Advances in Multimedia Information Processing – PCM 2002, pp. 344-352,
Springer-Verlag, Dec. 2002.

MPEG IPMP Concepts and Implementation

Cheng-Ching Huang[1], Hsueh-Ming Hang[2] , and Hsiang-Cheh Huang[2]

Department of Electronics Engineering, National Chiao-Tung University,
Hsinchu, Taiwan.
[1]cchuang.ee89g@nctu.edu.tw
[2]{hmhang, huangh}@cc.nctu.edu.tw

Abstract. Intellectual Property (IP) protection is a critical element in a multimedia transmission system. Therefore, ISO/IEC MPEG started the IP protection standardization project on MPEG-4 a few years ago. A basic IPMP (Intellectual Property Protection and Management) structure and interface was first defined in its System part. In this paper, we will first outline the MPEG-4 basic IP protection mechanism and then describe our simulation of an MPEG-4 IPMP system. An IP protection application is constructed using the MPEG-4 system software – IM1 (Implementation Model one). This application includes a client-server program, in which a client can request the keys from a server in a secure way using a hierarchical key distribution structure.

1 Introduction

With the rapid development in computer industry and the swift growth of Internet, there is a widespread use of the digital multimedia contents in our daily life. The progress in data compression techniques also makes transmission of multimedia data stream possible. However, Internet is an open environment, therefore, if the user data and information are not protected, it might be illegally used and altered by hackers. To protect privacy and intellectual property (IP) right, people often use cryptographic techniques to encrypt data, and thus the contents protected by encryption are expected to be securely transmitted over the Internet.

One requirement of typical multimedia applications is the demand for real-time transmission. In contrast, conventional security methods are often designed to protect digital data files, which might not be suitable and efficient for real-time applications. To fulfill the demands for both real-time distribution and data security, including the IP protection mechanism into the multimedia standard might be a feasible and effective way to achieve an unambiguous communication environment.

MPEG (Moving Picture Expert Group) is the ISO committee to set up the international standards for multimedia data exchange. MPEG-2 has been applied to digital video broadcasting with some access control specifications [1][2]. IPMP (Intellectual Property Management and Protection), proposed for MPEG-4 standard, aims at protecting the compressed multimedia. In this paper, we will describe and implement a multimedia transmission system using the MPEG-4 IPMP concepts.

This paper is organized as follows. Sec. 2 is an overview of the MPEG-4 System and IPMP standards. Sec. 3 describes the IPMP plug-ins in the MEPG-4 System reference software "IM1." Sec. 4 describes the procedure of constructing the MPEG-4 IP plug-ins and an application example is included. Sec. 5 concludes this paper.

2 MPEG-4 Standard Overview and IPMP Framework

MPEG-4 is an international standard defined by the ISO/IEC committee. Compared to it predecessors, MPEG-4 pays more attention on the following three subjects: (i) real-time streaming, (ii) object-based coding, and (iii) enriched user interaction.

MPEG-4 standards contain 10 parts. The portion related to IP protection is in the first part, Systems. The IPMP framework in ISO/IEC 14496 consists of a normative "interface" that permits an ISO/IEC 14496 terminal to host one or more IPMP subsystems. An IPMP sub-system is a non-normative component of terminal, which provides several intellectual property management and protection functions. At the moment, MPEG committee is refining and extending the MPEG-4 IPMP specifications. A Message Router mechanism is to be added into the third Amendment of 14496-1.

In the MPEG-4 standards, the IPMP interface consists of IPMP elementary streams and IPMP descriptors. The IPMP elementary streams usually convey time-variant information such as keys associated with the encryption algorithm, which may change very rapidly. IPMP descriptors often convey time-invariant information associated with a given elementary stream or a set of elementary streams. IPMP elementary streams are treated as regular media elementary streams. And the IPMP descriptors are transmitted as part of an object descriptor stream.

Fig.1 shows how an IPMP sub-system works in an MPEG-4 terminal. Almost all the streams may be controlled or accessed by the IPMP sub-system but the Object Descriptor streams shall not be affected by the IPMP sub-systems.

Stream flow controller is a conceptual element that accompanies with every elementary stream. Stream flow controller can take place between the SyncLayer decoder and the decoder buffer. As Fig. 1 indicates, elements of IPMP control can take place at other points in the terminal. For example, they can appear after decoding (as in the case with watermark extractors).

3 IPMP in IM1

IM1 is an MPEG-4 Systems software developed by the MPEG committee. It may be used to verify and demonstrate the functionalities of MPEG-4 [4].

The Systems Core module in IM1 defines the infrastructure to implement MPEG-4 players. It provides the functionality of MediaObject, the base class for all specific node types. The API for Decoder, DMIF and IPMP plug-ins is also supported by IM1. Moreover, the code is written in C++, which is fairly platform-independent [5].

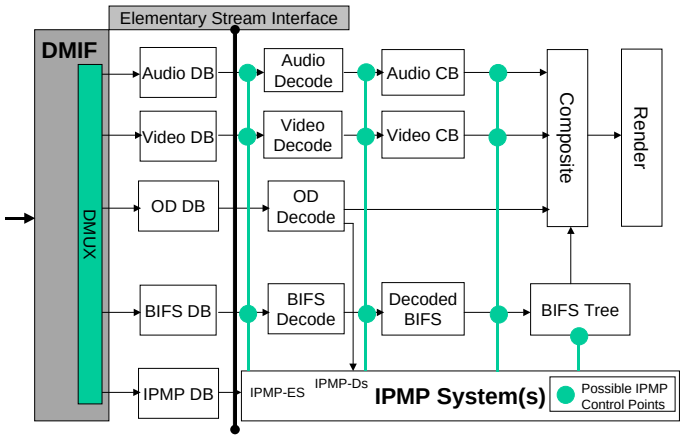


Fig. 1. IPMP sub-system in the ISO/IEC 14496 terminal architecture [3]

3.1 IPMPManager

In IM1, IPMP sub-systems are implemented by extending the IPMPManager class. IPMPManager is an interface between MPEG-4 player and the IPMP sub-system. Each media content access unit goes through the sub-system before it is stored in the decoding buffer. An implementation of IPMPManager can decrypt the encrypted content and thus block the unauthorized access to the media content.

IPMPManagerImp extends the IPMPManager interface, and it provides the major functionality of an IPMP sub-system. Simple implementations need to overload a few setup functions and the *Decrypt()* function, which decrypts one access unit using one IPMP stream. More complex implementations, for instance, when multiple IPMP streams are used to decrypt a single elementary stream, may overload the *Run()* function and implement different data flows by directly accessing the MediaStreams.

IPMP plug-ins interact with the core codes of the player through a special kind of buffer, known as MediaStreams. An IPMPManager object fetches an access unit, which is a kind of media, from one MediaStream object. After decrypting an access unit, it will dispatch one decrypted access unit into an output MediaStream object, which usually is a decoding buffer [6].

3.2 IPMPManagerImp

IPMPManagerImp extends the IPMPManager interface. It is the base class of all the IPMP sub-systems. IPMPManagerImp provides all the needed functions of a regular IPMP sub-system.

Each IPMP sub-system runs on its own thread. An IPMP sub-system is usually attached to three MediaStream objects – the encrypted input stream, the decrypted output stream, and the IPMP stream. According to the SDK [6], the workflow of a typical IPMP sub-system is shown in Fig.2. Our design procedure is modified from that in [6] and is outlined below.

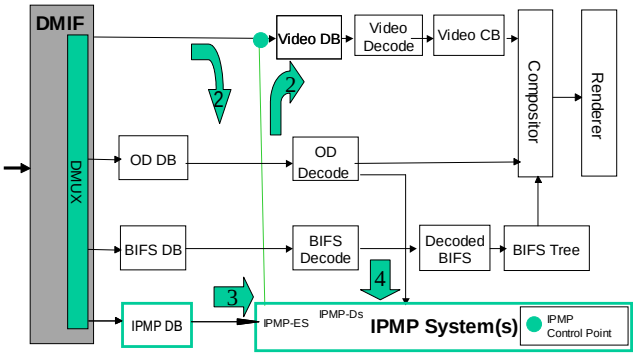


Fig. 2. A typical IPMP sub-system workflow

1. An object derived from IPMPManagerImp is instantiated by the IPMP sub-system module (usually a Dynamic Link Library, or DLL).
2. The application calls *IPMPManager::SetInputStream()* and *IPMPManager::Set OutputStream()* to attach input and output MediaStreams to the IPMP sub-system.
3. The application calls *IPMPManager::SetIPMPStream()* to attach an IPMP stream to the IPMP sub-system. This function may be called more than once if the elementary stream is protected by multiple IPMP streams.
4. The application calls *IPMPManager::SetDescriptor()* for each IPMP descriptor assigned to the elementary stream.
5. The application calls *IPMPManager::Init()* to initialize the IPMP sub-system and to confirm that the user has access to the protected elementary stream.
6. The application calls *IPMPManager::Start()*, which spawns the IPMP sub-system thread.
7. The IPMP sub-system thread fetches an access unit from the input stream and the corresponding access unit from the IPMP stream. Note that one IPMP access unit can control multiple content access units.
8. The IPMP sub-system calls a private virtual function, *Decrypt()*. This function is overloaded by specific IPMP sub-systems and performs the actual decryption.
9. The output of *Decrypt()* is stored in the output MediaStream.
10. Steps 7-9 are repeated until *IPMPManager::Stop()* is called by the application, or until reaching the end of the input stream.

Some of these steps have been implemented in IPMPManagerImp class, but in some special cases, we need to re-implement them again.

3.3 MediaStream

MediaStream class handles the buffering and synchronization of an elementary stream. It manages the memory buffer and fetch/disfetch access units from the buffer. The stored access unit maybe has time stamp on it. The current solution is to fetch the ac-

cess unit immediately and ignore the time stamp, fetch the matured unit only, or otherwise suspend.

4 Constructing an MPEG-4 IPMP Application Example

We will implement and demonstrate a multimedia transmission system with MPEG-4 IPMP by incorporating modern cryptographic techniques [7]. In designing the system, we adopt the Conditional Access (CA) concept by using a hierarchical key distribution structure as shown in Fig. 3.

In this system, we encrypt only the bitstreams in the TRIF files. The server generates and embeds the keys into the bitstream. When the keys are correctly retrieved, the decoded and decrypted video sequence can be played properly. Otherwise, the bitstreams cannot be decoded successfully.

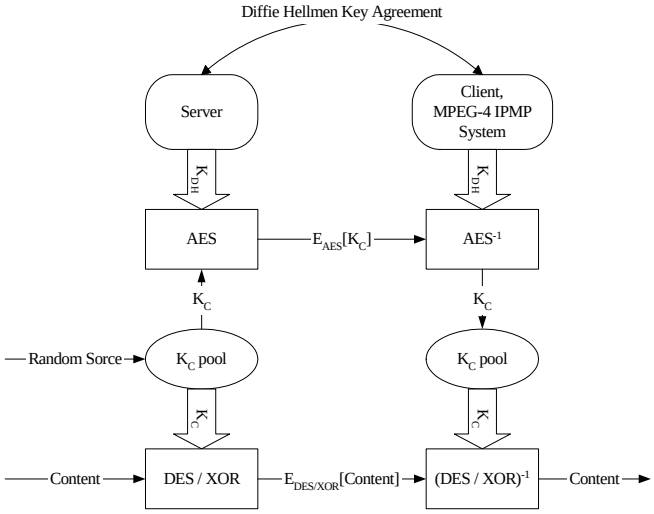


Fig. 3. The hierarchical key distribution structure

4.1 System structure and handshaking protocol

Our hierarchical key distribution system is illustrated by Fig. 3. At the upper level, we use the Diffie-Hellmen Key Agreement [8] that enables both the client-end and the server-end to securely retrieve the Session Key, K_{DH}, over the Internet. By applying the Advanced Encryption Standard (AES) [9], K_{DH} can serve as a secret key to encrypt K_C, and the encrypted K_C are then transmitted. The use of K_C is to serve as the key for the bottom layer encryptor. In our example, the contents to be encrypted are the compressed video, audio, or image bitstreams. Similar to the CA system in DVB, we achieve the security requirement by changing K_C frequently. The throughput of K_C is so high that we need a K_C pool to generate the keys constantly.

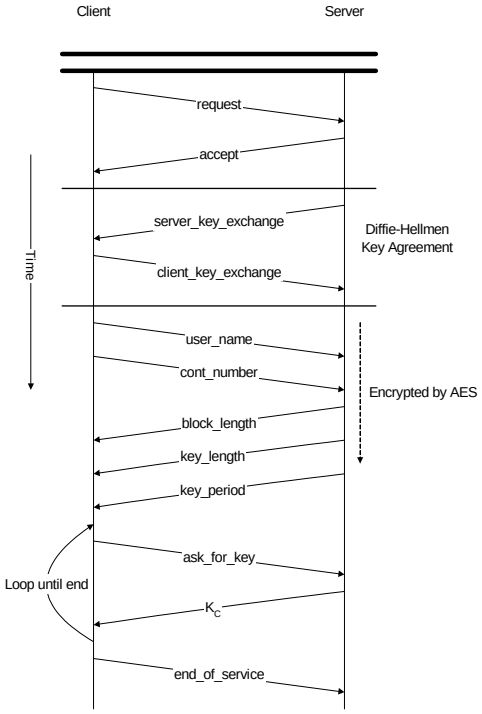


Fig. 4. The handshaking protocol

One of the most important elements of our system is the handshaking protocol. Fig. 4 shows the basic steps in establishing a connection between the client-end and the server-end. The procedure is stated as follows.

1. Client sends request = 0x31403 (4 bytes).
2. Server sends accept = 0x31403 (4 bytes).
3. Client and server proceed with the Diffie-Hellmen key agreement; all the forthcoming information will be encrypted with AES by this key.
4. Client sends user_name (44 bytes) and cont_number (4 bytes), representing the user name and content number, respectively.
5. Server sends block_length (2 bytes) and key_lengh (2 bytes) to initialize the encryptor, and key_period (1 byte) to tell the bottom layer encryptor the lifetime of K_C.
6. Client sends ask_for_key = 0x5327 (4 bytes) to ask for a new key from the server. Server sends a new K_C to client after receiving ask_for_key.
7. Client sends end_of_service = 0x0 (32 bytes) to terminate the handshaking.

4.2 The client-end IPMP plug-in

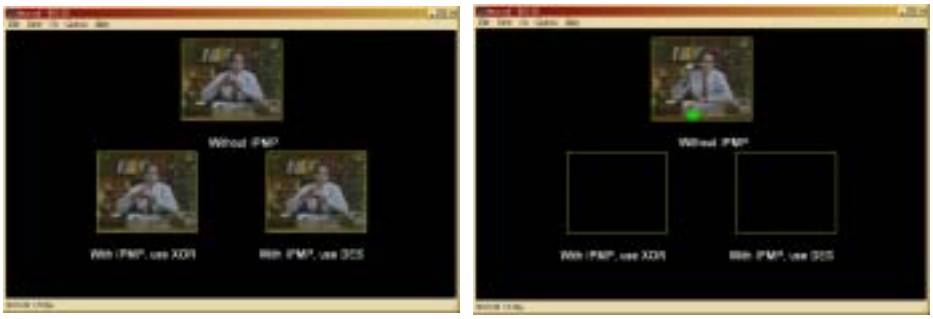
The player is the “IM1 2D player” executed under the Windows environment. Hence, the IPMP plug-in can be implemented with the DLL file in Windows.

There is one implementation of IPMP plug-in in IM1 core called IPMPNull. It works like a buffer to send the input MediaStream directly to the output side. Based on the existing IPMPNull program, we implement two new IPMP plug-ins in our system: IPMPXOR.dll and IPMPDES.dll. They are essentially two encryption methods. The first plug-in conducts the XOR operation between the received bitstreams and the key at the decryptor, a very simple encryption technique; the second one uses the DES [7] scheme for decryption.

In the MPEG-4 design, the IPMP stream is used to transmit keys. In our example, we transmit the key using TCP/IP, not DMIF, to avoid the incompatibility between our example system and the standard system.

The first step to implement IPMPDES is to create an IPMPDES class, and it inherits a class called "IPMPManagerImp." Then, we implement the *SetDescriptor()* function. The IPMPDescriptor within the TRIF file contains the information of the server location and the content identification number that is to be played. The *SetDescriptor()* function uses the above information to make a connection to the server and to initialize the decryptor locally. Next, we implement the *Decryptor()* function, which can decrypt the received MediaStream, and count the number of times K_C is used.

Figs. 5(a) and 5(b) are the demonstrations of two IM1 2D players. The video sequence is coded in the H.263 format. The bottom-left bitstreams in both figures are decrypted by IPMPXOR, and the ones on the bottom-right are decrypted by IPMPDES. The sequences on the upper side are not protected in both Figs. 5(a) and 5(b). In Fig. 5(a), we assume that the key can be reliably transmitted and received. Hence, the two encrypted bitstreams can be decrypted and displayed successfully. In Fig. 5(b), the keys are not retrieved. Thus, the encrypted bitstream cannot be decoded and displayed.



(a) (b)

Fig. 5. Demonstration of the proposed system: the unprotected bitstreams (*upper*) and the protected bitstreams (*lower*). (a) Correctly retrieved keys, and (b) keys not retrieved

4.3 The server-end

The server can be divided into two parts, one is the encryptor and the other part is responsible for sending keys. Fig.6 is a screenshot of the server-end application. We write it in C++ and it is a DOS command-line program. But the GUI is done in Java using the pipes stdin and stdout.

Fig. 6. The server that can turn on/off keys

5 Conclusions

In this paper, we first briefly describe the MPEG-4 IPMP system concepts. We then analyze the IPMP API in the reference software of the MPEG-4 Systems – IM1. After studying the IM1 Core and its IPMP API, we implement a functional IPMP sub-system by modifying IPMPNull – a prototype of the IPMP sub-system.

We use the hierarchical key architecture to construct an application example, following the MPEG-4 IPMP concepts. Our example simulates the functionalities suggested by the standard. We demonstrate that the MPEG-4 IPMP is a practical way for protecting the multimedia content.

6 Acknowledgement

This work was supported by National Science Council (Taiwan, ROC) under Grant NSC 90-2213-E-009-137.

References

1. ISO/IEC 13818-1 *Generic Coding of Moving Pictures and Associated Audio Information*: *Part 1 Systems* (ISO/IEC JTC1/SC29/WG11 N0801rev), April 1995.
2. H. Benoit, *Digital Television, MPEG-1, MPEG-2 and Principles of The DVB system*, Arnold, 1997.
3. ISO/IEC 14496-1:2000(E) *Coding of Audio-visual Objects*: *Part 1 Systems* (ISO/IEC JTC1/SC29/WG11 N3850), October 2000.
4. ISO/IEC JTC1/SC29/WG11 N4291, *MPEG Systems (1-2-4-7) FAQ*, Jul. 2001.
5. ISO/IEC JTC1/SC29/WG11 N4709, *MPEG-4 Systems Software Status and Implementation Workplan*, March 2002.
6. ISO/IEC JTC1/SC29/WG11 M3860, *IPMP Development Kit*, Aug. 1998.
7. B. Schneier, *Applied Cryptography, 2nd edition*, John Wiley & Sons, 1996.
8. *PKCS #3: Diffie-Hellman Key-Agreement Standard*, An RSA Laboratories Technical Note, ftp://ftp.rsa.com/pub/pkcs/ascii/pkcs-3.asc.
9. J. Daemen and V. Rijmen, *AES Proposal: Rijndael* (corrected version), http://www.esat.kuleuven.ac.be/~rijmen/rijndael/rijndaeldocV2.zip.