КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 2 (34), 2026
ISSN 2663 – 4023
DOI 10.28925/2663-4023.2026.34.1319

Yevhenii Ponomarenko
PhD Student
Department of Cybersecurity and Information Protection
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine
ORCID: 0009-0000-2647-3381
allagrir@gmail.com

Oleksandr Laptiev
Doctor of Technical Science, Senior Researcher, Associate Professor
Department of Cybersecurity and Information Protection
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine
ORCID: 0000-0002-4194-402X
olaptiev@knu.ua

IMPLEMENTATION ANALYSIS OF ENCODING MANIPULATION METHOD IN DEFLATE COMPRESSION ALGORITHM

Abstract. In this paper, we use a novel Encoding Decision Chain Model to develop a theoretical method of encoding steganographic data in a Deflate compression algorithm. Specifically, we propose three major strategies that could be used for embedding covert data in a compressed data stream that rely on detecting decision points within the compression encoding process and manipulating their outcomes to achieve objectives secondary to the primary compression objective of reducing the amount of redundancy in source data as much as possible. The first technique, stored block length manipulation, artificially constraints the window of possible Deflate block lenghts to a smaller range and using those to embed secondary information. Second technique, back-reference manipulation, takes advantage of the ability to represent a repetitive string of symbols as either literals or back-references pointing at a specific chunk of previous data within the context window. By reducing the length of the back-reference, it is possible to emit several of them and utilize their lengths to encode steganographic information. The third technique, Huffman code tree structure manipulation, involves manipulating the values of relative frequencies of symbols of the alphabet stored within dynamic Huffman code compressed blocks to encode steganographic information. In this paper we include general analysis of each technique's effectiveness with regards to the amount of encoded steganographic information against the amount of additional data needed to be stored within the carrier to ensure that no data is lost or altered as a result of proposed encoding manipulation techniques. Proposed methods have an advantage of not altering the carrier data directly, resulting in bit-perfect extraction of the original compressed data, reducing the surface for detecting this kind of steganographic apporach.

Keywords: steganography; information hiding; compression algorithms; data manipulation; Deflate; information protection; covert messaging; information theory; data analysis; data streaming

INTRODUCTION

Cryptography and encryption are ubiquitous in the modern world. They are used to protect the privacy of data transmitted over unsecure connections, prove the origin of the transmission party, and protect the secrets that should not be known by everyone. It is nearly impossible to interact with digital systems without interacting with a cryptography or encryption system. The data on our phones, computers, and laptops is protected behind a password, hashed and stored securely in a practically irreversible way, or used as a key to reverse the encryption on the rest of the data volume. Credit cards encrypt our payment information to make sure only us — authorized parties — can perform financial transactions on our accounts, protecting our money. Even the act of navigating the internet online, accessing the world wide web, is primarily done through a secure connection, ensuring that no middleman can peek through our data and masquerade as a real service, or simply observe the secrets we share as they pass on their way to the original server.

One of the universal properties of any encrypted data is that it is, in theory, indistinguishable from a random noise, preventing anyone from gaining any information about the contents hidden within the message unless they

know the key. While useful as a means of preventing unauthorized access, the result is undeniably recognizable as encrypted. Anyone looking at the encrypted data that is being transmitted, or has been stored, will be able to tell that there is something hidden behind the encryption. This fact alone may already become a source of concerns or problems. As the common line of thought may go, there is little reason to put a lock on a chest that doesn't have anything valuable in it. Therefore, the mere knowledge that something has been protected may induce third parties to seek measures to break through such protection. Though mathematically, current encryption methods, algorithms, and standards have been rigorously proven and tested to be impenetrable to computational attacks, it doesn't change the fact that *someone* knows the code, and that *someone* is a person that can be intimidated, tortured, and/or otherwise compelled to disclose the secret. As such, the need to hide the presence of a secret message itself arises.

The problem statement. Preventing third parties from detecting a presence of a secret data stream is a challenge that is tackled by the field of steganography — "the art or practice of concealing a message, image, or file within another message, image, or file" [1]. While many techniques have been developed, analysis of which will be provided in the next subsection, they all have the same limitation: they corrupt the perceptible information being transmitted in each file. Be it pixels, sound intensity values, or something else, most existing techniques rely on changing those in a way that happens to encode useful information, only able to be decoded by a person who knows where and how to look. The modifications of data on the perceptual layer leads to possibilities of statistical analysis that may highlight carrier files that have secret message embedded within them.

Other methods of steganography exist that do not introduce any perceptible changes to the output file, but they instead rely on adding special segments to the file that are treated like a valid block in the carrier file's encoding but largely ignored for the purposes of rendering the actual perceptual information to the end user, e.g. [2]. Those are either text blocks, comment blocks, or other kinds of blocks that the decoder may ignore for the purpose of decoding the file. While avoiding changing anything on the perceptive layer, these introduce separable artifacts that clearly stand out to the attacker who might try to find our secret message.

As such, the problem may be stated as follows: how can we encode information within digital data streams in a way that avoids both changing perceptual characteristics of a digital file (changing the pixel/sound sample intensity values, etc.) and adding separable chunks of data that stand out under basic steganalysis?

Analysis of recent studies and publications. Traditional steganography approaches can be split broadly within three domains: spatial domain, transform (frequency) domain, and distortion techniques [3].

Spatial domain focuses on editing samples of cover media directly. It achieves relatively high embedding capacity, allowing it to store a large amount of data within the cover media, but it's most susceptible to changes in the file. Processes like format conversion, cropping, rotation, shearing, time dilation, and other media editing techniques tend to destroy the embedded message. Examples include techniques like Least Significant Bit (LSB) [4], Pixel Value Differencing (PVD) [5], Gray Level Modification (GLM) [6], and Reversible Data Hiding (RDH) [7], among others.

Transform (frequency) domain techniques, instead of manipulating raw cover media elements directly, rely on applying modifications to the cover media's representation in the frequency domain. Since cover media is usually representing a media object meant to be consumed by humans through our senses, high-frequency variations, modifications, changes, and instabilities tend to have less perceptual ability through humans' visual or auditory senses. This means that converting a cover media to the frequency domain allows us to perform data manipulation precisely in said high-frequency areas, resulting in lesser chance of data manipulation to be noticeable to the outside observer. The main techniques, covered by [3], [8], include Discrete Fourier Transform (DFT) [9], Discrete Cosine Transform (DCT) [10], Discrete Wavelet Transform (DWT) [11], and others.

A new direction of research has been recently introduced by [12]: coverless steganography. Instead of modifying target cover media to match a certain set of characteristics that we want to transmit as our message, we collect a large library of images that naturally have various characteristics and then search through said library to retrieve the image that already composes the message we want to transmit. So-called mapping-based coverless steganography is one of the two major categories. The other main category, classified as generation-based coverless steganography by [13], relies on using generative artificial intelligence to create images with required properties. First proposed by [14], it has seen significant further developments.

As for detection, [15] discusses prospects of detecting covert signals, while [16] gleans further insight into pseudo-random number generation and its impact on detecting steganographic manipulation.

The article's goal. With Encoding Decision Chain Model (EDCM) already formalized and Encoding Manipulation Method (EMM) described in [17], this article aims at creating a specific implementation of EMM within Deflate compression algorithm.

THE RESULTS AND DISCUSSION

To encode is “to convert (something, such as a body of information) from one system of communication into another”, especially “to convert (a message) into code”, or “to convey symbolically” [18]. Encoding Chain Decision Model (EDCM), proposed in [17], treats an encoding process as a decision chain, where each symbol of a given encoding is described as a result in a decision chain produced by the encoding software, library, or function.

Compression algorithms, such as Deflate [19], are a form of information encoding that transforms raw data bytes into a compressed data stream, encoded with symbols of various bit lengths, which have a specific, unique meaning that is defined by the encoding format itself. To perform such encoding, one would need a program — encoder — that does the encoding. There are various different implementations of Deflate encoders [20]–[25], with the first one was developed by Phil Katz in 1989 [22]. The reference implementation, provided by RFC 1951 [19], that defines the Deflate algorithm, is stored within the zlib library [23].

Before we can discuss how EMM can be used within the Deflate compression algorithm, we need to first analyze how it works.

Deflate encoding analysis. Deflate encoded data stream consists of a series of blocks. Each block starts with a sequence of 3 bits. First bit tells the decoder whether this block is the last block in the data stream. The other two bits represent the block type. There are three block types defined by the standard:

Uncompressed blocks (BTYPE = 00): these blocks start with a header that contains 2 byte LEN value of the length of the block, as well as a 2 byte NLEN value of one’s complement of said length. After that, these of bytes are stored as-is in the data stream.

Fixed Huffman tree compressed blocks (BTYPE = 01): the blocks use a standard Huffman tree that encodes 285 symbols. Symbols 0–255 encode their literal values, symbol 256 encodes the end-of-block value, and symbols 257–285 encode a back-reference, with additional bits that describe the distance and length of the back-reference, with the distance values having a bound between 1–32768 (2^{15}), and length values being bounded within the range 3–258 (3–(2^8+2)).

Dynamic Huffman tree compressed blocks (BTYPE = 10): similar to fixed Huffman tree compressed blocks, they use the same 0–285 symbol alphabet to encode data, but instead of using a static, standard-defined Huffman tree value length distributions, they allow the encoder to define its own tree. To do so, the standard stipulates that the encoder first needs to provide code lengths for the literal/length alphabet, then the distance alphabet, and after that it can use the defined Huffman tree to encode values.

The length of uncompressed blocks is limited between 1–65536 (2^{16}) bytes. The length of compressed blocks, whether they use fixed or dynamic Huffman trees, is not explicitly limited by the standard, instead being defined by the location of the end-of-block symbol (code 256).

In compressed blocks, the back-references are required to have a length of at least 3. Any less and it would take more data to encode the back-reference than it would take to encode the values literally. An example of how back-references function in a compressed block of data is presented on Figure 1 below. Each back-reference encodes the entire span of characters it represents.

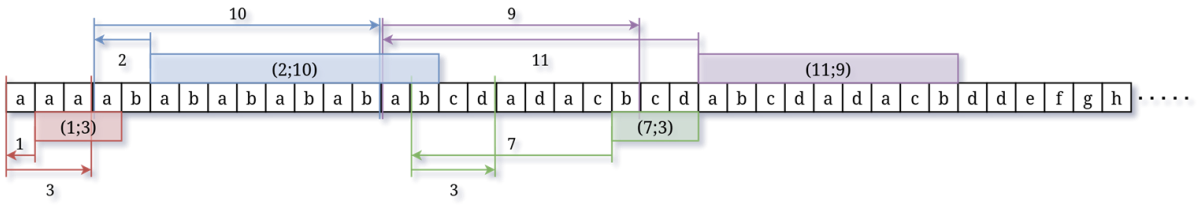

Figure 1 — Sample encoding process. Each coding block represents a distance-length back-reference element, with the distances and ranges illustrated

Decision manipulation. With the basics of Deflate encoding process laid out, we can use its details to encode a secondary, covert stream of data withing the compressed data.

One of the clearest ways this can be achieved is through the splitting of uncompressed chunks. Given the fact that their length is bounded by 2 bytes of data, we can use the length of the uncompressed block itself as a decision we can manipulate. This approach has a couple of considerations. First, each block adds 4 bytes of data to the output data stream — 2 bytes for the LEN value, and 2 bytes more for the NLEN value stored within the header. Second, the length value effectively expends said number of bytes within the uncompressed chunk of data,

including those in the block's body. As such, if a larger amount of uncompressed blocks is required to embed a certain value within the Deflate data stream, SHjE needs to explicitly limit the LEN values it will output to a smaller range, thus increasing the number of times uncompressed blocks will have to be used, and increasing the amount of blocks that will be used within the manipulated data stream. An example of such manipulation on a set of uncompressed blocks of data can be seen on Figure 2 below.

Similar to uncompressed data blocks, the same effect can be achieved by splitting compressed data blocks into multiple smaller blocks. The major difference in this case would be between the fixed and dynamic Huffman tree compressed blocks: fixed Huffman tree blocks do not have a header — other than the 2-bit block type identifier — instead immediately proceeding to outputting compressed data. Dynamic Huffman tree code blocks, on the other hand, require a rather lengthy header to include the data about the dynamic Huffman tree. Splitting the latter would require duplicating the header data for each new block, as well as outputting a new 256 symbol to signify an end of a previous block. The length of such compressed blocks can, therefore, be used to convey covert information.

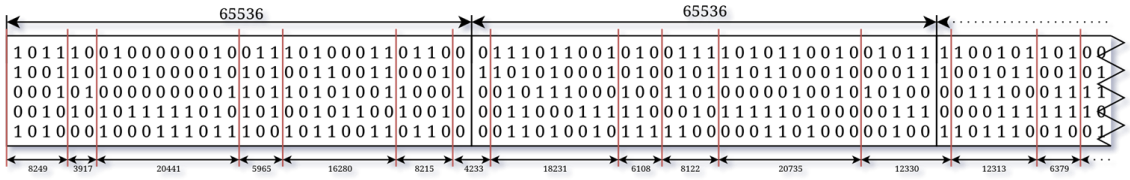

Figure 2 — Blocks of uncompressed data, with splits into smaller blocks, whose lengths represent covert data

Apart from existing considerations regarding the ratio between the amount of covert information transmitted through a creation of a new block and the amount of data required to encode a new block, as well as the considerations towards the density of such data points spread throughout the output stream, there exists another one. Namely, statistical properties of original data stream. Specifically, different parts of the data stream may contain different statistical patterns within them, and as such have a larger compression benefit from different Huffman tree codes. As such, forcefully enlarging the length of a given compressed data block will result in sub-optimal encoding performance of a data block that would've otherwise benefited from being encoded with a different Huffman tree. Therefore, SHjE must make sure to account for such considerations, if its goal is to change the resulting file size as little as possible.

The second way Deflate compression stream can be manipulated to encode a covert data stream is through splitting of back-references. Due to the fact that the back-reference length cannot be smaller than 3 symbols, there are two major cases we can consider: back-references of length 4–5, and back-references of length 6 and higher.

The first case — back-references of length 6 or higher — requires us to split one back-reference into several. While the decision space is going to grow proportional to the length of the back-reference, fundamentally, the remaining back-references can be treated as either of the two cases and processed recursively. An example is shown on Figure 3 below.

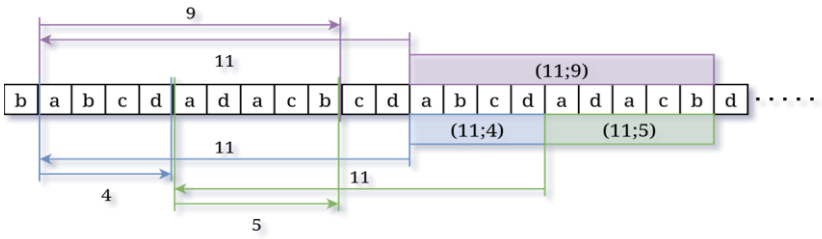

Figure 3 — A long back-reference split into two smaller ones

The second case — back-references of length 4–5 — provide us with another possibility: "shedding" or cutting away one or two symbols off of the back-reference, instead encoding it literally. In the case of back-references of length 4, this results in one of 3 choices: keeping them as-is, shedding away the first symbol, or

shedding away the last symbol. In the case of back-references of length 5, this results in one of 6 cases: keeping them as-is, shedding away only one symbol (either the first one or the last one), or shedding away two symbols: first two, first and last, or last two. An example is shown on Figure 4 below.

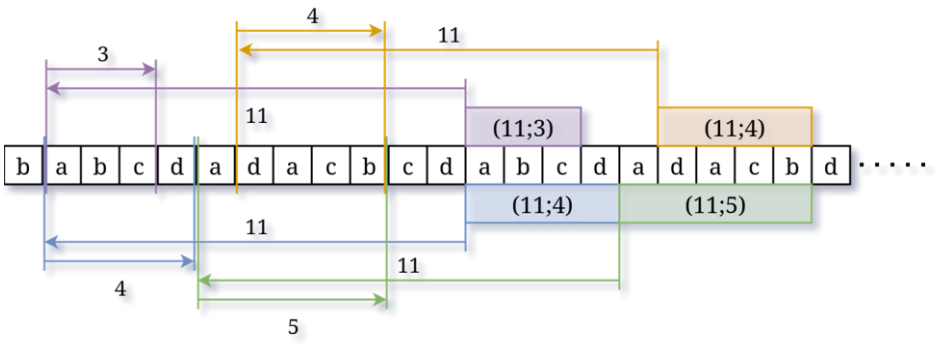

Figure 4 — Further splitting of back-references of length 4–5 into smaller ones

There are certain considerations with this approach, however. Unless compressing an already heavily-compressed or otherwise highly entropy-dense (e.g. random) original data stream, in practical scenarios the amount of back-references would highly exceed the amount of uncompressed data stream splits that can be performed within a set amount of file size. At the same time, utilizing arithmetic coding or other similar mixed-radix number systems, we can achieve at most $\log_2 3 \approx 1.58$ or $\log_2 6 \approx 2.58$ bits of information per every length 4 or length 5 back-reference, respectively, while necessitating anywhere between 12–31 bits per back-reference for fixed Huffman tree codes (7–8 bits for the length symbol, 0–5 additional bits for the length, 5 bits for the distance symbol, and 0–13 additional bits for the distance value). The only significant influence a dynamic Huffman tree structure would impose on the previous calculation would be changing the code lengths for the back-reference marker symbols (codes 257–285), though we do not expect it to be a significant difference to the result. Additionally, each symbol that has been shed from the back-reference would instead have to be encoded literally to preserve the original data stream.

The third avenue of possible encoding manipulation we can use to embed covert information within a Deflate compression stream is Huffman tree structure manipulation. The dynamic Huffman tree compressed block structure allows us to enter practically arbitrary values for Huffman code sizes for various values. These can be intentionally manipulated to produce a different Huffman coding tree. The unfortunate — and major — side effect of such an approach would be a significant increase in resulting file size, proportional to the amount of times affected codes are used within the data stream of a block. While necessary to point out as a theoretical possibility, its practical usage is limited to a very niche situation where the end user only needs a small amount of covert information to be transmitted (on the order of dozens of bits), while requiring a high degree of detection prevention.

CONCLUSIONS AND PROSPECTS FOR FURTHER RESEARCH

Deflate compression algorithm was designed as a precise formulation of how LZ77 [26], together with Huffman codes [27], can be applied in real-world computer systems. Its main objective is to compress an arbitrary input in a most optimal way. At the same time, there are many different implementations of encoders, each of which may produce slightly different results between each other, as well as have different outputs between versions, as new techniques are discovered to improve the opportunities for the most efficient possible encoding of information as a Deflate data stream. As such, it is possible to introduce inefficiencies in such encoding process in a way that allows us to encode a secondary, covert data stream.

While this research is purely theoretical, it creates a great prospect of further research. Specifically, it is possible to formalize exact techniques that should be used within a SHjE implementation to achieve such a covert channel encoding process that can be unambiguously decoded by a compatible receiver.

Additionally, there exist prospects of predictive encoding inefficiencies: throwing away back-references entirely, relying purely on the predictive capacity of the decoder to see where an encoder had a chance to use a back-reference, but elected not to. This would somewhat expand the capacity of SHjE, allowing for a higher

amount of data to be hidden within the resulting data stream, at the cost of higher computational complexity, possibilities of incorrect predictions, and reliance upon specific baseline encoder capabilities, restricted by its implementation and version.

Furthermore, an avenue of possible research would be a kind of predictive establishment of an approximate amount of covert data that could be encoded within the data stream for a given carrier file, encoding settings, as well as the amount of "bloat" this process would generate.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Merriam-Webster. (2025, September 18). *Steganography*. Merriam-Webster.Com Dictionary. https://www.merriam-webster.com/dictionary/steganography
2. Darwis, D., Fernando, Y., Mehta, A. R., Wamiliana, & Setiawansyah. (2025). Metadata-Based Video Steganography: Development of a New Model for Secure Information Embedding. *Engineering, Technology & Applied Science Research*, *15*(5), 27076–27088. https://doi.org/10.48084/etasr.11937
3. Laishram, D., & Tuithung, T. (2018). *A Survey on Digital Image Steganography: Current Trends and Challenges* (SSRN Scholarly Paper No. 3171494). Social Science Research Network. https://doi.org/10.2139/ssrn.3171494
4. Farhan Rafat, K., & Muhammad Sajjad, S. (2024). Advancing Reversible LSB Steganography: Addressing Imperfections and Embracing Pioneering Techniques for Enhanced Security. *IEEE Access*, *12*, 143434–143457. https://doi.org/10.1109/ACCESS.2024.3468988
5. Abdulazeez, Z. A. (2025). Implementation and Empirical Evaluation of Pixel Value Differencing (PVD) Steganography with Boundary Mitigation Techniques. *Journal of Kerbala University*, *22*(4), 113–127.
6. Kaw, J. A., Loan, N. A., Parah, S. A., Muhammad, K., Sheikh, J. A., & Bhat, G. M. (2019). A reversible and secure patient information hiding system for IoT driven e-health. *International Journal of Information Management*, *45*, 262–275. https://doi.org/10.1016/j.ijinfomgt.2018.09.008
7. R, Y. N. G., Shetty, N. R., & B, V. K. (2024). A Study and Analysis of Reversible Data Hiding Techniques. *2024 Second International Conference on Advances in Information Technology (ICAIT)*, *1*, 1–6. https://doi.org/10.1109/ICAIT61638.2024.10690366
8. Fkirin, A., Attiya, G., & El-Sayed, A. (2016). Steganography Literature Survey, Classification and Comparative Study. *Communications on Applied Electronics*, *5*(10), 13–22.
9. Ritonga, M. R., Siringoringo, M. F., & Situmorang, C. (2026). Implementation of the Discrete Fourier Transform (DFT) Steganography Method for Embedding Secret Messages in Image Media. *International Journal of Science and Informatics Technology (IJOSIT)*, *1*(01), 21–29.
10. Hameed, Y. J., & Baawi, S. S. (2026). Efficient audio Steganography based on Discrete Cosine Transform and Least Significant Bit Technique. *Proceedings of the 2026 2nd International Conference on Computing and Emerging Sciences, ICCES '26*, 346–353. https://doi.org/10.1145/3797491.3797506
11. Singh, J., & Singla, M. (2022). Image Steganography Technique based on Singular Value Decomposition and Discrete Wavelet Transform. *International Journal of Electrical and Electronics Research*, *10*(2), 122–125. https://doi.org/10.37391/ijeer.100212
12. Zhou, Z.-L. (2016). Coverless information hiding based on bag-of-words model of image. *Yingyong Kexue Xuebao/Journal of Applied Sciences*, *34*(05), 527–536. https://doi.org/10.3969/j.issn.0255-8297.2016.05.005
13. Xiang, X., Tan, Y., Qin, J., & Tan, Y. (2025). Advancements and challenges in coverless image steganography: A survey. *Signal Processing*, *228*, 109761. https://doi.org/10.1016/j.sigpro.2024.109761
14. Duan, X., & Song, H. (2018, February 10). *Coverless information hiding based on Generative Model*. arXiv.Org. https://arxiv.org/abs/1802.03528v1
15. Laptev, A., Laptev, S., & Lapteva, T. (2021). An Improved Method for Detecting Random Radio Signals by Deviations of the Main Signal Parameters. *Information Systems and Technologies Security*, (1 (5)), 37–45. https://doi.org/10.17721/ISTS.2021.1.35-43
16. Rahimova, I., Abbasquliyev, A. E., Sevriukova, Y., & Laptieva, T. (2025). Experimental Evaluation of the Efficiency of Cryptographic Generators of Pseudo-Random Numbers. *Information Systems and Technologies Security*, (2 (10)), 30–36. https://doi.org/10.17721/ISTS.2025.10.30-36
17. Ponomarenko, Y., & Laptiev, O. (2025). Mathematical model of steganography using suboptimal decisions in data compression algorithms. *Information Systems and Technologies Security*, *1*(9), 54–60. https://doi.org/10.17721/ISTS.2025.9.54-60

18. Merriam-Webster. (2026, June 4). *Definition of ENCODING*. Merriam-Webster.Com Dictionary. https://www.merriam-webster.com/dictionary/encoding
19. Deutsch, L. P. (1996). *DEFLATE Compressed Data Format Specification version 1.3* (Request for Comments RFC 1951). Internet Engineering Task Force. https://doi.org/10.17487/RFC1951
20. Biggers, E. (2026). *ebiggers/libdeflate* [C]. https://github.com/ebiggers/libdeflate (Original work published 2014)
21. *flate package - compress/flate - Go Packages*. (n.d.). Retrieved June 29, 2026, from https://pkg.go.dev/compress/flate
22. *PKWARE - PKZIP ZIP compression, ZIP programs, data file backup software*. (2006, March 13). https://web.archive.org/web/20060313041206/http://www.pkware.com/business_and_developers/compression/
23. Adler, M. (2026). *madler/zlib* [C]. https://github.com/madler/zlib (Original work published 2011)
24. ip7z. (2026). *ip7z/7zip* [C++]. https://github.com/ip7z/7zip (Original work published 2022)
25. Google. (2026). *google/zopfli* [C++]. https://github.com/google/zopfli (Original work published 2015)
26. Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, *23*(3), 337–343. https://doi.org/10.1109/TIT.1977.1055714
27. Huffman, D. A. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, *40*(9), 1098–1101. https://doi.org/10.1109/JRPROC.1952.273898

Отримано редакцією журналу / Received: 20.02.26
Прорецензовано / Revised: 28.02.26
Схвалено до друку / Accepted: 24.09.26