ITERATIVE SINGLE TARDOS DECODER WITH CONTROLLED PROBABILITY OF FALSE POSITIVE

Peter Meerwald and Teddy Furon

INRIA Rennes Bretagne Atlantique
Campus de Beaulieu, Rennes, France

ABSTRACT

We propose a blind iterative single Tardos decoder for traitor tracing designed to catch as many colluders as possible while controlling the probability of accusing an innocent. The key idea is that users accused in the previous iterations are used as side-information to build a more discriminative test. A fast implementation supporting millions of users is presented and compared with two recent fingerprinting codes.

Index Terms— Traitor tracing, fingerprinting, Tardos code, transactional watermarking

1. INTRODUCTION

This paper deals with traitor tracing which is also known as active fingerprinting, content serialization, user forensics or transactional watermarking. A typical application is, for instance, video-on-demand: a video portal distributes personal copies of the same content to n buyers. Some are dishonest users illegally redistributing pirated copies. The right holders are interested in identifying these dishonest users. For this purpose, a versioning process embeds a unique user identifier (so-called codeword) consisting on a sequence of m symbols in each video content thanks to a watermarking technique, thus producing n different (although perceptually similar) copies. This allows tracing back which user has illegally redistributed his copy. However, there might be a collusion of c dishonest users, $c > 1$. This collusion mixes their copies in order to forge a pirated content which contains none of their identifiers but a mixture of them. Traitor tracing codes are designed such that its decoding algorithm recovers some of these c identifiers from their mixture.

Most academic articles so far have studied traitor tracing codes from a theoretical point of view: For the following requirements (n users, c colluders, P_{fp} global probability of false positive), what is the shortest codelength m? and how to built such a code? The traitor tracing code invented by Gabor Tardos in 2003 [1] becomes more and more popular. It is provably good in the sense that m has the optimum scaling in $O(c^2 \log n/P_{\mathsf{fp}})$. The reader will find a pedagogical presentation of this code in [2].

Funded by national project MEDIEVALS ANR-07-AM-005.

Our paper doesn't pertain to this trend of academic works. We only focus on the accusation part of a Tardos code and our setup is different: a pirated copy had been found, and m bits have been decoded (depending on the bit rate of the watermarking technique and the length of the content). This content has been purchased by n users. Our accusation algorithm has no prior knowledge of the collusion process (i.e. its size c, its mixing process). The only assumption is that the number of colluders c is less than c_{max}. It tries to accuse a maximum of colluders even if m is less than the required length theoretically derived in the academics articles.

The first priority is to strictly control the probability of false positive P_{fp}, which is the probability of accusing at least an innocent. This task is not easy because a very small P_{fp} is of course required. A rare event analysis help us estimating this probability. Addressing a huge amount of users n is the second priority. This implies to have a very fast accusation process that scales well with n. Therefore, we restrict ourself to a single decoder which computes a score per user and accuses top ranked individuals. Some implementation tricks speed up the scores computation. The last priority is to catch as many colluders as possible as mentioned in overview article [3]. In an iterative process, we compute new and more discriminative scores by conditioning on the suspects we accused at the previous iterations.

2. SETUP

We now present motivations for a two layers scheme combining a traitor tracing code and a watermarking technique.

2.1. Industrial requirements

Traitor tracing doesn't prevent the act of piracy, it is a dissuasive weapon which is a second line of defense when the access control mechanism (i.e. the encryption scheme) has been broken or when the pirates capture the rendering of the content. This technique is often reserved for 'premium' contents like extremely recent movies for Home Cinema in super high quality sold at an expensive price. Watermarking is done giving consideration to the complex human audio visual systems in order to preserve the quality of these premium contents.

This takes a lot of time, and for industrial deployment, it is unacceptable to watermark a content on an user-by-user basis when n is large[1].

The trick indeed is to divide the content in m blocks (i.e. scenes of a movie) and to watermark off-line each block once embedding symbol '0', once embedding symbol '1'. The blocks are usually pre-encrypted for access control purposes as well. The drawback is that the content takes twice the storage size, but, versioning is made easy now: the server sequentially sends pre-watermarked pre-encrypted blocks containing the symbols of the unique user identifier. If $\mathbf{x}_j = (x_j(1), x_j(2), \ldots, x_j(m))$ is the codeword of user j, then the server sends the first block of the content watermarked with symbol $x_j(1)$, then the second block with symbol $x_j(2)$, etc. This trick is used in the traitor tracing mechanism implemented in AACS, the standard for Blu-ray disc protection. There are two design issues: the first one concerns the traitor tracing code (generation of the codewords and the accusation algorithm), and the second one is the watermarking technique (embedding and decoding processes). Again, this paper details only our work regarding the accusation algorithm.

2.2. Codeword generation

We use the famous Tardos code construction. At initialization, a sequence $\mathbf{p} = (p(1), \ldots, p(m))$ is randomly drawn. Each component is independent and identically distributed with law $f(p) : [0,1] \to \mathbb{R}^+$, $p \to (\pi^2 p(1-p))^{-1/2}$.When user j is buying the content, his codeword $\mathbf{x}_j$ is generated as follows. The binary symbols are independently drawn such that $\mathbb{P}(x_j(i) = 1) = p(i)$. Let the matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ of size $n \times m$ represent the sequences for n users.

2.3. Collusion

In this paper, we assume that the colluders know the partition of the content used by the versioning process. Therefore, they can create a pirated copy by swapping their blocks. This has the effect of shuffling the bits embedded in the pirated copy. Denote $\mathbf{y}$ the sequence extracted by the watermark decoder: $\forall i \in [m], y(i) \in \{x_{j_1}(i), \ldots, x_{j_c}(i)\}$ where $\mathcal{C} = \{j_1, \ldots, j_c\}$ denotes the indices of the colluders. This model of collusion is well-known as the *marking assumption*.

In traitor tracing literature, this collusion process is usually modeled by a vector $\boldsymbol{\theta}$, with $\theta(\sigma) = \mathbb{P}(y_i = 1 | \sum_{j \in \mathcal{C}} x_j(i) = \sigma)$. Basically, to decide which block (and therefore symbol) the colluders paste in the pirated copy, they flip a coin whose bias only depends on the number of '1' they have out of c. The marking assumption is enforced by setting $\theta(0) = 0$ (if they do not have any '1', they are forced to put a '0') and $\theta(c) = 1$ (if they all have a '1', they are obliged to put a '1'). The remaining $c-1$ coefficients live in $[0,1]^{c-1}$ and describe the collusion process. For instance, a majority vote is sketched by $\theta(\sigma) = 0$ if $\sigma < c/2$, $\theta(c/2) = 1/2$ (if c is even), and 1 otherwise. The interleaving attack is when the colluders randomly paste one of their symbols in the pirated copy. Statistically this implies that $\theta(\sigma) = \sigma/c$.

[1] However, for a small n like the size of the MPAA Oscar jury, this is often done like that.

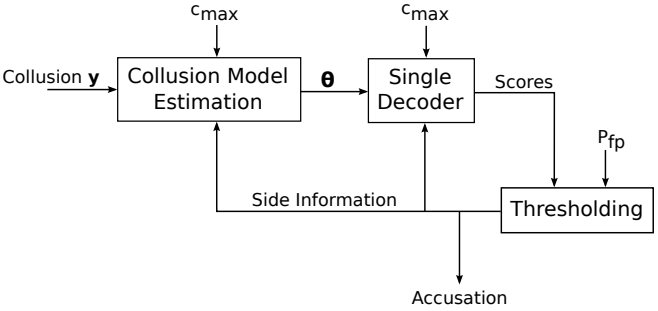

Fig. 1. Overview of the iterative, side-informed decoder.

3. ACCUSATION PROCESS

In this section, we assume that the accusation process knows the size of the collusion and the collusion process $\boldsymbol{\theta}$. We call this accusation process the optimal decoder. These assumptions are relaxed in the next sections.

3.1. Optimal single decoder

The best accusation is then based on the following log-likelihood ratio [4, Sec. 3.1]:

$$s_j = \sum_{i=1}^{m} \log \frac{\mathbb{P}(y(i)|x_j(i), p(i))}{\mathbb{P}(y(i)|p(i))}, \tag{1}$$

with

$$\mathbb{P}(y(i) = 1) = \textstyle\sum_{\sigma=0}^{c} \theta(\sigma) \mathbb{P}(\sigma|p(i)), \tag{2}$$

$$\mathbb{P}(\sigma|p(i)) = \tbinom{c}{\sigma} p(i)^{\sigma} (1-p(i))^{c-\sigma}, \tag{3}$$

$$\mathbb{P}(y(i) = 1|x_j(i)) = \textstyle\sum_{\sigma=x_j(i)}^{c-1+x_j(i)} \theta(\sigma) \mathbb{P}(\sigma|x_j(i), p(i)), \tag{4}$$

$$\mathbb{P}(\sigma|x, p(i)) = \tbinom{c-1}{\sigma-x} p(i)^{\sigma-x} (1-p(i))^{c-1-\sigma+x}. \tag{5}$$

Colluders are expected to have higher scores than innocent users. Three cases are possible: (i) m is big enough with respect to c and n and the c colluders' scores are ranked first, (ii) m is almost in order of magnitude to cope with such c and n and not all the colluders are ranked first, (iii) m is too short and one innocent has the biggest score.

To take a decision, we translate score s_j into the probability π_j that user j is innocent [4, Sec. 3.2.2]:

$$\pi_j = \frac{1}{1 + c(n-c)^{-1} \cdot \exp(s_j)}. \tag{6}$$

Denote ϵ as the probability of accusing a given innocent user whereas P_{fp} is the probability of accusing at least one innocent over the whole set of innocent users. Thus, $P_{\mathsf{fp}} =$

$1-(1-\epsilon)^{n-c} \approx \epsilon \cdot n$. Therefore, we only accuse users s.t. $\pi_j < \epsilon$. This is equivalent to accusing users whose score are higher than threshold τ with:

$$\tau = \log\left((\epsilon^{-1}-1)(nc^{-1}-1)\right). \quad (7)$$

If $P_{\mathsf{fp}} = 10^{-4}$ and $n = 10^6$, this amounts to accusing only when we are very sure, i.e. π_j is as low as 10^{-10}. In the first case (i) above-mentioned, all the colluders have such a low probability. In the third case (iii), no user j fulfilled this constraint and the accusation fails (false negative).

3.2. Iterative single decoder

In the second case (ii), at least one colluder is caught. How to make him/her denounce the remaining accomplices? This idea is simply translated into hypothesis testing by conditioning the test by this new side information. This is the main idea of the iterative decoder we propose in this paper. The first iteration is the single decoder as detailed above. Now suppose that from the previous iterations, we accused $k < c$ users. Denote $\mathcal{X}_{\mathsf{SI}}$ this set and $\rho_i = \sum_{j\in\mathcal{X}_{\mathsf{SI}}} x_j(i)$. Conditioning the test by this side information changes equations (2) - (5) to:

$$\begin{aligned}
(2) \quad &\leftarrow \quad \sum_{\sigma=\rho_i}^{c-k+\rho_i} \theta(\sigma)\mathbb{P}(\sigma|p(i)),\\
(3) \quad &\leftarrow \quad \tbinom{c-k}{\sigma-\rho_i}\, p(i)^{\sigma-\rho_i}(1-p(i))^{c-k-\sigma+\rho_i},\\
(4) \quad &\leftarrow \quad \sum_{\sigma=x_j(i)+\rho_i}^{c-k-1+x_j(i)+\rho_i} \theta(\sigma)\mathbb{P}(\sigma|x_j(i),p(i)),\\
(5) \quad &\leftarrow \quad \tbinom{c-k-1}{\sigma-\rho_i-x}\, p(i)^{\sigma-\rho_i-x}(1-p(i))^{c-k-1-\sigma-\rho_i+1}.
\end{aligned}$$

We accuse and include in the set $\mathcal{X}_{\mathsf{SI}}$ all users whose new score is higher than τ. Algorithm 1 and Figure 1 summaries our iterative single Tardos decoding algorithm.

Algorithm 1 Iterative Single Tardos Decoder.

Require: $\mathbf{y}$, $\mathbf{X}$, $\mathbf{p}$, c_{max}, P_{fp}
$\mathcal{X} \leftarrow \{j | 1 \le j \le n\}$, $\mathcal{X}_{\mathsf{SI}} \leftarrow \{\}$
$\epsilon \leftarrow n^{-1}P_{\mathsf{fp}}$
repeat
  $\hat{\boldsymbol{\theta}} \leftarrow estimate(\mathbf{y}, \mathbf{p}, \mathcal{X}_{\mathsf{SI}}, c_{\mathsf{max}})$
  $\mathbf{W} \leftarrow weights(\mathbf{y}, \mathbf{p}, \hat{\boldsymbol{\theta}}, \mathcal{X}_{\mathsf{SI}}, c_{\mathsf{max}})$
  $\mathbf{s} \leftarrow scores(\mathcal{X} \setminus \mathcal{X}_{\mathsf{SI}}, \mathbf{X}, \mathbf{W})$
  $\tau \leftarrow threshold(\epsilon)$
  $\mathcal{A} \leftarrow \{j | s_j > \tau\}$
  $\mathcal{X}_{\mathsf{SI}} \leftarrow \mathcal{X}_{\mathsf{SI}} \cup \mathcal{A}$
until $\mathcal{A} = \emptyset$ **or** $|\mathcal{X}_{\mathsf{SI}}| \ge c_{\mathsf{max}}$
return $\mathcal{X}_{\mathsf{SI}}$

4. DETAILS OF IMPLEMENTATION

In reality, the accusation algorithm is blind in the sense that it knows neither the collusion size nor its process. This prevents us from using the optimal score (1) and its translation (6) into probability π_j. Moreover, the single decoder is exhaustive with complexity $O(n)$ because it has to compute scores for all users. Because n can be large and scores must be computed at each iterations, we really need a fast implementation.

4.1. Collusion Model Estimation

For an estimated collusion size $\hat{c}$, the estimation of the collusion process $\boldsymbol{\theta}$ is possible from the observation of $\mathbf{y}$. We do this by maximizing the log-likelihood ratio:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in[0,1]^{\hat{c}+1} s.t. \theta(0)=0, \theta(\hat{c})=1}{\arg\max} \log \mathbb{P}(\mathbf{y}|\mathbf{p},\boldsymbol{\theta}), \quad (8)$$

with $\mathbb{P}(\mathbf{y}|\mathbf{p},\boldsymbol{\theta}) = \prod_{i=1}^{m} \mathbb{P}(y(i)|p(i))$ (see expression in (2)). However, due to a lack of identifiability, this approach cannot estimate c, but only $\hat{\boldsymbol{\theta}}$ for a given $\hat{c}$. We have heuristically noticed that efficient scores are obtained when plugging estimation $\hat{\boldsymbol{\theta}}$ in (1) provided that $\hat{c}$ is bigger than the real c. Therefore, we impose $\hat{c} = c_{\mathsf{max}}$. This describes the estimation algorithm of the first iteration. In the following iterations, side information is beneficial not only for computing more efficient scores, but also for obtaining a more accurate estimation $\boldsymbol{\theta}$ by using expressions of 3.2. This iterative blind decoder is sketched in Fig. 1.

4.2. Thresholding

The second issue is the translation of the scores into probabilities. Since $\hat{\boldsymbol{\theta}}$ is a priori different than $\boldsymbol{\theta}$, (1) is not the true log-likelihood ratio that user j is guilty and (6) is not the true probability of being an innocent.

However, at a given iteration, all the scores are computed in the same way. The idea is to generate new codewords based on the sequence $\mathbf{p}$ and to compute their scores. We are then sure to observe scores of innocents since these codewords could not have been used to forge $\mathbf{y}$. With a Monte Carlo simulation, we can estimate the probability that the score of an innocent is bigger than threshold τ, or the other way around, the threshold τ such that the probability of being an innocent is below ϵ. Colluding traitors with scores bigger than τ will be accused and integrated in the side information. This approach works whatever the way scores are computed.

The only problem is that a large n implies a very low probability $\epsilon = n^{-1}P_{\mathsf{fp}}$, and a Monte Carlo simulation is then bad at estimating accurately threshold τ. This is the reason why we implemented an estimator based on rare event analysis [5].

4.3. Fast Score Computation

Essentially, the runtime of the score computation – and, for large number of users, the time comsumption for the iterative decoder – depends on the memory throughout when accessing

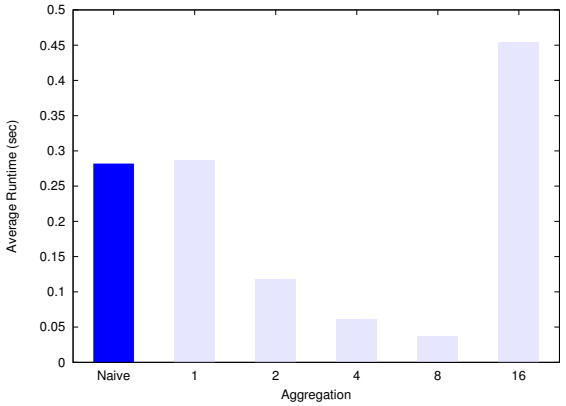

Fig. 2. Average runtime for score computation of $n = 10^5$ users, code length $m = 2048$, with different aggregation factors a.

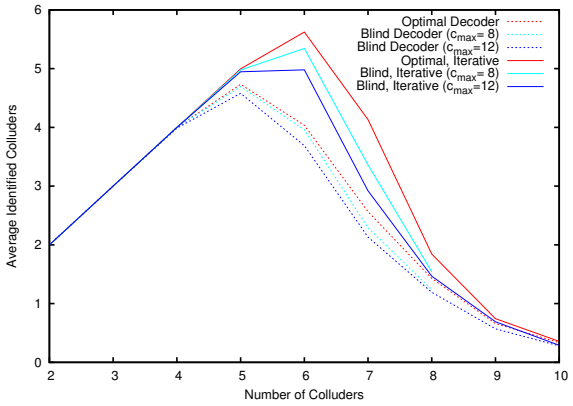

Fig. 3. Identified traitors performing interleaving collusion ($n = 10^5, m = 2048$, $P_{\mathsf{fp}} = 10^{-4}$) comparing the optimal and blind decoders with different $c_{\max}$.

the codewords of the individual users. We highlight two characteristics of the implementation: precomputation of weights and data aggregation/table lookup.

Computation of an individual's score s_j can be written as $s_j = \sum_{i=1}^{m} W[x_j(i)](i)$ where $\mathbf{W}$ is a $2 \times m$ matrix containing the precomputed log-likelihood ratios:

$$W[0](i) = \log \frac{\mathbb{P}(y(i)|p(i))}{\mathbb{P}(y(i)|0, p(i))},$$
$$W[1](i) = \log \frac{\mathbb{P}(y(i)|p(i))}{\mathbb{P}(y(i)|1, p(i))}.$$

In order to store the binary elements of an user's codeword in an efficient way, b bits are grouped together into an unsigned integer data type native to the processor, e.g. $b = 32$ or $b = 64$ typically. Instead of sequentially processing the bits, chunks of $a \leq b$ bits, e.g. $a = 4$ or $a = 8$, can be processed in parallel using a table lookup. b and m should be evenly divisible by a for practical purposes. The weight matrix $\mathbf{W}$ is turned into an aggregated weight matrix $\mathbf{W}'$ of size $2^a \times \lceil m/a \rceil$ with elements

$$W'[q](i') := \sum_{l=1}^{a} W[\mathrm{bit}(q, l)](a(i'-1)+l) \qquad (9)$$

where $1 \leq i' \leq \lceil m/a \rceil$, $q \in \{0,1\}^a$, and $\mathrm{bit}(q, l)$ denotes the l-th bit of value q. Hence, summation of weights can be partially replaced by a table lookup of a consecutive codeword bits in the precomputed, accumulated weight matrix $\mathbf{W}'$. Clearly, there is a limit on the aggregation factor a depending on the processor's cache size which is illustrated in Fig. 2. Best performance is obtained for $a = 8$ according to our experiments on an Intel Core2 (E6700) processor. The column labelled *naive* refers to an implementation storing the elements $x_j(i)$ in individual bytes of memory.

5. EXPERIMENTAL RESULTS

The iterative single Tardos decoder is implemented in C++ and compiled using GNU `g++` version 4.4.5 on a x86 Ubuntu/Linux 10.10 system with `-O3 -march=native -fomit-frame-pointer -mfpmath=sse`. The estimation of $\boldsymbol{\theta}$ uses approximate vectorized single-precision floating point arithmetic and Shigeo Mitsunari's fast approximative $\log()$ function[2]; the remaining components are implemented with double-precision. Pseudo-random numbers are generated with the SIMD-oriented Fast Mersenne Twister (dSFMT)[3] [6].

We first assess the performance of the proposed blind decoders against the optimal decoder having knowledge of the number of colluders and the chosen collusion strategy. Fig. 3 plots the average number of identified colluders versus the true number of colluders performing an *interleaving* collusion attack. The code length is set to $m = 2048$ and the number of users is $n = 100\,000$. The false-positive rate is set to $P_{\mathsf{fp}} = 10^{-4}$. $1\,000$ experiments have been performed for each setting. As expected, the blind decoders which rely on $c_{\max}$ and $\hat{\boldsymbol{\theta}}$ are slightly less efficient in terms of traitor tracing ability than the optimal decoders. A value of $c_{\max}$ closer to the true number of colluders c involved in the collusion process results in better performance. We set $c_{\max}$ to 8 and 12, respectively, and vary $c \in \{2, 3, \ldots, 10\}$. In case $c > c_{\max}$, the false-negative rate P_{fn}, i.e. the probability that no accusation can be made, increased dramatically. Note that the estimate $\hat{\boldsymbol{\theta}}$ of the collusion process is very reliable and extremely close to the true $\boldsymbol{\theta}$ when $\hat{c} = c$, hence we omit plots where c is known and $\boldsymbol{\theta}$ estimated. Finally, we observe that the iterative

[2]Available from `http://homepage1.nifty.com/herumi/soft/fmath.html`, version of February 16, 2010.

[3]Available from `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/`, version 2.1

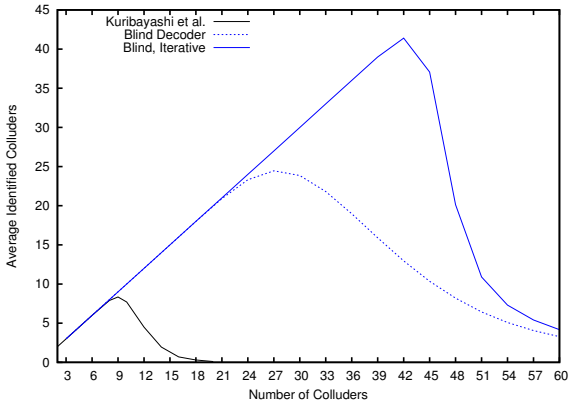

Fig. 4. Identified traitors performing majority collusion in the Kuribayashi setup ([7], $n = m = 10^4$, $P_{\mathsf{fp}} = 10^{-4}$); $c_{\mathsf{max}} = 80$.

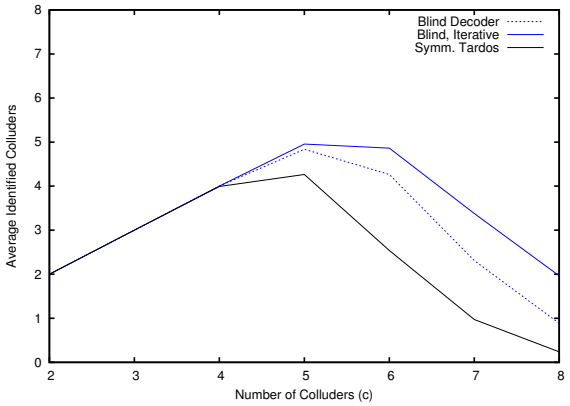

Fig. 5. Identified traitors performing *interleaving* collusion and AWGN attack ($\sigma^2 = 1$) in the Jourdas & Moulin setup ([8], $n \approx 3.3 \cdot 10^7$, $m = 7440$, $P_{\mathsf{fp}} = 10^{-3}$); $c_{\mathsf{max}} = 8$.

decoders (be it optimal or blind) can on average accuse more colluders than their non-iterated versions.

5.1. Decoding Comparison

We compare our iterative blind Tardos decoder with one of the few results reporting the number of detected colluders. Kuribayashi et al. [7] set the number of users $n = 10\,000$, the code length $m = 10\,000$; the collusion strategy is majority voting. The goal is to detect as many colluders as possible with a false-positive probability $P_{\mathsf{fp}} = 10^{-4}$.

Fig. 4 plots the average number of identified colluders versus the number of users c taking part in the collusion attack, $c \in \{2, 3, 6, \ldots, 60\}$. c_{max} has been set to 80, smaller values did improve the decoding performance only slightly. Experiments have been performed $1\,600$ times for each setting. We observe that the blind non-iterated decoder [4] outperforms the decoder of Kuribayashi et al. by a large margin. Our iterative blind decoder achieves to increase the average number of identified colluders. We omit detailed plots showing the experimental false-negative rate, i.e. the chance that no colluder can be accused. For both, [4] and our blind iterative decoders, no false-negatives occur for up to 48 colluders; for comparison, Kuribayashi et al. report false-negatives for 13 colluders and above.

Further, we provide results for a large-scale fingerprinting setup proposed by Jourdas and Moulin [8] involving $n = 33\,554\,432$ users and a code of length $m = 7\,440$. Codeword bits go through an antipodal modulation $(+1, -1)$; colluders *interleave* their modulated signals and add white Gaussian noise with unit variance. Jourdas and Moulin only consider $c = 5$ colluders and report an error probability $P_{\mathsf{e}} = P_{\mathsf{fp}} + P_{\mathsf{fn}} = 0.004$ – yet only a single traitor is accused. Fig. 5 shows the average number of identified colluders for our blind iterative decoder for $P_{\mathsf{fp}} = 10^{-3}$. Experiments have been performed $3\,000$ times for each setting, $c_{\mathsf{max}} = 8$. For $c = 5$, on average 4.83 colluders are accused at the end of the first round, while 4.95 of them are identified at the end of the iterative decoding. We observe $P_{\mathsf{fn}} = 0.0016$. While the P_{fn} is only slightly improved over [8], our blind iterative decoder allows to identify several more colluders on average which can be turned into additional evidence against the collaborating adversaries. We also compare against symmetric Tardos decoding for reference; soft decoding will likely improve the results and is subject to future work.

5.2. Runtime and Complexity Analysis

Fig. 6 shows the runtime of the blind iterative decoder in seconds averaged over 100 test runs for the Kuribayashi setup. The tests have been performed on an Intel Core2 CPU (E6700) clocked at 2.6 GHz in 32-bit mode with 4 GB of memory running Ubuntu/Linux 2.6.35. The runtime is split with respect to the components of the decoder (cf. Fig. 1): model estimation, thresholding, score computation. Model refinement aims to improve the initial estimation of the collusion model using side information. Fewer number of search steps are spent in refinement than in the initial estimation. Runtime is dominated by the rare event simulation to find the threshold for accusation.

We also plot the average number of iterations after the first round. For up to 18 colluders, all of them are usually accused in the first iteration and the accusation stops after the second round since no scores exceed the threshold. The maximum number of iterations is reached for 45 colluders. We have marked the regimes introduced in Section 3.1: cases (i) has 2 rounds, (ii) has several rounds, (iii) has 1 round.

Iterative decoding takes up to 50 seconds which is more than the hierarchical decoder proposed by Kuribayashi et

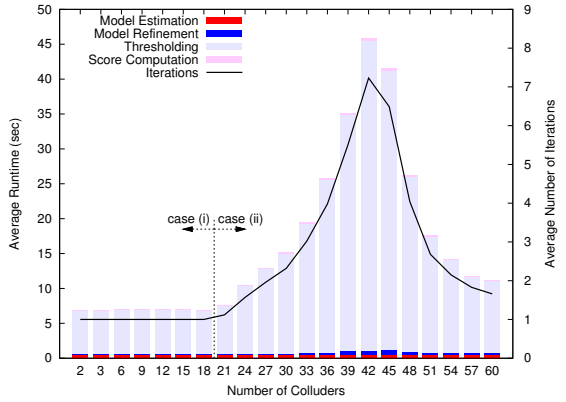

Fig. 6. Average number of iterations and runtime in seconds for the irterative decoder in the Kuribayashi setup.

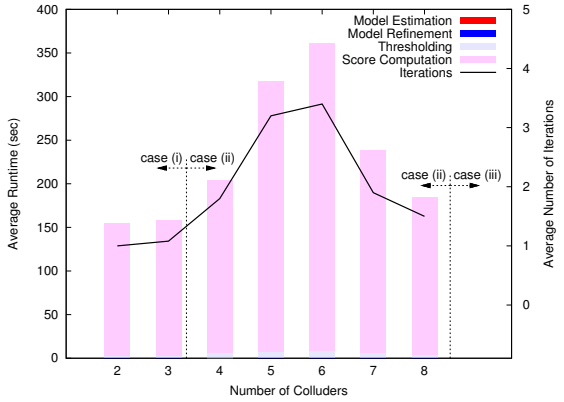

Fig. 7. Average number of iterations and runtime in seconds for the iterative decoder in the Jourdas & Moulin setup.

al. [7]. However, thresholding in [7] is based on the simplifying assumption that scores of users not involved in the collusion follow a Gaussian distribution. When the number of users increases, the runtime impact of our rare event simulation for threshold determination (which makes no assumption about the scores' distribution) diminishes compared to score computation. Score computation for $n = 10\,000$ users takes about 0.04 seconds. The implementation of our score computation appears to be at least 20 times more efficient analyzing the time consumption for different n given in [7].

The runtime results for the large scale scenario (Jourdas & Moulin [8]) are given in Fig. 7. Due to the storage requirements of the users' codewords, the runtime results are reported for an Intel Xeon CPU (X5650) clocked at 2.6 GHz in 64-bit mode with 48 GB of memory running Debian/Linux 2.6.26. The time consumption of the iterative decoder is less than 350 seconds and determined by the score computation. Note that only a single CPU core is used although the computation of accusation scores can be trivially parallelized. Also the memory requirements can be easily alleviated by grouping the codewords into chunk that fit in memory and loading the chunks sequentially from external storage.

6. CONCLUSION

We have proposed an iterative blind single Tardos decoder which allows to accuse more than one colluder while controlling the false-positive probability by means of rare event simulation. The fast implementation is fit for traitor tracing in large-scale multimedia distribution environments with millions of users. Our decoder can not improve the false-negative rate in case (i), i.e. when the code is so short that not a single accusation can be made. We plan to improve this with a joint decoder computing scores of tuples of users.

References

[1] G. Tardos, "Optimal probabilistic fingerprint codes," in *Proc. 35th ACM Symposium on Theory of Computing*, San Diego, CA, USA, 2003, pp. 116–125.

[2] T. Furon, A. Guyader, and F. Cérou, "On the design and optimisation of Tardos probabilistic fingerprinting codes," in *Proc. 10th Information Hiding Workshop*, Santa Barbara, CA, USA, May 2008, vol. 5284 of *LNCS*, pp. 341–356.

[3] M. Wu, W. Trappe, Z. J. Wang, and K. J. R Liu, "Collusion-resistant fingerprinting for multimedia," *IEEE Signal Processing Magazine*, vol. 21, no. 2, pp. 15–27, Mar. 2004.

[4] L. Pérez-Freire and T. Furon, "Blind decoder for binary probabilistic traitor tracing codes," in *Proc. First IEEE Int. Workshop on Information Forensics and Security*, London, UK, Dec. 2009, pp. 56–60.

[5] F. Cérou, T. Furon, and A. Guyader, "Experimental assessment of the reliability for watermarking and fingerprinting schemes," *EURASIP Jounal on Information Security*, 2008, ID 414962, 12 pages.

[6] M. Saito and M. Matsumoto, "A PRNG specialized in double precision floating point numbers using an affine transition," in *Proc. Int. Conference on Monte Carlo and Quasi-Monte Carlo Methods, MCQMC '08*, Montréal, Canada, July 2008, pp. 589–602, Springer.

[7] M. Kuribayashi, N. Akashi, and M. Morii, "On the systematic generation of Tardoss fingerprinting codes," in *Proc. IEEE Workshop on Multimedia Signal Processing, MMSP '08*, Cairns, Australia, Oct. 2008, pp. 748–753.

[8] Jean-Francois Jourdas and Pierre Moulin, "High-rate random-like spherical fingerprinting codes with linear decoding complexity," *IEEE Transactions on Information Forensics and Security*, vol. 4, no. 4, pp. 768–780, Dec. 2009.