International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

A CFFO Watermarking Algorithm for Protecting Ownership of Software Code.

Bharati Devidas Patil[1], Prof. Dr. Monika Tripathi[2]

[1]Research Scholar, [2]Research Supervisor
[1,2]Department of Computer Science and Engineering, P. K. University, Shivpuri 473665, Madhya Pradesh, India

Abstract
Software is the core element to execute various operations in the networks, which is widely suggested for use in e-commerce, daily life and industries. Software is a kind of digital object, which faces more issues in terms of piracy and security. In some cases, digital objects as well as documents are duplicated and easily distributed to various sources cheaply. Moreover, cyber-attacks are evolving at the distribution rate and it leads to ransomware attacks, data breaches, phishing attacks, malware and crypto-jacking. Implementing software in different domains leads to different cyber-attacks. Thus, to protect the software from unauthorized modifications and piracy, different techniques are designed and implemented. Among all, software-based watermarking techniques are employed to protect the source code from unauthorized modifications through extraction and embedding watermarking algorithms. In case, the source code is hacked or modified using watermarking techniques, the source code owner can restore the source code through watermarking extraction and embedding mechanism. Software watermarking mechanism introduces a CFFO which include CBO and CFO novel identifier in the software for protecting them from theft and piracy. So, the watermark or identifiers are included in the code without affecting the functionality of the software.

1. Introduction

The process of static software watermarking begins with the generation of a watermark from the owner's identification information. The information is converted into a suitable representation, such as a binary sequence, numerical value, or structural pattern, depending on the selected watermarking algorithm. The watermarking algorithm then analyzes the software program and identifies suitable locations where the watermark can be embedded. These locations may include independent instructions, basic blocks, variables, constants, register assignments, or control-flow structures. The algorithm selects program structures that can be modified without changing the intended behavior of the software. After selecting the appropriate locations, the watermark is embedded through controlled modifications to the program's existing structure. Depending on the technique and components. The watermarked program is subsequently tested to ensure that it continues to produce the expected results. During ownership verification, a static analyzer examines the suspected software, extracts the embedded information according to the predefined rules, and compares it with the owner's registered watermark. A successful match can provide supporting evidence of ownership.

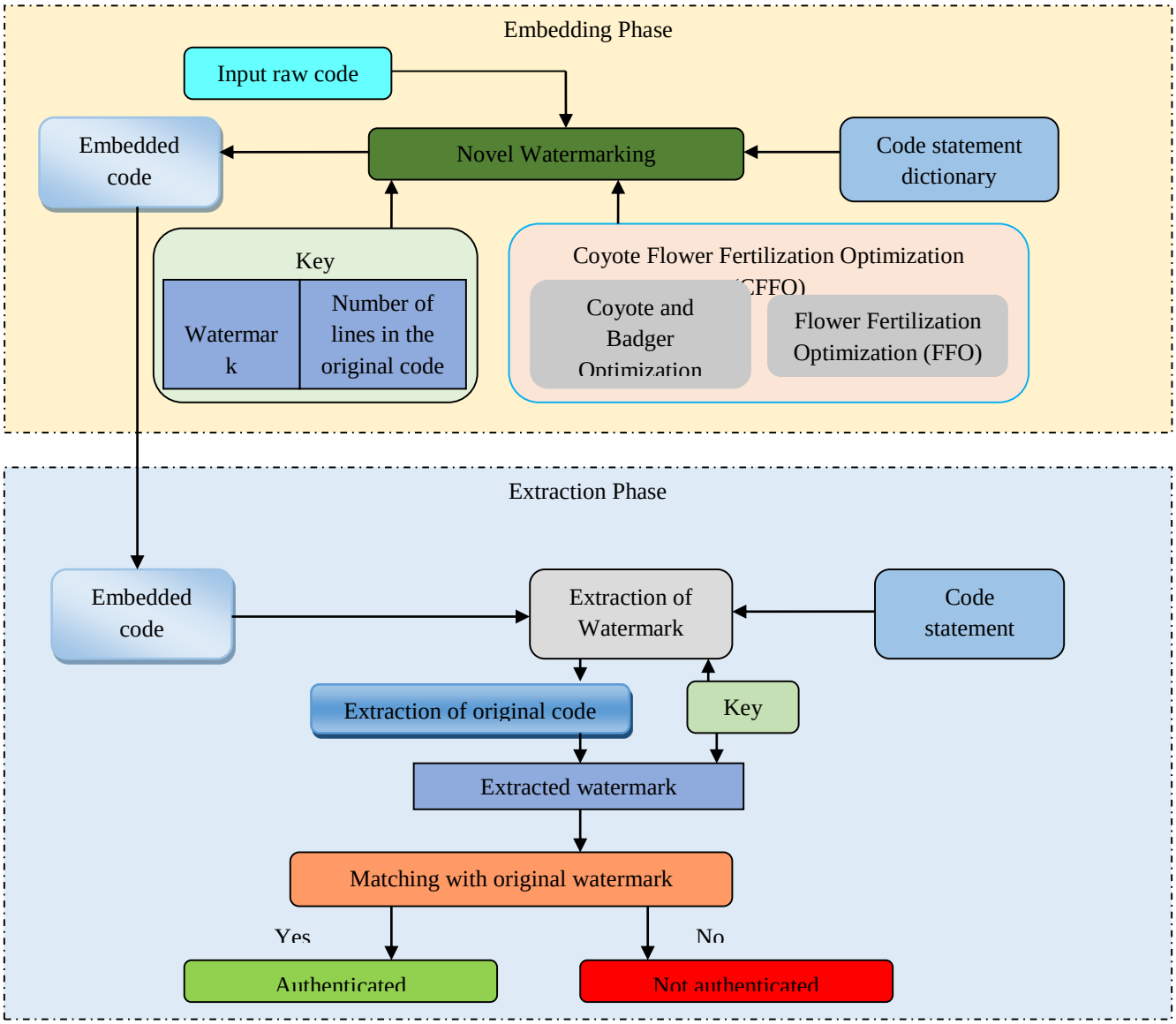
2. Literature Review

Control-Flow Graph (CFG) encoding represents watermark information through selected control-flow structures, graph properties, or transformations of the program's control-flow graph. Graph matching, graph optimization, constraint-solving methods, and evolutionary algorithms can be used to identify suitable embedding configurations. Although CFG-based watermarking may provide structural robustness, control-flow restructuring, code obfuscation, and compiler optimization can alter the embedded signature and affect extraction reliability [1], [2]. Instruction scheduling optimization encodes watermark information through the ordering of independent instructions within basic blocks while preserving data and control dependencies. Combinatorial optimization, list scheduling, and constraint-based methods can be employed to identify suitable instruction orderings. This approach may reduce additional code and runtime overhead when existing instructions are rearranged; however, subsequent compiler optimization may reorder instructions and destroy the watermark [3]. Source-code statement selection focuses on identifying suitable statements or embedding locations for watermark insertion while maintaining the original program's semantics. Genetic algorithms, swarm intelligence, evolutionary algorithms, and other metaheuristic approaches can be used to search for embedding locations that balance watermark extraction accuracy, robustness, and code modification. Nature-inspired optimization algorithms, such as the Flower Pollination Algorithm introduced by Yang and the Coyote Optimization Algorithm proposed by Pierezan and Coelho, provide possible foundations for searching candidate solutions, although their original publications do not directly establish their effectiveness in software watermarking. Finally, optimization-based hybrid watermarking combines multiple embedding strategies, such as graph-based encoding and source-code statement selection, to improve the balance between robustness, imperceptibility, extraction reliability, and computational cost. Multi-objective optimization and hybrid metaheuristics may help address competing requirements, but they can increase algorithmic complexity and implementation effort. The general principles of software watermarking and software protection discussed by Collberg and Nagra [1] provide a foundation for investigating such approaches. The effectiveness of these techniques should be established experimentally by evaluating watermark detection accuracy, code modification, execution overhead, and resistance to code transformations.

3. Proposed Coyote Flower Fertilization Optimization for code watermarking

Code watermarking involves embedding a unique identifier into software to safeguard intellectual property. However, current approaches frequently face challenges, such as limited robustness against code modifications, significant performance overhead, and poor imperceptibility. Many existing methods also necessitate access to the original code for verification purposes. Furthermore, implementing these techniques was complex and difficult to seamlessly integrate into existing development workflows. To tackle these challenges, this research proposes a CFFO_Wmark for code watermarking. This method includes the Embedding phase and the Extraction phase. In the embedding phase, novel watermarking is done by embedding the input raw data obtained from [4] into a code selected from a code statement dictionary using the key. Here, code watermarking is performed by employing the proposed CFFO. Moreover, CFFO is developed by merging CBO [5] and FFO [6]. Moreover, CFFO satisfies three objectives, such as Transparency, Harmless (code usability) and Robustness (accuracy). After the completion of embedding, the obtained embedded code is fed into the extraction phase. In the extraction phase, embedded code will undergo extraction, in which secret codes are extracted by using the key.

Finally, secret codes are obtained. Figure 1 shows the block diagram of CFFO_Wmark for Code watermarking. Figure 1 shows CFFO technique for code watermarking as given below,



The embedding phase [7] of code watermarking is the process of securely inserting a hidden watermark into the source code or executable without affecting its original functionality. In this phase, the input data to be watermarked is first encoded using a secret key. This encoded data is then mapped to specific code patterns or statements selected from a predefined code statement dictionary. These selected code statements are carefully integrated into the program at appropriate locations using the proposed CFFO. The CFFO ensures that the embedded watermark meets three essential criteria: Transparency, Harmlessness, and Robustness, by ensuring that the watermark can survive typical code transformations. The output of this phase is a watermarked version of the original code, ready for distribution or deployment, with the embedded watermark securely concealed within its structure.

3.2 Extraction phase

The extraction phase [8] of code watermarking is a critical step where the embedded watermark is retrieved from the watermarked code to verify ownership or authenticity. This phase begins by inputting the potentially altered or transformed code, along with the original secret key used during the embedding process. The key plays a vital role in identifying and interpreting the specific code patterns or structures that contain the embedded watermark. Using this key, the extraction algorithm scans the code and matches segments against the code statement dictionary to decode the hidden watermark data.

4. Conclusion

The proposed CFFO_Wmark method protects software by embedding a secret watermark through a key-guided statement-selection process. During embedding, the system considers candidate statements from a predefined dictionary and uses the hybrid CFFO optimization algorithm to search for a suitable statement and position. The selected statement is incorporated into the original code, with the aim of maintaining program functionality, keeping the watermark unobtrusive, and enabling later recovery. During extraction, the watermarked code, secret key, and statement dictionary are used to locate the embedded statement and reconstruct the watermark. The recovered watermark is then compared with the expected value to support ownership verification. CFFO combines CBO-inspired and FFO-inspired search mechanisms, including exploration, local refinement, movement toward promising candidates, fitness evaluation, and termination. The effectiveness of this method depends on a precisely defined encoding and decoding process, a consistent key-generation rule, behavior-preserving statement selection, and experimental evaluation against code modification and watermark attacks.

References

1. C. Collberg and J. Nagra, Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection. Upper Saddle River, NJ, USA: Addison-Wesley, 2009.
2. C. Collberg, C. Thomborson, and D. Low, “A taxonomy of obfuscating transformations,” Department of Computer Science, University of Auckland, Auckland, New Zealand, Tech. Rep. 148, 1997.
3. J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 6th ed. Cambridge, MA, USA: Morgan Kaufmann, 2017.
4. "Software-artifact Infrastructure Repository dataset," [Online]. Available: https://sir.csc.ncsu.edu/php/index.php. [Accessed June 2025].
5. M. Khata, M. El-Gamel, A. I. Saleh, A. El-Shenawy and A. H. Rabie, "Coyote and Badger Optimization (CBO): A natural inspired meta-heuristic algorithm based on cooperative hunting," Communications in Nonlinear Science and Numerical Simulation, vol. 140, pp. 108333, 2025.
6. H. Albedran, S. Alsamia and E. Koch, "Flower fertilization optimization algorithm with application to adaptive controllers," Scientific Reports, vol. 15, no. 1, pp. 6273, 2025.
7. C. Collberg and C. Thomborson, "Software watermarking: Models and dynamic embeddings," pp. 311-324, 1999.
8. W. Zhu and C. Thomborson, "Extraction in software watermarking," pp. 175-181, 2006.