RANDOMISED DYNAMIC TRAITOR TRACING

Jarrod Trevathan
School of Mathematical and Physical Sciences
James Cook University

Wayne Read
School of Mathematical and Physical Sciences
James Cook University

Keywords: Restricted multimedia content, piracy, watermarking, complexity analysis.

Abstract: Dynamic traitor tracing schemes are used to trace the source of piracy in broadcast environments such as cable TV. Dynamic schemes divide content into a series of watermarked segments that are then broadcast. The broadcast provider can adapt the watermarks according to the pirate's response and eventually trace him/her. As dynamic algorithms are deterministic, for a given set of inputs, the tracing algorithm will execute exactly the same way each time. An adversary can use this knowledge to ensure that the tracing algorithm is forced into executing at its worst case bound. In this paper we review dynamic traitor tracing schemes and describe why determinism is a problem. We ammend several existing dynamic tracing algorithms by incorporating randomised decisions. This eliminates any advantage an adversary has in terms of the aforementioned attack, as he/she no longers knows exactly how the tracing algorithm will execute. Simulations show that the randomising modifications influence each dynamic algorithm to run at its average case complexity in terms of tracing time. We provide an efficiency analysis of the ammended algorithms and give some recommendations for reducing overhead.

1 INTRODUCTION

The proliferation of high speed Internet connections, mobile phones and Cable TV (PayTV) has made online multimedia content accessible to the masses. Downloading movies, songs and pay-per-view content is becoming increasingly popular. While this technology is convenient, it has also been accompanied by a significant rise in piracy. File sharing programs such as DC++ [1] make it easy to for anyone to distribute pirated material to countless other illegitimate viewers.

An individual that makes a copy of, and/or views content in the above context is referred to as a *pirate*. The traditional method for preventing piracy is to encrypt the content and issue the users with a *decoder*. The decoder is a program or device that contains a set of keys capable of decrypting the message. Although this approach prevents unauthorised parties from eavesdropping on the content, piracy can still occur in the following ways:

- An authorised user retransmits their copy of the digital content to an unauthorised user; or
- A pirate has access to a set of legitimate user's keys that allows the pirate to construct a *pirate decoder* which is capable of interpreting the encrypted content.

In either scenario, such legitimate users who willingly aid a pirate are known as *traitors*.

While an ideal solution to piracy is to stop it from occurring in the first place, this is almost impossible to achieve in reality given the presence of traitors. It is therefore prudent to assume that some piracy will occur so that counter measures can be taken to deal with the threat. A *traitor tracing scheme* is a mechanism that is designed to trace and cut off the source of a pirate decoder or transmission once piracy has been detected (see (Chor *et al*, 1994; Pfitzmann 1996; Trevathan *et al*, 2003)).

A traitor tracing scheme can be considered as either *static* or *dynamic*. In static schemes, keys found in a pirate decoder are used to trace the legitimate user who was originally allocated the keys. Keys are allocated once and are not changed throughout the lifetime of the content. This makes static tracing schemes ideal for distribution of one-off content such as DVDs, but is of limited use to online/real-time con-

[1] http://www.dcplusplus.sourceforge.net

Trevathan J. and Read W. (2006).
RANDOMISED DYNAMIC TRAITOR TRACING.
In *Proceedings of the International Conference on Signal Processing and Multimedia Applications*, pages 323-332
DOI: 10.5220/0001571203230332

tent.

Dynamic schemes are able to adapt to a pirate's actions in real-time by changing key allocations at certain intervals throughout the content. Watermarks are embedded in the plaintext content. The tracing algorithm in this case examines the watermark found in a pirate copy and links this back to the particular users who received the same watermark. The scheme attempts to isolate suspected traitors by changing the marking scheme in response to the pirate's feed back. When a traitor is identified the algorithm disconnects them from future broadcast.

Various schemes for dynamic traitor tracing have been presented (see (Fiat and Tassa, 1999; Berkman *et al*, 2000; Safavi-Naini and Wang, 2000)). However, all of these schemes are deterministic. This is a problem as a traitor is able to run through the tracing algorithm in advance and totally deduce the marking strategy that will be used. While this does not prevent a traitor from being caught, this situation is undesirable, as it allows a traitor to force the algorithm to run at its worst case complexity through strategic use of feedback.

This paper discusses the implications of determinism in dynamic traitor tracing schemes. We propose several amendments to existing dynamic tracing algorithms that incorporate randomised decisions. Through the use of random decisions, an adversary is unable to predetermine the manner in which the tracing algorithm will execute. This effectively removes any benefit that can be gained from altering the input data for the tracing process. Simulations have shown that there is a significant gain in tracing efficiency and hence security by making our proposed amendments.

This paper is organised as follows: Section 2 introduces traitor tracing schemes. Section 3 describes how dynamic traitor tracing schemes operate. Section 4 shows how to incorporate randomness into several existing dynamic traitor tracing schemes. Section 5 provides an efficiency analysis of existing dynamic schemes and shows the effect on security that randomisation provides. Section 6 provides some concluding remarks.

2 TRAITOR TRACING SCHEMES

A traitor tracing scheme is an algorithm employed by a content provider that enables piracy to be traced back to its original source (a traitor). Once a traitor has been discovered the content provider is then able to take legal or extra-legal measures against them. The problem of tracing traitors can be addressed via careful key distribution or marking of content.

Tracing Goals

The goals of traitor tracing schemes and the desirable properties they should exhibit include:

- Capable of tracing the source of the piracy;
- Must not harm legitimate users by falsely incriminating them, or by allowing traitors to be able to frame them;
- The source of piracy should be denied from receiving future transmissions;
- Supply legal evidence of the pirate's identity;
- Provide some value in deterring potential traitors.

Terminology/Notation

The following terminology and notation is used throughout this paper. A user is denoted by u and is the authorised receiver of the digital content. U is the set of all authorised users $U = \{u_1, ..., u_n\}$, where u_i is the ith user in U. The broadcast provider encrypts digital content and distributes it to U. A session is the period for which the decryption keys are valid e.g., a few minutes of video. Every user is allocated a set of decryption keys that allows them to view the content. This is referred to as a user's personal key or codeword. Some users may collude in order to distribute illegal copies of the content to unauthorised users. We refer to such users as traitors and to their coalition as the pirate and denote them by $T = \{t_1, ..., t_p\}$, where $T \subset U$. Some of the traitors may combine a subset of their keys and construct a *pirate decoder*.

When a pirate decoder is found either in hardware or software, its behaviour can be examined to determine which keys it is comprised of. By treating a captured decoder as a black box and observing its output when given particular inputs, the decryption keys can be traced back to the members of T. Hence, there is no need to tamper with the decoder in any way. It is assumed that there are no mechanisms in place to thwart the tracing algorithm. The manner in which the tracing proceeds depends on the type of scheme used.

Components of a Traitor Tracing Scheme

There are three main components that jointly constitute a traitor tracing scheme:

1. *Key generation/distribution scheme:* used by the data supplier to generate and distribute users' personal keys. The data supplier has a *master key*, α, that defines a mapping $P_\alpha : U \mapsto K$, where U is the set of possible users and K is the set of all possible personal keys.

2. *Encryption/decryption scheme:* an encryption scheme, E_α, is used by the data supplier to encrypt the session key before broadcasting, and a decryption scheme, D_β, is used by each authorised user (i.e., its decoder) to decrypt the session key, where $\beta = P_\alpha(u_i)$. The session key is used to encrypt

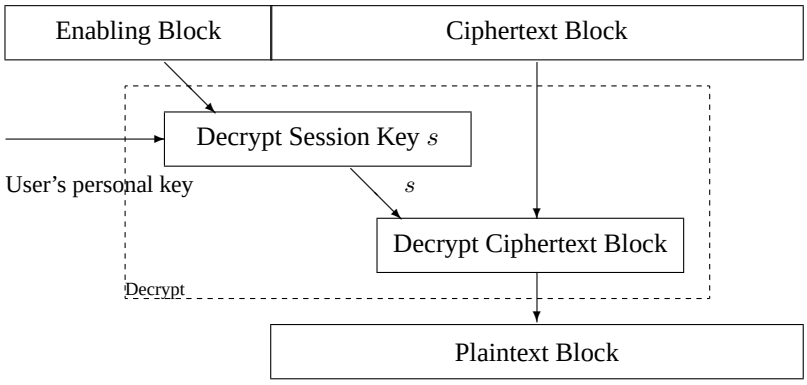

Figure 1: Encryption/Decryption of a segment.

data content using an off-the-shelf symmetric encryption scheme such as *Advanced Encryption Standard* (AES) (Daemen and Rijmen, 1998).

3. *Tracing algorithm:* used upon confiscation of a pirate decoder, to determine the identity of one or more traitors. It is assumed that the contents of a pirate decoder are unknown. Instead, the tracing algorithm must access a pirate decoder as a black box, and perform the tracing based on the decoder's response to different input ciphertexts.

The off-the-shelf encryption scheme E must be a block cipher. The data supplier divides the content into sessions whose size is a multiple of a block size accepted by E. For each content session M, a typical traitor tracing scheme will output two blocks. A *ciphertext block* B_c is the result of encrypting M by the encryption scheme E using a *session key*, s, randomly chosen from the key space of E : $B_c = E_s(M)$. A second block is called an *enabling block*, because it contains data that enables each authorised user to obtain the session key and decrypt the corresponding ciphertext block (see Figure 1.3).

3 DYNAMIC SCHEMES

Static schemes are often limited to one-time marking of content such as DVDs. Legal recourse is only taken after a pirate copy has been found which often involves the entire content being stolen before the tracing procedure can even begin. In situations where there is some ongoing relationship between the users and the content provider (such as PayTV or Internet multicast applications), static schemes are often too limited and miss out on opportunities to help trace and/or punish traitors.

Dynamic traitor tracing assumes there exists an online feedback channel from a pirate to the broadcast provider. Such feedback may be the detected retransmission of the content by the pirate or the publication of decryption keys in some publicly accessible place (such as on the Internet). By observing the feedback sequence, a distributor can search for traitors by adaptively changing the content marking scheme in response to pirate activity. Eventually an individual traitor can be isolated and disconnected from future transmittal of information.

Dynamic tracing can be best described by the following combinatorial game which is an abstraction of the scenario described above. This game serves as a model for all dynamic schemes presented throughout this section.

A *session* is the duration of a piece of information to be broadcast. A session is split up into *segments* $j_1, ..., j_m$. Each j is divided into ℓ *variants* $v_1, ..., v_\ell$, where each variant is unique (i.e., $v_i \neq v_k, i \neq k$). All variants have the same information but contain a robust watermark (using the methods of (Boneh and Shaw, 1995)). Given any set of variants $v_1, ..., v_l$ for a particular segment, it should be impossible to generate another variant that cannot be traced back to the original variants from which it was comprised. The terms *colour*, *mark* and variant are used interchangeably.

In each *round*, the algorithm assigns a variant v of the current segment to every user. This is followed by the pirate's response, which is a variant of one of its supporting traitors. The pirate's series of responses is referred to as a *feedback sequence* and is denoted by F. When feedback f_i, for segment j_i is detected from the pirate (where $1 \leq i \leq l$), f_i is appended to F.

When a particular variant that was only given to one set of users is found in F, this implies that the set contains a traitor. A traitor is *located* if only a single user (and no one else) was the recipient of a variant that is given as the pirate's response. Since the number of traitors is not known in advance, this is the only way to locate a traitor. A traitor that has been

located is *disconnected*, i.e., removed from U thereby excluding them from all succeeding rounds.

The pirate loses the game when s/he does not answer. This most likely means that s/he has gone out of business because the algorithm has located and disconnected all traitors. The performance of the scheme is measured by the maximal number of variants used in any single round (*marking alphabet*) and also the number of rounds needed in the worst case (*convergence time*). There is a trade-off between the two measures.

A pirate does not have to decide in advance who the p traitors are. Instead, s/he can choose the traitors adaptively throughout the game. However, once he decides to corrupt a user, s/he is committed to this decision until the end of the game.

(Fiat and Tassa, 1999) present three deterministic dynamic tracing schemes. These algorithms are referred to as the FT1, FT2 and FT3 algorithms respectively. (Fiat and Tassa, 1999) establish that the theoretical minimum bound on the number of variants required to trace any number of traitors in the dynamic model is $p + 1$. Two of the proposed algorithms use $p + 1$ variants, but have an exponential convergence time. The other algorithm has a convergence time polynomial in p, but uses $2p + 1$ variants.

(Berkman *et al*, 2000) propose an algorithm that traces all p traitors in $O(p^3 \log n)$ rounds using $p + 1$ variants. This is referred to as the *clique* algorithm. The clique algorithm is used to construct an optimal algorithm that is able to trace all traitors in polynomial time using the minimal number of variants. (Berkman *et al*, 2000) show that by increasing the number of variants used in the optimal algorithm, a measurable trade-off in convergence time can be obtained.

4 RANDOMISED DYNAMIC TRACING

All dynamic schemes created so far are deterministic. This means that a pirate is able to run through the tracing algorithm in advance and totally deduce the marking strategy that will be used. While this does not prevent a pirate from being caught, this situation is undesirable, as it allows a pirate to force the algorithm to run at its worst case complexity through strategic use of feedback.

This can be best illustrated using the Quicksort algorithm (Hoare, 1961). Given a set of data to sort, the Quicksort algorithm chooses an element from the array (referred to as the *pivot*). The data is partitioned either side of the pivot such that elements less than the pivot are below it, and those greater are above it. The algorithm then recursively sorts the partitions. Quicksort is deterministic and given the same set of input, will execute exactly the same way each time.

The performance of the Quicksort algorithm depends on the pivot point chosen. On average, this algorithm runs in $O(n \log n)$ steps where n is the number of items to be sorted. However, if the set of data is already nearly sorted, performance degrades to $O(n^2)$ steps.

Consider the case where an adversary of the Quicksort algorithm is permitted to order the array. The adversary can in each instance force the Quicksort algorithm to run in its worst case complexity by ensuring that the array is reverse sorted. By doing so, the array will always be split into two unbalanced sets, a set of one element and a set consisting of the remainder of the array. (Further attacks of this nature on Quicksort are described in (McIlroy, 1999).)

To protect against this type of attack, random choices can be introduced into the sorting process. For example, the array is randomly re-ordered prior to sorting, and/or the pivot point is randomly chosen. Under these conditions, the adversary does not benefit from pre-ordering the input data. Random decisions ensure that the algorithm runs at its average case complexity, regardless of the initial order of input data.

Dynamic traitor tracing schemes are no different to the Quicksort algorithm. The traitor can influence the algorithm to perform at its worst case complexity, thereby delaying his/her tracing. Likewise random decisions can also be used in dynamic traitor tracing to ensure average case complexity.

This section presents several amended dynamic schemes from literature. Each algorithm has been altered to include random decisions during the tracing process. This ultimately removes any advantage a traitor has in terms of the aforementioned attack.

4.1 FT1 Algorithm

The FT1 Algorithm uses the optimal number of variants (i.e., $p + 1$). Traitors are traced by marking designated combinations of users based on the current known traitor count t. Either all traitors will be isolated when the correct combination of users is selected, or there are more than the current number of known traitors in the system. The latter scenario allows us to increment the traitor count by one and repeat the same process with an extra variant. Convergence time is O($\binom{n}{p}$ + 2× $\sum_{t=0}^{p-1} \binom{n}{t}$).

Algorithm 4.1 - FT1

1. Set $t = 0$
2. Repeat forever:
   (a) For all selections of t users out of U, $w_1, ..., w_t \subset U$, produce $t + 1$ distinct variants of the current segment and transmit the ith variant to w_i, $1 \leq i \leq t$, and

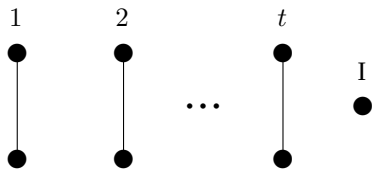

Figure 2: FT2 Algorithm using graph notation.

the $(t+1)th$ variant to all other users, until the pirate transmits a recognised variant.

(b) If the pirate ever transmits variant i for some $i \leq t$, *disconnect* the single user w_i and decrement t by one. Otherwise, increment t by one.

In step 2(a), users are added to the suspect group (i.e., those whom are issued with a unique variant) according to an ordered sequence. An adversary can ensure worst case timing by only using traitors that are towards the end of the sequence. We propose two methods that can be used to protect this algorithm from this type of attack. Firstly, the set of users U in step 1 can be randomised to prevent a pirate from initially ordering the data (i.e., placing traitors at the end). Secondly, the selection of the next user added to the suspect set (step 2(a)) can also be randomised. This ensures that it is no safer embed traitors at the end of the group, than it is anywhere else in the group.

4.2 FT2 Algorithm

The FT2 Algorithm uses $2p+1$ variants and locates all p traitors in $O(p \log n)$ rounds. The algorithm (shown below) uses a recursive approach by continuously dividing and recombining groups of users. This scheme introduces a 'trusted' set of users (denoted by I). Initially, all users are considered innocent and are part of I. It is only when piracy is detected from this set that all its members become suspects, allowing us to increase the known traitor count. When piracy is detected from any particular group (I inclusive) it is evenly divided and the resulting two subgroups are marked as a pair (L_i, R_i) forming part of the partition P of U. The partner set of the split group is added to I, which in a sense means that the algorithm also searches for innocent users. Although this greatly improves convergence time, it comes at the cost of increasing the number of variants.

Algorithm 4.2 - FT2

1. Set $t = 0$, $I = U$, $P = I$.
2. Repeat forever:
   (a) Transmit a different variant for every non-empty set of users $S \in P$.
   (b) If the pirate transmits a variant v of the current segment then:
   - If v is associated with I, increment t by one, split I into two equal sized subsets, L_i and R_i, add those sets to P and set $I = \emptyset$.
   - If v is associated with one of the sets L_i, $1 \leq i \leq t$, do as follows:
   i. Add the elements in R_i to the set I.
   ii. If L_i is a singleton set, *disconnect* the single traitor in L_i from the user set U, decrement t by one, remove R_i and L_i from P and renumber the remaining R_i and L_i sets in P.
   iii. Otherwise (L_i is not a singleton set), split L_i into two equal sized sets, giving new sets L_i and R_i.
   - If v is associated with one of the sets R_i, $1 \leq i \leq t$, do as above while switching the roles of R_i and L_i.

Graph notation can also be used to represent the data structures of dynamic algorithms. For example, the tracing process of the FT2 algorithm can be depicted by an undirected graph $G = (V, E)$. Each vertex represents a set of users and an edge (X, Y) connecting two vertices indicates that the subset $X \cup Y$ contains a traitor (see Figure 2). The allocation of variants effectively colours each vertex of the graph. When the pirate transmits a colour given to a vertex, the vertex is split and an edge is added. If the vertex only contains one user (referred to as a singleton vertex), then that user is a traitor and is disconnected.

As in the FT1 algorithm, users are in an ordered sequence and it is possible to determine precisely where a given set will be split. An adversary can ensure that s/he selects the traitors in a manner that results in the most number of splits. We recommend that to prevent this from occurring, the set of users U in step 1 should be randomised to prevent a pirate from initially ordering the data. Next, the division point for a set must not occur precisely in the middle of the ordered sequence. Instead, we endorse that the members of the set must be randomly reordered before splitting. This ensures that the pirate cannot determine where it is safe to embed traitors in the sequence.

4.3 FT3 Algorithm

The FT3 Algorithm effectively combines the previous schemes. As this algorithm uses $p+1$ variants, at each step either a group is split, or the FT1 algorithm is being run on an individual group. In order to do this, the algorithm keeps a lower bound on both the current known number of traitors, k, and also t, the

number of subsets of users $\{L_i, R_i\}$ in the partition P of U. Each pair is known to contain at least one traitor. This algorithm makes progress at every stage but is still exponential in convergence time, requiring $O(3^p p \log n)$ rounds.

Algorithm 4.3 - FT3

1. Set $t = 0$, $k = 0$, $I = U$, $P = I$.
2. Repeat forever:
   (a) For every selection of $S_1, ..., S_k \subset P$ where $S_i \in L_i, R_i$, $1 \leq i \leq t$, and S_i, $t+1 \leq i \leq k$ are any other $k - t$ sets from P, produce $k + 1$ variants, σ_i, $1 \leq i \leq k + 1$. Transmit σ_i to S_i for all $1 \leq i \leq k$, while all remaining users get variant σ_{k+1}.
   (b) Assume that the pirate transmits at some step a variant σ_i that corresponds to a single set in P (when $K < 2t$) those are the variants σ_i where $1 \leq i \leq k$; when $k = 2t$, on the other hand, all variants correspond to a single set, since then also σ_{k+1} is transmitted to just one set). In that case:
      i. If σ_i corresponds to an L_i set, then that set must contain a traitor. In that case add the complementary set, R_i, to I and split L_i into two equal sized sets giving a new L_i, R_i pair. In this case neither t nor k changes but eventually, when the size of the incriminated set is one, *disconnect* the traitor in that set. When this happens, restart the loop after decrementing k and t by one.
      ii. If σ_i corresponds to an R_i set, we act similarly.
      iii. If σ_i corresponds to I, it allows us to increment t by one (and k as well, if k was equal to t), split I into a new L_t, R_t pair, set $I = \emptyset$ and restart the loop.
   (c) If $k < 2t$ and the pirate transmits σ_{k+1}, after completing the entire loop increment k by one and restart the loop.

To prevent a pre-ordering attack, we recommend that the set of users U in step 1 be randomised. As this algorithm is a hybrid of the previous algorithms, the randomising amendments for the FT1 and FT2 algorithms also apply to this algorithm.

4.4 Clique Algorithm

The clique algorithm uses fully connected subgraphs in the tracing procedure. The clique algorithm utilises the basic strategies of (Fiat and Tassa, 1999) and is employed in the construction of a polynomial algorithm. The clique algorithm itself is not optimal. It traces all traitors in $O(p^3 \log n)$ rounds, using $p + 1$ variants.

The data structure for this algorithm is referred to as a (t, k)-graph where $t \geq k \geq 0$. A graph $G = (V, E)$ is a (t, k)-graph if G contains $t + k + 1$ vertices, including I. These vertices form a partition of U and all (except possibly I) are non-empty. Any edge $(X, Y) \in E$, implies that the subset $X \cup Y$ contains a traitor. The vertices in $V \backslash \{I\}$ are partitioned into k disjoint cliques $\{Q_1, ..., Q_k\}$, where $|Q_i| = t_i \geq 2$ for each $1 \leq i \leq k$. The number of vertices in a (t, k)-graph is at most $2t + 1$.

At each stage of the algorithm, the centre broadcasts a segment and observes the pirate's response. Based on the feedback, users are partitioned into disjoint cliques of suspected traitors (see Figure 3). As there may not be enough colours to give each vertex a unique colour, the algorithm pairs the k cliques. This forces the pirate to either transmit a colour given to only one vertex, or to disclose an edge given to a pair of cliques.

A response from a single vertex allows us to split the vertex creating a new clique of size two. Alternately, disclosed edges are added to the graph in order to eventually merge two cliques together. When a merging of cliques occurs, the users who appear to be the most innocent at this stage of the algorithm (least connected vertex in the new larger clique) are placed in I. The algorithm continues in this fashion until G only contains one clique of size $t + 1$. This then allows us to give each vertex a distinct variant, limiting the pirate to disclosing only one particular vertex as its response.

Algorithm 4.4 - Clique

Start with a $(0, 0)$-graph $G = (V, E)$ with $I = U$, $V = I$, and $E = \emptyset$. Repeat the following phases:

Phase 1: Distributing the colours to the vertices of G

Allocate $t_i - 1$ colours to each clique Q_i for a total of t colours. The last colour is allocated to I. We now describe how to distribute these allocated colours in each zone and block of G.

1. Colour each block in zone Z_1 separately, using the colours allocated to the pair of cliques in it: Let B be a block in Z_1 and let Q_i and Q_j be the two cliques in B. By the definition of the blocks in zone Z_1, there exist four distinct vertices X_1, $X_2 \in Q_i$ and Y_1, $Y_2 \in Q_j$ such that (X_1, Y_1), $(X_2, Y_2) \notin E$. Colour $X_1 \cup Y_1$ with one colour and $X_2 \cup Y_2$ with another colour. Colour the remaining $t_i - t_j - 4$ vertices of Q_i, Q_j using the remaining $t_i - t_j - 4$ colours allocated to those two cliques, one colour per vertex.
2. If Z_2 contains only the vertex I, then colour I using the colour allocated to it. Otherwise, Z_2 contains also a clique Q_i and there exists a vertex $X \in Q_i$, such that $(X, I) \notin E$. Use one colour for $X \cup I$, and colour the remaining $t_i - 1$ vertices of Q_i using $t_i - 1$ colours, one colour per index. (If I is empty, all vertices of Q_i are coloured with distinct colours.) Therefore, t_i colours are used, which is the total number of colours allocated to Q_i and I.

Phase 2: Reorganising the graph following the pirate's response

1. If the pirate broadcasts a colour given to only one vertex X: Remove X and the edges incident with it from G. Split X into two new vertices X_1, X_2 of equal size (or

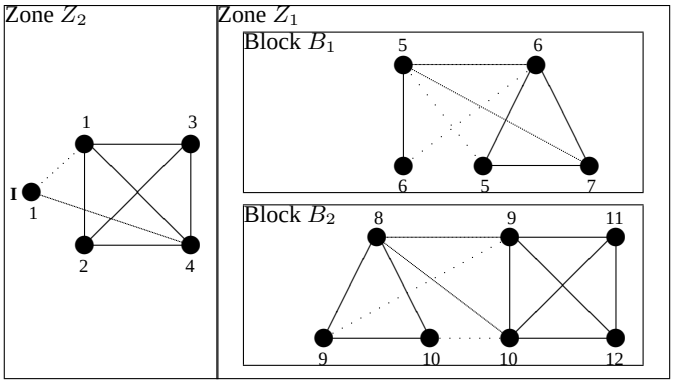

Figure 3: Clique Algorithm.

differing in size by one, if $|X|$ is odd), and add the edge (X_1, X_2).

(a) $X = I$: Define a new clique Q consisting of the two vertices X_1, X_2. Set $I = \emptyset$, $t = t+1$, and $k = k+1$. Incorporate the clique Q into the zones (see step 3).

(b) Otherwise: Let Q_i be the clique to which X belongs. Set $Q_i = Q_i \backslash X$.

i. If $|Q|_i = 1$ then remove all edges incident with the vertex remaining in Q_i, remove this vertex from Q_i and add it to I. Place X_1 and X_2 into Q_i. The clique Q_i now consists of the two vertices X_1, X_2.

ii. Otherwise, Q_i is still a legal clique. In this case the two sets X_1, X_2 will form a new clique Q. Set $k = k+1$ and incorporate the new clique Q into the zones (see step 3).

2. If the pirate broadcasts a colour given to a pair of vertices X, Y: Add the edge (X, Y) to G, and:

(a) If this edge connects vertices belonging to a pair of cliques Q_i, Q_j in some block B, and after adding this edge, the block B contains a large clique Q of size $t_i + t_j - 1$: Let Z be the remaining vertex in B that does not belong to this large clique (i.e., $Z \notin Q$). Remove Z and all edges incident with it from its clique, and add Z to I. Furthermore, set $k = k - 1$, remove the cliques Q_i, Q_j and their block B. Incorporate the new clique Q into the zones (in step 3).

(b) If this edge connects I and a vertex of the clique Q_i in zone Z_2, and after adding this edge, Q_i and I form a clique on $t_i + 1$ vertices: Add I as a vertex to Q_i and create a new special vertex $I = \emptyset$. Set $t = t+1$ (since t_i was increased by 1).

3. If a new clique Q was created during one of the above steps, reorganise the zones as follows to incorporate Q: If zone Z_2 contains only I, place Q in zone Z_2. Otherwise, zone Z_2 contains a clique R in addition to I. In this case, remove all edges incident with I, remove R from Z_2, pair it with the new clique Q, and create in zone Z_1 a new block containing the cliques Q and R.

We suggest the following randomising amendments to this algorithm. As usual, the set of users U in step 1 can be randomised to prevent a pirate from initially ordering the data. Since this algorithm utilises the approach of FT2, the random amendments for FT2 also apply to the clique algorithm. That is, before a vertex is split, the order of the users should be shuffled so that a pirate does not know which of the new vertices they will end up in.

Furthermore, during the colouring stage, the algorithm pairs vertices that do not have an edge and assigns them the same colour. In a sense the algorithm is probing for traitors. This process occurs in a deterministic manner and therefore the pirate gains some information about the colouring sequence. We advocate that this step be performed randomly so that the algorithm spontaneously selects the next pair of vertices for probing.

5 RESULTS

The algorithms described in this paper were implemented in C++. A series of simulations was performed to gauge the effectiveness of the proposed security enhancements. A pirate was given the task of engaging in piracy against a content provider within the scope of the game described in Section 3. The pirate had complete knowledge of the tracing algorithms and had the ability to run through the execution sequence of the algorithm in advance.

In order to describe our results, we must first examine the complexity of dynamic traitor tracing schemes. Table 1 lists the algorithms and their associated complexities. The convergence time for a dynamic algorithm is a function of the number of users, n, and the number of traitors, p, in the system. The number of variants used depends on the number of traitors p.

Clearly the FT2 algorithm achieves the best convergence time. However, this is at the expense of an

Table 1: Worst case bound on the dynamic algorithms and the performance of the deterministic algorithms vs the randomised algorithms when run on simulated data.

Scheme	Convergence Time	Number of Variants	Deterministic	Randomised
FT1 Algorithm	$\binom{n}{p} + 2 \times \sum_{t=0}^{p-1} \binom{n}{t}$	$p+1$	100%	4.1%
FT2 Algorithm	$O(p \log n)$	$2p+1$	100%	4.7%
FT3 Algorithm	$O(3^p p \log n)$	$p+1$	100%	3.2%
Clique Algorithm	$O(p^3 \log n)$	$p+1$	100%	4.5%

increased number of variants. The FT1 algorithm is exponential and therefore not practical. Neither is the FT3 algorithm. The clique algorithm uses $p+1$ variants and is an improvement over FT1 and FT3. (Berkman *et al*, 2000) use the clique algorithm to create an 'optimal' algorithm that runs in polynomial time with the minimum number of variants. However, this algorithm is extremely complicated and is outside the scope of this paper. We believe that while it is desirable to use the optimal number of variants, the results of the FT2 algorithm seem to indicate that it may be more practical to use a slightly higher number of variants in the interests of improving convergence time.

In order to test the average efficiency of our randomised amendments, both deterministic and randomised algorithms were implemented. Each algorithm was run 250 times and the effectiveness of the pirate in terms of the convergence time was recorded. (see right side of Table 1.) Our results show that the pirate consistently forced each deterministic algorithm to run at its worst case bound for 100% of the tests. However, the randomised algorithms only run in their worst case bound on average for 4.125% of the tests. Clearly this indicates that there is a definitive security advantage in performing simple randomisation tasks in dynamic traitor tracing schemes.

We also found that a reduction in overhead for dynamic traitor tracing schemes could be attained via the following means:

1. Keys do not necessarily have to be changed between segments for all users. Key updates are really only required for the users who are in a set which is split. Using this approach the FT2 algorithm, which typically uses $2p+1$ variants, can achieve $O(np)$ individual transmissions for all segments.

2. Not all of a session needs to be protected. Threshold traitor tracing schemes (see (Naor and Pinkas, 1998)) only require a small percentage of the movie to be protected to enable tracing. It is assumed that if a pirate copy misses even part of the session, then is not very valuable.

6 RESULTS/CONCLUSIONS

Dynamic traitor tracing schemes are used to trace the source of piracy in broadcast environments such as cable TV. Dynamic schemes divide content into a series of watermarked segments that are then broadcast. The broadcast provider can adapt the watermarks according to the pirate's response and eventually trace him/her. As dynamic algorithms are deterministic, for a given set of inputs, the tracing algorithm will execute exactly the same way each time. An adversary can use this knowledge to ensure that the tracing algorithm is forced into executing at its worst case bound.

In this paper we review dynamic traitor tracing schemes and describe why determinism is a problem. Using the Quicksort algorithm as an example, we show how an adversary can order the input data and can cause a similar problem to that faced by a dynamic tracing algorithm. We also show how previous literature thwarted the Quicksort's problem by introducing random decisions at critical points in the algorithm.

We amend several existing dynamic tracing algorithms by incorporating randomised decisions. This eliminates any advantage an adversary has in terms of the aforementioned attack, as s/he no longer knows exactly how the tracing algorithm will execute. Simulations show that the randomised amendments were successfully able to force a dynamic algorithm to run at its average case complexity.

Existing dynamic traitor tracing schemes strive to achieve the best convergence time using the minimal number of variants. While $p+1$ is the theoretical minimum, it may be possible to design a quicker algorithm if these constraints are somewhat relaxed. Although (Berkman *et al*, 2000) provide bounds for gains in convergence time through increasing the number of variants, this is done in terms of their somewhat complicated optimal algorithm. Future work involves constructing a conceptually simple tracing algorithm that achieves a better convergence time than existing tracing algorithms. Ideally the algorithm will aspire to use a number of variants that is between $p+1$ and $2p+1$.

REFERENCES

Berkman, O., Parnas, M. and Sgall, J. (2000). Efficient Dynamic Traitor Tracing, in *11th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2000), 586-595.

Boneh, D. and Shaw, J. (1995). Collusion-Secure Fingerprinting for Digital Data, IEEE *Transactions of Information Theory*, vol. 44, no. 5, 452-465.

Chor, B., Fiat, A. and Naor, M. (1994). Tracing Traitors, in *Advances in Cryptology - Proceedings of CRYPTO 1994*, vol. 839 of *Lecture Notes in Computer Science*, 257-270.

Daemen, J. and Rijmen, V. (1998). AES Proposal: Rijndael, in *First Advanced Encryption Standard (AES) Conference*, Ventura, CA.

Fiat, A. and Tassa, T. (1999). Dynamic Traitor Tracing, in *Advances in Cryptology - Proceedings of CRYPTO 1999*, vol. 773 of *Lecture Notes in Computer Science*, 354-371.

Hoare, C. A. R. (1961). ACM Algorithm 64: Quicksort, in *Communications of the ACM*, 4(7):321.

McIlroy, M. (1999). A Killer Adversary for Quicksort, in *Software, Practice and Experience*, vol. 29, 1-4.

Naor, M. and Pinkas, B. (1998). Threshold Traitor Tracing, in *Advances in Cryptology - Proceedings of CRYPTO 1998*, vol. 1462 of *Lecture Notes in Computer Science*, 502-517.

Pfitzmann, B. (1996). Trials of Traced Traitors, Information Hiding, vol. 1174 of *Lecture Notes in Computer Science*, 49-64.

Safavi-Naini, R. and Wang, Y. (2000). Sequential Traitor Tracing, in *Advances in Cryptology - Proceedings of CRYPTO 2000*, vol. 1880 of *Lecture Notes in Computer Science*, 316-332.

Trevathan, J., Ghodosi, H. and Litow, B. (2003). Overview of Traitor Tracing Schemes, in *Communications of the CCISA*, Selected Topics of Cryptography and Information Security, Taiwan, Volume 9, Number 4, 51-63.