A Novel Accusation Model for Document Fingerprinting

Bettina Fazzinga[1], Sergio Flesca[2], Filippo Furfaro[2], and Elio Masciari[3]

[1] ICAR-CNR, Italy
{fazzinga,masciari,pontieri}@icar.cnr.it
[2] DIMES, University of Calabria, Italy
{flesca,furfaro}@dimes.unical.it

Abstract. Watermarking digital content is a very common approach leveraged by creators of copyrighted digital data to embed fingerprints into their data. The rationale of such operation is to mark each copy of the data in order to uniquely identify it. These watermarks are embedded in a suitable way to prevent their stripping or modification by users for illegal distribution of the copy. If a copy is illegally distributed by a pirate user (or a set of users referred as coalition) it can be identified by the distributor that can analyze the fingerprint and accuse as traitor the person in charge of that copy. Actions can then be taken against this user, to prevent further illegal distribution. Many approaches have been defined to obtain optimal fingerprinting code based on the well-known Tardos encoding. However, such approaches suffer a great limitation, i.e., when the fingerprinting code embedded in a document is too short, it is not useful for accusing a traitor in a trial, as the probability that s/he might be innocent is too high, nevertheless, the fingerprint code can be used to drive further investigation. To overcome this limitation, in this work we provide a simple yet powerful accusation scheme that can be applied for a widely used approach such phrase substitution[4] that prove itself to be effective in many real world applications when the length of the fingerprinting codes is too short to be used for accusing a traitor in a trial with the classical approaches.

1 Introduction

Protection of copyrighted contents is a crucial activity for digital content producers in order to avoid unauthorized use of the artifacts or worse in order to prevent sensible information to be stealth (e.g. private documents of an administrative board). A common solution is the unique identification of each copy by embedding some distinguishing features. This activity is usually known as *fingerprinting* (a.k.a. *watermarking*) and the embedded content is referred as *code*.

In order to make this process robust against possible malicious users attacks, it is mandatory to hide the positions where the code is embedded. Indeed, attacks can be performed by group of malicious users (referred in what follows as pirates), who compare their copies and identify the positions where they differ as a position of the embedded code. The latter is referred as coalition attack. If the coalition succeeds in this

SEBD 2018, June 24-27, 2018, Castellaneta Marina, Italy.

identification process, pirates can then arbitrarily change the code in these positions. For the purpose of designing proper protection strategies, we can assume however that they do not know the positions of the hidden code where the bits of their codes agreed and therefore they cannot alter these positions. This assumption is referred as the *marking condition*.

A (collusion resistant) fingerprinting code can be built by a randomized procedure to choose codewords (the code generation) and a tracing algorithm tailored for tracing one of the pirates based on all these codewords and the forged codeword read from the unauthorized copy made by the pirates.

Obviously, we should avoid two type of errors: 1) accusing an innocent user and 2) not accusing a pirate. In this respect, the tracing algorithm fails if it falsely accuses an innocent user or outputs no accused user at all. The above mentioned errors should occur with small probability.

This problem have been largely investigated in the literature and all the approaches proposed so far shares a common terminology that we introduce here in order to ease the reading of next sections.

More in detail, we briefly recall the following key terms:

- *Alphabet size.* The codewords are sequences over a fixed alphabet Σ. Usually, fingerprinting codes are built by leveraging the binary alphabet $\Sigma = \{0, 1\}$, however larger alphabets can be used thus the size Σ of the alphabet is an important parameter;
- *Codelength.* This parameter refers to the length of the codewords, usually denoted by n;
- *Number of users.* Usually denoted by N, it coincides with the number of codewords.
- *Pirate Coalition Size.* This parameter takes into account the actual size of the coalition that could be lower than the expected one (say it c), in such a case, the accusation algorithm should achieve a small error probability;
- *Error probability.* A code is ϵ-secure against a coalition of c pirates if the probability of the error of the accusation algorithm is at most ϵ for any set of at most c pirates performing an arbitrary pirate strategy to produce the forged codeword under the marking assumption;
- *Code rate.* The rate R of a fingerprinting code is computed as $R = \frac{log(N)}{n}$, where the logarithm is binary.

The goal of fingerprinting schemes is to find efficient and secure fingerprinting codes while taking into account the high cost of embedding every single digit of the code. This implies that fingerprinting codes should be short. However, in literature many proposal have been defined based on the seminal work of Tardos[9] that state many interesting theoretical results. Tardos fingerprinting is optimum as the code length that is sufficient to deal with n users, c pirates and an innocent safety guarantee bounded by ϵ is asymptotically minimum. Moreover, the accusation algorithm allows to detect traitors by looking only at the code they have been assigned to, disregarding both the codes assigned

Many fingerprinting algorithm guarantee a small probability of accusing an innocent even if the number of pirates is greater than the expected one. However, in that case, the probability of producing no accusation increases.

to other users and the type of attack that have been performed. It is worth noticing that in literature have been defined many other approaches that slightly outperforms Tardos scheme while having the same asymptotical complexity [7, 8]. Unfortunately, Tardos based fingerprinting are not effective in accusation processes when the leveraged code is too short. For instance, in the case that the code has to be embedded in a textual document by applying some modification of words, phrases or generally speaking tokens appearing in the text of the document as described in [2], and the document is about 20 pages long it is expected that the longest fingerprint that can be embedded is at most 200 bit long. In such a case, the Tardos accusation algorithm fails in accusing any user with a suitable probability of being guilty as in the case that the maximum coalition size is 2 and the desired probability of being guilty is 90% it requires a code of length at least 800 bits. This code length could be impractical in many scenarios.

In order to overcome the above mentioned limitations, new approaches have been proposed and one of the most interesting is *joint-decoding*. Joint-decoders compute the guilty probability for a *set* of users instead of a single one. A first proposal has been made in [6, 5], however those algorithms are tailored for small coalition and do not scale-up properly. This drawback occurs as the search space computation grows up exponentially w.r.t. the number of users (or the maximum expected number of users that we may conjecture that could form a coalition for spreading the pirated copy).

To ameliorate this problem, joint-decoding has been investigated from the theoretical viewpoint in order to define efficient approaches that work properly for real life situations. In this respect, a *Markov Chain Monte Carlo (MCMC)* based approach has been proposed in [1]. The proposed approach leverages Gibbs sampling for estimating the marginal probability that a user joined a coalition for generating a pirated copy. However, this approach turns to be ineffective for code length greater than 1024 bit due to the low quality of the probability estimation (as noted by the authors themselves).

The main limitation of the above mentioned Tardos based approaches is that they perform satisfactorily when dealing with image or video fingerprinting[3] while their use for textual documents turns to be ineffective. More in detail, textual document watermarking is prone to several types of attacks, even very simple ones like the so called *cut & paste* attack. This attack allows to completely strip the watermark and the corresponding fingerprinting code by simply extracting the text in the source document and inserting it in a brand new document. The latter cause the deletion of eventual watermark inserted in the source text. This type of attack causes the fingerprint of the pirated copy to be empty, thus avoiding any accuse to users by using Tardos based schemes.

In order to overcome such limitations, we propose a simpler but still effective accusation model based on *Metropolis-Hastings (MH)* sampling. Next sections are devoted to our proposal description.

1.1 Main Contributions

In this paper we address the fingerprinting code design by leveraging joint decoding strategies based on Metropolis-Hastings scheme. More in detail:

- we implemented an accusation scheme based on MH Joint-decoding, which is remarkably accurate even in condition where the code rate R is very low;

– we performed a deep experimental assessment of our approach that resulted quite effective w.r.t. the actual baseline for this kind of approaches.

2 Background on Tardos code construction and exploitation

2.1 Preliminaries

In this section, we briefly describe the fingerprinting scheme proposed in [10]. We first recall the basic assumption for this kind of encoding: pirates are not aware on the position where the code is embedded in the document, i.e. the marking condition is verified.

Definition 1 (Marking condition). *A fingerprint set of length m for n users over the alphabet Σ is an n by m matrix X over Σ. A coalition of users is a subset C of $\{1, \ldots, n\}$. A pirated copy $y \in \Sigma^m$ generated by a coalition C satisfies the marking condition w.r.t. a coalition C and fingerprint set X iff, for all positions $1 \leq i \leq m$, if all the values X_{ji} with $j \in C$ agree with some letter $s \in \Sigma$ then $y_i = s$.*

The following definition introduce some key notions about fingerprinting codes.

Definition 2 (Fingerprinting Code). *A fingerprint code of length m for n users over the alphabet Σ is a distribution over the pairs (X, σ), where X is fingerprint set of length m for n users and σ is an algorithm that takes a string $y \in \Sigma^m$ (the pirated copy) as input, and produces a subset $\sigma(y) \subseteq [n] = \{1, 2, \ldots, n\}$ (the set of accused users). For $\emptyset \neq C \subseteq [n]$, a C-strategy is an algorithm ρ that takes the submatrix of X formed by the rows with indexes in C as input, and produces a string $y = \rho(X) \in \Sigma^m$ as output. If for each X $y = \rho(X)$ satisfy the marking condition w.r.t. C and X we say that ρ satisfies the marking condition. We say that a fingerprint code is ϵ-secure against coalitions of size c, if for any coalition C of size $|C| \leq c$ and for any C-strategy ρ, the error probability*

$$P[\sigma(\rho(X)) = \emptyset \vee \sigma(\rho(X)) \subseteq C]$$

is at most ϵ.

2.2 Building the Code

Let n and c be positive integers, $0 < \epsilon < 1$ and let $k = \lceil log(1/\epsilon) \rceil$. We define the binary fingerprint code $F_{nc\epsilon}$ of length $m = 100c^2k$ for n users to be the following distribution over the pairs (X, σ).

(X, σ) is constructed in two phases. First, let p_i be independent, identically distributed random variables from $[t, 1-t]$ for all $1 \leq i \leq m$ obtained as follows. Let $t = \frac{1}{300c}$, $t' = \arcsin(\sqrt{t})$ and r_i be selected by picking uniformly at random a value in $[t', \pi/2 - t']$. p_i is choosen equal to $\sin^2 r_i$.

In the second phase, we select the code matrix X, by selecting each entry X_{ji} independently from the binary alphabet $\{0, 1\}$ with $P[X_{ji} = 1] = p_i$. Notice that independence of the entries X_{ji} holds only in the second phase. That is, two random variables X_{ji} and $X_{j'i}$ are positively correlated as both of them tend to be 1 if p_i is large.

When constructing the code *log* always denotes the natural logarithm.

2.3 Accusation Algorithm

The accusation algorithm σ is built by leveraging the values p_i and the matrix X, as follows. We define the n by m matrix U with entries

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

Let σ accuse user j on the pirated copy $y \in \{0,1\}m$ as input if

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

where $Z = 20ck$ is a threshold parameter. In other words, $\sigma(y)$ consists of the indices j for which the jth entry of Uy^T exceeds Z.

2.4 Error and Code Length Bounds

The following two theorems bound the error probabilities of the codes $F_{nc\epsilon}$. Theorem 1 bounds the "soundness error" of accusing an innocent user, while Theorem 2 bounds the "completeness error" of not accusing any guilty one. For both theorems $n \geq c \geq 1$ and $0 < \epsilon < 1$ are arbitrary.

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

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

Theorem 2 (Completeness). *Let (X, σ) be distributed according to $F_{nc\epsilon}$. Let $C \in [n]$ be a coalition of size $|C| \leq c$, and let ρ be any C-strategy satisfying the marking condition. We have*

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

Based on the two theorems above the following corollary holds.

Corollary 1. *The fingerprint code $F_{nc\frac{\epsilon}{n}}$ is ϵ-secure against coalitions of size c if $c \geq 4$. The length of this code is $O(c^2 log(n/\epsilon))$.*

3 A MH-sampler based Joint Decoder

In this section, we describe our encoding scheme, based on Metropolis-Hastings algorithm.

3.1 Metropolis-Hastings Samplers

Metropolis-Hastings (MH) algorithm aims at approximating a probability density function $F(x_1, \ldots, x_n)$, named target distribution, whose exact formulation is unknown, exploiting the knowledge of a computable function $P(x_1, \ldots, x_n)$, named proposal distribution, that is proportional to $F(x_1, \ldots, x_n)$. The result of an executions of a MH sampler is a sequence of samples. This sequence of samples, which is typically represented as an histogram, yields an approximation of $F(x_1, \ldots, x_n)$ as it is generated with the guarantee that the occurrences of each sample $s \in S$ are proportional to $P(x_1, \ldots, x_n)$ (thus to $F(x_1, \ldots, x_n)$). In a sense, S has a similar shape to F. A possible implementation of MH algorithm is reported below.

Algorithm 1 The MH sampler

Input: N output samples; B: number of samples for burn-in;
Output: a sequence of k accused users $P = [p_1, \ldots, p_k]$

1: $S = \emptyset$
2: generate an initial sample s
3: **for** $i = 0$ to $B + N$ **do**
4: $\quad s' \leftarrow s$
5: $\quad$ perturb s'
6: $\quad jitter = random()$
7: $\quad$ **if** $jitter \leq min(1, \frac{P(s')}{P(s)})$ **then**
8: $\quad\quad s = s$
9: $\quad$ **if** $k \geq B$ **then**
10: $\quad\quad$ Add s to S
11: **return** S

The algorithm works as follows. We first generate an initial sample s by random picking up from the n-dimensional domain of P. At each step i a new candidate sample s' is generated by random perturbation of s. The perturbation is performed by modifying each component of s in order to obtain a new point in the sampling space. In order to make the perturbation strategy effective, we need to have specific implementation for the context at hand, i.e., we need to take into account the semantic of each dimension for the sample being considered.

Once s' is computed, we compute the ratio $r = \frac{P(s')}{P(s)}$. The latter operation measures the variation of the target function F as it is proportional to P. Tus, r is leveraged for deciding if s' can be accepted as a new sample for F. If s' can be used it is queued to S. More in detail, this action is performed according to a probabilistic evaluation: we generate a random number $jitter \in [0, 1]$ and we decide to accept s' if and only if $jitter \leq min(1, \frac{P(s')}{P(s)})$. The latter implies that s' will be accepted if $F(s')$ value is greater than $F(s)$ otherwise we conditionally accept s' with a probability score whose value is as lower as $P(s')$ is lower than $P(s)$. If s' is not acceptable, we add s to S's queue. Intuitively enough, it means that the adopted sampling generates samples with

higher P values (thus higher F values) while samples with low P values have few occurrences in S or are excluded at all.

It is worth noticing that MH algorithm generates $B + N$ samples where B and N are input parameters, namely the number of samples you may want to *burn* before collecting the actual samples is S and the expexted cardinality of S. We need to perform an initial burn-in for the first samples generated by MH as the initial samples tend to be highly correlated with the initial samples thus they could be generated according to a different distribution w.r.t. the target one. For practical use, the burn-in is effective when at least 1000 iterations have been performed.

3.2 A MH sampler for Joint Decoding

Algorithm 2 describes our accusation strategy based on MH sampling.

Algorithm 2 The MH accusation algorithm

Input: A pirated copy y, the code matrix X, the probability array p, an integer $accU$, an integer $cMaxS$

Output: a sequence of k accused users $P = [p_1, \ldots, p_k]$

$cCoal = \emptyset, cPcoal = 0$
$ST = \emptyset$
for $i = 1$ to $burnIt + It$ **do**
    $tCoal = cCoal$
    if $0 < |tCoal| < cMaxS$ **then**
        $tCoal = genericUpdate(tCoal)$
    else
        if $|tCoal| = 0$ **then**
            $tCoal = addUser(tCoal)$
        else
            $tCoal = remUser(tCoal)$
    $tPCoal = computeProb(tCoal, y)$
    $jitter = random()$
    if $jitter < min(1, \frac{tPCoal}{cPcoal})$ **then**
        $cCoal = tCoal, cPcoal = tPCoal$
    if $i > burnIt$ **then**
        $update(ST, cCoal)$
return $bestPirates(ST, accU)$

Herein: $cCoal$ is the current coalition, $cPcoal$ is the probability of the current coalition, $tCoal$ is the generated coalition and $tPCoal$ is the probability of the generated coalition. Finally, ST is the set of generated coalition (that could contain duplicated elements).

The algorithm works in two phases. The first phase initializes the coalition set by generating $burnIt$ coalitions that will be discarded as explained in previous section.

The second phase generates it coalition that are added to ST. Each new coalition is generated by applying the following operation to the current coalition:

1. a random user not yet included in the coalition is added to it if the length of the coalition in not maximum;
2. a random user is removed from the coalition (if at least one user is in the coalition);
3. a random user of the coalition is replaced with a random user not included in the coalition.

More in detail, function *genericUpdate* updates the coalition by applying one of the above mentioned operations. Function *addUser* implements the third operation while *remUser* implements the second modification strategy.
Function *computeProb(tCoal,y)* computes the probability that a pirated copy y has been generated by coalition $tCoal$ whose users hold the codes included in matrix X_{tCoal} as follows:

$$computeProb(tCoal, y) = {p_{Coal}}^{|tCoal|-1} \prod_{i=1}^{m} p(X_{tCoal}[i], y[i]) \quad (1)$$

where $p(X_{tCoal}[i], y[i])$ is defined as follows. $p(X_{tCoal}[i], y[i]) = \frac{1}{3}$, if $\exists h, k$ s.t. $X_{tCoal}[i][h] \neq X_{tCoal}[i][k]$ and if no such h, k exists, and otherwise

$$p(X_{tCoal}[i], y[i]) = \begin{cases} p_u, & \text{if } y[i] = -1 \quad (a) \\ (1-p_u)p[i], & \text{if } y[i] = 1 \text{ e } X_{tCoal}[i] = 1 \quad (b) \\ p_u p[i], & \text{if } y[i] = 1 \text{ e } X_{tCoal}[i] = 0 \quad (c) \\ p_u(1-p[i]), & \text{if } y[i] = 0 \text{ e } X_{tCoal}[i] = 1 \quad (d) \\ (1-p_u)(1-p[i]), & \text{if } y[i] = 0 \text{ e } X_{tCoal}[i] = 0 \quad (e) \end{cases} \quad (2)$$

Specifically, the definition $computeProb(tCoal, y)$ assumes that the probability that $tCoal$ generates y is the combination of two different independent contributions:

- the fact that $|tCoal|$ users formed a coalition, and
- the probability that the user in $tCoal$ generated y

The first contribution is represented by the factor ${p_{Coal}}^{|tCoal|-1}$ in Equation (1), where p_{Coal} is the probability that two randomly picked users cooperate. The second contribution is represented by the product, for each bit i of the pirated copy y, of the probability that the users in $tCoal$ generated the i-th bit of y. The latter probability is assumed to be p_u if the i-th bit of y is unreadable (case (a) of Equation (2)), where p_u is a constant representing the probability that a bit of the code is accidentally discovered by the pirates which made it unreadable or swithced its value even in the case that they all share the same value for that bit. In cases $(b), (e)$ of Equation (2), the probability that the users in $tCoal$ generated the i-th bit of y is assumed proportional to the probability the the bit is not accidentally discovered and the probability that that value of the bit was generated by the Tardos code generator. Finally, in cases $(c), (d)$ of Equation (2) the probability that the users in $tCoal$ generated the i-th bit of y is assumed proportional to

the probability the the bit is accidentally discovered and the probability that that value of the bit was generated by the Tardos code generator. Summarizing, given a generic bit of the pirated copy, the probability that a coalition generated that bit is equal to $\frac{1}{7}3$ if the corresponding bits of the coalition are different from one another, otherwise it is assigned the value p_u or $1 - p_u$ multiplied by the probability $p(i)$ (or $(1 - p(i))$ that it has been generated by the Tardos fingerprint generator.

4 Experiments

In this section, we will describe the extensive experimental evaluation that we performed in order to assess the validity of our joint-decoding technique. More in detail, we will first describe the dataset generation then we will describe the standard approach we compare to and finally, we will describe the performance metrics that we computed by our experiments.

4.1 Data Set Generation

Testing the accuracy of fingerprinting codes requires a deep experimental evaluation, thus, we generated a dataset composed of 1.296.000 test cases that have been obtained as described in the following.

The dataset generator takes as input the following parameters:

- N: is the number of users that are going to receive the target document;
- L: is the length of the code generated for each user;
- A: is the attack type (in our study we consider Random, Majority and Minority attacks that will be explained below);
- C: is the pirate coalition size;
- $Pchange$: is the probability that coalition users will change a bit. Each bit can be changed independently by other bits.

For each bit of the code, every attack perform a code hiding by setting it to -1 with $Pchange$ probability. After this step, if all the pirates have the same bit they emit this common bit. If a given bit is not the same for all the pirates the different attack types work as follows:

1. *Random*: the bit is set to 1 or 0 with the same probability $\frac{1}{2}$;
2. *Majority*: the bit is set to the value that occur in more than 50% of users. If the bit value occurrences are tied we apply Random strategy;
3. *Minority*: the bit is set to the value that occur in less than 50% of users. If the bit value occurrences are tied we apply Random strategy;

Once defined the possible attack types we can generate a test case instance by performing the steps below:

1. We generate the Tardos codes of length L for N users;
2. We generate a pirate coalition of size C;

3. We generate an attack of type A performed by the pirates in the coalition that produce a pirated copy y.

As mentioned above, we generated a huge amount of instances. More in detail:

- We used the code length in the set $\{100, 150, 200, 250, 300, 350\}$;
- Number of users in the set $\{50, 100, 150, 200, 250, 300\}$;
- $Pchange$ values $\{0.01, 0.03, 0.05, 0.07, 0.09\}$;
- Coalition size ranging from 1 to 8;
- Random, Majority and Minority attacks.

For each combination of the above values we generate 300 test cases, thus the total number of cases is $300 * 3 * 6 * 6 * 5 * 8 = 1.296.000$.

4.2 Term of comparison and accuracy measurement

In order to compare our results with a reliable baseline, in this section we compare the performance of WFinger with the Tardos accusation algorithm. However, since we consider very short codes the choice of an adeguate probability threshold for the Tardos algorithm is very hard. Indeed, for a code of 200 bits, assuming that the maximum coalition size is 3 and requiring a probability of accusing an innocent lower than 20% no user will be accused. Hence, we had to consider a slightly modified version of the Tardos accusation algorithm, which accuses the user having the gretest value of $\sum_{i=1}^{m} y_i U_{ji}$ (see Section 2.3).

Finally, in order to perform a fair comparison with this modified version of the Tardos accusation algorithm, we considered as accuracy measure for both WFinger and Tardos the accuracy at one (named Acc in the figures), that is the fraction of the experiments where the accused user (which is the first one returned by WFinger and Tardos, respectively) is a guilty user.

4.3 Effectiveness Results

In Fig. 1 we first report the accuracy values obtained by our algorithm (denoted in the following as *WFinger*) and the modified Tardos algorithm (denoted in the following as *Tardos*) by averaging the results on the three types of attack. Fig. 1(a),(b),(c) and (d) report, respectively, the accuracy w.r.t. the coalition size (a), the length of the code (b), the number of users (c) and the probability of changing a bit of the code (d). As a remarkable result, we observe that, on average, the accuracy of *WFinger* that is greater than *Tardos* by a 20%.

As regards the accuracy w.r.t. the coalition size (Fig. 1(a)), it is worth noticing that the accuracy of both algorithms decreases as the coalition dimension increases as expected. However, for *WFinger* it is over 80% until the coalition size is 6, while for *Tardos* this result is obtained only for coalition size lower than 3. The better accuracy of our approach is compelling also when the coalition size further increase to 7 or 8 pirates.

As regards the accuracy w.r.t. the length of the code (Fig. 1(b)), it is easy to see that, for code length greater than 100 bits the accuracy of *WFinger* is greater than 80% and

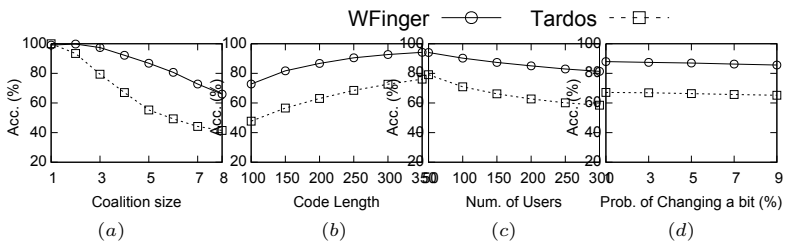

Fig. 1: Average Accuracy Results

for code length greater than 200 bits it is greater than 90%, while *Tardos* never reach an 80% accuracy.

As regards the accuracy w.r.t. the number of users (Fig. 1(c)), we can note that even for a huge number of users (greater than 250) the accuracy of *WFinger* is greater than 80% while *Tardos* never reach an 80% accuracy.

As regards the accuracy w.r.t. the probability of changing a bit of the code (Fig. 1(d)), we point out tha this analysis give an hint about the *robusteness* of our approach. More in detail, we introduce in our evaluation a parameter that states the possibility that a random bit of the code is accidentally changed. The probability of this event is very low, so we assign this parameter values lower than 10%. Also in this case *WFinger* (accuracy greater than 90%) performs better than *Tardos* (accuracy lower than 70%) for all probability values.

In order to provide a more detailed view of the obtained results, we report in Fig. 2, 3 and 4 the results obtained considering a single type of attack. It is easy to see that, also when considering a single type of attack, the accuracy of *WFinger* always overcome *Tardos* for every experimental setting.

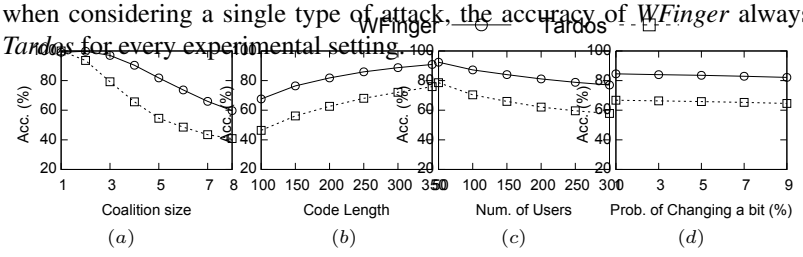

Fig. 2: Accuracy Results for the Random Attack

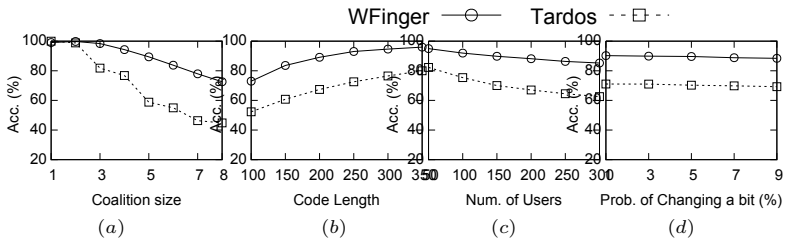

Fig. 3: Accuracy Results for the Majority Attack

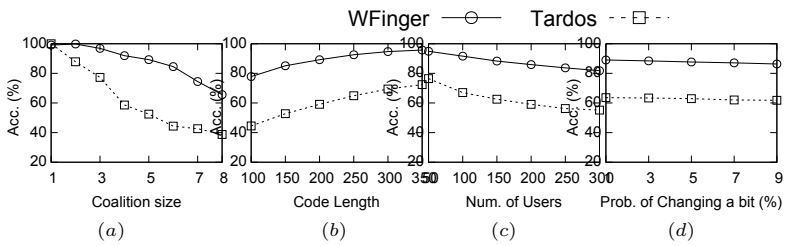

Fig. 4: Accuracy Results for the Minority Attack

5 Conclusion

In this paper we investigated the design of a fingerprinting code based on Metropolis-Hastings scheme. We implemented an encoding scheme that is quite accurate even if the code length is very short (300 bits). The deep experimental evaluation we performed, confirmed the validity of our approach in several stressing test bench.

References

1. Teddy Furon and Mathieu Desoubeaux. Tardos codes for real. In *2014 IEEE International Workshop on Information Forensics and Security, WIFS 2014, Atlanta, GA, USA, December 3-5, 2014*, pages 24–29, 2014.
2. Mohan S. Kankanhalli and K.F. Hau. Watermarking of electronic text documents. *Electronic Commerce Research*, 2(1):169–187, 2002.
3. A. Kot. Watermarking, data hiding and image forensic. In *2005 5th International Conference on Information Communications Signal Processing*, pages nil94–nil94, 2005.
4. S. H. Low, N. F. Maxemchuk, J. T. Brassil, and L. O'Gorman. Document marking and identification using both line and word shifting. In *INFOCOM '95. Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Bringing Information to People. Proceedings. IEEE*, pages 853–860 vol.2, 1995.
5. Koji Nuida. Short collusion-secure fingerprint codes against three pirates. In *Information Hiding - 12th International Conference, IH 2010, Calgary, AB, Canada, June 28-30, 2010, Revised Selected Papers*, pages 86–102, 2010.
6. Koji Nuida, Satoshi Fujitsu, Manabu Hagiwara, Takashi Kitagawa, Hajime Watanabe, Kazuto Ogawa, and Hideki Imai. An improvement of discrete tardos fingerprinting codes. *Des. Codes Cryptography*, 52(3):339–362, 2009.
7. Boris Skoric, Stefan Katzenbeisser, and Mehmet Utku Celik. Symmetric tardos fingerprinting codes for arbitrary alphabet sizes. *Des. Codes Cryptography*, 46(2):137–166, 2008.
8. Boris Skoric, T. U. Vladimirova, Mehmet Utku Celik, and Joop Talstra. Tardos fingerprinting is better than we thought. *IEEE Trans. Information Theory*, 54(8):3663–3676, 2008.
9. G. Tardos. Optimal probabilistic fingerprint codes. In *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 116–125, 2003.
10. G. Tardos. Optimal probabilistic fingerprint codes. *Journal of the ACM*, 55(2), 2008.