

# IMPLEMENTATION OF NON-INSTALL TYPE DRM SYSTEM TO PREVENT ILLEGAL COPYING OF WEB-BASED MEDIA CONTENT

<sup>1</sup> KI-BOK NAM, <sup>2</sup>KOO-ROCK PARK, <sup>3</sup>JOON-YOUNG KIM, <sup>4</sup>YOUNG-SUK CHUNG

<sup>1</sup> School of Computer Engineering, Kongju National University, Korea

<sup>2</sup> Key Laboratory of Computer Science & Engineering, Kongju National University

<sup>3</sup> School of Computer Engineering, Kongju National University, Korea

<sup>4</sup> School of Computer Engineering, Kongju National University, Korea

E-mail: <sup>1</sup>mtgood@naver.com, <sup>2</sup>ecgrpark@kongju.ac.kr, <sup>3</sup>musim-kum@hanmail.net, <sup>4</sup>merope@kongju.ac.kr

## ABSTRACT

With the development of IT technology today, many media content services are being used, and there are cases where they have to use both online and offline depending on the service type. However, there are various difficulties due to illegal distribution of content. In this paper, we study DRM (Digital Right Management) method which can run on a web browser that supports HTML5, which does not need to be installed separately, in order to prevent illegal distribution of media content. The existing DRM method is expensive, and since the encryption algorithm is not standardized even when it is operated off-line, it is difficult to use since a dedicated DRM viewer for each installation type have to be installed. To solve this problem, this paper proposes and implements a new DRM system using base64, an encrypted media source file, which can be used without installing separately in a web browser supporting HTML5.

**Keywords:** DRM, AES256, Base64, BLOB, Javascript

## 1. INTRODUCTION

Today, with the rapid advancement of IT technology, various media contents are being created and utilized. However, piracy, which is throwing a cold water in the development of such diverse content, is constantly harassing content producers. Piracy is widespread in all areas of content, whether it is publishing, music, movies, or games. In particular, piracy is a major threat to producers' livelihood in the digital content sector<sup>1</sup>. Analog piracy has the disadvantage of going through a cumbersome copying process and is of poor quality, but digital copying can be done relatively simple but the quality is the same as that of genuine products, and the spreading rate is very fast, which causes serious damage to the producers in material and psychologically. There is a strong and systematic anti-piracy technology called DRM, but there are various problems in servicing customers. The reason for this is as follows. First, it is possible to prevent piracy by using DRM technology. However, the introduction cost is high, and if the DRM is applied, the DRM exclusive

viewer program must be separately installed. Second, in an environment where the Internet is available, there are technologies to protect copyright by transmitting authentication codes to the authentication server for copyright protection<sup>2,3</sup>. However, in an off-line environment where the Internet is not available, if the content must be stored in the terminal, the corresponding DRM program must be installed for each content provider, and in all viewers that support WEB must be able to view the content. For this reason, there are limitations in applying DRM technology<sup>4</sup>, which is currently popular. To solve this problem in this paper, we propose and implement a new method of media content DRM system that enables viewers in all terminals that utilize web browser supporting HTML5, rather than the dedicated DRM viewer which is now popular.

## 2. Related Work

### 2.1 DRM

DRM (Digital Right Management) system is a limitation technology that allows only authorized users to access digital content. It is a system that

provides services for users to use safely and also protects rights and interests of copyright owners.

It is composed of a content packager that encrypts the file with the metadata of the content to prevent access by unauthorized users, and a license server that grants the user or authority to use the encrypted file on the device. And it performs the function of requesting permission to use to the encrypted file and DRM controller technology that decrypts the file only according to the granted authority and allows the user to use it <sup>5</sup>.

## 2.2 BLOB

A binary large object (BLOB) is a collection of binary data stored as an entity in a database management system. It is usually a picture, audio, or other multimedia object [3]. The purpose of the object is diverse. Among them, the method of accessing through the URL used in the WEB can grant a virtual URL to the object after the BLOB object is created. If you use this method, you can use JavaScript to call the `createObjectURL` method of the URL and transmit the BLOB or FILE object to create the URL as the first argument. Including BLOB, FILE objects that inherited BLOBs are composed of the same format as "Blob: 550e8400-e29b-41d4-a716-446655440000", which is a format that can be granted a URL. This URL is the same as a regular file, except that it is not actually a URL that exists on the server<sup>6</sup>.

## 2.3 Base64

The base64 is an encoding method that converts binary data to ASCII text or vice versa, and is one of the methods used by MIME. The base64 divides each 3 bytes of the original data into four 6-bit units so that it can be expressed as four 7-bit ASCII characters. This usually increases the file size by about 1/3 of the original size.

The background of its birth is that early e-mail was designed to handle only textual information. But as email has become widely used, it has been designed as part of Multipurpose Internet Mail Extensions (MIME)<sup>7</sup>, requiring the transmission of binary data such as images or attachment files. It also uses 64 ASCII codes (alphanumeric, upper and lower case letters, "+", "-") that are common to all platforms so that data is not broken or invisible on heterogeneous platforms. However, the platform does not support all ASCII codes<sup>8</sup>.

## 2.4 Javascript

JavaScript is an object-based script programming language. This language is mainly used in web browsers and has the ability to access

built-in objects of other application programs. JavaScript was initially named Mocha and originally developed by Brendan Eich of Netscape Communications Corporation and later, it was developed as LiveScript, and finally became JavaScript.

Although JavaScript is similar to Java and syntax of Sun Microsystems, it is because both are based on the basic syntax of the C language, and Java and JavaScript are not directly related. Beyond the name and syntax, there are many similarities with the self than Java. As of January 2013, the most recent version is JavaScript 1.8.5, which is supported in Firefox 3. The JavaScript version corresponding to the standard ECMA-262 3rd edition is 1.5. To put bluntly, an ECMA script is a standardized version of JavaScript.

As Mozilla 1.8 Beta 1 is introduced, it has been in part supported E4X (ECMA-357), an extension language corresponding to XML. JavaScript has a different version to each different browser, and the most commonly supported version is 1.5<sup>9</sup>.

## 2.5 AES

The Advanced Encryption Standard (AES) is a cryptographic scheme established by the National Institute of Standards and Technology (NIST)

AES allows the selection of three key lengths of 128, 196, or 256 bits, but the block length is suggested to be 128 bits.

The overall structure of AES is first, the Rijndael algorithm does not use the Feistel structure, and the entire data block is processed in parallel during permutation and permutation in each round.

Second, a 128-bit key given as an input is extended to four 32-bit words and four different words (128-bit) are used as round keys in each round.

Third, use four steps (Substitute bytes, Shift row, Mix columns, and Adround key) consisting of one permutation and three permutations.

Fourth, encryption and decryption both begin with a round-key addition phase, followed by nine rounds involving all four stages, followed by the tenth round, which includes only three stages (excluding heat blending).

Fifth, in practice, the Round Key step is not powerful by itself and provides chaos, spread and non-linearity in conjunction with the other three stages, but it does not provide security because it does not use keys.

The AES encryption is efficient and safe because it proceeds in the following order: a modified XOR operation of the block (addition of a

round key), a blending of blocks (byte substitution, row movement, and column mixing), and an XOR operation.

Sixth, in case of decryption, byte substitution, row movement, and column mixing steps use the inverse function, and round key addition step uses  $A + B + B = A$ .

Seventh, in most block cipher algorithms, the decryption algorithm is performed using the reverse order of the expansion key, but the decryption algorithm and the encryption algorithm are not the same. Both of them are composed of only three stages in the final round<sup>10</sup>.

### 3. THE PROPOSED SYSTEM

In this paper, we implemented a new method DRM for viewing media content in a web browser supporting HTML5. The following [Figure. 1] is the proposed system execution order.

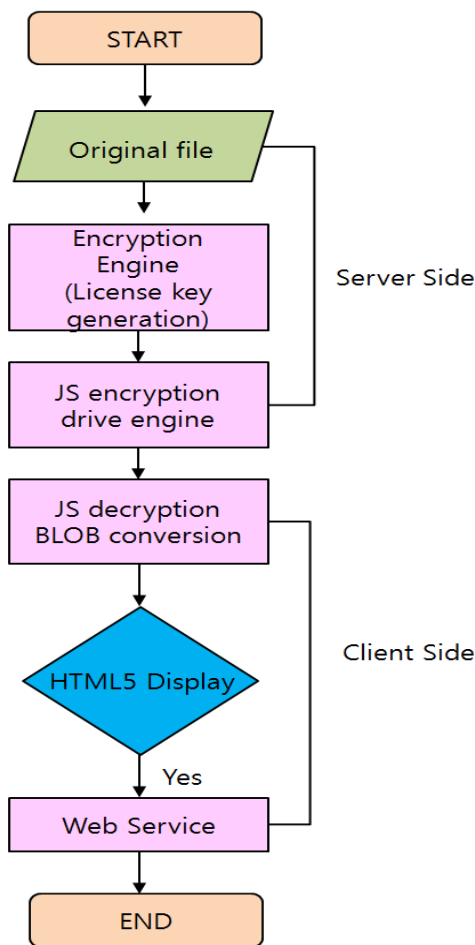


Figure 1: System execution order

First, on the computer where the server side original media file exists, an encrypted script file is created that can run a media file in JavaScript through the encryption engine.

Secondly, to prevent the tampering of the converted script, generate drive file and drive key with encrypted JavaScript.

Third, convert to Base64 and BLOB format so that location and decryption of the media file are impossible but able to run in HTML5.

The final step is to show the media converted to BLOB format in HTML5. Content converted to HTML5 can be used on PCs and mobile browsers with a web browser supporting HTML5 installed, and it runs both on-line and off-line.

Also, according to the condition of the encryption module, the code is inserted in the media file conversion process so that it can run only in the designated device, the designated domain, and the designated HTML. And the number of content viewers can also be designated, and if the number of times exceeds the specified number, the decode code of the content does not match and the content can no longer be viewed.

The following [Figure. 2] is a system configuration diagram implemented in this paper

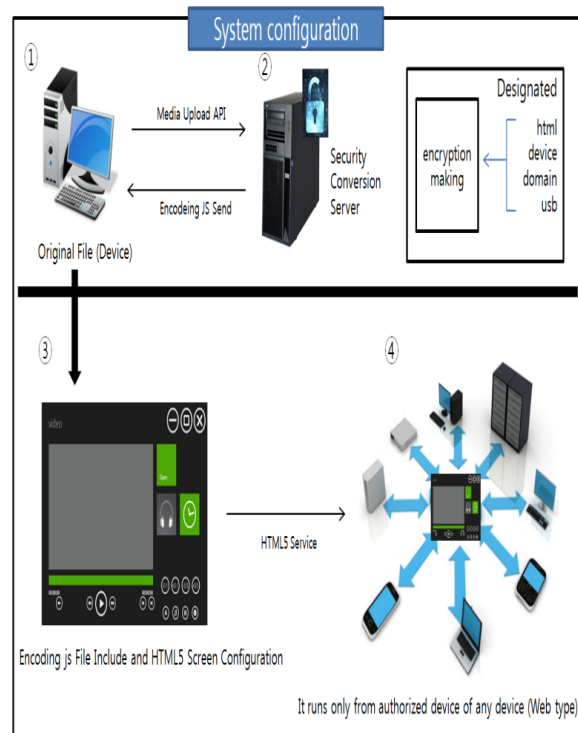


Figure 2: System configuration

The system in this paper is implemented so that files can be exchanged in API format like ① in order to convert multiple contents at the same time. When sending the API, send the original file and it converts it into the encrypted JS file from the security conversion server as shown in ②, and delivers it to the device. Also, it is generated as a file which can be run only in specific terminal and specific file according to API argument option, and is delivered to the device. If the converted file is configured for each service as shown in ③, it runs only by the device that meets the specific condition. The following [Figure. 3] is part of the source for file conversion.

```
String choice_1="",choice_2="";

if (args.length > 0) {
choice_1 = args[0];    //Conversion file
choice_2 = args[1];    //Original File
}
File file = new File(choice_2);
encode(file,new FileOutputStream(choice_2+".js"));

//base64 Conversion
String lastModified = file.length()+"";
String key="";

AES256Util en = new AES256Util();
key=en.aesEncode(lastModified);
key=key.substring(0,key.length()-2);
System.out.println("***License key
generation***"+key); //License key

last_key=base64_relocation(choice_2); //relocation of
base64 encryption
System.out.println("===last_key="+last_key);

byte[] temp = null;
emp = Files.readAllBytes(new
File(choice_2+".js").toPath());
String body = "var _DATA="+head_string+"/";
byte[] header = body.getBytes();
int alllen=temp.length+header.length;
byte[] data = new byte[alllen];
System.arraycopy(header, 0, data, 0, header.length);
System.arraycopy(temp, 0, data,
header.length,temp.length);
FileUtils.writeByteArrayToFile(new
File(choice_2+".js"), data);
System.out.println("success : ");
```

Figure 3: File conversion source

First of all, an argument is received to convert the file at the same time. choice\_1 is the name of the file where the converted file will be saved and the html source file is saved in choice\_2.

First, the original file goes through the conversion process to base64.

Second, generate a license key encrypted with AES256 for the original file. At this time, it is added to the license key according to the running condition. In this paper, we have generated an encryption key that can run only in a specific file.

Third, relocate the base64-converted file to base64 encryption by combining it with the generated run key to enhance security.

Fourth, the relocated file is generated as a SCRIPT file so that it can run in JAVASCRIPT.

Fifth, deliver the generated SCRIPT file to the DEVICE that sent the original file.

Sixth, you need to configure HTML5 with each file delivered and start servicing.

Following [Figure. 4] is part of a JavaScript function algorithm that reads a Base64 encrypted file, converts it to a BLOB, and decrypts the encrypted code.

```
_HEAD = _DATA.substring(0,_DATA.indexOf('/'));
_DEC = parseInt(_HEAD, 16)+"";
key_dim = _KEY.split("");
var last_key="";
for(i = 0; i < _DEC.length ; i++){

    num=parseInt(_DEC.substring(i,i+1));
    if(num>0)
    {

        chg=key_dim[num-1].charCodeAt(0);
        key_dim[num-1]=String.fromCharCode(chg+num);

    }
}

for(i = 0; i < key_dim.length ; i++){
    last_key=last_key+key_dim[i];
}

_DATA=
_DATA.substring(_DATA.indexOf('/')+1,_DATA.length
)

_DATA=
_DATA.replace(last_key,_DATA.substring(_DATA.leng
th-last_key.length,_DATA.length));
contentType = 'video/mp4';
```

Figure 4: Base64 decrypt

The main contents of the source are as follows.

At the first, it reads converted file to the base64 and decrypt the encrypted head part of the file again. The second, some of the content of Base64 is repositioned and the document converted is found in Base64 source, The third, it is relocated to a normal driving source what find in the base64 source.

Following [Figure. 5] Is a part of the JavaScript Function algorithm which reads a Base64-encrypted file and converts it to a BLOB to prevent reinterpretation of the source.

```
blob = Base64toBlob(nkbData, contentType);
blobUrl = URL.createObjectURL(blob);
document.getElementById(item).src=blobUrl;

function Base64toBlob(nkbData, contentType,
sliceSize) {
  contentType = contentType || "";
  sliceSize = sliceSize || 512;

  var byteCharacters = atob(nkbData);
  var byteArrays = [];

  for (var offset = 0; offset < byteCharacters.length;
offset += sliceSize) {
    var slice = byteCharacters.slice(offset, offset +
sliceSize);

    var byteNumbers = new Array(slice.length);
    for (var i = 0; i < slice.length; i++) {
      byteNumbers[i] = slice.charCodeAt(i);
    }

    var byteArray = new Uint8Array(byteNumbers);

    byteArrays.push(byteArray);
  }

  var blob = new Blob(byteArrays, {type:
contentType});
  return blob;
}
```

Figure 5: Convert Base64 to BLOB

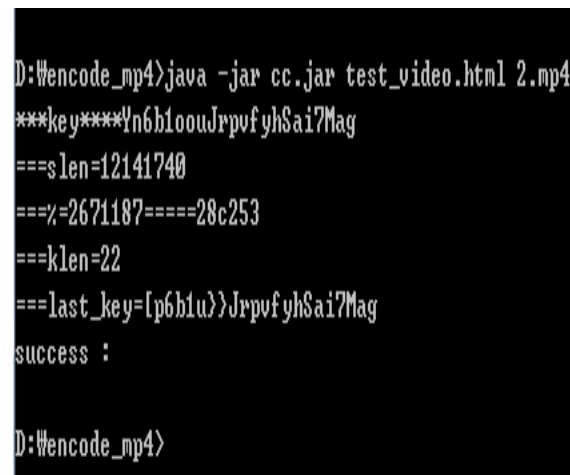
The original source encoded in Base64 is a source that can be easily retrieved from the Web and thus converted to a BLOB to prevent acquisition of the original source.

The source converted to BLOB is composed of URLs that can not be accessed from source view which is one of the functions of web browser. When re-reading the page of the web browser's URL, the URL was changed again to fix the web security vulnerability.

#### 4. EXPERIMENTAL RESULT

The source converted to BLOB is composed of URLs that can not be accessed from source view which is one of the functions of web browser. When re-reading the page of the web browser's URL, the URL was changed again to fix the web security vulnerability.

The proposed system in this paper is implemented on JAVA basis and the sample file is implemented by selecting a video file, mp4. The following [Figure. 6] is part of the screen that operates by converting the original file.



```
D:\Wencode_mp4>java -jar cc.jar test_video.html 2.mp4
***key***Yn6h1oouJrpfyhSai7Mag
===slen=12141740
===%2671187====28c253
===klen=22
===last_key=[p6h1u}>JrpfyhSai7Mag
success :

D:\Wencode_mp4>
```

Figure 6: Execute file conversion

The license key was generated by combining the HTML file (test\_video.html) and the original media file (2.mp4) file so that it can run only on a specific html. The mp4 file that converted last to the value of "last\_key" generates a Script file as shown in the [Figure. 7].

```
28C253/AAAAGGZ0EXBPC29TAAAAAWLZB21
HDMMXAABZC21VB3YAAABSXZOZAAAA
ADPFLH7Z35YEWAAALGAAMDMAAEAAAE
AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAQAAAAAAAAAAAAAAAAAAQ
AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAMAAAVAW9KCWAAAA
```

```
AQBWBP//8PFF8AAC3LDHJHAWAAAFX0A2H
KAAAAAC9+WHVPFLH7AAAAAQAAAAAA
MDWAAAAAAAAAAAAAAAAAAAAAAAAABA
AAAAAAAAAAAAAAAAAAAAAAAAQAAAAAA
AAAAAAAAAAQAAAAAQAAAAACQAAAAAA
TGW1KAWEEAAAAGBWROZAAAAADPFLH7Z
35YEWAADTAJU79VCQAAAAAAHHOZGX
AAAAAAAAAB2AWRLAAAAAAAAAAAAAA
AAABWN0ZW1WOTCZNBINJK0NWY2NGZM
ZS4YNJQJDMKZ86ZNBZPTI5LJK3ONBHCJ
0XNJXNSATIELTCG9YDGVKIHDPDGGGR1B
BQYAWLJUUMCIYZXY0MDY1AAAALOFTA
W5MAAAAFHZTAGQAAAAABAAAAAAAAAA
AAAAAKZGLUZGAAABXKCMVMAAAAAAA
AAEAAAAMDJSIAAAAAEAACYHC3RIBA
AAAL1ZDHNKAAAAAAAAAAAAEAACTYXZJ
MQAAAAAAAAABAAAAAAAAAAAAAAAAAA
AAAAPAAKAASAAAAEGAAAAAAAAAAQA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABJ//WAAABBWYXNWAA
AAEAAAAA8AAAAZYXZJQWFCWCJ/4QABZ0
LAKNSDWEM/8AEAAPEAAAMAZAAAF2QPG
DLGAQAFAMQHIAAAAAUYNRYDAABTNE
ALZWGAAWLCAAAABHZDHRZAAAAAAAA
AAEAAAN1AAAD6QAAAMBZDHNZAAAAAA
AACWAAAABAAAAPQAAAHKAAAC1AAA
A8QAAAS0AAAFUAAABIGAAACYAAAICAA
ACPGAAAN0AAAK2AAAC8GAAAY4AAANQ
AAC.....
```

Figure 7: Base64 conversion file

The generated file is not a basic BASE64 file but a result of the changed structure and a combination of encryption algorithm.

The following [Figure. 8] is a typical BASE 64 translation form.

```
/9j/4AAQSkZJRgABAQAAQABAAD/2wCEAA
kGBxMTEhUSExMWFhUXGBgZGRcYGBcYGH
gZGBsdGBcfGBoYHSggGh0IGxgZITEhJSkrLi4u
Fx8zODMtNygtLisBCgoKDg0OGxAQGy8IICUtl
S0vLS0tLS0tLS8tLS0tLS0vLS0tLS0tLS0tLS
0tLS0tLS0tLS0tLS0tLf/AABEIALoBDgMBI
gACEQEDEQH/xAAcAAABBQEBAQAAAAAA
AAAAAAAAAwQFBgcCAQj/xAA+EAABAWiE
AwUGBAYBBAMBAAABAAIRAYEEE.....
```

Figure 8: Base64 Standard file

The BASE64, which is inserted with the encryption key as shown in the general BASE64 and [Figure. 7], has a **28C253** code inserted in its head section to

prevent the BASE64 from being decrypted. How to create a key depends on the different conditions.

First, create a key that will only work with that HTML file.

Secondly, create a key that will only allow content to work for the duration you want.

Thirdly, create a key that makes the content work at the designated number of times to view it.

Fourth, create a key that enables content to function only in a particular domain.

Fifth, generate a key that only makes it operable on a particular device (PC, MOBILE)

All of these key generation methods are designed to be operational in a manner that is operational at Local without the need to install a separate DRM viewer file.

In addition, the code location of BASE 64 was changed with the corresponding key and the random code was inserted. The code is also decrypted with the key used to Decoding from HTML5 according to the conditions.

In addition, the JavaScript file, which consists of JS files, is the same structure as [Figure. 4], where anyone can analyze the sources. To solve this problem, To solve this problem, we have incoded the JavaScript file decoding.js as shown in [Figure. 9] below so that the JS file can not be decrypted as well.

```
eval(function(p,a,c,k,e,d){e=function(c){return
c};if(!".replace(/~/,String)){while(c--
){d[c]=k[c]|c;k=[function(e){return
d[e]}];e=function(){return"\w+";c=1};while(c--
){if(k[c]){p=p.replace(new
RegExp("\\b'+e(c)+'\\b','g'),k[c]}}return
p}('25=3.13(0,3.27(\\^));22=30(25,16)+\\^;11=32.36(\\^);
5
10=\\^;15(2=0;2<22.4;2++){9=30(22.13(2,2+1));31(9>0)
{28=11[9-1].29(0);11[9-
1]=34.35(28+9)};15(2=0;2<11.4;2++){10=10+11[2]}3=
3.13(3.27(\\^)+1,3.4)3=3.37(10,3.13(3.4-
10.4,3.4));6=\\33/46\\;14=26(18,6);23=53.49(14);48.51(5
2).50=23;38 26(18,6,8){6=6|\\^;8=8|47;5 17=41(18);5
21=[];15(5 7=0;7<17.4;7+=8){5 12=17.12(7,7+8);5
20=19 40(12.4);15(5 2=0;2<12.4;2++){20[2]=12.29(2);5
24=19 39(20);21.42(24);5 14=19 43(21,{45:6});44
14}',10,54,||i|_0|length|var|content|Type|offset|slice|Size|n
um|last_key|key_dim|slice|substring|blob|for|byteCharact
```

```
ers|nkbData|new|byteNumbers|byteArrays|_1|blobUrl|byt
eArray|_2|Base64toBlob|indexOf|chg|charCodeAt|parseIn
t|if|_3|video|String|fromCharCode|split|replace|function|U
int8Array|Array|atob|push|Blob|return|type|mp4|5|12|docu
ment|createObjectURL|src|getElementById|item|URL'.spl
it('|',0,{}))
```

Figure 9: Javascript conversion encoding

[Figure. 9] is implemented to prevent leakage of media contents by configuring [Figure. 4] and [Figure. 5] so that it is difficult to decode source code with converted JavaScript.

The following [Figure. 10] is the result screen showing with the converted file from the web browser.



Figure 10: Execution result

It is converted to blob so that it cannot be reinterpreted with HTML5 source. It is a URL that does not actually exist, and the URL is changed every time the screen is reloaded to protect the original content.

Following [Figure. 11] is a screen generated as a file operated by HTML5.

source.js	2017-07-05 ...	JScript	11,858KB
decodeing.js	2017-06-21 ...	JScript	2KB
license.js	2017-07-05 ...	JScript	1KB
test_video.html	2017-07-05 ...	HTML	1KB

Figure 11: List of generated drive files

At the First, the test\_video.html file is an HTML5 file that constitutes a screen, and is a file configured to load a js file composed of basically JavaScript and display it on the screen.

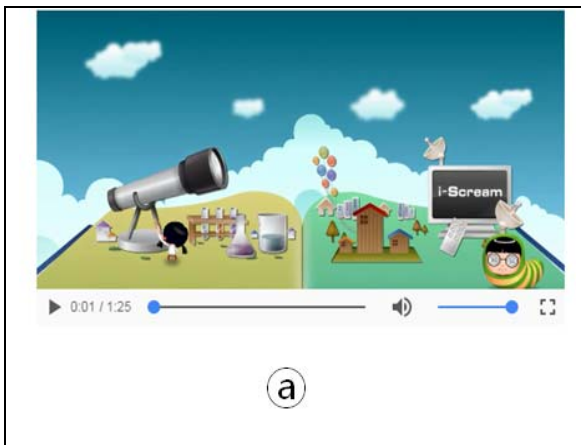
The second, the source.js file is a Base64 file that is encrypted and transformed a video originally.

The third, the decoding.js file decrypts for the encrypted source file after loading it. However, the decrypted file is a JavaScript file that is converted to a BLOB so that the original source can not be confirmed because be changed by the user.

The fourth, the license.js file is a license key generated for each video file.

In addition, it is the key used to recover the modified file from the decoding file and is the file that not working a normal operation if a change is made to the drive file.

Following [Figure. 12] is a screen assuming that a video has partially hacked of the deformed Base64



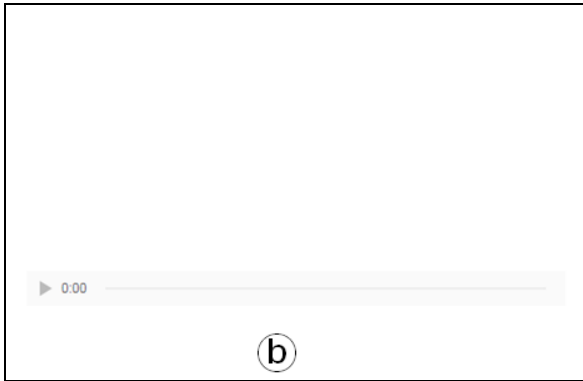


Figure 12: Screen after penetration testing video

[Figure 12- a] is a video of a document file written with modified code by normal HTML5.

[Figure 12- b] is a screen that does not operate normally because between the license key and decoding value do not match when file contents are partially changed due to penetration testing that is called simulation hacking.

In this paper, the license key is designed to operate normally without changing the file structure. If you change the file arbitrarily, the screen will not be displayed, or some contents will be displayed, and then it will stop in the middle.

Following [Figure. 13] is a screen assuming that the image has partially hacked of the deformed Base64.

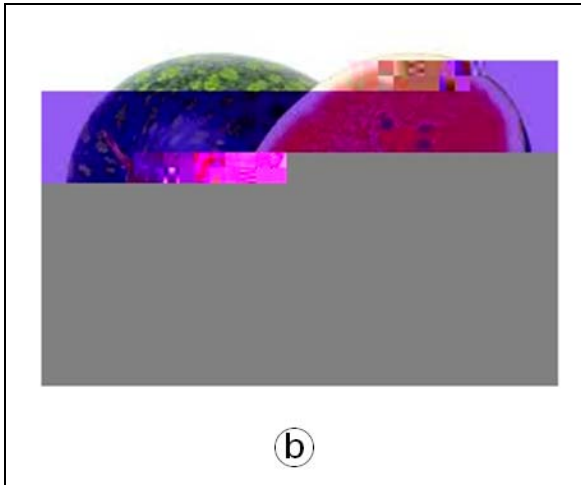
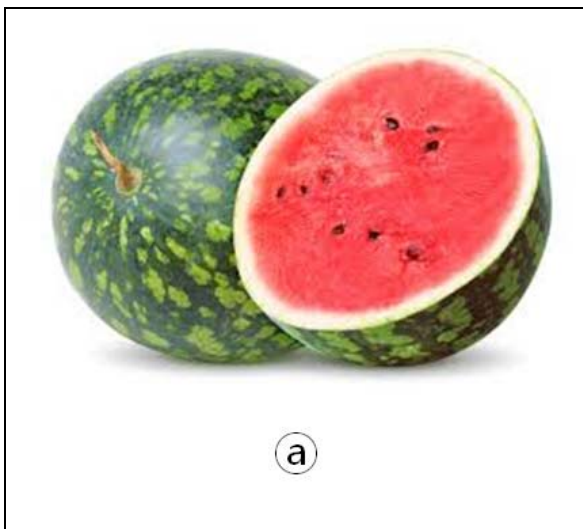


Figure 13: Screen after penetration testing image

[Figure 13- a] is an HTML5 image tag that is transformed into a normal code.

[Figure 13- b] is a screen that does not operate normally because between the license key and the decoded value do not match when the contents of the file are partially changed due to penetration testing that is called simulation hacking.

If the file structure is changed by hacking, some images may appear to be broken or some images may not be seen at all.

## 5. CONCLUSIONS

With the advancement of IT technology, various media contents are being countlessly generated. However, it is difficult to provide service due to indiscreet piracy.

In this paper, to solve this problem, we proposed a DRM that can protect new media content that can support in off-line while supporting a web browser supporting HTML5.

In addition, in the conventional method, only the dedicated DRM viewer can be used to view the media content. However, it is confirmed that the media content protection model implemented in this paper can carry out the service without difficulty as long as you have a web browser that supports HTML5.



In the future study, we will study the conversion technologies that do not increase through a new compression algorithm for solving the problem of a file that increases by 30% when converting to BASE64.

#### REFERENCES:

- [1] Kim Jae Wan, "A Study on the Illegal Use Regulation of the Works in Cyberspace : Focused on the Illegal Use Regulation of the Films on the Internet", The Journal of Image and Cultural Contents. Vol.5, pp.97-126(2012).
- [2] Donghyun Choi, Yunho Lee, Hogab Kang, Seungjoo Kim, Dongho Won, "A Secure License Sharing Scheme for Domain DRM System Against Replay Attack", Korea Institute Of Information Security And Cryptology, pp.97-101(2007).
- [3] Duk-Kyu Lee, Hee-Un Park, Im-Yeong Lee, "A DRM Model for Illegal Copyrights Protection based on Agent", Korea Information Science Society, pp.682-684(2001)
- [4] Jin-Kyoung Heo, "Distributed Security for Web Application Contents Protection", Digital Contents Society, pp.125-130(2008).
- [5] Eung Sup Jun, "An Efficient Application of eBSS DRM Method to eBook Contents based on ePub 3.0 for Smart Device", The Korean Society Of Computer And Information, pp.59-72(2016).
- [6] Giho Choi, Hyunchul Kang, "A Scheme for Classifying Assorted Types of BLOBs for Multimedia Data Management", Korea Information Science Society, pp.73-76(1993).
- [7] Byung-Do Lee, Seok-Gyu Kang, "Web Publishing Young-Suk Chung, " A study on anti-piracy model of WEB-based educational media contents ", The Korean Society Of Computer And Information, pp.15-18(2017).
- [8] Ki-Bok Nam, Koo-Rack Park, Joon-Yong Kim, Young-Suk Chung, " A study on anti-piracy model of WEB-based educational media contents ", The Korean Society Of Computer And Information, pp.15-18(2017).
- [9] Hongki Lee, Junho Jin, Sooncheol Won, Junhee Cho, Sukyoung Ryu, Yoonseok Ko, " SAFE: Scalable Analysis Framework for ECMAScript ", Korea Information Science Society, pp.283-289(2013).
- [10] Jeong Woo Cha, Chang Hoon Kim, " Design of FPGA Hardware Accelerator for Information Security System", Korea Society of Industrial Informantion Systems, pp.1-12(2013).