Enhancement of Plagiarism Detection Techniques via Watermarking

Dylan Ryman
Department of Engineering Education
University of Cincinnati
Cincinnati, OH, USA
rymandn@mail.uc.edu

P.K. Imbrie
Department of Engineering Education
University of Cincinnati
Cincinnati, OH, USA
imbriepk@ucmail.uc.edu

Jeff Kastner
Department of Engineering Education
University of Cincinnati
Cincinnati, OH, USA
kastnejy@ucmail.uc.edu

***Abstract*—Plagiarism in student submissions is a continual threat to the integrity of engineering education. Online learning environments enhance the accessibility of many forms of plagiarism, and a corresponding increase in the prevalence of academic misconduct has been observed. When unmitigated, plagiarism has severe detrimental effects on student learning outcomes, and therefore it is the responsibility of instructors to detect plagiarism by all practical means. Plagiarism identification techniques typically seek to discover abnormal similarities in student submissions, and although imperfect, research has shown that this approach is highly effective. This work in progress paper seeks to augment traditional similarity-focused plagiarism detection strategies through the use of watermarks, or student-specific unique identifiers embedded in files. This paper shows how existing watermarking techniques can be applied in the context of plagiarism detection by integrating watermarks into assignment template files. The applicability of watermarking to multiple submission types, including source code files and Microsoft Office documents, is demonstrated. Additionally, this paper presents methods for automated generation of watermarked templates, distribution of marked files, and evaluation of submissions for plagiarism. Finally, preliminary results are presented, and future work necessary to complete this in-progress project is discussed.**

***Index Terms*—plagiarism detection, digital watermarking, assessment automation**

I. Introduction

Academic misconduct, including plagiarism, is a perpetual concern for engineering educators. Academic dishonesty in higher education is associated with future unethical behavior [1], so it is the responsibility of instructors to identify and abate this conduct. Automated plagiarism detection tools are an indispensable plagiarism countermeasure, and have demonstrated effectiveness in many contexts. Most commonly used detection tools seek to uncover suspicious similarity in pairs of student submissions [2], however, this paper presents a plagiarism detection framework that applies novel techniques to a lesser-known approach.

Many forms of assessment involve a starter file or template file intended to be modified and submitted by students. Examples include starter code for a programming assignment, or an answer sheet in the form of a Microsoft Office file. The template provided to each student is typically specific to the assignment and uniform across students. This work seeks to imperceptibly individualize each template by the addition of a watermark, a small piece of embedded data, that can be traced back to a specific student.

This work presents methods for watermarking two common types of templates: source code files and Microsoft Office documents. Capabilities of modern text encoding schemes are leveraged to generate watermarks that are completely invisible in most text and document editors, a technique that has seen little to no application in the context of plagiarism detection. The complex internal structure of Microsoft Office documents is utilized to hide intangible yet robust watermarks. Additionally, methods for the automatic generation and distribution of marked templates, as well as the automatic analysis of submissions, are discussed.

These methods are not a replacement for similarity-based detection tools or other plagiarism countermeasures. Effectiveness is contingent on students plagiarizing by receiving and submitting modified templates from their peers, but not all cheating occurs in this manner. It is therefore inevitable that watermarking will miss many cases of plagiarism, including all manual copying and most instances of copy and paste, which is why it is intended to be used in conjunction with other tools, or when no alternatives are available.

II. Background

Digital watermarking was initially developed to aid in the detection of audiovisual copyright infringement, but it has found many other applications including modification tracking and plagiarism detection for multimedia, documents, and plain text [3].

A. Document Watermarking

This paper applies information hiding and watermarking techniques for both plain text source code files and Microsoft Office documents, making use of several previous works in both of these areas.

Many plain text watermarking methods have been studied. Most approaches are content-based, however, this is not suitable for templates that may contain little or no text, so a different method is nessesary [4]. Plain text may be stored using the UTF-8 code page, which enables a brilliant information hiding technique by means of particular invisible characters. The use of special UTF-8 characters to embed

hidden data into plain text files is uncommon, but it is not new. Several previous references to the technique, or some variation of it, were identified [5], [6], [7]. The substantial utility of this steganographic technique for embedding imperceptible watermarks into plain text appears to be somewhat overlooked, as recent reviews of text watermarking methods omit it [8], [9].

Microsoft Office documents are based on a file format known as Office Open XML (OOXML). Word, Excel, and PowerPoint documents all leverage the OOXML file format, and these document types will be known simply as OOXML for the remainder of this paper. The complexity of the OOXML format presents a rich ground for information hiding, and numerous OOXM steganographic methods have been discussed in previous works [10], [11], [12]. Such techniques have also been previously applied in the context of document watermarking [13].

B. Watermarking and Plagiarism Detection

Watermarking has seldom been applied in the context of plagiarism detection for either source code or OOXML documents, however, a few relevant works were identified.

Systematic reviews of source code plagiarism detection tools revealed only two instances of watermarks being leveraged for plagiarism detection in a population of more than one hundred distinct tools [14]. One such tool, RoboProf, did not distribute code templates with hidden watermarks, but rather utilized locally installed software on each student's computer to inject watermarks into files immediately after submission [15]. RoboProf's approach has several disadvantages compared to this paper's method, including the security and privacy concerns associated with locally installed software and the use of whitespace-based watermarks that are easier to both detect and manipulate than this paper's invisible watermarks.

In the context of OOXML documents, only one instance of plagiarism detection via hidden watermarks was identified in literature. Akinori Ito describes an effective and robust watermark encoding for templates that hides data by switching between similar fonts [16]. This paper's invisible watermarking technique presents a preferable alternative, as it is harder to detect and is applicable in templates with minimal starter text.

No previous work presenting a unified framework for template watermarking, distribution, and analysis was identified.

III. Methods

A. Source Code Watermarking

Discreetly watermarking source code files is particularly difficult due to the limited ways in which information can be hidden in plain text. This paper's method leverages the fact that most text editors correctly handle text data encoded with UTF-8, a modern standard for text encoding that enables many localized character set to be accessed as part of a single code page. This method exploits several UTF-8 "control characters," characters intended for manipulating text layout or rendering of typographic ligatures, for an unintended purpose. By specification, some of these control characters have a width of zero, which prevents them from being displayed in compliant text editors. As demonstrated by Sonnleitner [5], several such zero-width control characters can be used to encode data in a plain text file while being very difficult to detect in practice. Additional considerations are required for this method that prevent direct replication of Sonnleitner's approach. Since watermarked source code files must maintain the ability to be run or compiled, and control characters are rejected as invalid syntax by most interpreters and compilers, watermarks cannot be arbitrarily placed within the template. Rather, they can only be injected into a comment where they have no effect on the compilation or interpretation of the source code.

The watermarking process developed for this paper's framework has several steps. First, a binary representation of the target student's ID is obtained. Since student IDs use only alphanumeric characters, ASCII codes are used for this purpose. Next, a bijective mapping of binary to the set of selected control characters is applied to the encoded ID, enabling easy reversibility. This paper's implementation maps 0 to the "zero-width space" character, and 1 to the "zero-width joiner" character. The data encoding process is visualized in Figure 1. Finally, based on the specific language of the source code file, an arbitrary comment is identified, and the watermark is appended to the commented line. For this reason, each template file must contain at least one comment. At this point, the marked files are ready for distribution.

This paper's UTF-8 watermarks are not entirely undetectable. While the UTF-8 watermarks take up no space and therefore appear invisible, each of the invisible characters is handled as a real and editable character by text editors. The most notable artifact of this is seen when moving the cursor a single position via the arrow keys. When the watermark is reached, the cursor will "get stuck" for several arrow key presses as each invisible character is traversed. Additionally, editors that display the cursor's column position will show a seemingly incorrect value when the cursor is placed after a watermark, as the invisible characters are included in the count. Both of these effects can be dramatically mitigated by injecting watermarks only at the end of comment lines.

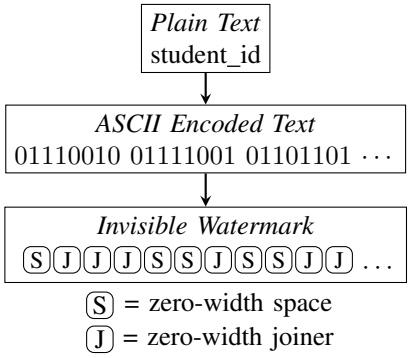

Fig. 1. UTF-8 watermark encoding process.

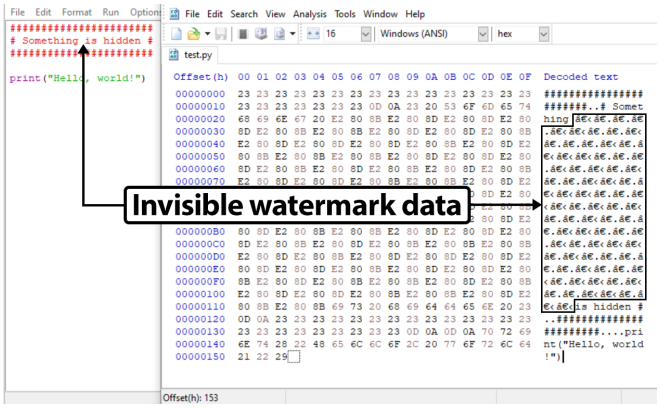

Fig. 2. The same watermarked Python file shown in IDLE and a hex editor.

Some text editors fail to correctly handle zero-width UTF-8 characters, and some cannot display UTF-8 at all. This is, however, not typically an issue as most modern source code editors correctly handle UTF-8 characters and do not render the watermark. Common editors that were tested and work as expected include Visual Studio, VS Code, Notepad++, IntelliJ IDEA, and Python IDLE. This invisibility is demonstrated for Python IDLE in Figure 2. The Notepad editor included with Windows by default exhibits the bizarre behavior of rendering watermarked lines in a different font from the rest of the file, which does not directly reveal the watermark but is certainly suspicious. The MATLAB editor cannot display UTF-8 characters and reveals the watermark.

B. OOXML Watermarking

This paper's method combines two independent watermarking methods in order to maximize robustness.

First, a little known feature of the OOXML format that enables arbitrary data to be embedded in a document is utilized. A custom data component containing the student ID and the required relationship entries, defined in Section 15.2.6 of the OOXML specification [17], are created within the compressed OOXML document. Fortunately, an existing procedure could be used for this step without modification. The selected procedure is somewhat complex, but a detailed description can be found in Chapter 3.3.4 of [18]. The custom data component is not used by the document editor, but it is persisted across modifications and saves. Retrieval of the watermark is simple. The custom data component is read by an external tool, from which the watermark data is trivially extracted. Since this method relies on the internal OOXML structure, it cannot survive format conversions such as exporting to PDF.

The second method reuses the invisible character technology discussed in Section III-A. Since source code correctness is not a concern for document files, the encoded student ID can be inserted at any location in the document. The precise location is unimportant, but incorporating some amount of unpredictability into the selected location improves resistance towards watermark removal efforts, should students come to learn of their existence. It is not necessary to save the selected locations, as searching even very large documents for these watermarks is computationally trivial.

C. Generation and Processing

An individualized template is required for each student, so the preparation of templates and subsequent analysis of submissions presents an impractical effort without automation. Fortunately, both of the watermarking techniques discussed in this paper are easily automated with a computer program. Therefore, when a batch of templates is required, a Python script is used to read the unmarked template and a course roster containing student IDs. The appropriate watermarking technique is selected automatically based on the template file type, and a marked template is automatically generated for each student in the roster. The distribution of these templates is discussed in Section III-D.

Detecting plagiarism is accomplished by performing a batch export of submissions for a given assignment from the learning management system. The LMS automatically prefixes each exported file's name with the student ID of the submitter. Additionally, each of the watermarking techniques discussed in this paper enables recovery of the hidden data, namely, the student ID associated with the template. A Python script reads the exported files, extracts the hidden student ID from each submission, and compares it to the student ID present in the file name. If the hidden ID is unable to be extracted, or it does not match the ID in the file name, the user is alerted for further investigation.

D. Template Distribution

The distribution of watermarked files to students presents a unique challenge, as each student must receive an individualized template file at the appropriate time. A custom web-application was developed for this purpose. Questions that require template distribution embed a hyperlink to this application with a question-specific secret key encoded in the URL. The secret key ensures that students cannot access the template file prematurely. When a student navigates to the application, the student is prompted to enter his or her student ID, after which the appropriate template file is automatically selected and downloaded to the student's computer. While generally effective, this method has multiple inherent risks.

1) Additional Complexity: Students must be able to use the web application correctly to receive a template file. Unfortunately, it is alarmingly common for students to forget their student IDs, neglect to learn their IDs entirely, or attempt to use a different ID than the requested type, when accessing the application. Repeat usage of this distribution system over several assignments mitigates this issue, however, this technology is most effective when students are unaware of the watermarks, so it is best used on only the most important assignments, such as exams. The potential for students to get stuck when attempting to download a template adds additional mental overhead for both students and instructors in an already stressful exam setting.

2) False Positives Potential: Since students must manually type their student IDs, it is possible for a student to obtain an incorrectly watermarked template by intentionally entering the student ID of a different student. While a nonsensical course of action to take, a claim could be made that this, rather than academic misconduct, is the cause of a mismatched submission. The ideal distribution platform would require no student-provided data, eliminating this concern. Mitigations for both of these risks are left to future work, and a potential solution is discussed in Section V-A.

IV. Preliminary Results

As this work is in progress, limited data on the effectiveness or robustness of the watermarking framework presented in this paper are available. The technology was piloted at the authors' institution for a midterm exam in a first-year engineering course. The exam was held in person with a digital format facilitated by a Canvas quiz. The question selected for the watermarking pilot was a Microsoft Excel problem where students downloaded a workbook file, introduced their own formulas, then uploaded the modified file to the Canvas quiz question. This selection was made because no other plagiarism detection methods were slated for use on that problem. Due to the nature of spreadsheet formula problems, most students with the correct answer share an identical solution with many other students, making similarity-based plagiarism detection impossible.

Of the 1,051 students who submitted a solution for the selected problem (n=1,051), 15 students were unable to use the file distribution tool and had to be provided with an unmarked template, six students submitted a corrupt or otherwise unreadable file, and eight students were identified as having submitted a modified template originating from a different student. This number is small, representing a 0.77% detection rate among the 1,030 successfully marked files, but this result is not unexpected. Consistent with the limitations of the described methods, only students who submitted a file received from another student were identified. Students who simply copied formulas from a different file were not detected, and therefore the actual rate of cheating was likely much higher than 0.77%. As mentioned, this method is designed to augment, not replace, other plagiarism countermeasures. In this pilot program, none of the identified students were charged with academic misconduct on the basis of these detections.

V. Future Work

As this work is in progress, several components of the project are incomplete and will be the subject of future work.

A. Canvas LMS Integration

For the reasons specified in Section III-D, an improved file distribution mechanism is a desired enhancement for this work. One approach that remains an active goal for this project is the seamless distribution of watermarked templates through the Canvas Learning Management System. Unfortunately, the distribution of a unique file to each student in Canvas is a nontrivial task. While the content of Canvas quizzes can be managed autonomously via the Canvas application programming interface, the Canvas quiz engines alone are not powerful enough to display a student-specific stimulus in the context of a quiz question. One possible solution is the use of a custom Canvas LTI tool. An LTI tool is a web-based application that implements the Learning Tools Interoperability standard. LTI tools can access information about the current student, and are often used for delivering dynamic student-specific content [19]. In fact, it appears that a Canvas-specific LTI feature known as "Deep Linking" was designed with precisely this use case in mind, noting in its documentation that deep linking may be used for "Embedding custom content into a rich text editor from a tool provider" [20]. It should be possible to embed such an LTI tool into the stimulus of a Canvas quiz question, enabling seamless watermarked template downloading. However, work on the custom LTI tool is still in progress, and a functional prototype does not yet exist, so it is not yet known with certainty if this method will work. One disadvantage of this approach is that LTI tools must be installed into Canvas by an institution-level administrator, as standard account access is not sufficient [21]. This may present procedural roadblocks for instructors or departments wishing to use this technology at some institutions.

B. Additional Watermarking Techniques

The number of different file types applicable as templates is vast, but a watermarking technique for each must be individually designed and integrated into this paper's framework. This paper presented methods for two of the most common use cases: Microsoft Office files and source code. Additionally, the authors have prototyped methods for printed and scanned paper documents as well as NI LabVIEW files, although length prevented their inclusion in this short paper. Future work will analyze the frequency of various template file types in engineering assessment, then implement watermarking support for each of the most relevant types when possible.

C. Analysis of Effectiveness

Section IV presented preliminary results for one watermarking technique in the context of a pilot program, however, the effectiveness of these techniques in general is not known. The design and implementation of a research methodology aimed at quantifying the effectiveness of these methods in various contexts is a task left for future work.

VI. Conclusion

This paper demonstrated novel applications of recently developed watermarking techniques in the context of plagiarism detection, and showed how these techniques may be leveraged within an automated plagiarism detection framework. While initial results are promising, future work is necessary for enhancing current methods and establishing effectiveness through further experimentation and analysis.

References

[1] T. S. Harding, M. J. Mayhew, C. J. Finelli, and D. D. Carpenter, "The theory of planned behavior as a model of academic dishonesty in engineering and humanities undergraduates," *Ethics & Behavior*, vol. 17, no. 3, pp. 255–279, 2007. [Online]. Available: https://doi.org/10.1080/10508420701519239

[2] T. Foltýnek, N. Meuschke, and B. Gipp, "Academic plagiarism detection: A systematic literature review," *ACM Comput. Surv.*, vol. 52, no. 6, oct 2019. [Online]. Available: https://doi.org/10.1145/3345317

[3] G. Voyatzis, N. Nikolaidis, and I. Pitas, "Digital watermarking: An overview," in *9th European Signal Processing Conference (EUSIPCO 1998)*, 1998, pp. 1–4.

[4] Z. Jalil, A. Mirza, and S. Maria, "Content based zero-watermarking algorithm for authentication of text documents," *International Journal of Computer Science and Information Security*, vol. 7, 03 2010.

[5] E. Sonnleitner, "A user-tracable watermarking scheme for dynamic web-page content," in *The First International Conference on Future Generation Communication Technologies*, 2012, pp. 121–125.

[6] M. T. Ahvanooey, H. D. Mazraeh, and S. H. Tabasi, "An innovative technique for web text watermarking (aitw)," *Information Security Journal: A Global Perspective*, vol. 25, no. 4-6, pp. 191–196, 2016. [Online]. Available: https://doi.org/10.1080/19393555.2016.1202356

[7] H. M. Bashir, Q. Li, and J. Hou, "A high capacity text steganography utilizing unicode zero-width characters," in *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, 2020, pp. 668–675.

[8] M. H. Alkawaz, G. Sulong, T. Saba, A. S. Almazyad, and A. Rehman, "Concise analysis of current text automation and watermarking approaches," *Security and Communication Networks*, vol. 9, no. 18, pp. 6365–6378, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1738

[9] M. Kaur and K. Mahajan, "An existential review on text watermarking techniques," *International Journal of Computer Applications*, vol. 120, pp. 29–32, 06 2015.

[10] Z. Fu, X. Sun, L. Zhou, and J. Shu, "New forensic methods for ooxml format documents," in *Digital-Forensics and Watermarking*, Y. Q. Shi, H.-J. Kim, and F. Pérez-González, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 503–513.

[11] A. Castiglione, B. D'Alessio, A. D. Santis, and F. Palmieri, "Hiding information into ooxml documents: New steganographic perspectives," *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, vol. 2, pp. 59–83, 2011.

[12] W. Guo, L. Yang, Y. Lu, Y. Yang, L. Li, and Z. Liu, "Information hiding in ooxml format data based on the splitting of text elements," in *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2019, pp. 188–190.

[13] N. Liao, C. Sun, L. Xiang, and F. Li, "A multiple watermarking scheme for content authentication of ooxml format documents," in *Cloud Computing and Security*, X. Sun, Z. Pan, and E. Bertino, Eds. Cham: Springer International Publishing, 2018, pp. 147–159.

[14] M. Novak, M. Joy, and D. Kermek, "Source-code similarity detection and detection tools used in academia: A systematic review," *ACM Trans. Comput. Educ.*, vol. 19, no. 3, may 2019. [Online]. Available: https://doi.org/10.1145/3313290

[15] C. Daly and J. m. Horgan, "A technique for detecting plagiarism in computer code," *Computer Journal*, vol. 48, 11 2005.

[16] A. Ito, "Demonstration experiment of data hiding into ooxml document for suppression of plagiarism," in *Advances in Intelligent Information Hiding and Multimedia Signal Processing*, J.-S. Pan, P.-W. Tsai, and H.-C. Huang, Eds. Cham: Springer International Publishing, 2017, pp. 3–10.

[17] ISO 29500-1:2016, *Information technology — Document description and processing languages — Office Open XML File Formats — Part 1: Fundamentals and Markup Language Reference*. ISO, Geneva, Switzerland, nov 2016.

[18] M. A. Raffay, "Data hiding and detection in office open xml (ooxml) documents," Master's thesis, University of Ontario Institute of Technology, 2011.

[19] C. Severance, T. Hanss, and J. Hardin, "Ims learning tools interoperability: Enabling a mash-up approach to teaching and learning tools," *Technology, Instruction, Cognition and Learning*, no. 7, 2009.

[20] "Using Deep Linking to Select Resources," Instructure, Inc, December 2021. [Online]. Available: https://canvas.instructure.com/doc/api/file.content_item.html

[21] "Configuring LTI Advantage Tools," Instructure, Inc, October 2021. [Online]. Available: https://canvas.instructure.com/doc/api/file.lti_dev_key_config.html