ISSN 2278-3091
Volume 8, No.6, November – December 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse99862019.pdf
https://doi.org/10.30534/ijatcse/2019/99862019

360 Video Protection and Streaming

Nikolai Valerievich Bradis[1], Sergei Sergeevich Smirnov[1], Dmitrii Aleksandrovich Sytnik[1], Andrei Ivanovich Kardakov[2], Alexey Yuryevich Lebedev[3]
[1]Complex Systems LLC, Tver, Russia
[2] Intelligent Solutions LLC, Tver, Russia
[3] Tver State Technical University, Tver, Russia

ABSTRACT

Video distribution systems require for protection of content against unauthorized access and copying. At present some approaches are available based on DRM (Digital Rights Management) which provide acceptable level of content protection against malicious users but do not protect content against copying by rightful users. The problem of video content protection against rightful users is essentially nontrivial, since video playback assumes possibility to read data and actually enables copying. Complete protection against rightful users is possible only in the case of complete control over their playback units and this is impossible at present. Commercial solutions to protection of 360 video are unavailable due to numerous types of playback units which do not provide such protection at their side. This work is aimed at development of protection of 360 and 360 3D video content on units supplied by content producers to final users.

Key words : Virtual reality, DRM, 360 video, AES128, H.264, HEVC, OpenGL, FFMPEG, GPU, Shader program, Analog hole, Madgwick filter.

1. INTRODUCTION

Presently modern video streaming services apply various protocols of content distribution in combination with DRM (Digital rights management) [1]. DRM is supported by such video streaming protocols such as RTMP (Real Time Messaging Protocol) [2] and MPEG-DASH [3]. The essence of DRM is that a customer receives the link to encryption key together with video stream from media server and then requests the key by this link. DRM server checks connection with the customer (HTTPS protocol) and decides whether to transfer the key for decryption of the stream. The main problem of such mechanism is that after transfer of the encryption key to the customer, this key can be compromised and the content can be captured. In addition, HTTPS vulnerabilities are also an issue.

When a customer receives not the stream but, instead, total encrypted video file, the flowchart generally remains the same and the customer receives a key for content decryption. The main issue not only of DRM but of any protection mechanisms of audio and video content is the necessity to transfer encryption key to customer, which requires for ппrotection of this key.

Modern DRM systems use strong algorithms (AES, RSA, and others), however, these methods do not provide comprehensive protection, since a secret key is required to get access to protected content. Therefore, rightful owners of access to contents can bypass the protection, since they have both encrypted data and the key. Due to this reason the DRM systems attempt to hide the used encryption keys from customers, which is rather difficult because modern playback units (PC and mobile devices) are sufficiently versatile and are controlled by users.

Permission to playback content and at the same to protect it against copying is a very difficult problem, since playback, in fact, is data reading, processing, and recording to output device. This differs from storage only by the fact that recording is carried out not on storage device but on output device. Thus, the problem appears known as analog hole [4]. Therefore, efficient protection against copying can be achieved only in the case when playback unit is under complete control of right holder or content distributor.

Due to low reliability of DRM protection and to problems preventing content playback by rightful customers, many companies presently do not apply such type of protection. For instance, Apple Inc. completely removed DRM protection in its iTunes service, German service Musicload also refused to apply DRM, since about three quarters of requests to support service were related with the customer problems due to DRM.

In the case of 360 and 360 3D video formats, no valid DRM solutions were presented, since the main playback units for 360 video are VR glasses [5] both together with either PC or smartphone and with standalone device. Video playback in such devices is performed using specialized players accounting for spatial position of customer head by means of various tracking mechanisms. Copying of 360 video via analog hole is complicated by the fact that it is required to

capture video in two positions (to obtain two hemispheres), to project spherical video onto plane, and to stitch the two videos.

A critical issue for 360 video is the playback speed. Playback of dynamic content at the speed of 60 frames per second (FPS) can be inconvenient for a spectator. In addition, high resolution (UltraHD, 4K and higher) is important for 360 video.

Several problems are solved upon playback of 360 video:

1. Decoding of video packets by H.264 [6] or H.265 [7] decoders,
2. Subdivision of frames for right and left eyes (for 3D),
3. Projection of the obtained frame onto internal sphere surface,
4. Sphere rotation according to readings of tracking systems,
5. Visualization of the obtained spheres for each spectator's eye.

Upon playback of protected by encryption content, this list is supplement by data decryption, which can additionally impair performances of playback system.

Presently various players are available for playback of 360 video. Most of these players are supplied with VR headsets and allow to playback video in combination with these headsets. There are ready SDK and libraries allowing to playback 360 video (for instance, Oculus Platform SDK [8]), however, these tools are mainly intended for operation with specific positioning devices and do not provide access to reading and processing of video file or stream at the level sufficient for content protection. In this regard it became necessary to develop proprietary low-level player for playback of 360 and 360 3D video.

2. METHODS

A. Content encryption methods

The existing DRM systems usually operate with symmetric encryption algorithms, such as AES, GOST, Blowfish, and others. RSA and Elgamal asymmetric encryption algorithms are also used. These algorithms are tested by time and cryptographic experts, therefore, it would be unreasonable to develop new encryption algorithms since verification of their strength could be time consuming. The most popular encryption algorithm is AES (Advanced Encryption Standard) [9]. This algorithm was approved in 2001 by National Institute of Standards and Technology after five years of testing. In addition to confirmed encryption strength, this algorithm is characterized by high efficiency which provides rapid data encryption and decryption, and upon its application for video content protection, this is the main factor following reliability.

Application of asymmetric encryption has no sense for content encryption, since the content is encrypted by public key and decrypted by private key. Provided that video should be played on client devices, the private key should be transferred to them (or to be generated on these devices). In fact, this provides possibility to protect server against user content decryption which has not sense since the server stores content encrypted with secret server key.

Encryption of video file is reduced to encryption of constituent video packets. Encryption of audio packets is reasonable only when they have commercial value without appropriate video data. Herewith, only data compressed by codec (H265, H264) should be encrypted, this can be substantiated by two reasons.

The first reason is that after encryption of the initial image, we obtain disconnected data set and the quality of compression of these data by codec will be impaired. Herewith, after compression the data can be modified depending on codec settings and compression quality, and not the initially encrypted data will be obtained after unpacking which can lead to problems upon decryption.

The second reason is that after compression there are less data in video packet in comparison with initial image, which is accompanied by less amount of work for encryption algorithm and, as a consequence, higher efficiency.

Upon testing encryption algorithms using OpenSSL library [10], the best performances were obtained with AES algorithm. Decryption of one 3880×1440 video packet encrypted by AES algorithm with 128-bit key requires in average 7 ms using Core i5, 9th generation. Therefore, per one second of using this equipment it is possible to decrypt 142 frames. If it is required to playback 4K video at the speed of 60 fps, these performances cannot be acceptable, since in addition to decryption, it is required to unpack video packet by decoder and to present the obtained image to the screen. In this regard it is sufficient to encrypt only key video packets, since all other packets contain only variations in respective frames with regard to the key packet. This is the principle of video file compression.

The number of key packets in video file does not generally exceed 3% of total number of packets, and most often it is below 1%.

While encrypting only key packets, it can be safely assumed that complete image recovery without key frames is so difficult that the respective labor consumptions are comparable with those for complete redoing of this video. That is, reasonability of such breach is completely zeroed. Upon encryption of only key packets, it is possible to reallocate resources for solution of the problems directly related with video playback.

Therefore, content protection can be based on AES128 algorithm with encryption and decryption of only key video packets irrespective of method of video transfer to client.

The main problem of this method and any other DRM system is that it is required to transfer secret key to client and to reliably protect it both on server side and client side. Under ideal conditions, the content integrity should be provided

even in the case of unauthorized access to distribution server. That is, illegal intruder obtaining access to distribution server should not have possibility to obtain video materials in unprotected form.

B. Server-side video protection

It is required to protect encryption key at all levels, otherwise, irrespective of the method used by illegal user to obtain the secret key, all content encrypted with this key will be lost. Rightful client of distribution server should also be considered by the system as a potential illegal user. Server administrator and any person having direct access to it should be considered by the system as potential illegal users. Hence, video materials and encryption keys should be protected not only on client side but on server side as well. In order to eliminate possibility to capture encryption key during its transmission over the network, it is required to provide key generation and transmission only once over protected communication line, or when playback unit is not transferred to client. The latter is possible only with B2B distribution system, that is, the clients of distribution server are auctions or VR streaming services and not final users' devices.

First of all, all video materials should be stored on the server in encrypted form. That is, upon uploading any video to distribution server, it should be automatically encrypted and stored only in encrypted form. Then, the content on the server should be encrypted only in separate subsystem based on compiled programming language using obfuscation, anti-debugging, and protection against code injection. It is prohibited to store user encryption keys in open form in the server database. In addition, upon calling encryption subsystem, it is prohibited to transmit to it encryption keys in open form. The encryption subsystem of the server should not have external interfaces of access to video decryption methods. Access to content encryption key and users' secret key should be provided only to encryption subsystem without possibility to withdraw it.

Encryption subsystem for protection of video materials upon their uploading is illustrated in Figure 1.

Encryption subsystem is aimed at creation of binary file which will operate in two modes. The first mode is to obtain open data from distribution server, their encryption with secret key, and storage of encrypted data according to server specified path. The second mode is intended for user secret key importation from distribution server, location of data to be re-encrypted and path to store the data encrypted with user key. In this mode the encryption subsystem should decrypt data in portions, encrypt them with user encryption key after its preliminary decryption, and save the data encrypted with user key at server specified path. Herewith, the user key should be decrypted every time dynamically immediately before re-encryption of next data portion and stored during minimum time required for decryption only in process dynamic memory. Herewith, the key for decryption of user secret keys should not be mentioned in explicit form in initial code of encryption subsystem, it should be dynamically assembled according to predefined algorithm using obfuscation immediately before decryption of user secret key. The same is applied to content encryption key for storage on the server. Data portions should be decrypted in random order with subsequent (after encryption with user key) recording into the required file segments. Decrypted data portions should be stored in process memory only during the time required for their encryption.

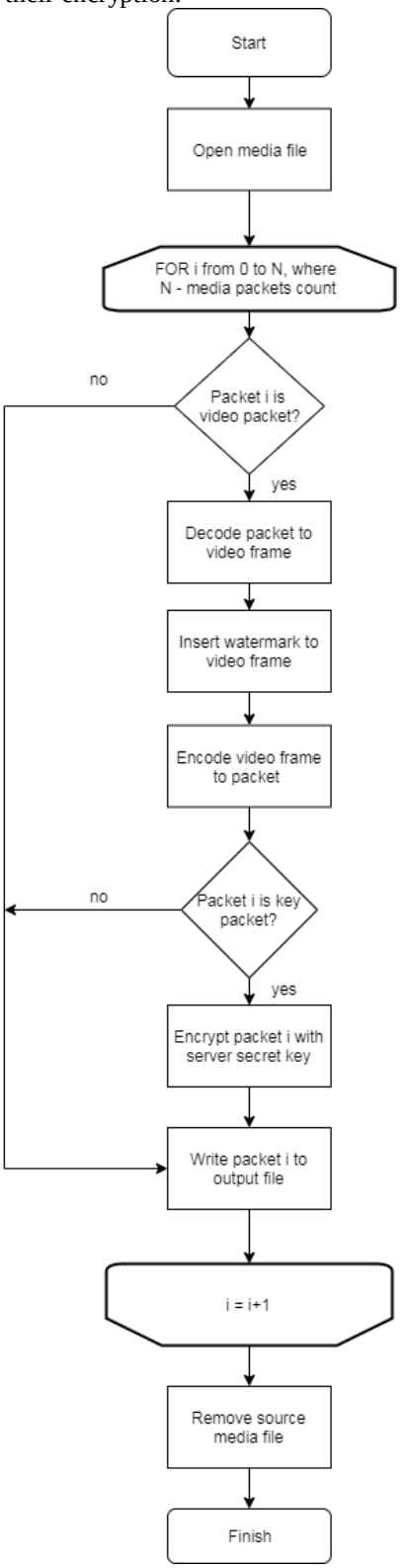

Figure 1: File encryption upon uploading

C. Client-side protection of encryption key

Protection of encryption key content against rightful owner is the most complicated task. Upon content distribution to business clients, it is possible to organize efficient protection by providing them with hardware key, for instance, USB-key, Guardant line [11]. In this case the distribution system operates via media streaming server which is equipped with the required software to control VR simulator and content. User sets are connected to this server, whether these are stand-alone sets (Oculus Go, Oculus Quest), or sets operating in combination with smartphones (Samsung Gear VR), or sets connecting to external PC (Total Vision, Samsung Odyssey, Oculus Rift). The process of simulator activation and video streaming to user sets is controlled from media streaming server. Therefore, it is possible to write encryption key into protected memory cell of hardware key and to use it for content decryption. Data from the key will be read by the server software and transmitted to the player of certain set over encrypted channel of local network.

Two serious vulnerabilities exist in such key scheme. The first is that the data from protected memory cell of hardware key are read using API methods of manufacturer library of this hardware key. Such inquiry can be captured by illegal user provided that debugging measures are cracked. Therefore, it is prohibited to store encryption key in open form even in hardware key, that is, secret user key should be written into hardware key only in encrypted form. Herewith, the encryption key of user secret key, similarly to encryption subsystem of distribution system, should not be written in initial code in explicit form but be assembled according to specified algorithm using obfuscation immediately before decryption of user secret key. The second vulnerability occurs due to transmission of user secret key via local network of media streaming service or simulator. This problem can be solved by two methods: the first is to use network protection keys, the second is to use SSL sockets for data transmission via TCP/IP for channel protection by asymmetric encryption. It should be mentioned that the key is transmitted to devices in encrypted form, since it is stored in hardware key and is decrypted immediately before video decryption.

Obviously, the same requirements to protection of encryption ley are applied to software player. Herewith, Java language should not be used for Android, since the mechanism of its disassembling and debugging is widely known and allows to obtain higher level code (in comparison with assembler code), thus strongly simplifying application cracking. It would be more reasonable to use C++ with the same anti-debugging mechanisms as in encryption subsystem of distribution server.

D. Video packet decryption

All key video packets in video file are preliminary encrypted by distribution server with user secret key. Encryption of key video packets by AES128 is illustrated in Figure 2.

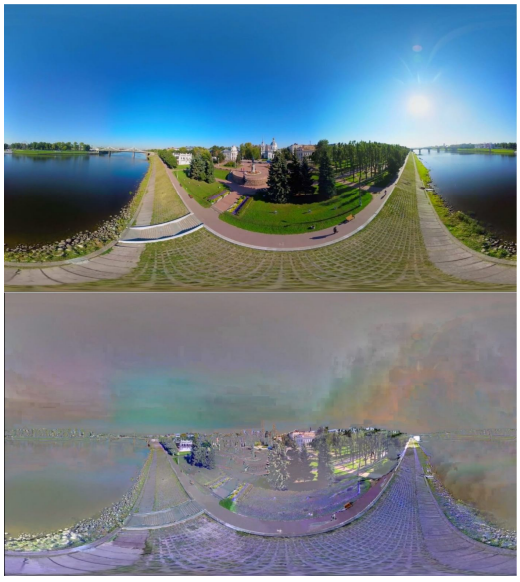

Figure 2: Playback of video file without encryption (at the top) and with AES128 encrypted key packets (below).

Therefore, receiving of video packets before their decryption without customer secret key is impossible.

For further video playback it is required to decrypt only key packets. The decryption can be performed by CPU and not involve GPU which is used for video decoding and presentation. Since the key packets in video are large, it would be reasonable to decrypt packet by several CPU cores, each would decrypt certain packet portion. Packet reading and decryption are the initial steps to playback 360 video.

E. Video decoding

Video decoding is the most complicated and resource consuming stage. H264 and H265(HEVC) are the main video codecs used for compression of 360 video. Numerous variants of coders and decoders are available for these codecs, including those supporting hardware acceleration comprised of video decoding by GPU, completely or partially.

In general case, the player to be developed for 360 video should unpack video on any device where it could be installed. Herewith, if such device has hardware tools for video decoding, they should be used at full scale in order to achieve the required efficiency for playback of high-resolution video. The most popular solution for operations with video is Libav library [12] in FFMPEG packet [13], distributed under LGPL 2.1 license. This packet allows to operate with all popular video coders and decoders, including hardware. In terms of playback, we are more interested in decoders, however, the distribution server also should be equipped with coders, since video compression after watermark embedding should be performed by the server.

Hardware acceleration of decoding in most cases allows to obtain image in GPU memory with pixels in various formats (depending on the used decoder). Conversion of pixels into

suitable format by means of OpenGL library [14] can be carried out using sws_scale method of FFMPEG packet, however, only when the obtained image is in random access memory. In order to transfer the obtained image from GPU memory to RAM, it is required to apply also av_hwframe_transfer_data method of FFMPEG packet. Scaling and conversion of pixel formats by sws_scale methods are carried out at CPU. As a consequence, excessive work is performed which wastes all advantages of hardware decoding. Data transfer from GPU memory to RAM occupies CPU resources and consumes much time, moreover, displaying finally is performed by GPU, that is, upon visualization, the data should be retransferred to GPU memory. FFMPEG packet has no embedded mechanism to bypass this problem, hence, at this stage (after image decoding) it is required to apply methods and libraries supplied by developers of hardware acceleration, namely: CUDA for hardware acceleration on NVidia cards, Android Media SDK for hardware acceleration in Android, DirectX for hardware acceleration in AMD video cards in Windows. In addition, pixel format can be converted using fragment shaders. Fragment shader for image conversion from YUV420P to RGBA is exemplified in Figure 3. Application of shaders reduces processing time by using GPU cores.

```
uniform sampler2D _textureY;
uniform sampler2D _textureU;
uniform sampler2D _textureV;

varying vec2 v_texcoord;

void main()
{
    float y=texture2D(_textureY, v_texcoord).r;
    float u=texture2D(_textureU, v_texcoord).r-0.5;
    float v=texture2D(_textureV, v_texcoord).g-0.5;

    float r = y + 1.13983*v;
    float g = y - 0.39465*u - 0.58060*v;
    float b = y + 2.03211*u;

    gl_FragColor =vec4(r,g,b,1.0);
}
```

Figure 3: Code of fragment shader intended for conversion of image pixels from YUV420P to RGBA.

Image scaling for resolution of playback unit is not required at this stage. It is possible to combine this process with displaying and to consider the scale upon imposition of texture with image onto the sphere.

Finally, at the stage of reading and decoding video packets, it is possible to use only FFMPEG, and further data processing and visualization will depend completely on hardware acceleration.

Efficiency is also an issue at this stage, since the decryption time of key packets can lead to delays upon visualization. The thread of reading and decryption should transfer not a single packet for unpacking but to fill packet buffer in advance with the thread of decoding and visualization, thus smoothing rate irregularities of packet preparation for decoding.

F. Visualization and positioning (tracking)

In order to visualize 360 and 360 3D video in VR glasses, it is required to superimpose consecutively video frames onto internal surface of sphere for each eye. Herewith, viewing position should be in the center of this sphere to allow viewing of a half of internal surface of sphere at any specific time. Herewith, upon frame visualization of panoramic 360 video, it is sufficient to project one and the same image onto both spheres, and upon 3D visualization of 360 video, each frame has its image. Processing in virtual reality is generally supported by “over and under” format where images for each eye are positioned one over the other. Such 360 3D frame is exemplified in Figure 4.

The same video upon playback in player for VR glasses is illustrated in Figure 5.

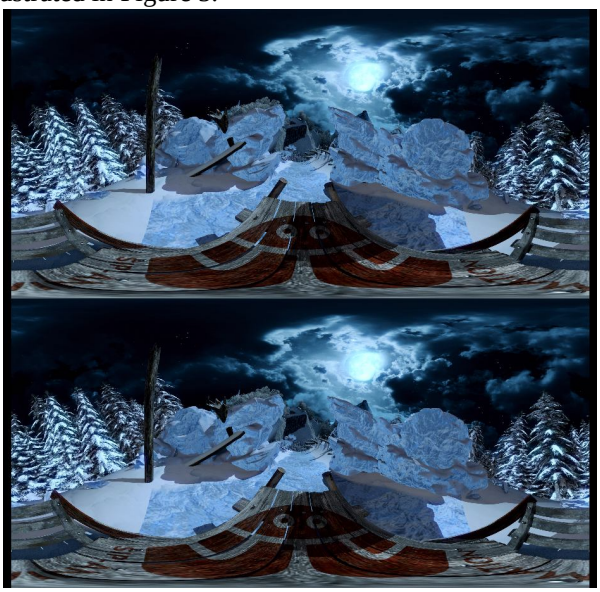

Figure 4: A sample of 360 3D video frame.

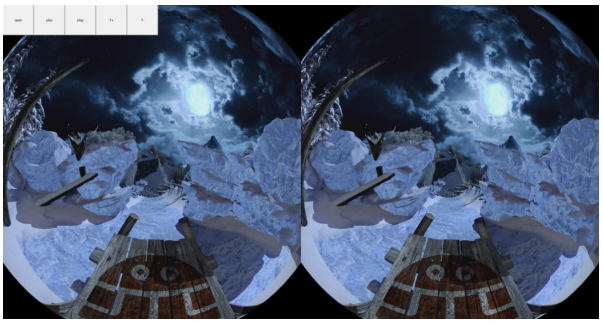

Figure 5: A sample of 360 3D video upon playback in player for VR glasses.

The sphere itself is a set of vertex coordinates. The step between the points determines the sphere resolution. The higher is the step, the lower is the sphere resolution and vice versa. The higher is the sphere resolution, the smoother is the image imposition onto the sphere. For high quality, visual perception without loss of efficiency, the step of 2° is sufficient.

In order to visualize sphere at required angle, it is possible to use vertex shader which allows to predict screen position for each sphere vertex with accounting for transformation

matrix determined by spatial position of VR glasses. The texture based on image frame can be superimposed onto sphere using fragment shader [15]. Shader software makes it possible to increase visualization rate by performing the same procedures for different vertices in parallel using cores of GPU.

G. Tracking

Most VR headsets are equipped with proprietary developer kits. For instance, Samsung Gear VR and Oculus Go are supported by Oculus Mobile SDK. This developer kit includes VrApi library containing all methods required for positioning.

The issue of positioning for available in market VR glasses is solved by presented API methods. For Total Vision headset this software (SDK) is presently unavailable. In order to interact with this headset, it was required to develop positioning software and to arrange it in the form of library in order to provide possibility of interaction between third party developers and this headset.

The Total Vision headset is equipped with ICN-20602 chip including three-axial gyroscope and accelerometers for three axes. This chip is integrated into micro-USB unit based on STM32 microcontroller. It is connected to PC using USB 3.0.

Positioning system is based on calibration of these accelerometers and gyroscope as well as on Madgwick filter [16] to filter noses and to adjust accelerometer readings by gyroscope readings.

Upon playback, positioning can be performed by two methods. The first one is that variations of VR headset positions are applied only to next visualized image. That is, after displaying image in glasses, its position is not varied any more. This allows to decrease load on graphical adapter and to increase total efficiency of the system, though, this method can be applied only at the playback speed higher than 60 fps, at lower speed tracking delays will be noticeable. The second method is to vary position of the sphere with superimposed texture (image) upon each variant of VR headset position. In this variant, positioning is smooth and without delays, however, load on graphical adapter increases. Both variants of positioning can be applied together. It is possible to apply the first variant at high playback speed, and the second variant – at low speed.

H. Video playback with encrypted key packets

The algorithm of protected playback is nonlinear and contains forks depending on video decoding method, playback speed, applied VR headset, and other factors. The algorithm should provide multithreading. Reading and decryption, packet decoding and visualization should be performed in separate threads so that to prepare subsequent video packets during decoding of a next frame, and to decode subsequent packets during visualization of next frame. In addition, at low playback speed positioning should be carried out in separate thread so that to avoid delays upon visualization of motions of VR glasses. Playback algorithm of protected 360 video is illustrated in Figure 6.

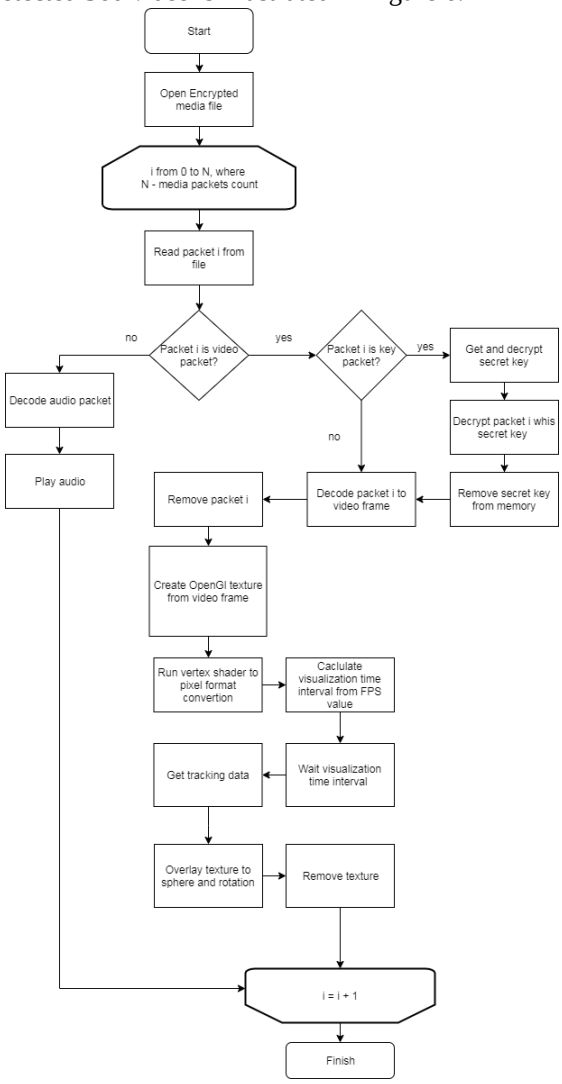

Figure 6: Algorithm of video playback with encrypted key packets.

2. RESULTS

In this work video distribution system has been implemented which allows to upload video to server which protects it and serves to users after encryption with secret keys.

On the client's side, the player has been developed operating according to the algorithm illustrated in Figure 6. Video decryption and decoding rates are summarized in Tables 1 and 2, respectively. Final playback speed using various GPU is summarized in Table 3.

Decryption rate of AES128 video packet has been analyzed using 3840×3840 video compressed by H265(HEVC) codec, and 3840×1920 video compressed by H264 codec. The tests were performed using Intel Core i5-9600K CPU 3.70GHz. The obtained decryption rates are summarized in Table 1.

Table 1: Decryption rate of key packets

Parameter	3840×3840 HEVC (FPS)	3840×1920 H264 (FPS)
Video packets	5700	3716
Key packets	11	149
Nonkey packets	5689	3567
Average key packet size (KB)	656.348	632.196
Max key packet size (KB)	2392.39	918.042
Average decrypt key packet time (ms)	**6.97784**	**6.72538**
Max decrypt key packet time (ms)	**26.1215**	**10.4042**

It follows from Table 1 that in the case of 3840×3840 video, the worst decryption rate of key packet is 26 ms. Hence, it can be concluded that reading and decryption of packets should be carried out in separate flow with decryption by at least 10 frames in advance.

Various hardware decoders were compared using 3840×3840 video compressed by H265(HEVC) codec and 3840×1920 video compressed by H264 and H265(HEVC) codecs. The comparisons of decoding rates are summarized in Table 2.

Table 2: Hardware decoding rate

Hardware acceleration method	Equipment	3840×3840 HEVC (FPS)	3840×1920 HEVC (FPS)	3840×1920 H264 (FPS)
CUVID	NVidia GeForce GTX 1080Ti	136	295	219
	NVidia GeForce GTX 1060	125	265	187
DXVA2	Radeon RX Vega 64	28	108	106
	NVidia GeForce GTX 1080Ti	70	184	173
	NVidia GeForce GTX 1060	34	138	136
D3D11	Radeon RX Vega 64	30	97	94
	NVidia GeForce GTX 1080Ti	72	189	168
	NVidia GeForce GTX 1060	34	126	124
QSV	QSV Intel Core I7 7700	66	192	180
	QSV Intel Core I5 7400	38	122	106

It follows from Table 2 that the highest video decoding rate is achieved using hardware acceleration on NVidia GeForce. Herewith, hardware decoding is not supported by older NVidia GPU, it is available starting from GeForce GTX 1050. The essence of hardware acceleration of decoding is not only in increased unpacking rate but also in decreased power consumption and deallocation of CPU resources for other tasks (for instance, decryption of key frames).

The data in Table 2 characterize only the rate of reading and decoding of video packets without their preliminary and subsequent decryption. Generally, upon subsequent conversion of pixel format of the obtained image into suitable for visualization form and scaling of the obtained picture with superimposition onto sphere, the processing rate will be significantly lower than the data in Table 2.

In order to develop Windows version of the player, dynamic detection of hardware acceleration was applied depending on the installed equipment. Table 3 summarizes the rates of complete cycle of video playback for NVIDIA GPU using CUVID hardware decoding. GeForce GTX 1080Ti and GTX 1060 GPU were selected to estimate the difference between the worst and the best video cards in this product line.

Table 3: Hardware decoding rate

GPU	3840×3840 HEVC (FPS)	3840×1920 HEVC (FPS)	3840×1920 H264 (FPS)
NVidia GeForce GTX 1080Ti	94	156	127
NVidia GeForce GTX 1060	79	128	104

It follows from Table 3 that both GPU allow to playback Ultra HD 4K 3D video at the speed higher than 60 FPS, and Ultra HD 4K panoramic video – at the speed higher than 120 FPS, provided that HEVC codec is used.

3. DISCUSSION

The obtained results demonstrate that encryption decreases rates insignificantly when only key packets are encrypted. This work does not describe methods and algorithms of watermark embedding since they should be discussed independently. Finally, video encryption does not guarantee its protection against copying by rightful users due to analog hole, however, it allows to significantly complicate unauthorized copying of video content. In addition, it is recommended to test the rates on Android using Android Media Codec for hardware video decoding.

5. CONCLUSION

By means of low-level implementation of proprietary player for 360 and 360 3D video, it was possible to embed encryption mechanisms of key packets and to obtain

sufficiently high rate for commercial application of protection system.

The developed mechanisms of video protection in combination with watermark embedding system are successfully used for VR simulators and 360 cinemas.

ACKNOWLEDGEMENT

This work was supported by the Ministry of Science and Higher Education of the Russian Federation (project no. 14.576.21.0102; Unique identifier RFMEFI57617X0102).

REFERENCES

1. **Digital rights management.** *Wikipedia, the free encyclopedia* https://en.wikipedia.org/wiki/Digital_rights_management Accessed on 03.09.2019.
2. **Real-Time Messaging Protocol.** *Wikipedia, the free encyclopedia*. https://en.wikipedia.org/wiki/Real-Time_Messaging_Protocol. Accessed on: 10.09.2019.
3. **Dynamic Adaptive Streaming over HTTP.** *Wikipedia, the free encyclopedia* https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP Accessed on: 10.09.2019.
4. S. Haber, B. Horne, J. Pato, T. Sander & R.E. Tarjan. **If Piracy Is the Problem, Is DRM the Answer?.** In: Becker E., Buhse W., Günnewig D., Rump N. (eds) *Digital Rights Management. Lecture Notes in Computer Science*, vol 2770. Springer, Berlin, Heidelberg, 2003. https://doi.org/10.1007/10941270_15
5. J.P. Rolland & H. Hua. **Head-Mounted Display Systems.** *Encyclopedia of Optical Engineering*, 2005. doi: 10.1081/E-EOE-120009801
6. **Advanced Video Coding.** *Wikipedia, the free encyclopedia.* https://en.wikipedia.org/wiki/Advanced_Video_Coding Accessed on: 25.09.2019.
7. **High Efficiency Video Coding.** *Wikipedia, the free encyclopedia* https://en.wikipedia.org/wiki/High_Efficiency_Video_Coding. Accessed on: 25.09.2019.
8. **Oculus/developers.** https://developer.oculus.com/ Accessed on: 01.10.2019.
9. J. Daor, J. Daemen & V. Rijmen. **AES proposal: rijndael**, 1999.
10. **OpenSSL Cryptography and SSL/TLS Toolkit** https://www.openssl.org/ Accessed on: 07.10.2019.
11. **All products. Guardant.** https://www.guardant.ru/products/all/ Accessed on: 09.10.2019.
12. **Libav Open source audio and video processing tools.** https://www.libav.org/ Accessed on: 09.10.2019.
13. **FFmpeg. A complete, cross-platform solution to record, convert and stream audio and video**. http://ffmpeg.org/ Accessed on: 09.10.2019.
14. **OpenGL. The Industry's Foundation for High Performance Graphics.** https://www.opengl.org/ Accessed on: 11.10.2019.
15. **Shader.** *Wikipedia, the free encyclopedia* https://en.wikipedia.org/wiki/Shader Accessed on: 11.10.2019.
16. O.H. Sebastian. **Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays.** *Report x-io and University of Bristol (UK)*, vol. 25, 113–118, 30.04.2010.