Towards Joint Tardos Decoding: The 'Don Quixote' Algorithm

Peter Meerwald* and Teddy Furon

INRIA Rennes Bretagne Atlantique,
Campus de Beaulieu, Rennes, France
{peter.meerwald, teddy.furon}@inria.fr

Abstract. 'Don Quixote' is a new accusation process for Tardos traitor tracing codes which is, as far as we know, the first practical implementation of joint decoding. The first key idea is to iteratively prune the list of potential colluders to keep the computational effort tractable while going from single, to pair,... to t-subset joint decoding. At the same time, we include users accused in previous iterations as side-information to build a more discriminative test. The second idea, coming from the field of mismatched decoders and compound channels, is to use a linear decoder based on the worst case perceived collusion channel. The decoder is tested under two accusation policies: to catch one colluder, or to catch as many colluders as possible. The probability of false positive is controlled thanks to a rare event estimator. We describe a fast implementation supporting millions of users and compare our results with two recent fingerprinting codes.

Keywords: traitor tracing, fingerprinting, transactional watermarking, joint decoder

1 Introduction

Traitor tracing or passive fingerprinting has witnessed a flurry of research efforts since the invention of the now well-celebrated Tardos codes [13]. The codes of G. Tardos are optimal in the sense that the code length m necessary to fulfill the following requirements (n users, c colluders, probability of accusing an innocent below P_{fp}) has the minimum scaling in $O(c^2 \log n P_{\mathsf{fp}}^{-1})$. The accusation process (more precisely its symmetric version proposed by B. Skoric *et al.* [12]) is based on a scoring per user, so-called accusation sum, whose statistics only depend on the collusion size c, but not on the collusion attack (*e.g.* minority vote, majority vote, interleaving, etc). The alternative accusation strategy has also been tested: the accusation process estimates the collusion attack in order to resort to a matched scoring which is more discriminative [10].

However, these two previous strategies pertain to the same family: the single decoders, *i.e.* processes computing a score per user independently of the other

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

codewords and finally accusing users whose score is above a given threshold. Another family is that of the joint decoders, *i.e.* processes computing a score per subset of t users. As far as we know, K. Nuida was the first to propose *and* experiment some sort of a joint decoder [7]. The accusation algorithm only works for very limited collusion size and it doesn't scale well when the number of users n is more than some hundreds. Indeed, so far joint decoders are of particular interest only in theoretical analysis of fingerprinting. P. Moulin [6], and, independently, E. Amiri and G. Tardos [2] show that the capacity of fingerprinting is given by a *maxmin* game whose pay-off is the mutual information $I(Y;X^c|P)\cdot c^{-1}$ where Y is a r.v. representing the symbol decoded from the pirated copy, P is the r.v. denoting the secret of the code, and $X^c=\{X_{j_1},\ldots,X_{j_c}\}$ is the set of the c symbols assigned to the colluders.

Both papers proposed a joint decoder based on the empirical mutual information computed on the joint type of the observations $(\mathbf{y},\boldsymbol{\varphi},\mathbf{p})$ where $\boldsymbol{\varphi}=\sum_{k=1}^{t}\mathbf{x}_{j_k}$ (termed accumulated codeword in the sequel) for the t-subset of users $\{j_1,\ldots,j_t\}$, $t\leq c$. Note that these papers are theoretical studies and that they do not contain any experiment. A practical implementation is hampered by two drawbacks: this is not a linear decoder [1, Def. 1] and the complexity is proportional to $\binom{n}{t}$, the number of t-subsets, *i.e.* in $O(n^t)$. In the quest of practical implementations of this theoretical decoder, 'Don Quixote' is a milestone based on two key ideas: (i) there is no need to compute a score for all t-subsets if we can invent a mechanism preselecting a small number of suspects who are the most likely guilty users; (ii) a linear decoder allowing a fast implementation of the scoring function.

These ideas are indeed not easily translated into practical algorithms. In real life scenarios such as Video-on-Demand portals, m bits are in copies of a movie, which are afterwards distributed to n clients (the parameters (m,n) vary from one Work to another). The collusion size c is neither known at the code construction nor at the accusation side. Therefore, one never knows how the rate $m^{-1}\log n$ compares to the theoretical capacity which depends on c. However, for (i), we need to identify suspects whenever it is possible (*i.e.* when the rate is below capacity) while guaranteeing a probability of false alarm P_{fp}. Efficient linear decoders pertaining to (ii) are based on likelihood ratio, which can't be computed in practice since we do not know the collusion strategy. Fortunately, information theorist have recently come up with a very elegant solution providing universal linear decoders performing well (*i.e. capacity achieving*) over a family of channels while ignoring the active channel [1]. This theory of compound channel fits very well with the traitor tracing framework.

2 Structure of the Don Quixote algorithm

Before detailing the structure of the proposed decoder, let us briefly remind the construction of a Tardos code. Let (m,n) be the length and the size of the code. First, draw randomly m variables $\mathbf{p}=(p(1),\ldots,p(m))^T$ s.t. $p\overset{\text{i.i.d.}}{\sim}f(p)$. Then draw randomly and independently the elements of the j-th codeword $\mathbf{x}_j=$

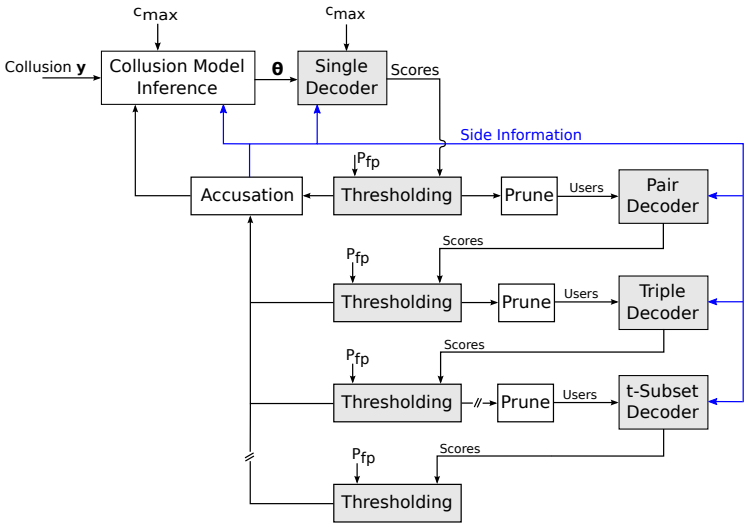


Fig. 1. Overview of the iterative, side-informed joint Tardos fingerprint decoder.

$(x_j(1), \ldots, x_j(m))$ s.t. $\mathbb{P}(x_j(i) = 1) = p(i)$. We chose the pdf $f(p)$ recommended by G. Tardos.

The iterative architecture of our joint decoder is sketched in Figure 1. For sake of clarity, we postpone details about the computation of scores and the thresholding (shaded blocks of Fig. 1) to the next sections. The decoder can be employed in *catch-one* and *catch-many* traitor tracing scenarios [14]. In the first case, our iterative decoder simply stops after the first accusation; in the later case, the decoder stops when $c_{\max}$ accusations have been made or when no further accusations can safely be made using the $t_{\max}$-subset decoder ($2 \leq t_{\max} \leq c_{\max}$).

2.1 Iterative, joint decoding

The theoretical papers [2, 6] tell us that scores computed from subsets of t users are more discriminative as t increases, provided that $t \leq c$, c being the real collusion size. Discriminative means that scores of subsets of innocent users are statistically more separated from scores for guilty users – the Kullback-Leibler distance between their pdf is significantly higher. Our point is that, indeed, hybrid subsets containing κ colluders and $t - \kappa$ innocents have also greater scores in expectation as κ increases. Therefore, by pruning out users involved in subsets of small score, we are likely maintaining a list of suspects with a good number of colluders.

At the beginning, $\mathcal{X}$ is the set of n codewords or users. The t-th iteration of our algorithm takes a set of codewords $\mathcal{X}^{(t)} \subseteq \mathcal{X}$ and computes the scores for each subset of t codewords from $\mathcal{X}^{(t)}$. Denote $n^{(t)} = |\mathcal{X}^{(t)}|$, there are then $\binom{n^{(t)}}{t} = O((n^{(t)})^t)$ scores to be computed. For instance, in the first iteration, $\mathcal{X}^{(1)} = \mathcal{X}$

Subset size (t)	2	3	4	5	6	7	8
Users suspected ($p^{(t)}$)	3 000	300	103	58	41	33	29
Computed subsets $\binom{p^{(t)}}{t}$	4 498 500	4 455 100	4 421 275	4 582 116	4 496 388	4 272 048	4 292 145
Total subsets $\binom{n}{t}$	$\sim 10^{11}$	$\sim 10^{17}$	$\sim 10^{22}$	$\sim 10^{27}$	$\sim 10^{33}$	$\sim 10^{38}$	$\sim 10^{43}$

Table 1. Maximal number $p^{(t)}$ of suspected users input to the joint t-subset decoder versus total number of subsets without pruning out users for $n = 10^6$.

and the scores are just the $n^{(1)} = n$ outputs of a single decoder. We assume to have the computation power scaling as $O(n)$ such that this first iteration is feasible. The key idea is to gradually reduce $n^{(t)}$ such that the computation of scores remains tractable. For instance, if $n^{(t)} = O(n^{1/t})$ then the t-th iteration relies on a $O(n)$ scores computation just like the first iteration.

During each iteration, some users might be deemed guilty (cf. Section 2.4) and added to the side information (cf. Section 3.1).

The main operation is to construct the subset $\mathcal{X}^{(t+1)}$ of suspects to be passed to the following iteration. Suspects are users so far neither accused nor declared as innocent. The users get ranked (with guilty users most likely placed in top positions) and the first $n^{(t+1)}$ users compose the set $\mathcal{X}^{(t+1)}$ while the others are discarded (*i.e.* considered as innocents). The $(t+1)$-th iteration starts by computing scores for subset of size $t+1$ from $\mathcal{X}^{(t+1)}$, see Section 3.3 for details. For $t > 1$, the size $n^{(t+1)} \leq p^{(t+1)}$ where $p^{(t+1)}$ is the upper size and runtime limit that our computer can handle. Table 1 gives values s.t. the number of subsets is kept approximately constant at about 4 500 000. The choices for $p^{(t)}$ are presumably not optimal – other values may allow a better distribution of resources – but a necessaary trade-off between the computational effort and the decoding performance.

2.2 Pruning out

First iteration. The first iteration computes a score per user: the bigger the score, the more likely the user is a colluder. If some conditions are met, users with the highest scores might be accused (cf. Section 2.4). Users are ranked according to their score in decreasing order. The first $n^{(2)} \leq p^{(2)}$ users are included in the set $\mathcal{X}^{(2)}$.

***t*-th iteration, $t > 1$.** Once the scores for all t-subsets are computed, they are ranked in decreasing order. Again, if accusations can be safely made, some users from the first-ranked subset are deemed guilty. The others are included in set $\mathcal{X}^{(t+1)}$. The algorithm browses down the sorted list of subsets and includes their users in $\mathcal{X}^{(t+1)}$ if they have not been already included and if they have not been accused. This stops when $n^{(t+1)} = p^{(t+1)}$ (the users are listed in arbitrary order in a t-subset, therefore for the last subset under suspicion, the last users might be relaxed while the first are suspected) or when the last subset of the sorted list has been analyzed.

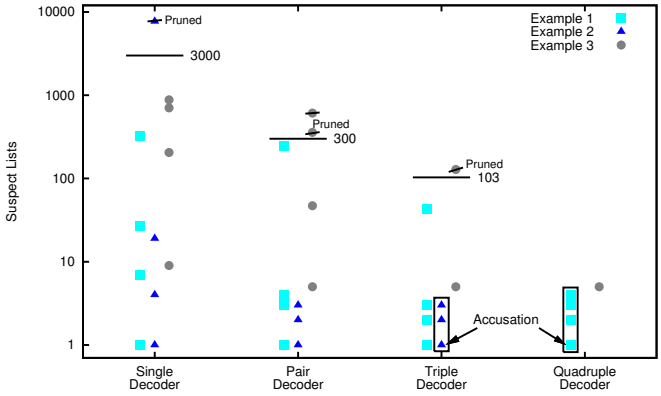


Fig. 2. Examples of the iterative decoding and pruning process for $c = 4$.

Figure 2 illustrates the iterative decoding and pruning process where the positions of the colluders are marked with different symbols in three examples. The first stage denoted *single decoder* represents the list of suspects/users ranked according to their accusation scores after single decoding. Since no user's score is above the accusation threshold, the process continues in the following iteration. Only suspects ranked within the first 3 000 positions are passed to the *pair decoder* and the illustration shows the suspect list ranked after computing the scores of user pairs. Users within a pair are ordered according to the criterion defined in Section 2.4. After pruning – now the list is limited to 300 positions – the remaining suspects are fed to the *triple decoder*. Note that the positions of the colluders generally move towards the top of the list (the bottom of the illustration) with each iteration as shown in the examples. This observation allows us to reduce the suspect list at each iteration while likely retaining the colluders. On the other hand, colluders may be discarded, as visualized in *Examples 2 & 3*. Pruning is the necessary trade-off to reduce the computational burden of the decoder. The users of the subset whose score is the highest and above the threshold are framed in the illustration. Their top-ranked user is accused and added to the side information.

2.3 Enumerating all t-subsets

The joint t-subset decoder has to enumerate all $\binom{n^{(t)}}{t}$ subsets and compute the corresponding scores. One way to implement the generation of all t-subsets of $\mathcal{X}^{(t)}$ is the *revolving door* algorithm[1] [9] which changes exactly one element of the subset at each enumeration step.

In particular, the score of the k-th t-subset $\mathbf{t}_k$ only depends on the accumulated codewords $\boldsymbol{\varphi}_k = \sum_{j_\ell \in \mathbf{t}_k} \mathbf{x}_{j_\ell}$. The revolving door is initialized with the first

[1] Termed algorithm **R** by Knuth [5, Chap. 7.2.1.3].

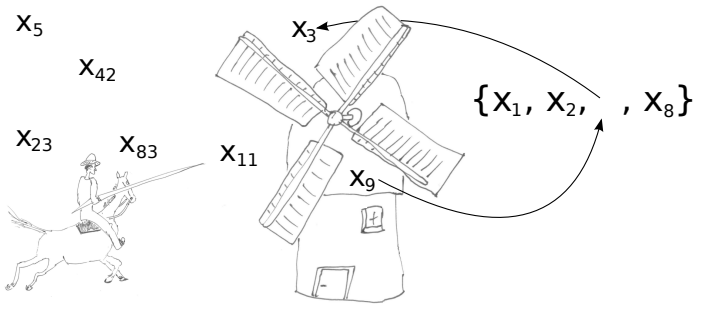


Fig. 3. Illustration of the revolving door algorithm for $t = 4$.

subset $\mathbf{t}_1 = \{j_1, \ldots, j_t\}$ whose accumulated codeword is $\boldsymbol{\varphi}_1$. At each step, the algorithm replaces one user $j_\dagger \in \mathbf{t}_k$ with a new user $j_\star$ and computes the updated code sequence relating to the combination $\mathbf{t}_{k+1}$ as $\boldsymbol{\varphi}_{k+1} = \boldsymbol{\varphi}_k - \mathbf{x}_{j_\dagger} + \mathbf{x}_{j_\star}$. Fig. 3 provides an illustration. The benefit of the *resolving door* is that the computational effort to generate a t-subset and its associated accumulated codeword is independent of size t.

2.4 Accusation

Let $\mathbf{t}^\diamond$ denote the subset with the highest score. We accuse one user of the t-subset $\mathbf{t}^\diamond$, only if its score is greater than a threshold: $s_{\mathbf{t}^\diamond} > \tau$. The computation of threshold τ is explained hereafter in Section 3.4. The thresholding operation ensures that subsets with score above τ contain at least one colluder with a very high probability. Assume now that this condition is met. Obviously, for the first iteration, $t = 1$ and the single user in subset $\mathbf{t}^\diamond$ is accused. For $t > 1$, we propose the following method. In order to identify and accuse the most probable traitor in $\mathbf{t}^\diamond$, we record for each user $j \in \mathcal{X}^{(t)}$ the subset leading to that user's highest score:

$$\mathbf{t}_j^\diamond = \arg\max_{\mathbf{t}} \{s_{\mathbf{t}} \mid j \in \mathbf{t}\}. \tag{1}$$

Next, we count how often each user $j_k \in \mathbf{t}^\diamond$ appears in the recorded subsets $\{\mathbf{t}_j^\diamond\}_{j \in \mathcal{X}^{(t)}}$ and denote this value a_{j_k}. Finally, we accuse the user $j^\diamond$ appearing most often:

$$j^\diamond = \arg\max_{j \in \mathbf{t}^\diamond} a_j. \tag{2}$$

3 Technical details

This section describes four remaining operations: side-information, inferences about the collusion model, scoring, and the thresholding.

3.1 Side Information

The knowledge of the identity of some colluders is beneficial in two operations: derivation of better inferences about the collusion channel, and derivation of more discriminative scores. It is well known in estimation and detection theory that conditioning (*i.e.* to side-inform with prior knowledge) is always helpful on average. Denote by $\mathcal{X}_{\mathsf{SI}}$ the set of accused users (subscript SI denotes Side Information). At the beginning, $\mathcal{X}_{\mathsf{SI}} = \emptyset$. If a user is accused as described in Section 2.4, then he is removed from $\mathcal{X}^{(t)}$ and included into $\mathcal{X}_{\mathsf{SI}}$. If nobody is accused, then iteration $(t+1)$ starts with $\mathcal{X}^{(t+1)}$. If someone is accused, then iteration t is not over. Since new side information is available, we can benefit from it right away. A new inference process is run with the new $\mathcal{X}_{\mathsf{SI}}$, and the scores for t-subsets are computed again with the new inference, the new conditioning $\mathcal{X}_{\mathsf{SI}}$ and over the new set $\mathcal{X}^{(t)}$. The t-th iteration breaks this loop whenever no additional colluder is identified.

3.2 Inferences about the collusion model

A long tradition in Tardos traitor tracing codes is to model the attack led by c colluders by a vector $\boldsymbol{\theta}^{(c)} = (\theta_0^{(c)}, \theta_1^{(c)}, \ldots, \theta_c^{(c)})$ where $\theta_\sigma^{(c)} = \mathbb{P}(y_i = 1 | \sum_{k=1}^{c} x_{j_k}(i) = \sigma)$ [10]. In words, when the colluders have σ symbols '1' over c, they flip a coin of bias $\theta_\sigma^{(c)}$ to decide whether they put symbol '1' or a '0' in the pirated sequence. The marking assumption holds if $\theta_0^{(c)} = 1 - \theta_c^{(c)} = 0$. The main difficulty is that $\boldsymbol{\theta}^{(c)}$ cannot be estimated from the observations $(\mathbf{y}, \mathbf{p})$ since, for any integer $c' > c$ there exists $\boldsymbol{\theta}^{(c')}$ s.t. $\mathbb{P}(y = 1 | p, \boldsymbol{\theta}^{(c')}) = \mathbb{P}(y = 1 | p, \boldsymbol{\theta}^{(c)}), \forall p \in (0, 1)$. We call $\boldsymbol{\theta}^{(c')}$ the equivalent attack of $\boldsymbol{\theta}^{(c)}$ of size c'. The parameter of the model cannot be identified except if we were knowing the collusion size c. We chose the maximum log-likelihood estimator (MLE) for a given $\hat{c}$:

$$\hat{\boldsymbol{\theta}}^{(\hat{c})} = \underset{\boldsymbol{\theta} \in [0,1]^{\hat{c}+1} \;\; s.t. \;\; \theta(0)^{(\hat{c})} = 0, \theta(\hat{c})^{(\hat{c})} = 1}{\arg\max} \log \mathbb{P}(\mathbf{y} | \mathbf{p}, \mathcal{X}_{\mathsf{SI}}, \boldsymbol{\theta}), \tag{3}$$

with $\mathbb{P}(\mathbf{y} | \mathbf{p}, \boldsymbol{\theta}) = \prod_{i=1}^{m} \mathbb{P}(y(i) | p(i), \mathcal{X}_{\mathsf{SI}}, \boldsymbol{\theta})$. Section 3.3 details the computation of this likelihood. However, due to the lack of identifiability, this approach cannot estimate c, but only $\hat{\boldsymbol{\theta}}^{(\hat{c})}$ for a given $\hat{c}$. For a long enough code, the MLE accurately finds $\boldsymbol{\theta}^{(c)}$ if $\hat{c} = c$, or its equivalent attack of size $\hat{c}$ if $\hat{c} > c$; yet there is no way to distinguish the two cases.

We have experimentally noticed that scores based on $\hat{\boldsymbol{\theta}}^{(\hat{c})}$ are only slightly less powerful than the optimal ones (based on the real $\boldsymbol{\theta}^{(c)}$) provided that $\hat{c}$ is bigger than the real c. Therefore, we assume that $c < c_{\mathsf{max}}$ and set $\hat{c} = c_{\mathsf{max}}$. We estimate not the real collusion parameter but its equivalent attack of size c_{max} (this is why we rather speak of collusion inference than collusion estimation).

The theory of compound channel justifies this approach. Suppose $c < c_{\mathsf{max}}$ and consider the family of collusions $\{\boldsymbol{\theta}^{(c')}\}_{c'=c}^{c_{\mathsf{max}}}$ gathering the real collusion $\boldsymbol{\theta}^{(c)}$ and its equivalent attacks of size from $c+1$ to c_{max}. It can be shown[2]

[2] The journal version of this paper contains the proof.

that this family is a one-sided compound channel [1, Def. 3, Eq.(8)]. Therefore by [1, Lemma 5], we know that a good (*i.e.* information theorists say *capacity achieving*) linear decoder is the maximum likelihood decoder tuned on the worst element of the family, which is in our case the collusion $\boldsymbol{\theta}^{(c_{\max})}$.

3.3 Score computation

The score is just the log-likelihood ratio tuned on the inference $\hat{\boldsymbol{\theta}}^{(c_{\max})}$. We give its most generic expression for a subset $\mathbf{t}$ of t users and side information $\mathcal{X}_{\mathsf{SI}}$ containing n_{SI} codewords of already accused users. Denote by $\boldsymbol{\rho}$ and $\boldsymbol{\varphi}$ the accumulated codewords of $\mathcal{X}_{\mathsf{SI}}$ and $\mathbf{t}$: $\boldsymbol{\rho} = \sum_{j\in\mathcal{X}_{\mathsf{SI}}} \mathbf{x}_j$ and $\boldsymbol{\varphi} = \sum_{j\in\mathbf{t}} \mathbf{x}_j$. We have $\forall i \in [m]$, $0 \le \rho(i) \le n_{\mathsf{SI}}$ and $0 \le \varphi(i) \le t$.

Denote $\mathcal{H}_0$ the hypothesis where subset $\mathbf{t}$ is composed of innocent users. Then $\mathbf{y}$ is statistically independent from its codewords which in turn only depend on $\mathbf{p}$:

$$\mathcal{H}_0 : \quad \mathbb{P}(\mathbf{y}, \{\mathbf{x}_j\}_{j\in\mathbf{t}} | \mathbf{p}, \hat{\boldsymbol{\theta}}^{(c_{\max})}, \mathcal{X}_{\mathsf{SI}}) = \mathbb{P}(\mathbf{y}|\mathbf{p}, \boldsymbol{\theta}, \mathcal{X}_{\mathsf{SI}}) \mathbb{P}(\{\mathbf{x}_j\}_{j\in\mathbf{t}}|\mathbf{p}) \tag{4}$$

Denote $\mathcal{H}_1$ the alternative where subset $\mathbf{t}$ is composed of colluders. Then $\mathbf{y}$ is statistically dependent of its codewords:

$$\mathcal{H}_1 : \quad \mathbb{P}(\mathbf{y}, \{\mathbf{x}_j\}_{j\in\mathbf{t}} | \mathbf{p}, \hat{\boldsymbol{\theta}}^{(c_{\max})}, \mathcal{X}_{\mathsf{SI}}) = \mathbb{P}(\mathbf{y}|\{\mathbf{x}_j\}_{j\in\mathbf{t}}, \mathbf{p}, \hat{\boldsymbol{\theta}}^{(c_{\max})}, \mathcal{X}_{\mathsf{SI}}) \mathbb{P}(\{\mathbf{x}_j\}_{j\in\mathbf{t}}|\mathbf{p}) \tag{5}$$

All these sequences are composed of independent r.v. thanks to the code construction and the memoryless nature of the collusion. Moreover, the collusion only depends on the number of symbol '1' present in the codewords of a subset, *i.e.* the accumulated codeword. Therefore, the score of subset $\mathbf{t}$ is just the log-ratio of the two previous probability expressions which simplifies to:

$$s = \sum_{y(i)=1} \log \frac{\alpha(i)}{\beta(i)} + \sum_{y(i)=0} \log \frac{1-\alpha(i)}{1-\beta(i)}, \tag{6}$$

with the following expressions:

$$\alpha(i) = \mathbb{P}(y=1|(\varphi(i),t),(\rho(i),n_{\mathsf{SI}}),p(i),\hat{\boldsymbol{\theta}}^{(c_{\max})}) = P(\varphi(i)+\rho(i), t+n_{\mathsf{SI}}, p(i), \hat{\boldsymbol{\theta}}^{(c_{\max})})$$
$$\beta(i) = \mathbb{P}(y=1|(\rho(i),n_{\mathsf{SI}}),p(i),\hat{\boldsymbol{\theta}}^{(c_{\max})}) = P(\rho(i), n_{\mathsf{SI}}, p(i), \hat{\boldsymbol{\theta}}^{(c_{\max})})$$

and function $P(\cdot)$ is defined by:

$$P(u,v,p,\hat{\boldsymbol{\theta}}^{(c_{\max})}) = \sum_{\sigma=u}^{c_{\max}-v+u} \hat{\theta}(\sigma)^{(c_{\max})} \binom{c_{\max}-v}{\sigma-u} p^{\sigma-u}(1-p)^{c_{\max}-v-\sigma+u} \tag{7}$$

This expression is compact, involved, but very generic. In words, it gives the probability that $y=1$ knowing that the symbol '1' has been distributed to users with probability p, the collusion model $\hat{\boldsymbol{\theta}}^{(c_{\max})}$, and the identity of v colluders who have u symbol '1' and $v-u$ symbol '0' at this index.

The inference on the collusion model searches for a $\boldsymbol{\theta}^{(c_{\mathsf{max}})}$ maximizing the following likelihood:

$$\log \mathbb{P}(\mathbf{y}|\mathbf{p}, \boldsymbol{\theta}, \mathcal{X}_{\mathsf{SI}}) = \sum_{y(i)=1} \log \beta_i + \sum_{y(i)=0} \log 1 - \beta_i. \tag{8}$$

In the first iteration, a single decoder is used: $t = 1$ and $\boldsymbol{\varphi} = \mathbf{x}_j$ for user j. If nobody has been deemed guilty so far, then $\rho(i) = n_{\mathsf{SI}} = 0, \forall i \in [m]$. The t-th iteration works on subsets of size t. However, our scoring is only defined if $t + n_{\mathsf{SI}} \leq c_{\mathsf{max}}$. Therefore, for a given size of side-information, we cannot conceive score for subset of size bigger than $t_{\mathsf{max}} = c_{\mathsf{max}} - n_{\mathsf{SI}}$. This implies that in the catch-all scenario, the maximal number of iterations depends on how fast $\mathcal{X}_{\mathsf{SI}}$ grows.

3.4 Thresholding

The issue here is the translation of the scores into probabilities. At a given iteration and a given state of the side information, all the subset scores are computed in the same deterministic way. The idea is to generate subsets composed of new codewords and to compute their scores. We are then sure to observe scores of subset of innocents since these codewords have not been used to forge $\mathbf{y}$. With a Monte Carlo simulation, we can estimate the probability that the score of an innocent subset is bigger than threshold τ, or the other way around, the threshold τ such that this probability is below ϵ. This approach works whatever the way scores are computed.

In the first iteration, the subset is just a singleton, the codeword of one user, and that user is either innocent either guilty. Therefore, users whose scores are above the threshold are accused and included in $\mathcal{X}_{\mathsf{SI}}$. Denote P_{fp} the total probability of false positive, *i.e.* accusing at least one innocent, and ϵ the probability of wrongly accusing a given innocent user. Since the codewords are i.i.d. and $c \ll n$, we have $P_{\mathsf{fp}} = 1 - (1 - \epsilon)^{n-c} \approx n\epsilon$. P_{fp} is stipulated in the requirements and we fix $\epsilon = {}^{P_{\mathsf{fp}}}/_{n}$. In the t-th iteration ($t > 1$), the same Monte Carlo simulation over subsets of size t is run. It estimates the threshold s.t. the score of a subset of innocents is greater than τ only with a probability ϵ. In other words, scores above τ indicate subset with at least one colluder. A further analysis identifies and accuses the most likely one among the t users (cf. Section 2.4). Again, ϵ should be set as low as ${}^{P_{\mathsf{fp}}}/\binom{n}{t}$ to control the total probability of false alarm.

The only problem is that a large n implies a very low probability ϵ for both cases ($t = 1$ and $t > 1$), and a Monte Carlo simulation is then bad at estimating accurately threshold τ. This is the reason why we implemented an numerical estimator based on rare event analysis [3].

4 Experimental results

The 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[3]; the remaining components are implemented with double-precision. Pseudo-random numbers are generated with the SIMD-oriented Fast Mersenne Twister (dSFMT)[4] [11].

All runtime results are reported for a single core of a x86 Intel Core2 CPU (E6700) clocked at 2.6 GHz with 2 GB of memory running Ubuntu/Linux 10.10.

The joint decoder receives lists of suspects whose length are upper bounded by values of Table 1.

4.1 Catch-one scenario

Here the aim is to catch the most like colluder – this is the tracing scenario most commonly considered in the literature. We compare our single and joint decoder performance against the results provided by Nuida *et al.* [8]. These authors assumed that c is known for the code construction and the decoding. For a fair comparison, our decoder uses this assumption: $c_{\mathsf{max}} = c$.

The experimental setup considers $n = 1\,000\,000$ users and $c \in \{2, 3, 4, 6, 8\}$ colluders performing *worst-case* attack [4]. In Fig. 4, we plot the empirical probability of error $P_{\mathsf{e}} = P_{\mathsf{fp}} + P_{\mathsf{fn}}$ obtained by running at least 10 000 experiments for each setting versus the code length m. The false-positive error is controlled by thresholding based on rare-event simulation, $P_{\mathsf{fp}} = 10^{-3}$. For shorter code length, almost exclusively false-negative errors occur. As expected, we observe a huge decoding performance improvement for the joint decoder over the single decoder. The advantage is much more pronounced when a larger number of colluders collaborates.

Table 2 compares the code length to obtain an error rate of $P_{\mathsf{e}} = P_{\mathsf{fp}} + P_{\mathsf{fn}} = 10^{-3}$ for our proposed Tardos decoders with the results reported by Nuida *et al.* [8] under marking assumption. While the joint decoder only marginally improves the decoding performance for two colluders, it almost halves the code length for four colluders.

The column *hypothetical* of Table 2 reports simulation results of a joint decoder that knows the identity of the colluders and just computes scoring and thresholding for the colluders. The simulation allows to judge the performance gap between the proposed joint decoder operating on a *pruned* list of suspects (potentially discarding colluders) and the unconstrained joint decoder.

Figure 5 (a) shows in which iteration the first out of $c = 4$ colluders is successfully identified for varying code length, $P_{\mathsf{fp}} = 10^{-3}$. The benefit of joint decoding is best seen for intermediate code lengths between $m = 352$ and $m = 864$. For longer codes the single decoder is sufficient to make the first accusation. Figure 5 (b) illustrates the average runtime in seconds for score computation and

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

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

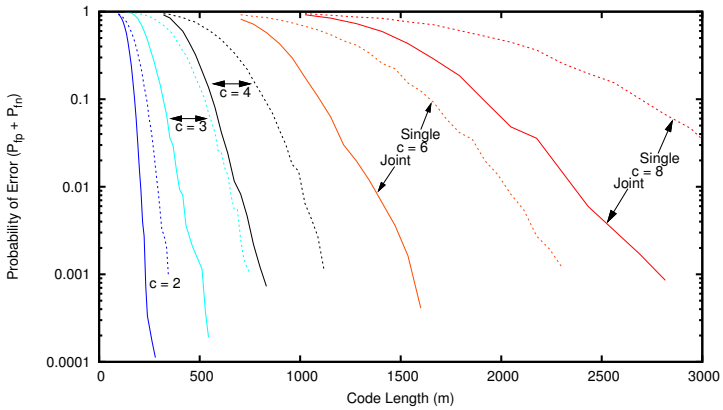


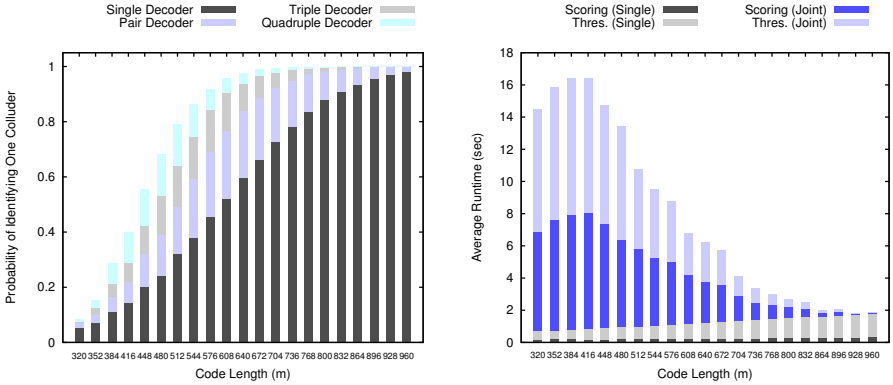
Fig. 4. Probability of error versus code lengths for the single and joint decoder for different collusion sizes; $n = 10^6$, *worst-case* attack.

Colluders (c)	Nuida *et al.* [8]	Proposed Decoder		Hypothetical
		Single	Joint	
2	253	~ 344	~ 232	~ 232
3	877	~ 752	~ 512	~ 400
4	1454	~ 1120	~ 784	~ 720
6	3640	~ 2304	~ 1568	~ 1440
8	6815	~ 3712	~ 2688	~ 2432

Table 2. Code length comparison for $n = 10^6$, *worst-case* attack, $P_e = 10^{-3}$.

thresholding for the single and joint decoders in that scenario. For short code length all decoding stages (up to $t = 4$) have to be run – often unsuccessfully. A significant amount of the execution time is spent in thresholding relative to scoring for the number of computed subsets, $\binom{p^{(t)}}{t} \sim 4\,500\,000$.

In Fig. 6 we plot the probability of correctly identifying one colluder and the iteration number leading to this accusation. This time, we vary the number of score computations performed in each iteration from 10^5 to 10^9 by controlling the suspect list sizes $\{p^{(t)}\}$. The rightmost results relate to the hypothetical joint decoder which does not have to enumerate all combinations but just computes the accusation scores for the colluders. Surprisingly, a significant difference in accusation performance can only be observed at the last iteration (*i.e.* the quadruple decoder). An equal weighting of the computation resources over the iterations is certainly not optimal. This experiment seems to conclude that the emphasis should be put on the last iterations. Yet, it is not clear what the optimal resources distribution is.



(a) Accusations (b) Runtime

Fig. 5. Iteration making the first accusation (a), and average runtime in seconds for score computation and thresholding for the single and joint decoders (b); $n = 10^6$, $c = 4$, *worst-case* attack.

4.2 Catch-many scenario

We now consider the more realistic case where the code length m is fixed. The only assumption at the decoder side is that $c \leq c_{\mathsf{max}}$. The aim is to identify as many colluders as possible. Figure 7 shows the average number of identified colluders by the symmetric Tardos single decoder, our non-iterated single, the iterative side-informed single and our iterative side-informed joint decoders. The experimental setup considers $n = 1\,000\,000$ users, code length $m = 2048$, and *worst-case* collusion attack carried out by between two and eight colluders. The global probability of false positive is fixed to $P_{\mathsf{fp}} = 10^{-3}$. The performance advantage of the more sophisticated decoders is evident. The joint decoder has a good chance to catch most of the colluders even when $c = 8$.

In Fig. 8 (a) we analyse the average number of accusations made per iteration (same setup as above). Figure 8 (b) shows the average runtime in seconds accounted for score computation and thresholding of the iterative single and joint decoders. The longest runtimes are observed for $c = 2$ and $c = 8$. In the first case, both colluders are caught by the single decoder, yet the remaining iterations up to the 6-subset decoder have to be run since $c_{\mathsf{max}} = 8$.

4.3 Runtime performance analysis

Table 3 provides average runtime results in seconds split up per decoder component for two traitor tracing scenarios with $n = 10\,000$ and $n = 1\,000\,000$ users. The runtime for the collusion model estimation and refinement is negligible and independent of the number of users, $O(c \cdot m)$.

Single decoding can be efficiently implemented to compute more than ten million scores for a code of length $m = 320$ per second. The complexity is

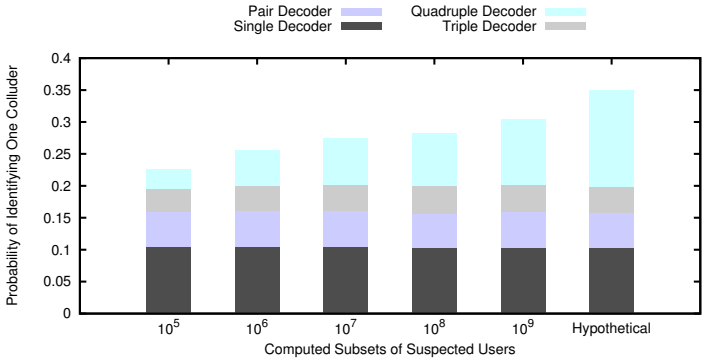


Fig. 6. Iteration making the first accusation with different number of subsets; $n = 10^6$, $m = 384$, $c = 4$, *worst-case* attack.

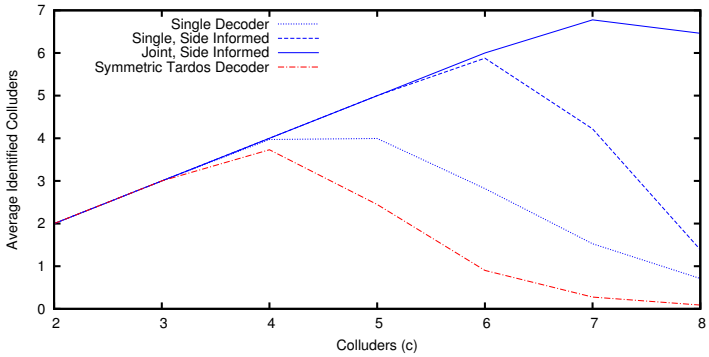
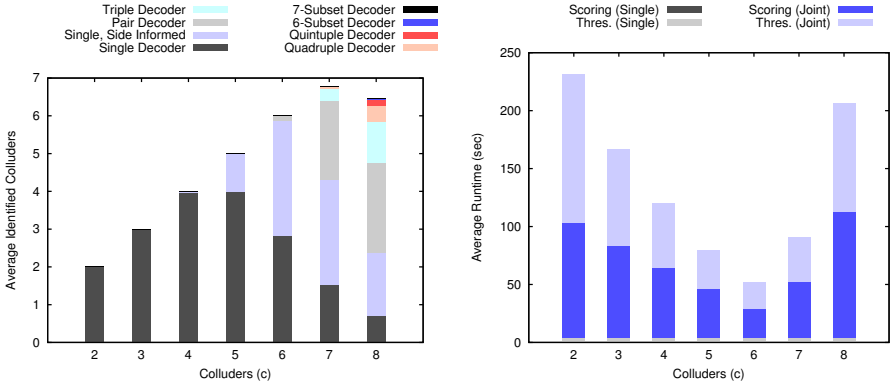


Fig. 7. Average identified traitors for different number of colluders performing *worst-case* attack; $n = 10^6$, $m = 2048$, $P_{\mathsf{fp}} = 10^{-3}$, $c_{\mathsf{max}} = 8$.

$O(n \cdot m + n \cdot \log n)$. The second term relates to sorting the results which consumes a substantial parts of the runtime for small m. The runtime contribution of the joint decoding stage clearly depends on the size of pruned list of suspects, $O(m \cdot p)$ and is independent of the subset size t thanks to the *revolving door* enumeration method. Our implementation performs almost two million joint score computations per second.

Thresholding accounts for more than half of the runtime in the experimental setups investigated in this work. However, this is not a serious issue for applications with a large user base or when p becomes large. Thresholding depends on the subset size t because a large number of random codeword combinations must be generated and because we seek lower probability level in $O(P_{\mathsf{fp}}/n^t)$. Therefore, the complexity is in $O(m \cdot t^2 \cdot \log(n/P_{\mathsf{fp}}))$.

Note that all runtime results have been obtained with single CPU core although a parallel implementation can be easily achieved. The score computation (Eq. 6) has been implemented using pre-computed weights which reduce the



(a) Accusations (b) Runtime

Fig. 8. Average number of accusations per iteration (a); average runtime in seconds for score computation and thresholding for the single and joint decoders (b); $n = 10^6$, $m = 2048$, *worst-case* attack, $P_{\text{fp}} = 10^{-3}$.

computation effort to a single table lookup for each codeword symbol and the accumulation of the values.

5 Conclusion

'Don Quixote' is built on three main pillars. Joint decoding is made affordable by an iterative algorithm pruning out users that are likely not guilty. The theory of compound channel gives fast linear and discriminative scores. The rare event simulation guarantees the reliability of the accusation by controlling the probability of false positive. The collusion size and process are nuisance parameters that are neither needed for the construction of the code, nor at the accusation side. The decoding performance is at the forefront of the state of the art.

References

1. Abbe, E., Zheng, L.: Linear universal decoding for compound channels. IEEE Transactions on Information Theory 56(12), 5999–6013 (Dec 2010)
2. Amiri, E., Tardos, G.: High rate fingerprinting codes and the fingerprinting capacity. In: Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '09. pp. 336–345. SIAM, New York, NY, USA (Jan 2009)
3. Cérou, F., Furon, T., Guyader, A.: Experimental assessment of the reliability for watermarking and fingerprinting schemes. EURASIP Jounal on Information Security (2008), iD 414962, 12 pages
4. Furon, T., Pérez-Freire, L.: Worst case attacks against binary probabilistic traitor tracing codes. In: Proc. First IEEE Int. Workshop on Information Forensics and Security. pp. 46–50. London, UK (Dec 2009)

Avg. Runtime (sec) / Decoder Component	$n = 10\,000$		$n = 1\,000\,000$	
	$m = 320$	$m = 640$	$m = 320$	$m = 640$
Collusion Model ($\hat{\boldsymbol{\theta}}^{(4)}$)	0.00	0.01	0.00	0.01
Single Decoder (s_j)	0.01	0.01	0.17	0.23
Thresholding	0.48	0.90	0.51	0.98
Pair Decoder	2.34	4.46	2.38	4.53
Thresholding	1.48	2.69	1.68	3.07
Triple Decoder	2.26	4.41	2.29	4.45
Thresholding	2.32	4.18	2.79	5.00
Quadruple Decoder	2.19	4.43	2.20	4.42
Thresholding	3.17	5.76	4.02	6.96
Total	14.34	26.85	16.04	29.65

Table 3. Average runtime in seconds per decoder component; the number of joint accusation score computations is fixed to approximately 4.5 million.

5. Knuth, D.E.: The Art of Computer Programming, Generating All Combinations and Partitions, vol. 4. Addison-Wesley (Jul 2005), Fascicle 3
6. Moulin, P.: Universal fingerprinting: Capacity and random-coding exponents. In: Proc. IEEE International Symposium on Information Theory, ISIT '08. pp. 220–224. Toronto, ON, Canada (Jul 2008)
7. Nuida, K.: Short collusion-secure fingerprint codes against three pirates. In: Proceedings of the Information Hiding Workshop, IH '10. Lecture Notes in Computer Science, vol. 6387, pp. 86–102. Springer, Calgary, Canada (Oct 2010)
8. Nuida, K., Fujitsu, S., Hagiwara, M., Kitagawa, T., Watanabe, H., Ogawa, K., Imai, H.: An improvement of discrete Tardos fingerprinting codes. Designs, Codes and Cryptography 52(3), 339–362 (Mar 2009), http://eprint.iacr.org/2008/338
9. Payne, W.H., Ives, F.M.: Combination generators. ACM Transactions on Mathematical Software 5(2), 163–172 (Jun 1979)
10. Pérez-Freire, L., Furon, T.: Blind decoder for binary probabilistic traitor tracing codes. In: Proc. First IEEE Int. Workshop on Information Forensics and Security. pp. 56–60. London, UK (Dec 2009)
11. Saito, M., Matsumoto, M.: A PRNG specialized in double precision floating point numbers using an affine transition. In: Proc. Eighth Int. Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing, MCQMC '08. pp. 589–602. Springer, Montréal, Canada (Jul 2008)
12. Skoric, B., Katzenbeisser, S., Celik, M.: Symmetric Tardos fingerprinting codes for arbitrary alphabet sizes. Designs, Codes and Cryptography 46(2), 137–166 (Feb 2008)
13. Tardos, G.: Optimal probabilistic fingerprint codes. In: Proc. 35th ACM Symposium on Theory of Computing. pp. 116–125. San Diego, CA, USA (2003), `http://www.renyi.hu/~tardos/publications.html`
14. Wu, M., Trappe, W., Wang, Z.J., Liu, K.J.R.: Collusion-resistant fingerprinting for Multimedia. IEEE Signal Processing Magazine 21(2), 15–27 (Mar 2004)