https://doi.org/ 10.31865/2709-840092025348874

УДК 004.9

Д.М. Самойленко,
кандидат фізико-математичних наук, доцент,
Одеський технологічний університет «ШАГ»
https://orcid.org/ 0000-0002-3205-1174

М.М. Суліма,
кандидат технічних наук, доцент
Одеський технологічний університет «ШАГ»
https://orcid.org/0009-0002-3404-9905

ПІДТВЕРДЖЕННЯ ІНТЕЛЕКТУАЛЬНОЇ ВЛАСНОСТІ НА ВИХІДНІ ПРОГРАМНІ КОДИ ЗАСОБАМИ ТЕКСТОВОЇ СТЕГАНОГРАФІЇ МОВОЮ JAVASCRIPT

Задачі підтвердження авторства чи виявлення плагіату набули поширення з появою генеративного штучного інтелекту. Чільне місце у таких задачах посідає авторство на програмні коди. Окрім захисту інтелектуальної власності, виникають перешкоди у навчальній та науковій діяльності, експертні організації вишукують нові методи засвідчення чи спростування оригінальності робіт. Метою даної роботи є розвинення, обґрунтування та випробування методики внесення авторських міток у програмні коди за стеганографічною технологією цифрових «водяних» знаків та ANSI X9.17. Запропонована методика дозволила забезпечити крипторгафічно надійну імовірність стійкості до підробок для коду, що мають принаймні 43 рядки, зі збереженням статистичної непомітності внесених міток. Наведена реалізація мовою JavaScript. Такі рішення дозволять підтверджувати авторство за окремими фрагментами кодів достатньої довжини.

Ключові слова: інтелектуальна власність, програмний код, цифрові «водяні» знаки, стеганографія, ANSI X9.17.

D.M, Samoilenko, M.M. Sulima
Odesa Technological University ‘STEP”

CONFIRMATION OF AUTHORITY ON SOURCE PROGRAM CODES USING TEXT STEGANOGRAPHY IN JAVASCRIPT

Authorship verification or plagiarism detection tasks have become widespread with the advent of generative artificial intelligence. Authorship of software codes occupies a prominent place in such tasks. In addition to protecting intellectual property, obstacles arise in educational and scientific activities, and expert organizations are looking for new methods of certifying or disproving the originality of works. The purpose of this work is to develop, substantiate and test the methodology for introducing copyright labels into software codes using steganographic technology of digital "watermarks" and ANSI X9.17.

The proposed methodics allowed to provide a cryptographically reliable probability of resistance to forgery for a code with at least 43 lines, while maintaining the statistical invisibility of the inserted labels. An implementation in JavaScript is provided. Such solutions will allow you to confirm authorship for individual code fragments of sufficient length.

Keywords: intellectual property, software code, digital watermarks, steganography, ANSI X9.17.

Постановка проблеми в загальному вигляді. Проблема захисту інтелектуальної власності у відкритих інформаційних мережах набуває глобального значення. Поява і поширення засобів штучного інтелекту, що базуються на аналізі та використанні опублікованих у мережі ресурсів, може призводити до появі авторських розробок у формуванні відповідей на питання користувачів і, як наслідок, недотримання авторських обмежень суб'єктом запитання. Поширені засоби підтвердження власності на електронні ресурси, як-то цифровий підпис або токен, можуть підтвердити лише право на об'єкт у цілому, без можливості виявити використання у його складі окремих частин, що належать іншому власнику. Більш того, незначні спотворення структури початкового об'єкту, що можуть бути внесені іншим користувачем, призводять до повного порушення цифрового підпису, що вже не може бути використано як підтвердження авторства.

Особливу роль у цій проблемі грають програмні коди як текстові об'єкти. Зазвичай програмний код одного автора не абсолютно повністю включається до творів іншого автора. І хоча таке теж поширене у вигляді підключених бібліотек чи модулів, все ж частіше вживаються окремі фрагменти вихідного коду. Інколи без зазначення початкового авторства фрагменту.

Для того щоб забезпечити захист авторства окремих фрагментів коду, а не весь твір у цілому, необхідно вживати додаткові заходи захисту.

Аналіз літературних даних та постановка проблеми. У роботі [1] запропонована методика виявлення плагіату у програмних кодах, призначена для виявлення використання засобів штучного інтелекту у програмуванні. Методика націлена на аудиторію навчального закладу та базується на статистичному аналізі робіт студентів з метою виявлення абсолютно однакових великих фрагментів кодів. Результати роботи не є безпосереднім засобом захисту кодів, проте, надають ґрунтовне підтвердження загальної проблеми плагіату, її масштабу та поширеності, а також актуалізують подальші дослідження.

У роботах [2-3] пропонуються різні методики впровадження стеганографічних міток у вигляді цифрових «водяних» знаків у програмні коди. У роботі [2] акцент робиться на обфускацію коду, у роботі [3] - на модифікацію імен змінних вихідного коду. Слід зазначити, що обфускація кодів не завжди є можливою, особливо, якщо у коді вживається об'єктна рефлексія із зазначенням назв літералів у вигляді програмних рядків, або при використанні платформ-орієнтованого коду в крос-платформних застосунках через кодові канали, де так само назви методів передаються рядковими даними. В обох роботах не робиться акценту на цілісності захисту. Можна підтвердити авторство усієї програми, але для вибраного фрагменту це значно ускладняється.

У роботі [4] аналогічна задача розглянута для підтвердження авторства творів живопису за допомогою методу k-найближчих сусідів. Одним з висновків, методу є врахування того факту, що при відсутності чіткої відповідності ознак, оцінка може бути сформована з дослідження найменш віддалених «сусідів». Даний підхід може бути адаптований для галузі, що розглядається, у разі спотворення прямо внесених ознак ідентичності задля збільшення робастності методики.

У роботах [5-8] розглядається інших підхід до імовірнісного підтвердження авторства шляхом введення різних метрик до алгоритмів. Так, у роботі [5] пропонується використання інтелектуального алгоритму, що навчається на прикладі репозиторіїв відкритого коду. Алгоритм, за звітом авторів, дозволяє визначати результати щодо авторства з точністю у діапазоні 92-96%. Проте, вимагає великої бази для навчання, яка збиралась авторами за 9 років історії з близько 2000 репозиторіїв. Очевидно, що дана методика буде ускладнена для нових авторів, що тільки починають публікацію репозиторіїв. Аналогічне дослідження виконане у роботі [6], тільки в якості основної моделі навчання застосована авторська розробка «бінарний фреймворк», що базується на вивченні стильових особливостях різних авторів програмного коду.

У роботі [7] вживається інструмент лексичного аналізу даних щодо програмних кодів та використовується поетапна обробка. На першому етапі визначається множина з найбільш імовірних авторів, на другому застосовуються додаткові інструменти дослідження.

У роботі [8] використане максимальне заглиблення до компонентного аналізу кодів: операцій, циклів, масивів, патернів, виразів, ідентифікаторів, методів та класів. Також застосоване машинне навчання для прийняття підсумкового рішення.

Аналіз літературних даних дозволяє дійти висновку, що більшість досліджень спрямовані на дві групи методів визначення авторства – глобальну та імовірнісну. Перша група вводить певні мітки до програмних кодів та відстежує їх цілісність, друга вимагає великої бази попереднього навчання інтелектуальних алгоритмів. Актуальним залишається задача утворення розподілених міток, які дозволять гарантовано (або з криптографічною надійністю), без потреби використання попередніх творів автора довести авторство не лише на весь програмний комплекс, а й на його складові частини та фрагменти.

Метою дослідження є розроблення та експериментальне випробування методики впровадження надійних стеганографічних міток до програмних кодів. Гіпотезою дослідження є можливість методики до встановлення авторства за фрагментом програмного коду при його повному копіюванні до іншого джерела.

Для досягнення мети були поставлені такі завдання:

– розробити та обґрунтувати методику генерації стеганографічних міток з контролем цілісності, розглянути способи їх впровадження у коди;

– провести практичні випробування методики на програмах різного розміру;

– оцінити надійність підтвердження авторства фрагментів кодів у залежності від їх розмірів.

Об'єктом дослідження виступає авторська інтелектуальна власність, предметом є авторство на програмні коди. Методи дослідження поєднують інструментарій криптології та теорії імовірностей. За умови підтвердження гіпотези

це дасть можливість підтверджувати авторство не лише за повним твором, а й за його фрагментам.

Матеріали та методи досліджень. За основу методу було прийнято накладання на вихідні програмні коди криптографічної гамма-послідовності (гамми) за допомогою стеганографічних практик цифрових «водяних» знаків.

Криптографічна надійність гамми повинна забезпечити стійкість послідовності як до пошкодження, так і до реплікації без знання вихідних даних. Стеганографічна інтервенція має забезпечити непомітність впровадження міток та їх розподіл за всіма частинами тексту програми.

До гама-послідовності було висунуто додаткові вимоги:

- послідовність має демонструвати усі ознаки випадковості, не повинно спостерігатись статистичних закономірностей у ній;
- послідовність не повинна мати обмежень на довжину, за потреби повинна існувати можливість як завгодно довго подовжувати раніше створену гамму;
- послідовність має базуватись на парольній інформації, знання якої дозволить відновити усю послідовність, підтверджуючи право автора на неї;
- послідовність повинна бути стійкої до атак як зворотного, так і прямого прогнозування.

На відміну від літературних текстів, програмні коди є дуже незручними щодо стеганографії. Для них неможливі найбільш ємнісні практики символів однакового нарису, у тому числі недрукованих символів Unicode. Також для них діють правила оформлення коду, які обмежують застосування засобів різних відступів блоків. До правил слід віднести і загальні традиції щодо іменування змінних, зокрема, циклових, а також деяких операцій. Їх модифікація на кшталт описаній у роботі [3] заміні імен змінних може погіршити маскувальний ефект міток.

Також традиційними є способи розділення рядків. Один з стего-методів, що базується на різному порядку роздільних символів CR-LF або LF-CR було відкинуто через автоматичне виправлення подібних послідовностей редакторами програмних кодів.

Серед методів, що залишились, було обрано метод кінцевих пробілів, що полягає у додаванні певної кількості пробілів після останнього друкованого знаку програмної інструкції перед розривом рядка. Оскільки вирівнювання за шириною тексту у програмних кодах не практикується, даний метод не має наочних демаскуючих ознак. Більш того, практика рефакторингу показує, що реорганізація коду частіше за все не супроводжується видаленням таких кінцевих пробілів, які неодмінно виникають через перенесення рядків, що мали відступи ліворуч. Останнє додатково засвідчує вживаність методу.

Виклад основного матеріалу. У відповідності до сформульованих вимог до гамма-послідовності було запропоновано використання схеми формування псевдовипадкових послідовностей за стандартом ANSI X9.17 [9] з кількома модифікаціями.

Схема, що описана у [9], базується на послідовному обчисленні криптографічних перетворень зі зворотним зв'язком між блоками. Це забезпечує цілісність послідовності та можливість її повторної генерації при відомих початкових

даних. Однак, для задач, що поставлені у даній роботі, схема була змінена. Модифікований генератор наведено на рис. 1.

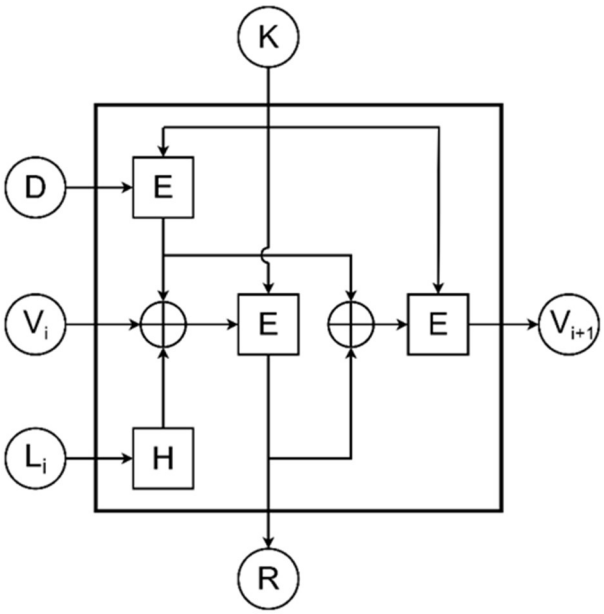

Рисунок 1 – Схема генератора послідовності: K – пароль користувача (ключ), D – мітка дати-часу, E – криптографічне перетворення, V – вектор ініціалізації, R – випадкове число (вихід генератора), L – рядок програмного коду, H – функція хешування, i – номер ітерації (номер рядка коду).

Було вжито ряд відмінностей від оригінальної схеми [9], що обґрунтовується наступними твердженнями:

- Додано вхідний параметр Li – рядок вихідного коду. З метою приведення його до спільної з криптографічним модулем бітової довжини виконується попереднє обчислення хешу від рядка. Семантика даного параметру відповідає вектору ініціалізації V, тому він подається на вхід за тим же каналом.

- Послідовність EDE (Encrypt-Decrypt-Encrypt) замінено на єдине криптографічне перетворення (E). Сучасні алгоритми шифрування [10] мають підвищену у порівнянні з вжитим у [9] алгоритмом DES кількість ітерацій, що знімає необхідність потрійного використання перетворень.

- Як наслідок попереднього пункту, два ключі, що необхідні для послідовності перетворень, замінено на один. Обґрунтуванням є те, що ентропія двох ключів DES (по 56 біт кожен) цілком покривається ентропією навіть найменшого ключа сучасних перетворень [10] (128 біт).

- Дата-час, яка у [9] обчислюється на момент генерації кожного блоку, приймається постійною для всіх блоків. Це вжито з метою підтвердження моменту накладання міток та усунення потенційних суперечок щодо першості авторів. Можна сказати, що момент часу грає роль другого ключа при створенні міток.

Вихідний параметр R (див. рис. 1) є станом криптографічної системи E та відповідає її налаштуванням (128, 256 чи 512 біт).

У якості криптографічних систем у схемі вжито національні стандарти України у галузі захисту інформації: криптографічний алгоритм «Калина» [10] та функція хешування «Купина» [11].

Схема має зворотні зв'язки як всередині блоку, так і між блоками через вектор V. Це забезпечує цілісність послідовності, що генерується, та неможливість її модифікації вставками чи вилученнями проміжних ланок.

Ключ K подається на всі ітерації перетворення. Для схеми зі зворотнім зв'язком це не є необхідним. Проте такий підхід обґрунтовується підвищеною стійкістю схеми до атак продовження, які дозволять генерувати послідовність за відомим початковим блоком, але без знання паролю (ключа), який було вжито для обчислення першого блоку.

Реальна розмірність вихідного параметра R обмежується стеганографічною ємністю контейнера, що ним є текст вихідного програмного коду. При застосуванні метода кінцевих пробілів, зазначеному у п.4, можна очікувати, що природно непомітними будуть пробіли, кількість яких відповідає кількості початкових пробілів, вжитих для вирівнювання коду. Обґрунтовуючи непомітність міток тим, що при рефакторингу коду часто ігноруються пробіли, які переносяться при злитті рядків, вирішено було зупинитись саме на таких обмеженнях.

Для різних мов програмування існують різні традиції вирівнювання, проте, зазвичай рівень відступів не перевищує 4-5 вкладень. Це відповідає 8-20 пробілам в залежності від абсолютної величини відступів (розмірів табуляції). Відповідно, мінімальна величина у 8 кінцевих пробілів може вважатись певною реалістичною межею, що не викликатиме підозр необізнаного користувача. Одержання зменшеної величини виходу виконане через використання найстарших бітів повної величини R.

З метою практичного дослідження методики було складено програму мовою JavaScript під оточення виконання NodeJS. У якості об'єкту внесення міток виступила сама ж програма, що виконувалась. Лістинг коду та результати виконання наведені у табл. 1.

Таблиця 1 – Лістинг коду програми внесення міток та результати її роботи (R – вихід генератора рис. 1, S – кількість кінцевих пробілів).

№	Рядок коду	R	S
1	import fs from 'node:fs';	2a2b1e49cfed3f71eec3e44ca2789cf1	2
2	import { Kalyna } from "./kalyna.js";	cc0902aead3d79cd48de49865a86bece	4
3	import { Kupyna } from "./kupyna.js";	85c5e700a354a0ee8869a5e1bc0d00b5	0
4	import { xorHex } from "./xor.js";	091c7e02d8db34afef6cdae868c97c6b	0
5		9bc02bb1179a1551d2e3485ba026cf9f	1
6	const k = new Kalyna(128, 128);	a5ba43b70c4eda39978bb076fc89e85d	2
7	k.setKeyHex("000102030405060708090A0B0C0D0E0F");	8a443ab0b6a2104b0a53ed5f47c0c4e2	0
8	const h = new Kupyna(128);	713baa312a3fd717dee13baea7cc469e	7
9	const timestamp = new Date().getTime().toString();	17704c5cbbe7de385e8f6e85c61279ce	1
10	console.log(timestamp);	d3f6a368cb685ba57cc7c81476535a82	5
11	const timestampEnc = k.encryptStrHex(timestamp);	c29609a358056f82cdbe71b078cba0b8	4
12	let v = '00000000000000000000000000000000';	7c8a5a20c4aac6035e6175f60113b0e5	7
13	let newLines = [];	303391d4c40473178d4d2237c1540759	3
14	fs.readFileSync('main.js', 'utf8').split(/\r?\n/).forEach(line => {	af6b79d1aeb177b632b6820d6e00d80b	2
15	line = line.trimEnd();	6f96f91e0f44af6d2ef6f68661bff2b5	6
16	let lineHash = h.digest(line, "UTF8");	177bee7789753280a115521fe6045678	1
17	let e1 = xorHex(v, lineHash, timestampEnc);	2aab28a84fd715a3455c0382918d7d7f	2

18	let r = k.encryptHex(e1);	9e199bf3b4a484108629bba42139ebf1	1
19	let spacesCount = parseInt(r[0], 16) % 8;	4ea7388dc189e0c78ab1a5650b8803a4	4
20	console.log(line, r, spacesCount);	839779b11f7418323038295d3bfedae5	0
21	newLines.push(line + ' '.repeat(spacesCount));	5580edac4e544f271eeb0440d5fde075	5
22	v = k.encryptHex(xorHex(r, timestampEnc));	3b96d8ac368619f16dc5207cc240e791	3
23	});	55c476480715953995fb78fdc6c7ac4e	5
24		5dc9ce06a6f71c138fe593c3aba41ce7	5
25	fs.writeFileSync('main.js', newLines.join("\r\n"));	8b9b4a30c382e81b306db1cc9e8f920a	0
26		cb844daa054e1cb1257be2124d5ce588	4

Для роботи було використано 128-бітні варіанти криптографічних модулів як шифрування, так і хешування. Відповідно, розмірність вихідного значення R також є 128 біт. Величини, одержані для кожного рядку коду наведені у табл. 1 у шістнадцятирічному представленні.

З метою усунення впливу самих кінцевих пробілів на результат хешування рядків коду, а також задля гарантування заданої їх кількості здійснюється попереднє видалення усіх кінцевих пробілів (рядок 15 таблиці). Аналогічна процедура має здійснюватися і при перевірці міток.

Дієвість зворотного зв'язку можна побачити за результатами оброблення повністю однакових порожніх рядків (5, 24, 26). Величини R, одержані для них мають суттєві відмінності, що призводить і до різної підсумкової кількості кінцевих пробілів.

На перший погляд, величини S у табл. 1 мають усі ознаки випадковості, очевидні закономірності не спостерігаються. Статистичний аналіз на виборці з 26 елементів не дає високу довірчу імовірність і призводить до обчислених моментів: середнє 3 (теоретично 3.50), дисперсія 5 (теоретично 4.08). У межах невеликої вибірки це вкладається у довірчий діапазон (з t-критерієм 1.71 для N=26) . Проте, для більш надійного дослідження статистичної рівномірності методики слід провести додаткові експерименти.

Ентропія множини з 8 елементів дорівнює 3 біти.

Надійність захисту фрагменту коду за допомогою міток можна оцінити за формулою «межі Сіммонса» [12]:

$$\log_2 p \geq H_2(MES) - H_2(E) - H_2(M) \ , \tag{1}$$

де p – імовірність підроблення мітки; H_2 – функція інформаційної ентропії, виражена у бітах: M – множина повідомлень, E – множина правил перетворень, MES – перетин множин M, E та множини джерел повідомлень (S).

В описаній схемі впровадження міток присутнє лише одне правило перетворення та одне джерело повідомлень. Враховуючи, що ентропія одиничної множини дорівнює нулю, з (1) отримаємо

$$\log_2 p \geq -H_2(M) \tag{2}$$

Ентропія множини повідомлень оцінена у 3 біти на один рядок коду. Для фрагменту коду, що має N рядків коду матимемо

$$\log_2 p \geq -3N, \quad p_{\min} = 2^{-3N} \ . \tag{3}$$

Достатнім можна вважати захист з ентропією від 64 біти, а криптографічно надійним – від 128 біти. Згідно з виразом (3) це вимагатиме наявності 22 та 43 рядків

коду відповідно. Це цілком узгоджується з типовими рекомендаціями рефакторингу щодо поділу коду на фрагменти подібної довжини. Це обґрунтовує дієвість запропонованої методики з урахуванням надійного захисту фрагментів коду типового розміру.

Момент часу запуску обчислення грає роль додаткового паролю і має бути зафіксований для можливості відновлення послідовності R у процедурі перевірці міток. Його значення виводиться у рядку 10 коду (табл. 1) та становить 1752667919256 для наведених результатів. Як видно з формату даних, момент часу фіксується з точністю до мілісекунди, додаючи надійності захисту.

Методика була побудована на базі надійної схеми стандарту ANSI X9.17 [9], що засвідчила свою дієвість протягом тривалого часу випробувань. Внесені зміни продиктовані сучасним розвитком національних засобів криптографії, а також призначенням методики. Тим не менш припускається, що схема може бути дещо удосконалена шляхом заміни більш вимогливого до ресурсів шифратора на хеш-перетворення. Для збереження захисту паролем може використовуватись режим автентифікації повідомлення (імітовставка) [11, додаток В]. Це обґрунтовує перспективність додаткових досліджень з порівняння показників надійності та, особливо, швидкодії різних реалізацій методики.

Також вбачається за доцільне проведення серій експериментів на великих обсягах програмних кодів з метою виявлення або спростування можливості виявлення статистичних закономірностей розподілу міток. Результати таких випробувань також будуть корисними як для загальної характеристики схеми, так і для оцінок її складових елементів – криптографічних блоків.

Основним недоліком стеганографічних методів є необхідність тримання у секреті як самі алгоритми, так і контейнери через які впроваджується інформація – «підсвідомий канал» [12]. У даному сенсі перспективним вбачається додатковий пошук та дослідження ємності альтернативних підходів, окрім методу кінцевих пробілів.

Методика може використовуватись як для окремих файлів з кодом, так і для їх репозиторіїв. Мова програмування не має значення, якщо в ній код поділяється на рядки та ігноруються кінцеві пробіли. У випадку повного ігнорування множинних пробілів як, наприклад, у HTML, методика може бути застосовна не лише для кінцевих, а й для роздільних пробілів.

За наявності можливості методика може бути імплементована у вигляді додатків (плагінів) для інструментів розробника (редакторів). Перспективним у практичному сенсі вбачається розробка відповідних плагінів для поширених редакторів коду. В такому разі процес внесення міток повністю автоматизується і не вимагатиме пост-обробки кодів перед публікацією у репозиторіях.

Робастність методики є низькою. При модифікаціях коду, у тому числі пробілів, актуальність міток руйнується. Методика може підтвердити авторство тільки у випадку збереження цілісності вихідного фрагменту коду.

З одного боку, саме така задача і була поставлена для методики, з урахуванням того, що програмні коди є предметом авторського права і захищаються «як є». Довільні зміни формально свідчать про участь інших авторів. З іншого боку, зберігається захист вихідних кодів, який ураховує момент часу внесення міток, і

стійкість таких міток є високою. Це залишає можливість підтвердити авторство вихідного коду шляхом аналізу міток у ньому та оцінити ступінь спотворення іншого блоку коду на відповідність нормам плагіату.

Робастність може бути покращено шляхом зменшення ентропії однієї мітки до 1 біту. В такому разі зменшується імовірність автоматичної заміни подвоєних пробілів та збільшується непомітність міток. Проте, у відповідності до (2), для забезпечення того ж рівня надійності фрагмент коду повинен містити щонайменше 60-100 рядків. Для проєктних робіт це цілком реальні показники, які обґрунтовують перспективність дослідження міток з різною ентропією.

Висновки. 1. Розроблено методику генерації стеганографічних міток та внесення їх до програмних кодів засобами цифрових «водяних» знаків за технологією кінцевих пробілів. Методика базується на стандарті ANSI X9.17, адаптованому до національних стандартів криптографії шифрування та хешування. Використання сучасних криптосистем дозволило розширити ентропію схеми до 128-512 біт і використовувати різні режими роботи. Оцінено ємність стеганографічного каналу величиною до 8 пробілів.

2. Методику імплементовано у вигляді програми мовою JavaScript. Проведено випробування методики на тій програмі, яка її реалізує. Використано 128-бітні версії криптографічних модулів шифрування та хешування. Показано, що закладений у методику зворотній зв'язок призводить до відмінності результатів при однакових вхідних даних, якими виступили порожні рядки коду. Статистичні показники міток вкладаються до довірчого діапазону, проте, при низькій довірчій імовірності. Показана перспективність подальших досліджень у цьому напрямі.

3. Проведено оцінку надійності підтвердження авторства фрагментів кодів у залежності від їх розмірів, запропоновано узагальнену формулу. При закладеній ентропії міток у 3 біти (8 комбінацій) криптографічна надійність захисту з імовірністю успішної атаки на рівні 2-128 досягається при наявності у програмі 43 рядків коду, у тому числі порожніх рядків. Відзначені перспективи подальших досліджень в галузі удосконалення швидкодії методики та її імплементації у вигляді програмних додатків (плагінів).

Список використаних джерел

1. Dickey, Ethan. (2025). The Failure of Plagiarism Detection in Competitive Programming. Cornell University, Computers and Society https://doi.org/10.48550/arXiv.2505.08244
2. Gehao Zhang, Eugene Bagdasarian, Juan Zhai, Shiqing Ma (2025). Disappearing Ink: Obfuscation Breaks N-gram Code Watermarks in Theory and Practice. Cornell University, *Cryptography and Security*, https://doi.org/10.48550/arXiv.2507.05512
3. Samoilenko, D. (2016). PERMUTATION-BASED POLYMORPHIC STEGO-WATERMARKS FOR PROGRAM CODES. *Proceedings of National Aviation University*, 67(2), 44–51. https://doi.org/10.18372/2306-1472.67.10431
4. A.A. Martynenko, A.D. Tevyashev, N.E. Kulishova, B.I. Moroz System analysis of the problem of establishing the authenticity and authority of painting works. ISSN

1681–6048 *System Research & Information Technologies*, 2022, No 1 https://doi.org/10.20535/SRIT.2308-8893.2022.1.04

5. Mohammed Abuhamad, Tamer Abuhmed, David Mohaisen, and Daehun Nyang. 2021. Large-scale and Robust Code Authorship Identification with Deep Feature Learning. ACM Trans. Priv. Secur. 24, 4, Article 23 (November 2021), 35 pages. https://doi.org/10.1145/3461666
6. Song, Qige & Zhang, Yongzheng & Ouyang, Linshu & Chen, Yige. (2022). BinMLM: Binary Authorship Verification with Flow-aware Mixture-of-Shared Language Model. https://doi.org/10.48550/arXiv.2203.04472
7. Czibula, Gabriela & Lupea, Mihaiela & Briciu, Anamaria. (2022). Enhancing the Performance of Software Authorship Attribution Using an Ensemble of Deep Autoencoders. *Mathematics*. 10. 2572. https://doi.org/10.3390/math10152572
8. Chunxia Zhang, Sen Wang, Jiayu Wu & Zhendong Niu (2017). Authorship Identification of Source Codes Conference: Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data DOI: https://doi.org/10.1007/978-3-319-63579-8_22
9. Ameen, Siddeeq & Al-Naima, Fawzi. (1977). EFFICIENT COMBINATION OF SEVERAL TECHNIQUES IN THE DESIGN AND IMPLEMENTATION OF A NETWORKS SECURITY SYSTEM.
10. Roman Oliynykov, Ivan Gorbenko, Oleksandr Kazymyrov, Victor Ruzhentsev, Oleksandr Kuznetsov, Yurii Gorbenko, Oleksandr Dyrda, Viktor Dolgov, Andrii Pushkaryov, Ruslan Mordvinov, Dmytro Kaidalov (2015). A New Encryption Standard of Ukraine: The Kalyna Block Cipher. IACR Cryptol. ePrint Arch. 2015: 650 https://web.archive.org/web/20170716144750/https://eprint.iacr.org/2015/650.pdf
11. Національний Стандарт України «Інформаційні технології. Криптографічний захист інформації. Функція гешування». ДСТУ 7564:2014. - Київ: Мінекономрозвитку України 2015. https://usts.kiev.ua/wp-content/uploads/2020/07/dstu-7564-2014.pdf
12. Simmons, G. J. The Prisoners' Problem and the Subliminal Channel [Text] / G. J. Simmons // Advances in Cryptology. – 1985. – Vol. 209. – P. 364–378

denniksam@gmail.com

nik.nik.sulima@gmail.com

Прийнято до редакції:20.11.2025

Прийнято до друку: 29.11.2025

Опубліковано: 29.12.2025