DRM-Protected Adaptive Video Streaming Practical Work

Video Streaming protected using PBKDF-2

Miguel Angel Barrero Díaz

- **INTRODUCTION**

The objective of the program is cast multimedia content using https protocol, specifically video content. This stream content is protected with a password derived from the use of a Password Based Key Deviation Function in such a way that in order to decrypt the content a password must be provided to compute the derived key.

The system has three multimedia files protected with a different password and a different manifest file must be created for each of them, the manifest file must contain a KID, this value are a 16 byte word and a 16 byte length key obtained from the password and the salt provided to the PBKDF-2 function.

- **CODE TO GENERATE THE KID AND THE KEY**

To create de KID and the KEY I have developed a simple web-app to automate the process where we introduce a password and a salt, then the KID and the KEY are both showed in the screen. The KID is obtained using a pseudo random generator and the KEY is obtained using a PBKDF-2 function, the javascript code to carry out this computations has been developed using node-js packets by means of the Node Package Manager (npm) writing the code with the Visual Studio Code IDE.

The code of both the javascript and html scripts is following depicted (file compute.js):

```
/*The browserify npm packet is used to derive a bundle file that contains the current file
and all the required libraries merged into a single file that we can call with the script tag
directly from the html code, the command used is browserify main.js -o bundle_compute.js*/

var crypto = require('pbkdf2');
var prg = require('crypto');

/*Function to compute the KID and the KEY*/

function compute(){

var secret = document.getElementById('password').value;
var salt = document.getElementById('salt').value;
console.log(secret);
console.log(salt);

/*Calling the pbkdf2 function to create a KEY */

crypto.pbkdf2(secret, salt , 10000, 16, 'sha512',(err,derivedKey)=>
{
    if (err) throw err;
    console.log(derivedKey.toString('hex')+'--KEY');
    document.getElementById('key').innerHTML = `KEY=0x${derivedKey.toString('hex')}`;
  });

/*Calling randomBytes function to create a rabdon KID */

prg.randomBytes(16, (err, buf) => {
    if (err) throw err;
    console.log(`${buf.toString('hex')}--KID`);
    document.getElementById('kid').innerHTML = `KID=0x${buf.toString('hex')}`;
  });
}

/*Calling the compute function when the compute button is clicked*/

var button = document.getElementById('button');
var saltfield = document.getElementById('salt');
button.addEventListener("click", compute, false);

/*Clearing the salt field*/

window.addEventListener('load', (event) => {
  this.value = "";
  console.log('The page is loaded');
  saltfield.value = this.value;
  console.log('Fields cleared');
});
```

I have created a function named compute() used to carry out the necessary computations. The function read the content of the input fields and obtain the key calling the function pbkdf-2 provided with npm

packet crypto, we pass to the function the secret,i.e. the password, the salt,the number of iterations (10000 in my case), the length of the the derived key, in my case 16 bytes and the hashing primitive used that I have decided to be SHA512.
To compute the KID I call the function randomBytes, we must provide the length of output string,in this case 16,finally, in both cases the returned values are written in a <p> html tag.
An event listener is implemented to call the compute() function when the button compute is clicked.

```
button.addEventListener("click", compute, false);
```

The idea is that all this computations to being performed in the client side,to achieve this, i used the npm packet browserify, this solution bundle all the dependencies in javascript format in a single file that can be loaded in the html code using the <script> tag with the attribute type set to "module". Once installed the browserify module in the development environment with *npm install -g browserify* we can bundle the code and dependencies in a single file with the command *browserify compute.js -o bundle_compute.js.*

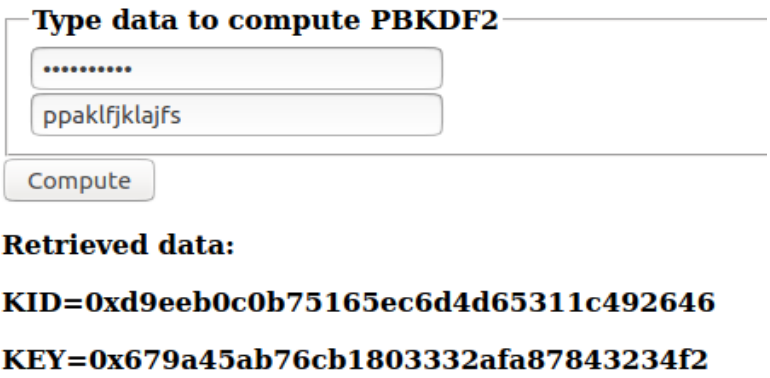
The html code is:

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Compute pbkdf2</title>

</head>

<body>
    <form name="data">
        <fieldset>
            <legend><strong>Type data to compute PBKDF2</strong></legend>
            <input id="password" type=password placeholder=password><br>
            <input id="salt" type=text placeholder=salt><br>
        </fieldset>
        <input id="button" type=button value="Compute">
    </form>
    <strong><p> Retrieved data:</p></strong>
    <strong><p id="kid"></p></strong>
    <strong><p id="key"></p></strong>
</body>
<script type="module" src="./pbkdf-2/bundle_compute.js"></script>
</html>
```

Once the server is running we can execute the web-app, the execution will return a KID and a KEY that we will use later.

```
(base) miguel@miguel-N24-25JU:~/URV/MULTIMEDIA SECURITY/Practical/Activity_1/mpe
gdash$ python server.py
127.0.0.1 - - [18/Apr/2020 13:09:39] "GET /Pbkdf2.html HTTP/1.1" 200 -
127.0.0.1 - - [18/Apr/2020 13:09:39] "GET /pbkdf-2/bundle_compute.js HTTP/1.1" 2
00 -
```

In this example I have typed a password and a salt and the app returns the KID and the KEY, to access the web app we type in the browser *https://localhost:4443/Pbkdf2.html*



- **WRITING THE XML FILES**

The xml files will contain the KID and the KEY obtained with the web-app Pbkdf2 along with other configuration information.
To obtain the values i have always executed the Pbkdf.html app using "MiguelAngel"as salt value. The passwords are included in the main folder into a file named "passwords" and in this document in the last page. The KID will be different in each execution since we are using a pseudo random generator. Note that i maintain always the same Initialization Vector (IV).
The content of the three xml files are:

-drmed.xml

```
<GPACDRM type="CENC AES-CTR">
  <DRMInfo type="pssh" version="1">
    <BS ID128="1077efecc0b24d02ace33c1e52e2fb4b"/>
    <BS bits="32" value="1"/>
    <BS ID128="bea0134a060547b39555e9adc2a19e6f"/>
  </DRMInfo>
  <CrypTrack IV_size="8" first_IV="0x603f39bfdb3d9799" isEncrypted="1" saiSavedBox="senc" trackID="1">
    <key KID="0xbea0134a060547b39555e9adc2a19e6f" value="0xda03c419732bb61e04f6f66a62ea9026"/>
  </CrypTrack>
</GPACDRM>
```

In the GPACDRM tag we indicate the protection scheme used,i will use CENC AES-CTR for all the streams. The pssh system ID will be the same for all the files,i.e 1077efecc0b24d02ace33c1e52e2fb4b, the Initialization vector will be the same for all files,i.e. 0x603f39bfdb3d9799.
For the ed_hd.mp4 file the computed KID is 0xbea0134a060547b39555e9adc2a19e6f and the derived KEY from the password videoed and the salt MiguelAngel is 0xda03c419732bb61e04f6f66a62ea9026.

-drmpopeye.xml

The sole difference with respect to the previous file are the KID and KEY fields,in this case the obtained values was 0xbe445832617eb0b7fcb196b65e81b694 for the KID and 0xfeaea129030758fd025bc0c4fa555e3d for the KEY.

```
<GPACDRM type="CENC AES-CTR">
  <DRMInfo type="pssh" version="1">
    <BS ID128="1077efecc0b24d02ace33c1e52e2fb4b"/>
    <BS bits="32" value="1"/>
    <BS ID128="be445832617eb0b7fcb196b65e81b694"/>
  </DRMInfo>
  <CrypTrack IV_size="8" first_IV="0x603f39bfdb3d9799" isEncrypted="1" saiSavedBox="senc" trackID="1">
    <key KID="0xbe445832617eb0b7fcb196b65e81b694" value="0xfeaea129030758fd025bc0c4fa555e3d"/>
  </CrypTrack>
</GPACDRM>
```

-drmsintel.xml

In this case the values are 0xea3ddb3fab812f86c70f18fb4fa51873 and 0xe4884b39de7ecab983bf1569050a1275 respectively.

```
<GPACDRM type="CENC AES-CTR">
  <DRMInfo type="pssh" version="1">
    <BS ID128="1077efecc0b24d02ace33c1e52e2fb4b"/>
    <BS bits="32" value="1"/>
    <BS ID128="ea3ddb3fab812f86c70f18fb4fa51873"/>
  </DRMInfo>
  <CrypTrack IV_size="8" first_IV="0x603f39bfdb3d9799" isEncrypted="1" saiSavedBox="senc" trackID="1">
    <key KID="0xea3ddb3fab812f86c70f18fb4fa51873" value="0xe4884b39de7ecab983bf1569050a1275"/>
  </CrypTrack>
</GPACDRM>
```

This files will be used after to generate the encrypted files for each video and audio stream.

- **PREPARING THE VIDEO STREAMS**

We must prepare all the DASH video streams in the same way that we do in the tutorial.

-**Sintel vídeo:** I will show the process with the Sintel video,but it is the same for the rest.

Using ffmpeg we genreate the three streams of three different bit rates, 500 kbps,1000 kbps and 2000 kbps. I modifies the text superposed in each case.

```
(base) miguel@miguel-N24-25JU:~$ ffmpeg -i sintel-2048-stereo_512kb.mp4 -vf "sca
le=640:-1,drawtext=fontfile=SourceSansPro-Regular.otf:text="500Kb":fontcolor=whi
te:fontsize=24:box=1:boxcolor=black:x=20:y=80,drawtext=fontsize=21:fontfile=Sour
ceSansPro-Regular.otf:timecode='00\:00\:00\:00':rate=24:fontsize=32:fontcolor='w
hite':box=1:boxcolor=black:x=20:y=50" -r 24 -g 50 -codec:v libx264 -s 1280x720 -
b:v 500k output_500k_sintel.mp4
```

```
(base) miguel@miguel-N24-25JU:~$ ffmpeg -i sintel-2048-stereo_512kb.mp4 -vf "sca
le=640:-1,drawtext=fontfile=SourceSansPro-Regular.otf:text="2000Kb":fontcolor=wh
ite:fontsize=24:box=1:boxcolor=black:x=20:y=80,drawtext=fontsize=21:fontfile=Sou
rceSansPro-Regular.otf:timecode='00\:00\:00\:00':rate=24:fontsize=32:fontcolor='
white':box=1:boxcolor=black:x=20:y=50" -r 24 -g 50 -codec:v libx264 -s 1280x720
-b:v 2000k output_2000k_sintel.mp4
```

```
(base) miguel@miguel-N24-25JU:~$ ffmpeg -i sintel-2048-stereo_512kb.mp4 -vf "sca
le=640:-1,drawtext=fontfile=SourceSansPro-Regular.otf:text="1000Kb":fontcolor=wh
ite:fontsize=24:box=1:boxcolor=black:x=20:y=80,drawtext=fontsize=21:fontfile=Sou
rceSansPro-Regular.otf:timecode='00\:00\:00\:00':rate=24:fontsize=32:fontcolor='
white':box=1:boxcolor=black:x=20:y=50" -r 24 -g 50 -codec:v libx264 -s 1280x720
-b:v 1000k output_1000k_sintel.mp4
```

We define the text format,the font type,the font size and color,the box that content the text,its background and position,we also include a timer with its respective position and color,The codec to be used (H.264/MPEG-4 AVC),the video size, in this case 1280x720 and also we define 24 fps and a GOP size of 50.
This process is reproduced with the other missing videos.

Once we have created the three streams we must separate the audio stream from the video streams.

```
(base) miguel@miguel-N24-25JU:~$ ffmpeg -i output_500k_sintel.mp4 -an -c copy output_500k_v_sintel.mp4
```

```
(base) miguel@miguel-N24-25JU:~$ ffmpeg -i output_1000k_sintel.mp4 -an -c copy output_1000k_v_sintel.mp4
```

```
(base) miguel@miguel-N24-25JU:~$ ffmpeg -i output_2000k_sintel.mp4 -an -c copy output_2000k_v_sintel.mp4
```

Here we have applied a filter where the audio is retrieved.

Then we map the audio stream 0:1 from the video into an audio file.

```
(base) miguel@miguel-N24-25JU:~$ ffmpeg -i output_2000k_sintel.mp4 -map 0:1 -c copy audio_only_sintel.m4a
```

Once we have created the previous 4 stream,three of them are video streams without audio and an audio stream, we must create the encrypted files. In order to do that I use the MP4Box software using the xml file that corresponds to each video, i will continue with the sintel example.

```
(base) miguel@miguel-N24-25JU:~$ MP4BOX -crypt drmsintel.xml output_500k_v_sintel.mp4 -out output_500k_v_enc.mp4
```

```
(base) miguel@miguel-N24-25JU:~$ MP4Box -crypt drmsintel.xml output_1000k_v_sintel.mp4 -out output_1000k_v_sintel_enc.mp4
```

```
(base) miguel@miguel-N24-25JU:~$ MP4Box -crypt drmsintel.xml output_2000k_v_sintel.mp4 -out output_2000k_v_sintel_enc.mp4
```

```
(base) miguel@miguel-N24-25JU:~$ MP4Box -crypt drmsintel.xml audio_only_sintel.m4a -out audio_only_sintel_enc.m4a
```

Once i have created all the encrypted files I can create the DASH stream in order to be served over an http connection

```
(base) miguel@miguel-N24-25JU:~$ MP4Box -dash 5000 -segment-name "output/outputs
eg-%s" -url-template -bs-switching no -out sintel.mpd -rap audio_only_sintel_enc
.m4a output_2000k_v_sintel_enc.mp4 output_1000k_v_sintel_enc.mp4 output_500k_v_s
intel_enc.mp4
```

The execution of this command will create the manifest file,in this case i have named it sintel.mpd, and the DASH stream files. We pass to the command the names of the four encrypted files in order to generate the video segments that will be stored in the folder output/ with the name outputseg- and the

correspondent name in each case,for example, the file outputseg-audio_only_sintel_enc21.m4s, corresponds to the segment 21 of the audio file.

The returned mpd file has the following content:

```
<?xml version="1.0"?>
<!-- MPD file Generated with GPAC version 0.8.0-rev188-gedca0771-master  at 2020-04-15T22:45:09.961Z-->
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500S" type="static" mediaPresentationDuration="PT0H14M48.064S"
maxSegmentDuration="PT0H0M6.625S" profiles="urn:mpeg:dash:profile:full:2011" xmlns:cenc="urn:mpeg:cenc:2013">
 <ProgramInformation moreInformationURL="http://gpac.io">
  <Title>sintel.mpd generated by GPAC</Title>
 </ProgramInformation>

 <Period duration="PT0H14M48.064S">
  <AdaptationSet segmentAlignment="true" lang="eng" startWithSAP="1">
   <ContentProtection schemeIdUri="urn:mpeg:dash:mp4protection:2011" value="cenc" cenc:default_KID="ea3ddb3f-ab81-2f86-c70f-18fb4fa51873"/>
   <SegmentTemplate media="output/outputseg-audio_only_sintel_enc$Number$.m4s" initialization="output/outputseg-audio_only_sintel_encinit.mp4"
timescale="48000" startNumber="1" duration="240000"/>
   <Representation id="1" mimeType="audio/mp4" codecs="mp4a.40.2" audioSamplingRate="48000" bandwidth="130000">
    <AudioChannelConfiguration schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
   </Representation>
  </AdaptationSet>
  <AdaptationSet segmentAlignment="true" maxWidth="1280" maxHeight="720" maxFrameRate="24" par="16:9" lang="eng" startWithSAP="1">
   <ContentProtection schemeIdUri="urn:mpeg:dash:mp4protection:2011" value="cenc" cenc:default_KID="ea3ddb3f-ab81-2f86-c70f-18fb4fa51873"/>
   <Representation id="2" mimeType="video/mp4" codecs="avc1.64001F" width="1280" height="720" frameRate="24" sar="1:1" bandwidth="2073208">
    <SegmentTemplate media="output/outputseg-output_2000k_v_sintel_enc$Number$.m4s" initialization="output/outputseg-output_2000k_v_sintel_encinit.mp4"
timescale="12288" startNumber="1" duration="61440"/>
   </Representation>
   <Representation id="3" mimeType="video/mp4" codecs="avc1.64001F" width="1280" height="720" frameRate="24" sar="1:1" bandwidth="1022808">
    <SegmentTemplate media="output/outputseg-output_1000k_v_sintel_enc$Number$.m4s" initialization="output/outputseg-output_1000k_v_sintel_encinit.mp4"
timescale="12288" startNumber="1" duration="61440"/>
   </Representation>
   <Representation id="4" mimeType="video/mp4" codecs="avc1.64001F" width="1280" height="720" frameRate="24" sar="1:1" bandwidth="504112">
    <SegmentTemplate media="output/outputseg-output_500k_v_sintel_enc$Number$.m4s" initialization="output/outputseg-output_500k_v_sintel_encinit.mp4"
timescale="12288" startNumber="1" duration="61440"/>
   </Representation>
  </AdaptationSet>
 </Period>
</MPD>
```

- **PLAYING THE VIDEOS**

To play the videos i have written other script with node-js ,in this case the client only compute the pbkdf function when the load button is clicked,here is the code of the file main.js:

```
var crypto = require('pbkdf2');


/*Function to compute the key from the password */

function compute(){

  var secret = document.getElementById('password').value;
  var salt = "MiguelAngel";
  console.log(secret);
  console.log(salt);

  /*Calling the pbkdf2 function to derive the KEY from the password */

  crypto.pbkdf2(secret, salt , 10000, 16, 'sha512',(err,derivedKey)=>
    {
      if (err) throw err;
      console.log(derivedKey.toString('hex')+'--KEY');
      document.getElementById('key').value = `${derivedKey.toString('hex')}`;
    });
}


/*Calling the compute function when the compute button is clicked*/

var button = document.getElementById('loadButton');
var key = button.addEventListener("click", compute, false);
console.log(key);
```

The salt is fixed to the string "MiguelAngel", the script is quite simple and it is very similar to the designed to compute the KID and the KEY that we have previously seen.
As did before,the script is bundled with browserify in a file named bundle.js with the command *browserify main.js -o bundle.js* . This file is loaded with a <script> tag into the html code. The returned KEY value is written in a hidden field in the web page.

The html code is developed over the provided example code *indexTest.html*, some modifications are applied, for example, is implemented and selector for the video that automates the manifest file selection.

```
<label>Select Video:</label>
<select id="videoSelector">
    <option value="ed.mpd">Ed</option>
    <option value="popeye.mpd">Popeye</option>
    <option value="sintel.mpd">Sintel</option>
</select>
```

The function initApp() is also modified. An if-else statement is added to automate the kid selection process in function of the selected video,the appropriate value is written in the kid tag when the load button is clicked. It is also included a setTimeout() function, due to the asynchronous execution of javascript, the browser need time to compute the pbkdf function output.

```
function initApp() {
        // 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()) {
        // Everything looks good!
        player = initPlayer();
    } else {
        // This browser does not have the minimum set of APIs we need.
        console.error('Browser not supported!');
    }

    var button = document.getElementById('loadButton')
    button.addEventListener("click",(event) => {

        //Clear the password and error fields,and destroy the previous player
         document.getElementById('error').innerHTML = ''
        document.getElementById('password').value = '';
        player.destroy();
        //Timeout to compute the derived key
        setTimeout(function(){

        manifest = document.getElementById('videoSelector').value
        console.log(video);

        //Statement to load the correct kid
        if(manifest =="ed.mpd"){
            kid = 'bea0134a060547b39555e9adc2a19e6f';
            document.getElementById('kid').innerHTML = 'KID:'+kid;
        }
        else if(manifest =="popeye.mpd"){
            kid = 'be445832617eb0b7fcb196b65e81b694';
            document.getElementById('kid').innerHTML = 'KID:'+kid;
        }
        else{
            kid = 'ea3ddb3fab812f86c70f18fb4fa51873';
            document.getElementById('kid').innerHTML = 'KID:'+kid;
        }
        key = document.getElementById('key').value

        console.log(manifest)
        console.log(kid)
        console.log(key)

        player = initPlayer()
        loadVideo(player, manifest, kid, key)
        //1'5 seconds of timeout
        },1500);
    });
}
```

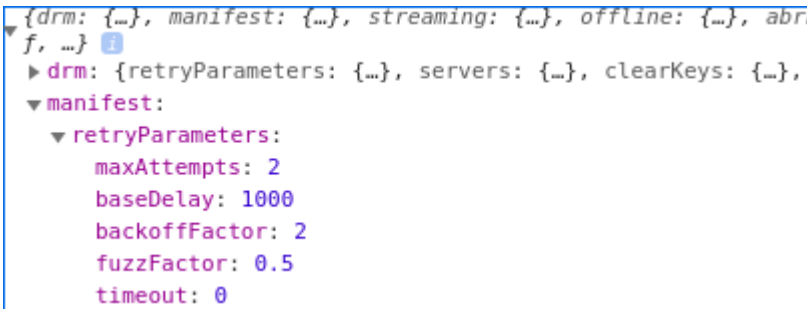
Other function is added to clear the fields kid and key when the page is reloaded.

```
function clearContent(){
     document.getElementById('key').value = '';
     document.getElementById('kid').value = '';
}
```

The function is called from and event listener.

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

When an erroneous password is typed an error 3016 is prompted,this error is captured by the event listener into the function initPlayer() (player.addEventListener('error', onErrorEvent);) and is managed into the function onErrorEvent. We print a message on screen informing the user that the max amount of tries are 2. This value is described into the manifest field retryParameter:maxAttemps.



The program have been tested in Chromium browser ver. 80.0.3987.163 for Ubuntu without issues.

- **PBKDF-2 OPERATIVE**

As we know the key is derived using PBKDF-2 ,the operative of this function is as follows:

$$D_{key} = PBKDF2(PRF, pass, salt, iterations, length)$$

The derived key is obtained from the concatenation of the selected length in several blocks:

$D_{key} = T_1 || T_2 ..... || T_i$ where i is $length / T \quad block\,length$

Each block is obtained:

$T_i = F(pass, salt, iterations, i)$ The function F is the xor of the output PRF functions (for example HMAC-SHA-512) carried out in the following way:

$U_1 = PRF(pass, salt || \text{INT\_32\_Big Endian (i)})$ using the password as key of the PRF function.
$U_2 = PRF(pass, U_1)$
.
.

.

$$U_{iterations} = PRF(pass, U_{iterations-1})$$

Finally:

$$T_i = F(pass, salt, iterations, i) = U_1 \oplus U_2 \oplus ..... \oplus U_{iterations}$$

- **PASSWORDS OF VIDEOS**

VIDEO	PASSWORD
sintel	**videosintel**
popeye	**videopopeye**
ed	**videoed**