RJ10390 (A0610-018) October 4, 2006
Computer Science

IBM Research Report

Efficient Traitor Tracing

Hongxia Jin, Jeffery Lotspiech, Nimrod Megiddo
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Efficient Traitor Tracing

Hongxia Jin, Jeffery Lotspiech and Nimrod Megiddo

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

Abstract. In this paper we study the traitor tracing problem, which originates in combating piracy of copyrighted materials. When a pirated copy of some copyrighted material is observed, a traitor tracing scheme should identify at least one of the real users (traitors) who participate in the construction of the pirated copy. Content is divided into multiple segments and each segment comes with multiple variations. Each user can only play back one variation through the content. Given some practical restrictions on the number of variations one can put into the content and the number of users to accommodate, we will show an efficient tracing scheme that can achieve superlinear traceability in terms of the number of recovered pirate copies of the content needed as a function of the number of traitors involved in a coalition. The efficiency of the tracing scheme is one of the enabling factors for the first commercial use of a traitor tracing technology within the AACS [1] (Advanced Access Content System) content protection standards for next generation high-definition video optical disc.

Keywords: *Content protection, media security, broadcast encryption, anti-piracy*

1 Introduction

This paper is concerned with the protection of copyrighted materials. The technology is termed “traitor tracing” in the literature. The term refers to a way of identifying the source of unauthorized copies of digital content. A number of business models has emerged whose success hinges on the ability to securely distribute digital content only to paying customers. Examples of these business models include pay-TV systems (Cable companies) or movie rental companies like Netflix, and massively distributing prerecorded and recordable media. There is a anonymous pirate attack in these business models. For example, an attacker re-digitizes the analogue output from a compliant device and redistribute the content in the clear. In this case, the only forensic evidence is the unprotected copy of the content (content attack). Or, the attackers may compromise the tamper-resistance of one or more players to extract the content decryption keys.

[1] AACS is in business available for licensing now, see press release at http://www.aacsla.com/press

They can then set up a server that sells decryption keys on demand (key attack). To defend against these types of attacks, one might need different versions of the content/keys for different users.

For the rest of this paper, we will assume that the content protected is copyrighted digital movies. We do this for concreteness, and because we were inspired by the AACS application. This movie application is true of many different one-to-many content distribution channels, in that it is generally infeasible to prepare and send individualized (e.g., watermarked) movies to each user. On the other hand, it is also infeasible, usually for security reasons, to customize each copy at the receiving end[2]. A feasible technical approach is to choose certain points in the movie and create different variations for each of those points. Each variation is differently encrypted. The movie is thus augmented by all the variations. Each user receives the same bulk-encrypted movie. However, each user can only decrypt one of the variations at each point. In other words, each recipient would follow a different path through the variations during playback time. It effectively creates multiple versions of a movie. Over time, when recovering enough pirate movies, it may be possible to detect the devices in a copyright attack by examining the variations recovered in the unauthorized copies of the movies. The devices/users used in the pirate attack are called "traitors" or "colluders".

There have been a lot of studies in the area of traitor tracing. A traitor tracing scheme usually consists of two basic steps:

1. Assign variations or keys for the content to devices.
2. Based on the recovered version keys/content, trace back to the traitors.

A tracing scheme is static if it pre-determines the first step before the content is broadcast and does not change afterwards. A traitor can be identified when enough copies of keys/content are recovered. On contrast, if the first step can be updated based on observed pirate keys/content, the scheme is dynamic. Fiat and Tassa introduced a dynamic traitor tracing scheme [5]. It involves realtime computation to decide the new assignment for the first step.

A tracing scheme is deterministic if it detects at least one exact traitor without any chance of incriminating an innocent user, otherwise it is probabilistic. The first traitor tracing scheme was proposed by Chor, Fiat, Naor and Pinkas in [3]. The traitor tracing schemes in [3],[4] are static and probabilistic. There is also a sequential traitor tracing[6] [7] which is static and deterministic. More formal analysis of its traceability is shown in [8], [9]. For the static assignment, it can be random or systematic. The traitor tracing schemes in [3],[4] randomly assign the decryption keys to users before the content is broadcast. The main goal of their scheme is to make the probability of exposing an innocent user negligible under as many real traitors in the coalition as possible. The traitor tracing schemes in [6] [7] used systematic assignment.

The authors have been involved in what we believe is the first large-scale deployment of a tracing traitors approach. It is used to defend against the anony-

[2] Unauthorized copies generally imply a break the correct operation of the client. How can a broken client be expected to correctly generate the customizing marks?

mous attack for the (Advanced Access Content System) content protection standards for next generation of high-definition optical DVDs. In the AACS context, each movie is divided into multiple segments and each segment is augmented with multiple variations. As one can imagine, the variations takes extra space on the disc. A practical traitor tracing scheme on a prerecorded optical movie disc should take no more than 10% of the space on the disc to store the variations. This puts practical restriction on the number of variations one can put into a movie. The market for such discs is huge, involving literally a billion playing devices or more. This means a tracing scheme needs to be able to accommodate large number of devices. After meeting these requirements, it is also important to detect the coalition of traitors using as few recovered movies as possible.

In summary, a traitor tracing scheme for AACS needs to meet all the following requirements:

1. the number of variations for each movie cannot be big
2. the number of devices/users *must* be big
3. after the above two requirements are met, the number of movies necessary to detect a coalition of should be as small as possible

Much of the literature on traceability codes has taken the approach of fixing the number of colluders and the number of recovered movies and trying to find codes to support an optimal number of devices/users for a given number of variations of each movie. In the AACS context, a traitor tracing scheme must first meet the above two requirements. Existing schemes do not fit here. For example, the code shown in [7] either has too few codewords (accommodates a small number of devices) or the number of variations is too large (requires too much space on the disc). Bringing the long-standing theoretical work to practice was the major effort we undertook in the AACS system.

We have designed a way to meet the first two requirements as shown in [10]. In summary, for each movie, there is an "inner code" used to assign the different variations at the chosen points of the movie; it effectively creates different movie versions. For example, 16 variations are created at each of the 15 points in the movie, effectively generating 256 versions for each movie. For a sequence of movies, there is an "outer code" used to assign movie versions to different players. For example, each player is assigned one of the 256 versions for each movie in a sequence of 255 movies. By concatenating the two levels of codes, we managed to avoid having a big number of variations at any chosen point but can still accommodate the billions of devices we anticipate.

After meeting the first two requirements, we have studied the relationship between the number of traitors and the number of movies required. In this paper, the efficiency of the tracing (traceability) is measured by the number of recovered movies needed in order to detect traitors involved in a coalition, given the practical constraints of the first two requirements. In fact, the first two requirements have more to do with the first step in a traitor tracing scheme, and the third requirement has more to do with the second step in the scheme.

For the second step, there is one thing common with all the existing schemes. When the traitors in a coalition collude together in the pirate attack, these schemes are defined to detect and incriminate the one traitor who has the highest score. They suppose this traitor can be disconnected and tracing continues after that. The main contribution of this paper is to develop the first tracing scheme that tried to detect all the traitors in the coalition together. With the recovered movies, we try to detect which coalition of players may have involved in the attack, instead of which one particular player may have been involved. In other words, we could incriminate more than one player each iteration of the algorithm. It turns out that it is much less likely that coalitions appear by random chance, than that individual players randomly have high scores. This truism is the essence of the efficient tracing underneath our new tracing scheme. We believe providing an efficient tracing algorithm is one of the most important steps to make this AACS commercial deployment possible.

In rest of the paper, we will present our tracing scheme in Section 3. We will analyze its false positive rate in Section 4 and its performance/efficiency in Section 5. We show simulation results in Section 6 and conclude in Section 7.

2 Pirate model

There are two well-known models for how a pirated copy (be it the content or the key) can be generated:

1. Given two variants v_1 and v_2 of a segment, the pirate can only use v_1 or v_2, not any other valid variant v_i.
2. Given two variants v_1 and v_2 of a movie segment ($v_1 \neq v_2$), the pirate can generate any valid variant v_i out of v_1 and v_2

The second model, of course, assumes the attackers have more power. As shown in [11], when using this model, the lower bound of the number of movies it takes to detect traitors in a coalition of T is T^2. We believe the second model fits documents or software better than audio or video. Since in this paper we are concerned with audio and video, assuming the first model is acceptable in the AACS context. Indeed, this so-called marking assumption is often made by other traitor tracing schemes shown in the literature. Also, in a key attack, the first model says it is impossible to calculate a valid random cryptographic key from combining two other valid random keys–which is obviously true.

3 Tracing Scheme

The AACS traitor tracing scheme also consists of two basic steps:

1. Assign movie version keys to devices.
2. Based on the recovered movies/keys, identify the devices used in the attack.

For the first step of the AACS tracing scheme, instead of directly assigning variation keys for each movie to the devices, a level of indirection allows us to assign movie version keys to the devices, from which the devices can obtain the actual variation decrypting keys. This level of indirection allows us to save space on the devices used to store the keys. In other words, the "outer code" was used to assign keys to devices in the first step. For example, each device is assigned a set of 255 keys, corresponding to the 255 movies in the sequence. Each key comes with 256 versions in the world. These keys are called "sequence keys" in the AACS specification. Many players will receive any given sequence key, but no two players will receive exactly the same set of 256 keys. These sequence keys are placed in the players at manufacturing time. For most types of players, it is impossible to dynamically update those keys. So our scheme has to be static.

The sequence key assignment can be random or systematic. In rest of the paper, we will just assume that the licensing agency assigns sequence keys to devices uniformly at random and will focus on the second step for the actual tracing algorithm.

The traditional approach taken in the second step is simple and straightforward: you take your sequence of recovered movies, and simply score all the devices based on how many movies match with what each device has been assigned. You incriminate the highest scoring device.

While the classic method for detecting a traitor is to score each individual player, a different method suggests itself: should not the problem be finding *every* member of the coalition? Although this second method seems more useful, the classic method has some obvious advantages:

1. It seems easier.
2. The number of coalitions is exponential compared to the number of individuals. For example, if there are a 1,000,000,000 devices in the world, there are roughly 500,000,000,000,000,000 pairs of devices.
3. It seems essential against the "scapegoat" strategy. In this strategy, the coalition sacrifices a few devices and uses them heavily while using the others lightly, to keep some in reserve. Note that even without the scapegoat strategy, simulation results usually show some unlucky innocent devices intermixed with guilty players when the devices are scored in the classic way.

It may seem counter-intuitive, but we believe it is easier to find the entire coalition than to find the individual traitor. It turns out that it is much less likely that coalitions appear by random chance, than that individual players randomly have high score. An example can informally illustrate the underlying idea. Suppose there are 4 people involved in a colluding attack, and we have a random sequence of 20 recovered movies. Each movie originally has 256 variations of which a given player only plays 1. The attackers wish to see that high scoring device can happen by chance. If the four attackers are using round robin, each guilty player will evenly score 5. Can we incriminate any player that share 5 movies with the recovered sequence? No, there will be about 15 completely innocent players scoring 5 or greater due to chance alone. What can you do then? You have to recover more movies before you can incriminate any player. In general, with N players and q variations for each movie, the expected number of individuals who can score x among m movies are:

$$N * (1/q)^x * \binom{m}{x} \tag{1}$$

However, the above 4 guilty players together can explain all the movies in the sequence. What is the chance that a coalition of size 4 might have all the variations in the sequence? The answer is roughly 0.04. In other words, while there are plenty of players that can explain 5 movies, it is unlikely that any four of them can "cover" all twenty movies. If we find four players that do cover the sequence, it is unlikely that this could have happened by chance. It is more likely that that some devices in the coalition are indeed guilty.

On the other hand, the attackers may use scapegoat strategy. Some player is used heavily, for example, score 9 or 10. The traditional approach can correctly identify him, but it is hard to find the lightly used player and the true coalition size. Our new tracing algorithm can nonetheless find the other members in the coalitions and find out the coalition size.

In section 3.1, we will show how we find a coalition to explain the recovered movies. After we find the suspect coalition, in section 3.2 we will how we identify the actual guilty players in the suspect coalition and filter out the innocent ones.

3.1 Finding a coalition

Let us formalize the above intuition a bit more. If there are N players, and a sequence of m movies are selected, each movie having one random variation out of q, the expected number of coalitions of size T are:

$$\binom{N}{T} * (1 - (1 - 1/q)^T)^m \tag{2}$$

If the expected number of coalitions is less than 1, this formula also gives an upper bound on the probability that a random sequence of m movie variations is covered by a coalition of size T.

If T is noticeably less than q, a simplification of this is a close upper bound:

$$\binom{N}{T} * (T/q)^m \tag{3}$$

The problem of finding a coalition of players that covers a sequence of movies is equivalent to a well-known problem in computer science called Set Cover. It is NP hard. But in reality the calculation time is still reasonable, for the parameters that AACS is concerned with. Below we show a sample Set Cover algorithm that can be used in the tracing.

Assume the licensing agency has observed a sequence of movies and determined the particular variation (the "symbol") in use for each. We also introduce the parameter k, the number of symbols that would probabilistically identify a single player. For example, k could be set to $log_q N$, where N is the total number of players.

The following recursive procedure *COVER*, if given a suspected number of traitors T and a list of the m encoded symbols discovered, returns true if and only if there is at least one coalition of size T that can explain the observed symbols:

1. If $T * k$ is greater than the number of symbols, print “many” and return true.
2. Calculate the minimum number of symbols that the largest-scoring traitor must have:
$$min = \lceil \frac{m}{T} \rceil$$
3. For each possible combination of k symbols, calculate whether the single player assigned to that combination covers 'min' number of symbols. If it does, perform the following:
   (a) If $T = 1$, print the player ID and return true.
   (b) If $T > 1$, recursively call *COVER* passing the symbol list after removing all the symbols from the suspect player and with $T = T - 1$.
      i. If the recursive call returns false, continue to loop through the other combinations.
      ii. If the recursive call returns true, print the player ID and return true.
   (c) If all combinations have been checked, return false.

The tracing algorithm assumes that the size of the coalition is unknown, and proceeds to calculate both the size of the coalition as well as the actual players involved. Below is the method that uses the above procedure *COVER* (or any other Set Cover procedure):

1. Set T = 1.
2. Run *COVER*.
3. If *COVER* returns true, exit.
4. Otherwise set $T = T + 1$ and loop to step 2.

Eventually the procedure must exit at step 3. Why? Once the number of movies is less than $T * k$, *COVER* is guaranteed to return true (see step 1 in *COVER*). But the interesting thing happens if you exit “early”. In this case, you have found a coalition, and you can calculate the probability that a larger completely different coalition could have incriminated this coalition of size T, as explained in Lemma 1.

3.2 Identify guilty individuals in the found suspect coalition

Once we have found a coalition, who in the coalition should we incriminate? What is the chance that some of the players in the purported coalition of size T might be actually innocent, being victimized by a scapegoat strategy that is hiding a few lightly used guilty players? We calculate this as follows:

For each combination of T players, perform the following steps:

1. Temporarily assume that the players in the particular combination are guilty.
2. If the number of players in this combination is c, subtract c from T
3. Temporarily subtract from the list of movies all the movies that can be explained by this combination of players.
4. Use the formula 2 above using the new number of movies m and T, to evaluate the probability that the remaining players are completely innocent. If the formula yields a number greater than 1, assume the probability is 1.

When this procedure has ended, there will be a list of all possible combinations of players together with the chance that the remaining players are innocent. If some of these combinations indicate that there is a good chance that a player is innocent under those circumstances, the licensing agency would be well advised not to take action against the player (yet). On the other hand, some players will seem guilty under all combinations. In other words, the license agency can use the minimum guilty probability of the each player under all combinations as the probability of guilt of the player. In general, players that score higher in terms of the number of movies they could have encoded are also more likely to show up as guilty after the procedure. It is also reassuring that after this procedure any player that is identified only as "many" in the *COVER* procedure will show up as likely innocent.

Note it is possible that two of the players in the coalition may have a high overlap in movies. In this case, the procedure above might reveal that if player A is guilty, there is a good chance that player B is innocent, and vice versa. In this case, the licensing agency would be well advised to avoid making a decision about either of them until more movies have pointed to one or the other. Note that using the "*min*" probability rule, both players show up as likely innocent for the time being. However, the policy used by the licensing agency is outside of the scope of this paper. This algorithm provides the necessary tool to the licensing agency: a short list of potentially guilty players and probability of their actual innocence or guilt.

We now discuss a few optimizations. Before calling *COVER* the first time, it is usually faster to pre-calculate the $\binom{m}{k}$ potential players. Then, in step 3 of cover, you simply iterate through the pre-calculated list, seeing if each player is still a candidate under the current circumstances. Determining which player corresponds to particular list of k symbols can often be optimized. It is always possible to exhaustively search through all the players to see which one is indicated, but this can be obviously sped up by well-known techniques like table look-up and hashing. Furthermore, if the encoding method used is a linear code, as it was shown in our previous paper [10], it is possible to identify the player by algebraic means. For example, each list of k symbols defines k equations in k unknowns, which can be solved by Gaussian elimination.

4 False positive

Our tracing algorithm assumes that the size of the coalition is unknown, and proceeds to calculate both the size of the coalition as well as the actual players involved. If the size of the coalition is known from other sources, the answers may be exact; otherwise, the answer is always probabilistic. The problem is, from the attackers side, they do not know what sequence would incriminate an innocent player, so they are just guessing. We can make the probability they guess correctly arbitrarily small by just collecting more movies. The following lemma shows the false positive rate in our detection.

Lemma 1. *Assume that a coalition of guilty players cannot deduce the movie assignment of any other player in the world, for a coalition C, $|C| = T$, found by algorithm* COVER*, the probability that every member in coalition C is innocent is bounded by formula 2. In other words, the formula gives the false positive probability in the detection.*

Proof: Imagine that the process of assignment is the opposite of the way it works in real life: instead of starting with the assignment of variations to the population, the coalition randomly picks their assignment and then picks the particular variations of m movies in any way they choose. Only then does the licensing agency, not knowing what the coalition has picked, assign the variations for the remaining innocent players randomly. The chance that this assignment would result in a coalition of size T amongst the innocent players is clearly bounded by equation 2. And since there is no way to distinguish the "real life" case from the "thought experiment" case based on the player assignment (they are both equally random), the equation does represent the best that the attackers can do. □

The licensing agency can choose any acceptable value for the false positive rate. The smaller the false positive rate, the more pirate movies it needs to recover. We can get any kind of confidence level desired, but it will just take us more recovered movies to achieve. If the attack is ongoing, we always have the option of increasing our confidence by recovering more movies. In general, for each movie recovered, our confidence that the guilty players are, in fact, guilty is increased by roughly q/T. Since our entire tracing is probabilistic, we can factor in some false positives from the underlying watermarking technology (that is determining which variations were recovered) as well.

5 Tracing efficiency

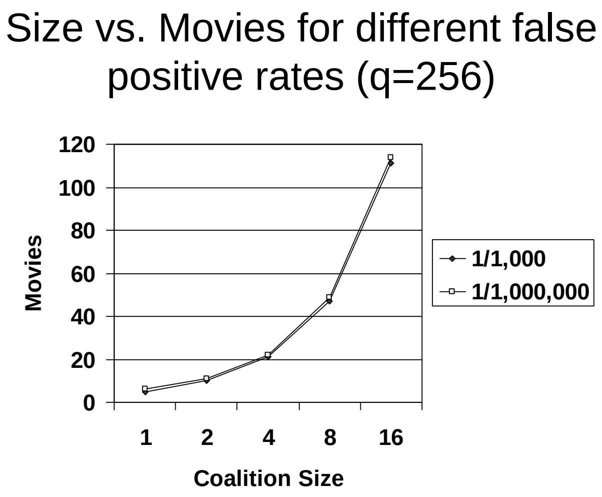
From formula 3, we can calculate the number of movies m it takes for a coalition of size T to achieve any level of confidence (or false positive rate), for example, λ. We obtained a superlinear relationship between m and T.

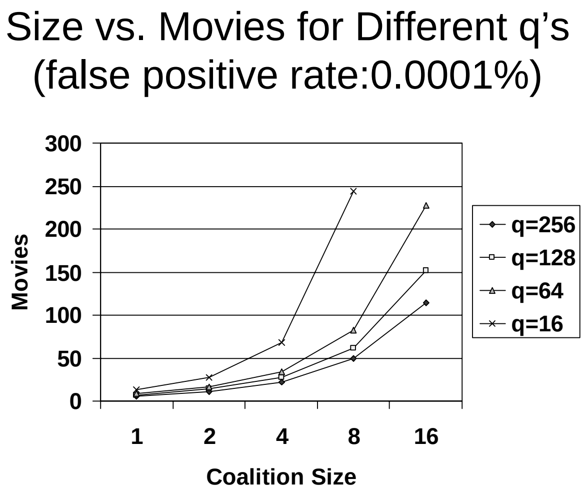
$$\binom{N}{T} * (T/q)^m = \lambda \tag{4}$$

Because N is much larger than T, $\binom{N}{T}$ can be approximated to be N^T. Solving the above equation gives us:

$$m = \frac{T * lnN - ln\lambda}{lnq - lnT} \tag{5}$$

For the parameters of our choice for AACS, it is easy enough to use a spreadsheet on the formula 3 to show the relationship among these numbers. The following two graphs show this relationship when the number of device is 1 billion.





Interestingly, it takes almost the same number of movies(roughly $6T$) to achieve a super high confidence (below 0.0001%) as it does to achieve a moderately high confidence (below 0.1%) We also notice that, for a fixed-size coalition, when one uses a larger q, it takes fewer movies to detect traitors. Of course this is not surprising. A larger q means better traceability.

The traditional approach incriminates the highest score device, i.e. the device whose codeword is at the smallest Hamming distance from the pirated copy. This is same as the standard decoding rule for error correcting codes. Of course, you have to have confidence, based on your key assignment scheme, that a high-scoring device is not just an unlucky innocent device that just happened to have many movies in common with the attackers. A traceability code enables one to decode to the nearest neighbor of a pirate code and the nearest neighbor is deterministically a traitor.

A sufficient condition is shown in [4, 8] for a $t - traceability$ code.

Lemma 2. [4] Assume that a code $\mathcal{C}$ with length n and distance d is used to assign the symbols for each segment to each user and that there are t traitors. If code $\mathcal{C}$ satisfies

$$d > (1 - 1/t^2)n, \tag{6}$$

then $\mathcal{C}$ is an t-traceability-code.

Based on the above formula, if using the parameters of choice for AACS, a simple Reed-Solomon assignment for both inner and outer code will allow one to deterministically identify traitors in a coalition of nine after recovering 256 movies. In contrast, for the same coalition size, our algorithm takes 56 movies and the false positive rate can be low at 0.0001%. While we believe this already gives us super high confidence, it seems deterministic tracing takes a lot more movies in order to exclude even the tiny possibility of being wrong.

However, one cannot perform deterministic tracing in real applications like AACS, because in real life, the tracing agency rarely knows the size of the coalition in advance. As a result, the answers the tracing scheme gets are always qualified. For example, an answer might be as follows: “If N players are involved, it must be exactly this N. However, different innocent coalitions of $N + M$ players may have produced the same result.” In our probabilistic tracing, we suppose the coalition size is unknown and try to find a coalition that can explain all the recovered movies. If we succeed, the algorithm outputs a probability. This probability is both a confidence that the identified coalition contains guilty players *and* a confidence that the attack is of that size. However, the identified coalition may contain some innocent players, so we go to a refinement phase, where we examine the individual guilty probabilities of each player in the identified coalition. We can also calculate the probability that a completely different coalition could have incriminated the suspect coalition we found. This so-called false positive rate—i.e. the probability that all members in the suspect coalition are innocent—can be made to be arbitrarily small.

Please also note that the probabilistic tracing we have is also different from the probabilistic tracing in [3, 4]. Their goal is to make the probability of exposing an innocent user as small as possible, while we try to make the probability of catching the actual traitor to be reasonably high.

6 Simulation results

We have also performed simple simulations to confirm the above analysis. Because of the nature of the probabilistic detection, it means some false positive. For a coalition of size 4, we know it takes about 22 movies to detect the traitors with very high confidence. Of course, to confirm a very low probability like that would take an unreasonably large number of simulations. Instead, we used a test with a larger false positive rate, namely a 20 movie sequence. We randomly picked a coalition of size 4, and create 20 pirate movies out of the chosen 4 traitors. We tried both random and round robin methods for the traitors' strategy. We confirmed (at the 95% confidence level) that equation 2 holds. Similarly, for a coalition size of 6, from the formula we know it takes about 34 movies to reach a confidence 0.005% false positive. We simulated using only 32 movies. After 100 simulations we tested, we found 6 cases that involve a completely innocent coalition, which is consistent with the bound from equationa 2, which is 9.5%.

We also notice a slight difference of the behavior when we use round robin to create the pirate movies than when we use random selection. In the case of random selection, one player often contributes a lot. In other words, the highest score is significantly above the average number that would evenly distribute among attacker players. For example, in the case of coalition of size 4 and with 20 movies, many times the highest score is above 6, sometimes 9 or 10. That particular player dominates all the coalitions found, so much so even the other guilty players are identified with low confidence. This partially explains why traditional score ranking could work to some extent against the random selection attacker strategy. But with our new tracing scheme, the other coalition members are nonetheless found, unless they made a negligible contribution to the attack.

On the other hand, in the case of round robin, the movies are contributed evenly from the attackers. None of the attackers particularly stands out, as some do with random selection. It is hard to incriminate the highest scoring player in this case. For example, in the case of a coalition of size 4 and with 20 movies, all 4 players explain 5 movies. In our simulation, in most cases, the new tracing algorithm found the exact one coalition that together can explain all 20 movies. Once again, this explains why our new tracing algorithm is more efficient than the traditional approach.

7 Conclusions

In this paper, we study the problem of tracing the legitimate users (traitors) who instrument their devices and illegally resell the pirated copies by redistributing the content or the decryption keys on the Internet. In particular, we focus on distributing prerecorded movies in the context of Advanced Access Content System copy protection standard for the next generation of high-definition DVDs.

Given the restrictions on the number of variations within a movie and the billion devices needing to be supported, we measure the efficiency of tracing (traceability) using the number of recovered movies it takes to identify the traitors in the coalition. We have designed the first tracing scheme that is efficient enough for commercial use. Our probabilistic tracing scheme achieves super linear traceability. Different from existing approaches which try to detect traitors one by one, we detect traitors in the coalition all together. This idea enables faster tracing with less recovered content, at the cost of higher computational overhead. The traceability becomes one of the enabling factors for its commercial use. In the future, we will continue to improve its traceability, not only theoretically, but also by taking into consideration of real implementations. We are also interested in overcoming different barriers during its deployment.

References

1. http://www.aacsla.com
2. A. Fiat and M. Naor, “Broadcast Encryption,” *Crypto'93, Lecture Notes in computer science*, Vol. 773, pp480-491. Springer-Verlag, Berlin, Heidelberg, New York, 1993.
3. B. Chor, A, Fiat and M. Naor, “Tracing traitors,” *Crypto'94, Lecture Notes in computer science*, Vol. 839, pp480-491. Springer-Verlag, Berlin, Heidelberg, New York, 1994.
4. B. Chor, A, Fiat, M. Naor and B. Pinkas, “Tracing traitors,” *IEEE Transactions on Information Theory*, Vol 46(2000), 893-910.
5. A. Fiat and T. Tassa, “Dynamic traitor tracing,” *Crypto'99, Lecture Notes in computer science*, Vol. 1666, pp354-371. Springer-Verlag, Berlin, Heidelberg, New York, 1999.
6. R. Safani-Naini and Y. Wang, “Sequential Traitor tracing,” *IEEE Transactions on Information Theory*, 49, 2003.
7. Tran van Trung and Sosina Martirosyan, "On a class of Traceability Codes", *Design, code and cryptography*, 31(2004), pp 125-132.
8. J. N. Staddon, D.R. Stinson and R. Wei, “Combinatorial properties of frameproof and traceability codes,” *IEEE Transactions on Information Theory*, 47 (2001), 1042-1049.
9. D.R.Stinson and R. Wei, “Combinatorial properties and constructions of traceability schemes and frameproof codes,” *SIAM Journal on Discrete Mathematics*, 11:41-53, 1998.
10. anonymous
11. G. Tardos, "Optimal Probabilistic fingerprint codes", in proceedings of the *Theory of Computing*, pp. 116-125, June 9-11, 2003, San Diego, CA.