Abstract

Digital fingerprinting technologies are commonly used for identifying the users who make illegal copies of digital data; such task is achieved by assigning a unique codeword to each copy of the data and registering fingerprinted copies to users. Due to the uniqueness of the digital fingerprint, users are able to detect its existence by comparing several copies, which means users are able to modify or erase the detected codeword before releasing a pirate copy. To protect the fingerprint data from being destroyed, a collusion-secure fingerprinting scheme is needed.

This paper shows how three fingerprinting schemes, namely, Wagner's scheme, Boneh and Shaw's scheme and most importantly, Tardos' scheme, are implemented. In addition, experiments are conducted to evaluate their performance, especially when they are under attacks. For Tardos' scheme, extra criteria are taken into account, such as the time spent on code generation and memory usage. Moreover, some factors that are stopping fingerprinting technologies from being widely used are briefly discussed at the end of the report.

Table of Contents

1. Introduction

Intellectual property privacy is a problem that has been bothering authors, publishers and consumers for a long time. An American study claimed that film piracy alone cost the economy 20 billion dollars in the US in 2006 [1]; in the same year, software piracy resulted in a loss of 34 billion dollars worldwide [2] and the number jumped to 50 billion dollars in 2008, according to a study conducted by The Business Software Alliance (BSA) [3].

Many efforts have been made to stop the piracy; a typical example of anti-privacy methods is Digital Rights Management (DRM), which is a generic technology for enforcing access control. It is a collection of many mechanisms and algorithms, for example, Apple Inc. uses FairPlay to manage the music sold/played on iTunes Store or iPod, so that their uses are only able to play the music on a limited number of players or computers. SecuRom is another DRM software that is often used for computer games, it attempts to stop users installing the game on multiple computers, even the number of installation is limited to three or five times.

As DRM can be very restricted and complicated, it causes controversies. Many users complain that they are treated as thieves and some of them filed a lawsuit against Electric Art for its use of SecuRom in the video game Spore [4] in 2008; Spore even became the most illegally downloaded games because customers are so frustrated about the fact that the game they paid for can only be activated three times [5]. Moreover, DRM requires an internet connection to activate the software, which does not make much sense if the software itself does not provide any network functions. Take games for example, it is not appropriate to ask players to connect to the internet to play single player games. DRM also suffers from management issues: when Microsoft decided to shut down the MSN music licence server, customers were told that “If you attempt to transfer your songs to additional computers after August 31, 2008, those songs will not successfully play” [6]. Although Microsoft later extended the deadline to 2011, the problem is still there.

Some companies have started ditching DRM to avoid losing customers. For example, Apple announced that after 6 January 2009, iTunes Store would sell all songs DRM-free; BioShock, a computer game that employed SecuRom to counteract privacy has removed the activation limit as a compromise.

Some anti-piracy methods provide hints of the source of pirated copies. Product key, or CD key, would be the most obvious example of such method. It is frequently used to authenticate software, if keys are associated with users, the source of illegal copies can be easily identified by retrieving the product key from them. However, product key is

slightly out of fashion now, since almost all keys can be faked by using key generators (keygen); the need of product key can even be removed by applying certain crack programmes.

An example of a more sophisticated solution is NexGuard Forensic Marking. Its creator, Thomson, claims that it will "enable full traceability of content using invisible and inaudible marks which can be recovered from audiovisual data, even if the content is highly modified" [7]. In a way, NexGuard Forensic Marking is a robust digital watermarking scheme.

The digital fingerprinting algorithm presented in this report is similar to NexGuard Forensic Marking in several aspects. First of all, the fingerprinting algorithm provides tracing ability; secondly, the fingerprinted data is able to survive some kinds of attacks. Unlike NexGuard Forensic Marking, or any other digital watermarking schemes, however, the fingerprinting algorithm emphasizes on tracing and especially anti-collusion tracing, it does not care about the robustness or fidelity. The difference will be explained in details in later chapters.

Fingerprinting Techniques: At a Glance

It is said that the fingerprinting technique was used in logarithm table publishing several hundred years ago [8]. The idea is to use the insignificant bits to hide the unique identification in each copy.

x	log(x)
1	0.000000000000
2	0.301029995663
3	0.477121254719

(a)

x	log(x)
1	0.000000000000
2	0.301029995664
3	0.477121254719

(b)

x	log(x)
1	0.000000000000
2	0.301029995663
3	0.477121254718

(c)

Table.1 logarithm table and its fingerprinted versions

For example, table 1(a) is the original logarithm table. To publish a fingerprinted table 1(b), the 12^{th} digit after the decimal of log(2) is changed to 4; similarly, the last digit of log(3) is changed to 8 in table 1(c). By introducing these tiny differences, each copy of the logarithm table is unique; if the publisher keeps a record of where these differences take place and who each copy is sold to, the source of any illegal copies will be traceable by investigating the difference between the copy and the original.

Although no one is interested in protecting the logarithm table using fingerprinting nowadays, this simple example does include key elements in modern fingerprinting systems. These elements will be referred to throughout the report and are defined as the

following: the **object** is a piece of work that needs to be fingerprinted. In above example, the logarithm table is the object. An object can be physical and digital, but in this report, only digital objects are going to be discussed. The **distributor**, who is normally the owner or the publisher of the object, has the right to exclusively supply fingerprinted objects to **users**, who are authorised to use objects. Each user is given a unique copy of the object by the distributor, if several users compare their objects, they will be able to locate some parts of the objects that are different in each copy. The difference is caused by the **fingerprint**, which is a collection of codes that are embedded into the objects to identify their users. If a small group of users try to manipulate the difference they find in order to publish their own copy of the object, they become **pirates**, the way they manipulate the fingerprinted objects can be called **collusion attack**.

By using the fingerprint, distributors may not be able to stop pirates making illegal copies but they hope that they could identify at least one of the pirates who is involved with piracy when an illegal copy is found; or in better circumstance, the pirates could realise the risk of making illegal copies of fingerprinted objects and give themselves up. Pirates, however, try as hard as they can to avoid being caught; their options include removing the fingerprint or changing the fingerprint so that no one can be identified, or some innocent user is framed.

Earlier Results

N. Wagner is considered to be the first person who introduced the concept of digital fingerprinting. He described a perfect fingerprinting system in [9]; the key character in such system is that any modification to fingerprint will make the object unusable, hence, the distributor can always identify the pirate from any illegal copy. Obviously, the perfect fingerprinting system does not exist; pirates can always modify the fingerprint without damaging the object too much. Even so, it is still very risky to publish illegal copies if the pirate works along, as the fingerprint will allow the distributor to identify the pirate. However, the fingerprint can be discovered when several pirates work together. The fingerprinting scheme that resists against pirate coalition was first brought out in [18], two years after Wagner's work. The discussion at that time was focusing on "long forgiving messages", which is a kind of message that is able to remain understandable after 0.1% of it is altered, such as images. In recent years, fingerprinting systems become increasingly sophisticated. Chor, Fiat and Naor's scheme was one of the most studied schemes in the 90s, the concept of identifiable parent property (IPP) was best known from their work; they used pay-per-view television broadcast and CD-ROMs as examples and assumed that pirates "must output a pirated copy such that for any i the i^{th} position of the pirated copy is identical to the i^{th} position of a legitimate copy the pirates have access to" [10], although some researchers believe such assumption is too restrictive. Compare

with such assumption, it is realistic to assume that by comparing their fingerprinted objects, they do not alter the objects on positions where all of the copies they see agree. This is called “Marking Assumption” in [8] or “Marking Condition” in [10]. Many collusion-secure fingerprinting systems rely on this assumption as they need to reserve some information to reveal the identity of the pirates.

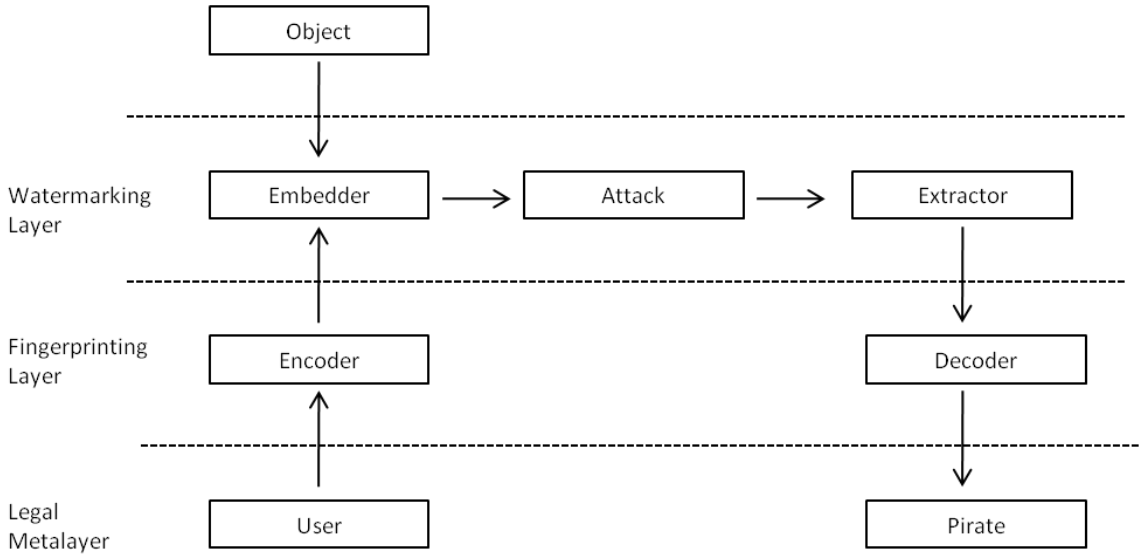

Figure.1 Layered Watermarking/Fingerprinting Model [11]

In addition to the mark assumption, fingerprinting systems are discussed in a layered model, as shown in Figure 1, in [11]. All fingerprinting algorithms presented in this report are assumed to be designed for the fingerprinting layer; in other words, this report will investigate how digital fingerprints are generated and how pirates can be identified using retrieved fingerprinting codes, it does not consider how fingerprinting codes are embedded in to the objects and are extracted from them. Attacks on fingerprinting codes, however, will be studied, even though they seem to take place in the watermarking layer.

Leaving the watermarking layer outside the scope of this report is beneficial because fingerprinting and watermarking techniques serve different purposes and have different requirements; for example, the primary task for fingerprinting system is to identify pirates while for watermarking system, the task becomes to robustly embed certain information into the host without causing too much interference; moreover, embedding the fingerprinting codes in images, video, audio or documents may require different watermarking techniques but the fingerprint can be generated using the same algorithm. Constructing the two techniques in one system increases its complexity unnecessarily, although researchers in [20, 21] did so, it is still a good idea to conduct the research

within the fingerprint layer as long as the watermarking layer and fingerprinting layer can communicate seamlessly.

Chapter Summary

Chapter 2 introduces two important fingerprinting schemes before Tardos' scheme was published: Wagner's scheme and Boneh and Shaw's scheme, the chapter also shows how the two schemes are implemented and tested. The purpose of doing so is to gradually familiarise fingerprinting technologies and study how they evolved in the past; chapter 3 exclusively discusses the implementation of Tardos' scheme, as well as some attack strategies on Tardos' fingerprint code; the limitations of the scheme are listed at the end of the chapter; chapter 4 reviews possible methods of improving Tardos' scheme; finally, conclusions are make in the last chapter.

2. Other Fingerprinting Schemes

Wagner's Statistical Fingerprinting

Wagner's perfect fingerprinting scheme has been proven to be unrealistic; however, his statistical fingerprinting scheme described in [9] is more practical. According to his definition, the character of such fingerprint is: "given sufficiently many misused objects to examine, the distributor can gain any desired degree of confidence that he has correctly identified the compromised user". The author specially mentioned that the pirate identified by the system is never certain because the scheme relies on statistics, there is no absolutely right or wrong accusation; interestingly, modern fingerprinting systems cannot guarantee 100% correctness either.

Fingerprinting Data Values

Wagner's statistical fingerprinting scheme is a "data-oriented" scheme, the fingerprinting system has to know the number of users and what kind of data that is needed to be fingerprinted in advance. Assume that there are *m* users and *n* "real data values" V_1, V_2, V_3, ... V_n. Some data values V_j will be associated with a "delta value" D_j ($D_j > 0$), so that different users may get a different version of V_j. D_j and V_j are associated by the following rules:

For about 50% of the data values, users are provided either V_j or $V_j + D_j$; for the remaining cases, users are given either V_j or $V_j - D_j$. All choices are made pseudo-randomly. Obviously, users will need to accept data values in the range of $[V_j + D_j, V_j - D_j]$, for example, if the data values are pixel values of a greyscale image, users will need to make sure that $[V_j + D_j, V_j - D_j]$ are in the range of 0-255. Some extra procedures may be needed to achieve this requirement but they will not be discussed here. For pirates, they either get the original V_j, or its modified version in a fingerprinted object but they will not be able to distinguish them; the accusation part of the fingerprinting system cannot tell them apart either, so it has to deal with both versions, be it original or modified. Hence, for user *i*, the version of the j^{th} data value sent to him is uniformly denoted V_{ij}.

It is worth pointing out that V_{ij} is not a fingerprint code; Wagner didn't define a code matrix in his fingerprinting scheme, the concept of alphabet didn't appear either. These elements do exist in the scheme; the only problem is that they are closely related to the data values, it is hard to study them separately.

```
for (int i = 0; i < dataSize; i++) {
   for (int j = 0; j < numberOfUsers; j++) {
      fingerprintedData[j][i] = data[i];
      if (r.nextDouble() < 0.25)
         fingerprintedData[j][i] += D;
      else if (r.nextDouble() > 0.75))
         fingerprintedData[j][i] -= D;
   }
}
```

Code Fragment.1: Wagner's Code, Fingerprinting Data Values

Part of the fingerprint generation process is demonstrated by code fragment 1. It employs two *for* loops to generate fingerprinted data values; the outer loop goes through every data value and the inner one produces the fingerprinted data value and assigns it to each user. Variable *r* is an instance of Java class Random, whose seed is provided by the distributor. It is used to simulate the rules for delta value association, which states that there are about 25% of the chances that a user gets $V_j + D_j$ and the chance of getting $V_j - D_j$ is also 25%. As *r.nextDouble()* will return a uniformly distributed random real number between 0 and 1, the possibility of getting a number that is less than 0.25 is 25%; similarly, the change of getting a number that is larger than 0.75 is 25%, which perfectly matches the requirement. The fingerprinted data values are stored in a two-dimensional array named *fingerprintedData*; the original data values are also reserved for accusation.

Accusation

Assume that the distributor retrieves an illegal copy and wants to find out the source by using returned data values V'_1, V'_2, V'_3, ... V'_n. In order to identify the pirate, the distributor needs to test all users one after another. For each user, the first thing to do is to calculate L_{ij}, which is defined as:

$$L_{ij} = (1 / D) (V'_j - V_{ij}),\ 1 \leq j \leq n,\ fixed\ i$$

L_{ij} are the normalised differences between the returned data values and the one that are assigned to user *i*. Next, the distributor calculates means from two sets of L_{ij}. The first set, MH_i, is equal to the mean of the L_{ij} such that $V_{ij} = V_j + D_j$; the other set, ML_i, is equal to the mean of the L_{ij} such that $V_{ij} = V_j - D_j$. This means that the distributor will have to remember how fingerprinted data values are given to user *i* (and to any other user), so that they can be compared with the original ones and therefore, MH_i and ML_i can be calculated. Apparently, MH_i and ML_i are disjoint subsets of L_{ij} and the distributor uses them to make a final calculation:

$$M_i = ML_i - MH_i$$

Suppose user *i* is the pirate and suppose he does not try to change the fingerprint codes, the distributor will find that the returned data values match perfectly with the data values that are given to user *i*, which means:

$$M_i = MH_i = ML_i = 0.0$$

If, on the other hand, user *i* is not the pirate, the distributor will expect to have:

$$MH_i \approx -0.5$$
$$ML_i \approx 0.5$$
$$M_i = ML_i - MH_i \approx 1.0$$

In reality, if a user decides to distribute his copy illegally and is aware the existence of the fingerprinting codes, which is normally the case, it is rational to assume that he will try to alter the fingerprinted data to avoid being caught. Another assumption that the distributor makes here is the pirate does not know whether data values (fingerprinted data values) V_{ij} are the original data values V_j or $V_j \pm D_j$. Hence, if there are enough data, the distributor can still expect pirate's M_i to be very close to 0 and innocent users' M_i to be close to 1.

```
for (int i = 0; i < numberOfUsers; i++) {
   for (int j = 0; j < dataSize; j++) {
      Vij = fingerprintedData[i][j];
      L = (1 / delta) * (Vp[j] - Vij);
      if (Vij > data[j]) {
         mh += L;
        mhcntr++;
      }
      else if (Vij < data[j]) {
         ml += L;
         mlcntr++;
      }
   }

   if (mlcntr != 0)
      mli = ml / (double)mlcntr;
   if (mhcntr != 0)
      mhi = mh / (double)mhcntr;
   mean[i] = mli - mhi;
   ....

}
```

Code Fragment.2: Wager's Code, Accusation

Code fragment 2 shows how accusation algorithm is implemented; for this part of the system, an array *Vp* is used to store returned data values, its elements, together with the ones from array *Vij* are used to calculate L. Several variables are used to help calculating the means of two subsets of L, such as *mh* and *mhi*.

If the pirate makes no change to the fingerprint codes, *ml*, *mlcntr*, *mh* and *mhcntr* are all going to be zero; in order to prevent arithmetic exception, two *if* statements are used at the end. Eventually, *Mi* for each user is calculated and the one which is closest to zero is returned.

Attack

Wagner suggested the following random alteration strategy to attack the fingerprinted data:

50 percent of time: round to nearest integer
25 percent of time: truncate to nearest integer
12.5 percent of time: round and add 1
12.5 percent of time: round and subtract 1

There are many other strategies that pirates can use but they should be aware that any alteration strategy is likely to be detected if there are enough users, because the altered data will be statistically standing out; also, introducing to many changes to the fingerprinted data may destroy the data, which makes the attack strategy pointless.

Tests and Evaluation

Wagner's attack strategy is implemented and is tested with rest of the fingerprinting system. For a simple test, 200 real numbers are randomly generated in the range from 10.0 to 100.0 and are used as data values; the delta value is set to be 0.5, in addition, assume that there are 25 users in the system, they are given users ID 0, 1, 2, ..., 24.

After the fingerprinted data values are generated and are given to each user, one of the users is randomly selected to be the pirate. His fingerprinted data values will be processed according to the attack strategy and the modified version will be sent back to test whether the accusation algorithm is able to correctly reveal his user ID.

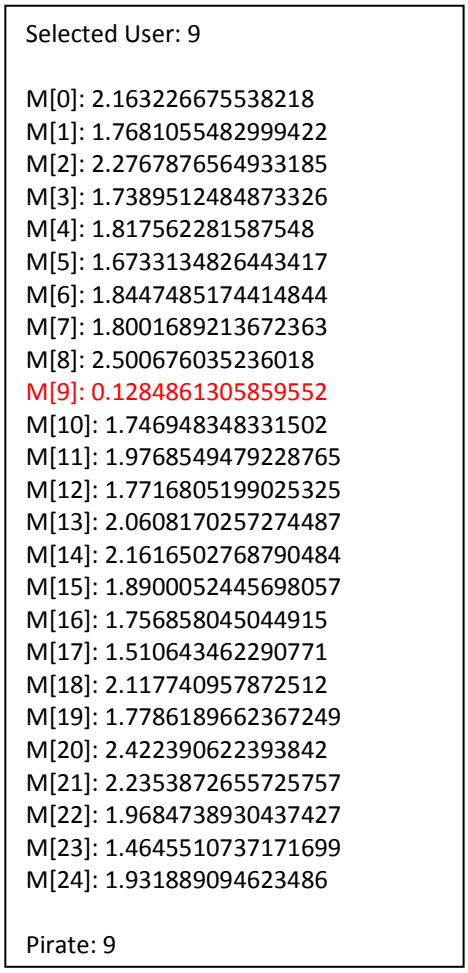

Figure.2: Accusation Result

Figure 2 shows the sample output of the accusation algorithm. Among all the 25 users, user 9 clearly stands out as he has the lowest M value and he is marked as a pirate by the accusation algorithm, which correctly reflexes the fact that he is the randomly selected pirate. The accusation result also indicates that the attack strategy fails to interfere with the accusation algorithm; even though the fingerprinted data is modified, the pirate is still identifiable.

Next, the above test is repeated 500 times and another 500 times with some parameters changed; for the repeated tests, any users that are wrongly accused will be counted as false positive. The results of the experiments are listed below:

Range of Data Values	Number of Data Values	Number of Users	Delta Value	False Positive
10.0 – 100.0	200	25	0.5	0
-100.0 – 100.0	400	500	1.0	0

Table.2 Experiments Result for Wagner's Fingerprinting System

The result indicates that the fingerprinting system remains robust under Wagner's simple attack, no innocent users are accused. The system seems to be able to support large volume of data values and users; in the second experiment, the range of the data value is obviously wider than Wagner's example but it does not seem to cause any trouble. The time spent on fingerprinting data and accusing pirates is fairly short as there is no complicated calculation in any part of the system. In addition, adding new users or new data values to the system were not discussed in original paper, but these operations are presumably easy to implement.

As the very first digital fingerprinting scheme, it is not expected to be perfect. A significant drawback of the scheme is that it cannot defeat collusion attack. A coalition of pirates can easily find out whether a data value is original or modified by comparing their fingerprinted copies; as a result, pirates will be able to publish a fingerprint-free copy or frame an innocent user, which are absolutely unacceptable to the distributor.

Strictly speaking, Wagner's scheme cannot fit into the layered model mentioned in the previous chapter because the scheme operates on the actual data directly. Such setting could be a disadvantage as it makes the scheme easy to fingerprint certain types of data/files such as images, audio and video but not the types of data/files that are sensitive to changes, such as documents or applications; therefore, it would be very difficult to make the scheme universal.

Boneh and Shaw's Collusion-Secure Fingerprinting

Boneh and shaw are not the pioneering researchers who try to solve the problem caused by collusion attack but their contribution is considered to be very significant. Their basic fingerprint code has the length of $O(n^3 log(n/\varepsilon))$ with an ε error rate (ε-secure) for any collusion size.

Code Generation

There are a couple of definitions and symbols that are needed to be introduced first, they are used in [8] and will be used in this section. Let Σ denote the alphabet; as the fingerprint code is binary in Boneh and Shaw's scheme, $\Sigma = \{0, 1\}$. A set $\Gamma = \{w^{(1)}, \ldots, w^{(n)}\} \subseteq \Sigma^l$ will be called an *(l, n)-code*, the codeword $w^{(i)}$ will be assigned to user u_i, for $1 \leq i \leq n$. A fingerprinting scheme Γ_r is *c-secure* with *ε-error* if there exists a tracing algorithm A satisfying the following condition: if a coalition C of at most c users generates a word x then $Pr[A(x) \in C] > 1 - \varepsilon$, where the probability is taken over the random bits r and the random choices made by the coalition.

To initiate the fingerprint code generation, code generator will need to know the number of users n and the error rate ε, eventually, the generator will generate a code matrix, which has n rows and many columns. The number of columns is actually the length of the fingerprint code, the following quotation shows how Boneh and Shaw determine the length of code: "let c_m be a column of height n in which the first m bits are 1 and the rest are 0. The code $\Gamma_0 (n, d)$ consists of all columns $c_1, \ldots, c_{n-1}$, each duplicated d times. The amount of duplication determines the error probability ε". For example, the code $\Gamma_0 (4, 3)$ for 4 users u_1, u_2, u_3 and u_4 literally looks like this:

u_1	111111111
u_2	000111111
u_3	000000111
u_4	000000000

Figure.3: A Boneh and Shaw's $\Gamma_0 (4, 3)$ Code

At this stage, the distributor needs to solve two problems. First of all, how is the code generator able to decide the value of d? Boneh and Shaw shows that for $n \geq 3$ and $\varepsilon > 0$, $d = 2n^2 log (2n / \varepsilon)$ and hence the length of the code is $d (n - 1)$. The other problem is that the fingerprint code looks too neat and predictable. The solution to this problem is to introduce a permutation algorithm so that pirates will not know "which mark in the object encodes which bit in the code", in other words, the pirates will not be able to tell the fingerprint code they detect belongs to which column in the code matrix. The purpose of doing so is to make sure that the fingerprint code is truly *n-secure* with *ε-error*; the permutation must be hidden from the users and the same permutation must be applied to all users.

Code fragment 3 shows how the code matrix is constructed before permutation. Variable *duplication*, which is equivalent to d mentioned above, is first calculated using Boneh and Shaw's formula, the size of the code matrix can then be determined. Note that the

main reason of using boolean instead of integer as the data type of the code matrix is to reduce memory usage, relevant discussions will be presented in the next chapter. Next, two *for* loops are used to fill the code matrix with "true" values, which are equivalent to ones. Unlike what Boneh and Shaw stated, the value assignment is done one user after another instead of one column after another, but the outcomes are the same.

```
duplication = (int)(2 * Math.pow(numberOfUsers, 2.0)
   * Math.log(2 * n / epsilon));
lengthOfCodes = (numberOfUsers - 1) * duplication;
codeMatrix = new boolean[numberOfUsers][lengthOfCodes];
int startPnt = 0;

for (int i = 0; i < numberOfUsers; i++) {
   for (int j = startPnt; j < lengthOfCodes; j++) {
      codeMatrix[i][j] = true;
   }
   startPnt += duplication;
}
```

Code Fragment.3: Constructing Boneh and Shaw's Code

```
pidx = new int[lengthOfCodes];
for (int i = 0; i < lengthOfCodes; i++) {
   pidx[i] = i;
}
permutate(pidx);

boolean[] initCode = null;
for (int i = 0; i < numberOfUsers; i++) {
   initCode = codeMatrix[i].clone();
   for (int j = 0; j < lengthOfCodes; j++) {
      codeMatrix[i][j] = initCode[pidx[j]];
   }
}
```

Code Fragment.4: Boneh and Shaw's Code Permutation

The next step is to apply permutation to the code matrix. As mentioned earlier, the purpose of permutation is to shuffle the fingerprint code a little bit; but instead of shuffling the actual fingerprint code, the code generator shuffles the indexes of columns of the code matrix so that the permutation can be easily applied to all users.

The details of the shuffle algorithm are shown in code fragment 5. It takes an array as input and randomly swaps the position of each element in the array with another, when it reaches the end, all elements in the array will be in different positions. The code is the modified version of [22]; the original source code is written in C, it is translated into Java with little details changed.

```
private void permutate (int[] arr) {
   int temp, randmax = Integer.MAX_VALUE, arrlength = arr.length;
   Random r = new Random();
   for (int i = 0, j = 0; i < arrlength; i++) {
      j = r.nextInt(randmax) * arrlength / (randmax + 1);
      temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
   }
}
```

Code Fragment.5: Permutation

Accusation

The accusation algorithm takes a pirate version, which is produced by a coalition of several users, and output a member of the coalition with an error probability of ε. In order to make a proper accusation, the permutation has to be inversed in the pirate copy. Some new definitions and symbols are needed to explain the algorithm.

Similar to the notion of c_m in code generation, b_m is used in accusation to denote "the set of all bit positions in which the first *m* users see a 1 and the rest see a 0". Apparently, the length of each b_m is equal to *d*. Secondly, for $2 \leq s \leq n-1$, define $R_s = B_{s-1} \cup B_s$. Lastly, for a given piece of fingerprint code *x*, let *weight(x)* equal to the number of 1 in *x*. To find a subset of the coalition that produced *x*, the algorithm follows rules listed below:

1. If $weight(x \mid b_1) > 0$, then accuse user 1;
2. If $weight(x \mid b_{n-1}) < d$, then accuse user *n*;
3. For all $2 \leqslant s \leqslant n - 1$, let $k = weight(x \mid r_s)$, if

$$weight(x \mid b_{s-1}) < \frac{k}{2} - \sqrt{\frac{k}{2} log \frac{2n}{\varepsilon}}$$

then accuse user *s*.

```
   boolean[] b = new boolean[duplication];
   for (int i = 0; i < duplication; i++) {
     b [i] = ippCopy[i];
   }
   if (getWeight(b) > 0)
     id[idcntr++] = 0;

   for (int i = lengthOfCodes - duplication, j = 0; i < lengthOfCodes; i++, j++) {
     b [j] = ippCopy[i];
   }
   if (getWeight(bitchunk) < duplication)
     id[idcntr++] = numberOfUsers - 1;
```

Code Fragment.6: Boneh and Shaw's Accusation Algorithm, Part 1

Boneh and Shaw's accusation algorithm can be treated as a point-based system; each part of the pirate copy will gain a score and the score will be compared to a threshold. However, the accusation cannot be done in a single iteration; as the rules state, the first user and the last user must be accused separately using different thresholds. Code fragment 6 shows the implementation of accusation rule no.1 and 2, while the third rule is implemented in code fragment 7.

```
boolean[] rs = new boolean[duplication * 2];
for (int i = 1; i < numberOfUsers - 1; i++) {
   for (int j = 0; j < duplication; j++) {
      b[j] = rs[j] = ippCopy[(i - 1) * duplication + j];
      rs[j + duplication] = ippCopy[i * duplication + j];
   }
   int k = getWeight(rs);

   if (getWeight(b) < (k / 2 - Math.sqrt(k / 2 * Math.log(2 *
      numberOfUsers / epsilon)))) {
      id[idcntr++] = i;
      break;
   }
}
```

Code Fragment.7: Boneh and Shaw's Accusation Algorithm, Part 2

Even though Boneh and Shaw's accusation algorithm is mostly designed to identify one pirate, its implementation still prepares for the situation where 2 or more pirates are identified. The ID of all accused users are stored in an array so that there will be enough space.

The codes for inverse permutation and the weight calculation are not shown since they are fairly simple. It is worth mentioning that using boolean values to represent fingerprint codes very slightly increase the complexity of weight calculation. In this case, an extra *if* statement is need pick out all the ones.

Attack

Two attack strategies are implemented to simulate the collusion attack, as shown in code fragment 8. Under the marking assumption, the first strategy uses random bits to fill the code, while the second one uses the majority value in the collusion to make up the pirate version. Majority attack is used in [8] to prove the correctness of theorem 4.2, which states: "for $c \geq 2$ and $n \geq 3$ there are no totally *c-secure (l, n)-codes*". However, since the fingerprint codes are shuffled in Boneh and Shaw's final proposal, such attack shouldn't cause much interference.

```
for (int i = 0; i < lengthOfCode; i++) {
   boolean codesAgree = true;
   for (int j = 0; j < collusionSize - 1; j++) {
      if (codeMatrix[pirateId[j]][i] != codeMatrix[pirateId[j + 1]][i]) {
         codesAgree = false;
         break;
      }
   }

   if (codesAgree) {
      pirateCopy[i] = codeMatrix[pirateId[0]][i];
   }
   else {
      switch(strategy) {
         case 0: pirateCopy[i] = rand.nextBoolean();    break;
         case 1: pirateCopy[i] = getMajority(i, codeMatrix, pirateId);    break;
      }
   }
}
```

Code Fragment 8: Attack Strategies for Boneh and Shaw's Scheme

Tests and Evaluation

Two sets of simple tests are conducted to test the Boneh and Shaw's fingerprint codes. The first set uses random choice attack to make a pirate copy; the accusation algorithm will then try to identify as many pirates as possible by examine the pirate copy; accusing innocent users will be counted as a false positive error. This procedure will be repeated 500 times and the test results are listed in table 3:

Number of Users	Size of Collusion	Error Rate	Length of Fingerprint Code	Average Number of Identified Pirates	Average False Positive
40	5	1%	1121601	1.21	0
40	35	1%	1121601	1.85	0

Table.3: Results for the First Set of Tests

The second set of tests is very similar to the first one, except the attack strategy is changed to majority choice attack and the error rate is increased to 5%. In one case, the number of users is reduced to 30 to show the relationship between the number of users and the length of fingerprint code. Each test is repeated 500 times.

Number of Users	Size of Collusion	Error Rate	Length of Fingerprint Code	Average Number of Identified Pirates	Average False Positive
30	5	5%	370098	1	0
40	35	5%	920712	1	0

Table.4: Results for the Second Set of Tests

Tests confirm that the number of users and the error rate are the factors that affect the length of code; the accusation algorithm works well in all tests, the error rate in the second test is set to 5%, which is a lot higher than practical usages and yet, no false positive occurs; also, the tests prove that majority choice attack does not defeat Boneh and Shaw's code.

The accusation algorithm is able to correctly identify one or two of the pirates even when the majority of the users are involved with piracy. On the other hand, the accusation algorithm is never able to identify all of the pirates, even if there are only 5 pirates in the coalition. In the second set of experiment, the accusation algorithm always accuses one pirate; the reason is that the pirate copy produced by majority attack exactly matches one of fingerprinted copies. Take the code matrix in figure 3 for example, assume that user 1, 2 and 4 form a coalition, by using the majority attack, a pirate copy should be 000111111, which perfectly matches user 2's copy.

One of the serious issues of Boneh and Shaw's code is the length of the code, $O(n^3 \log n / \varepsilon)$; it is very closely related to the number of users, which makes the code incredibly long when the number of users is large; while it is rational to assume that the number of users is always large. In fact, Java cannot even support more than 50 users because the length of code is outside the scope of integer data type.

Another problem of the scheme is that the distributor must define the number of users before the fingerprint codes are generated. This is a rather impractical requirement because it might be difficult to estimate how many users will eventually be using the fingerprinted data; such restriction also means that the distributor cannot add new users into the fingerprinting system, in this case, the distributor can either generates a new, larger code matrix for its existing users and new users, or constructs a new code matrix for the new users alone.

In addition, a small problem of Boneh and Shaw's accusation algorithm is that there is no universal threshold. In their algorithm, two thresholds are needed for the first user and the last user in the code matrix and a third one for the rest of the users. Such setting makes the implementation of the accusation algorithm long-winded.

3. Tardos' Optimal Probabilistic Fingerprint Codes

Tardos improved Boneh and Shaw's scheme and developed another collusion-secure fingerprinting code. Compare with one of its ancestor, Tardos reduced the length of the code to the square of its length. Another feature of Tardos' code is that the fingerprint codes are randomly generated over a binary alphabet. Tardos stated the formal definition of the fingerprint code in [10]:

A fingerprint code *of length m for n users over the alphabet Σ is a distribution over the pairs (X, σ), where X is an n by m matrix over Σ and σ is an algorithm that takes a string* $y \in \Sigma^m$ *(the pirated copy) as input, and produces a subset* $\sigma(y) \subseteq [n] := \{1, 2, \ldots, n\}$ *(the set of accused users). For* $\emptyset \neq C \subseteq [n]$ *a* C-strategy *is an algorithm ρ that takes the submatrix of X formed by the rows with indices in C as input, and produces a string* $y = \rho(X) \in \Sigma^m$ *as output 1 and satisfies the* marking condition *that, for all positions* $1 \leqslant i \leqslant m$*, if all the values* X_{ji} *for* $j \in C$ *agree with some letter* $s \in \Sigma$ *then* $y_i = s$*. We say that a fingerprint code is ε-secure against coalitions of size c, if for any* $C \subseteq [n]$ *of size* $|C| \leqslant c$ *and for any C-strategy ρ, the error probability*

$$P[\sigma(\rho(X)) = \emptyset \text{ or } \sigma(\rho(X)) \notin C]$$

is at most ε.

Fingerprint Codes Generation

Tardos' fingerprint code is an *n* by *m* binary matrix, where *n* denotes number of users and *m* denotes the length of the fingerprint codes. Unlike Wagner's code, Tardos' code can be generated based on some parameters such as *n* and *epsilon* instead of the actual data values; moreover, similar Boneh and Shaw's code, Tardos' code is binary (the alphabet $\Sigma = \{0, 1\}$), which is relatively easy to be embedded into objects.

The key parameters that decide the size of the fingerprint code matrix are the number of user *n*, the size of collusion *c* and the error bound *epsilon*. It is the distributor's responsibility to provide these parameters to the fingerprinting system. Tardos' design allows the distributor to add new users dynamically, so the number of users does not matter that much but the size of collusion and the error rate will have to be correctly estimated in advance.

Code fragment 9 shows how *n*, *c* and *epsilon* are used to initialise some intermediate variables for fingerprint codes generation. Note that the number of users and the size of collusion have to be positive integers, $1 < epsilon < 0$ and *log* always denotes natural logarithm.

```
k = Math.log(1.0 / epsilon);
lengthOfCodes = (int)(100.0 * Math.pow((double)collusionSize, 2.0) * k);
codeMatrix = new boolean[numberOfUsers][lengthOfCodes];

double t = 1 / (300 * collusionSize);
double tp = Math.asin(Math.sqrt(t));
double rmin = (float)tp;
double rmax = (float)(Math.PI / 2.0 - tp);
p = new float[lengthOfCodes];
```

Code Fragment.9: Tardos' Code, Variables Initialisation

First of all, let $k = log\ (1/epsilon)$; it is declared as a global variable since it will be used for accusation by another function; the length of the fingerprint codes m can be calculated using $m = 100c^2k$ and then an empty two dimensional boolean matrix is created to store fingerprint codes. An obvious choice of storing fingerprint codes is to use an integer matrix, but it is not used here for two reasons. First of all, the alphabet of the fingerprint code is binary, meaning that a fingerprint code is either 0 or 1; the two states of a boolean variable, true or false, are perfectly capable of representing the values of the fingerprint code; in the programme, false is equivalent to 0 and true is equivalent to 1. Another reason of using boolean is that it lowers the memory requirement of generating fingerprint codes. In Java, "int" data type is always 32-bit signed two's complement integer, although it is unclear how many bits of memory are used for boolean exactly, less memory is needed for sure.

In the fingerprint code matrix, each element is set to 0 or 1 (false or true) according to its probability p, $0 < p < 1$; for fingerprint codes of length m, there will be m probability values, each one of them should be an independent, identically distributed random variable from t to $1 - t$, where $t = 1/300c$. To calculate each p, first let $t = sin^2t'$ to get t' ($0 < t' < \pi/4$), next, randomly generate an r value in the range from t' to $\pi/2 - t'$ and finally, the probability value p can be calculated using $p = sin^2r$. The distributor should reserve the probability array for accusation.

In code fragment 9, variable t and tp are equivalent to t and t' that are defined in the previous paragraph. Variable *rmin* and *rmax* define the range of r, i.e. $rmin = t'$ and $rmax = \pi/2 - t'$. Probability values are stored in a "float" array; "double" data type is certainly an option here but it is not used to save memory space.

Having the range of r value defined, the probability array and the fingerprint code can be therefore generated. The first *for* loop in code fragment 10 shows the construction of the probability array, an r value is randomly generated so that p can be calculated using $p = sin^2r$. The next *for* loop constructs the fingerprint code matrix according to the probability array.

```
double r = 0.0;
for (int i = 0; i < lengthOfCodes; i++) {
   r = rmin + Math.random() * (rmax - rmin);
   p[i] = (float)Math.pow(Math.sin(r), 2.0);
}

for (int i = 0; i < numberOfUsers; i++) {
   codeMatrix[i] = getFingerprintCodes(p);
}
```

Code Fragment.10: Tardos' Code: *p* and Fingerprint Code Generation

Note that an auxiliary function *getFingerprintCodes(float[] p)* is called to generate the fingerprint code for every user. The details of this function are shown in code fragment 11.

```
private boolean[] getFingerprintCodes(float[] p) {
   boolean[] fp = new boolean[p.length];
   for (int i = 0; i < p.length; i++) {
     fp[i] = Math.random()  < p[i];
   }
   return fp;
}
```

Code Fragment.11: Tardos' Code, Fingerprint Code Selection

This function returns a piece of fingerprint codes according to the *p* values. The rule for using *p* values is: the more *p* value closes to 1, the more likely the fingerprint code is selected to be 1 (or more likely the fingerprint code is selected to be "true", as the fingerprint codes are stored in a boolean matrix). The fingerprint code selection rule is implemented by generating a random number and comparing it with a *p* value; the results of the comparison are directly recorded by the boolean array.

The reason of separating code fragment 11 from code fragment 10 is to increase code readability and reusability, such design also allows new users to be added easily. More details about adding new users can be found in the next section.

Adding New Users

As mentioned previously, the distributor does not have to know the number of users in advance; new users can be added to the system dynamically. Such feature is possible because Tardos' fingerprinting scheme relies on the size of collusion and an error probability to determine the size of the fingerprint codes. The following code fragment shows details of how new users are added.

```
public boolean[][] getFingerprintMatrix(int numberOfExtraUsers) {
   boolean[][] tempCodeMatrix = codeMatrix;
   codeMatrix = new boolean[numberOfUsers + numberOfExtraUsers][lengthOfCodes];
   for (int i = 0; i < numberOfUsers; i++) {
      codeMatrix[i] = tempCodeMatrix[i];
   }

   for (int i = numberOfUsers; i < numberOfExtraUsers + numberOfUsers; i++) {
      codeMatrix[i] = getFingerprintCodes(p);
   }
   numberOfUsers += numberOfExtraUsers;
   return codeMatrix;
}
```

Code Fragment.12: Tardos' Code, Adding New Users

This function takes the number of new users as a parameter and returns a new, expanded fingerprint code matrix. The task of producing the new matrix is split into two parts. The first part creates an empty fingerprint code matrix that is large enough for the old users and new users, then a *for* loop is used to copy the data from the old matrix to the new one.

The second part is functionally identical to the codes that are used for fingerprint code generation; the function *getFingerprintCodes(float[] p)* is called here again to generate extra fingerprint codes for newly added users. At the end, the number of users is updated and the new fingerprint code matrix is returned.

Accusation

The accusation algorithm largely depends on *p* values (the probability array) and the fingerprint code matrix to make accusation. Assume that the number of users is *n* and the length of the fingerprint code is *m*, Tardos defined the *n* by *m* matrix *U* and its entries as follow (p_i is an element in the probability array and X_{ji} is an element in the fingerprint code matrix):

$$U_{ji} = \begin{cases} \sqrt{\dfrac{1-p_i}{p_i}} \; if \; X_{ji} = 1 \\ -\sqrt{\dfrac{p_i}{1-p_i}} \; if \; X_{ji} = 0 \end{cases}$$

Let the algorithm accuse user *j* on the pirate copy $y = \{0, 1\}^m$ as input if:

$$\sum_{i=1}^{m} y_i \, U_{ji} > Z$$

where $Z = 20ck$ as a threshold parameter.

Tardos claimed that the accusation algorithm has two advantages. First of all, the construction of the algorithm is simple as it is a score-based system, if the score for a user is higher than the threshold Z, the user can be accused as a pirate; moreover, the score calculation is fairly straightforward. Another advantage is that the probability of accusing an innocent user is lower than the error bound and if the accusation algorithm is not sure about certain user, it will not accuse anyone.

Overall, the whole idea of the accusation algorithm is to construct a *U* matrix whose entries are calculated based on the fingerprint codes and the probability matrix; the fingerprint code is extracted from a pirate copy and each bit of the code is multiplied with the corresponding element in *U*, the sum of the products is a score for this user and it is then compared with the threshold *Z*, and if the score is greater than *Z*, the user will be accused. It is worth mentioning that in Tardos' definition, $Z = 20ck$, *c* is the size of collusion and *k* is the global variable that has been initialised in the fingerprint generation part. It means that the value of *Z* is rigid, the distributor will need to generate a new fingerprint code matrix for any change made to *Z*.

```
for (int i = 0; i < numberOfUsers; i++) {
   for (int j = 0; j < lengthOfCodes; j++) {
      if (codeMatrix[i][j] && piratedCopy[j]) {
         sum += Math.sqrt((1 - p[j]) / p[j]);
      }
      else if (!codeMatrix[i][j] && piratedCopy[j]){
         sum += -Math.sqrt(p[j] / (1 - p[j]));
      }
   }

   if (sum > Z) {
      id[cntr++] = i;
   }
   sum = 0.0;
}
```

Code Fragment.13: Tardos' Code, Pirate Accusation

Code fragment 13 shows how Tardos definition is implemented. Because a boolean matrix is used to store the fingerprint codes, the implementation is slightly different from Tardos' definition. Suppose the programme is calculating a score for user *j* on a pirate copy *y*, during the calculation, the score is stored in variable *sum*. It is the mathematical law that any number multiplies by zero is zero, and adding zero to a number does not change the value of this number; hence, the programme only calculates $y_i U_{ji}$ and adds the product to *sum* if y_i is 1; also, because *y* is actually boolean data type and "true" is equivalent to 1 and "false" is equivalent to 0, what the programme eventually does is to only calculate $y_i U_{ji}$ and add the product to *sum* if y_i is true.

The *if-else* statements in the inner loop of code fragment 13 have two conditions, the second one, *piratedCopy[j]*, examines whether or not y_i is equal to 1, as discussed in the paragraph above. The first one determines which of U_{ji} values is added to *sum*. Each U_{ji} is added to *sum*, but its value is never reserved because the value is not used anywhere else.

The algorithm goes through every user to search for the pirate; for each user, the programme calculates a score and it is compared with Z; if the sum is greater than Z, this user will be identified as pirate and his ID (his index in the user group) will be stored in an array. It is pretty obvious that the accusation algorithm is able to accuse more than one user; all users who are involved with producing the pirate copy will all likely be accused. If the size of the collusion is larger than the programme can handle, the accusation algorithm should accuse no one or identify one of the pirates.

Attack

When pirates try to manipulate the fingerprint codes to avoid being caught, it is assumed that they are under the marking assumption, which means they can only modify fingerprint codes that they see disagree.

As discussed in the previous chapter, if several pirates work together to find all differences between their copies, they may come up with some attack strategies to replace the code they see different to make a pirate copy: for example, they can randomly choose 0 or 1 to make up the code; or they can randomly use each other's code to form a hybrid version; another method could be to exclusive-or their codes.

TardosCodesAttacker.java implements the four types of attack strategies based on examples stated above. The code is able to simulate any number of pirates work together. A *for* loop is used at the very beginning to determine whether or not the fingerprint codes from all pirates agree; if not, the boolean variable *codesAgree* will be set to false and break the loop. The next *if* statement will examine the status of *codesAgree*, if it is true, the pirated copy will follow the marking assumption and makes no change to the original code; otherwise, the programme moves on to execute the attack strategies. Each *case* in the *switch* block implements an attack strategy. The first two are straightforward, case 0 is the implementation of "random choice" attack and case 1 is the implementation of making a pirate copy using each pirate's codes – the "mix and match" attack; both of case 2 and case 3 use exclusive-or but they do opposite things. The former one is the so-called minority choice, which makes the pirate copy using the fingerprint code that appears less in all pirates. For example, let 5 pirates form the collusion and at certain position their fingerprint codes are: 0, 0, 1, 0, 0. Since 1 is the minority code, the pirate copy will use 1; in case 3, which is the implementation of majority choice, 0 will be used to make the pirate copy. Case 2 and 3 are restricted to work when the number of pirates is odd, because there may not be a minority or majority when it is even (for instance, 0, 0, 1, 1). Note that in Java, the exclusive-or operation on boolean values is valid; if two boolean values are the same, the xor result will be true; otherwise the result will be false.

```
for (int j = 0; j < collusionSize - 1; j++) {
   if (codeMatrix[pirateId[j]][i] != codeMatrix[pirateId[j + 1]][i]) {
      codesAgree = false;
      break;
   }
}
if (codesAgree)
   pirateCopy[i] = codeMatrix[pirateId[0]][i];
else {
   switch(strategy) {
      case 0: pirateCopy[i] = rand.nextBoolean();    break;
      case 1:
         double mk = 1.0 / (double)collusionSize;
         int zone = (int)(rand.nextDouble() / mk);
         pirateCopy[i] = codeMatrix[pirateId[zone]][i];    break;
      case 2:
         boolean xor = codeMatrix[pirateId[0]][i];
         for (int j = 1; j < collusionSize; j++)
            xor = xor ^ codeMatrix[pirateId[j]][i];
         pirateCopy[i] = xor;    break;
      case 3:
         xor = codeMatrix[pirateId[0]][i];
         for (int j = 1; j < collusionSize; j++)
            xor = xor ^ codeMatrix[pirateId[j]][i];
         pirateCopy[i] = !xor;    break;
     }
}
```

Code Fragment.14: Attacking Strategies for Tardos Code

Experiments and Evaluation

A fingerprint code matrix for 100 users (*n = 100*) is constructed for a simple test, it has an error rate of 1% (*epsilon = 10^{-2}*) and is generated to be secure again 5 pirates (*c = 5*). Three users, user 0, user 17 and user 25, are chosen to be the pirates; they compare their fingerprint codes and construct a pirate version. The pirate version of the fingerprint code does not belong to any legitimate users, it is sent to the accusation algorithm and the result of pirate identification is displayed.

```
Z: 460.51701859880916

Accused User ID: 0
905.3408417242935

Accused User ID: 17
900.5543473827586

Accused User ID: 25
877.5998110034834
```

Figure.4 Accusation Result

Figure 4 shows the output of the accusation algorithm. The first line displays the threshold parameter $Z \approx 460$, those users whose score are exceeding this threshold are accused as pirates. Scores for the selected users are significantly larger than the threshold; hence, the algorithm correctly detects all pirates.

Eight experiments are conducted to further evaluate the performance of the fingerprinting system and the results are shown in table 5. Experiments 1-4 assume that there are 1000 users and experiment 5-8 assumes that there are 5000. Fingerprint codes in experiment 1, 3, 5 and 7 are generated to be able to identify 5 pirates, in these experiments, 5 users are randomly selected to work together to make a pirate fingerprint code; similarly, the rest of the experiments will generate fingerprint codes that are capable of identifying 10 pirates and will be attacked by 10 randomly selected users. The Java code for pirates selection is shown below, variable *n* in line 4 is the number of users (i.e. 1000 or 5000).

```
int[] pirateId = new int[collusionSize];
Random rand = new Random();
for (int j = 0; j < collusionSize; j++) {
   pirateId[j] = rand.nextInt(n);
}
```

Code Fragment.15: Pirates Selection

The fingerprint code will have 0.1%, 1% or 5% chance of making an error. There are two types of errors that the system makes, namely false negative and false positive. The former mistake occurs when the system fails to accuse all pirates in the collusion while the latter occurs when the system accuses innocent users, which is considered to be a lot worse. It is possible that the two types of mistakes occur at the same time, for this reason, they will be counted separately.

```
int validlength = accusedId.length, searchResult = 0;
for (int j = 0; j < accusedId.length; j++) {
   searchResult = Arrays.binarySearch(pirateId, accusedId[j]);
   if (searchResult < 0) {
      fpflag = true;
      validlength--;
   }
}
if (fpflag)
   fp++;
if (validlength < (pirateId.length - duplicateItem(pirateId)))
   fn++;
```

Code Fragment.16: Counting False Positive and False Negative

After the accused users are returned by the accusation algorithm in TardosCodes.java, false negative and false positive will be counted in the following way, as shown in code fragment 16: each accused user is verified by searching the original array that stores the randomly selected pirates, if the user is not in the array, it means that the accusation algorithm has accused an innocent user and therefore the false positive flag is set to true, and the variable *validlength*, which is initialised to be equal to the number of accused users, is deducted by 1. This process is repeated until all accused users are verified, at the end, *validlength* will be the number of pirates that are correctly accused.

The two types of errors are counted by the two *if* statements. The first one examines the false positive flag, if it is true, the false positive counter will increase by 1; the other one examines whether the variable *validlength* is smaller than the actual number of pirates, if the result is true, it means that certain pirates are not accused, hence false negative occurs. Note that the function *duplicateItem* is called to calculate how many pirate IDs are identical because the random number generator may generate the same number twice.

For all experiments, the pirates use attack strategy 0, as described in the previous section, to produce a pirate copy, meaning that they use random bits to make up the code they see disagree.

Each experiment will be repeated 2000 times. The average time spent on generating the fingerprint code matrix and the average time spent on accusation will be recorded. Memory space occupied for generating fingerprint code is also a criterion. All experiments take place on a PC that is equipped with an Intel Q9550 2.83GHz processor and 4 Gigabytes RAM. The computer runs 64-bit Windows Vista. Memory usage is gathered from Windows task manager.

In table 5, T1 is the average time spent on generating fingerprint code matrix, T2 is the average time spent on identifying pirates; T1 and T2 are measured in seconds.

No.	Length of Fingerprint Code (Bit/User)	Number of Users	Error Bound	Size of Collusion	Avg. False Negative	Avg. False Positive	T1 (Sec)	T2 (Sec)	Memory Usage (MB)
1	7489	1000	5%	5	0.75%	0%	0.359	0.116	33.89
2	29957	1000	5%	10	2.4%	0.1%	1.280	0.449	102.44
3	11512	1000	1%	5	0%	0%	0.499	0.179	44.66
4	46051	1000	1%	10	0%	0%	1.919	0.722	112.49
5	11512	5000	1%	5	0.1%	0%	2.402	0.847	138.38
6	46051	5000	1%	10	0.3%	0%	9.687	3.430	472.57
7	17269	5000	0.1%	5	0%	0%	3.682	1.333	219.32
8	69077	5000	0.1%	10	0%	0%	14.258	5.442	696.18

Table.5: Experiment Results for Tardos Fingerprint Code

By Tardos' definition, the length of the fingerprint code depends on the size of collusion and error bound. Such property is confirmed by experiments 3 and 5, which share the same length of fingerprint code, even though their number of users is different. The same observation can also be seen in experiment 4 and 6. Tardos also concluded that his fingerprint code for n users that are ε-*secure* against c pirates have length $O(c^2 \log(n/\varepsilon))$. In other words, the length of the fingerprint code is largely decided by the size of collusion; for example, when the size of collusion is increased from $c_1 = 5$ to $c_2 = 10$ and let other parameters remain the same, the length of fingerprint codes is increased by $(c_2 / c_1)^2 = (10 / 5)^2 = 4$ times; such relationship can be verified by experiment 1 and 2, where the length of fingerprint code in the latter is 29957 / 7489 4 times longer than the former one. The error bound also contributes to the length of fingerprint codes, but it does not cause too much change; for example, the error bound in experiment 1 is lowered 50 times to 0.1% in experiment 7 and the size of collusion remains the same, the length of fingerprint codes only increases by approximately 2.3 times.

In all experiments, the error bound varies from 5% to 0.1% but a false positive hardly occurs; false negative is more likely to happen than false positive, but it is still quite rare. On the one hand, it means that Tardos code is unlikely to cause trouble to innocent users, because when the accusation algorithm is not sure about certain users, it will not make accusation; on the other hand, it indicates that using random bits on positions where the pirates see disagree to make up a pirate copy hardly affects the accusation algorithm.

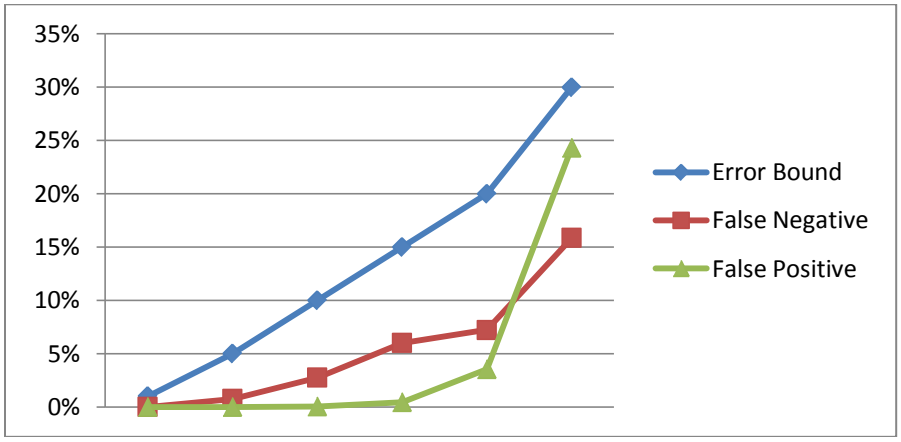

Figure.5: Error Bound, False Negative and False Positive

The figure above further explains the relationship between error bound, false negative and false positive. Six sets of fingerprint codes are generated for 1000 users and 5 pirates; the accusation algorithm is called 2000 times to get average values on false negative and false positive. Data shows that when the error bound is set to 1% and 5%, there is no false positive at all; one false positive error (0.05%) shows up when the error bound is set to 10% and it appears more often in the rest cases. The false positive error rate in figure 5 suggests that if the error bound is set to be low enough, false positive will rarely happens; the cost of doing so is that the distributor will have to work with lengthy fingerprint codes. False negative gradually increases as the error bound increases, meaning that more pirates are getting away without being accused; however, in all cases, there's at least one pirate is accused, which leaves some information for the distributor to chase. Overall, both false positive and false negative reach a high level at the end of the test, but they are still under the error bound. In addition, the observation made in the figure suggests that the distributor could use a relatively high error bound (i.e. 10%) to get shorter fingerprint codes without suffering from serious consequences if the distributor knows for sure that the pirates are very likely to use random bits to make up a pirates copy for the code they see disagree. Error rate and characters of other attack strategies shall be discussed later.

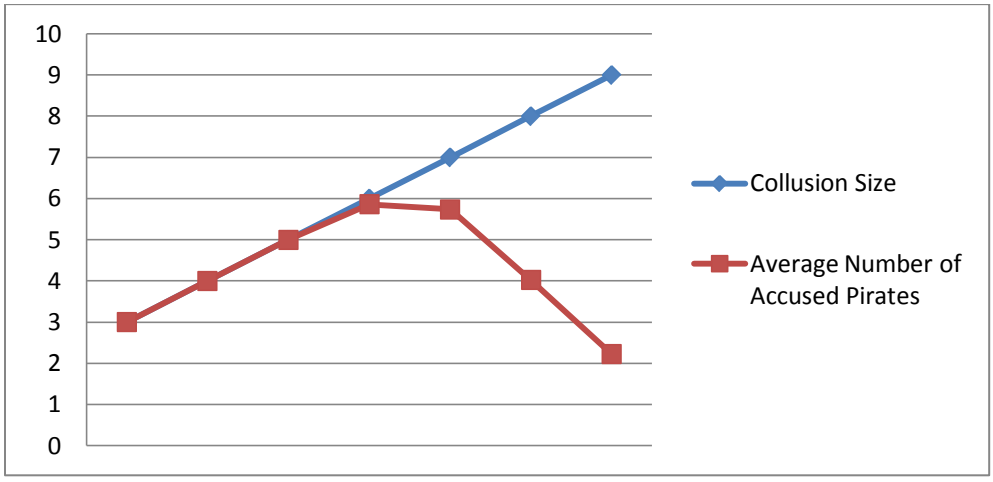

Figure.6: Collusion Size Comparison

Collusion Size	3	4	5	6	7	8	9
Avg. Number of Accused Pirates	3.0	4.0	4.98	5.86	5.74	4.02	2.22
False Negative	0%	0%	0.6%	10.6%	72.2%	98.6%	100%
False Positive	0%	0%	0%	0%	0%	0%	0%

Table.6: Collusion Size and Error Rate

The interconnection between the preset collusion size and the actual collusion size is revealed in figure 6 and table 6. The distributor is asked to estimate the possible size of collusion before constructing the fingerprint code and it cannot be changed afterwards. It is possible, however, that the distributor underestimates pirates' action and the actual size of collusion is larger than expected. Experiments show that even in circumstance like this, some pirates can still be identified.

To test how Tardos codes react to oversized collusion, a fingerprint code matrix for 1000 users and 5 pirates are constructed, the error bound is set to 1%, all pirates use "random choice" as the attack strategy and the accusation algorithm is called 2000 times in order to calculate the average number of pirates being correctly identified. The experiment starts with collusion of 3 pirates; the number is increased by 1 until it reaches 9.

According to experiment results, when the actual size of collusion is less or equal to the estimated value, almost all pirates can be correctly identified. When the actual size of collusion just passes expectation, the accusation algorithm still has a chance of identifying all pirates but in some cases, only a few of them can be picked up. As the actual size of collusion gradually increases, fewer and fewer pirates are accused; in the last experiment, false negative reaches 100%, meaning that the accusation algorithm is no longer be able to identify all pirates, in some cases, no one is accused at all; hence the average number of accused pirates drops to around 2. This observation suggests that the distributor should set the size of collusion slightly higher than the possible size of collusion for better accusation result, because the experiments show that when the actual size of collusion is smaller than expectation, the accusation algorithm is very likely to identify all pirates.

A very noticeable finding from the experiments is that the accusation algorithm never accuses an innocent user in any circumstances. Such property indicates that at this stage, Tardos fingerprint code is very reliable; it seemingly reflexes Tardos' theoretical design, which is to make sure that the number of pirates and marking assumption do not affect the error probability.

Performance-wise, generating a 10,000-bit long fingerprint code averagely requires $4.27 * 10^{-4}$ seconds, the average time spent on processing that fingerprint code and deciding whether or not a user is a pirate is around $1.50 * 10^{-4}$ seconds. As the number of users increases, the time spent on generating the code matrix and accusation increase accordingly; for example, the length of fingerprint code in experiment 3 and 5 is the same, but the number of users in experiment 5 is five times larger than experiment 3; correspondingly, T1 and T2 for experiment 5 are also roughly five times longer than experiment 3. Table 5 shows that even in the last experiment, which is the most intense one, the time spent on code generation and accusation is still reasonable; it is predictable that the system wouldn't spend more than 3 minutes for a system for 50,000 users and 10 pirates with a error bound of 0.1%; if the number of users increases to 500,000, the system is still capable of constructing the fingerprint code matrix in less than half an hour, which is bearable. However, the fingerprinting programme is "RAM hungry". The testing computer has no problem of supporting the last experiment but the memory requirement has already reaches nearly 700M; the testing computer needs 1.2G of memory to run Windows Vista itself and some background programmes, therefore, a large memory space is essential for the fingerprinting programme.

Now it is time to further explore the behaviour of Tardos' fingerprinting codes by using different attack strategies. Similar to previous experiments, a fingerprint code matrix for 1000 or 5000 users will be generated; accusation algorithm for each case will be executed

2000 times; however, the error bound will stick to 5%. It is set to a relatively high value in the hope of magnifying false negative and false positive errors. Experiments have shown that the fingerprint codes are resistant to "random choice" attack, the following table shows how the fingerprint codes react to "Mix and Match" attack.

No.	Number of Users	Error Bound	Size of Collusion	Average False Negative	Average False Positive
1	1000	5%	5	1.1%	0.0%
2	1000	5%	10	1.1%	0.0%
3	5000	5%	5	1.4%	0.0%
4	5000	5%	10	2.1%	0.1%

Table.7: "Mix and Match" Attack

Sometimes the mix and match attack is the only choice for pirates, for example, [12] stated that "the commonly used restricted digit model only allows colluders to 'mix and match' their copies of the content, i.e. the unauthorized copy is only composed of symbols that the attackers have available." Tardos' fingerprint codes are binary; pirates have no other choice except 0 and 1, which makes such attack less restrictive. Table 7 indicates that such attack works better when the number of users and the size of collusion are large. In the last experiment, the false negative rate reaches 2.1%, the accusation algorithm even produces false positive errors.

No.	Number of Users	Error Bound	Size of Collusion	Average False Negative	Average False Positive
1	1000	5%	5	1.15%	0.0%
2	1000	5%	9	4.3%	0.0%
3	5000	5%	5	0.7%	0.0%
4	5000	5%	9	1.2%	0.0%

Table.8: Minority Attack

The size of collusion is lowered to 9 from 10 because odd numbers are needed to make a minority choice. It seems that the fewer user the system have, the more effective the attack is, as the 4.3% false negative rate in the second experiment really stands out; in addition, the more pirates are involved in this attack, the more likely pirates may get away. This can be proved by comparing experiment 1 and 2; as well as 3 and 4. The latter character can be observed in "random choice" attack, "mix and match" attack seems to cause similar but less obvious results.

No.	Number of Users	Error Bound	Size of Collusion	Average False Negative	Average False Positive
1	1000	5%	5	1.2%	0.0%
2	1000	5%	9	2.1%	0.0%
3	5000	5%	5	1.3%	0.0%
4	5000	5%	9	1.6%	0.0%

Table.9: Majority Attack

Majority attack is similar to minority attack in many aspects; they share similar characters, for example, the false negative rate in experiment 2 sticks out. But generally

speaking, majority attack works better than minority attack, the average false negative rate in many experiments are higher than the corresponding ones in previous experiments.

Both minority attack and majority attack fail to trigger any false positive error. The only false positive error in "mix and match" attack is as low as 0.1%. These experiment results provide evidence that Tardos fingerprint codes are robust against many typical attacks. It is believed that if the error bound is low enough, false positive error can be largely avoided.

An Effective Attack Strategy

In the previous section, 4 attack strategies are implemented and are used to test Tardos' fingerprint code. These strategies use different approaches to make up the pirate version but they have one thing in common, that is they never take the accusation algorithm into account. In this section, a new attack strategy is presented; it guarantees that if there are *c* pirates in the coalition, the accusation algorithm will accuse at most *c -1* of them.

Theory

Tardos' accusation algorithm is a score-based system; scores are added or subtracted according to the rules stated in the previous section. A very important fact about the accusation algorithm is that the probability values are involved and they are able to significantly influence the score.

For example, assume that the accusation algorithm is inspecting pirate *j* for a pirate copy *y*, at the i^{th} position, a probability value p_i is equal to *0.9*, according to the rules:

$$U_{ji} = \begin{cases} \sqrt{\frac{1-0.9}{0.9}} \approx 0.33, if\ X_{ji} = 1 \\ \\ -\sqrt{\frac{0.9}{1-0.9}} = -3, if\ X_{ji} = 0 \end{cases}, score = y_i U_{ij}$$

The rules indicate that if the i^{th} position of pirate copy $y_i = 1$ and the original fingerprint code for pirate *j* is 0, the score for this pirate will be hugely deducted. In the opposite situation, where the value of p_i is low, for example, $p_i = 0.1$, the value of U_{ji} will become:

$$U_{ji} = \begin{cases} \sqrt{\frac{1-0.1}{0.1}} = 3, if\ X_{ji} = 1 \\ \\ -\sqrt{\frac{0.1}{1-0.1}} \approx -0.33, if\ X_{ji} = 0 \end{cases}$$

It means that if the i^{th} position of pirate copy $y_i = 1$ and the original fingerprint code for pirate j is also 1, the score for this pirate will be significantly increased. In general, if pirate j wants to avoid being accused, he should adjust the pirate copy accordingly. At the i^{th} position, if the probability value p_i is high and his original code $X_{ji} = 0$, the pirate copy y_i should be set to 1 to take advantage of large deduction, for the rest of the case, y_i should be 0 to void increasing the score.

A problem of this strategy is how to get p_i because the probability array is kept secret from users. However, within the coalition, pirates could be able to estimate p by comparing their copies. For example, at i^{th} position, if most of their codes are 1, p_i will be set to a high value; otherwise p_i is set to be a low value. To conduct an experiment, a fingerprint code matrix for 1000 users and 5 pirates with an error bound of 5% is generated. Figure 7 shows that the estimation method is fairly accurate. The data are gathered from the experiment, where the difference between the actual probability values and the estimated values are recorded; in total, there are 3836 entries, most of their values are between -0.2 to 0.2.

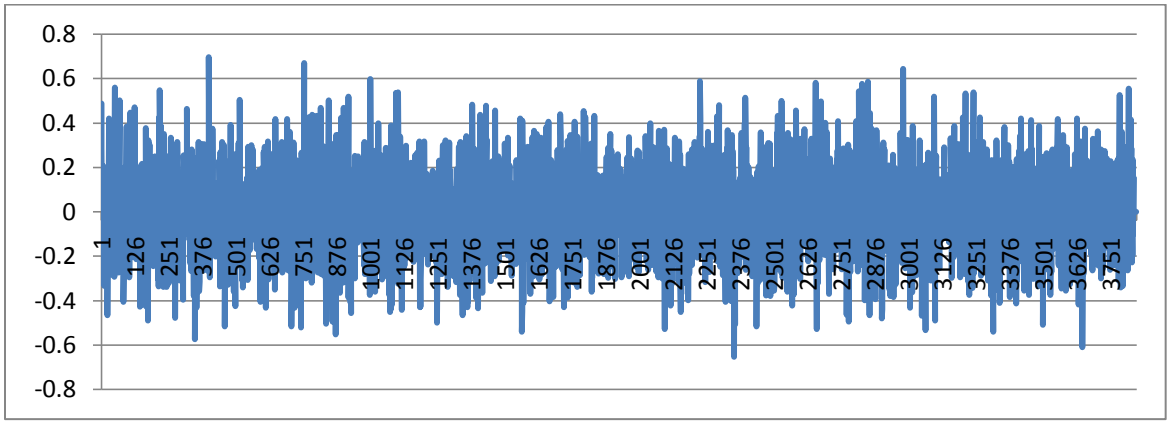

Figure.7 Probability Estimation

Implementation

```
case 4:
   int oneCount = 0;
   for (int j = 0; j < collusionSize; j++) {
      if (codeMatrix[pirateId[j]][i])
      oneCount++;
   }
   double prob = (double)oneCount / (double)collusionSize;
   if (prob > 0.5 && !codeMatrix[pirateId[4]][i])
      pirateCopy[i] = true;
   else
      pirateCopy[i] = false;
break;
```

Code Fragment.17: An Effective Attack Strategy

The new attack strategy is the 5th attack strategy in TardosCodesAttacker.java. Details of how the probability values are estimated are explained in the code fragment: a *for* loop is used to calculate how many 1s in the i^{th} position, then the number is divided by the size of collusion to get the estimated probability value. Apparently, the larger the collusion is, the more accurate the estimation becomes.

Experiments and Evaluation

No.	Number of Users	Error Bound	Size of Collusion	Average Number of Pirates Detected	Average False Negative	Average False Positive
1	1000	5%	5	3.99	100%	0%
2	1000	5%	10	8.95	100%	0%

Table.10 Test Results for the New Attack Strategy

Two experiments are conducted to test whether or not the new attack strategy can help one pirates get away; some key parameters are listed in the table above, each experiment is repeated 200 times. The experiments show that the new attack strategy is effective, in each experiment, at least one pirate is exempted from being accused, very occasionally, more than one pirate is exempted. Because the accusation algorithm is no longer able to identify all pirates, the false negative error occurs in all accusation, which makes the average false negative rate reaches 100%. The good news is that in such circumstance, no innocent user is accused.

To sum up, letting one pirate getting away does not seem to be a major breakthrough, but it could cause significant impact on Tardos' fingerprinting scheme, because such attack strategy always works, the pirates could decide to let their most important member be exempted from being accused.

Limitation

Between the two types of errors, false positive and false negative, Tardos fingerprint codes treats the former one as the important one and makes efforts to make sure that an innocent user is not accused. The latter type of error, one the other hand, appears more frequently, even though it is under the error bound. If the pirates have developed some smart attack strategies, or the size of collusion is larger than the distributor expected, or pirates somehow break the marking assumption, there will be a good chance that the accusation algorithm is forced to accuse no one. Such outcome could make the whole fingerprinting system practically useless. After all, why would the distributor want to spend lots of money on the system and get no pirate in the end? However, Tardos admitted that making the false negative error disappear seems to be a tough job; a possible solution to this problem is to develop a fingerprint that would "mix some of the deterministic features of the IPP codes with the probabilistic properties of the Boneh-Shaw codes and the codes of this paper".

Another flaw in Tardos fingerprint codes is caused by the fact that the length of fingerprint code *m* does not depend on number of users *n*. In his paper, Tardos stated that "if *n* is larger than 2^{m+1}, then most users must share their codeword with another user. In this case even a single pirate distributing his copy is impossible to catch without a high risk of failure". The problem unveiled here seems to be serious, but a pirate may not be so easy to take advantage of it because 2^{m+1} could be an extremely large number, it may be too difficult for the distributor to support so many users in the fingerprinting system. Even so, the distributor should still be aware of this problem in order to make the fingerprinting system is somehow future-proof.

In his work [13], H. Schaathun stated that Tardos fingerprint code is subject to adverse selection, that is, "pirates who perceive a high error probability after comparing their copies are more likely to go on with the crime, because they face a lower risk of being discovered and penalised". Adverse selection could be a problem to the distributor because Tardos' error bound is based on statistics and it cannot deal with exceptions well. However, due to the scale of this report, the implementation of adverse selection will not be discussed.

The experiments conducted in the previous section have shown a major performance issue in Tardos fingerprinting system: an overwhelming need for memory space. The first experiment in table 3 surely does not look so bad but not the last one. One may argue that 700M of memory is universally affordable, however, the last experiment in table 3 is only close to practical use, for example, the number of readers of a book can easily surpass 5000; in music industry, a CD album can be sold millions of copies. It is natural to assume that the best sellers attract more pirates, so to make sure that the fingerprinting system works properly, the size of collusion will have to be reasonably high. Also, for a commercial organisation, it is always a bad idea to accuse its legal users for piracy, so the error bound will need to be low enough. Tardos commented that the length of fingerprint code is optimal "within a constant factor amongst all codes for *n* users that are *ε-secure* against coalitions of size *c* if $\varepsilon < 1 / (100c^a)$ for a fixed $a > 1$ and $\varepsilon < 1 / n^b$ for a fixed $b > 0$". As *c* and *n* are expected to be large, the error bound *ε* is going to be very small. Combining all these factors, generating the fingerprint code matrix will become a heavy burden on the distributor; such task may not even be performed on mainstream hardware.

Another hardware-related issue is that the Java Virtual Machine uses only one core of a multi-core processor. For a quad-core processor, only one fourth of its computational power is used for compiling and executing codes, which is a waste of resource and time. But there is nothing can be done at the moment as redesigning JVM is a massive task; Intel has optimised its C compiler for multi-core processor, programmers can expect Sun Microsystems to do so soon.

Regarding the implementation of Tardos fingerprint code, experiments have shown that there exists some "logic traps" in the code. To understand what a logic trap is, let's have a look at how a serial number verification programme can be cracked by hackers. Assume that the verification programme takes a serial number as input and returns a boolean value as output; naturally, if the given serial number is authentic, the programme

will return true so that the rest of the software can be activated. To crack the verification programme, a hacker can simply add a "NOT" mark to the return value of the programme, as a result, the programme will return false if the serial number is valid, now the hacker can feed whatever he wants to the programme, as long as it is not a valid serial number, the verification programme will return true.

Building a logic trap requires minimum changing to the programme but the outcome is significant. There are several places in the implementation of Tardos fingerprint codes are suitable for planting a logic trap, for example, the boolean expression in code fragment 11 could be a target. Using the original code, the accusation algorithm may have the following output:

```
FN Count: 0, False Negative: 0.0
FP Count: 0, False Positive: 0.0
Accusation Threshold (Z): 299.57322735539907
Average Score for Innocent Users: -4.71314868202395
Average Score for Pirates: 473.08583053166257
```

Figure.7: Normal Feedback Info

If the line

$$fp[i] = Math.random() < p[i];$$

in code fragment 5 is changed to this one:

$$fp[i] = Math.random() > p[i];$$

which reverses the rule for Tardos fingerprint code generation, the accusation algorithm will return a very different output to the distributor:

```
FN Count: 0, False Negative: 0.0
FP Count: 20, False Positive: 100.0
Accusation Threshold (Z): 299.57322735539907
Average Score for Innocent Users: NaN
Average Score for Pirates: 71228.36579683977
```

Figure.8: Interfered Feedback Info

Figure 8 shows that the result of accusation is completely messed up, because of 1 bit of change, the accusation algorithm accuses all innocent users and none of the pirates. Such logic trap could be hard to debug as it looks like there is something wrong with the accusation algorithm.

The accusation algorithm itself is also a good place for setting up the trap. For example, the hacker can change the construction of the U matrix and score calculation method in code fragment 7 to the following:

```
if (!codeMatrix[i][j] && piratedCopy[j]) {
   sum += Math.sqrt((1 - p[j]) / p[j]);
}
else if (codeMatrix[i][j] && piratedCopy[j]){
   sum += -Math.sqrt(p[j] / (1 - p[j]));
}
```

Code Fragment.18: Altered Score Calculation Method

Similar to the previous example, the effect of such modification is to reverse Tardos definition. Experiments show that the impact of the modification is that all users are considered to be innocent and therefore, no one is accused. The typical output of the accusation algorithm after the modification is shown below:

```
FN Count: 20, False Negative: 100.0
FP Count: 0, False Positive: 0.0
Accusation Threshold (Z): 299.57322735539907
Average Score for Innocent Users: -404742.4633185016
Average Score for Pirates: NaN
```

Figure.9: Feedback Info for Altered Score Calculation Method

Logic traps can be avoided by introducing some mechanisms to protect the programme, which means that the distributor will have to employ extra computing power to deal with the code protection; it may also make the fingerprinting system unnecessarily complicated. Alternatively, logic traps can be seen as a managerial problem but the related discussion is outside the scope of this report.

4. Improvements of Tardos Fingerprint Codes

Code Length Oriented Improvements

Japanese researchers Koji Nuida et al proposed an improved Tardos fingerprinting scheme in [14], it differs from the original version in several ways. For example, the authors claims that using their formula of code lengths, the length of their fingerprint code is reduced to less than or almost equal to 1/20 of Tardos codes in many practical settings. The evidence of such claim can be found in examples given in the paper, where 2, 4, 6 and 8 pirates are assumed to be working together.

To make their fingerprint code more practical, a relaxation of marking assumption (δ marking assumption) is introduced. It states that "the number of undetectable positions in which *y* differ from the pirates' codewords is not more than $m\delta$", where *y* is the pirate copy, *m* is the code length and $\delta \geqslant 0$ is a fixed parameter. In other words, the authors now assume that there is a chance that the pirates may change the code they see agree. When $\delta = 0$, δ marking assumption is identical to marking assumption.

A very interesting assumption made in the improvement scheme is that besides modifying the fingerprint codes, the pirates may also erase some of them; hence, the pirate version of the fingerprint codes may have a larger alphabet {0, 1, ?}, where ? denotes a erased bit. As a result, the accusation algorithm is modified to deal with the extra bit.

In addition, the original accusation algorithm is altered so that only one pirate is accused at the end. Koji Nuida et al still use scores for accusation, but instead of comparing scores for each user to a fixed threshold, they are compared with each other and the user with the highest score is accused. It has been proved in their work that if all other attributes are remain the same, the error probability of such accusation algorithm is not more than the original one; in fact, it "results in significant reduction of the error probability".

Oded Blayer and Tamir Tassa attempted to reduce the length of Tardos fingerprint code form a different approach in [15]. They retraced Tardos' analysis of the scheme and extracted from it all constants that were arbitrarily selected. Eventually, they identified seven constants that can be modified to shorten the length of fingerprint code.

They also tried to reduce the length of the code by studying the error probability ε. In [10], Tardos stated the following two theorems:

Theorem 1: Let (X, σ) be distributed according to $F_{nc\varepsilon}$. Let $j \in [n]$ be an arbitrary user, let $C \subseteq [n]\backslash\{j\}$ be a coalition of arbitrary size not containing j, and let ρ be any C-strategy. We have

$$P[j \in \sigma(\rho(X))] < \varepsilon.$$

Theorem 2: Let (X, σ) be distributed according to $F_{nc\varepsilon}$. Let $C \subseteq [n]$ be a coalition of size $|C| \leqslant c$, and let ρ be any C-strategy. We have

$$P[C \cap \sigma(\rho(X)) = \emptyset] < \varepsilon^{c/4}$$

All symbols in these theorems are used consistently as earlier. Now, let ε denote the error bound in theorem 1 and let ε' denote the error bound in theorem 2. In Tardos fingerprinting scheme, ε' is much less than ε. However, Oded Blayer and Tamir Tassa believe that it is better to set ε' >> ε, because accusing an innocent user is far worse than letting a pirate get away. They also believe that pirates can hardly remain undetected for long because they tend to repeat their actions.

Memory Usage Oriented Improvements

Some efforts have been made to save memory space in implementation section; in [16], Koji Nuida et al stated that Tardos fingerprint code requires large memory space because it uses "certain continuous probability distribution in codeword generation" and the probability distribution must be stored in high accuracy to ensure the code works, hence, more bits in memory are used to represent the probability distribution. A possible solution appears in [17], where a scheme for shorten the fingerprint code is given by using finite possibilities. Koji Nuida et al reviewed such solution in their paper, but they claimed that the proposed scheme cannot be practically implemented; on the other hand, they further exploited the so called c-indistinguishability condition, which was discussed in [17], in their own proposal to shorten Tardos' code.

Experiments results listed in [16, Table 1] showed that memory usage is significantly lowered. The only problem is that the collusion size they assumed, 2 in Case 1 and 4 in Case 2, was too small. Koji Nuida et al never discussed the memory usage of the whole code matrix; they instead talked about how Tardos' code can be shortened in most part of their paper.

Overall, the improvements proposed in abovementioned literatures tend to limit their discussions with small number of pirates, usually fewer than 8. Some suggestions and assumptions, such as the relaxation of marking assumption, are surely practical but using small number of pirates cancels the benefits.

5. Conclusion

The first part of this report studies two historic fingerprinting schemes, namely Wagner's code and Boneh and Shaw's code; they are milestones in the history of digital fingerprinting development but their weaknesses are also exposed by implementing and testing the schemes. Tardos' fingerprinting scheme is more advanced than its ancestors; it employs full randomisation to generate shorter codeword. It is also implemented and tested, plus, literatures regarding the improvements of the scheme are reviewed.

All fingerprinting schemes studied in this report have been widely inspected by many researchers, but what makes this report special is that these fingerprinting schemes are actually implemented and evaluated from a practical point of view, some of the problems revealed in the report, such as the "logic trap" or the memory usage problem, can never be precisely studied in theory. To test Tardos' fingerprint code, some experiments are conducted with relatively large collusion size. Many literatures discuss collusion up to 8; in this report, the number has been raised to 10, which makes it one step closer to realistic usage.

"Inconvenient Truths"

In [8], Boneh and Shaw have discussed how fingerprinted objects can be distributed. They suggested that a fingerprinted object can be viewed as a combination of two parts: one of them is the public data D, which would be universally received by all users; the other part is some extra bits - the private data M_u, it would be unique to each user. The reason that Boneh and Shaw came up with this scheme is that they believe sending each user an entirely unique copy is impractical for products that are mass produced, such as "electronic books, software or CD-ROMs".

However, their scheme does not seem to be practical either, because nowadays, distributors still rely on CDs or DVDs to distribute digital contents like music or video games; the most economic and technically possible approach to mass produce CDs or DVDs is to make each copy from a master copy. A master copy can be used to duplicate or replicate discs; the two processes sounds really similar but the difference is that former uses a laser on a disc duplication machine to quickly burn exact copies of a master copy onto blank discs; the latter process uses a glass master of the original copy, which is then used to stamp the data directly onto discs. Duplication is relatively cheap and is able to handle up to 1000 copies while replication is fairly expensive to setup but is capable of creating millions of copies. It is regretful to say that neither of these methods supports disc customisation. In other words, the distributor cannot write M_u on discs. A possible solution to this problem is to modify the disc duplicator or replicator to allow more data to be added to the disc after duplication/replication is completed, and then at the end of the production line, uses an extra machine to write small amount of unique fingerprint data on discs. But whether or not this method is practical in industrial scale is unknown.

As broadband becoming faster and cheaper, some distributors start to publish their products via the Internet. Perhaps online music stores such as iTunes Store are the pioneers in commercially distributing products over the network, as music files, such as MP3, are usually small, even dial-up connection can handle it. Although digital files are easier to edit, it is still time consuming to embed certain amount of fingerprinting data into the objects.

Another reason that may stop fingerprinting technologies becoming popular is the immaturity of watermarking technologies. Digital watermarking is believed to be as old as digital fingerprinting, but it somehow attracts more researchers. And yet, most of the watermarking schemes are not good enough to be used with fingerprinting schemes, as the layered model suggests. The major problem of watermarking is that if the watermarked objects are attacked or transformed (i.e. format transformation), it cannot be guaranteed that the watermark can be fully recovered. Such problem is critical because digital fingerprint relies on digital watermarking techniques to embed itself into objects, if it cannot be correctly retrieved from the pirates, the distributor will be forced to use the broken fingerprint code to accuse innocent users or let pirates get away and neither of these cases is favoured.

To sum up, it is difficult to deploy digital fingerprinting schemes using current technologies. Alternative solutions suggested in this report are either too time consuming or too expensive. The distributor will have to wait until faster hardware or a novel distribution method and watermarking schemes become available.

In addition, the feasibility of using digital fingerprint to trace information leakage is slim. There are reasons why digital fingerprinting is not popular for such purpose. For example, assume that the government suddenly finds existence of extraterrestrial life on the earth and tries to do some research while remaining it secret. Fingerprinting technologies can be used to fingerprint all relevant documents or photos. But if someone who has enough creditability tells the media that “we are not alone”, the confidentiality will be broken immediately and fingerprinting has nothing to stop or trace the “deep throat”. The point of this example is to show that controlling information largely requires managerial techniques instead of computer technologies, digital fingerprint alone cannot stop people taking advantage of what they know.

Finally, the fingerprinting code itself may have some reliability issues. Many papers discuss fingerprinting schemes with assumptions, typically, marking assumption, and statistics. The cruel reality is, however, assumptions and statistics can be wrong.

Future work

The future work of this dissertation will be focusing on two aspects. One of them is investigating improvement schemes for Tardos’ code in details. Most of such schemes attempt to produce shorter codeword and some of them have been reviewed in chapter 4;

another improvement could be reducing false negative rate. Tardos has mentioned that such improvement is not going to be easy but it is worth trying.

Moreover, pirates' behaviour will be studied in the future. In many academic papers, pirates only exists in theory, they are studied as models or assumptions; no one really cares who they are and why they make pirate copies. Good cops crack cases easily because they know criminals' strategies, they know how they think, and the same rule applies to academic researches. If the scientists know who they are dealing with, they may come up with some brilliant ideas.

6. Reference

[1] BBC News (30 September 2006), *Film piracy 'costs economy $20bn'*, http://news.bbc.co.uk/1/hi/entertainment/5395218.stm, Accessed on 25 May 2009

[2] Candace Lombardi, CNET News (23 May 2006), *Study: Software piracy costs $34 billion*, http://news.cnet.com/Study-Software-piracy-costs-34-billion/2100-1014_3-6075629.html, Accessed on 25 May 25, 2009

[3] The Business Software Alliance and IDC, www.bsa.org, (May 2009), *Sixth Annual BSA-IDC Global Software 08 Piracy Study*

[4] Pigna, Kris, 1UP.com, (24 September 2008), *EA Hit with Class Action Lawsuit over Spore DRM*, http://www.1up.com/do/newsStory?cId=3170131, Accessed on 26 May 2009

[5] Schonfeld, Erick, TechCrunch (washingtonpost.com) (14 September 2008), Spore And The Great DRM Backlash, http://www.washingtonpost.com/wp-dyn/content/article/2008/09/14/AR2008091400885.html, Accessed on 26 May 2009

[6] Cheng, Jacqui, Ars Technica (April 22, 2008), *DRM sucks redux: Microsoft to nuke MSN Music DRM keys*, http://arstechnica.com/news.ars/post/20080422-drm-sucks-redux-microsoft-to-nuke-msn-music-drm-keys.html. Accessed on 26 May 2009

[7] Thomson, *NexGuard Forensic Marking*, http://www.thomson.net/GlobalEnglish/Products/content-tracking-and-security/nexguard/nexguard-forensic-marking/Pages/default.aspx, accessed on 26 May 2009.

[8] Dan Boneh and James Shaw (1998), *Collusion-Secure Fingerprinting for Digital Data*, Information Theory, Volume 44, Issue 5, Sep 1998 Page(s):1897 – 1905

[9] N. Wagner (1983), *Fingerprinting*, Security and Privacy, IEEE Symposium on, pp. 18, 1983

[10] G. Tardos (2003), *Optimal Probabilistic Fingerprint Codes*, Proceedings of the 35th Annual ACM Symposium on Theory of Computing, 2003

[11] H.G. Schaathun (2006), *On Watermarking/Fingerprinting for Copyright Protection*, Proceedings of the First International Conference on Innovative Computing, Information and Control - Volume 3

[12] Boris Skoric, Stefan Katzenbeisser, Hans Georg Schaathun, Mehmet U. Celik (28 May 2009), *Tardos Fingerprinting Codes in the Combined Digit Model*

[13] H.G. Schaathun (2004), *Binary Collusion-Secure Codes, Comparison and Improvements*, No.275 Reports in Informatics, University of Bergen

[14] Koji Nuida, Satoshi Fujitsu, Manabu Hagiwara, Takashi, Kitagawa, Hajime Watanabe, Kazuto Ogawa, Hideki Imai (2007), *An Improvement of Discrete Tardos Fingerprinting Codes*, Applied Algebra, Algebraic Algorithms and Error-Correcting Codes

[15] Oded Blayer and Tamir Tassa (22 November 2007), *Improved Versions of Tardos' Fingerprinting Scheme*, Designs, Codes and Cryptography archive, Volume 48, Issue 1

[16] Koji Nuida, Manabu Hagiwara, Hajime Watanabe, and Hideki Imai (2008), *Optimization of Memory Usage in Tardos' Fingerprinting Codes*

[17] M. Hagiwara, G. Hanaoka and H. Imai (2006), *A short random fingerprinting code against a small number of pirates*, Proc. AAECC, 2006, LNCS 3857, pp. 193–202.

[18] G. R. Blakley, C. Meadows and G. B. Purdy (1985), *Fingerprinting long forgiving messages*, Proc. of Crypto ’85 Springer-Verlag Berlin, Heidelberg, 1985, pp. 180–189.

[19] B. Chor, A. Fiat and M. Naor (1994), *Tracing traitors*, Proc. of Crypto’94 LNCS 839, Springer-Verlag Berlin, Heidelberg, 1994, pp. 257–270.

[20] Jie Yang, Xiaoxia Xu (2006), *A Robust Anti-collusion Coding in Digital Fingerprinting System*, Signal Processing, 2006 8th International Conference

[21] Ho Wook Jang, Won-Gyum Kim, and Seon Hwa Lee (October, 2003), *An Illegal Contents Tracing System Based on Web Robot and Fingerprinting Scheme*

[22] Ingemar J. Cox, Matthew L. Miller and Jeffrey A. Bloom (200), *Digital Watermarking*, Morgan Kaufmann Publishers, pp. 472-473