

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

---

SCUOLA DI SCIENZE  
Corso di Laurea in Informatica - 8028

**Testing Attacks  
on Structural Text  
Watermarking Techniques**

**Relatore:**  
Chiar.mo Prof.  
Danilo Montesi

**Corelatore:**  
Dott.  
Fabio Bertini

**Presentata da:**  
Simone Branchetti

**Controrelatore:**  
Chiar.mo Prof.  
Maurizio Gabbrielli

**Sessione II  
Anno Accademico 2022/2023**

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
<b>2</b>	<b>State of the art</b>	<b>9</b>
2.1	Zero Watermarking Techniques . . . . .	10
2.2	Image based Watermarking . . . . .	10
2.3	Syntactic based Watermarking . . . . .	11
2.4	Semantic Watermarking . . . . .	11
2.5	Structural Techniques . . . . .	12
<b>3</b>	<b>Main Techniques</b>	<b>13</b>
3.1	Grayscale Watermarking . . . . .	13
3.2	Whitespace Watermarking . . . . .	17
3.3	Homoglyph Watermarking . . . . .	20
3.4	Combining the different methods . . . . .	24
3.5	Embedding Capabilities . . . . .	25
<b>4</b>	<b>Dataset</b>	<b>26</b>
4.1	Finding the right Dataset . . . . .	26
4.2	TenTen Family - Most Languages . . . . .	27
4.3	NYTC - English . . . . .	27
<b>5</b>	<b>Robustness Testing</b>	<b>30</b>
5.1	Types of Attacks . . . . .	30
5.1.1	Partial Copy and Paste . . . . .	31
5.1.2	Insertion: . . . . .	31
5.1.3	Deletion . . . . .	32
5.1.4	Replacement . . . . .	32
5.1.5	Retyping . . . . .	33
5.1.6	Reformatting/Recoloring . . . . .	33
5.2	Test Execution . . . . .	34

<b>6</b>	<b>Experimental Results</b>	<b>40</b>
6.1	64 Bits of watermark . . . . .	41
6.2	128 Bits of watermark . . . . .	43
6.3	256 Bits of watermark . . . . .	45
<b>7</b>	<b>Conclusion</b>	<b>47</b>
<b>8</b>	<b>Bibliography</b>	<b>48</b>

# List of Figures

3.1	Embedding of a 63bits long watermark in the first seven characters of the string. . . . .	15
3.2	Embedding of 24 bits of watermark in the first whitespace of a string. . .	18
3.3	The two characters, in this font, share the same glyph but have a different Unicode code. Homoglyph watermarking exploits this property to hide information in a text. . . . .	20
3.4	Image that show the procedure for embedding bits of information into a text using homoglyphs. Green highlighted characters in the first text are those that have a homoglyph. Based on the corresponding watermarking bit, the character is changed to its homoglyph (red highlight) or maintained.	22
5.1	Partial Copy and Paste attack example of size equal to 20% of the text. In this case the affected character count is 40. . . . .	31
5.2	Insertion attack example of size equal to 20% of the text. In this case the affected character count is 40. . . . .	32
5.3	Deletion attack example of size equal to 20% of the text. In this case the affected character count is 40. . . . .	32
5.4	Replacement attack example of size equal to 20% of the text. In this case the affected character count is 40. In this example all characters have been replaced with *. . . . .	33
5.5	Recoloring attack example of size equal to 20% of the text. In this case the affected character count is 40. Whitespaces affected have been highlighted so as to make them more apparent. . . . .	34

# List of Tables

3.1	Table of a subset of whitespaces, their corresponding Unicode code and a coloured example to illustrate their differences. . . . .	20
3.2	This table shows which levels of character encoding are affected by the structural watermarking techniques presented in this paper. . . . .	25
3.3	Table that shows how many characters (whitespace or not) are needed to hide at least once different lengths of watermark with all possible combinations of the techniques presented in this thesis. . . . .	25
4.1	Table that shows the number of scanned articles, their character count, whitespace count, number and their total and average grouped by year of publication. . . . .	29
6.1	Homoglyph Watermarking Experimental Results table. The Recoloring attack is not included because it doesn't affect this technique at all. . . .	40
6.2	Number of positive 64 bits watermark extractions for the Deletion attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	41
6.3	Number of positive 64 bits watermark extractions for the Insertion attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	41
6.4	Number of positive 64 bits watermark extractions for the Replacing attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	42
6.5	Number of positive 64 bits watermark extractions for the Recoloring attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	42
6.6	Number of positive 64 bits watermark extractions for the Partial Copy and Paste attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	42
6.7	Number of positive 128 bits watermark extractions for the Deletion attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	43

6.8	Number of positive 128 bits watermark extractions for the Insertion attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	43
6.9	Number of positive 128 bits watermark extractions for the Replacing attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	44
6.10	Number of positive 128 bits watermark extractions for the Recoloring attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	44
6.11	Number of positive 128 bits watermark extractions for the Partial Copy and Paste attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	44
6.12	Number of positive 256 bits watermark extractions for the Deletion attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	45
6.13	Number of positive 256 bits watermark extractions for the Insertion attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	45
6.14	Number of positive 256 bits watermark extractions for the Replacing attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	46
6.15	Number of positive 256 bits watermark extractions for the Recoloring attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	46
6.16	Number of positive 256 bits watermark extractions for the Partial Copy and Paste attack on 100 examples of 200 characters texts with each of the main techniques and their combinations. . . . .	46

# List of Algorithms

1	GRAYSCALE EMBEDDING . . . . .	16
2	GRAYSCALE VALIDATION . . . . .	16
3	WHITESPACE EMBEDDING . . . . .	19
4	WHITESPACE VALIDATION . . . . .	19
5	HOMOGLYPH EMBEDDING . . . . .	23
6	HOMOGLYPH VALIDATION . . . . .	24
7	DELETION ATTACK . . . . .	35
8	INSERTION ATTACK . . . . .	36
9	REPLACEMENT ATTACK . . . . .	36
10	RECOLORING ATTACK . . . . .	37
11	PARTIAL COPY AND PASTE ATTACK . . . . .	38
12	TESTING PROCESS . . . . .	39

# Chapter 1

## Introduction

In recent times, certifying content paternity is becoming an interesting challenge to overcome, and a pressing one at that. Every day more and more content is published on the internet in social media sites, cloud sharing services or websites and this content becomes easy pickings for people who want to steal it and make it theirs. When this sort of “content theft” occurs it is very difficult to trace the stolen piece of content back to its original author. Another linked issue is that publishers and content creators are interested in a lightweight and easy to use solution to this problem in order to protect their works against different types of attack. The most common solution to this problem is Digital Watermarking, defined as the process of digitally embedding a certain amount of information (the watermark) into a piece of multimedia content such as a picture, video or written document. The most apparent form of Digital Watermark is a photographer’s name printed on top of their picture in order to protect it from improper use, as it would carry the watermark if it were to be copied and pasted in another website, thus linking the picture to its owner. Different pieces of media need different watermarking techniques because, for example, an image has different properties from a piece of text. Speaking of the latter, text watermarking is especially difficult, because text as a medium has a low embedding rate for external information: for example if one inserts a new word into a piece of text, any attacker would be able to understand where the watermark is and how to delete it. This means that text watermarking is done through other more discreet means like converting text to an image and watermarking the result, using synonyms or with the help of structural watermarking methods. This thesis will focus on text watermarking because of its abundance in literature and the inherent omnipresence of text on the internet which make it almost omnipresent in blogs, social networks, forums and scientific journal’s websites. In the next chapters we will explain what text watermarking is, how it is done and, most importantly, what attacks can be used to break every technique that will be mentioned and how likely it is for such an attack to be successful. A particular focus will be placed on three structural watermarking techniques: Homoglyph based watermarking, Grayscale based



watermarking and Whitespace color based watermarking, with an in-depth look into their robustness to several types of attacks.

# Chapter 2

## State of the art

Watermarking a piece of text poses lots of different problems because text as a medium has low embedding bandwidth (meaning that there's not much room to hide information compared to an image, where every pixel can hide many bits of watermark) and only allows a restricted number of alternative syntactic and semantic permutations [1]. Of course, there are many text watermarking methods already used in practice with a large amount of literature backing them so, among them, a characterization of text watermarking techniques has emerged. Many papers like [2] and [3] define the basic characteristics for a generic watermarking technique as:

- **Readable or Detectable:** if the watermark is readable, the user can clearly see it. Conversely, it is detectable if a function can identify it but the user alone can't.
- **Visible or Invisible:** A visible watermarking method is visually perceptible by the user. On the contrary, the method is invisible if it is hidden in the original digital content and is not noticeable by the user. A visible watermark may be non-readable if a user can visually detect it but cannot read its content.
- **Blind or Non Blind:** If the original digital content is not needed in the extraction process, the watermarking is blind. Otherwise, the watermarking process belongs to the non blind category.
- **Simple or Multiple:** If a watermark can be applied only once the watermarking is simple. A multiple watermarking method means that a payload of data can be embedded more than one time in the same document without affecting the whole process.
- **Fragile, Semi-Fragile, Robust:** A fragile watermark is detectable and can be altered or erased, thus, it is used for integrity authentication; a robust watermark is detectable and not erasable and it is most suitable for copyright protection. A semi-fragile watermarking is suited for content authentication.

These characteristics make up the baseline for defining all watermarking techniques. Keeping with the example of the photographer's name on their picture as the watermark, we can identify it as a Readable, Visible, Blind, Multiple and Fragile technique. The focus of this thesis is on Text Watermarking and, even though all Text watermarking techniques can be defined using the previously mentioned characteristics, the literature regarding these is especially prolific, with techniques being grouped in different categories:

## 2.1 Zero Watermarking Techniques

Zero watermarking techniques don't hide any information inside the content, instead they extract characterizing information from a text, and then store this information into an Intellectual Property Right (IPR) database [4]. They are counted among watermark techniques even though using this method the association between the content and the author does not rely on the watermark, but on the proof from a trusted authority. Using these techniques, the amount of data that the IPR has to store is greatly reduced if compared to a method that stores the entire text for paternity purpose but, on the other hand, Zero watermarking has a clear weakness: the text and identity of the owner must be deposited to a third party and this can lead to privacy issues [1].

## 2.2 Image based Watermarking

A text watermarking technique is called Image based if the process of embedding involves transforming the text into an image (by scanning or taking a screenshot of it) and then watermarking the resulting image. These techniques are some of the most explored in literature, with the first dating back to the mid-nineties [5, 6]. For example, in a grayscale image of a text, a large amount of information can be hidden in the image by tuning the luminance of some pixels based on the watermark data [7]. Another example of this category is this article from Kim and Oh [8] that shows an image watermarking technique based on altering image histograms in order to hide information into an image. Some other techniques implement a shift in text elements of an image, like a line being slightly higher or lower [5, 9], or tweaking the space between different words [10, 11], changing serifs and strokes or elongating single characters [5, 12, 13].

Even though Image based techniques are abundant, they all share two key weaknesses: the resulting file must be kept in an image file format like JPEG or PNG and it can therefore be shared only through avenues that allow it like as a physical printed page or a fax machine. The second vulnerability is one that is shared by almost all categories of text watermarking: if an attacker were to retype the whole text by reading it from the watermarked image the information hidden inside of it would be lost. Because of

this strong focus on printed paper for the protected medium, Image based watermarking may become less relevant in the future due to the increasing pressure from society on the printed press in favor of digital alternatives as the preferred way to consume textual information.

## 2.3 Syntactic based Watermarking

Instead of focusing on the shape and positioning of words, Syntactic techniques alter the very structure of the text in order to embed a watermark. The first step to this process is the construction of a syntactic tree for a sentence, after which a series of operations like clefting, passivization or activization are applied in order to encode the watermark bits [14]. Clefting is the process of transforming a simple sentence into a complex one using clefting particles like “what”, “it” or similar [15], for example the sentence “I ate steak” can be transformed into a cleft sentence like “Steak is what I ate” (“what” clefting) or “It is steak that i like” (“it” clefting). Activization and passivization are two faces of the same coin, whereas the first transforms a passive sentence into an active one, the second does the opposite. Other morpho-syntactic transformations exist that preserve the original meaning of the sentence like switching from “ ’s ” to “of” and vice-versa when talking about possession [16]. The major limit of these techniques is their low embedding capabilities because in situations where text length is restricted like SMS messages or social network posts there is often no room to apply any of the techniques just described. Another disadvantage is that these techniques can change the meaning of a sentence because different syntactic forms don’t always preserve idiomatic interpretations [17].

## 2.4 Semantic Watermarking

Semantic Watermarking exploits the similar meanings of different words to replace word with their synonyms [18]. This results in non-blind watermarking techniques because the original text is required in order to understand which synonyms have been put into the text. This particular semantic approach can be mixed with previously described syntactic techniques like Topkara et al. do in [19] in order to obtain an higher embedding capacity. Other semantic techniques use sentences’ presuppositions in order to add to a sentence, embedding some information in the meantime. A presupposition is a fact that must be true for the sentence to make sense; for example if the sentence to modify is “Paul walks his dog”, the implicit information is that Paul has a dog and, by making that information explicit, or in other cases removing it, it is possible to hide a watermark into a piece of text. Of course, these techniques are not perfect and, in fact, they are quite reliant on language and on the correctness of the text to watermark, in addition to the fact that

the content of the text can be heavily altered in order to apply a semantic watermark.

## 2.5 Structural Techniques

Structural methods include all those methods that do not alter the text content but only its structure, meaning the underlying representation or the visual features. These have recently emerged as methods that embed watermark or hidden payloads by changing the underlying encoding of symbols or adding invisible symbols, without actually altering the readable content of the text. The Unicode standard has several different symbols for whitespaces, some of different width, others practically identical. By putting many of these whitespace symbols at the end of a paragraph, or by filling an empty line, relatively long payloads have been hidden in Microsoft Word documents [20]. A similar technique has been effectively applied to watermark Arabic language text: by using a different Unicode whitespace between words depending on the bits of the watermark's binary representation a payload of data can be hidden in the text [21]. A more recent method uses multiple ASCII white spaces to embed a covert message [22] for PDF steganography instead. This last technique works on justified text and is able to embed 4 bits for each host line, where a host line is a line with at least 9 normal spaces and 3 wider spaces. Apart from whitespaces, the Unicode standard also provides some totally invisible symbols, which are coded as zero-width whitespaces. These symbols, together with whitespaces, have been exploited to hide hidden messages in text [23] and to watermark HTML pages [24, 25]. As mentioned earlier, these methods have the important advantage of keeping the original content unaltered without transforming the text to an image, or relying on an external database. The above structural methods are blind, meaning that the original text is not needed in order to extract the watermark. This, together with the easiness of removing multiple whitespaces, makes these approaches fragile in both malicious and benign attacks. This is particularly true for methods that use consecutive whitespaces and whitespaces before or after the whole text, because it has been proven that many digital platforms and social media automatically remove them [26]. A similar problem can also occur through selection for copy and paste: selection may easily exclude the white portion where the watermark is embedded. Apart from whitespaces, homoglyphs and invisible characters, some image based techniques where lines or words are slightly shifted without altering the text content [27, 28] have been also considered as structural methods [29].

# Chapter 3

## Main Techniques

The main techniques in this thesis are all structural techniques and, in this section, we will explain each of them in detail. If other techniques were just mentioned by name, these will get a proper explanation complete with algorithms for both encoding and decoding information into a text, all for the interest of later explaining how attacks can influence negatively the recovering process of an embedded watermark. The thing they have in common is that, while they all embed watermark bits into a text in a different way, the original watermark is always created by encrypting the full text with an encryption algorithm the user chooses. Different encryption algorithms produce different watermark lengths, but all of the following watermarking methods use this resulting array of bits in the same way regardless of its length. In particular, in [30], watermark lengths of 64, 128 and 256 bits are used, among others, with the respective hashing algorithms being SipHash [31], SHA128 and SHA256 [32] so, to keep our measures coherent our tests will use the same watermark lengths. The key for every algorithm used is inputted by the user and it is the final proof of paternity for the document so, for example, even if two documents are watermarked with the same technique, if the two users choose different keys the resulting documents will be different in and of themselves.

### 3.1 Grayscale Watermarking

The first structural watermarking technique we want to present is called Grayscale Text Watermarking or just Grayscale Watermarking in short. We described its functioning in deeper detail in a paper from 2022 cited here [30].

This technique is detectable, invisible, blind, fragile and preserves content and length of the watermarked text.

It works by embedding watermark bits into the shades of grey of the characters in a text document. We define a grayscale as a palette of colours that starts with pure black and ends with pure white, usually represented as a series of hexadecimal codes (e.g., #171717

is a shade of grey very closely resembling pure black #000000). After the process is complete each character has a different shade of grey and, as such, stands for different parts of the original bit sequence of the watermark.

The main problem that needs to be solved is that not all shades of grey are indistinguishable when confronted with pure black text. To solve this issue, Grayscale watermarking uses a two-step process to ensure maximum bit embedding per character while maintaining the outcome invisible to the majority of people. The first step is thresholding the maximum possible shades of grey we use to embed a sequence of bits in a character. This reduces the number of bits we can embed in a single character but improves indistinguishability compared to using the whole palette of grey or the whole colour spectrum. The specifics of these thresholds are in the article [30] but in this thesis we report that a survey made with 255 people on the visibility of different shades of gray showed that the best thresholds to use in this application are from #070707 to #272727. We will use one of these two when making examples on this technique.

The second step is the separation of the hexadecimal components of the character's colour (i.e., *red*, *green*, *blue*). If we used the entire colour spectrum and a maximum threshold of #070707, there would be several colours represented with smaller numerically values that would be visually further away from black and more visible to the user (e.g., the numerical value #00ffff representing the colour yellow is numerically less than #070707). The solution is to consider a separate threshold of #07 for each colour component and then reconstruct the full colour to use for the character reassembling the components.

As shown in Figure 3.1, the shade of each component encodes a portion of the sequence of bits of the watermark and helps to form the grey shade that respects the defined threshold. In particular, the grayscale is chosen converting a part of the watermark bits into a hexadecimal number lower than the current colour component's threshold. Considering the previous #070707 threshold, the maximum number of embeddable bits into a single colour component is three because the binary value 111 is equal to 07 in hexadecimal.

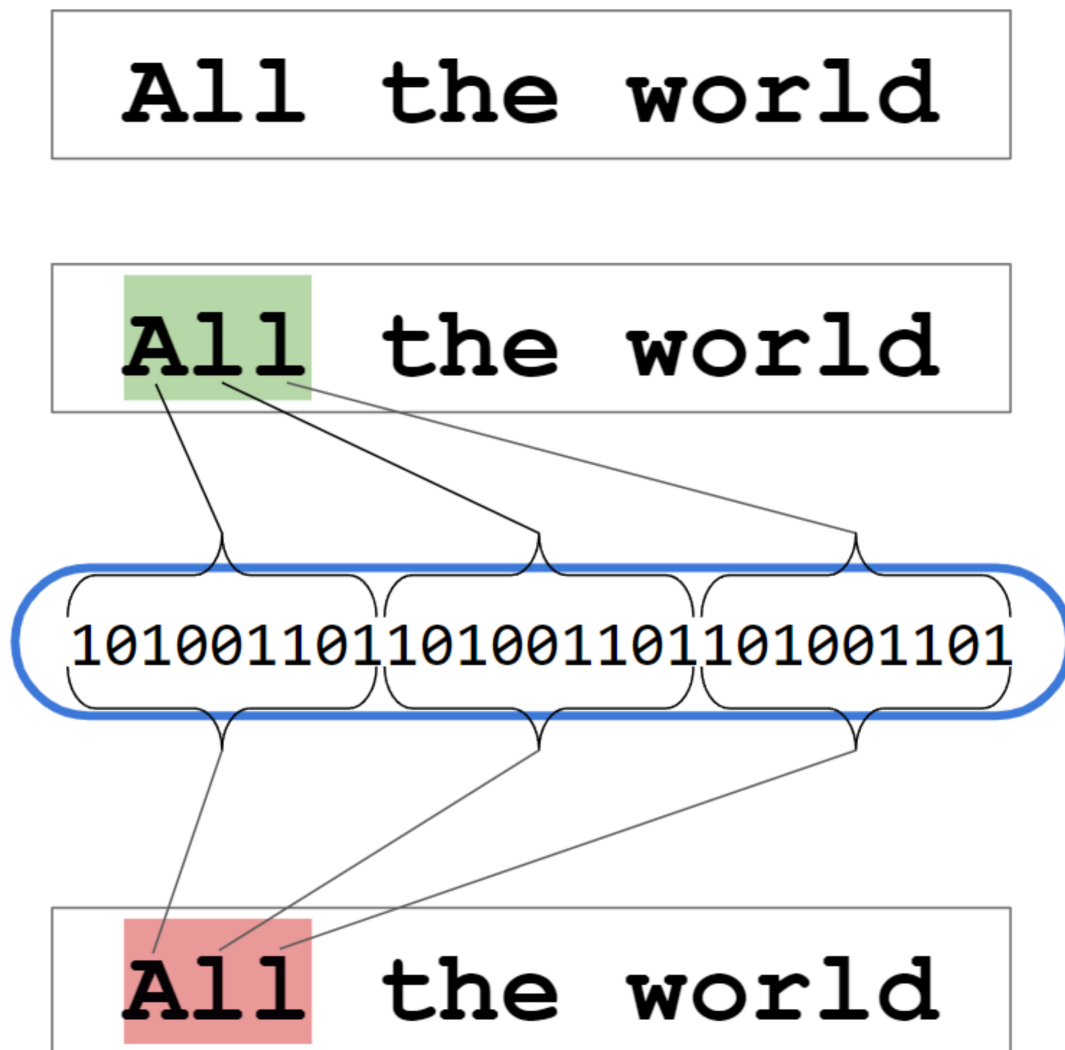


Figure 3.1: Embedding of a 63bits long watermark in the first seven characters of the string.

The following algorithms show a possible implementation of this technique with respects to both the embedding and the recovering of a watermark composed of a string of bits. In this example we will use a threshold of #272727 for our embedding.



---

**Algorithm 1** GRAYSCALE EMBEDDING

---

```
1: // The SHA256 hashing algorithm returns a 256bits watermark.
2: function grayscaleEmbedding(origText, userPassword)
3:   watermark  $\leftarrow$  SHA256(origText, userPassword)
4:   threshold  $\leftarrow$  #272727
5:   for each char  $\in$  origText do
6:     redBits  $\leftarrow$  toGreyShade(pop(watermark, 3))
7:     greenBits  $\leftarrow$  toGreyShade(pop(watermark, 3))
8:     blueBits  $\leftarrow$  toGreyShade(pop(watermark, 3))
9:     colour  $\leftarrow$  redBits + greenBits + blueBits
10:    wtmText  $\leftarrow$  wtmText + colouring(char, colour)
11:   return wtmText
```

---

This is the first set of algorithms for embedding and retrieving a watermark in a structural technique so we can highlight some of the common themes we will see between all of them: the embedding algorithms will scan every character of the text in order to find the correct ones to modify, the watermark bits are embedded repeatedly into the text so that every part of the text is equally protected and the validation algorithms need the original text to validate a watermark, making all of these techniques non-blind.

---

**Algorithm 2** GRAYSCALE VALIDATION

---

```
1: // This function wipes out the watermark.
2: function grayscaleClean(wtmText)
3:   for each char  $\in$  wtmText do
4:     text  $\leftarrow$  text + colouring(character, #000000)
5:   return text
6:
7: // The extracted watermark is compared with the new one
8: // generated using a new password.
9: function gyscaleValidation(wtmText, newPassword)
10:  origText  $\leftarrow$  clean(wtmText)
11:  newWtm  $\leftarrow$  SHA256(origText, newPassword)
12:  for each char  $\in$  wtmText do
13:    colourString  $\leftarrow$  string(getColor(char))
14:    red, green, blue  $\leftarrow$  extractComponent(colourString)
15:    recoveredWtm  $\leftarrow$  recoveredWtm + red + green + blue
16:  return (newWtm == recoveredWtm) ? True : False
```

---

## 3.2 Whitespace Watermarking

This structural watermarking technique is similar in concept to the previously described Grayscale watermarking technique but with a substantial difference: this time, only whitespace characters are considered for the recolouring process.

The process is simple and allows for more embedding in a single character. A whitespace character is invisible to the naked eye but, in every collaborative text editor, it can be coloured nonetheless with the full hexadecimal scale. We can use this by taking a large chunk of watermark bits (24) and converting it into a hexadecimal code that is applied to the first whitespace character in the text as its colour. This process is repeated for every whitespace character, looping over the watermark as many times as necessary to provide enough bits to this process. Because of this repetition, the technique protects text to a sentence level with only 3 whitespace characters needed to hide a 64 bits watermark. With this technique, for every whitespace character present in a text we can hide 24 bits of information by removing all colour limits we had previously with Grayscale. The similarity between these methods of watermarking can be seen in figure 3.2 that demonstrates how the technique works.

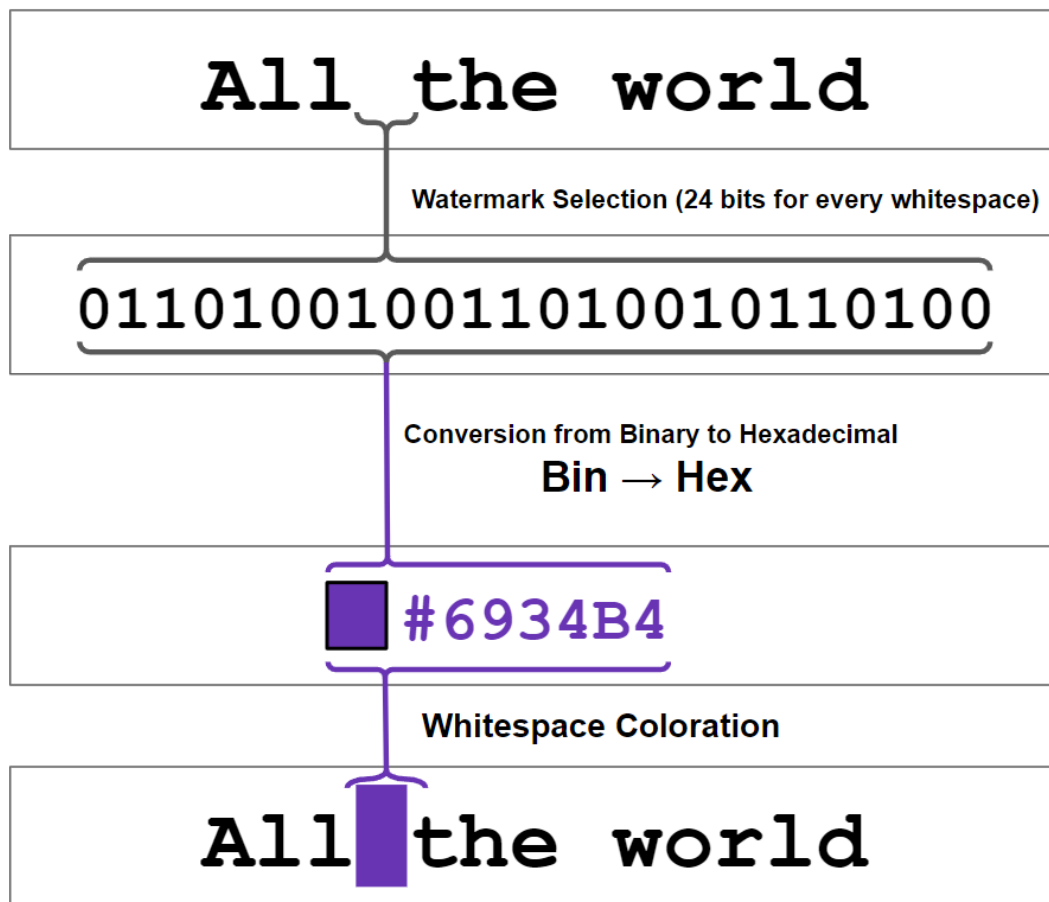


Figure 3.2: Embedding of 24 bits of watermark in the first whitespace of a string.

The resulting technique is, just like Grayscale watermarking, invisible, detectable, fragile, non-blind and content and length preserving. No character is added, all the whitespaces are just coloured. Inevitably, the number of available characters for this technique is lower than the other ones we present in this paper but the trade off is the huge embedding capability of this technique. To find the average split of characters to whitespaces we analysed the full text of over 129 thousand New York Times articles from the NYT Annotated Corpus [33]. For each article, its full text has been analysed and each character has been counted as either a whitespace or a normal character. The end results show that only around 20% of characters in a text are whitespace characters. The following algorithms show a possible implementation of this technique with respects to both the embedding and the recovering of a watermarked composed of a string of bits.

---

**Algorithm 3** WHITESPACE EMBEDDING

---

```
1: // The SHA256 hashing algorithm returns a 256bits watermark.
2: function spaceEmbedding(orgText, userPassword)
3:   watermark  $\leftarrow$  SHA256(orgText, userPassword)
4:   for each whitespace  $\in$  orgText do
5:     colour  $\leftarrow$  toHexColour(pop(watermark, 24))
6:     wtmText  $\leftarrow$  wtmText + colouring(whitespace, colour)
7:   return wtmText
```

---

Embedding and validation for Whitespace Watermarking are very similar in concept to the corresponding algorithms for Grayscale Watermarking but they are further simplified and streamlined because using the full hexadecimal color space we are not restricted in the number of bits we can hide in an affected character. The trade off for this increased embeddability is, of course, the increased rarity for whitespaces compared to visible characters.

---

**Algorithm 4** WHITESPACE VALIDATION

---

```
1: // This function wipes out the watermark.
2: function spaceClean(wtmText)
3:   for each whitespace  $\in$  wtmText do
4:     text  $\leftarrow$  text + colouring(whitespace, #000000)
5:   return text
6: // The extracted watermark is compared with the new one
7: // generated using a new password.
8: function spaceValidation(wtmText, newPassword)
9:   origText  $\leftarrow$  clean(wtmText)
10:  newWtm  $\leftarrow$  SHA256(origText, newPassword)
11:  for each whitespace  $\in$  wtmText do
12:    colourString  $\leftarrow$  string(getColor(whitespace))
13:    recoveredWtm  $\leftarrow$  recoveredWtm + colourString
14:  // If the two passwords were equal they generated the same
15:  // watermark, proving the paternity of the text.
16:  return (newWtm == recoveredWtm)? True : False
```

---

We use the term “whitespace” and not simply space because the Unicode standard has a family of characters that serve to put a blank space between two others. Every single one of them has its own uses and the table below shows some of them with their relative code. The characters are coloured for ease of reading and comparison.

White space	Unicode	Example
Space	0x0020	The <span style="background-color: #ccccff;"> </span> Example
En Quad	0x2000	The <span style="background-color: #ccccff;">   </span> Example
Three-per-em	0x2004	The <span style="background-color: #ccccff;">    </span> Example
Four-per-em	0x2005	The <span style="background-color: #ccccff;">     </span> Example
Punctuation Space	0x2008	The <span style="background-color: #ccccff;">   </span> Example
Thin Space	0x2009	The <span style="background-color: #ccccff;"> </span> Example
Narrow No-Break Space	0x202f	The <span style="background-color: #ccccff;"> </span> Example
Medium Mathematical Space	0x205f	The <span style="background-color: #ccccff;">   </span> Example

Table 3.1: Table of a subset of whitespaces, their corresponding Unicode code and a coloured example to illustrate their differences.

### 3.3 Homoglyph Watermarking

When typing a text, every character has its own glyph, or visual representation, based on the font used. A character has a homoglyph if there’s another character in Unicode that shares the same glyph but has a different underlying code. For example the character “C”, which is the uppercase third letter of the English alphabet, has a homoglyph in the character “Ĉ”, the roman numeral for one hundred. This property provides a useful avenue for structural text watermarking. The homoglyph based watermarking technique is described in this paper [1] however, just like with Grayscale watermarking, this thesis will contain a short summary to explain its functioning.

To understand how this watermarking method works, one first has to understand how we can use homoglyphs to hide information in a text. The following image provides a graphical display of how two characters with different codes can have the same glyph and thus be homoglyphs.

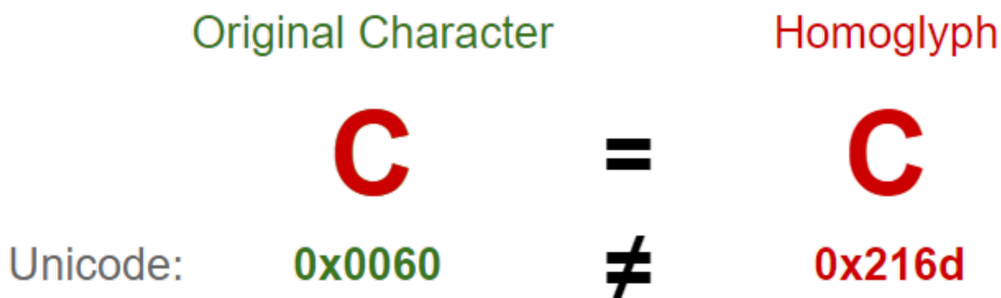


Figure 3.3: The two characters, in this font, share the same glyph but have a different Unicode code. Homoglyph watermarking exploits this property to hide information in a text.

A particular subclass of characters is whitespaces: every whitespace has, by the definition, the same glyph, the only difference is the length. In table 3.1 that we used in the previous subsection we have identified a subset of 8 whitespace characters. This subset can be mapped to all the possible binary digits from 000 to 111. By doing this we can encode 3 bits of information using different whitespace homoglyphs. This approach extends the embeddability of the technique and, as [1] proves, does not change the text visually any more than other structural techniques. The following image shows how characters are hidden in a text using this technique: for every character in a text that has a homoglyph, one bit of watermark is taken then, if the bit is 1 the character gets swapped for its homoglyph. We apply the same procedure with whitespaces but we substitute them with the three bits resulting from a lookup to the table we constructed before.

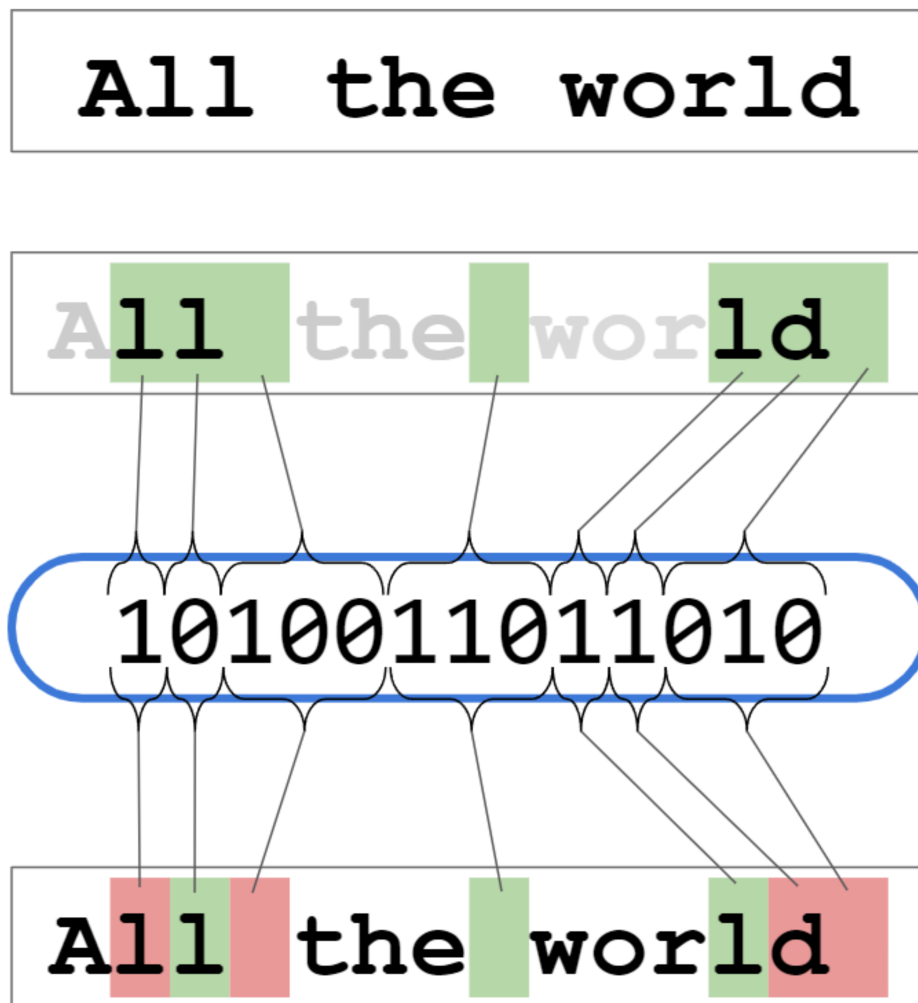


Figure 3.4: Image that show the procedure for embedding bits of information into a text using homoglyphs. Green highlighted characters in the first text are those that have a homoglyph. Based on the corresponding watermarking bit, the character is changed to its homoglyph (red highlight) or maintained.

Homoglyph Watermarking relies heavily on the set of homoglyph being used: a text that uses a language written in a Latin script like Italian or English can benefit from Homoglyphs found in the Cyrillic alphabet but languages that use a different writing system like Korean can not. The algorithms below show a possible way to implement Homoglyph Watermarking Embedding and Validation with a Homoglyph set created for Latin scripts comprised of Normal characters and Whitespaces.

---

**Algorithm 5** HOMOGLYPH EMBEDDING

---

```
1: function homoglyphEmbedding(orgText, userPassword)
2:   wtmText  $\leftarrow$  ""
3:   // List of confusable original symbols
4:   Originals  $\leftarrow$  {U+002c, U+002d, U+002e,...}
5:   // List of confusable duplicate symbols
6:   Duplicates  $\leftarrow$  {U+a4f9, U+2010, U+a4f8,...}
7:   // List of whitespace confusable characters
8:   Spaces  $\leftarrow$  {U+0020, U+2002, U+2005,...}
9:   Confusables  $\leftarrow$  Originals  $\cup$  Spaces
10:  GetDuplicate : Originals  $\rightarrow$  Duplicates
11:  GetSpace : {000, ..., 111}  $\rightarrow$  Spaces
12:
13:  // The SHA256 hashing algorithm returns a 256bits watermark.
14:  watermark  $\leftarrow$  SHA256(orgText, userPassword)
15:  for each char  $\in$  orgText do
16:    if char  $\in$  Confusables then
17:      if char  $\in$  Spaces then
18:        char  $\leftarrow$  getSpace(pop(watermark, 24))
19:      else
20:        bit  $\leftarrow$  pop(watermark, 1)
21:        if bit = 1 then
22:          char  $\leftarrow$  getDuplicate(char)
23:        wtmText  $\leftarrow$  wtmText + char
24:  return wtmText
```

---

It goes without saying that the Validation algorithm only works if the same set of Homoglyphs is used when recovering a watermark so we repeated the same set of Homoglyphs in this algorithm too.



---

**Algorithm 6** HOMOGLYPH VALIDATION

---

```
1: // List of confusable original symbols
2: Originals  $\leftarrow$  {U+002c, U+002d, U+002e,...}
3: // List of confusable duplicate symbols
4: Duplicates  $\leftarrow$  {U+a4f9, U+2010, U+a4f8,...}
5: // List of whitespace confusable characters
6: Spaces  $\leftarrow$  {U+0020, U+2002, U+2005,...}
7: Confusables  $\leftarrow$  Originals  $\cup$  Spaces
8: GetDuplicate : Originals  $\rightarrow$  Duplicates
9: GetSpace : {000, ..., 111}  $\rightarrow$  Spaces
10: Watermark  $\leftarrow$  ""
11: function homoglyphValidation(wtmText, newPassword)
12:   for each char  $\in$  wtmText do
13:     if char  $\in$  Spaces then
14:       Watermark = Watermark + GetSpace(char)
15:     else if char  $\in$  Originals then
16:       Watermark = Watermark + "0"
17:     else if char  $\in$  Duplicates then
18:       Watermark = Watermark + "1"
19:   return Watermark
```

---

### 3.4 Combining the different methods

All of our different structural watermarking techniques work on a particular aspect of character encoding and these levels are shown in table 3.2. These techniques are not mutually exclusive and any combination of them can be used in the same text to protect it further. The two colouring techniques in particular are weak to recolouring, which is defined as the act of colouring a text once it has been already watermarked with Grayscale or Whitespace Colouring. This weakness can be overcome by combining them with our Homoglyph approach that is completely immune to this attack. Vice versa, our homoglyph technique has a lower embedding rate than our colouring techniques so, to protect smaller parts of the text, mixing them together can prove useful. The way to combine the different techniques is by applying them to each character they affect when they appear in the text, using Homoglyphs before any coloring techniques so any replaced character retains the correct color, for example as soon as a whitespace character appears we watermark it first with the Homoglyph technique and then with the Whitespace technique using the same watermark sequentially.

Watermarking technique	Layer	Example
Grayscale & Whitespace	Color layer	All the world
	Font	<i>All the world</i>
Homoglyphs	Characters	All the world
	Bits	01000101...


High Level  
  
 Low Level

Table 3.2: This table shows which levels of character encoding are affected by the structural watermarking techniques presented in this paper.

### 3.5 Embedding Capabilities

Before proceeding with the testing, an important piece of information to know is how many bits of watermark each of the previously mentioned technique embeds in a single character. Something to keep in mind is that all three of the previously mentioned watermarking methods are finegrained, meaning that they embed the watermark repeatedly into the same text and, by boasting really high embedding capabilities, they can protect the text in a finer detail than most other watermarking techniques.

Method ( <i>threshold</i> )	Length		
	64 bits	128 bits	256 bits
Grayscale ( <i>#272727</i> )	16	31	62
Homoglyphs	101	198	357
Whitespace	17	34	67
Grayscale ( <i>#272727</i> ) + Homoglyphs	12	23	46
Grayscale ( <i>#272727</i> ) + Whitespace	8	16	32
Homoglyphs + Whitespace	15	29	58
Grayscale ( <i>#272727</i> ) + Homoglyphs + Whitespace	8	15	30

Table 3.3: Table that shows how many characters (whitespace or not) are needed to hide at least once different lengths of watermark with all possible combinations of the techniques presented in this thesis.

# Chapter 4

## Dataset

When looking at a written text, from a structural point of view not all characters are created equal. At first glance, we can divide all the characters in **visible characters** or **whitespaces** depending on if the glyph associated with the character is visible or not under normal circumstances. Because the main techniques presented in this paper all have different targets for their manipulations, we need a method to understand how many characters there are in a text, how frequent whitespaces are and how effective different watermarking techniques can be.

### 4.1 Finding the right Dataset

The first issue is defining what constitutes a “normal” text or, in other words, what collection of texts can give us the best representation of what the average text looks like. First and foremost, we must decide on the language and alphabet more suited to this application because different languages use different alphabets and this has important effects on the techniques presented. We can, of course, color every glyph from every alphabet supported in a common text editing software, finding homoglyphs in other alphabets can be more difficult. The challenge here arises because of the large number of symbols languages like certain languages tend to have which make it hard to find a comprehensive set of homoglyph that would make Homoglyph Watermarking work just as well as it is doing with the Latin alphabet. Fortunately, the problem of having a collection of texts in a specific language for testing purposes is in no way a new one and the principal solution is that of corpora. A corpus is a collection of texts that are usually purposefully collected, structured and catalogued for research purposes. This thesis will focus on two of these: the TenTen Corpus family which has a Corpus for most languages and collects web texts and the New York Times Corpus which collects articles from the titular newspaper.

## 4.2 TenTen Family - Most Languages

The TenTen Corpus Family is a project by Jakubíček et al. [34] aimed at producing corpora for different languages by crawling the internet and categorizing different website contents with metadata born in 2013. TenTen is built upon the Sketch Engine, which is a tool that analyzes billions of texts to produce, in this case, a corpora from web content. Corpora in this family are all built using the same flowchart:

- Texts are crawled from the Internet using a web spider designed for linguistic purposes.
- Texts are cleaned to remove navigation links, advertisements etc.
- Tokens are generated using a tokenization process on the texts.
- Using a language filter, longer sentences in different languages are eliminated, while foreign words like movie titles are kept in the text.
- An apposite tool deletes duplicate content at the paragraph level.
- Content with poor quality and spam is removed.
- Each text is classified into a genre (writing style) and a topic (a category).
- A lemmatizer and a tagger tool lemmatize and tag parts of speech.
- The user can check the finished corpus before publishing it.

This schematized process can produce a comparable corpora (corpora consisting of texts from the same domain in more languages) in any language chosen, with most of the major world languages being represented in this family like Arabic, Chinese, English, French, German, Italian, Japanese, Korean, Portuguese, Russian and Spanish.

Ultimately, this is an interesting set of corpora that showcase with effectiveness a method with which to create a corpus for a language. The issue for us is the need for an account to access the corpora and/or the creating tool and the unpredictable nature of creating a corpus using seed URLs and a crawler means that we will have no idea until the end of the process about what content is or is not in the created corpus.

## 4.3 NYTC - English

The last option, and the one chosen in this thesis is the New York Times Corpus [33], already used when testing Homoglyph Watermarking, which is a collection of almost 2 million articles from 1987 to 2006 of the New York Times. This corpus is suited for this research and free to use for these types of application. The articles within it are

catalogued as XML files that contain a large amount of information about each article in the daily edition of the NYT. The majority of that information is of no use here but the one field that we analysed is `full_text` that, as the name implies, contains the full text of any given article. A Python script we wrote for this purpose combed through 1'791'891 articles that had the `full_text` tag and counted every character and whitespace present. With a final count of more than 5 billion visible characters and almost 1 billion whitespaces scanned, we compiled the results and performed some statistical analysis on them.

The average article had 2802 visible characters and 544 whitespaces which means that for every visible character we can count 0.19 whitespaces when thinking in terms of watermark bits embedded per character. The techniques presented in this paper work either on visible characters like GTW and HBW or on whitespace characters WTW so having a coherent unit of measure for all of them helps with comparing the efficacy of the techniques. From this point onward, all performance metrics will be calculated and expressed assuming we are watermarking the average text and all characters that a technique ignores will be added to the resulting metric to keep the measurement as fair as possible. For example in the text “All the world” we count 3 whitespaces and 11 visible characters and if we were to delete 10% of this text it would be 2 characters chosen at random, regardless of what the watermarking technique applied to it is.

<b>Year</b>	<b>Character count</b>	<b>Whitespace Count</b>	<b>Article Count</b>
1987	270'612'216	51'450'406	106'104
1988	270'572'825	51'655'946	104'161
1989	264'180'289	50'423'406	101'760
1990	259'222'470	49'542'292	97'663
1991	222'823'312	43'317'424	83'995
1992	219'521'115	42'569'113	81'361
1993	214'460'525	41'628'894	77'653
1994	211'063'756	41'102'431	73'257
1995	226'378'041	44'231'326	83'783
1996	226'979'997	44'346'019	77'819
1997	231'505'524	45'067'168	84'406
1998	257'161'463	50'074'104	88'179
1999	263'542'092	51'419'165	90'111
2000	276'671'454	54'182'011	93'369
2001	268'401'276	52'402'883	95'142
2002	273'897'490	53'456'283	95'925
2003	268'054'422	52'523'328	92'829
2004	262'092'827	51'086'897	90'054
2005	259'565'516	50'558'312	88'786
2006	262'799'467	51'177'801	85'534
<b>TOT</b>	<b>5'009'506'077</b>	<b>972'215'209</b>	<b>1'791'891</b>
<b>AVG</b>	<b>250'475'304</b>	<b>48'610'760</b>	<b>89'595</b>

Table 4.1: Table that shows the number of scanned articles, their character count, whitespace count, number and their total and average grouped by year of publication.

This corpus is perfect for our purposes: it's open-source, easily accessible, was already used to test Homoglyph Watermarking so we can maintain testing homogeneity and it is composed of complete texts from a reliable source with rigid publication requirements so each text in it is high quality. When testing the techniques in this thesis for robustness the texts on which the tests will be performed will be chosen from this corpus.

# Chapter 5

## Robustness Testing

This chapter will focus on exploring how to test Grayscale and Whitespace Watermarking plus their combinations with each other and Homoglyph Watermarking for vulnerabilities to different attacks. It is divided in sections that explain the methods with which the tests were performed and the results of said tests. The important thing to remember is that we will focus on Structural Text Watermarking Techniques for our testing, meaning that attacks directed at videos and pictures are not considered.

### 5.1 Types of Attacks

In order to test how well the main watermarking techniques fare against different types of attack we must of course specify what attacks we will be testing for. The attacks we considered have the purpose to delete completely or partially the watermark embedded in a text in order to avoid detection. This can be achieved in many different ways depending on the technique used and the kind of watermarking employed (attacks useful against image-based watermarking may not be effective for structural watermarking methods, for example). All attacks in this section can be used on other text watermarking techniques to measure their robustness to some of the most common attacks that happen to a text. The list of attacks we developed is partly derived from literature [35, 36] and partly derived from the need to highlight possibly problematic areas in the watermarking techniques presented.

In order to make sure that the techniques are properly challenged, the attacks will have to target all of their properties, with the finegrained property first and foremost. Because the watermark will be embedded multiple times into the text, the best chance an attack will have to disrupt them all, while still resembling something a human would reasonably do, is for each of the attacked characters to be chosen at random.

With this premise, the following subsections form a list of attacks that were considered possible on the techniques presented in this article, with an example of each of the at-

tacks on an example test with 20% of the text affected by that attack. All images in this section are captured from a Google Documents document, as such, they are subjected to change if the software itself changes.

### 5.1.1 Partial Copy and Paste

A very common scenario where the content of a text file, or part of it in this case, is copied and pasted into an attacker's file. This attack is insidious because usually, only a small amount of text is copied from the protected text to another in order to avoid plagiarism checks. In our case the attack consists on randomly choosing a point in the text and trying and recover the watermark in a portion of text starting with that point (or ending with it if it is too close to the end of the text); in the figure below we can see how a partial copy of 20% of the text affects it. As the example shows, in our tests it is not a requirement for the copied portion of text to be coherent or have any sense because it would be too difficult to implement and it would limit the amount of attacks possible and their extension.

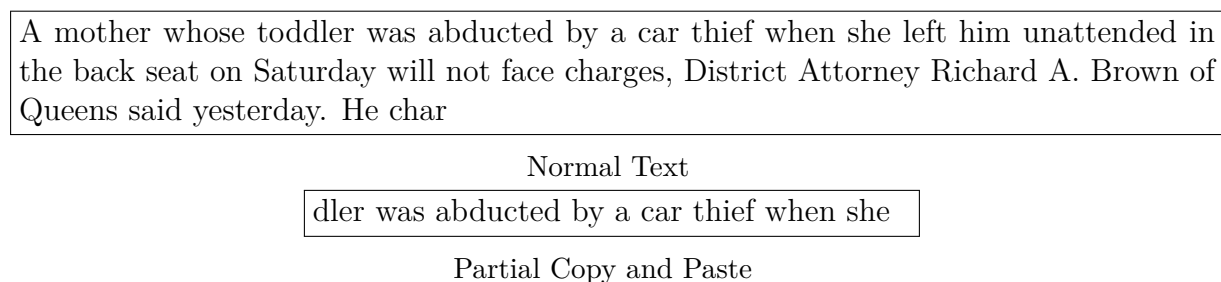


Figure 5.1: Partial Copy and Paste attack example of size equal to 20% of the text. In this case the affected character count is 40.

### 5.1.2 Insertion:

Insertion attacks work by randomly adding new words or characters in the watermarked text [37], with the goal of altering the watermark. We can think of this attack in the simplest form as an attacker adding the word “not” in front of a sentence to alter the meaning of the text. An insertion of 20% of the text's length changes it drastically as we can see from the example below:



A mother whose toddler was abducted by a car thief when she left him unattended in the back seat on Saturday will not face charges, District Attorney Richard A. Brown of Queens said yesterday. He char

Normal Text

A m\$oth\$er w\$ho\$se tod\$dler was\$ abd\$uc\$ted by a\$ car t\$hi\$ef wh\$en she lef\$t him unatt\$ended i\$n \$the ba\$ck se\$at \$on \$atur\$day \$will \$not fac\$e charg\$es, Dis\$\$trict A\$tto\$rn\$ey Rich\$a\$rd A. Br\$own of \$Quee\$ns sa\$id yesterd\$ay. He ch\$ar

Insertion

Figure 5.2: Insertion attack example of size equal to 20% of the text. In this case the affected character count is 40.

### 5.1.3 Deletion

In this attack, some parts of the text are removed [29]. If the deleted portion is part of the watermark, the watermark may be destroyed or completely removed [37]. An attacker may delete part of the text to make it fit a certain narrative or to hide some information while retaining the general meaning but, just like with the Insertion attack, if we delete 20% of characters at random the text is unrecognisable.

A mother whose toddler was abducted by a car thief when she left him unattended in the back seat on Saturday will not face charges, District Attorney Richard A. Brown of Queens said yesterday. He char

Normal Text

A mthrwos odler as bdutedby ar thef wen sh lf him unttendd i the bak sat on Saurday will nt facechrges, Dstrct Aatorne RichardA. Bown f Quens sid ysterdy. Hechar

Deletion

Figure 5.3: Deletion attack example of size equal to 20% of the text. In this case the affected character count is 40.

### 5.1.4 Replacement

In a replacement attack, a set of words or characters in the watermarked text are replaced with other words or characters [29, 37]. It can be considered as a deletion attack followed by an insertion attack in the same location. An example could be, in a gendered language like Italian, changing the suffix of a noun, an adverb or a pronoun in order to make it masculine instead of feminine.

A mother whose toddler was abducted by a car thief when she left him unattended in the back seat on Saturday will not face charges, District Attorney Richard A. Brown of Queens said yesterday. He char

Normal Text

A m\*ther whose\*toddl\*r was abducted \*y \* car thief\*when \*he\*left\*him una\*t\*nd\*d\*n\*t\*\* \*ack seat o\* \*aturday will\*not face c\*arge\*, Dis\*r\*ct\*At\*or\*ey Ric\*ar\*\*\*. Brown\*of\*Queens\*said \*esterday. He char

Replacement

Figure 5.4: Replacement attack example of size equal to 20% of the text. In this case the affected character count is 40. In this example all characters have been replaced with \*.

### 5.1.5 Retyping

In a retyping attack, a malicious user retypes the text in a different file or platform. Structural methods are all fragile to retyping by definition. In fact, because the structural methods embed the watermark by altering the layout such as formatting features, spaces, Unicode, and ASCII encodings without altering the content, retyping a text will always destroys a watermark. There will be no examples in this subsection because a retyping attack doesn't modify the text, the attacker just manually retypes it. The first obvious failure of all structural techniques is this Retyping attack: when typing a text after looking at it, we cannot expect the attacker to also manually color each character and, by virtue of its innate invisibility, homoglyphs will also be lost in the process. We can therefore mark all of these as a failure and this type of attack will not show up in any result tables for brevity.

### 5.1.6 Reformatting/Recoloring

Reformatting attacks including the change of formatting features of the text such as fonts or color. Copy and paste, retyping, and OCR (*Optical Character Recognition*) have been also considered sub-types of formatting attacks [29]. In particular, Reformatting refers in this case only to a Recoloring of the text. Fonts do have an impact on Homoglyph Watermarking but they only affect its visibility, not the capacity to extract a valid watermark from the text so, because this is outside of this thesis's scope, we will refer to Recoloring attacks only. In this spirit, the next example shows a Recoloring Attack where 20% of the original characters (whitespaces and visible characters) have been recolored to red.

A mother whose toddler was abducted by a car thief when she left him unattended in the back seat on Saturday will not face charges, District Attorney Richard A. Brown of Queens said yesterday. He char

Normal Text

A mother whose toddler was abducted by a car thief when she left him unattended in the back seat on Saturday will not face charges, District Attorney Richard A. Brown of Queens said yesterday. He char

Recoloring

Figure 5.5: Recoloring attack example of size equal to 20% of the text. In this case the affected character count is 40. Whitespaces affected have been highlighted so as to make them more apparent.

One important measure to decide is what percentage of the text must be affected by the attacks just described. Bashardoost et al. in [38] assert that 10% of a text’s length is a high attack size for any watermarking technique to endure but, in order to push the limits of techniques that embed a watermark into a text in a finegrained manner [1] we will explore extreme scenarios where up to 60% of a text is affected by an attack. A notable outlier is the Partial Copy and Paste attack where increasing the portion of text that is copied and pasted makes it *easier* rather than harder to conserve at least one repetition of the watermark. In this case we will test portions of text from 10% down to just 5% of the original length.

## 5.2 Test Execution

To execute our tests we created a Python software that gathers the first 200 characters from 100 articles of the New York Times Corpus [33] chosen randomly from all years, to make sure the techniques work even if the “style” of writing changes, in order to form our starting texts. Our techniques embed a watermark in very few characters like we previously saw in 3.3 so, because we want to have a larger number of articles as a sample we decide to limit the amount of characters from each of them to 200 which is more than enough for the watermark to be embedded several times. We don’t have to worry about Homoglyph Watermarking because its robustness has already been tested in [1] and in this application we will only be combining it with the other techniques. For the watermark, a function selects 64, 128 or 256 random bits that serve as a watermark then, each original text gets watermarked with the technique tested, some for of attack get performed that disturbs the watermarked text and finally a function tries to extract a full watermark from the perturbed text. After 100 repetitions we count how many successful watermark extractions happened and that gives us an indication of the robustness of

that particular technique to a particular type of attack expressed in percentages. As for the singular tests, we tested Deletion, Insertion, Replacement, Recoloring and Copy and Paste with every attack altering a certain percent of the text. Following are the algorithms for the attacks that were used.

---

**Algorithm 7** DELETION ATTACK

---

```
1: // Delete a percentage of the text one random character at the time.
2:
3: originalText ← “In a new twist in the dispute over who first...”
4:
5: function deletion(originalText, percent)
6:
7:   deletionNumber ← length(originalText) * percent / 100
8:
9:   for (i ← 0; i < deletionNumber; i ← i + 1) do
10:     n ← randomBetween(0, length(originalText) - 1)
11:     popFrom(originalText, n)
12:
13:   return originalText
```

---

The Deletion algorithm deletes a percent of the text’s length one character at the time. A human attacker would never perform a Deletion attack like this but we wanted to develop attacks that had the highest chance possible of disrupting a watermark. Because the watermark is embedded multiple times into the same text, deleting some words like a human attacker might do would just leave watermarks dispersed in the untouched sections of text, which would make the techniques seem incredibly robust. The random approach of this version of the attack lets us consider the worst possible scenario to test the techniques by pushing them to their limit.

---

**Algorithm 8** INSERTION ATTACK

---

```
1: // Insert characters into a text up to a percentage of it one random character at the
   time.
2: originalText ← “In a new twist in the dispute over who first...”
3: function insertion(originalText, percent)
4:   // approximately one in five characters in a text is a whitespace.
5:   characters ← [‘a’, ‘b’, ‘c’, ‘d’, ‘ ’]
6:   insertionNumber ← length(originalText) * percent / 100
7:   for (i ← 0; i < insertionNumber; i ← i + 1) do
8:
9:     n ← randomBetween(0, length(originalText) − 1)
10:    char ← randomBetween(0, 4)
11:    insertInto(originalText, characters[char])
12:
13:   return originalText
```

---

---

**Algorithm 9** REPLACEMENT ATTACK

---

```
1: // Replaces characters into a text up to a percentage of it one random character at
   the time.
2: originalText ← “In a new twist in the dispute over who first...”
3: function replacement(originalText, percent)
4:   // approximately one in five characters in a text is a whitespace.
5:   characters ← [‘a’, ‘b’, ‘c’, ‘d’, ‘ ’]
6:   replacementNumber ← length(originalText) * percent / 100
7:   for (i ← 0; i < replacementNumber; i ← i + 1) do
8:
9:     n ← randomBetween(0, length(originalText) − 1)
10:    char ← randomBetween(0, 4)
11:    replaceRandom(originalText, characters[char])
12:
13:   return originalText
```

---

Insertion and Replacement share the same array called **characters** in the algorithms which serves to mimic how a real text is written. When talking about the New York Times Corpus we stated that there are 0.19 whitespaces per visible character in the “average” text so, in order to reproduce a truly random Insertion or Replacement, we pick the character to Insert or Replace from an array composed of 4 visible characters (letters from “a” to “d”) and one whitespace, respecting the ratio the original texts have.

---

**Algorithm 10** RECOLORING ATTACK

---

```
1: // Recolors characters from the text up to a percentage of it one random character
   at the time.
2:
3: originalText ← “In a new twist in the dispute over who first...”
4: function recoloring(originalText, percent)
5:   recoloringNumber ← length(originalText) * percent / 100
6:
7:   for (i ← 0; i < recoloringNumber; i ← i + 1) do
8:
9:     n ← randomBetween(0, length(originalText) - 1)
10:
11:    // This example recolors with pure black. The results will be the same
12:    // with every other color if not better.
13:
14:    recolorCharacter(originalText[n], “#ff0000”)
15:
16:   return originalText
```

---

Recoloring is a type of attack unique to Grayscale and Whitespace Watermarking and it can be simulated by transforming each character into a tuple containing the character and its color. The attack then simply consists of changing the color of the character to something like pure red that would ruin our watermark. Using colors inside the Grayscale threshold for this process would incur the miniscule risk that a character gets recolored with the same color he had. The red color used here doesn't eliminate the same risk for Whitespace Watermarking but the risk is so small that it was not worth to avoid it, seeing as getting pure red from 24 bits has a probability of  $1/2^{24}$ .

---

**Algorithm 11** PARTIAL COPY AND PASTE ATTACK

---

```
1: // Copies a contiguous portion of the text.
2: //This test consists in recovering the watermark from that copied portion.
3:
4: originalText  $\leftarrow$  "In a new twist in the dispute over who first..."
5:
6: function copypaste(originalText, percent)
7:
8:   replacementNumber  $\leftarrow$  length(originalText) * percent / 100
9:   n  $\leftarrow$  randomBetween(0, length(originalText) - 1)
10:
11:   // Taking a slice of 10 or less percent from a text
12:   // we have to make sure we don't go over the length of the text.
13:
14:   if ((n + percent) > (length(originalText) - 1)) then
15:     originalText  $\leftarrow$  substring(originalText, n - percent, n)
16:   else
17:     originalText  $\leftarrow$  substring(originalText, n, n + percent)
18:
19:   return originalText
```

---

The Copy and Paste attack is singular in that a larger attack size only makes recovering watermarks easier because more characters are preserved. In this case we select a portion of the original text and treat that as the text where to recover the watermark, simulating the situation where a small part of the original text is copied and pasted into a new text that is not watermarked.

All of these functions were used to test the watermarking techniques in an algorithm like the following that cycles over all 100 texts, it watermark them, perturbs them, tries to recover the watermark and counts how many of them actually contained a recoverable watermark. In this example we will use Deletion as the perturbation technique and Grayscale as the watermarking technique but the process would be the same no matter what perturbation or watermarking technique is chosen.

---

**Algorithm 12** TESTING PROCESS

---

```
1: // Watermarks 100 texts, perturbs them and tries
2: // to recover a complete watermark, counting the successful recoveries.
3: key ← “password”
4: positive ← 0
5: negative ← 0
6:
7: texts ← [...] // Array of 100 texts, each of them is a string.
8:
9: for each text ∈ texts do
10:   //Create the watermark.
11:   //SHA256 is used here as an example of a hashing function. It returns 256 bits.
12:   watermark ← SHA256(text, key)
13:
14:   //Watermark the text. Grayscale Watermarking is used here as an example
15:   //but the algorithm doesn’t change even if the technique is different.
16:   wmarkedText ← grayscale(text, watermark)
17:
18:   //Perturb the watermarked text. 10 percent of perturbation.
19:   perturbedText ← deletion(wmarkedText, 10)
20:
21:   //Recover the watermark.
22:   recovered ← resolveGrayscale(perturbedText)
23:
24:   if watermark == recovered then
25:
26:     positive ← positive + 1
27:   else
28:
29:     negative ← negative + 1
30:
31: return positive, negative
```

---



# Chapter 6

## Experimental Results

This chapter catalogues the results from our testing of structural watermarking techniques. In all of these tests Grayscale Watermarking was used with a threshold of #272727 in order to hide the maximum amount of bits into a single character. The results are divided into 3 groups based on the length of the watermark hidden inside each of the 100 texts used for the tests; one table for each type of attacks then displays the number of those where a successful recovery of the watermark happened. As a reminder, 10% of the text affected by an attack is already a “high” quantity attack and, more importantly, the attacks performed on the texts all choose random sets of characters in a process that a human attacker would never do. We specifically chose this style of attack in order to better target the finegrained properties of our techniques, making the extraction process harder.

Homoglyph Watermarking has already been tested in [1] so it will not appear in any of the tables below. We will, however, report the results for ease of access to the information. The tests on Homoglyph Watermarking were performed as an evaluation on 1000 news articles of recent, Latin-based structural text watermarking and steganography methods. The results of the attacks are shown for 10% attack size.

<b>Partial CopyPaste</b>	<b>Insertion</b>	<b>Deletion</b>	<b>Replacement</b>	<b>Retyping</b>
99.92%	96.4%	98.2%	98.3%	0%

Table 6.1: Homoglyph Watermarking Experimental Results table. The Recoloring attack is not included because it doesn’t affect this technique at all.

## 6.1 64 Bits of watermark

This first section shows how resilient to attacks our watermarking techniques are. The most concerning technique is the Whitespace one which shows poor resistance to a partial Copy and Paste attack, even at a size of 10% of the text, with 30 watermarks retrieved successfully when used alone and 33 when combined with Homoglyphs watermarking. This can be attributed to how densely packed the watermark bits are in each whitespace and how few of them get selected in an attack like Copy and Paste. The rest of the attacks generally start taking their toll on the watermarked texts at about 50% of their length in affected characters which, as the reader will concede, results in a dramatic change to the original content of the text regardless of the effort in protecting it. For example, Grayscale watermarking keeps 36 out of 100 watermarks when 60% of the watermarked text is deleted but such a massive removal of characters is sure to modify the text in a more than meaningful way.

Deletion						
	10%	20%	30%	40%	50%	60%
<b>Grayscale</b>	100	100	100	90	62	36
<b>Whitespace</b>	100	99	96	94	69	41
<b>Grayscale + Whitespace</b>	100	100	100	100	89	45
<b>Homoglyphs + Grayscale</b>	100	100	100	95	65	21
<b>Homoglyphs + Whitespace</b>	100	98	90	58	39	22
<b>Complete</b>	100	100	100	100	91	53

Table 6.2: Number of positive 64 bits watermark extractions for the Deletion attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

Insertion						
	10%	20%	30%	40%	50%	60%
<b>Grayscale</b>	100	100	100	100	100	100
<b>Whitespace</b>	100	98	95	91	94	93
<b>Grayscale + Whitespace</b>	10	100	100	100	100	100
<b>Homoglyphs + Grayscale</b>	100	100	100	100	100	100
<b>Homoglyphs + Whitespace</b>	98	98	99	96	92	94
<b>Complete</b>	100	100	100	100	100	100

Table 6.3: Number of positive 64 bits watermark extractions for the Insertion attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

<b>Replacing</b>						
	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>
<b>Grayscale</b>	100	100	100	97	93	68
<b>Whitespace</b>	99	94	86	68	57	43
<b>Grayscale + Whitespace</b>	100	100	100	100	98	94
<b>Homoglyphs + Grayscale</b>	100	100	100	100	97	83
<b>Homoglyphs + Whitespace</b>	98	91	81	53	51	19
<b>Complete</b>	100	100	100	100	100	95

Table 6.4: Number of positive 64 bits watermark extractions for the Replacing attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

<b>Recoloring</b>						
	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>
<b>Grayscale</b>	100	100	100	99	89	71
<b>Whitespace</b>	100	100	92	83	64	46
<b>Grayscale + Whitespace</b>	100	100	100	100	100	97
<b>Homoglyphs + Grayscale</b>	100	100	100	100	97	81
<b>Homoglyphs + Whitespace</b>	100	94	83	56	36	25
<b>Complete</b>	100	100	100	100	100	99

Table 6.5: Number of positive 64 bits watermark extractions for the Recoloring attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

<b>Partial Copy and Paste</b>							
	<b>20%</b>	<b>10%</b>	<b>9%</b>	<b>8%</b>	<b>7%</b>	<b>6%</b>	<b>5%</b>
<b>Grayscale</b>	100	100	100	100	99	100	93
<b>Whitespace</b>	90	30	29	21	7	6	3
<b>Grayscale + Whitespace</b>	100	100	100	100	100	100	99
<b>Homoglyphs + Grayscale</b>	100	100	100	100	100	99	97
<b>Homoglyphs + Whitespace</b>	94	33	28	18	14	8	2
<b>Complete</b>	100	100	100	100	100	100	99

Table 6.6: Number of positive 64 bits watermark extractions for the Partial Copy and Paste attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

## 6.2 128 Bits of watermark

With a watermark size of 128 we can see that some of the higher percentages of affected characters greatly hamper the robustness of our techniques. Once again, Whitespace Watermarking shows remarkable weakness to Copy and Paste attacks but performs admirably in all other aspects. The results from the previous subsection foreshadowed this results but the techniques still hold up to every type of attack extremely well considering that at least 10% of each text is being modified each time.

Deletion						
	10%	20%	30%	40%	50%	60%
<b>Grayscale</b>	100	94	52	15	3	1
<b>Whitespace</b>	98	81	41	20	6	1
<b>Grayscale + Whitespace</b>	100	100	74	30	5	0
<b>Homoglyphs + Grayscale</b>	100	92	40	10	3	0
<b>Homoglyphs + Whitespace</b>	90	49	20	7	1	0
<b>Complete</b>	100	99	77	34	6	1

Table 6.7: Number of positive 128 bits watermark extractions for the Deletion attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

Insertion						
	10%	20%	30%	40%	50%	60%
<b>Grayscale</b>	100	98	79	46	32	15
<b>Whitespace</b>	95	82	58	40	32	13
<b>Grayscale + Whitespace</b>	100	100	99	91	77	48
<b>Homoglyphs + Grayscale</b>	100	100	83	57	36	19
<b>Homoglyphs + Whitespace</b>	98	88	79	56	44	24
<b>Complete</b>	100	100	100	93	78	61

Table 6.8: Number of positive 128 bits watermark extractions for the Insertion attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

<b>Replacing</b>						
	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>
<b>Grayscale</b>	100	93	54	24	9	1
<b>Whitespace</b>	84	55	18	3	1	2
<b>Grayscale + Whitespace</b>	100	99	85	59	20	9
<b>Homoglyphs + Grayscale</b>	100	92	55	25	7	2
<b>Homoglyphs + Whitespace</b>	81	36	10	4	1	0
<b>Complete</b>	100	98	91	54	29	13

Table 6.9: Number of positive 128 bits watermark extractions for the Replacing attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

<b>Recoloring</b>						
	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>
<b>Grayscale</b>	100	90	55	32	6	4
<b>Whitespace</b>	88	50	28	11	4	1
<b>Grayscale + Whitespace</b>	100	100	86	49	35	14
<b>Homoglyphs + Grayscale</b>	100	98	68	38	12	4
<b>Homoglyphs + Whitespace</b>	79	29	5	5	3	0
<b>Complete</b>	100	99	92	59	29	20

Table 6.10: Number of positive 128 bits watermark extractions for the Recoloring attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

<b>Partial Copy and Paste</b>							
	<b>20%</b>	<b>10%</b>	<b>9%</b>	<b>8%</b>	<b>7%</b>	<b>6%</b>	<b>5%</b>
<b>Grayscale</b>	100	93	85	61	50	16	5
<b>Whitespace</b>	35	1	0	0	0	0	0
<b>Grayscale + Whitespace</b>	100	100	100	98	79	47	25
<b>Homoglyphs + Grayscale</b>	100	100	92	74	49	23	3
<b>Homoglyphs + Whitespace</b>	33	0	0	0	0	0	0
<b>Complete</b>	100	100	99	97	87	55	27

Table 6.11: Number of positive 128 bits watermark extractions for the Partial Copy and Paste attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

### 6.3 256 Bits of watermark

With a watermark size as large as 256 bits, it is to be expected that the results would drop but, importantly, Grayscale Watermarking mixed with Whitespace Watermarking and the Complete technique maintain a respectable > 90% positive extractions at 10% disturbance, making these particular techniques highly reliable.

<b>Deletion</b>						
	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>
<b>Grayscale</b>	79	18	3	0	0	0
<b>Whitespace</b>	59	24	1	1	0	0
<b>Grayscale + Whitespace</b>	92	30	3	0	0	0
<b>Homoglyphs + Grayscale</b>	74	9	1	0	0	0
<b>Homoglyphs + Whitespace</b>	37	3	0	0	0	0
<b>Complete</b>	97	35	4	1	0	0

Table 6.12: Number of positive 256 bits watermark extractions for the Deletion attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

<b>Insertion</b>						
	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>
<b>Grayscale</b>	80	16	2	0	1	0
<b>Whitespace</b>	53	24	3	1	0	0
<b>Grayscale + Whitespace</b>	97	52	25	3	4	0
<b>Homoglyphs + Grayscale</b>	88	22	9	0	1	0
<b>Homoglyphs + Whitespace</b>	67	26	8	10	1	1
<b>Complete</b>	99	63	20	6	5	2

Table 6.13: Number of positive 256 bits watermark extractions for the Insertion attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

<b>Replacing</b>						
	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>
<b>Grayscale</b>	79	9	1	0	0	0
<b>Whitespace</b>	22	2	0	0	0	0
<b>Grayscale + Whitespace</b>	96	47	7	3	0	0
<b>Homoglyphs + Grayscale</b>	85	16	2	0	0	0
<b>Homoglyphs + Whitespace</b>	19	0	0	0	0	0
<b>Complete</b>	93	34	15	1	0	0

Table 6.14: Number of positive 256 bits watermark extractions for the Replacing attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

<b>Recoloring</b>						
	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>
<b>Grayscale</b>	69	10	0	0	0	0
<b>Whitespace</b>	25	5	3	0	0	0
<b>Grayscale + Whitespace</b>	98	39	6	2	0	0
<b>Homoglyphs + Grayscale</b>	78	11	2	1	0	0
<b>Homoglyphs + Whitespace</b>	11	1	0	0	0	0
<b>Complete</b>	96	45	10	1	0	0

Table 6.15: Number of positive 256 bits watermark extractions for the Recoloring attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

<b>Partial Copy and Paste</b>							
	<b>20%</b>	<b>10%</b>	<b>9%</b>	<b>8%</b>	<b>7%</b>	<b>6%</b>	<b>5%</b>
<b>Grayscale</b>	96	2	0	0	0	0	0
<b>Whitespace</b>	0	0	0	0	0	0	0
<b>Grayscale + Whitespace</b>	100	25	11	1	0	0	0
<b>Homoglyphs + Grayscale</b>	99	9	0	0	0	0	0
<b>Homoglyphs + Whitespace</b>	0	0	0	0	0	0	0
<b>Complete</b>	99	27	15	7	0	0	0

Table 6.16: Number of positive 256 bits watermark extractions for the Partial Copy and Paste attack on 100 examples of 200 characters texts with each of the main techniques and their combinations.

# Chapter 7

## Conclusion

Text Watermarking is a research field that is always expanding because content paternity certification is an interesting and worthwhile process for both private people and industries alike. The rapid and uncontrolled spread of content over the internet, mainly textual, makes protecting one's intellectual property challenging. In this thesis we posit that one of the most effective ways to protect a piece of content is through the use of Digital Watermarking, a process that embeds some information, the watermark, into a piece of content using some specific method.

After classifying watermarking methods by characteristics and set of techniques, we focused on Structural techniques which embed a watermark by exploiting the underlying characteristics of text and characters. The focus of this thesis has been on three structural watermarking techniques in Grayscale, Whitespace and Homoglyph Watermarking and building a suit of test in order to quantify their robustness against a series of attacks that represents common operations done on a text in order to steal its content or purposefully deleting the present watermark.

Because the presented techniques apply the watermark in a finegrained fashion, meaning that the same payload of bits is embedded back to back, we wanted to create a testing environment capable of highlighting the pros and cons of this approach. We explained the suit of tests, the reasoning behind each attack and we provided pseudocode for each of them so that they can be reproduced if further testing is needed or if the same attacks were to be used on other watermarking techniques.

Finally, the results show that our techniques resist all attacks admirably with a 64 bits watermark up to 60% of the text affected by the attack, a result that proves that the high embedding capacity of our techniques translates into a capillar protection for every transformed text. Other watermarking payload sizes were considered in 128 bits and 256 bits and our results were convincing up to the industry standard 10% of the text being affected. These results appear even better when we remember that the attacks on the techniques were designed to be random to challenge the techniques' finegrained properties. The only vulnerability we found was with the Partial Copy and Paste attack



and the Whitespace watermarking technique which shows poor performance compared to the other options at all payload size probably because each affected whitespace hold much more of the payload than other techniques.

# Bibliography

- [1] Stefano Giovanni Rizzo, Flavio Bertini, and Danilo Montesi. Fine-grain watermarking for intellectual property protection. *EURASIP Journal on Information Security*, 2019:1–20, 2019.
- [2] Manmeet Kaur and Kamna Mahajan. An existential review on text watermarking techniques. *International Journal of Computer Applications*, 120(18), 2015.
- [3] Nurul Shamimi Kamaruddin, Amirrudin Kamsin, Lip Yee Por, and Hameedur Rahman. A review of text watermarking: theory, methods, and applications. *IEEE Access*, 6:8011–8028, 2018.
- [4] Yaxun Zhou and Wei Jin. A novel image zero-watermarking scheme based on dwt-svd. In *2011 International Conference on Multimedia Technology*, pages 2873–2876. IEEE, 2011.
- [5] Jack T Brassil, Steven Low, Nicholas F. Maxemchuk, and Lawrence O’Gorman. Electronic marking and identification techniques to discourage document copying. *IEEE Journal on Selected Areas in Communications*, 13(8):1495–1504, 1995.
- [6] Steven H Low, Nicholas F Maxemchuk, Jack T Brassil, and Lawrence O’Gorman. Document marking and identification using both line and word shifting. In *Proceedings of INFOCOM’95*, volume 2, pages 853–860. IEEE, 1995.
- [7] Anoop K Bhattacharjya and Hakan Ancin. Data embedding in text for a copier system. In *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, volume 2, pages 245–249. IEEE, 1999.
- [8] Young-Won Kim and Il-Seok Oh. Watermarking text document images using edge direction histograms. *Pattern Recognition Letters*, 25(11):1243–1251, 2004.
- [9] Steven H Low, Nicholas F Maxemchuk, and Aleta M Lapone. Document identification for copyright protection using centroid detection. *IEEE Transactions on Communications*, 46(3):372–383, 1998.

- [10] Ding Huang and Hong Yan. Interword distance changes represented by sine waves for watermarking text images. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(12):1237–1245, 2001.
- [11] Young-Won Kim, Kyung-Ae Moon, and Il-Seok Oh. A text watermarking algorithm based on word classification and inter-word space statistics. In *ICDAR*, pages 775–779. Citeseer, 2003.
- [12] Tomio Amano and Daigo Misaki. A feature calibration method for watermarking of document images. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR'99 (Cat. No. PR00318)*, pages 91–94. IEEE, 1999.
- [13] Xiaofeng Wang. Digital watermarking research based on text. In *2013 IEEE Third International Conference on Information Science and Technology (ICIST)*, pages 433–436. IEEE, 2013.
- [14] Mikhail J Atallah, Victor Raskin, Michael Crogan, Christian Hempelmann, Florian Kerschbaum, Dina Mohamed, and Sanket Naik. Natural language watermarking: Design, analysis, and a proof-of-concept implementation. In *Information Hiding: 4th International Workshop, IH 2001 Pittsburgh, PA, USA, April 25–27, 2001 Proceedings 4*, pages 185–200. Springer, 2001.
- [15] Peter Craig Collins. Pseudocleft and cleft constructions: a thematic and informational interpretation. 1991.
- [16] Hasan Mesut Meral, Bülent Sankur, A Sumru Özsoy, Tunga Güngör, and Emre Sevinç. Natural language watermarking via morphosyntactic alterations. *Computer Speech & Language*, 23(1):107–125, 2009.
- [17] Rodney Huddleston. *Introduction to the Grammar of English*. Cambridge University Press, 1984.
- [18] Umut Topkara, Mercan Topkara, and Mikhail J Atallah. The hiding virtues of ambiguity: quantifiably resilient watermarking of natural language text through synonym substitutions. In *Proceedings of the 8th workshop on Multimedia and security*, pages 164–174, 2006.
- [19] Mercan Topkara, Cuneyt M Taskiran, and Edward J Delp III. Natural language watermarking. In *Security, Steganography, and Watermarking of Multimedia Contents VII*, volume 5681, pages 441–452. SPIE, 2005.
- [20] Lip Yee Por, KokSheik Wong, and Kok Onn Chee. Unispach: A text-based data hiding method using unicode space characters. *Journal of Systems and Software*, 85(5):1075–1082, 2012.

- [21] Reem A Alotaibi and Lamiaa A Elrefaei. Improved capacity arabic text watermarking methods based on open word space. *Journal of King Saud University-Computer and Information Sciences*, 30(2):236–248, 2018.
- [22] Behrooz Khosravi, Behnam Khosravi, Bahman Khosravi, and Khashayar Nazarkardeh. A new method for pdf steganography in justified texts. *Journal of information security and applications*, 45:61–70, 2019.
- [23] Salwa Shakir Baawi, Mohd Rosmadi Mokhtar, and Rossilawati Sulaiman. Enhancement of text steganography technique using lempel-ziv-welch algorithm and two-letter word technique. In *Recent Trends in Data Science and Soft Computing: Proceedings of the 3rd International Conference of Reliable Information and Communication Technology (IRICT 2018)*, pages 525–537. Springer, 2019.
- [24] Nighat Mir. Copyright for web content using invisible text watermarking. *Computers in Human Behavior*, 30:648–653, 2014.
- [25] Milad Taleby Ahvanooy, Hassan Dana Mazraeh, and Seyed Hashem Tabasi. An innovative technique for web text watermarking (aitw). *Information Security Journal: A Global Perspective*, 25(4-6):191–196, 2016.
- [26] Stefano Giovanni Rizzo, Flavio Bertini, Danilo Montesi, and Carlo Stomeo. Text watermarking in social media. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, pages 208–211, 2017.
- [27] Jack T Brassil, Steven Low, and Nicholas F Maxemchuk. Copyright protection for the electronic distribution of text documents. *Proceedings of the IEEE*, 87(7):1181–1196, 1999.
- [28] Jack Brassil, Steven Low, Nicholas Maxemchuk, and Larry O’Gorman. Hiding information in document images. In *Proc. Conf. Information Sciences and Systems (CISS-95)*, pages 482–489. Citeseer, 1995.
- [29] Nurul Shamimi Kamaruddin, Amirrudin Kamsin, Lip Yee Por, and Hameedur Rahman. A review of text watermarking: theory, methods, and applications. *IEEE Access*, 6:8011–8028, 2018.
- [30] Simone Branchetti, Flavio Bertini, and Danilo Montesi. Grayscale text watermarking. In *Proceedings of the 26th International Database Engineered Applications Symposium, IDEAS ’22*, page 166–170, New York, NY, USA, 2022. Association for Computing Machinery.

- [31] Jean-Philippe Aumasson and Daniel J Bernstein. Siphash: a fast short-input prf. In *Progress in Cryptology-INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings 13*, pages 489–508. Springer, 2012.
- [32] FIPS Pub. Secure hash standard (shs). *Fips pub*, 180(4), 2012.
- [33] The new york times annotated corpus. <https://catalog.ldc.upenn.edu/LDC2008T19>.
- [34] Miloš Jakubiček, Adam Kilgarriff, Vojtěch Kovář, Pavel Rychlý, and Vít Suchomel. The tenten corpus family. In *7th international corpus linguistics conference CL*, pages 125–127, 2013.
- [35] Yingli Zhang, Huaiqing Qin, and Tao Kong. A novel robust text watermarking for word document. In *2010 3rd International Congress on Image and Signal Processing*, volume 1, pages 38–42. IEEE, 2010.
- [36] Pradeep Kaur and Pankaj Bhambri. To design an algorithm for text watermarking. *The Standard International Journals (The SIJ)*, 3(5):62–67, 2015.
- [37] Milad Taleby Ahvanooy, Qianmu Li, Hiuk Jae Shim, and Yanyan Huang. A comparative analysis of information hiding techniques for copyright protection of text documents. *Security and Communication Networks*, 2018, 2018.
- [38] Morteza Bashardoost, Mohd Shafry Mohd Rahim, Tanzila Saba, and Amjad Rehman. Replacement attack: A new zero text watermarking attack. *3D Research*, 8:1–9, 2017.