CDL. MAGISTRALE INFORMATICA (POLITECNICO DI BARI)

Control methods for computer networks

Relazione tema d'anno: Digital Right Management (DRM)

Prof. Ing. S. Mascolo
Ing. V. Palmisano

Studenti: G.Allegretta E.Iacobellis F.Scarangella V.Specchiulli

Anno 2016/2017

INTRODUZIONE

SEZIONE I

SEZIONE II

RIFERIMENTI

Introduzione

Il presente lavoro ha l'obiettivo di gestire la creazione e la riproduzione di contenuto multimediale protetto da DRM (*Digital Right Management*), simulando il funzionamento di alcuni servizi di distribuzione multimediale a pagamento. Per raggiungere tale fine si è deciso di utilizzare lo standard Common Encryption e di usufruire di un server di test (usando il sistema ClearKey) implementato all'interno della specifica EME (*Encrypted Media Extension*).

Il progetto consiste in due fasi:

1) Studio teorico riguardante i seguenti temi
    - DRM (*Digital Right Management*)
    - EME (*Encrypted Media Extension*)
    - *Common Encryption*
    - *ClearKey*

2) Attività pratica
    - Transcodifica/Transrating di un file multimediale in formato H.264/AAC
    - Generazione del cryptfile con MP4Box e crittografia del file multimediale
    - Segmentazione e Packaging in DASH del contenuto crittografato
    - Realizzazione di un piccolo sito per la visualizzazione del video

Sezione I

I.1 DRM (Digital Right Management)

Con Digital Rights Management (DRM), il cui significato letterale è "gestione dei diritti digitali", si intendono i sistemi tecnologici mediante i quali chi detiene i diritti d'autore può farli valere nell'ambiente digitale. Questi sistemi sono fortemente voluti dalle varie case di produzione (discografiche, cinematografiche, ecc), ma sono altrettanto osteggiati dagli utenti, in quanto impone numerose limitazioni ai loro diritti, tanto da "coniare" il termine Digital Restriction Management.

I.1.1 Perché usare i DRM?

Con l'avvento delle tecnologie digitali, copiare un file multimediale (audio o video) è diventato semplice e non comporta, a differenza dei supporti analogici, una diminuzione della qualità. Grazie alla diffusione di strumenti digitali per l'accesso a contenuti multimediali (PC, telefonini di nuova generazione, ecc) e alla diffusione dell'accesso a Internet a banda larga e delle reti peer to peer, l'accesso e la distribuzione in tutto il mondo di contenuti multimediali è alla portata di ogni singolo utente, creando nuovi scenari capaci di modificare il consolidato sistema autore-distributore-cliente. Lo studio di soluzioni DRM nasce dal tentativo di poter gestire il controllo sugli aspetti legati alla distribuzione e all'utilizzo. Le prime forme di DRM, avute negli anni '80, venivano applicate essenzialmente sui contenuti artistici e letterali. Il boom dell'utilizzo dell'uso dei DRM si è avuto con la diffusione dell'audio e del video digitale. Il primo vero esempio di DRM digitale si è avuto nel 1996 con il sistema CSS (Content Scrambling System). Il CSS è un semplice algoritmo di cifratura utilizzato dalle case cinematografiche per la diffusione dei propri film con l'inclusione di restrizioni di riproduzione e copia.

I.1.2 Funzionamento

I file audio/video vengono codificati e criptati in modo da garantire la protezione contro la copia e la diffusione non autorizzata, consentendo un uso limitato (nel tempo o nell'utilizzo) e predefinito dalla licenza fornita.

La codifica permette di includere informazioni aggiuntive, quali copyright, licenza, biografia dell'autore, ecc. L'accesso ai dati è consentito tramite procedure di autenticazione, che permettono di distribuire i file richiesti secondo le modalità previste dalla licenza sottoscritta.

Il controllo viene eseguito sull'hardware usato. Questo significa che, se un certo file protetto da DRM è comprato da un utente per il dispositivo X, quello stesso utente non potrà visualizzare/riprodurre lo stesso file su un dispositivo Y.

I.1.3 Esempi di DRM

Chiave di registrazione (seriali)

Si basa sull'attivazione tramite codice seriale, fornito in dotazione con il prodotto o tramite registrazione in rete.

Attivazione software

L'utente che acquista il prodotto originale riceve un codice alfanumerico di 25 caratteri, che viene inizialmente verificato dallo stesso software. Entro 15 giorni deve eseguire una verifica online, comunicando un codice numerico generato in base alla Product Key e alla configurazione hardware del dispositivo in uso.

CSS (*Content Scrambling System*)

Questo sistema è stato ideato per la protezione dei DVD. I supporti vengono crittografati con una chiave segreta fornita ai produttori di hardware e software di lettura, previa stipulazione di specifiche condizioni di licenza e pagamento di una quota. Qualunque supporto di lettura che non abbia le caratteristiche necessarie alla decodifica CSS non sarà in grado di riprodurre questo tipo di DVD.

Watermarking

Tecnica basata sull'acquisizione di informazioni dell'utente. In caso di violazione di copyright, da queste informazioni e dall'ID del prodotto venduto si può risalire al responsabile per avviare procedure penali.

Rootkit Sony BRG

Quando erano inseriti nel computer, i CD installavano un rootkit (software malevolo), che inseriva una sorta di DRM nel sistema operativo, modificandolo. Questo software si attivava durante la copia del CD, inserendo del rumore bianco all'interno delle tracce. Il programma non poteva essere disinstallato con facilità e, inoltre, creava delle falle che sono state sfruttate da vari malware.

I.2 Encrypted Media Extension

EME (*Encrypted Media Extensions*) è un'estensione di HTMLMediaElement che permette la riproduzione di contenuti protetti sui browser che la supportano, evitando l'installazione di ulteriori plugin.

I.2.1 Definizioni

CDM (*Content Decryption Module)*

Componente del client che fornisce diverse funzionalità, tra cui la decrittazione e la riproduzione di file protetti, per uno o più Key System.

Key System

Termine generico per indicare i meccanismi di decrittazione e/o i servizi di protezione dei contenuti. Sono usate per selezionare uno dei CDM e identificare l'evento correlato. La stringa associata al Key System è sempre l'inverso del nome di dominio.

License key server

Server che fornisce le chiavi per decriptare i media, la cui negoziazione è delegata all'applicazione.

I.2.2 Oggetti e metodi principali

navigator.requestMediaKeySystemAccess(keySystem, SupportedConfiguration)

Verifica che il KeySystem specificato sia disponibile e che la configurazione sia corretta. In caso positivo restituisce l'oggetto MediaKeySystemAccess

MediaKeySystemAccess

Oggetto che fornisce l'accesso al Key System

MediaKeys

Oggetto che contiene un set di chiavi usate per decriptare i file durante la riproduzione. Viene creato dal metodo *createMediaKeys()*

MediaKeys.createSession(SessionType)

Metodo che crea un oggetto di tipo MediaKeySession partendo dal tipo di sessione

MediaKeySession

Oggetto che contiene informazioni sulla durata della licenza e delle relative chiavi

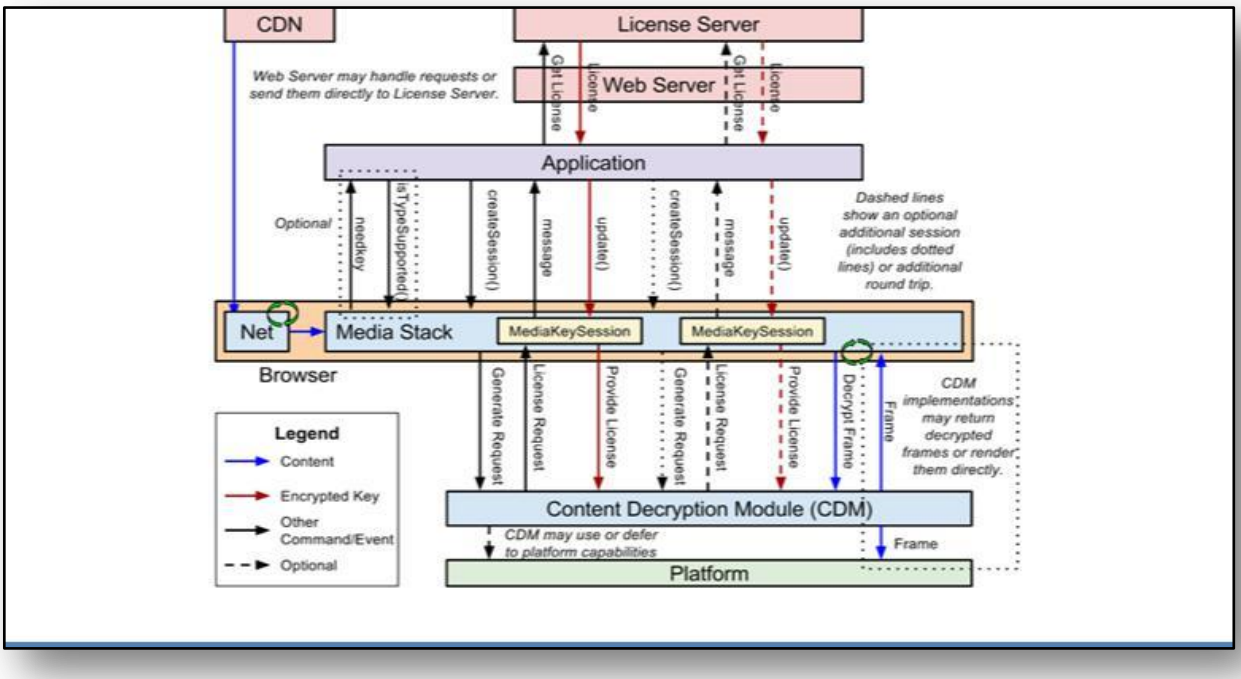
MediaKeySession.generateRequest()

Metodo che genera una richiesta al server sulla base dei dati passati

MediaKeySession.update(response)

Metodo che fornisce messaggi, tra cui la licenza, al CDM

I.2.3 Procedura per ottenere la chiave dal server di licenza



È possibile vedere come il browser si interfacci con l'applicazione e il CDM.

Tramite l'applicazione, il browser comunica con il server di licenza, scambiando messaggi relativi alla sessione.

Inoltre, dialoga con il CDM per formulare la richiesta della licenza e ha uno scambio diretto per la decrittazione e la visualizzazione dei contenuti.

Nel nostro progetto le funzioni di EME sono supportate dalla libreria Shaka-Player e sono contenute all'interno del file drm_engine.js.

I.3 Common Encryption (CENC)

La Common Encryption è una normativa ISO che definisce un formato comune per la crittografia, la decrittazione e la mappatura di chiavi. Pertanto la riproduzione di file multimediali protetti è possibile anche utilizzando DRM differenti.

Inoltre offre la possibilità di crittografare un contenuto multimediale specificando una o più chiavi, separatamente per i canali audio e video. L'utilizzo di chiavi differenti garantisce un grado di sicurezza maggiore.

Il vantaggio del metodo Common Encryption consiste, quindi, nel facilitare la distribuzione di uno stesso contenuto protetto verso numerose piattaforme e dispositivi di riproduzione, riducendo i costi e la complessità del flusso di distribuzione.

Poiché la Common Encryption standardizza solo crittografia e decrittazione, non viene utilizzata per gestire le altre attività che riguardano i DRM. Pertanto i dettagli relativi alla distribuzione delle licenze, al rispetto delle autorizzazioni e alle regole di conformità tra DRM sono gestiti e controllati dai sistemi DRM stessi o da altri sistemi che supportano il metodo CENC.

I.3.1 Sistemi DRM che supportano la Common Encryption

Microsoft PlayReady: impiegato in molti servizi di Microsoft

Google Widevine: utilizzato nelle applicazioni di Google

FairPlay Streaming: adottato dai servizi Apple

CMLA-OMA: usato per abilitare il DRM su dispositivi mobili Android che non supportano Widevine. É usato anche per riprodurre flussi protetti MPEG-DASH su dispositivi iOS che non supportano FairPlay Streaming.

Adobe Primetime (Access): usato per la riproduzione di video in Adobe Flash Player

In base a quanto affermato finora, un video crittografato in maniera conforme allo schema CENC può essere decifrato dalla maggior parte dei browser.

I.4 Clearkey

Sebbene EME non definisca nessuna funzionalità per i DRM, la specifica attualmente impone che tutti i browser che la supportano debbano implementare il metodo Clearkey. Con questo sistema, il file multimediale può essere crittografato con una chiave e riprodotto semplicemente fornendo quella chiave in chiaro. Clearkey può essere integrato nel browser: non richiede l'uso di un modulo separato di decrittografia ma la chiave viene passata e letta direttamente dal browser attraverso un’applicazione Javascript.

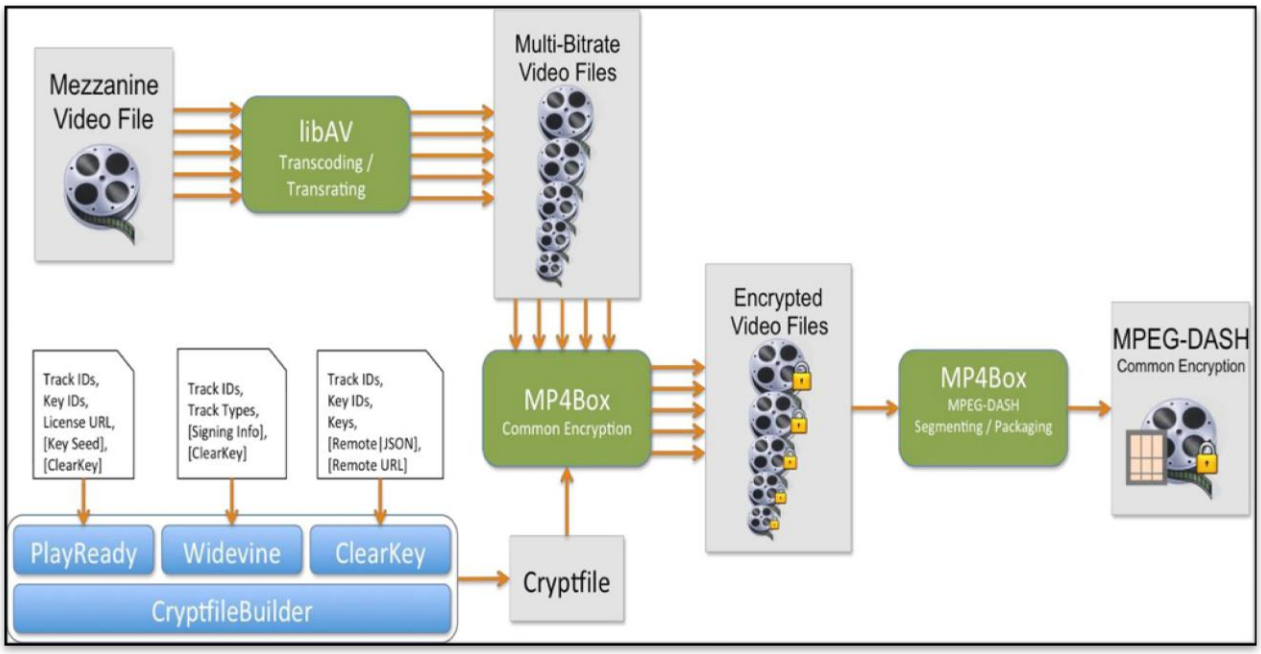
Clearkey supporta molti tipi di contenuti commerciali ed è completamente interoperabile su tutti i browser che supportano EME. Risulta inoltre utile per testare le implementazioni EME e applicazioni che lo utilizzano, senza la necessità di richiedere una chiave del contenuto da un server di licenza.

I.4.1 Clearkey License Server

Nel tipico uso commerciale, un determinato contenuto viene crittografato, codificato e segmentato con uno strumento apposito. Quando la risorsa viene resa disponibile online, un client web che voglia visualizzare la risorsa deve richiedere la chiave da un server di licenza. Una volta ottenuta la utilizza per de crittografare e riprodurre il contenuto.

A differenza di PlayReady di Microsoft, dove la chiave generata dal server di licenza è differente dalla vera chiave di cifratura della risorsa (la chiave generata è una chiave hardware e viene ottenuta a seguito di un algoritmo proprietario), nel server di licenza Clearkey la chiave inviata è la stessa chiave utilizzata per decriptare il file multimediale.

Sezione II



II.1 Transcodifica/transrating di un file multimediale in formato H.264/AAC

Il nostro lavoro ha inizio da un video di alta qualità “COSTARICA” (video AVC/H.264 e audio AAC).

Come si vede dallo schema precedente, il primo passaggio consiste nel trancoding/transrating, ossia nella generazione del video di partenza in varie risoluzioni e bitrate. Per fare questo abbiamo utilizzato, tra le varie proposte open source, il programma **FFMpeg.**

FFmpeg è il framework multimediale leader in grado di decodificare, codificare, transcodificare, muxare e demuxare i flussi, filtrare e riprodurre praticamente qualsiasi cosa che si possa creare. Supporta vari tipi di formati, dai più obsoleti ai più recenti. Un aspetto importante, inoltre, è l’estrema portabilità: FFmpeg è utilizzabile sui diversi sistemi operativi (oltre che su Linux, anche su Windows e Mac OS X).

Contiene le librerie libavcodec, libavutil, libavformat, libavfilter, libavdevice, libswscale e libswresample che possono essere utilizzate dalle applicazioni. Così come ffmpeg, ffserver, ffplay e ffprobe possono essere utilizzate dagli utenti finali per la transcodifica, lo streaming e la riproduzione.

I comandi da terminale da eseguire per installare ffmpeg sono i seguenti:

sudo apt-get install build-essential git yasm

sudo apt-get install libx264-dev libfdk-aac-dev

git clone git://git.ffmpeg.org/ffmpeg.git

cd ffmpeg

./configure --enable-nonfree --enable-gpl --enable-libx264 --enable-libfdk-aac make

sudo make install

II.1.1 Comando ffmpeg

ffmpeg -i <input_file> video_options audio_options <output_file>

I campi che lo caratterizzano sono i seguenti:

- *-i <input_file>* specifica la sorgente su cui effettuare la codifica
  video_options indica le caratteristiche che deve avere il video in uscita (codec, profile, level, bitrate)
- *audio_options* indica le caratteristiche che deve avere l'audio in uscita (codec, profile)
- *<output_file>* specifica il percorso e il nome del file di uscita

Di seguito, un esempio di utilizzo del comando appena descritto:

```
$ ffmpeg -i input_720p.mov
-codec:v libx264 -profile:v high -level 31 -b:v 2000k -codec:a libfdk_aac -profile:a aac_low abr_720p_h264-2Mb-high-3.1_aac-lc.mp4

$ ffmpeg -i input720p.mov
-codec:v libx264 -profile:v high -level 31 -b:v 3000k -codec:a libfdk_aac -profile:a aac_low
abr_720p_h264-3Mb-high-3.1_aac-lc.mp4
```

Alcune opzioni vengono applicate al flusso, ad esempio il bitrate o il codec. Degli identificatori di flusso vengono utilizzati per specificare con precisione a quali Stream appartiene una data opzione. Essi figurano come stringhe in genere aggiunti al nome dell'opzione e separati da esso da due punti.

-codec: seleziona un codificatore per uno o più flussi
-b: imposta il bitrate del file di output
-profile: imposta il profilo tra “high”, “main” e “baseline”
-level: parametro definito dallo standard H.264

Utilizzando più comandi ffmpeg è possibile generare flussi molteplici, ciascuno caratterizzato da un proprio valore di bitrate e da una propria risoluzione.

L'obiettivo del nostro progetto è creare un contenuto compatibile con il profilo DASH-AVC/264. Pertanto, è necessario separare i componenti audio e video in due canali differenti e assicurarci che la durata dei segmenti non vari più del 50% della durata stabilita. Tali operazioni vengono svolte all'interno dello script transcoding.sh lanciato dal terminale.

II.1.2 Script transcoding.sh

./transcoding.sh COSTARICA ./transcoded

Questo script fa chiamate distinte a ffmpeg per ogni bitrate e per ogni componente audio o video in modo da ottenere l'output demultiplexato che richiediamo.

L'analizzatore "ffprobe" è utilizzato per rilevare il framerate del supporto di origine mentre le opzioni "keyint", "min-keyint" di x264 dettano l'intervallo dei fotogrammi chiave. L'argomento "noscenecut" assicura che ffmpeg non inserisca fotogrammi arbitrari quando rileva un cambiamento di scena visiva nel contenuto d'origine.

```
ffmpeg=ffmpeg
ffprobe=ffprobe
function usage {
  echo ""
  echo "Transcode/Transrate Script"
  echo "usage:"
  echo "   video_scaled_demux_5bitrates  "
}
if [ -z $1 ]; then
  echo "Must provide input media file"
  usage
  exit 1
fi
if [ -z $2 ]; then
  echo "Must provide output directory for transcoded/transrated files"
  usage
  exit 1
fi
mkdir -p $2
framerate=$((`./ffprobe $1 -select_streams v -show_entries stream=avg_frame_rate -v quiet -of csv="p=0"`))
function transcode_video {
  vbitrate=$1
  res=$2
  profile=$3
  level=$4
  outdir=$5
  infile=$6
  outfile=video_${infile}_`echo $res | sed 's/:/x/'`_h264-${vbitrate}.mp4
$ffmpeg -i $infile -s $res -map_chapters -1 -maxrate $vbitrate -minrate $vbitrate -bufsize $vbitrate -an -codec:v
libx264 -profile:v $profile -level $level -b:v $vbitrate -x264opts "keyint=$framerate:min-keyint=$framerate:no-
scenecut" $outdir/$outfile
}
```

```
function transcode_audio {
  abitrate=$1
  outdir=$2
  infile=$3
  outfile=audio_${infile}_aac-lc_${abitrate}.mp4

  $ffmpeg -i $infile -map_chapters -1 -vn -codec:a libfdk_aac -profile:a aac_low -b:a $abitrate $outdir/$outfile
}

transcode_video "360k" "512:288" "main" 30 $2 $1
transcode_video "620k" "704:396" "main" 30 $2 $1
transcode_video "1340k" "896:504" "high" 31 $2 $1
transcode_video "2500k" "1280:720" "high" 32 $2 $1
transcode_video "4500k" "1920:1080" "high" 40 $2 $1

transcode_audio "128k" $2 $1
transcode_audio "192k" $2 $1
```

II.2 Crittografia del file multimediale

Il secondo passaggio consiste nella crittazione dei file ottenuti dalla fase precedente. Per fare questo abbiamo utilizzato il tool di GPAC chiamato MP4Box.

MP4Box è un pacchetto multimediale, utilizzabile da riga di comando, che supporta un vasto numero di funzionalità: conversione, frammentazione, segmentazione di file audio e video, ecc...

Può essere anche utilizzato per crittare o decrittare stream in diversi formati e utilizza il linguaggio XML per definire i parametri di crittazione.

La funzione utilizzata per criptare è la seguente:

MP4Box –crypt <criptfile> -out <output file> <input file>

II.2.1 Cryptfile

<crypt file> è un file XML che contiene tutte le informazioni riguardo il DRM e il metodo di crittazione per una determinata traccia, la sintassi tipica è:

```
<GPACDRM
 type="...">
 <DRMInfo >
   …
 </DRMInfo>
 <CrypTrack trackID="..." IsEncrypted="..." IV_size="..." first_IV="..." selectiveType="..."
saiSavedBox="..." keyRoll="...">
 <key KID="..." value="..."/>
 <key KID="..." value="..."/>
  </CrypTrack>
</GPACDRM>
```

- **Type:** descrive lo schema di crittazione da utilizzare, nel nostro caso è CENC AES-CTR, uno schema che utilizza la crittografia AES 128 bit in Counter-Mode. Altri possibili valori da assegnare a questo campo sono:
    - ISMA o iAEC : ISMA E&A (ISMACryp) Scheme
    - Adobe o adkm: Adobe Scheme
    - CENC AES-CBC Pattern o cbcs
- **DRMInfo:** contiene informazioni riguardanti il DRM utilizzato, è possibile aggiungere un altro tag DRM e specificarne più di uno
- **TrackID:** specifica l'ID della traccia da criptare. Si utilizza il valore 1 per le tracce video, il valore 2 per le tracce audio. E' un campo obbligatorio.
- **IsEncrypted:** è un flag che indica se i campioni all'interno della traccia sono stati già criptati o meno
- **IV_size:** è la lunghezza in byte dell'InizializationVector, può essere 0 (se il file non è già criptato), 8 o 16
- **First_IV:** è il primo InizializationVector della traccia
- **SelectiveType:** specifica di criptare solo determinati campioni selezionati, può assumere diversi valori, ossia i seguenti
    - None -> tutti i campioni sono criptati
    - Clear -> nessun campione viene criptato, ma il file è segnato come criptato
    - RAP -> saranno criptati solo i campioni segnati come Random Access Samples (key frame)
    - NORAP -> saranno criptati tutti i campioni tranne quelli segnati come Random Access Samples (no key frame)
    - Rand -> saranno criptati campioni selezionati casualmente
    - X -> cripta un campione ogni X campioni (deve essere un intero maggiore di due)
    - RandX -> cripta un campione selezionato casualmente ogni X campioni (deve essere un intero maggiore di due)
- **keyRoll:** metodo di selezione delle chiavi da utilizzare. Può assumere i seguenti valori:
    - IDX=N utilizza solo le chiavi con indice N per la traccia specificata
    - Roll=N permette di criptare i vari campioni della stessa traccia utilizzando chiavi differenti. Ogni N campioni viene utilizzata una chiave diversa
- **Key:** la chiave AES 128 bit. Deve essere specificato l'ID della chiave e il suo valore, è un campo obbligatorio, può essere specificata più di una chiave

2.2.2 MP4Box Cryptfile Generator

Per ottenere il cryptfile abbiamo utilizzato il programma in Java MP4Box_CryptFile_Generator, fornito da CableLabs, che permette di creare un file XML da utilizzare per la criptazione con il KeySystem Clearkey, passando come parametri l'ID delle chiavi e il loro valore.

Digitando il comando *java –jar MP4Box_CryptFile_Generator.jar -help* è possibile visualizzare la guida del programma.

usage: **CryptfileGen** [OPTIONS] <**track_id**>:{**@<key_file>|<key_id>**=[<key>][,<key_id>=[<key>]...]} [<track_id>:{@<key_file>|<key_id>=[<key>][,<key_id>=[<key>]...]}]...

<**track_id**> is the track ID from the MP4 file to be encrypted. After the '<track_id>:', you can specify either a file containing key/keyID pairs OR a comma-separated list of keyID/key pairs separated by ':'. Key IDs are always represented in GUID form (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx). Key values are always in hexadecimal. Multiple key IDs indicate the use of rolling keys.

<**keyid_file**> is a file that contains a list of key IDs, one pair per line in GUID form.

<**keyid**> is a key ID in GUID form.

<**key**> is a 16-byte key value in hexadecimal (with or without the leading '0x'). If key is ommitted, a random key will be generated for you.

OPTIONS:

-**help** Display this usage message.

-**out** <filename> If present, the cryptfile will be written to the given file. Otherwise output will be written to stdout.

-**roll** <sample_count> Used for rolling keys only. <sample_count> is the number of consecutive samples to be encrypted with each key before moving to the next.

-**cp** Print a DASH <ContentProtection> element that can be pasted into the MPD

Abbiamo scelto di utilizzare le seguenti chiavi:

A1B1C1D1-A2B2-A3B3-A4B4-A5B5C5D5E5F5:ccc0f2b3b279926496a7f5d25da692e9
D1C1B1A1-B2A2-B3A3-A4B4-A5B5C5D5E5F5:3a2a1b68dd2bd9b2eeb25e84c4776668

Quindi, digitando da terminale ubuntu, il comando:

java –jar MP4Box_cryptfile_Generator.jar –out drm_file.xml -roll 2 1:@chiavi 2:@chiavi

abbiamo ottenuto il seguente cryptfile:

```
Ensure the following keys are available to the client: a1b1c1d1a2b2a3b3a4b4a5b5c5d5e5f5 :
ccc0f2b3b279926496a7f5d25da692e9 (obHB0aKyo7OktKW1xdXl9Q : zMDys7J5kmSWp_XSXaaS6Q)
d1c1b1a1b2a2b3a3a4b4a5b5c5d5e5f5 : 3a2a1b68dd2bd9b2eeb25e84c4776668 (0cGxobKis6OktKW1xdXl9Q :
OiobaN0r2bLusl6ExHdmaA) a1b1c1d1a2b2a3b3a4b4a5b5c5d5e5f5 : ccc0f2b3b279926496a7f5d25da692e9
(obHB0aKyo7OktKW1xdXl9Q : zMDys7J5kmSWp_XSXaaS6Q) d1c1b1a1b2a2b3a3a4b4a5b5c5d5e5f5 :
3a2a1b68dd2bd9b2eeb25e84c4776668 (0cGxobKis6OktKW1xdXl9Q : OiobaN0r2bLusl6ExHdmaA)

<GPACDRM type="CENC AES-CTR"> <DRMInfo type="pssh" version="1"> <BS
ID128="1077efecc0b24d02ace33c1e52e2fb4b"/> <BS bits="32" value="2"/> <BS
ID128="a1b1c1d1a2b2a3b3a4b4a5b5c5d5e5f5"/> <BS ID128="d1c1b1a1b2a2b3a3a4b4a5b5c5d5e5f5"/>
</DRMInfo>

<CrypTrack IV_size="8" first_IV="0x5721b5e57022aa3c" isEncrypted="1" keyRoll="roll=2" saiSavedBox="senc"
trackID="1"> <key KID="0xa1b1c1d1a2b2a3b3a4b4a5b5c5d5e5f5"
value="0xccc0f2b3b279926496a7f5d25da692e9"/> <key KID="0xd1c1b1a1b2a2b3a3a4b4a5b5c5d5e5f5"
value="0x3a2a1b68dd2bd9b2eeb25e84c4776668"/> </CrypTrack>

<CrypTrack IV_size="8" first_IV="0xbd50d9b98bd51e89" isEncrypted="1" keyRoll="roll=2" saiSavedBox="senc"
trackID="2"> <key KID="0xa1b1c1d1a2b2a3b3a4b4a5b5c5d5e5f5"
value="0xccc0f2b3b279926496a7f5d25da692e9"/> <key KID="0xd1c1b1a1b2a2b3a3a4b4a5b5c5d5e5f5"
value="0x3a2a1b68dd2bd9b2eeb25e84c4776668"/> </CrypTrack>
```

Abbiamo inoltre utilizzato lo script encrypt.sh per criptare tutti i flussi audio e video nei diversi formati, salvandoli in una stessa cartella usando un unico comando:

./encrypted.sh –o encrypted –c drm_file.xml –i video

Il codice dello script encrypted.sh è il seguente:

```
while getopts ":o:c:i:" opt; do
 case $opt in
o) output_dir=$OPTARG
;;
c) cryptfile=$OPTARG
 ;;
i) input_dir=$OPTARG
 ;;
\?)
echo "Invalid option: -$OPTARG" >&2
  usage
  exit 1
;;
 :)
  echo "Missing options argument for -$OPTARG" >&2
  usage
  exit 1
 ;;
esac
done
```

```
shift $((OPTIND - 1))
if [ -z $cryptfile ]; then
  echo "Must provide cryptfile argument!"
  usage
  exit 1
 fi
if [ -z $output_dir ]; then
echo "Must provide output directory for encrypted media files!"
  usage
  exit 1
   fi
if [ -z $input_dir ] && [ ${#@} -eq 0 ]; then
  echo "No input media files specified!"
  usage
  exit 0
fi
mkdir -p $output_dir
# Encrypt all files in optional input directory
if [ ! -z $input_dir ]; then
  for file in `ls $input_dir`; do
      MP4Box -crypt $cryptfile $input_dir/$file -out $output_dir/`basename $file`
 done
 fi
# Encrypt individual files specified at the end of the command
for file in $@; do
    MP4Box -crypt $cryptfile $file -out $output_dir/`basename $file`
 done
```

II.3 Segmentazione e packaging DASH del contenuto crittografato

Il protocollo applicativo DASH (*Dynamic Adaptive Streaming over HTTP)* nasce con lo scopo di standardizzare il settore video streaming ponendo fine ad una serie di problematiche tra cui bassa qualità, mancanza di plugin e formati non disponibili.

Mediante la creazione di un file MPD (*Media Presentation Data model*) il protocollo DASH consente al server di trasmettere al client solo i frammenti richiesti, evitando il download dell'intero file. Allo stesso tempo, offre la possibilità di adottare la risoluzione più consona alla banda disponibile durante lo scaricamento del video, migliorando le prestazioni del download progressivo.

In base all'approccio formalizzato nel protocollo DASH, il video codificato e crittografato nei formati audio/video viene, in seguito, segmentato e indicizzato all'interno del file MPD, scegliendo un profilo opportuno (live, onDemand, sample, main). L'MPD contiene tutte le informazioni relative alla descrizione delle risorse (codec, DRM, risoluzione, banda, linguaggio,..) e gli URL che consentono di accedere a ogni segmento delle risorse presenti.

Ogni segmento ISOBMFF (*ISO Base Media File Format*) ottenuto è costituito dalla successione di due elementi i: **moof** (*Movie Fragment Box*) e **metadat** (contenente i campioni audio e video).

L'insieme dei segmenti e l'header - a sua volta costituito dai blocchi **ftyp** *(File Type Box*), contenente le informazioni di base come il tipo di formato audio o video, e **moov** (*Movie Box*), contenente informazioni sulla codifica - costituiscono il file MP4, un container usato nel web per raggruppare audio e video all'interno di un unico oggetto.

Il file MPD, pertanto, tiene traccia delle posizioni in cui sono memorizzati i moduli ftyp e moov, necessari per il download del video.

Per eseguire l'operazione di segmentazione si è utilizzato, anche in questo caso, lo strumento MP4Box fornito da GPAC.

Il comando MP4Box generico è il seguente:

MP4Box -dash <seg_duration> -profile <profile> -out <mpdfile><inputfiles>

I campi che lo caratterizzano sono

- **seg_duration:** specifica la durata di ciascun segmento espressa in millisecondi
- **profile:** indica il profilo MPEG-DASH utilizzato per generare il manifest MPD. Tra i profili disponibili sono stati utilizzati onDemand e live. In particolare, **onDemand** genera un singolo segmento per ogni rappresentazione video e audio, garantendo il supporto delle librerie VOd, minima gestione dei contenuti, scalabilità ed efficienza nell'utilizzo di server http; di contro, **live** genera segmenti multipli e un segmento di inizializzazione per ciascuna rappresentazione audio e video. Ogni frammento può essere richiesto utilizzando il modello URL creato, senza la necessità di aggiornare l'MPD prima di ogni richiesta di segmento
- **mpdfile:** specifica il nome del file MPD DASH
- **inputfiles:** indica i file multimediali di input che devono essere analizzati e segmentati per produrre il nuovo file di output

Per eseguire la segmentazione e il packaging del video "COSTARICA", precedentemente demultiplexato e crittografato, abbiamo lanciato lo script dash.sh da riga di comando, usando MP4Box:

./dash.sh -o ./packaged/live -p live -m live.mpd ./encrypted/audio_COSTARICA_aac-lc_128k.mp4./encrypted/audio_COSTARICA_aac-lc_192k.mp4./encrypted/video_COSTARICA_512x288_h264-360k.mp4./encrypted/video_COSTARICA_704x396_h264-620k.mp4./encrypted/video_COSTARICA_896x504_h264-1340k.mp4./encrypted/video_COSTARICA_1280x720_h264-2500k.mp4 ./encrypted/video_COSTARICA_1920x1080_h264-4500k.mp4

II.3.1 Script dash.sh

Nella prima parte dello script (omessa nel codice inserito in coda alla spiegazione) è implementata una funzione che stampa in output la descrizione del comando MP4Box -dash:

dash.sh -o <output_dir> -p <onDemand|live> [-m <manifest_file> [input_media_desc]]

-o è il flag che precede il nome della cartella di output **<output_dir>** che ospiterà il manifest (live.mpd o onDemand.mpd) e i segmenti generati
-p precede il profilo dash scelto (**onDemand** o **live**)
-m è il flag opzionale che precede il nome del manifest specificato dal campo **<manifest_file>**
input_media_desc è un campo opzionale usato per fornire una descrizione dei file multimediali in input

Successivamente l'assegnazione

mpd_file="dash.mpd"

attribuisce l'identificativo di default "dash.mpd" al file MPD.

Vengono, inoltre, eseguiti i controlli sui parametri di input. Se tutti i parametri sono stati inseriti correttamente, il comando **mkdir** crea la cartella di output in cui verranno inseriti i segmenti e il file MPD. Il comando MP4Box esegue infine la segmentazione e il packaging:

MP4Box -dash 2000 -rap -bs-switching no -sample-groups-traf \ -profile $profile –out $output_dir/$mpd_file $@

dove

- **2000:** è la durata richiesta per ciascun segmento, corrispondente a due secondi
- **rap:** è il parametro usato per forzare l'inizio di ciascun segmento con un punto di accesso casuale (I-frame). Con questa opzione, la durata dei segmenti potrebbe non rispecchiare quella richiesta dal comando -dash, poiché i video codificati non sono modificati
- **bs-switching no:** disabilita il passaggio automatico, basato sul bitrate, tra le diverse rappresentazioni audio e video
- **sample-groups-traf:** memorizza la descrizione del gruppo dei campioni nei traf (*Track Fragment Box*), anziché nel moov (*Movie Box*)

```
mpd_file="dash.mpd"

while getopts ":o:m:p:" opt; do
  case $opt in
    o)
      output_dir=$OPTARG
      ;;
    m)
      mpd_file=$OPTARG
      ;;
    p)
      profile=$OPTARG
      ;;
    \?)
      echo "Invalid option: -$OPTARG" >&2
      usage
      exit 1
      ;;
    :)
      echo "Missing options argument for -$OPTARG" >&2
      usage
      exit 1
      ;;
  esac
done
shift $((OPTIND - 1))

if [ -z $output_dir ]; then
  echo "Must provide output directory for DASH manifest and media files"
  usage
  exit 1
fi
if [ -z $profile ]; then
  echo "Must provide DASH profile"
  usage
  exit 1
fi
if [ ${#@} -eq 0 ]; then
  echo "No input media files specified!"
  usage
  exit 0
fi
mkdir -p $output_dir

MP4Box -dash 2000 -rap -bs-switching no -sample-groups-traf \
  -profile $profile -out $output_dir/$mpd_file $@
```

II.3.2 File MPD live

Il file MPD ottenuto presenta una struttura XML costituita da un tag **<Period></Period>** che descrive il contenuto del file indicandone tempo di inizio e durata. Al suo interno sono presenti due blocchi **<Adaption Set></Adaption Set>** contenenti i flussi multimediali nelle differenti risoluzioni.

Il primo Adaption Set contiene due tag **<Representation></Representation>** riferiti alle due possibili risoluzioni del flusso audio. Al suo interno sono specificati i campi che descrivono le caratteristiche generali di ciascun frammento audio (identificativo, tipo mime, codec, tasso dicampionamento e banda occupata).

Analogamente, il secondo Adaption Set contiene cinque tag **<Representation> </Representation>** relativi ai cinque flussi video. Come per l'audio, fornisce informazioni riguardanti le caratteristiche di ciascun segmento video (identificativo, tipo mime, codec, risoluzione e banda).

Infine si osserva che in ciascun blocco **<Representation></Representation>** sono presenti due ulteriori tag:

- **<Content Protection></Content Protection>** contenente i dettagli relativi alla crittografia dei contenuti (schemeIdUri, value, default_KID)
- **<SegmentTemplate></SegmentTemplate>** con dentro i campi che descrivono il modello della segmentazione (timescale, contenuto multimediale, startNumber, durata, file di inizializzazione)

```
<?xml version="1.0"?>
<!-- MPD file Generated with GPAC version 0.5.2-DEV-revVersion: 0.5.2-426-gc5ad4e4+dfsg5-1build1 at 2017-02-10T19:03:21.591Z-->

<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500S"
type="static" mediaPresentationDuration="PT0H5M8.523S"
maxSegmentDuration="PT0H0M1.996S" profiles="urn:mpeg:dash:profile:isoff-live:2011"
xmlns:cenc="urn:mpeg:cenc:2013">

 <ProgramInformation
 moreInformationURL="http://gpac.sourceforge.net">
 <Title>./packaged/live/live.mpd generated by GPAC</Title>
 </ProgramInformation>

 <Period duration="PT0H5M8.523S">
 <AdaptationSet segmentAlignment="true" lang="und">

 <Representation id="1" mimeType="audio/mp4"
 codecs="mp4a.40.2" audioSamplingRate="44100" startWithSAP="1"
 bandwidth="137552">
```

```
    <AudioChannelConfiguration
  schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
    <ContentProtection schemeIdUri="urn:mpeg:dash:mp4protection:2011"
  value="cenc" cenc:default_KID="a1b1c1d1-a2b2-a3b3-a4b4-a5b5c5d5e5f5"/>
    <SegmentTemplate timescale="44100" media="audio_COSTARICA.mov_aac-
  lc_128k_dash$Number$.m4s" startNumber="1" duration="88023"
  initialization="audio_COSTARICA.mov_aac-lc_128k_dashinit.mp4"/>
   </Representation>


   <Representation id="2" mimeType="audio/mp4" codecs="mp4a.40.2"
  audioSamplingRate="44100" startWithSAP="1" bandwidth="201552">
    <AudioChannelConfiguration
  schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
    <ContentProtection schemeIdUri="urn:mpeg:dash:mp4protection:2011"
  value="cenc" cenc:default_KID="a1b1c1d1-a2b2-a3b3-a4b4-a5b5c5d5e5f5"/>
    <SegmentTemplate timescale="44100" media="audio_COSTARICA.mov_aac-
  lc_192k_dash$Number$.m4s" startNumber="1" duration="88023"
  initialization="audio_COSTARICA.mov_aac-lc_192k_dashinit.mp4"/>
   </Representation>
  </AdaptationSet>
 <AdaptationSet segmentAlignment="true" maxWidth="1920" maxHeight="1080"
  maxFrameRate="30000/1001" par="16:9" lang="und">
   <Representation id="3" mimeType="video/mp4" codecs="avc1.4d401e" width="512"
  height="288" frameRate="30000/1001" sar="1:1" startWithSAP="1" bandwidth="365953">
    <ContentProtection schemeIdUri="urn:mpeg:dash:mp4protection:2011"
  value="cenc" cenc:default_KID="a1b1c1d1-a2b2-a3b3-a4b4-a5b5c5d5e5f5"/>
    <SegmentTemplate timescale="30000"
  media="video_COSTARICA.mov_512x288_h264-360k_dash$Number$.m4s"
  startNumber="1" duration="59040"
  initialization="video_COSTARICA.mov_512x288_h264-360k_dashinit.mp4"/>
 </Representation>
   <Representation id="4" mimeType="video/mp4" codecs="avc1.4d401e" width="704"
  height="396" frameRate="30000/1001" sar="1:1" startWithSAP="1" bandwidth="625354">
    <ContentProtection schemeIdUri="urn:mpeg:dash:mp4protection:2011"
  value="cenc" cenc:default_KID="a1b1c1d1-a2b2-a3b3-a4b4-a5b5c5d5e5f5"/>
    <SegmentTemplate timescale="30000"
  media="video_COSTARICA.mov_704x396_h264-620k_dash$Number$.m4s"
  startNumber="1" duration="59040"
  initialization="video_COSTARICA.mov_704x396_h264-620k_dashinit.mp4"/>
   </Representation>

 <Representation id="5" mimeType="video/mp4" codecs="avc1.64001f" width="896" height="504"
 frameRate="30000/1001" sar="1:1" startWithSAP="1" bandwidth="1345581">
 <ContentProtection schemeIdUri="urn:mpeg:dash:mp4protection:2011" value="cenc"
 cenc:default_KID="a1b1c1d1-a2b2-a3b3-a4b4-a5b5c5d5e5f5"/>
 <SegmentTemplate timescale="30000" media="video_COSTARICA.mov_896x504_h264-
 1340k_dash$Number$.m4s" startNumber="1" duration="59040"
 initialization="video_COSTARICA.mov_896x504_h264-1340k_dashinit.mp4"/>
  </Representation>
```

```
  <Representation id="6" mimeType="video/mp4" codecs="avc1.640020" width="1280"
height="720" frameRate="30000/1001" sar="1:1" startWithSAP="1" bandwidth="2503819">
   <ContentProtection schemeIdUri="urn:mpeg:dash:mp4protection:2011"
value="cenc" cenc:default_KID="a1b1c1d1-a2b2-a3b3-a4b4-a5b5c5d5e5f5" />
   <SegmentTemplate timescale="30000"
media="video_COSTARICA.mov_1280x720_h264-2500k_dash$Number$.m4s"
startNumber="1" duration="59040"
initialization="video_COSTARICA.mov_1280x720_h264-2500k_dashinit.mp4" />
  </Representation>
  <Representation id="7" mimeType="video/mp4" codecs="avc1.640028" width="1920"
height="1080" frameRate="30000/1001" sar="1:1" startWithSAP="1" bandwidth="4502264">
   <ContentProtection schemeIdUri="urn:mpeg:dash:mp4protection:2011"
value="cenc" cenc:default_KID="a1b1c1d1-a2b2-a3b3-a4b4-a5b5c5d5e5f5" />
   <SegmentTemplate timescale="30000"
media="video_COSTARICA.mov_1920x1080_h264-4500k_dash$Number$.m4s"
startNumber="1" duration="59040"
initialization="video_COSTARICA.mov_1920x1080_h264-4500k_dashinit.mp4" />
  </Representation>
 </AdaptationSet>
 </Period>
</MPD>
```

II.4 Realizzazione di un piccolo sito per la visualizzazione del file multimediale

L'ultima parte del nostro progetto include la realizzazione di un sito in locale che simuli la messa in rete di file elaborati come descritto precedentemente.
Per la realizzazione del sito abbiamo usato i seguenti strumenti:

- Server Apache, per poter sviluppare il sito in locale
- Node.js, per simulare un server di licenza da cui recuperare la chiave di decrittazione
- Google Shaka Player, per poter riprodurre il video

II.4.1 Server Apache

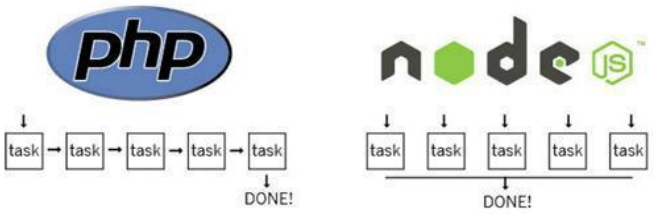
Il web server Apache è il più popolare tra i web server disponibili, oltre ad avere a disposizione una vasta documentazione (complice il vasto uso negli anni).
In Ubuntu, Apache può essere facilmente installato usando il package manager

sudo apt-get install apache2

Finita l'installazione, aprendo il browser e digitando *localhost,* vedremo la pagina di benvenuto di Apache. Nella pagina di benvenuto viene indicata la *document root* di default (*var/www/html*). Per modificarla in *var/www* basterà modificare il file *000-default.conf*, presente nella cartella *etc/apache2/sites-enabled*.

II.4.2 Node.js

Node.js è una tecnologia lato server con un modello di I/O asincrono basato sugli eventi, che risulta particolarmente adatto ad essere programmato in Javascript e per le applicazioni web. Il codice viene eseguito tramite un motore Javascript sviluppato da Google (V8) con ottime prestazioni sia nel browser che stand-alone.
Normalmente le applicazioni server-side sono scritte in PHP. Queste applicazioni prevedono lo scorrere di una serie di istruzioni che coinvolgono chiamate a servizi esterni (es. query al database o operazioni di lettura/scrittura). Queste sono tutte operazioni bloccanti, quindi lo script rimane in attesa della risposta dal componente esterno di turno, aumentando di molto i tempi d'attesa. È evidente una inefficienza generale nello sfruttamento delle risorse a disposizione.
La asincronicità delle applicazioni realizzate con node.js permette una notevole ottimizzazione dei tempi e delle risorse perché vengono evitati blocchi e tempi morti.



Node.js può essere installato facilmente dal package manager di Ubuntu usando i seguenti comandi

sudo apt-get install nodejs
sudo apt-get install npm

II.4.3 Google Shaka Player

Shaka Player è una libreria JavaScript per lo streaming di video adattativo. È in grado di mostrare contenuti DASH senza l'utilizzo di ulteriori plugin perché si basa sulla *MediaSource Extensions* (per il prefetching e il buffering dello streaming direttamente in Javascript, lato client) e sulla *Encrypted Media Extensions*.

Per poter utilizzare Shaka servono:

- Un server locale
- Node package manager
- Java Runtime Environment 7+
- Python 2.7.x
- Git 1.7.10+

Seguendo il tutorial messo a disposizione in rete, abbiamo clonato il progetto presente su github con i comandi

git clone https://github.com/google/shaka-player.git cd shaka-player

Abbiamo poi compilato la libreria con il comando

python build/all.phy

Continuando a seguire il tutorial, abbiamo creato delle semplici pagine html contenenti un elemento video e le rispettive applicazioni Javascript.

II.4.4 Contenuto pagine html

Abbiamo creato una pagina index, il cui compito è semplicemente quello di fare da collegamento alle altre pagine.

La prima pagina linkata nell'index mostra il video Costarica originale. All'interno del tag body è presente un semplice tag video.

```
<!DOCTYPE html>
 <html>
<head>
<!--Specify character encoding (Unicode)-->
     <meta charset="UTF-8">
     <title>Simple video</title>
</head>
<body><center><br/>
   <video id="video" src="video/COSTARICA"
    width="640"
    poster="//shaka-player-demo.appspot.com/assets/poster.jpg"
    controls autoplay></video>
</center></body>
</html>
```

Le altre pagine usano il video Costarica crittato. Il contenuto del body è invariato, ma nell'head sono presenti, oltre al charset e al titolo della pagina, i collegamenti ai tre script per far funzionare lo Shaka player e allo script che permette la decodifica del video.

II.4.5 Applicazione Javascript

La nostra piccola applicazione Javascript consente la decrittazione e la visione del video, a patto che siano fornite le chiavi.

Il codice di base del nostro Javascript è il seguente (e NON permette la visualizzazione del video criptato)

```
var manifestUri = 'video/packaged/live/live.mpd';

function initApp() {
 // Verbose logs, which can generate a lot of output
 shaka.log.setLevel(shaka.log.Level.V1);
 // Install built-in polyfills to patch browser incompatibilities.
shaka.polyfill.installAll();
 // Check to see if the browser supports the basic APIs Shaka needs.
 if (shaka.Player.isBrowserSupported()) {
    initPlayer();
     } else { console.error('Browser not supported!');  }
 }

function initPlayer() {

// Create a Player instance.
var video = document.getElementById('video');
var player = new shaka.Player(video);

// Attach player to the window to make it easy to access in the JS console.
window.player = player;

// Listen for error events
player.addEventListener('error', onErrorEvent);

// Try to load a manifest.
 player.load(manifestUri).then(function() {

// This runs if the asynchronous load is successful.
console.log('The video has now been loaded!');
 }).catch(onError); // onError is executed if the asynchronous load fails.
}

function onErrorEvent(event) {

 // Extract the shaka.util.Error object from the event.
onError(event.detail);
 }

function onError(error) {

 // Log the error.
console.error('Error code', error.code, 'object', error);
 }

document.addEventListener('DOMContentLoaded', initApp);
```

La variabile manifestUri contiene l'indirizzo del manifest del video.

La funzione initApp:

1. Imposta il livello dei log da mostrare in console; nel nostro caso abbiamo scelto il livello intermedio, V1, che consente di avere dei log abbastanza accurati e non troppo "verbosi"
2. Installa le polyfills di Shaka, utili per la verifica delle incompatibilità del browser con le API della libreria
3. Verifica se il browser in uso è in grado di supportare le API; se sono supportate avvia il player, altrimenti mostra nella console un messaggio di errore

La funzione initPlayer:

1. Crea un oggetto player che contenga l'elemento video
2. Mette il player in ascolto degli errori
3. Tenta il caricamento del manifest; se non si verificano errori, nel player sarà visibile il video

Per permettere la visualizzazione del nostro video, è necessario fornire al player la chiave di decrittazione. Usando Clearkey, abbiamo due modi per fornire tale chiave: la possiamo esplicitare direttamente nell'applicazione o possiamo richiederla da un server esterno.

Per la prima opzione, basta aggiungere nel codice in alto il seguente comando

```
player.configure ({ drm: { clearKeys: {
  'a1b1c1d1a2b2a3b3a4b4a5b5c5d5e5f5':
  'ccc0f2b3b279926496a7f5d25da692e9',
  'd1c1b1a1b2a2b3a3a4b4a5b5c5d5e5f5':
  '3a2a1b68dd2bd9b2eeb25e84c4776668'}
    } });
```

prima del caricamento del manifest.
Per la seconda opzione è necessario specificare il server di licenza, configurarlo e creare la richiesta da inviare.
Per specificare il server da usare, basterà aggiungere in alto una variabile che specifichi l'indirizzo e la porta del server. Nel nostro caso avremo come indirizzo localhost e come porta 8584 (come vedremo fra poco, il server da noi utilizzato rimane in ascolto su tale porta)

var licenseServer = 'http://localhost:8584';

Per la configurazione del server, dobbiamo indicare come tipologia di server un server Clearkey (org.w3.clearkey) e come server il nostro *licenseServer*.

player.configure({ drm: { servers: { 'org.w3.clearkey': licenseServer}});

La richiesta da inviare invece sarà realizzata specificando il parametro che ci interessa, l'ID della chiave per la decrittazione.

```
player.getNetworkingEngine().registerRequestFilter(function(type,
 request) { if (type ==
 shaka.net.NetworkingEngine.RequestType.LICENSE) {
  // This is the specific parameter name and value the server
  wants: request.uris[0] +=
  '?keyid=a1b1c1d1a2b2a3b3a4b4a5b5c5d5e5f5';
 }

 });
```

Anche in questo caso, i due metodi del player devono essere richiamati prima del caricamento del manifest.

II.4.6 Server di licenza

Sul sito di **cablelabs** è disponibile il codice di un piccolo server node.js. In seguito alla ricezione, tramite protocollo http o https, di una richiesta, contenente l'id di una chiave, ne restituisce il valore, a patto che tale chiave sia tra quelle definite nel server.

Per le richieste tramite protocollo https il server rimane in attesa sulla porta 8585, mentre per le richieste tramite http la porta è la 8584.

Sono inserite anche le CORS headers, che permettono alle applicazioni client di richiedere i dati dal server indipendentemente da dove risiede quest'ultimo.

```
var  https=require('https');
var http = require('http');
var fs = require('fs');
var url = require('url');

console.log("");
console.log("CableLabs ClearKey License Server");
console.log("");

keys = {
   'a1b1c1d1a2b2a3b3a4b4a5b5c5d5e5f5': new
   Buffer("ccc0f2b3b279926496a7f5d25da692e9", 'hex'),
   'd1c1b1a1b2a2b3a3a4b4a5b5c5d5e5f5': new
   Buffer("3a2a1b68dd2bd9b2eeb25e84c4776668", 'hex')
};
var options = {
 key: fs.readFileSync('security/cl_clearkey-key.pem'),
 cert: fs.readFileSync('security/cl_clearkey-cert.pem')
};
```

```
var addCORSHeaders = function(res, length) {
  res.writeHeader(200, {
    "Content-Length": length,
    "Content-Type": 'application/json',
    "Access-Control-Allow-Origin": '*',
   "Access-Control-Allow-Methods": 'GET, PUT, POST, DELETE, OPTIONS',
   "Access-Control-Allow-Headers": 'Content-Type, Authorization, Content-Length, X-Requested Width'});
  };

https.createServer(options, function(req, res) {

   addCORSHeaders(res);
   res.end("hello world\n");

}).listen(8585);

http.createServer(function(req, res) {
   var parsed_url = url.parse(req.url, true);
   var query = parsed_url.query;
   console.log("Received key request! Query = %j", query);

    // Validate query string
 if (query === undefined || query.keyid === undefined) {
    console.error("Illegal request!");
    res.writeHeader(400, "Illegal query string");
    res.end();
 }
var keyIDs = [];
 if (query.keyid instanceof Array) {
    keyIDs = query.keyid; } else {
           keyIDs.push(query.keyid); }

var keyarray = [];
    for (var i = 0; i < keyIDs.length; i++) {
        var keyID = keyIDs[i];
       if (!keys.hasOwnProperty(keyID)) {
          console.warn("KeyID %s not registered in our lookup table!", keyID);
          continue;
     }
     var keypair = {
     kid: new Buffer(keyIDs[i], 'hex').toString('base64'),
     k: keys[keyIDs[i]].toString('base64')
     }
    keyarray.push(keypair);
     }
 var response = {
       keys: keyarray
   };
console.log("Returning key array: %j", response);
var json_str_response = JSON.stringify(response);
addCORSHeaders(res, json_str_response.length);
res.write(json_str_response);
res.end();

}).listen(8584);
```

Riferimenti

[1] https://it.wikipedia.org/wiki/Digital_rights_management

[2] http://www.cs.unibo.it/~margara/page2/page6/page25/assets/drm.pdf

[3] https://en.wikipedia.org/wiki/Sony_BMG_copy_protection_rootkit_scandal

[4] https://www.html5rocks.com/en/tutorials/eme/basics/

[5] http://docs.unified-streaming.com/documentation/drm/mpeg-dash.html

[6] https://nougat.cablelabs.com/Innovation/mse-eme/tree/master

[7] https://html5.cablelabs.com/mse-eme/doc/creation.html#packaging

[8] https://ffmpeg.org/

[9] https://gpac.wp.imt.fr/

[10] https://tech.ebu.ch/docs/events/webinar043-mpeg-dash/presentations/ebu_mpeg-dash_webinar043.pdf

[11] https://shaka-player-demo.appspot.com/docs/api/tutorial-welcome.html

[12] https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-16-04#conclusion

[13] http://www.mrwebmaster.it/javascript/introduzione-node-js_12163.html4

[14] https://linuxconfig.org/how-to-install-node-js-on-ubuntu-16-04-xenial-xerus-linux-server#h5-1-node-js-installation-from-ubuntu-repository

[15] https://shaka-player-demo.appspot.com/docs/api/tutorial-welcome.html

[16] https://en.wikipedia.org/wiki/Media_Source_Extensions