Flexible Traitor Tracing for Anonymous Attacks

Hongxia Jin[1] and Jeffery Lotspiech[2]

IBM Almaden Research Center
San Jose, CA, 95120
{jin,lotspiech}@us.ibm.com

Abstract. Copyrighted materials are divided into multiple segments and each segment has multiple variations that are differently watermarked and encrypted. The combination of watermark and encryption enables one to identify the actual users (traitors) who have engaged in the piracy. In this paper, we shall present a traitor tracing scheme that can efficiently and flexibly defend against anonymous attacks.

1 Introduction

This paper is concerned with content protection in a one-to-many type of distribution system, for example, a pay-per-view TV system or for prerecorded/recordable media. The security threat is a Napster-style "anonymous" attack which comes in two forms. An attacker redigitizes the analogue output from a device and redistributes the unprotected content (content attack). Or the attacker extracts the decryption keys from the device and sells them on demand (title key attack).

For both types of anonymous attack the only way to trace the users (traitors) who engaged in the piracy is to use different versions for different users. The content is divided into multiple segments and each segment has multiple variations. Each variation is differently watermarked and encrypted with a different key. The keys are assigned such that any given device has access to only one variation for each segment. After recovering enough unauthorized copies of different content, the scheme can determine the traitors.

Our scheme systematically allocates the variations based on an error-correcting code. Furthermore, we concatenate codes [1]. In our construction, variations in each segment are assigned following an inner code, which are then encoded using an outer code. We call the nested code the *super code*. This super code avoids the bandwidth problem by having a small number of variations at any single point. For example, suppose both inner and outer codes are Reed-Solomon codes. For the inner code, we can choose 16 variations at each of the 15 segments and hamming distance being 14. This effectively creates 256 versions per movie. For the outer code, we can use 256 versions per movie throughout the 255 movie sequence and have a Hamming distance 252. This example can accommodate more than 4 billion users. and the extra bandwidth needed is about 10% of the bandwidth needed for a normal 2 hour movie. These parameters fit very well in a practical setting.

2 A flexible scheme

Each device is assigned 255 keys, namely "sequence keys", from a key matrix with 255 columns and 256 rows, exactly one key from each column. Each key correponds to one movie in the sequence. It can be used to decrypt one of the 256 tables, each containing the encrypted 15 variant keys for that movie. The scheme can be flexible during deployment because only the outer code needs to be fixed to assign the sequence keys when the devices are manufactured. The inner code and those variant encrypting keys, even the necessity of an inner code are movie-by-movie decisions that can be delayed to the movie distribution time.

We improve our scheme by extending the number of rows in the key matrix for the sequence key assignment. For example, each sequence key comes with 1024 versions in the world even though there are only 256 versions per movie created from the inner code. Then there would be 1024 variant encrypting key tables. To keep this number at 256, we can use an array to index every 4 sequence keys into the same key k_i which is randomly chosen to encrypt the table i. Each k_i is encrypted with 4 sequence keys. The only overhead is the array of 1024 entries, which is negligible compared to the storage of a high-definition movie.

With this extension, we improve the flexibility of the inner code. It now can accommodate an inner code that creates more than 256 versions. The extension also achieves better traceability for key tracing. Intuitively with every recovered movie with q variations, we can trace down to $1/q$ of the population, assuming there is only a single user involved in the attack. A larger q means faster tracing.

Another important benefit of the extension is that the length of the movie sequence is in effect increased, without increasing the number of keys in each device. After 255 movies in our example, the scheme starts reusing columns. The attackers, noticing this, should use the same variation in a new movie that they previously used in an older movie at the same point in the sequence (column). This tactic would give tracing agency no new information from the new movie. But with our extension, the grouping of four sequence keys to each variation can be reshuffled in a new sequence of movies. The tactic at least reveals which of the possible four sequence keys the attackers have had in the first sequence.

Yet another important benefit of the extension is the improvement of the scheme's overall resistance to attacks. Every time the attacker redistributes keys, fewer keys remained in the system are useful for the future. When the number of exposed keys is big enough, the system is broken. Suppose the movies are randomly being attacked, a simple combinatorial analysis tells us that it takes $q \log q$ movies to expose all the q versions of keys in each column in the key matrix with high probability. Apparently, the extended scheme with a larger q can survive longer. As future work, we are continually interested in overcoming the practical barriers to bring the work to real practice.

References

1. H. Jin, J.Lotspiech and S.Nusser, "Traitor tracing for prerecorded and recordable media", ACM DRM workshop, Oct. 2004.