Project 1 - DVD Manufacturer

Introduction to Computer Security

Floriane Magera (S111295)
Fabrice Servais (S111093)

March 27, 2015

1 Introduction

2 `PlayerKeys`

This section focuses on the keyfile generation. First we will explain how we derived the keys, and then which system we used to encrypt them.

2.1 Key Generation

We derive the player's keys thanks to two arguments : the AACS password and the node id relative to the key. First we need to generate an AACS key from the AACS password. We used the function `CreateAESKeyMaterial`, provided in `KeyTree`. Then we created our own function `generateKey` that creates the player key. We apply MD5 to the result of xoring the node id and the AACS key. We use MD5 because it is convenient to create a 16 bytes key as we need. We know it is not safe but as the key itself is supposed to be secret, it does not matter.

2.2 File Generation

After generating the keys, we encrypt each key and the corresponding node id with AES-128 in CTR mode. Then we generate a MAC of the encrypted keys with SHA256. We used in both cases `SecretKeySpec` in order to derive the keys from the user password but in the encryption case, we used the `CreateAESKeyMaterial` to have a sufficient key length.

As the the MAC length is 32 bytes, about 2^{128} computations are needed in order to have a probability of 50% to have a collision. We think it is good enough. Concerning the encryption, we decided to encrypt the initialization vector too, as an extra security layer. We had to do it with ECB, which is not the best mode of AES, but as the size of the IV is short, there is not too much redundancy. For the key, we first hashed the password, it allows for convenient size. Then the usage of AES in CTR mode seems to us to be a very good choice for safety.

We were forced to add the initialization vector to the foreseen content of the file, because otherwise we would have used AES in ECB mode, which would not be safe enough, as ECB allows for redundancy in the crypted text.

3 `DVDManufacturer`

The first step of the `main` function of the class is to parse the revocation list in order to get the unauthorized players. Given that list, the AACS password the title of the file and the path to the file to encrypt, the method `encryptContent` that takes care of reading the file to encrypt and writing the encrypted content. It also asks `KeyTree` to get the cover id's depending on the revocation list.

The encryption part is done by `encrypt` and has several steps :

1. Generation of K_t : Usage of `KeyGenerator`. We used HmacSHA256 to produce a 32-bytes-long key, to make it harder to compute it back.
2. Generation of K_{enc} and K_{mac} :
   - $K_{enc} = HMAC(K_t, \text{"enc"})$ using `Mac` with MD5.

- $K_{mac} = HMAC(K_t, \text{"mac"})$ using `Mac` with SHA1.

MD5 and SHA1 are fast algorithms that do not provide a maximal quality in terms of collisions,... However, there is no special need for the opposite, it relies on the fact that K_t is secret and that there is no need to get back to K_t from K_{enc} or K_{mac}.

3. Encryption of the content : Using AES in CTR mode. The initialization vector is also recovered to be stored later, in order to be allowed to decrypt it afterwards.
4. Computation of the cover : From the cover id's, it computes the cover of the keys thanks to `PlayerKeys`.
5. Encryption of K_t : K_t is encrypted with the cover using AES and the pairs <node, encrypted K_t> are stored in a `HashMap`.
6. Generate the file : The data are aggregated in an array of byte.
7. Generate MAC : The MAC is generated from the data (above) and K_{mac} using HmacSHA512 and is added to this data. Finally, it is encrypted and returned. The format of the file before the encryption is given in Table 1.

Item	**Number of bytes in the file (bytes)**
Size of the title	4
Title	'Size of the title'
Number of pairs <node ; key>	4
Size of the node	1
Size of the key	1
Node || Key	('Size of the node' + 'Size of the key') × 'Number of pairs'
Size of the IV	1
IV	'Size of the IV'
Size of the content	4
Content	'Size of the content'
MAC	64

Table 1: Format of file

4 `DVDPlayer`

4.1 Retrieving the keys

This part occurs in two steps : decryption and analyze of the keys, these actions are performed by `decryptKeys` and `generateKeys` respectively. Given the keyfile, we must decode it, check its

integrity and retrieve the keys in a convenient format. We know how the file is formatted and thus we can identify the bytes of the file corresponding to the initialization vector, the keys and the mac. The first thing to chek is the integrity of the document. If the MAC matches, then we decode the content thanks to the password and the initialization vector and return it.

As we have an upper bound on the complexity of the decryption of K_t, we decided that we would put the node id and the corresponding key of each node in a HashMap. Again as we know the format of the encoded content, we can retrieve the node id and the key, only on indexes basis.

4.2 Decryption

The first step of the decryption is to read the file, getting back the field following the TABLE 1 and saving the pairs <node ; encrypted K_t> in a `HashMap`.

After that, the method iterates through the keys belonging to the player, and the nodes associated to it. For each node, if it is contained in the encrypted file, then the methods decrypt the associated encryption of K_t with the current key. After deriving K_{mac}, it computes the MAC of the file and checks if it is the same as in the file. If no, it triggers an `ContentMACException`. If yes, it continues by decrypting K_t, deriving K_{enc} and decrypting the content. The result is then written in a file.

If the method has checked all the keys of the player without finding a matching in the file, it raises a `PlayerRevokedException`.