già presente il file di configurazione `openssl.cnf` necessario per le fasi successive, prima di proseguire spostarlo nella directory `/root/intermediate` appena creata.

```
root@intermediate:$ mv /openssl.cnf /root/intermediate/openssl.cnf
```

In alternativa, tramite firefox raggiungere il seguente link https://pastebin.com/raw/FaDrvurT e copiare il contenuto del file di sul file `/root/intermediate/openssl.cnf`.

Creazione della chiave privata per la CA intermediate:

```
root@intermediate:$ openssl genrsa -aes256 -out private/intermediate.key.pem 4096

> Enter pass phrase for intermediate.key.pem: interpassword
> Verifying - Enter pass phrase for intermediate.key.pem: interpassword

chmod 400 private/intermediate.key.pem
```

Creazione di una certificate signing request (CSR) per la Intermediate CA. In generale i campi corrispondono a quelli della CA root, il campo **Common Name** invece deve essere diverso.

```
root@intermediate:$ cd /root/intermediate
root@intermediate:$ openssl req -config openssl.cnf -new -sha256 \
      -key private/intermediate.key.pem \
      -out csr/intermediate.csr.pem

> Enter pass phrase for intermediate.key.pem: interpassword
> You are about to be asked to enter information that will be incorporated
> into your certificate request.
> -----
> Country Name (2 letter code) []: IT
```

```
> State or Province Name []:Italy
> Locality Name []:
> Organization Name []:
> Organizational Unit Name []:Certificate Authority
> Common Name []:Intermediate CA
> Email Address []:
```

Preparazione dell'ambiente della CA root:

```
root@root:$ mkdir /root/ca/intermediate/
root@root:$ cd /root/ca/intermediate
root@root:$ mkdir certs csr
root@root:$ cd csr
root@root:$ nc -l -p 8080 > intermediate.csr.pem
```

Invio della CSR alla CA root:

```
root@intermediate:$ cd /root/intermediate
root@intermediate:$ nc ca_root 8080 < csr/intermediate.csr.pem -q1
```

Creazione del certificato da parte della CA root (specificare l'estensione v3_intermediate_ca):

```
root@root:$ cd /root/ca
root@root:$  openssl ca -config openssl.cnf -extensions v3_intermediate_ca \
      -days 3650 -notext -md sha256 \
      -in intermediate/csr/intermediate.csr.pem \
      -out intermediate/certs/intermediate.cert.pem

> Enter pass phrase for ca.key.pem: rootpassword
> Sign the certificate? [y/n]: y

root@root:$ chmod 444 intermediate/certs/intermediate.cert.pem
```

Nota: Adesso file `index.txt` contiene un record che fa riferimento alla CA intermediate

Verifica del certificato appena creato:

```
root@root:$ cd /root/ca
root@root:$ openssl x509 -noout -text \
      -in intermediate/certs/intermediate.cert.pem

root@root:$ openssl verify -CAfile certs/ca.cert.pem \
      intermediate/certs/intermediate.cert.pem
```

Creazione del certificate chain file (verrà utilizzato in seguito dal client per accedere al server Web in esecuzione su *server*):

```
root@root:$ cd /root/ca
root@root:$ cat intermediate/certs/intermediate.cert.pem \
      certs/ca.cert.pem > intermediate/certs/ca_chain.cert.pem

root@root:$ chmod 444 intermediate/certs/ca_chain.cert.pem
```

Invio del certificato dalla CA root alla CA intermediate:

```
cd /root/intermediate/certs
root@intermediate:$ nc -l -p 8080 > intermediate.cert.pem
```

```
cd /root/ca/intermediate/certs
root@root:$ nc ca_intermediate 8080 < intermediate.cert.pem -q1
```

Invio della certificate chain alla CA intermediate:

```
root@intermediate:$ cd /root/intermediate/certs
root@intermediate:$ nc -l -p 8080 > ca_chain.cert.pem
```

```
root@root:$ cd /root/ca/intermediate/certs
root@root:$ nc ca_intermediate 8080 < ca_chain.cert.pem -q1
```

Server

Preparazione dell'ambiente per il server Web in esecuzione su *server*. Anche se 4096 bits, sono molto più sicuri rispetto a 2048 bits, in questa esercitazione per evitare rallentamenti durante il TLS handshaking useremo una chiave a 2048 bits. Inoltre per evitare di inserire la password della chiave ad ogni avvio del server (implementato da Apache2) si ometterà l'opzione `-aes256` creando di conseguenza una chiave senza password.

```
root@server:$ mkdir /root/server
root@server:$ cd /root/server
root@server:$ mkdir private certs csr
root@server:$ openssl genrsa -out private/www.fake.com.key.pem 2048

root@server:$ chmod 400 private/www.fake.com.key.pem
```

Nella root directory dovrebbe essere già presente il file di configurazione `openssl.cnf` necessario per le fasi successive, prima di proseguire spostarlo nella directory `/root/server` appena creata.

```
root@server:$ mv /openssl.cnf /root/server/openssl.cnf
```

In alternativa, tramite firefox raggiungere il seguente link https://pastebin.com/raw/d40FZgQK e copiare il contenuto del file di sul file `/root/server/openssl.cnf`.

Creazione di una CSR partendo dalla chiave privata appena generata.

Nota: *L'unico vincolo che occorre rispettare nei parametri di configurazione della CSR per il server è relativo al* ***Common Name****, che deve rispettare quello del dominio del server (www.fake.com).*

```
root@server:$ cd /root/server
root@server:$ openssl req -condif openssl.cnf -key private/www.fake.com.key.pem \
      -new -sha256 -out csr/www.fake.com.csr.pem

> You are about to be asked to enter information that will be incorporated
> into your certificate request.
> -----
> Country Name (2 letter code) [XX]:IT
> State or Province Name []:Modena
> Locality Name []:Modena
> Organization Name []:Fake
> Organizational Unit Name []:Fake Web Services
> Common Name []:www.fake.com
> Email Address []:
```

Invio della CSR del server alla CA intermediate. Preparazione dell'ambiente su **caintermediate**:

```
root@intermediate:$ mkdir /root/intermediate/servers
root@intermediate:$ cd /root/intermediate/servers
```

```
root@intermediate:$ mkdir csr certs
root@intermediate:$ nc -l -p 8080 > csr/www.fake.com.csr.pem
```

Invio della CSR del server:

```
root@server:$ cd /root/server/csr
root@server:$ nc ca_intermediate 8080 < www.fake.com.csr.pem -q1
```

Creazione e verifica del certificato da parte della CA intermediate:

```
root@intermediate:$ cd /root/intermediate
root@intermediate:$ openssl ca -config openssl.cnf \
      -extensions server_cert -days 375 -notext -md sha256 \
      -in servers/csr/www.fake.com.csr.pem \
      -out servers/certs/www.fake.com.cert.pem
root@intermediate:$ chmod 444 servers/certs/www.fake.com.cert.pem
root@intermediate:$ openssl x509 -noout -text -in servers/certs/www.fake.com.cert.pem
```

Nota: *Adesso file* ***index.txt*** *della CA intermediate contiene un record che fa riferimento al certificato del server appena creato.*

Verificare che il nuovo certificato di *www.fake.com* rispetti la certificate chain creata precedentemente:

```
root@intermediate:$ cd /root/intermediate
openssl verify -CAfile certs/ca_chain.cert.pem servers/certs/www.fake.com.cert.pem

> www.fake.com.cert.pem: OK
```

Invio del certificato al server Web *server*:

```
root@server:$ cd /root/server/certs
root@server:$ nc -l -p 8080 > www.fake.cert.pem
```

```
root@intermediate:$ cd /root/intermediate/servers/certs
root@intermediate:$ nc www.fake.com 8080 < www.fake.com.cert.pem -q1
```

Invio della certificate chain al server:

```
root@server:$ cd /root/server/certs
root@server:$ nc -l -p 8080 > ca_chain.cert.pem
```

```
root@intermediate:$ cd /root/intermediate/servers/certs
root@intermediate:$ nc www.fake.com 8080 < ca_chain.cert.pem -q1
```

Configurazione del Web Server Apache2 con certificali SSL/TLS.

Per funzionare con SSL/TLS Apache2 necessità di del modulo `ssl`. Verificare la presenza del modulo con il comando `apachectl -M` che elenca tutti i moduli caricati, se il modulo `ssl` non è presente abilitarlo con il comando `a2enmod ssl`.

Copiare la configurazione di default di Apache2:

```
root@server:$ cd /etc/apache2/sites-available
root@server:$ cp default-ssl.conf fake-ssl.conf
root@server:$ cd /etc/apache2/sites-enabled
root@server:$ ln -s ../sites-available/fake-ssl.conf
```

Verificare che nel file `/etc/apache2/sites-enabled/fake-ssl.conf` sia presente la seguente direttiva che abilita il supporto SSL per il sito specifico:

```
SSLEngine on
```

Copia del certificato e della chiave privata:

```
root@server:$ cp /root/server/private/www.fake.come.key.pem /etc/ssl/private/
root@server:$ cp /root/server/certs/www.fake.come.cert.pem /etc/ssl/certs/
root@server:$ cp /root/server/certs/ca_chain.cert.pem /etc/ssl/certs/
```

Ultimare la configurazione di Apache2 nel file `/etc/apache2/sites-available/fake-ssl.conf` modificando, e aggiungendo se non presenti, le seguenti direttive:

```
ServerName www.fake.com

SSLCertificateFile
/etc/ssl/certs/www.fake.com.cert.pem
SSLCertificateKeyFile
/etc/ssl/private/www.fake.com.key.pem
SSLCertificateChainFile
/etc/ssl/certs/ca_chain.cert.pem
```

Riavviare Apache2 in esecuzione su server:

```
root@server:$ service apache2 restart
```

Client

I browser di default quando vengono installati importano la maggior parte dei certificati delle CA root per poter verificare l'attendibilità dei certificati rilasciati dai server che utilizzano il protocollo HTTPS. Simuliamo questa situazione importando il certificato della CA root sull'host **client**.

```
root@client:$ nc -l -p 8080 > ca_root.cert.pem
```

```
root@root:$ cd /root/ca/certs
root@root:$ nc client 8080 < ca.cert.pem -q1
```

Ora provare a raggiungere dal client il sito *www.fake.com* utilizzando HTTPS.

```
root@client:$ w3m_ssl ca_root.cert.pem https://www.fake.com
```

Analisi del traffico TSL generato da Firefox con Wireshark

Se si dispone dei file di log delle chiavi dal browser di un client, è possibile importarli in Wireshark per decodificare il traffico da tale client. I file di log delle chiavi vengono generati dai browser Google Chrome, Firefox e Opera che hanno la variabile di ambiente `SSLKEYLOGFILE` impostata.

Impostare la variale d'ambiente `SSLKEYLOGFILE`:

```
$ export SSLKEYLOGFILE=sslkeylog
```

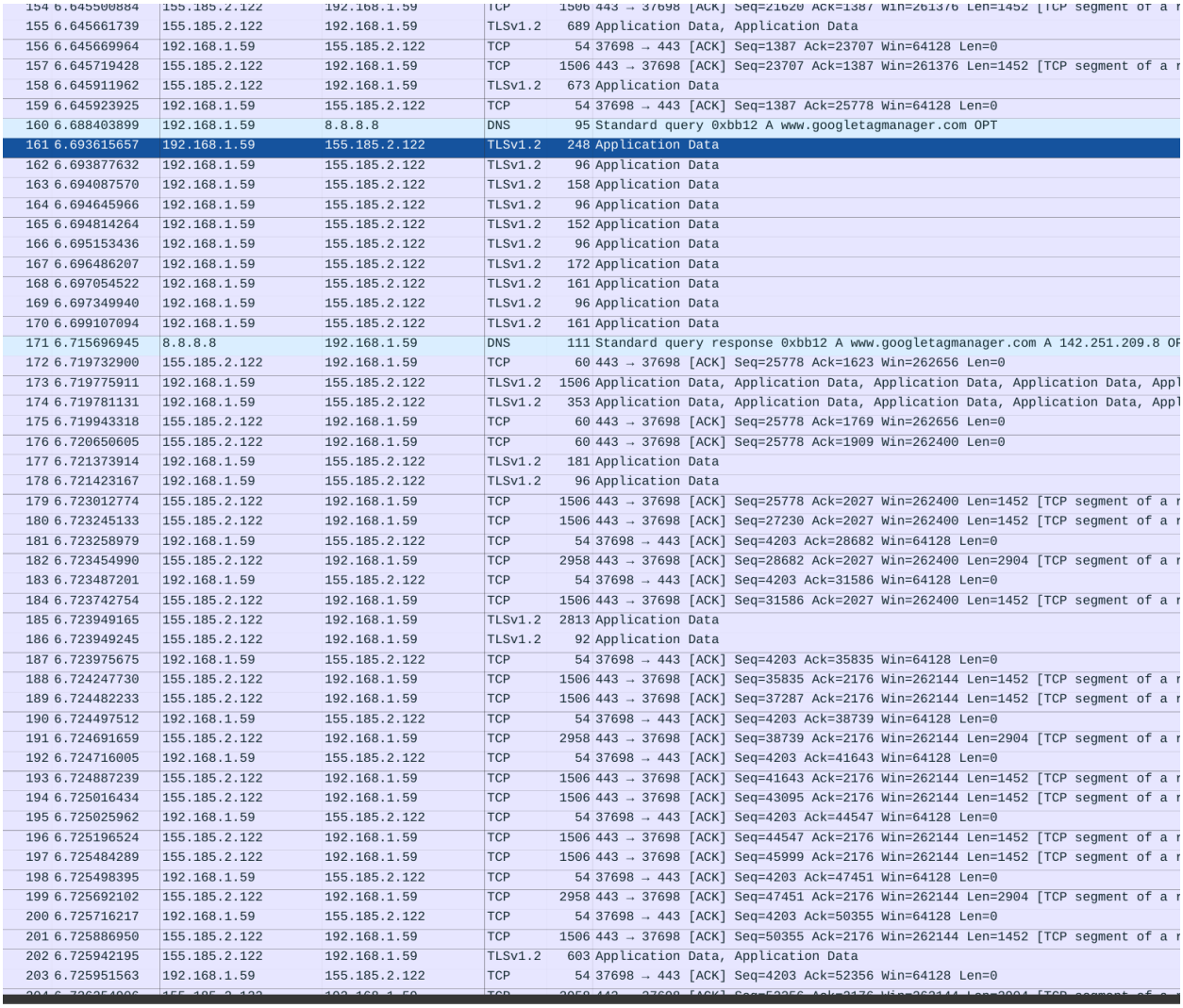
Avviare Wireshark ed iniziate a catturare il traffico (tasto in alto a sinistra) **Nota**: Sulle macchine di laboratorio serve `sudo` per poter catturare il traffico dell'interfaccia *ens3*):

```
$ sudo wireshark &
```

Avviare firefox da linea di comando (**Nota**: è fondamentale avviarlo nello stesso terminale in cui è stata fatta la export precedente)

```
$ firefox &
```

Tramite Firefox ora navigate all'indirizzo `www.unimore.it`, una volta che la pagina è stata caricata completamente, fermate la cattura di rete di wireshark.



154	6.645500884	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=21620 Ack=1387 Win=261376 Len=1452 [TCP segment of a r
155	6.645661739	155.185.2.122	192.168.1.59	TLSv1.2	689 Application Data, Application Data
156	6.645669964	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=1387 Ack=23707 Win=64128 Len=0
157	6.645719428	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=23707 Ack=1387 Win=261376 Len=1452 [TCP segment of a r
158	6.645911962	155.185.2.122	192.168.1.59	TLSv1.2	673 Application Data
159	6.645923925	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=1387 Ack=25778 Win=64128 Len=0
160	6.688403899	192.168.1.59	8.8.8.8	DNS	95 Standard query 0xbb12 A www.googletagmanager.com OPT
161	6.693615657	192.168.1.59	155.185.2.122	TLSv1.2	248 Application Data
162	6.693877632	192.168.1.59	155.185.2.122	TLSv1.2	96 Application Data
163	6.694087570	192.168.1.59	155.185.2.122	TLSv1.2	158 Application Data
164	6.694645966	192.168.1.59	155.185.2.122	TLSv1.2	96 Application Data
165	6.694814264	192.168.1.59	155.185.2.122	TLSv1.2	152 Application Data
166	6.695153436	192.168.1.59	155.185.2.122	TLSv1.2	96 Application Data
167	6.696486207	192.168.1.59	155.185.2.122	TLSv1.2	172 Application Data
168	6.697054522	192.168.1.59	155.185.2.122	TLSv1.2	161 Application Data
169	6.697349940	192.168.1.59	155.185.2.122	TLSv1.2	96 Application Data
170	6.699107094	192.168.1.59	155.185.2.122	TLSv1.2	161 Application Data
171	6.715696945	8.8.8.8	192.168.1.59	DNS	111 Standard query response 0xbb12 A www.googletagmanager.com A 142.251.209.8 OP
172	6.719732900	155.185.2.122	192.168.1.59	TCP	60 443 → 37698 [ACK] Seq=25778 Ack=1623 Win=262656 Len=0
173	6.719775911	192.168.1.59	155.185.2.122	TLSv1.2	1506 Application Data, Application Data, Application Data, Application Data, Appl
174	6.719781131	192.168.1.59	155.185.2.122	TLSv1.2	353 Application Data, Application Data, Application Data, Application Data, Appl
175	6.719943318	155.185.2.122	192.168.1.59	TCP	60 443 → 37698 [ACK] Seq=25778 Ack=1769 Win=262656 Len=0
176	6.720650605	155.185.2.122	192.168.1.59	TCP	60 443 → 37698 [ACK] Seq=25778 Ack=1909 Win=262400 Len=0
177	6.721373914	192.168.1.59	155.185.2.122	TLSv1.2	181 Application Data
178	6.721423167	192.168.1.59	155.185.2.122	TLSv1.2	96 Application Data
179	6.723012774	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=25778 Ack=2027 Win=262400 Len=1452 [TCP segment of a r
180	6.723245133	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=27230 Ack=2027 Win=262400 Len=1452 [TCP segment of a r
181	6.723258979	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=28682 Win=64128 Len=0
182	6.723454990	155.185.2.122	192.168.1.59	TCP	2958 443 → 37698 [ACK] Seq=28682 Ack=2027 Win=262400 Len=2904 [TCP segment of a r
183	6.723487201	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=31586 Win=64128 Len=0
184	6.723742754	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=31586 Ack=2027 Win=262400 Len=1452 [TCP segment of a r
185	6.723949165	155.185.2.122	192.168.1.59	TLSv1.2	2813 Application Data
186	6.723949245	155.185.2.122	192.168.1.59	TLSv1.2	92 Application Data
187	6.723975675	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=35835 Win=64128 Len=0
188	6.724247730	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=35835 Ack=2176 Win=262144 Len=1452 [TCP segment of a r
189	6.724482233	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=37287 Ack=2176 Win=262144 Len=1452 [TCP segment of a r
190	6.724497512	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=38739 Win=64128 Len=0
191	6.724691659	155.185.2.122	192.168.1.59	TCP	2958 443 → 37698 [ACK] Seq=38739 Ack=2176 Win=262144 Len=2904 [TCP segment of a r
192	6.724716005	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=41643 Win=64128 Len=0
193	6.724887239	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=41643 Ack=2176 Win=262144 Len=1452 [TCP segment of a r
194	6.725016434	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=43095 Ack=2176 Win=262144 Len=1452 [TCP segment of a r
195	6.725025962	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=44547 Win=64128 Len=0
196	6.725196524	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=44547 Ack=2176 Win=262144 Len=1452 [TCP segment of a r
197	6.725484289	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=45999 Ack=2176 Win=262144 Len=1452 [TCP segment of a r
198	6.725498395	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=47451 Win=64128 Len=0
199	6.725692102	155.185.2.122	192.168.1.59	TCP	2958 443 → 37698 [ACK] Seq=47451 Ack=2176 Win=262144 Len=2904 [TCP segment of a r
200	6.725716217	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=50355 Win=64128 Len=0
201	6.725886950	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=50355 Ack=2176 Win=262144 Len=1452 [TCP segment of a r
202	6.725942195	155.185.2.122	192.168.1.59	TLSv1.2	603 Application Data, Application Data
203	6.725951563	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=52356 Win=64128 Len=0

Figure 1: Traffico TLS prima dell'applicazione della sslkeylog

Come si può notare dalla Figura 1 molte delle comunicazioni catturate da Wireshark mostrano il protocollo `TLSv1.2` e non rivelano il contenuto del pacchetto.

Ora da Wireshark:

Edit → Preference... → Protocols → TLS e alla voce `(Pre)-Master-Secret log filename` cercate il file `sslkeylog` ed importatelo.

154	6.645560884	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=21620 Ack=1387 Win=261376 Len=1452 [TCP segment of a reassemb
155	6.645661739	155.185.2.122	192.168.1.59	HTTP2	689 HEADERS[15]: 200 OK, DATA[15]
156	6.645669964	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=1387 Ack=23707 Win=64128 Len=0
157	6.645719428	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=23707 Ack=1387 Win=261376 Len=1452 [TCP segment of a reassemb
158	6.645911962	155.185.2.122	192.168.1.59	HTTP2	673 DATA[15], DATA[15] (text/html)
159	6.645923925	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=1387 Ack=25778 Win=64128 Len=0
160	6.688403899	192.168.1.59	8.8.8.8	DNS	95 Standard query 0xbb12 A www.googletagmanager.com OPT
161	6.693615657	192.168.1.59	155.185.2.122	HTTP2	248 HEADERS[17]: GET /_css/layout.css?v=25082022, WINDOW_UPDATE[17]
162	6.693877632	192.168.1.59	155.185.2.122	HTTP2	96 RST_STREAM[17]
163	6.694087570	192.168.1.59	155.185.2.122	HTTP2	158 HEADERS[19]: GET /_js/scriptvarii.min.js, WINDOW_UPDATE[19]
164	6.694645966	192.168.1.59	155.185.2.122	HTTP2	96 RST_STREAM[19]
165	6.694814264	192.168.1.59	155.185.2.122	HTTP2	152 HEADERS[21]: GET /_js/responsive-comments.min.js, WINDOW_UPDATE[21]
166	6.695153436	192.168.1.59	155.185.2.122	HTTP2	96 RST_STREAM[21]
167	6.696486207	192.168.1.59	155.185.2.122	HTTP2	172 HEADERS[23]: GET /_img/Sigillo2015.svg, WINDOW_UPDATE[23]
168	6.697054522	192.168.1.59	155.185.2.122	HTTP2	161 HEADERS[25]: GET /PPimg/820023507_820000361_i4.0400x300.jpeg, WINDOW_UPDATE[25]
169	6.697349940	192.168.1.59	155.185.2.122	HTTP2	96 RST_STREAM[23]
170	6.699107094	192.168.1.59	155.185.2.122	HTTP2	161 HEADERS[27]: GET /PPimg/820022273_820000361_lgbt400x300.jpeg, WINDOW_UPDATE[27]
171	6.715696945	8.8.8.8	192.168.1.59	DNS	111 Standard query response 0xbb12 A www.googletagmanager.com A 142.251.209.8 OPT
172	6.719732900	155.185.2.122	192.168.1.59	TCP	60 443 → 37698 [ACK] Seq=25778 Ack=1623 Win=262656 Len=0
173	6.719775911	192.168.1.59	155.185.2.122	HTTP2	1506 HEADERS[49]: GET /raccoltavideo/media/intervista-cabri-maggio2023.jpg, WINDOW_UPDAT
174	6.719781131	192.168.1.59	155.185.2.122	HTTP2	353 RST_STREAM[49], RST_STREAM[53]
175	6.719943318	155.185.2.122	192.168.1.59	TCP	60 443 → 37698 [ACK] Seq=25778 Ack=1769 Win=262656 Len=0
176	6.720650605	155.185.2.122	192.168.1.59	TCP	60 443 → 37698 [ACK] Seq=25778 Ack=1909 Win=262400 Len=0
177	6.721373914	192.168.1.59	155.185.2.122	HTTP2	181 HEADERS[55]: GET /_img/halftone.png, WINDOW_UPDATE[55]
178	6.721423167	192.168.1.59	155.185.2.122	HTTP2	96 RST_STREAM[55]
179	6.723012774	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=25778 Ack=2027 Win=262400 Len=1452 [TCP segment of a reassemb
180	6.723245133	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=27230 Ack=2027 Win=262400 Len=1452 [TCP segment of a reassemb
181	6.723258979	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=28682 Win=64128 Len=0
182	6.723454990	155.185.2.122	192.168.1.59	TCP	2958 443 → 37698 [ACK] Seq=28682 Ack=2027 Win=262400 Len=2904 [TCP segment of a reassemb
183	6.723487201	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=31586 Win=64128 Len=0
184	6.723742754	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=31586 Ack=2027 Win=262400 Len=1452 [TCP segment of a reassemb
185	6.723949165	155.185.2.122	192.168.1.59	HTTP2	2813 HEADERS[23]: 200 OK, DATA[23]
186	6.723949245	155.185.2.122	192.168.1.59	HTTP2/…	92 DATA[23]
187	6.723975675	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=35835 Win=64128 Len=0
188	6.724247730	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=35835 Ack=2176 Win=262144 Len=1452 [TCP segment of a reassemb
189	6.724482233	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=37287 Ack=2176 Win=262144 Len=1452 [TCP segment of a reassemb
190	6.724497512	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=38739 Win=64128 Len=0
191	6.724691659	155.185.2.122	192.168.1.59	TCP	2958 443 → 37698 [ACK] Seq=38739 Ack=2176 Win=262144 Len=2904 [TCP segment of a reassemb
192	6.724716005	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=41643 Win=64128 Len=0
193	6.724887239	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=41643 Ack=2176 Win=262144 Len=1452 [TCP segment of a reassemb
194	6.725016434	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=43095 Ack=2176 Win=262144 Len=1452 [TCP segment of a reassemb
195	6.725025962	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=44547 Win=64128 Len=0
196	6.725196524	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=44547 Ack=2176 Win=262144 Len=1452 [TCP segment of a reassemb
197	6.725484289	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=45999 Ack=2176 Win=262144 Len=1452 [TCP segment of a reassemb
198	6.725498395	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=47451 Win=64128 Len=0
199	6.725692102	155.185.2.122	192.168.1.59	TCP	2958 443 → 37698 [ACK] Seq=47451 Ack=2176 Win=262144 Len=2904 [TCP segment of a reassemb
200	6.725716217	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=50355 Win=64128 Len=0
201	6.725886950	155.185.2.122	192.168.1.59	TCP	1506 443 → 37698 [ACK] Seq=50355 Ack=2176 Win=262144 Len=1452 [TCP segment of a reassemb
202	6.725942195	155.185.2.122	192.168.1.59	HTTP2	603 HEADERS[25]: 200 OK, DATA[25]
203	6.725951563	192.168.1.59	155.185.2.122	TCP	54 37698 → 443 [ACK] Seq=4203 Ack=52356 Win=64128 Len=0

Figure 2: Traffico TLS dopo l'applicazione della sslkeylog

Ora invece lo stesso traffico riportato nella Figura 2 è evidenziato in verde e il protocollo è `HTTP2`, si è quindi riusciti a decifrare tutte le comunicazioni verso `www.unimore.it`