Impact Factor: 9.9 ISSN-L: 2544-980X

Improving the Steganographic Robustness and Capacity of Text Containers Based on Preliminary Huffman Entropy Coding

Mokhirakhon Khusanova
Fergana State Technical University
Senior Lecturer, Department of Software engineering and Cybersecurity
Fergana, Uzbekistan
mokhira.khusanova@gmail.com

Abstract: This study proposes and validates a methodology aimed at optimizing the imperceptibility and capacity characteristics of steganographic systems operating within text environments. The underlying principle involves the sequential integration of the Huffman prefix compression algorithm and the injection of hidden data via a system of Unicode zero-width control characters. Based on the computational experiment results, a 52.2% reduction in the volume of introduced artifacts was recorded. This factor proportionally minimizes the probability of detecting the hidden transmission via statistical steganalysis (including the χ^2 criterion test). The high entropy level of the generated bitstream ensures data masking as natural noise. The paper provides a detailed description of the software architecture of the complex developed in Python, alongside structured evaluation results of its efficiency.

Keywords: steganography, Huffman coding, Unicode zero-width characters, data compression, steganalysis, chi-square criterion, information security, entropy.

Introductio

Ensuring confidentiality during data exchange across global networks necessitates the systematic modernization of existing information security paradigms. Digital steganography occupies a distinct niche within this domain; unlike cryptography, which focuses on obscuring the semantic content of a message, steganography aims to mask the very fact of communication.

Media files are traditionally used to establish hidden channels due to their high inherent redundancy. However, the routine transmission of anomalously large images or audio streams within corporate and institutional networks is rapidly flagged by modern Data Loss Prevention (DLP) systems. Conversely, textual documents form the baseline of standard network traffic, rendering them highly viable containers for embedding steganograms.

The primary challenge in text steganography stems from severe capacity constraints, as any modification of characters or punctuation marks is immediately apparent under visual inspection. Applying non-displayable Unicode control codes (Zero-Width Characters) addresses this limitation. Nevertheless, direct injection of uncompressed bitstreams introduces distinct structural anomalies into the file matrix, which are easily detected by automated statistical steganalysis tools.

Therefore, this study aims to develop and programmatically implement a method that enhances both the statistical imperceptibility and the effective capacity of text containers by subjecting the hidden data to preliminary entropy processing using the Huffman algorithm.

Methodology

The Huffman method relies on the lossless entropy compression of discrete messages by utilizing variable-length code combinations. The underlying mechanism is governed by the source probability distribution: symbols with a higher frequency of occurrence are assigned shorter prefix codes, thereby minimizing the weighted average length of the resulting codeword.

Let the hidden payload be defined as a string S over an alphabet ***A = {a1, a2, ..., an}***. Each element ai is associated with a specific frequency f(ai). The construction of the optimal code tree follows an iterative procedure applying a priority queue:

1. A sorted set of isolated leaf nodes is initialized, where each node corresponds to a character and its respective weight.
2. The two nodes containing the lowest weight coefficients (f1 and f2) are sequentially extracted from the structure.
3. A parent node is generated with a combined weight of fnew = f1 + f2, and the extracted elements are assigned as its child nodes.
4. The newly created node is inserted back into the priority queue. This sequence repeats until a single root element remains.

Within the context of a steganographic framework, this approach solves two fundamental challenges simultaneously:

- Capacity Optimization: The physical volume of the embedded data block is reduced by a factor of 2 to 3.
- Robustness Enhancement: The compressed bit sequence exhibits properties akin to random noise (maximizing entropy), which fundamentally degrades the efficacy of frequency-based steganalysis.

Implementation of Invisible Unicode Markers

To integrate the bitstream into the textual field, a specialized set of Unicode control codes devoid of graphical representation is utilized:

- U+200C (Zero Width Non-Joiner) — represents a logical zero ("0");
- U+200D (Zero Width Joiner) — represents a logical one ("1");
- U+2060 (Word Joiner) — functions as a service delimiter marking the boundaries of the stego-block.

Because these markers remain unrendered during standard text display, the container maintains absolute visual authenticity while shifting only its physical size in bytes.

Frequency Calculation f(ai) → Code Tree Generation → Code Mapping → String Compression → Stego-Injection

A critical condition for the proper execution of this system is providing the recipient with the actual code table, as accurate prefix decoding is impossible without the corresponding tree topology.

The steganographic software suite was engineered using Python 3.x. To accelerate performance when interacting with the minimum heap structure, the built-in heapq library was utilized.

Encoder Implementation Module:

```
import heapq
from collections import Counter
class Node:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None
    def __lt__(self, other):
        return self.freq < other.freq
def build_huffman_tree(text):
```

```
    frequency = Counter(text)
    priority_queue = [Node(char, freq) for char, freq in frequency.items()]
    heapq.heapify(priority_queue)
    while len(priority_queue) > 1:
        left = heapq.heappop(priority_queue)
        right = heapq.heappop(priority_queue)
        merged = Node(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
        heapq.heappush(priority_queue, merged)
    return priority_queue[0]
def build_codes(node, current_code, codes):
    if node is None:
        return
    if node.char is not None:
        codes[node.char] = current_code
    build_codes(node.left, current_code + '0', codes)
    build_codes(node.right, current_code + '1', codes)
```

Stego-Injector Implementation Module:

```
        BIT0 = "\u200C"  # ZERO WIDTH NON-JOINER → bit '0'
BIT1 = "\u200D"  # ZERO WIDTH JOINER     → bit '1'
SEPARATOR = "\u2060"  # Start/End marker
def embed_in_text(cover_text, secret_bits):
    invisible = SEPARATOR
    for bit in secret_bits:
        invisible += BIT1 if bit == '1' else BIT0
    invisible += SEPARATOR
    words = cover_text.split(' ', 1)
    if len(words) == 1:
        return cover_text + invisible
    return words[0] + invisible + ' ' + words[1]
```

Experimental Verification and Results Analysis

The phrase "стеганография и сжатие данных" (length — 29 characters), possessing a highly contrastive distribution of symbol frequencies, was selected as the secret payload. The baseline text container comprised a character array with a length of 84 characters.

During the initial phase of the execution cycle, the system generated the frequency-code matrix presented in Table 1.

Table 1. Analysis of character frequency distribution and derived prefix codes.

Character	Frequency	Huffman Code	Length (Bits)
' ' (space)	3	000	3
и	3	001	3
н	3	010	3
а	4	110	3
с	2	0110	4
е	2	0111	4

г	2	1010	4
т	2	1110	4
ы	1	10000	5
о	1	10001	5
ф	1	10010	5
х	1	10011	5
д	1	10110	5
р	1	10111	5
ж	1	11110	5
я	1	11111	5

To formally demonstrate the advantages of the proposed methodology, comparative calculations were performed between a standard character-by-character insertion scheme (UTF-8 encoding at 8 bits per character) and the developed compression-injection method.

Table 2. Comparative analysis of functional steganographic parameters.

Monitored Parameter	Classical Approach	Proposed Methodology
Original text length (characters)	29	29
Final steganogram volume (bits)	232	111
Compression ratio (Kc)	1.00	2.09
Net capacity gain	0.0%	52.2%
Number of altered container positions	232	111
Resulting stego-text size (bytes)	318	197

The empirical data compiled in Table 2 establishes that integrating the Huffman algorithm reduced the total number of required hidden Unicode codes from 232 to 111 units. This optimization yields a container capacity saving of 52.2%.

Minimizing the physical density of the embedded zero-width characters directly enhances system immunity against steganalytic attacks relying on the χ^2 (chi-square) criterion. Under standard embedding conditions, the distribution density of hidden markers inherently mirrors the linguistic features of the source language (i.e., high frequencies of specific letters). In contrast, introducing a pre-compressed, high-entropy bitstream flattens the distribution of invisible markers into a uniform structure indistinguishable from baseline background noise. This uniformity effectively neutralizes automated steganography detection algorithms.

Despite these optimal outcomes, the proposed framework introduces specific architectural dependencies that must be contextually addressed:

1. Decoding on the recipient side is mathematically impossible without access to the original Huffman tree topology. Transmitting this tree within the body of the same container expands the

payload overhead. A more viable solution involves utilizing a static frequency table pre-calculated across a representative text corpus of the target language.

2. When processing atypical messages characterized by broad alphabets and highly uniform symbol distributions, the compression efficiency decreases. Under such boundary conditions, it is appropriate to employ alternative compression algorithms, such as arithmetic coding or dictionary-based approaches from the LZ77/LZ78 families.

Conclusion

This study designed and empirically validated a text-based steganographic information security framework that systematically combines Huffman entropy coding with data injection into the non-displayable spaces of a text container.

Experimental results verified a 52.2% reduction in embedding density, which simultaneously expands the effective throughput of the communication channel and minimizes structural file distortion. The elevated entropy of the generated bitstream provides robust protection against statistical steganalysis. The operational integrity of the system was verified across all operational stages, spanning encoding and injection to extraction and complete decompression of the target message.

References

1. Gribunin, V. G., Okov, I. N., & Turintsev, I. V. (2017). Digital Steganography. Moscow: SOLON-Press. (In Russ.)
2. Konakhovich, G. F., & Puzyrenko, A. Yu. (2016). Computer Steganography: Theory and Practice. Kyiv: MK-Press. (In Russ.)
3. Salomon, D. (2014). Data, Image and Sound Compression. Moscow: Tekhnosfera. (In Russ.)
4. Qurbonaliyevna, X. M. (2025). Modern solutions for building VPN networks in information and communication systems of enterprises and organizations. Al-Farg‘oniy Avlodlari, 1(2), 122-125.
5. Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. Proceedings of the IRE, 40(9), 1098-1101.
6. Sotvoldiyeva, D. B., & Husanova, M. K. (2021). Analysis of discrete convolution in the MATLAB program. In Modern Trends in the Development of Fundamental and Applied Sciences (pp. 187-191).
7. Por, L. Y., Wong, K., & Chee, B. (2021). Information hiding in text document using zero-width characters. Journal of Computer Science, 17(3), 244-253.
8. Khusanova, M. K. (2022). Network security and monitoring. Research Focus, 1(4), 177-183.
9. Al-Nefaiee, A. H., & Al-Shatnawi, A. M. (2023). High-capacity text steganography method based on zero-width character and Huffman coding. International Journal of Advanced Computer Science and Applications, 14(5), 412-419.