Renewable Traitor Tracing: A Trace-Revoke-Trace System For Anonymous Attack

Hongxia Jin and Jeffery Lotspiech

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

Abstract. In this paper we design renewable traitor tracing scheme for anonymous attack. When pirated copies of some copyrighted content or content decrypting key are found, a traitor tracing scheme could identify at least one of the real users (traitors) who participate in the construction of the pirated content/key. When traitors are identified, the renewable scheme can revoke and exclude the decryption keys used by the traitors during piracy. Moreover, the revocation information included in the newly released content needs not only to disallow traitors to playback the new content, but also provide new tracing information for continuous tracing. This trace-revoke-trace system is the first such system for anonymous attack. It poses new challenges over the trace-revoke system that has extensively studied in the literatures for the pirate decoder attack. We hope the technologies described in this paper can shed new insights on future directions in this area for academia research.

Keywords: Content protection, traitor tracing, broadcast encryption, anti-piracy.

1 Introduction

This paper is concerned with the protection of copyrighted materials. There are many business scenarios that content needs to be distributed through broadcast channels. Examples of these business models include pay-TV systems (Cable companies) or movie rental companies like Netflix, and massively distributing prerecorded and recordable media. A broadcast encryption and traitor tracing scheme can be used to protect the content copyright and make sure the content can only be recovered by a privileged group of users. Since broadcast content is usually large, hybrid encryption system can be used to keep asymptotically optimal transmission rate (i.e., the ratio between cipher text size and plaintext size). More concretely, each device is assigned a set of unique secret keys (called device keys) but the broadcaster selects another random key (called media key) to indirectly encrypt the content. Distributed together with the content is a structure called Media Key Block (MKB) where the media key is encrypted by valid device keys again and again. Only those enabled devices can decrypt the

J. Biskup and J. Lopez (Eds.): ESORICS 2007, LNCS 4734, pp. 563–577, 2007.

MKB and obtain the correct media key to decrypt the content. The disabled devices, with their device keys revoked, cannot decrypt the MKB correctly to access the content. Traitor tracing schemes are used to identify who have involved in the pirate attack when the disclosed pirate evidence is recovered.

There are three major pirate attacks in the above content protection system. The forensic evidences are different for different pirate attacks, which correspond to different types of traitor tracing system.

1. Pirates disclose secret device keys by building a clone pirate decoder.
2. Pirates disclose content encrypting key (media key): pirates stay anonymous.
3. Pirates disclose decrypted content: pirates stay anonymous.

The first type of pirate attack has been studied extensively. It is called "pirate decoder attack". Attackers hack legitimate players, extract secret device keys from the players and build a pirate decoder that will decrypt and extract the plaintext content. The pirates could widely redistribute the pirate decoders so that anyone can extract the clear content for themselves. For example, pirate attackers broke the CSS (Content Scrambling System), built and widely distributed DeCSS, a pirate program for decrypting encrypted DVD content. In order to find out which device keys are in the decoder, the tracing scheme interacts with the pirate decoder using carefully crafted cipher text (we call it "forensic MKB") and observes the decoder's outcome. Most existing broadcast encryption and traitor tracing schemes [2,3,4,5,6,7,8,9,10,11,12] targeted on this type of "pirate decoder attack". Some of these schemes are combinatorial [4,5,2], some are algebraic [6,9,11,12].

In this paper we are concerned with a different attack, namely, anonymous attack. In anonymous attack, instead of distributing a pirate decoder, attackers redistribute the actual content encrypting key or the decrypted content. Keep in mind when using hybrid encryption the content encrypting key and the clear content is same for every user/player and cannot be used to identify traitors. To defend against these types of anonymous attacks, one needs different versions of the content and decryption keys (media key) for different users.

Unfortunately in most one-to-many content distribution channels, like massively distributing DVDs, it is generally infeasible to prepare and send individualized (e.g., watermarked) movie discs to each user. On the other hand, it is also infeasible, usually for security reasons, to customize each copy at the receiving end[1]. 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 players in a copyright attack by examining the

[1] 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?

variations recovered in the unauthorized copies of the movies. We together with [15,16,17,18,19] use this model to defend against anonymous attack. It is interesting to notice that all schemes for this attack are combinatorial. Traitor tracing schemes can be public-key based or secret symmetric key based. Combinatorial systems are typically designed for secret key setting.

AACS [1], *Advanced Access Content System*, was founded in July 2004 by eight companies, Disney, IBM, Intel, Matsushita, Microsoft, Sony, Toshiba, and Warner Brothers. It develops content protection technology for the next generation of high-definition DVD optical discs. Compared to the previous DVD CSS system, which could not revoke users because it is a shared-key scheme, AACS adopts a broadcast encryption scheme shown in [2]. The pirate attack this scheme was concerned with is the pirate decoder attack. The mechanism is designed to exclude clones pirate devices, such as the infamous "DeCSS" application used for copying "protected" DVD Video disks. It utilizes hybrid encryption and media key block (MKB). Once the attackers have been detected, they are excluded from newly released content because the new media key blocks in the new content exclude the keys known to the attackers. The detection of the traitors is performed by the traitor tracing approach in the scheme [2] when a pirate decoder is found. This responds to the first pirate attack mentioned above.

However, the AACS founders do not believe that it is sufficient to only respond to pirate decoder attack. AACS found it desirable to be able to respond to the anonymous attacks discussed above. To have a practical useful system, AACS is concerned with problems arise during the lifetime of a traitor tracing system. Basically AACS demands a traitor tracing system for anonymous attack that not only traces traitors, but also revokes traitors and can continue tracing traitors throughout its lifetime. Unfortunately existing schemes [16,17,15] for anonymous attack focus on the first part of a "trace-revoke-trace" system. They assume the detected traitor can be disconnected from the system in some way and the tracing can simply be repeated. The multiple needs from AACS on a practical tracing system are therefore still unsatisfied. AACS needs a complete trace-revoke-trace system for anonymous attack. That is the focus on this paper.

1.1 Main Results

The main contribution of this paper is that we provided a complete and practical trace-revoke-trace system to meet AACS' demand to defend against anonymous attack. The technologies we will describe in this paper have been adopted by this new industry content protection standard to protect the next generation of high-definition optical DVDs.

We build our trace-revoke-trace system on top of an existing tracing scheme [20] for anonymous attack. The first part "trace" of our trace-revoke-trace system uses the key assignment described in [20]. On top of that, we go on adding revocation capability similar to the design of Media Key Block in a broadcast encryption system. The traitor tracing keys serve the role of the device keys in a broadcast encryption system, and the actual content encrypting keys in our traitor tracing scheme serve the role of the media key in a broadcast encryption

system. Of course, there are multiple versions of content encrypting key in our scheme while there is only one media key in a normal broadcast encryption system. Unfortunately, the above combined scheme cannot support multi-time continued tracing. It is possible that the license agency does not gain any useful information to continue detecting the unexposed traitors. We identified this new challenge and improved our scheme to support multi-time tracing.

Adding renewability and continued traceability was a crucial enabling factor for this first time adoption for large scale commercialization of a tracing traitor technology for anonymous attack. We hope the experience we gained when developing this complete system can shed new insights on future research directions on some problems that we believe have been overlooked by research community so far. We will discuss a little bit on these issues in the concluding Section 7.

In rest of the paper, we will describe our trace-revoke-trace system in three continuous sections. Section 3 is for the first part "trace" in the trace-revoke-trace system. We will summarize the existing tracing scheme shown in [20]. Section 4 is for the second "revoke" part of our trace-revoke-trace system. We will show how to add renewability into the scheme. Section 5 is for the third part of our trace-revoke-trace system, there we will show the challenges to provide continued traceability and show our solution to support multi-time tracing. We analyze the revocation capability and continued traceability in Section 6. We conclude in Section 7 with future work.

2 Pirate Model

AACS founders find it acceptable to makes the following marking assumption on the pirate model. 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. The exact reason that has made AACS to accept this model is outside the scope of this paper. Indeed, this so-called marking assumption is often made by other traitor tracing schemes shown in the literatures. 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 First Part: Trace

In this section we show the restrictions on designing a practical traitor tracing system for AACS, and show how AACS can use the scheme shown in [20] as its first part.

In the AACS context, each movie is divided into multiple segments, among which n segments are chosen to have differently marked variations. Each of these n segments has q possible variations. Each playing device receives the same disc with all the small variations at chosen points in the content. However, each device plays back the movie through a different path, which effectively creates a different movie version. Each version of the content contains one variation for each seg-

ment. Each version can be denoted as an n-tuple or *codeword* $(x_0, x_1, \ldots, x_{n-1})$, where $0 \leq x_i \leq q-1$ for each $0 \leq i \leq n-1$.

As one can imagine, the variations take 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. For a normal 2-hour movie, it corresponds to 8 additional minutes (480 seconds) of video. 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.

Unfortunately these requirements are inherently conflicting. Let us show some intuition on the conflicts between these parameters. Take a look at a code $[n, k, d]$, where n is the length of the codewords, k is the source symbol size and d is the Hamming distance of the code, namely, the minimum number of symbols by which any two codewords differ. q is the number of variations. Mathematically these parameters are connected to each other. The number of codewords is q^k, and Hamming distance has the property that $d <= n - k + 1$. q is also related to n, for example, for a "maximal difference separable" (MDS) code, $n <= q$. We know the number of variations q decides the extra bandwidth needed for distributing content. Without variations, $q = 1$. The extra bandwidth needed for the content is $(q-1) * length\_of\_each\_variation * n$. The Hamming distance d decides its traceability. To defend against a collusion attack, intuitively we would like the variant assignment to be as far apart as possible. In other words, the larger the Hamming distance is, the better traceability of the scheme. On the other hand, to accommodate a large number of devices, e.g. billions, intuitively either q or k or both have to be relatively big. Unfortunately a big q means big bandwidth overhead and a big k means smaller Hamming distance and thus weaker traceability. It is inherently difficult to defend against collusions.

In order to meet these practical requirements, AACS uses the key assignment scheme shown in [20]. The basic idea there is to use two-level assignment and concatenate them. 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, its "inner code" generates 256 versions for each movie out of all the possible 16^{15} combinations. Any two of the 256 versions are guaranteed to differ at 14 points out of the entire 15 points in the 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, the scheme managed to avoid having a big number of variations at any chosen point but can still accommodate the billions of devices we anticipate. The inner code and outer code assignments can be random or systematic. For example, both inner and outer codes can use Reed-Solomon codes. Suppose each variations takes 2-second clip, the extra video needed in this example is 450 seconds, within the 10% constraint placed by the studios. This example can accommodate more than 4 billion devices. In fact, there does not exist a single level MDS code

that can satisfy all the practical requirements. The two-level concatenated code is essential to meet the practical requirements for AACS.

During playback time, each device needs to decrypt exactly one variation for each segment. Using the above parameters, a device needs to know $255 * 15$ variation encrypting keys. 255 corresponds to the outer code length for the 255 movies in the sequence, 15 corresponds to the inner code length. However, by adding a level of indirection, each device only stores 255 movie version keys corresponding to the 255 movies in the sequence. For each movie, the device can use its movie version key to unlock a table on the DVD disc that contains the actual variation encrypting keys for that movie. In other words, the variation encrypting keys which are assigned using "inner code" are embedded on the DVD disc. These movie version keys which are assigned using "outer side" are burned into the devices during manufacture time. Those outer code keys are called "sequence keys" in AACS.

The sequence keys for AACS are assigned from a large matrix. Each column corresponds to a movie. The number of columns corresponds to the number of movies considered in the sequence. The rows for each column correspond to multiple versions of the sequence keys for that movie. For example, if the variations inside the movie segments create 256 versions for each movie and we consider 255 movies in a sequence, then the matrix is 255 by 256. Each device is assigned a set of 255 keys, exactly one key from each column, corresponding to the 255 movies in the sequence. The key for each column has 256 versions. Many players will receive any given sequence key, but no two players will receive exactly the same set of 255 keys. Again, these sequence keys are placed in the players at manufacturing time. Figure 1 is an example of the matrix organization of the keys.

Based on the pirate model we mentioned in Section 2, the attackers can collude and construct the pirate copy of the content/key based on the available versions to them. When the license agency recovers pirated movies/keys, it tries to match the recovered movies/keys with the versions assigned to the devices to detect traitors. This detection algorithm must handle the collusion attack when attackers collude in the piracy. It is possible to use the typical highest-score approach as shown in [20,16]. AACS chose to use a more efficient detection algorithm. However, the actual detection algorithm is out of the scope of this paper.

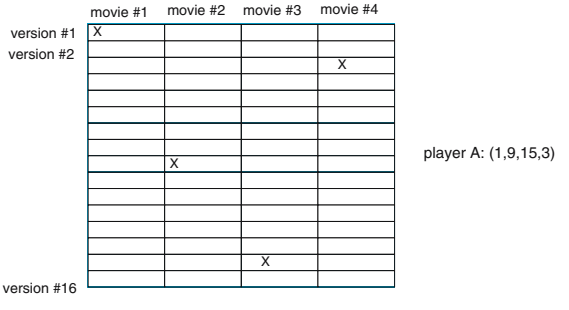

Fig. 1. Key assignment from a matrix

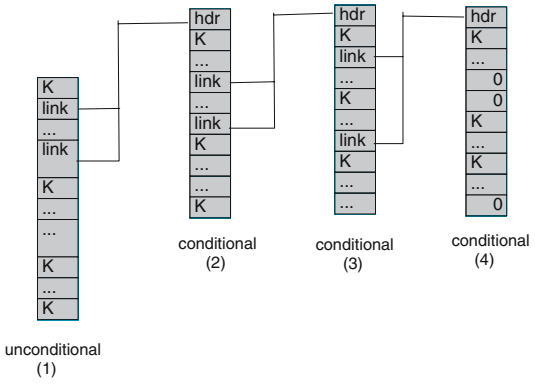

Fig. 2. Sample SKB

4 Second Part: Revocation

A traitor tracing scheme is defined to be finding at least a traitor if though there may exist a coalition. Indeed, all traitor tracing schemes [15,16,17,20] for anonymous attack stop when they detect a traitor. They assume this traitor can be disconnected in some way. If there still is piracy, the same scheme just needs to be repeated.

In reality, how does one disconnect a traitor technically? It is by rendering the compromised keys no longer usable for future content. In other words, a traitor tracing system must also be able to revoke compromised keys to be actually useful in real world.

Of course, many devices might share a single compromised key. Therefore, revocation of a single key is impossible. On the other hand, revocation of a unique set of keys is very possible; in fact, that is precisely what the Sequence Key Block (SKB) achieves.

The fundamental principle is that no two devices have many keys in common, so even if the system has been heavily attacked and a significant fraction of the Sequence Keys is compromised, all innocent devices will have many columns in which they have uncompromised keys. The purpose of the Sequence Key Block is to give all innocent devices a column they can use to calculate the correct answer, while at the same time preventing compromised devices (who have compromised keys in all columns) from getting to the same correct answer.

In an SKB there are actually many correct answers, one for each variation in the content. For the purpose of explanation, however, it is helpful to imagine that a single SKB is producing a single answer. We will call that answer the output key.

As shown in Figure 2, the SKB begins with a first column, called the unconditional column. By column, we mean a column of Sequence Keys in the matrix will be used to encrypt. (To be precise, the key used to encrypt is derived from the Sequence Key, not the Sequence Key itself.) The first column will have an

encryption of the output key (denoted K in the figure) in every uncompromised Sequence Keys cell. Devices that do not have compromised keys in that column immediately decrypt the output key. Devices, both innocent and otherwise, that do have compromised keys instead decrypt a key called a link key that allows them to process a further column in the SKB. To process the further column they need both the link key and their Sequence Key in that column. Thus the subsequent columns are called conditional columns because they can only be processed by the device if it were given the necessary link key in a previous column.

The subsequent additional conditional columns are produced the same way as the first column: They will have an encryption of the output key in every uncompromised Sequence Keys cell. Devices with a compromised key will get a further link key to another column instead of the output key. However, after some number of columns depending on the actual number of compromised keys, the AACS licensing agency will know that only compromised devices would be getting the link key; all innocent devices would have found the output key in this column or in a previous column. At this point, rather than encrypting a link key, the agency encrypts a 0, and the SKB is complete. All innocent devices will have decrypted the output key, and all compromised devices have ended up decrypting 0.

How do the devices know they have a link key versus the output key? The short answer is they do not, at least not at first. Each conditional column has a header of known data (for example, "DEADBEEF16") encrypted in the link key for that column. The device decrypts the header with the key it currently has. If the header decrypts correctly, the device knows it has a link key and processes the column. If it does not decrypt correctly, the device knows it has either the output key or a link key for a different column. When the device reaches the end of the SKB without decrypting 0, it knows it must have an output key. Note that this device logic allows the licensing agency to send different populations of devices to different columns by having more than one link key output from a single column. For example, in the figure, column (1) links to both column (2) and column (5). This flexibility can help against certain types of attacks.

The preceding description is the basics of an AACS SKB and described in a simplified version. In an actual AACS SKB there is not a single output key, but multiple output keys called Variant Data D_v.

The SKB is generated by the AACS license agency and allows all compliant devices, each using their set of secret Sequence Keys to calculate the Variant Data, D_v, which in turn allows them to indirectly decrypt a table that contains the actual movie variation encrypting keys for the playback path of the movie assigned to that device. If a set of Sequence Keys is compromised in a way that threatens the integrity of the system, an updated SKB can be released that causes a device with one or more compromised sets of Sequence Keys to calculate invalid Variant Data. In this way, the compromised Sequence Keys are revoked by the new SKB. In fact, if a device correctly processes an SKB using Sequence Keys that are revoked by that SKB, the resulting final D_v will have the special value 0000000000000000000016. This special value will never

be an SKBs correct final D_v value, and can therefore always be taken as an indication that the devices Sequence Keys are revoked. Device behavior in this situation is implementation defined. As an example, a device could exhibit a special diagnostic code, as information to a service technician. If at any point, a device calculates a D_v value of zero, it should discontinue processing of the SKB and may conclude that it has been revoked.

5 Third Part: Trace Again After Revocation

The above solution provides a key management scheme that not only can trace traitors but also can revoke compromised keys. A real world traitor tracing system must be able to trace again after revocation. Although none of the existing traitor tracing system for anonymous attack even provides revocation capability, there are quite some trace-revoke systems out there for pirate decoder attack. Unfortunately there is big difference when coming to design a trace-and-revoke system for clone decoder attack and anonymous attack. In this section, we shall describe the new challenge and our solution for that.

5.1 Another Type of Collusion Attack: Combine Revoked Keys with Non-revoked Keys in Future Attacks

For pirate decoder attack, the license agency has the clone box in the testing lab. The tracing agency carefully produces some testing messages (called forensic Media Key Blocks) and feeds into the box in order to trace keys contained in the clone box. The real production MKB and the forensic MKB are two different things. Forensic MKBs are for tracing purpose only, they are not necessary production MKBs. On the other hand, real production MKB used in a broadcast encryption scheme, like [2], only needs to contain the revocation information and disable the detected compromised keys. It itself doesnt have to be able to trace. However, for anonymous attack, it demands the same SKB containing the revocation information in the newly broadcasted content to not only revoke the compromised keys but also enable continued tracing of new traitors. It poses new challenges to our design.

This is best understood by example. The bulk of a movie is encrypted by the media key, which in turn is calculated by each device using its set of device keys. At the point of variations in the movie, the media key alone does not suffice, the device must also use the variant key it has at that point. In a clone decoder attack, all that is needed is to exclude the clone's compromised device keys. The clone can no longer play the movie. In an anonymous attack, it is necessary but not sufficient to exclude the attacker's device keys; the attacker's variant keys must also be made unusable. If not, if the attackers are able to obtain a new set of device keys, they simply combine them with the previously-used variant keys and give the tracing agency no new information. This tactic is useless against the clone decoder attack because the tracing agency already has the clone in the lab.

The challenge here is to make sure the newly released SKB can continue to provide tracing information to the license agency to enable continued tracing.

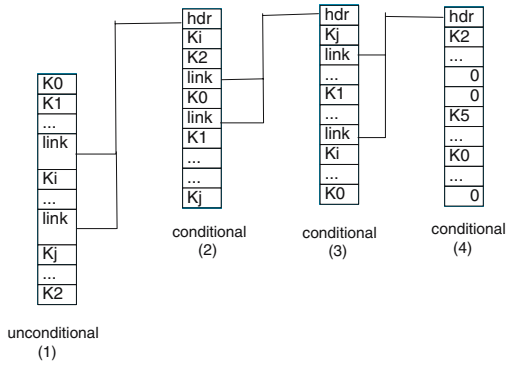

Fig. 3. It is possible that the new SKB will not provide new tracing information for continued tracing

As we know, each column in a SKB contains an encryption of the output key in every uncompromised sequence key's cell. More precisely, in every uncompromised sequence key's cell in each column, it contains an encryption of one of the variant data $D_{vi}(0 \leq i < k), k$ is the number of different variant data in the system. Notice the same set of variant data are encrypted in each column, although they are distributed differently in different columns. For a particular variant data D_{vi}, it can be obtained from any column in the SKB. An uncompromised device will process SKB and obtain a correct variant data from the first column in SKB that it has a non-revoked key. However, when attackers collude and the system still has undetected attackers at large. The attacker can mix-and-match their revoked keys and non-revoked keys when processing SKB. In turn they have multiple ways to process SKB and get a valid variant data to play back the content. They can choose in which column they want to use a non-revoked key to get a valid variant data. It does not have to be in the first column. Moreover, different keys need to be used in different columns to obtain the same variant D_{vi}. When the license agency observes a pirate copy corresponding to a particular variant data D_{vi}, since it can be obtained from any column, the license agency has no way to know which key has been used in obtaining that variant data. The entire path that the undetected traitors goes through to process SKB can even look like from an innocent device or from a path that was never assigned to any device, thus untraceable.

The figure 3 illustrates the issue discussed here. Keep in mind that the output key has multiple valid versions. If the attackers combine the revoked keys with the keys that have not been detected, it is not always possible to know from which column the SKB processing ends to get a valid key.

To force the undetected traitors to reveal the keys they use when processing SKB, we must make sure each column gets different variations so that when recovering a key/variation, the scheme knows from which column it comes from. Only by observing that, the tracing scheme can continue tracing. Unfortunately that means the q variations have to be distributed among the columns contained

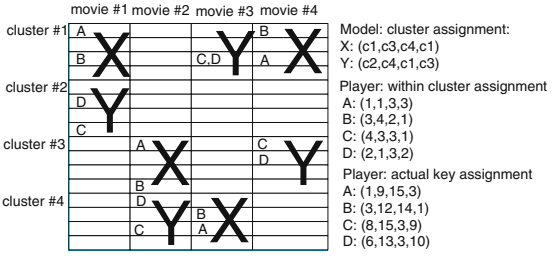

Fig. 4. Key assignment using "slots"

in the SKB. Each column only effectively gets q/c variations where c is the number of columns in the SKB. It is clear that traceability degrades when the effective q decreases. When the number of columns c becomes big enough, the traceability degrades to so low that it basically becomes untraceable. The scheme is overwhelmed and broken in that case. As a result, that puts a limit on the revocation capability of the scheme.

5.2 Improved Solution

In order to improve the above scheme and lift the limit on revocation capability, we have designed a two phase defense. In order to use this defense, the scheme assigns the sequence keys as shown in Figure 4 instead of Figure 1.

Basically we used a new concept called "slot". Now the rows in the key matrix are grouped into clusters. A slot is defined to be an assignment of row clusters, one cluster for each column. At any given column, two slots are either identical or completely disjoint. Slots can be assigned to individual manufacturers/models and the keys within the clusters are assigned to the devices made by the manufacturer/model. In effect, the outer code that is used to assign sequence keys to devices is now itself a two-level code. The first level codes assign clusters to the manufacturer/models X and Y and the second level codes assign keys to players A,B within model X, and players C, D within model Y. Model X gets the slot (1,3,4,1), which means it is assigned cluster #1 for movie #1, cluster #3 for movie #2, and etc. Note that the second level code is the assignment inside the cluster. For example, player A gets *(1,1,3,3)* within the clusters assigned to model X, which makes its actual key assignment be $(1, 9, 15, 3)$ from the key matrix.

In AACS context, we anticipate about 4000 manufacturers/models. We divide all the rows in each column to have 64 clusters. Using Reed-Solomon code, $q = 64$, it takes $k = 2$ to accommodate 64^2 slots. Suppose we have $4K$ keys in each column, there will be 64 keys in each cluster. Again using Reed-Solomon code, $q = 64$, it takes $k = 3$ to accommodate 64^3 devices within each slot. The assignment can totally accommodate 64^5 devices. Each slot can be assigned to one manufacturer/model, A big manufacturer would, of course, overflow a single slot. He would just have more than one slot.

In the two-phase defense, when pirated movies/keys are found, the first SKB's would determine the slot used in the attack (or slots, but that is unlikely). Since the slot is assigned from the first level by using Reed-Solomon code $q = 64$, $k = 2$, there are only 64 variations needed per column in this case. Recall the "inner code" generates totally 256 variations for each movie. One can use 4 columns in the SKB and there is no problem with dividing the 256 variations across 4 columns. Each column would get 64 variations, which is all we need per column. By Reed-Solomon code's property, it takes only two ($k = 2$) movies to uniquely detect the slot. The above attack scenario does not work here.

Once the licensing agency detects the slot, it can produce new SKB's that are only trying to detect the device within the slot. In the SKB, all other slots in the column(s) would go to a single variation that we would expect would never be recovered. We would use all the remaining variations within the single slot. Again, the above attack scenario is not much a problem, because we can get up to 4 columns and still have unique keys for each variation. As long as we only need 4 columns in SKB, the above attack cannot work.

However, it still puts a limit on how many devices the scheme can revoke before it is overwhelmed. A single slot can be overwhelmed if it gets a lot of attacks. For example, if there are just 32 devices revoked from a single manufacturer, then 50% of the keys in the slot are compromised, and the SKB takes about 18 columns to winnow out that manufacturer's innocent devices. The 18 columns means the above attack can cause a problem, even with the two-phase approach. Because of this, a larger number of keys per column is preferred, for example, $16K$ keys/column would be a better number. These keys are still grouped by 64 clusters for slot purposes. Each slot can have 256 keys instead of 64 keys. Then we would be back down to 6 columns. Of course, anything more than 4 columns introduces the above attack problem, but it is still manageable.

6 Revocation Capability and Traceability

More formally, the revocation capability is calculated by the following formula. Suppose the number of rows in the matrix is m, p is the acceptable maximum probability for an innocent device to be revoked when revoking the actual guilty devices. r is the number of guilty devices to be revoked in SKB. c is the number of columns in SKB. The system still survives when the following holds.

$$(1 - (1 - 1/m)^r)^c < p \tag{1}$$

This formula can be used to determine how many columns c needed in a SKB when the licensing agency wants to revoke r devices. Due to the above attack scenario, there is a limit on the number of columns. We can also easily see that, the limit on the number of columns c in SKB induces a limit on the number of revoking devices that the system can survive. For example, suppose sequence keys are assigned from a matrix of 4096 by 256, there are 4096 rows.

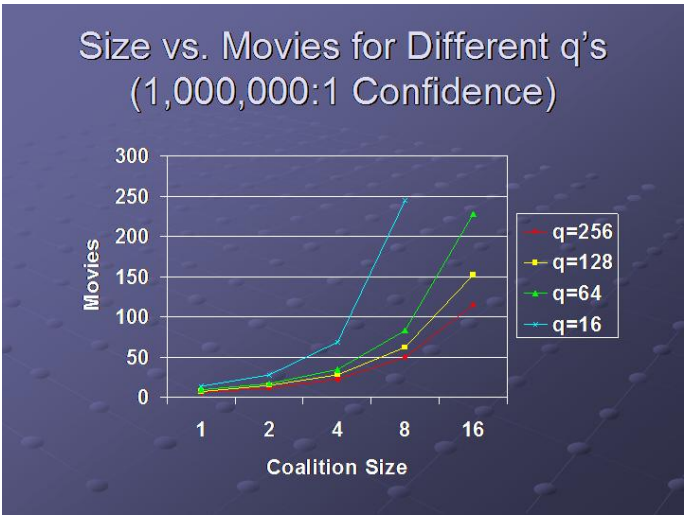

Fig. 5. Traceability with different q

We divide rows into 64 clusters for slot assignment. Suppose the acceptable rate for revoking innocent devices is $1/1,000,000$. Suppose there can be at most 18 columns in SKB. In the case of random hacking where we assume the attackers are distributed randomly from any slot and there is no evil manufacturers, Substituting $m = 4096$ gives us that maximum number of guilty devices that the system can revoke but still get useful tracing information for continued tracing. In the case of evil manufacturers or most hacking occurs in one slot, breaking one slot breaks the entire system. In the above example, there are 64 cluster in a $4096X256$ matrix, there are 64 rows in each cluster for slot assignment. Substituting $m = 64$ gives us that the system can maximally survive up to 32 revoked devices within a slot.

We have performed preliminary simulation on how many columns the scheme needs in its SKB in terms of the number of devices that needs to be revoked. It confirms the above formula 1.

The traceability is measured by the number of movies it takes to recover in order to detect traitors in a coalition of T traitors. Of course the traceability depends on the actual detection algorithm used when a series of pirated movies are recovered. We didnt show the actual traitor detection algorithm in this paper since it is out of the scope of this paper. However, we have used the same detection algorithm on different q to show the impact of deceased q on traceability. The traceability result is shown in graph 5. It includes both the two-phase solution and the basic single-phase scheme. There is different traceability in the life of the system depending on how many devices are currently being revoked in the SKB. The more devices revoked, the more columns it needs in SKB, the smaller the effective q is, the worse traceability.

7 Conclusions

In this paper, we study the problem of traitor tracing for anonymous attack where the legitimate users (traitors) illegally redistributing the clear content or the decryption keys on the Internet. This is a different type of attack with the pirate decoder attack which has been extensively studied in literature. In this paper we designed a complete trace-revoke-trace system for anonymous attack. Our system has been adopted by the new industry content protection standard AACS to protect the next generation DVDs.

Existing schemes for anonymous attack only focus on detecting a traitor. They assume the detected traitor can be disconnected in some way. In this paper we designed a system that can actually disable the detected traitor by revoking the identified compromised keys. We also studied the continued tracing problem which has been overlooked by existing schemes. They assumed that the tracing can simply be repeated for new traitors and the traceability will be same as before. We showed why this is not always possible to perform multi-time tracing and how traceability can be decreased. We have also analyzed its traceability and revocation capability. The revocation and continued tracing capability is one of the enabling factor for its adoption to be the first large scale commercialization of tracing traitor technology for anonymous attack.

We hope our work described in this paper sheds new insights on future directions in this area. So far, revocation is considered inside a broadcast encryption system, and tracing is considered inside a traitor tracing system. Broadcast encryption and traitor tracing have been viewed as two orthogonal problems. However, our experience working on a solution for real world has taught us that a traitor tracing scheme without revocation capability is practically useless in real world. Revocation should be considered together with tracing.

In fact, this is also true in the case of pirate decoder attack. Even though pirate decoder attack has been studied very extensively in literatures, they did not realize that tracing must be accompanied by revocation in order to be useful. Furthermore it is not always easy to add revocation on top of tracing if revocation is an afterthought. For example, for the very recent traitor tracing scheme [8] appeared in Eurocrypto 2006, if one needs to revoke a detected hacking device i, one would also have to revoke devices $i + 1...N$. Of course this is unacceptable.

Furthermore, a traitor tracing system must provide multi-time continued traceability in order to be useful. It is not as easy as simply repeating the same tracing. Again this issue can arise for pirate decoder attack too [11]. For example, after a pirate decoder of certain type is brought to the testing lab and rendered unusable, if there are undetected traitors in a coalition out there, the attackers can build a new clone decoder combining the previously revoked keys with untraced keys. This may or may not put difficulty on providing continued traceability. We believe this issue needs to be further studied in the future.

In the future, we will also continue to improve our revocation capability as well as traceability, not only theoretically, but also by taking into considerations of other features existing in a real system.

References

1. Pre-recorded Video Book: v0.91, ch. 4, http://www.aacsla.com/specifications
2. Naor, D., Naor, M., Lotspiech, J.: Revocation and Tracing Schemes for Stateless Receivers. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 41–62. Springer, Heidelberg (2001)
3. Fiat, A., Naor, M.: Broadcast Encryption. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 480–491. Springer, Heidelberg (1994)
4. Chor, B., Fiat, A., Naor, M.: Tracing traitors. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 480–491. Springer, Heidelberg (1994)
5. Chor, B., Fiat, A., Naor, M., Pinkas, B.: Tracing traitors. IEEE Transactions on Information Theory 46, 893–910 (2000)
6. Naor, M., Pinkas, B.: Efficient Trace and Revoke Schemes. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 1–20. Springer, Heidelberg (2001)
7. Boneh, D., Gentry, C., Waters, B.: Collusion Resistant Broadcast Encryption With Short Ciphertexts and Private Keys. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 258–275. Springer, Heidelberg (2005)
8. Boneh, D., Sahai, A., Waters, B.: Fully Collusion Resistant Traitor Tracing With Short Ciphertexts and Private Keys. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 573–592. Springer, Heidelberg (2006)
9. Boneh, D., Franklin, M.: An efficient public key traitor tracing scheme. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 338–353. Springer, Heidelberg (1999)
10. Boneh, D., Waters, B.: A collusion resistant broadcast, trace and revoke system. ACM Communication and Computer Security (2006)
11. Kurosawa, K., Desmedt, Y.: Optimum traitor tracing and asymmetric schemes. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 145–157. Springer, Heidelberg (1998)
12. Chabanne, H., Phan, D.H., Pointcheval, D.: Public traceability in traitor tracing schemes. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 542–558. Springer, Heidelberg (2005)
13. Stinson, D.R., Wei, R.: Key preassigned traceability schemes for broadcast encryption. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, Springer, Heidelberg (1999)
14. Gafni, E., Staddon, J., Yin, Y.L.: Efficient methods for integrating traceability and broadcast encryption. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (1999)
15. Fiat, A., Tassa, T.: Dynamic traitor tracing. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 354–371. Springer, Heidelberg (1999)
16. Safani-Naini, R., Wang, Y.: Sequential Traitor tracing. IEEE Transactions on Information Theory 49 (2003)
17. van Trungand, T., Martirosyan, S.: On a class of Traceability Codes. Design, code and cryptography 31, 125–132 (2004)
18. Staddon, J.N., Stinson, D.R., Wei, R.: Combinatorial properties of frameproof and traceability codes. IEEE Transactions on Information Theory 47, 1042–1049 (2001)
19. Stinson, D.R., Wei, R.: Combinatorial properties and constructions of traceability schemes and frameproof codes. SIAM Journal on Discrete Mathematics 11, 41–53 (1998)
20. Jin, H., Lotspiech, J., Nusser, S.: Traitor tracing for prerecorded and recordable media. In: ACM DRM workshop, ACM Press, New York (October 2004)
21. Tardos, G.: Optimal Probabilistic fingerprint codes. In: Proceedings of the Theory of Computing, San Diego, CA, pp. 116–125 (June 9-11, 2003)