КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 3 (23), 2024
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2024.23.284309
УДК 004.4.62.056

Фесенко Андрій Олексійович
к.т.н., доцент, доцент кафедри кібербезпеки та захисту інформації
Київський національний університет імені Тараса Шевченка, Київ, Україна
ORCID 0000-0001-5154-5324
aafesenko88@gmail.com

Гапон Ростислав Сергійович
студент спеціальності "Кібербезпека"
Київський національний університет імені Тараса Шевченка, Київ, Україна
ORCID 0009-0004-8021-7724
rostikgapon587@gmail.com

ПОРІВНЯННЯ АЛГОРИТМІВ СТИСНЕННЯ ТА ГЕШ-ФУНКЦІЙ ГОТОВИХ ПРОГРАМНИХ РІШЕНЬ У РОЗРІЗІ СТВОРЕННЯ ВЛАСНОГО ЗАСТОСУНКУ

Анотація. Із стрімким розвитком інформаційних технологій робота над електронними даними стала простою в реалізації та часто використовуваною в роботі. Як наслідок більшість організацій з часом перейшли повністю на електронні системи, де проводиться зберігання даних. Однак кількість інформації з кожним роком збільшується в геометричній прогресії, що зумовлює використання більших сховищ та ресурсів для їхнього опрацювання. Крім того, споживання інформації в системах зумовлюються створенням все більших ризиків компрометації даних. Одним з шляхів вирішення описаних проблематик – є використання програмного рішення для стиснення даних із забезпеченням контрою цілісності криптостійкими геш-функціями. У статті проводиться аналіз алгоритмів стиснення без втрат даних та геш-функцій, які використовуються в готових програмних рішеннях, додатково геш-функції порівнюються з Державним стандартом України 7564:2014 "Купина" та відповідний аналіз готових рішень на функціональність. За результатами дослідження для практичної реалізації було обрано алгоритм стиснення Deflate. Геш-функція "Купина" в порівнянні з іншими варіантами проявляє себе на рівні або перевершує деякі показники, що показує якісну роботу алгоритму. Більшість готових програмних рішень не мають крипостійкої функції для перевірки цілісності інформації. Метою цієї статті на базі дослідження всіх складових побудувати власний програмний застосунок. У статті доведено, що Deflate проявив себе краще за перелічені алгоритми в характеристиках алгоритмів стисненння: об'єм пам'яті, швидкодія, коефіцієнт стиснення, кількість проходів, чи з'являється надмірність; визначальною характеристикою при порівнянні геш-функцій було значення крипостійкості щодо однієї з атак (за колізіями), де "Купина" має кращі характеристики і має високе значення стійкості; готові програми стиснення інформації були проаналізовані на алгоритми, які вони використовують, геш-функції та переваги з недоліками. Наведено порівняльні таблиці з кожною складовою для застосунку та для створених програмних рішень. Також продемонстровано частини кодової реалізації, результати власного застосунку стиснення інформації з контролем цілісності. Проведено порівняння ефективності власного засобу з проаналізованим програмним забезпечення.

Ключові слова: стиснення; гешування; Deflate; геш-функція "Купина"; програмне рішення; контроль цілісності; криптостійкість.

ВСТУП

Практичний перехід організацій на використання лише електронних систем зумовлює використання більшої кількості ресурсів для їхнього збереження, опрацювання та передачі. Кількість використаної інформації збільшується з кожним

роком у надзвичайній кількості, і як один з факторів пришвидшення цього процесу стала пандемія COVID-19 [10], [11], [23]. Використання програмного забезпечення стиснення інформації допомагає вирішити проблему використання більшої кількості сховищ, зменшуючи розмір в середньому до 50-60% [25]. Крім цього при зменшені розміру інформації, скорочується час для передачі по каналам зв'язку або для завантаження у систему. Використання програмного забезпечення стиснення даних також допомагає вирішити економічні наслідки: при зменшені відповідного місця в сховищах, організації заощаджують на використанні додаткового апаратного забезпечення зберігання даних. Це також допомагає знизити витрати на мережеву інфраструктуру за передачі даних, оскільки зменшується споживання пропускної здатності.

Постановка проблеми. Зі збільшенням використання електронних ресурсів підвищується ризик компрометації даних [12]. Наразі більшість програмних застосунків стиснення вміщують в собі контроль цілісності лише у вигляді некриптостійких геш-функцій, які можна з легкістю підробити, використовуючи атаки на виявлення колізій чи за попереднім зображенням. Тим паче жоден із засобів немає в собі імплементовану геш-функцію "Купина", хоча вона має достатні показники криптостійкості, аби бути на рівні, на приклад, з сімействами геш-функцій SHA-2 [14], SHA-3 [15], Blake2 [2].

Аналіз останніх досліджень і публікацій. За останніми дослідженнями [12] в цьому році було скомпрометовано надзвичайно багато інформаційних даних. Задля цього було додатково проаналізовано характеристики геш-функцій, які використовуються в сучасних програмних рішеннях стиснення інформації, в порівнянні з геш-функцією "Купина" [2], [14], [15], [20], [22], [28]. Крім цього, на всі засоби [3], [4], [13], [17], [18] було проведено аналіз на рахунок використаних геш-функцій у своєю використанні та вмісту алгоритмів стиснення [1], [5-9], [16], [19], [21], [24].

Мета статті є створення програмного застосунку стиснення та контролю цілісності, на базі проаналізованих перелічених компонентів (алгоритму стиснення та геш-функції), який вирішує в собі поставлені в цій статті проблему.

ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ

Для вдалої розробки власного програмного застосунку потрібно побудувати план функціонування системи, детально вказавши важливість кожної складової. Першочерговим кроком є аналіз готових програмних рішень на використання алгоритмів, завдяки яким можна виконати стиснення даних та геш-функцій.

Дослідження програм стиснення даних. Програми стиснення даних використовуються для відповідно стиснення з розпакуванням даних у вигляді файлів та тек. Інакше в результаті виконання створюються архіви, які містять в собі перелічені файли, що полегшує майбутнє використання в розміщенні сховища. Процес вимагає кодування задля створення меншого формату інформації, тим самим знижуючи розмір і забезпечуючи цілісність.

Однією з основних функцій, яка слугує для програм такого типу, - це видалити зайву надмірність з використанням алгоритмів стиснення, щоб вкінці представити закодовані дані. Крім цього, вони допомагають реалізувати і обернений процес розпаковки стиснутих даних і дозволяють обрати відповідний формат стиснення. Також програми стиснення даних мають додаткові функції, такі як забезпечення архіву паролем та перевірки цілісності на модифікації.

Для дослідження були обрані такі програмі засоби:

ISSN 2663 - 4023

WinZip – характерний програмний засіб, який орієнтований на операційну систему Windows. Надає можливості користувачам вибору з багатьох форматів стиснення, та має зрозумілий інтерфейс використання [17], рис. 1.

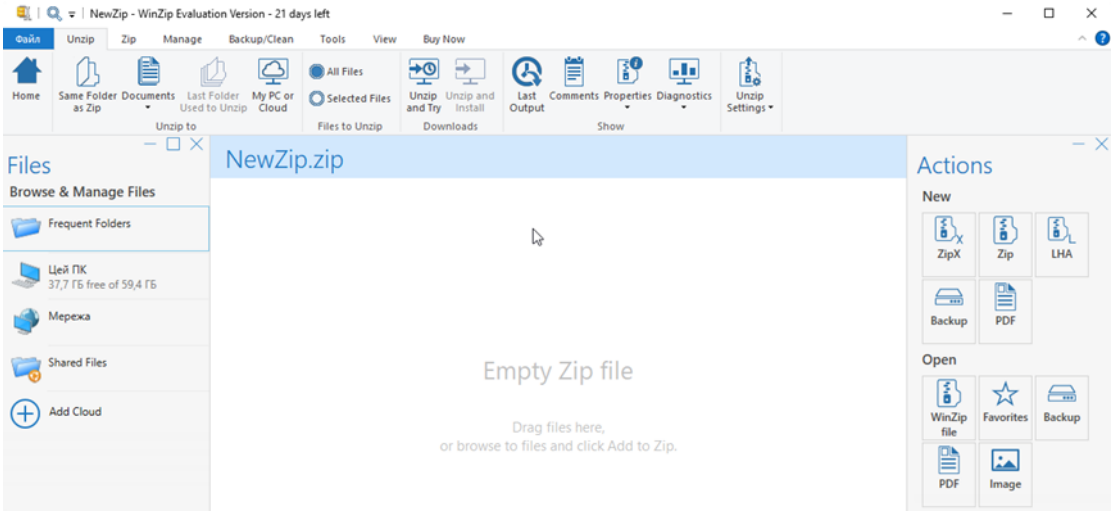

Рис. 1. Інтерфейс WinZip

PeaZip – програмний засіб з відкритим кодом, що надає можливість додатково використати перевірку саме криптостійкими геш-функціями, однак має менший перелік використаних форматів [18], рис. 2.

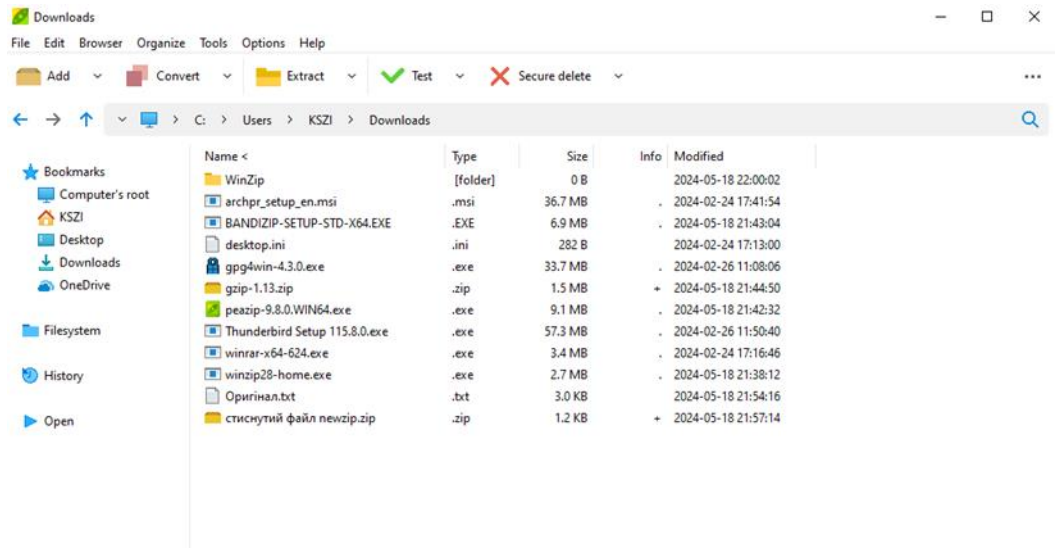

Рис. 2. Інтерфейс PeaZip

Bandizip – характерний застосунок, який також орієнтований на операційну систему Windows, додатково надаючи широкої вибірки форматів стиснення. З усіх варіантів єдиний має інтерфейс українською мовою [3], рис. 3.

ISSN 2663 - 4023

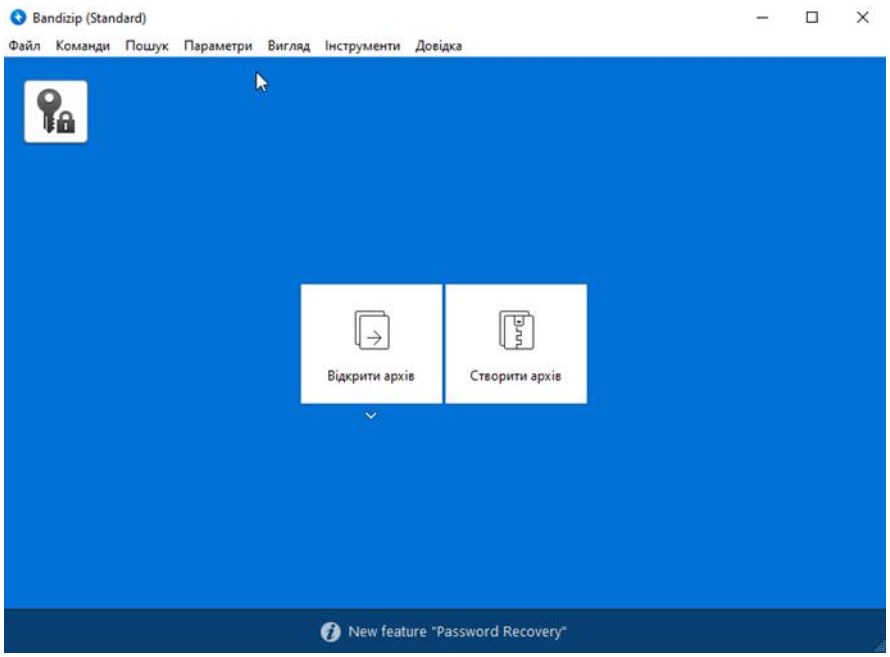

Рис. 3. Інтерфейс Bandizip

gzip – програмний засіб, орієнтований на використання в Unix-подібних систем. Немає графічного інтерфейсу та великого вибору форматів стиснення, однак має гарні показники стиснення з високою швидкодією [13], рис. 4.

```
hapon@hapon-virtual-machine:~$ gzip original.txt
```

Рис. 4. Консольний варіант gzip

BreeZip – програмний засіб, який пропонує Microsoft Store, з достатньою вибіркою форматів стиснення та можливістю перевірки цілісності архівів [4], рис. 5.

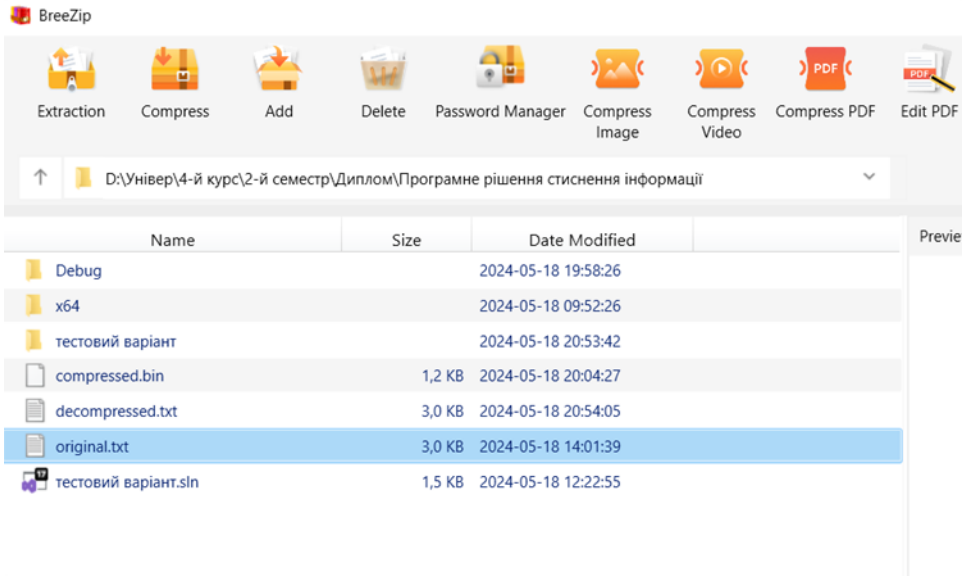

Рис. 5. Інтерфейс BreeZip

Результати дослідження продемонстровані в таблиці 1.

Таблиця 1

Порівняння програмних засобів стиснення інформації

Назва програмного засобу	Переваги	Недоліки	Використані алгоритми стиснення	Використані функції гешування
WinZip	Має зручний інтерфейс, підтримує багато різних форматів стиснення; може працювати досить швидко; виконує відновлення пошкоджених архівів, захист паролем, розпакування файлів під час завантаження.	Повна функціональність WinZip вимагає платної ліцензії; існують безліч альтернативних програм для стиснення файлів, які пропонують схожий функціонал	Deflate, LZMA, BZIP2, ZIPX, та інші	CRC32, для ZIP-файлів
PeaZip	Є безкоштовною програмою; підтримує широкий спектр форматів стиснення; має інтуїтивно зрозумілий інтерфейс; пропонує деякі додаткові корисні функції, такі як шифрування файлів, розділення архівів на частини, створення саморозпаковуючих архівів.	Може працювати трохи повільніше, особливо при стисненні великих обсягів даних; може не підтримувати деякі менш поширені формати архівів, які підтримують інші програми.	Deflate, BZip2, PPMd, LZMA/LZMA2, та інші	Adler32, CRC16, CRC24, CRC32, CRC64, Ripemd160, SHA-1, SHA-2 (SHA256, SHA512), SHA-3 (SHA-3 256, SHA-3 512), BLAKE2S і BLAKE2B, Whirlpool
Bandizip	Відомий своєю швидкістю роботи як при стисканні, так і при розпакуванні файлів; має дружній та інтуїтивно зрозумілий інтерфейс; підтримує багато різних форматів архівів; має додаткові функції, такі як розділення архівів на частини, захист паролем, перевірка цілісності файлів	Деякі розширені функції доступні лише в платній версії; у безкоштовній версії Bandizip можуть бути обмеження щодо підтримки деяких форматів архівів або функцій	Deflate, BZip2, LZMA/LZMA2, ZipX, та інші	CRC32
gzip	Зазвичай надає високий рівень стиснення; процес стиснення і розпакування займає мало часу; є частиною більшості Unix-подібних систем, що означає	Підтримує тільки один формат стиснення, що може бути недостатнім для деяких завдань; хоча gzip може бути використаний з командного	Deflate	CRC32

	можливість його використання безпосередньо з командного рядка	рядка, він не має графічного інтерфейсу, що може зробити його використання менш зручним; у деяких випадках gzip може не надавати найкращого рівня стиснення для певних типів файлів, зокрема для вже стиснутих або невеликих файлів.		
BreeZip	BreeZip може обробляти архіви в багатьох форматах; має простий та зручний інтерфейс; BreeZip доступна безкоштовно в Microsoft Store; програма працює швидко, забезпечуючи високий рівень продуктивності при розпаковуванні файлів;	У безкоштовній версії присутні рекламні оголошення; порівняно з деякими іншими архіваторами, BreeZip може мати обмежений набір функцій; іноді можуть виникати проблеми з розпаковуванням архівів.	Deflate, LZMA, PPMD та інші	CRC32

Дослідження алгоритмів стиснення. Стиснення даних використовує певний алгоритм щодо визначення зайвої надмірності та після виконання компактного представлення даних. Крім того, варто слідувати правилу неспотвореності джерела даних, щоб реалізувати можливе декодування в попередній вигляд інформації [27], [29].

Зазвичай схема, яка використовується при стисненні та розгортанні даних, складається з кодера (компресора) та декодера (декомпресора). Проміжними результатами в схемі виконання слугують стислі дані та відновлені дані, рис. 6.

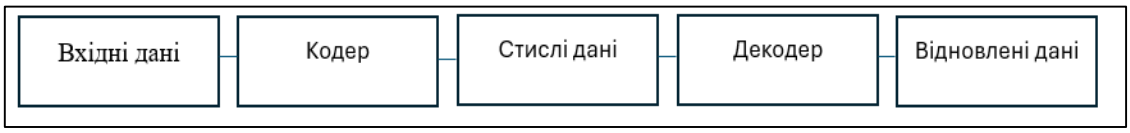

Рис. 6. Схема процесів стиснення і розгортання

Однак, якщо користуватися цією схемою, то результат після складової декомпресора має два варіанти повернення вихідних даних: або без втрат інформації, або частково мати деякі спотворення. Від цього посилається класифікація за критерієм стиснення - на неруйнуюче стиснення та руйнуюче. Перший вид використовується до текстової інформації, оскільки будь-яка зміна спотворює вихідні дані, що робить їх не

придатними для використання, в той же час друга – орієнтується на чуття людини, тобто відеозаписи, звукова інформація чи графічна.

Відповідно до налаштування методи стиснення можна розділити на:

• Адитивного стиснення;

• Напівадитивного стиснення;

• Локально-адитивного стиснення;

Адитивне стиснення виконується з метою зміни свої параметрів, операцій і налаштувань в залежності від даних, над якими виконується стиснення. Відповідно іншими словами, цей метод тестує вхідні, неопрацьовані дані, а потім підлаштовує власні параметри з операціями до результату перевірки.

Напівадитивне стиснення використовує в своєму алгоритмі двопрохідний метод компресії: у першому проході збирається статистика стискання даних, а другий – саме стиснення з використанням параметрів.

Методи локально-адитивного стиснення відповідає за здатність обрати власні параметри відповідно до особливостей файлу і далі змінювати їх, від області до області вхідної інформації.

Розглядаючи використання кодера і декодера, розрізняються такі методи, як:

• Симетричного стиснення – кодер і декодер використовує один і той самий алгоритм;

• Асиметричного стиснення – один з перелічених елементів, або кодер, або декодер має виконати більшу частину роботи.

Крім того, увагу варто звертати при виборі методів стиснення саме на використання в певній системі й користуватися такими характеристиками, як:

- Коефіцієнт стиснення;
- Фактор стиснення (ступінь стиснення);
- Швидкість стиснення/розгортання даних;
- Об’єм пам’яті, який необхідний для зберігання;
- Оцінка складності практичної реалізації;
- Тип даних, що може мати максимальне значення при стисненні;
- Можливість виконання в потоці;
- Ступінь спотворення.

Алгоритми, які досліджувалися:

Алгоритм статичного кодування Гаффмана - при розгляді статичного кодування Гаффмана, можна вказати те, що це успішний метод стиснення, який є першочерговим під час використання стиснення тексту. Оскільки в будь-якому тексті деякі символи зустрічаються набагато частіше, ніж інші, ідея цього виду кодування полягає в тому, щоб замість використання коду фіксованої довжини, представити символ, що часто зустрічається в джерелі, коротшим кодовим словом; а символ, що зустрічається рідше, - відповідно довшим кодовим словом. Це призведе до того, що загальна кількість бітів такого представлення значно зменшується для джерела символів з різною частотою. Кількість необхідних бітів зменшується в середньому для кожного символу [19], [21].

При розгляді стиснення інформації кодуванням Гаффмана спочатку будують бінарне дерево. Існує два підходи до побудови бінарного дерева: один починається з листка (вузла), щоб побудувати дерево знизу вгору до кореня дерева. Цей підхід «знизу-вгору» використовується у кодуванні Гаффмана. Інший підхід полягає у побудові дерева від кореня до листя. Підхід «зверху-вниз» використовується у кодуванні Шеннона-Фано.

Алгоритм динамічного кодування Гаффмана - для покращення ступеня стиснення використовується динамічне(адаптивне) кодування Гаффмана, застосовуючи до моделі статистику, засновану на вихідному контенті статичного кодування. Частотна таблиця з алфавітом символів динамічно коригується після зчитування кожного елементу в процесі стиснення або розпакування [1], [19], [21].

Порівняно зі статичним кодуванням Гаффмана, динамічна модель набагато ближча до реальної ситуації джерела після початкових кроків.

У динамічному кодуванні Гаффмана алфавіт і частоти його символів збираються і підтримуються динамічно відповідно до вихідного файлу на кожній ітерації. Дерево Гаффмана також оновлюється на основі алфавіту і частот динамічно. Коли кодер і декодер знаходяться в різних місцях, обидва підтримують ідентичне дерево Гаффмана для кожного кроку незалежно. Тому немає необхідності передавати дерево Гаффмана

Алгоритм стиснення LZ77 - У підході алгоритму стиснення LZ77 словник є частиною попередньо закодованої послідовності. Кодування в цьому випадку розглядається, як вхідна послідовність через рухоме (“ковзне”) вікно, яке в свою чергу складається з двох частин: буфера пошуку, що містить частину закодованої послідовності, і буфера перегляду, що містить наступну частину послідовності, що підлягає кодуванню [21].

Для кодування символьної послідовності в буфері перегляду кодер переміщує покажчик пошуку через буфер пошуку, поки не буде знайдено збіг з першим елементом. Відстань вказівника від буфера перегляду називається також зміщенням.

Кодер досліджує елементи, що йдуть від місця розташування вказівника, аби перевірити, чи є збіг з послідовними символами у буфері перегляду. У цьому випадку кількість послідовних елементів у пошуковому буфері, які співпадають з послідовними елементами в буфері перегляду, починаючи з першого символу, називається довжиною збігу. При наявному знаходженні найдовшої довжини збігу відбувається кодування трійкою <a,b,c>, де a - зсув, b - довжина збігу, а c – кодове слово, що відповідає символу в буфері перегляду, який слідує за збігом. Коли елемент не знаходить попередній, тобто він перший, то в трійці зсув та довжина збігу дорівнюють нуль, в той же час кодове слово і є цим елементом.

Алгоритм стиснення LZSS - є модифікацією алгоритму LZ77. Порівнюючи з попереднім алгоритмом, то було помічено, що окремі символи, що не збігаються, або рядки з одного чи двох символів, які навпаки збігаються, займають більше місця при кодуванні, ніж якщо їх залишити не закодованими [9].

Тому було запропоновано ставити перед кожним елементом повідомлення прапорець, який відповідає за його кодування чи не кодування. Також, якщо кодування відбувається не за довжиною від нуля до певної точки, то ця певна точка може бути використана, як заміщення. Також у порівнянні з попереднім алгоритмом замість трійки, використовується двійка <a, b>, де a – зсув у словнику, b – довжина підрядка.

Алгоритм стиснення LZ78 - Використання алгоритму стиснення LZ77 припускає, що будь-який патерн, який повторюється протягом періоду, що більший за вікно кодера, не буде захоплений. Як наслідок, у гіршому випадку, якщо послідовність з елементів, яку потрібно закодувати, є періодичною з певним періодом, більшим за буфер пошуку, то в цьому вигляді жоден з нових символів не матиме збігів у пошуковому буфері і повинен бути представлений окремими кодовими словами. Оскільки це пов'язано з передачею надлишкових даних, кінцевим результатом буде розширення, а не стиснення [5], [21].

Алгоритм стиснення LZ78 вирішує вищеописану проблему, відмовляючись від буфера пошуку, аби зберігати словник. Цей словник має бути однаково створений як на кодері, так і на декодері. Вхідні дані кодуються у вигляді подвійного числа <a, b>, де a - індекс, що відповідає словниковому підрядку, який найдовше збігається з вхідними даними, b - код символу у вхідних даних, що слідує за частиною вхідних даних, яка збігається з ними. Порівнюючи з LZ77, значення індексу нуля використовується у випадку відсутності збігів. Цей подвійний символ стає новим підрядком у словнику, що спричиняє створення кожного запис, який буде створюватися у словнику, - це один новий символ, об'єднаний з уже створеним раніше підрядком.

Алгоритм стиснення LZW - є модифікацією LZ78. В порівнянні з попереднім алгоритмом, ці коди мають постійну довжину і для цього необхідна кількість бітів відносно довжини словника. Словник початково проводить ініціалізацію за усіма рядками, створюючи довжину 1 відповідними елементами алфавіту. Для опису кроків нехай текстовий сегмент a, b - наступний символ (наступний за входженням рядка a). Якщо ab немає в словнику, індекс a записується у вихідний файл і додається ab до словника. Потім відбувається заміна a на b і обробляється наступний символ, який йде після b. Якщо ab є в словнику, йде обробка наступного символу, з відрізком ab замість a. Спочатку відрізок a встановлюється на перший символ вихідного тексту, аби вказати, що «a належить словнику» і це вказує на те, що цей елемент є інваріантом щодо першого кроку операції [6], [7].

Алгоритм стиснення BWT – характерний тим, що він заснований на перетвореннях з метою стиснення даних, особливо текстових. Якщо детальніше розглянути зміст алгоритму, то його компресор складається з додаткових алгоритмів, і зазвичай це поєднання MTF та RLE [8], [16].

Варто вказати, що при використані цього перетворення відбувається саме форматування даних, відповідно до подальшої компресії. І однією його особливістю є те, що алгоритм опрацьовує відразу повний блок даних. Тому при використанні стиснення даних, яке використовується в словниках, тобто елемент за елементом, то можлива складність при реалізації. Це спричиняє те, що в деяких моментах він вимогливіший навіть за алгоритми стиснення сімейства LZ.

Однак варто вказати, що реалізації обробки даних посимвольно також можливо, на противагу опрацювання блоків. Хоча це спричинить зменшить продуктивність використання з точки зору швидкості, бо виконується більша частина роботи.

Алгоритм стиснення RLE - це алгоритм стиснення даних, який підтримується більшістю форматів растрових файлів, однак його можна використовувати і для текстових вхідних даних. Дивлячись на тип даних, може виникнути вплив на ступінь стиснення цього алгоритму.

Використання цього типу алгоритму до даних спричиняє отримання не високих ступенів стиснення, але особливістю цього алгоритму є те, що він простий в практичній реалізації та швидко виконує стиснення [24].

Алгоритм стиснення Deflate - складається з комбінації двох алгоритмів, а саме: LZ77 та кодування Гаффмана. Перший допомагає описати потік даних таким чином, щоб створити послідовність пар вказівників, відстаней і букв. Як описано вище, вихідні дані LZ77 повністю складаються з потрійних відстаней. Літерал вказує на те, що знайдено відповідний символ, який знаходиться у попередній послідовності. Однак, якщо символ не знайдено, вказівник і відстань дорівнюють нулю. Перш за все, цей алгоритм не визначає, як кодуються відповідні пари трійок [27].

Хоча одним з найбільш очевидних способів є кодування кожного з елементів трійки фіксованою кількістю бітів залежно від кількості значень, яких може набувати елемент. З іншого боку, використання методу ентропійного кодування призводить до покращення коефіцієнта стиснення.

Алгоритм кодування Гаффмана використовується в Deflate для кодування даних, що виводяться з LZ77. Таким чином, пари трійок розбиваються на різні алфавіти: літерали і довжини в одному, відстані в іншому. Тому як прямі коди довжин, так і коди відстаней кодуються зі змінною кількістю бітів до відносної частоти.

У цьому випадку кодування складається з генерації коду відстані та коду довжини, і якщо не знайдено жодного збігу, створюється єдиний код для літералу. Під час декодування декодер визначає тип коду; і якщо він виявляє літерал, він декодує послідовність, інакше він зчитує наступний код, щоб отримати відстань для виведення відповідної послідовності. Незважаючи на те, що коди в цьому випадку мають змінну кількість бітів, декодер знає, де закінчується код, оскільки кодування Гаффмана проектується на префіксні коди.

Дослідивши частину алгоритмів стиснення даних без втрат, необхідно обрати з перелічених методів найкращий для наступної її програмної реалізації. Порівняння відбувалося за частиною характеристик алгоритмів стиснення, де значення від 1-10 зображені в таких значеннях: низький рівень (оцінка 1-3), середній рівень (оцінка 4-7) та високий рівень (8-10), для пунктів коефіцієнт стиснення, швидкість виконання алгоритму, використання об’єму пам’яті, збільшення надлишковості. Значення подані в таблиці 2.

Таблиця 2

Порівняння алгоритмів стиснення щодо характеристик

Алгоритм стиснення	**Коефіцієнт стиснення**	**Швидкість виконання алгоритму**	**Використання об’єму пам’яті**	**Чисельність проходів у алгоритмі**	**Збільшення надлишковості**	**Загальне використання**
Статичний алгоритм кодування Гаффмана	5	8	9	2	2	Універсальні методи, стиснення тексту та бінарних даних
Динамічний алгоритм Кодування Гаффмана	5	7	6	1	2	
LZ77, LZSS	9	4	3	1	2	Універсальні методи, стиснення тексту, стиснення зображень
LZ78, LZW	5	6	5	1	2	
BWT	7	2	5	>1	2	Універсальні методи, стиснення тексту
RLE	4	9	4	1	2	Метод при використанні повторюв

						аних текстових елементів, або з малою кількістю кольорів
DEFLATE	7	7	2	2	2	Архівування файлів, зображень, протоколів, ресурсів у реальному часі та веб-ресурсів

Дослідження геш-функцій. Процес гешування перетворює вхідний масив даних будь-якого розміру в бітовий рядок (або зазвичай у вигляді хексу) розміру, який зазначено в алгоритмі. Такі перетворення називаються геш-функціями або функціями згортки, а результати - гешем, геш-кодом або дайджестом. Іншими словами, алгоритм гешування - це послідовність математичних перетворень, яка створює унікальну двійкову послідовність фіксованої довжини з двійкової послідовності змінної довжини. Функція, яка реалізує цей алгоритм, називається геш-функцією [26].

Гешування використовується для порівняння даних: якщо геш-коди двох масивів відрізняються, то масиви гарантовано різні; якщо вони збігаються, то масиви, швидше за все, однакові. Прямої відповідності між вихідними даними та геш-кодом немає, оскільки кількість значень геш-функції менша за кількість можливих вхідних даних; існує багато масивів, які можуть мати однаковий геш-код - це так звані колізії. Ймовірність колізій є важливим критерієм якості геш-функції.

Існує багато алгоритмів гешування з різними характеристиками, такими як розрядність, обчислювальна складність і криптографічна стійкість. Вибір конкретної геш-функції залежить від специфіки завдання. Найпростішим прикладом геш-функції є контрольна сума.

Для того, щоб перетворення даних вважалося геш-функцією, воно повинно одночасно відповідати наступним умовам:

- обробляти послідовності різної довжини;
- виробляти послідовність бітів фіксованої довжини;
- бути достатньо простою, щоб її можна було обчислити з будь-якою вхідною інформацією
- мати властивість незворотності, тобто неможливість відновлення вхідного значення за результатом;
- бути однозначними, тобто різні вхідні послідовності не повинні давати однаковий геш-код.

Одне з важливих застосувань геш-функцій - перевірка даних. Це дозволяє перевіряти інформацію на відповідність оригіналу без використання самого оригіналу, використовуючи тільки геш-значення. До основних сфер такого застосування відносяться

• перевірка даних на наявність помилок;

• прискорення пошуку даних.

Оскільки пошкодження даних може бути спричинене простими технічними проблемами, а саме: помилками передачі, деградацією підтримки пам'яті, які зазвичай можна легко виявити за допомогою швидких та обчислювально легких функцій – контрольних сум, або може бути наслідком підробки, навмисної модифікації даних, створеної зловмисником для використання слабкості простіших механізмів виявлення помилок - що вимагає обчислення більш потужних функцій перевірки, таких як криптографічно сильні геш-алгоритми.

Щодо атак, яким можуть протистояти геш-функції, то в переважній більшості вони націлені на знаходження даних, якими можна замінити оригінал:

• Атаки на виявлення колізій – використовується зловмисником для знаходження двох різних входів, щоб вивести одне і те значення геш-функції. Ця можливість допомагає замінити відповідні дані на зловмисні.

• Атака за попереднім зображенням – використовується для знаходження вхідних даних, які дають дайджест без вихідних даних. Ця можливість допомагає повернути процес гешування і відновити вихідні дані.

Всі порівнянні відбувалися відносно геш-функції “Купина” на базі криптографічних геш-функцій, які використовувалися в програмних засобах стиснення інформації.

Було проведено дослідження таких геш-функцій:

Геш-функція “Купина” структурно схожа на SHA-3, але відрізняється перетвореннях круглого типу. Оскільки деякі константи раунду додаються за допомогою модулярного додавання 2^{64}, а не тільки за додаванням за модулем 2, це допомагає запобігти застосуванню певних атак, які пов'язані щодо функції стиснення подібних до AES-конструкцій. Ця функція в собі використовує схему Меркля-Дамгорда у вигляді конструкції Девіса-Мейєра та на основі конструкції Івена-Мансуна [20], [28].

Щодо криптостійкості, то практичної реалізації для повних кіл ітерації результатів немає, однак є дані наближені на рахунок утворення колізій: якщо теоретичне значення відповідає відповідно режимів: 2^{128} і 2^{256}, то в порівнянні з практичним значенням: для п’яти раундів Купини-256 відповідає значення приблизно 2^{120}, для Купини-512 не було виявлено.

Геш-функції сімейства SHA-2 - налічує в собі шість функцій, які відрізняються виведенням дайджесту від 224 до 512 біт. Найпопулярнішими функціями, які використовуються в архіваторах, це SHA-256 та SHA-512. Ключовим принципом роботи SHA-2 є лавинний ефект та всі варіанти SHA-2 використовують конструкцію Меркля-Дамгорда [14].

Розглядаючи криптостійкість щодо колізій, то в теоретичному значенні відповідно це 2^{128} та 2^{256}, на практичній реалізації було виявлено у двох функціях на тридцять першій ітерації такі значення: $2^{49.8}$ та $2^{115.6}$.

Геш-функції сімейства SHA-3 - найпопулярнішими функціями, які використовуються в архіваторах, аналогічно до минулого пункту є SHA-256 та SHA-512.

Функції сімейства SHA-3 базуються на криптографічній конструкції Keccak. Це означає те, що в собі вони використовують структуру Губки, яка складається з двох функцій: поглинання (абсорбції) та видавання (видачі). Структура відповідної губки складається з трьох частин, а саме: розміру блоку, яка відповідає за поглинання вхідних даних та видачі дайджесту в залежності від функції; розміру ємності, яка

використовується для безпеки перетворення; та стан, який містить в собі, як розмір блоку, так і розмір ємності. В сумі один блок містить 1600 біт [15].

Розглядаючи криптостійкість стосовно колізій, то теоретичне значення буде відповідно минулому пункту, але натомість практичне значення є тільки для SHA-256 – 2^{33} лише для двох раундів.

Геш-функції сімейства BLAKE2 - це сімейство криптографічних геш-функцій, що базуються на алгоритмі BLAKE. Ці алгоритми характеризуються високою швидкодією обчислень, що роблять їх одними з використовуваних геш-функціях в різних застосуваннях, включаючи архіватори, цифровий підпис, парольні гешувальники. Також ці функції мають кілька конфігураційних параметрів, що дозволяють вибирати розмір вихідного гешу та розмір входу, [2], [20].

Сімейство містить в собі дві основні геш-функції: BLAKE2b та BLAKE2s. Перший більш оптимізований для довших вхідних даних, тоді як другий оптимізований для навпаки для коротших вхідних даних. Це дозволяє використовувати BLAKE2 в широкому спектрі сценаріїв.

Функції сімейства BLAKE2 використовуються для багатьох криптографічних завдань, включаючи аутентифікацію, цифровий підпис, гешування паролів, ідентифікацію даних та інші застосування.

Щодо криптостійкості, то теоретичні значення дорівнюють відповідно 2^{128} та 2^{256}, щодо практичних значень відповідно для 2,5 раундів 2^{112} та 2^{224}.

Геш-функція WHIRLPOOL - це геш-функція на основі блочного шифрування, призначена для реалізації безпеки і продуктивності, порівнянної ніж у геш-функцій, що не базуються на блочному шифруванні, на прикладі таких як SHA [20], [22].

Щодо криптостійкості, то теоретичне значення є 2^{256}, а практичне було виведено для 4,5 раундів із значенням 2^{120}.

Результати досліджень продемонстровані в таблиці 3.

Таблиця 3

Порівняння геш-функцій

Геш-функція		**Довжина вхідної інформації (біт)**	**Довжина блоку (біт)**	**Довжина гешу (біт)**	**Кількість ітерацій**	**Теоретичне значення криптостійкості**	**Практичне значення щодо колізій**
“Купина”	“Купина” (8-256)	$2^{96} - 1$	512	[8;512]	10	2^{128}	2^{120} (5/10)
	“Купина” (257-512)		1024		14	2^{256}	-
SHA-2	SHA-256	2^{64}-1	512	256	64	2^{128}	$2^{49.8}$ (31/64)
	SHA-512	2^{128}-1	1024	512	80	2^{256}	$2^{115.6}$ (31/80)
SHA-3	SHA-256	2^{256}-1	1600(1152)	256	24	2^{128}	2^{33} (2/24)
	SHA-512	2^{256}-1	1600(576)	512	24	2^{256}	-
Blake2	Blake2s	2^{64}-1	512	256	10	2^{128}	2^{112} (2,5/10)
	Blake2b	2^{128}-1	1024	512	12	2^{256}	2^{224} (2,5/14)
WHIRLPOOL		2^{256}-1	512	512	10	2^{256}	2^{120} (4,5/10)

ISSN 2663 - 4023

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

На базі проведених аналізів було виявлено, що більшість програм з стисненням інформації надають широку вибірку форматів стиснення інформації, однак натомість перевага перевірки цілісності відбувається не криптостійкими геш-функціями. Для розробки власного програмного засобу було прийнято за основу алгоритм стиснення Deflate, оскільки він мав найкращі показники відносно решти алгоритмів. Геш-функція "Купина" проявила себе на рівні з іншими криптостійкими функціями і навіть деяких перевершує у своїх характеристиках. Реалізація відбувалася на мові програмування С++.

А. Побудова програмного коду для об'єднання двох складових в один застосунок

Реалізація алгоритму стиснення Deflate. Для розробки було використано бібліотеку zlib, оскільки, окрім теоретичних алгоритмів, вона має додаткові методи стиснення із визначенням коефіцієнта стиснення.

Для стиснення і розгортання було використано інші методи, оскільки перші працюють із вхідними даними, а на противагу інші з вихідними.

Для побудови програмного рішення було використано методи для стиснення: deflateInit(z_streamp strm, int level), deflate(z_streamp strm, int flush), deflateEnd(z_streamp strm).

Метод deflateInit() характеризується ініціалізацією стану внутрішнього потоку для стиснення. Рівень стиснення має бути Z_DEFAULT_COMPRESSION, або між 0 та 9, де 1 - найкраща швидкість, 9 - найкраще стиснення, 0 - жодного стиснення (вхідні дані просто копіюються блок за блоком). Z_DEFAULT_COMPRESSION запитує компроміс між швидкістю та стисненням за замовчуванням (наразі еквівалентно рівню 6).

Метод deflateInit() повертає Z_OK у разі успіху, Z_MEM_ERROR, якщо не вистачило пам'яті, Z_STREAM_ERROR, якщо рівень не є допустимим рівнем стиснення, або Z_VERSION_ERROR, якщо версія бібліотеки zlib (zlib_version) несумісна з версією, яку припускає користувач (ZLIB_VERSION).

Метод deflate() стискає якомога більше даних і зупиняється, коли вхідний буфер стає порожнім, або вихідний буфер заповнюється. Це може призвести до деякої затримки під час читання вхідних даних без виведення, окрім випадків, коли це примусовий вивід.

Метод deflate() виконує одну або обидві з наступних дій:

• Стискає більше вхідних даних, починаючи з початкових значень next_in, та оновлює next_in і avail_in відповідно. Якщо не всі вхідні дані може бути оброблено (наприклад, через брак місця у вихідному буфері), то next_in і avail_in буде оновлено і обробка відновиться з цього моменту для наступного виклику deflate().

• Генерує більший вивід, починаючи з next_out, і оновлює next_out і avail_out відповідно.

Метод deflate() повертає Z_OK, якщо було оброблено більше вхідних даних або створено більше вихідних даних, Z_STREAM_END, якщо всі вхідні дані було спожито всі вхідні дані і вироблено всі вихідні дані

При використанні методу deflateEnd() усі динамічно виділені структури даних для цього потоку звільняються. Ця функція відкидає будь-який необроблений ввід і не очищає будь-який відкладений виведення.

```
z_stream zs;
memset(&zs, 0, sizeof(zs));
if (deflateInit(&zs, compressionlevel) != Z_OK)
    throw(std::runtime_error("deflateInit failed while compressing."));
```

ISSN 2663 - 4023

```
zs.next_in = (Bytef*)str.data();
zs.avail_in = str.size();
int ret;
char outbuffer[32768];
std::string outstring;
do
{
    zs.next_out = reinterpret_cast<Bytef*>(outbuffer);
    zs.avail_out = sizeof(outbuffer);

    ret = deflate(&zs, Z_FINISH);

    if (outstring.size() < zs.total_out)
    {
        outstring.append(outbuffer, zs.total_out - outstring.size());
    }
} while (ret == Z_OK);

deflateEnd(&zs);

if (ret != Z_STREAM_END)
{
    std::ostringstream oss;
    oss << "Exception during zlib compression: (" << ret << ") " << zs.msg;
    throw(std::runtime_error(oss.str()));
}
return outstring;
```

Методи inflateInit() та inflateEnd() за своєю структурою відповідають до попередників, однак використовуються навпаки не вхідні дані, а вихідні.

У випадку з методом inflate(), то його застосування розпаковує якомога більше даних і зупиняється, коли вхідний буфер стає порожнім або вихідний буфер заповнюється. Але решта дій відповідають методу deflate().

```
z_stream zs;
memset(&zs, 0, sizeof(zs));
if (inflateInit(&zs) != Z_OK)
    throw(std::runtime_error("inflateInit failed while decompressing."));
zs.next_in = (Bytef*)str.data();
zs.avail_in = str.size();
int ret;
char outbuffer[32768];
string outstring;
do
{
    zs.next_out = reinterpret_cast<Bytef*>(outbuffer);
    zs.avail_out = sizeof(outbuffer);

    ret = inflate(&zs, 0);

    if (outstring.size() < zs.total_out)
    {
        outstring.append(outbuffer, zs.total_out - outstring.size());
    }

} while (ret == Z_OK);

inflateEnd(&zs);

if (ret != Z_STREAM_END)
{
    std::ostringstream oss;
    oss << "Exception during zlib decompression: (" << ret << ") "
```

ISSN 2663 - 4023

```
                << zs.msg;
            throw(std::runtime_error(oss.str()));
        }
        return outstring;
    }
```

Для розрізнення стиснутого та розгорнутого файлів було виконано запис у різні формати. Для стиснутого файлу запис відбувався у бінарний файл.

```
    ofstream outputFileStream("compressed.bin", ios::out | ios::binary);
    outputFileStream.write(reinterpret_cast<char*>(&size_compress_data),
sizeof(int));
    outputFileStream.write(compress_data.c_str(), size_compress_data);
    outputFileStream.close();
```

Для розгорнутого навпаки в оригінальний і у випадку створеної програми до текстового файлу

```
    ofstream decompress_data;
    decompress_data.open("Шлях до файлу");
    decompress_data << decompress;
    decompress_data.close();
```

Реалізація геш-функції “Купина”. Реалізація геш-функції відбувалася без використання бібліотек, власним написаним кодом. Для цього було створено такі методи, які пов’язані саме з перетворенням текстового формату в бітове значення та десяткового значення числа в бітове. Крім цього решта функцій на пряму пов’язані з реалізацією ітерацій в одному блоці, наприклад метод multiply_GF256() використовується для множення рядків константних значень незвідного поліному та стовпців матриць стану в бієктивних функціях T_xor(), яка відповідає за додавання констант за модулем 2, та T_264(), яка відповідає за додавання констант за модулем 2^{64}. Метод one_loop() поєднує в собі ці функції перетворення з роботою майбутніх ітераційних значень гешу та з блоками повідомлень. У той же час R_ln() повертає фінальне значення після виконання додаткової T_xor() відповідно до обраного режиму.

```
    int from10to2text(string h, int i, int q);
    int from10to2number(long long block, int q, int q1);
    int multiply_GF256(int a, int b);
    string T_xor(string hex, int hash);
    string T_264(string hex, int hash);
    string one_loop(string* message, string* iterations, int hash, int
array_number);
    string R_ln(string final_hex, int hash);
```

Загальна структура реалізації геш-функції “Купина” зображено на рисунку 7, [28].

ISSN 2663 - 4023

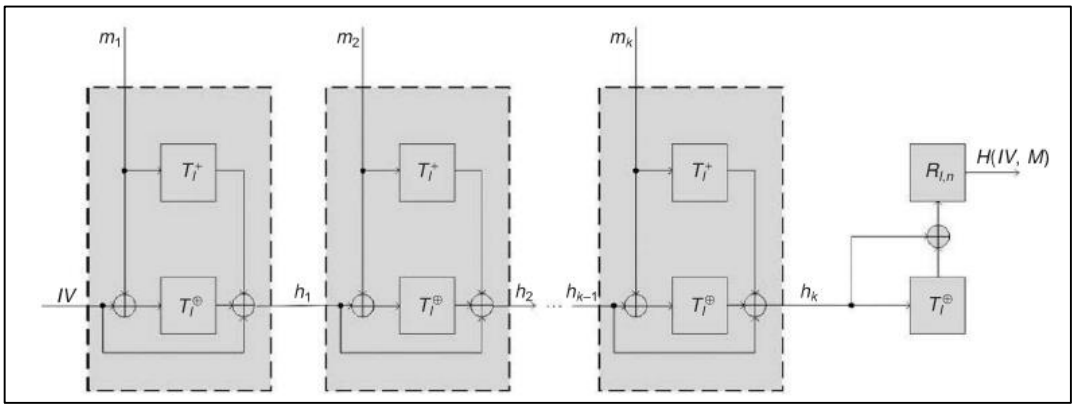

Рис. 7. Структурна схема алгоритму геш-функції "Купина"

Робота з вхідними даними відбувалася через двовимірні масиви, аби було легше працювати з матрицями стану. Однак початковими етапами були розбиттям даних на блоки фіксованої довжини відповідно до обраного режиму (наприклад від 8 до 256 біт) гешування та доповненням першим елементом 1 біт, далі нулями, поки не залишиться 96 біт на додавання довжини текстового повідомлення.

```
int number_blocks;
number_blocks = (message.size() * 8 / 416) + 1;
int* kupyna_blocks = new int[number_blocks * 512];
string bitString = "";
for (char c : message)
{
 bitset<8> bits(static_cast<unsigned char>(c));
 bitString += bits.to_string();
}
for (int x = 0; x < 1; x++)
{
 for (int y = 0; y < message.size() * 8; y++)
 {
        kupyna_blocks[y] = int(bitString[y]) % 2;
 }
}
kupyna_blocks[message.size() * 8] = 1;
for (int x = (message.size() * 8) + 1; x < number_blocks * 512 - 96; x++)
{
 kupyna_blocks[x] = 0;
}
int data_size[96] = {};
for (int x = number_blocks * 512 - 96, y = 0; x < number_blocks * 512 && y < 96;
x++, y++)
{
 data_size[y] = from10to2number(message.size() * 8, x - (number_blocks * 512 -
96), 96);
}
int data_size_bits[12][8] = {};
int z = 0;
for (int y = 0; y < 12; y++)
{
 for (int u = 0; u < 8; u++)
 {
        data_size_bits[y][u] = data_size[z];
        z++;
 }
}
int little_endline_data_size[96] = {};
```

ISSN 2663 - 4023

```
int z1 = 0;
for (int y = 11; y >= 0; y--)
{
  for (int u = 0; u < 8; u++)
  {
        little_endline_data_size[z1] = data_size_bits[y][u];
        z1++;
  }
}
for (int x = number_blocks * 512 - 96, y = 0; x < number_blocks * 512 && y < 96;
x++, y++)
{
  kupyna_blocks[x] = little_endline_data_size[y];
}
```

Для побудови бієктивних функцій було виконано такі кроки: додавання з константами відповідно до модулю 2 та 2^{64}, нелінійна перестановка (за S таблицями), зсув по рядкам та додавання елементів за полями Галуа. Нижче наведено приклад для блоку додавання констант 2^{64}.

```
int result_add_c[8][8];
for (int x = 0; x < 8; x++)
{
    unsigned long long vector = 0;
    unsigned long long column = 0;
    for (size_t y = 0; y < 8; ++y)
    {
        if (y == 7)
        {
            unsigned int limitedElement_c = (add_c[y][x] ^ p) & 0xFF;
            vector |= static_cast<unsigned long long>(limitedElement_c) << (y *
8);
        }
        else
        {
            unsigned int limitedElement_c = add_c[y][x] & 0xFF;
            vector |= static_cast<unsigned long long>(limitedElement_c) << (y *
8);
        }
        unsigned int limitedElement_mass = mass[y][x] & 0xFF;
        column |= static_cast<unsigned long long>(limitedElement_mass) << (y *
8);
    }
    unsigned long long result_add = (vector + column) & 0xFFFFFFFFFFFFFFFF;
    for (size_t y = 0; y < 8; ++y)
    {
        result_add_c[y][x] = static_cast<unsigned char>((result_add >> (y * 8))
& 0xFF);
    }
}
```

Після реалізації геш-функції було проведено порівняння з прикладами, які наведені в державному стандартами. Нижче приведено приклад відносно режиму 256.

На рис. 8 продемонстровано результати відносно власного програмного забезпечення.

ISSN 2663 - 4023

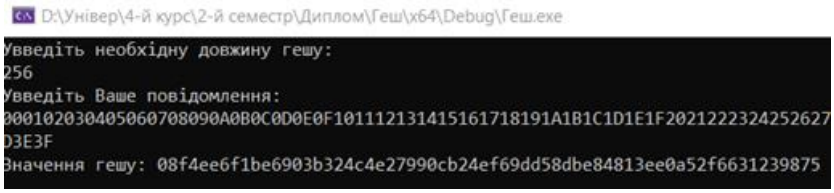

Рис. 8. Результат програмного коду геш-функції

На рис. 9 показано відповідні значення, які мають бути виведені в Державному стандарті України [28].

```
ТЕСТ ГЕШУВАННЯ (N = 512)
INPUT:
    000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
    202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
padded:
    000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
    202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
    80000000000000000000000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000200000000000000000
block[ 0].f:
    332851203C1F0C904DEED6410B7531CB1BAFE41FF27C23BD7448E40F7DF16FBB
    A6FEB5CFA65C0CBFC723F896F23C3408EE529641899E84A5CB1ADF3A82884883
block[ 1].f:
    56C926AFF9CDE45340DFB15C75941BEB6A6500DE35319B4A4905B56585F6B1F2
    933E3737FC1A0BDC9F94B3254066917E00FC289D027EF13871320FA9A545304C
final:
    86A9D24E23F4B103B72B8C69D1F1BBB5117EC3017604DCDF6BF04F3DA95C0268
    08F4EE6F1BE6903B324C4E27990CB24EF69DD58DBE84813EE0A52F6631239875
HASH-256:
    08F4EE6F1BE6903B324C4E27990CB24EF69DD58DBE84813EE0A52F6631239875
```

Рис. 9. Результат в Державному стандарті України

Реалізація віконного додатку. Оскільки попередні складові були протестовані та проаналізовані, то для об’єднання їх в одне програмне забезпечення варто побудувати схему, як відповідно буде працювати цей засіб, рис. 10.

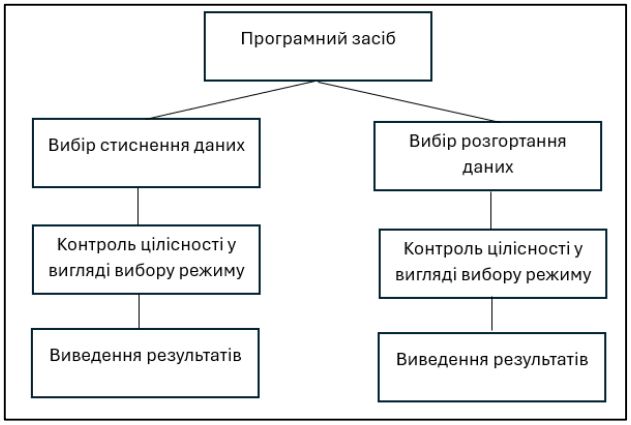

Рис. 10. Логічна структура роботи власного програмного засобу

Загалом програмний застосунок складається з двох кнопок, де можна обрати стиснення чи розгортання даних, текстових полів: виведення шляху до файлу після його вибору через кнопки, вибору режиму геш-функції та виведення результатів. Всі складові були об’єднанні по групах, що можна буде потім прослідкувати.

ISSN 2663 - 4023

Для того, аби полегшити користувачу пошук відповідних файлів до стиснення і розгортання, було використано файлові діалоги, де додатково було вказано фільтрацію. Для стиснутих даних це виведення текстових файлів.

```
this->openFileDialog1->FileName = L"openFileDialog1";
this->openFileDialog1->Filter = L"Text Files (*.txt)|*.txt|All Files (*.*)|*.*";
```

Для розгорнутих – бінарних.

```
this->openFileDialog2->FileName = L"openFileDialog2";
this->openFileDialog2->Filter = L"Text Files (*.bin)|*.bin|All Files (*.*)|*.*";
```

Загалом при об’єднанні складових додаток має зрозумілий інтерфейс, оскільки він надає можливість відкриття файлового діалогу і вибору, саме файлів того розширення, яке необхідне для роботи (стиснення/розгортання). Також користувач може обрати режим для геш-функції, аби перевірити чи проводилася модифікація над файлами та вкінці отримати результат про успішне стиснення/розгортання інформації з виведеним дайджестом.

Б. Демонстрація власного застосунку та порівняння характеристик

Об’єднавши попередні компоненти у вигляді алгоритму стиснення інформації Deflate та геш-функції “Купина”, було реалізовано програмне рішення у вигляді віконного додатку з вибором дій для користувача (обирання шляху до файлу та вибір режиму перевірки цілісності даних), рис. 11.

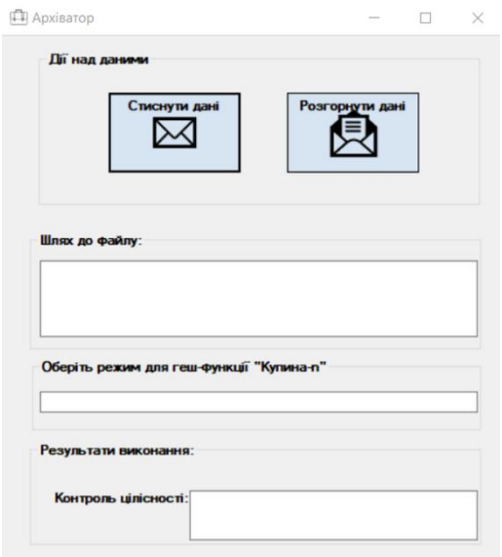

Рис. 11. Інтерфейс власного програмного застосунку

На прикладі стиснення буде показано функціональність додатку. Аби стиснути файл, необхідно ввести режим для геш-функції, потім обрати кнопку “Стиснути дані”, що спричинить відкриття файлового діалогу, рис. 12.

ISSN 2663 - 4023

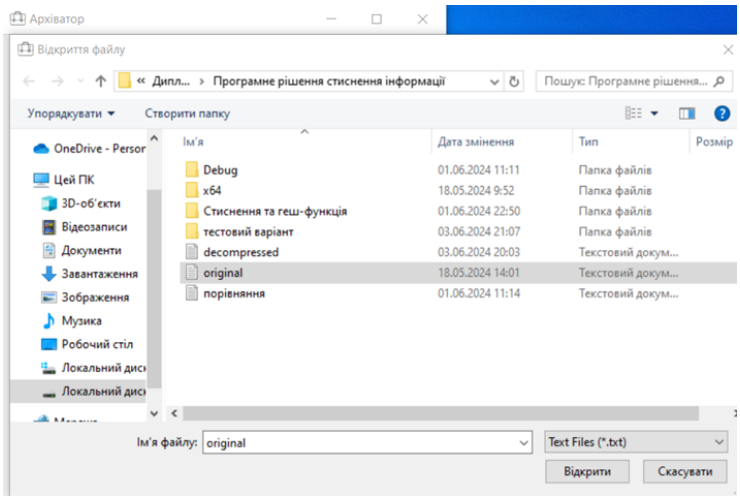

Рис. 12. Вибір необхідного файлу для стиснення

У віконному додатку після вибору файлу, над яким необхідно виконати дії стиснення, з'являється у текстовому полі шлях до цього файлу, та в результатах виводиться напис про успішне стиснення даних та контроль у вигляді дайджесту розміром, який вибере користувач, рис. 13.

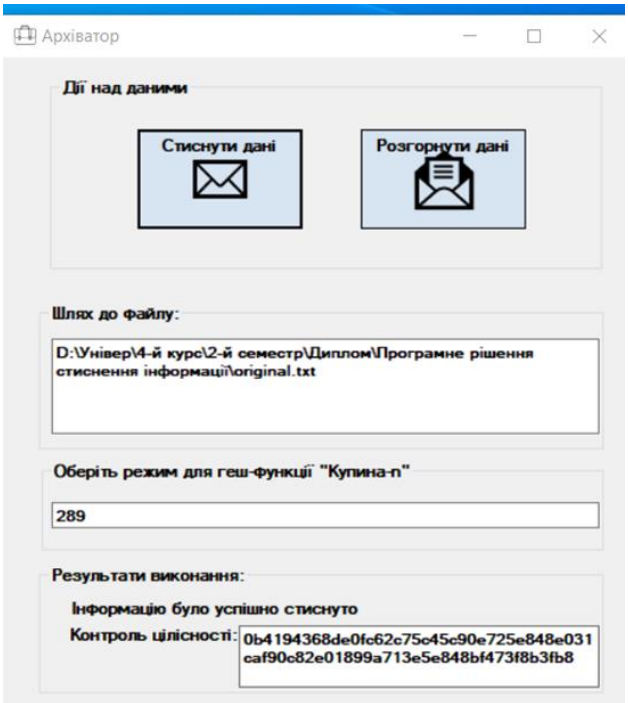

Рис. 13. Результати стиснення та виведення дайджесту

Відносно результатів стиснення було додатково проведено дослідження на відповідність до характеристик, які характеризують алгоритми стиснення, про практичного значення [19].

Коефіцієнт стиснення - це співвідношення розміру вихідного файлу до розміру вхідного файлу, тобто розмір стисненого файлу після стиснення до розміру вихідного файлу до стиснення.

Ступінь стиснення - це величина, зворотна до коефіцієнту стиснення.

ISSN 2663 - 4023

Відсоток заощадження - показує стиснення у відсотках.

Результати порівнянь зображені в таблиці 4.

Таблиця 4

Порівняння практичних можливостей щодо коефіцієнту стиснення та ступеня стиснення програм-архіваторів

Програмна реалізація стиснення даних	Розмір файлу до стиснення, Кб	Розмір файлу після стиснення, Кб	Коефіцієнт стиснення	Ступінь стиснення	Відсоток заощадження, %
Власний програмний засіб	3.01	1.17	0.39	2.57	61
Архіватор WinZip	3.01	1.06	3.35	2.84	65
Архіватор PeaZip	3.01	1.10	0.37	2.74	63
Архіватор Bandizip	3.01	1.27	0.42	2.37	58
Архіватор BreeZip	3.01	1.33	0.44	2.26	56
Архіватор gzip	3.01	0.34	0.11	8.85	89

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Було проведено аналіз готових програмних рішень щодо використання алгоритмів стиснення та контрою цілісності у вигляді геш-функцій. Більшість рішень мають широкий спектр вибору форматів стиснення, однак натомість перевірка відбувається лише контрольними сумами, тобто геш-функціями з низькою криптостійкістю. На базі дослідження було обрано для стиснення Deflate. За контроль цілісності відповідала геш-функція “Купина” з Державного стандарту України 7564, яка проявила себе нарівні з геш-функціями світового рівня. При програмній реалізації алгоритму стиснення, було використано бібліотеку zlib з метою забезпечення додаткових методів стиснення пов’язаних з коефіцієнтом. Потім після проведених тестів справних дій запакування даних та їх розгортання, було розроблено програмну реалізацію для геш-функції, провівши додаткове тестування на правильне обрахування дайджесту повідомлень.

При імплементації складових в один засіб, було враховано наскільки зручним для розуміння є інтерфейс, щоб користувачі могли проводити всі дії з легкістю. Після вибору шляху до файлу та режиму геш-функції, програмний застосунок показував чи було успішне здійснення операції стиснення/розгортання та відповідний результат щодо цілісності. Створене програмне забезпечення, за рахунок вмісту державного стандарту, дозволить відповідним структурам використовувати його в своїй роботі також.

Розробка програмного засобу стиснення з контролем цілісності у вигляді геш-функції “Купина” відкриває нові перспективи в подальших дослідженнях:

1. Додатково забезпечити сервіс конфіденційності у вигляді додавання парольного захисту для створених стиснутих даних. Для реалізації можна також використати національний стандарт шифрування ДСТУ 7624:2014, а саме блоковий симетричний шифр – “Калина”. На базі вибору стандарту можна провести дослідження на відповідні алгоритми шифрування, які використовуються в програмних засобах стиснення.

2. Розширити формати стиснення з додатковим врахуванням більшого вмісту форматів вхідних даних. Оскільки реалізований програмний застосунок має один алгоритм стиснення, для одного варіанту формату даних, можна реалізувати більший спектр форматів стиснення і для алгоритмів з втратами даних, аби розширити функціональність, наприклад для стиснення відеоданих, звукових чи графічних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Adaptive huffman coding*. (б. д.). courses.cs.washington.edu. https://courses.cs.washington.edu/courses/csep590a/07au/lectures/lecture02small.pdf
2. Aumasson, J.-P., Neves, S., & Wilcox-O'Hearn, Z. (б. д.). *Blake2*. BLAKE2. https://www.blake2.net/blake2.pdf
3. *Bandizip*. (б. д.). Bandisoft - Bandizip, BandiView, Honeycam, Honeyview. https://uk.bandisoft.com/bandizip/
4. *Breezip - the best free zip&unzip utility for windows*. (б. д.). Breezip - The Best Free Zip&Unzip Utility for Windows. https://www.breezip.com/
5. Budhrani, D. (2020, 3 вересня). *How LZ78 compression algorithm works | hackernoon*. HackerNoon - read, write and learn about any technology. https://hackernoon.com/how-lz78-compression-algorithm-works-x7103tlm
6. Crochemore, M., & Lecroq, T. (б. д.). *Text data compression algorithms*. Researchgate.net. https://www.researchgate.net/publication/2243831_Text_Data_Compression_Algorithms
7. Dheemanth, H. N. (б. д.). *LZW data compression*. AJER. https://www.ajer.org/papers/v3(2)/C0322226.pdf
8. Dipperstein, M. (б. д.-a). *Burrows-Wheeler transform discussion and implementation*. Michael Dipperstein's GitHub Site. https://michaeldipperstein.github.io/bwt.html
9. Dipperstein, M. (б. д.-b). *LZSS (LZ77) discussion and implementation*. Michael Dipperstein's GitHub Site. https://michaeldipperstein.github.io/lzss.html
10. Duarte, F. (2023, 16 березня). *Amount of data created daily (2024)*. Exploding Topics. https://explodingtopics.com/blog/data-generated-per-day
11. *The future of data: Unstructured data statistics you should know - congruity 360*. (б. д.). Congruity 360. https://www.congruity360.com/blog/the-future-of-data-unstructured-data-statistics-you-should-know/
12. *Global data breaches and cyber attacks*. (б. д.). www.itgovernance.co.uk. https://www.itgovernance.co.uk/blog/global-data-breaches-and-cyber-attacks-in-january-2024-29530829012-records-breached
13. *The gzip home page*. (б. д.). The gzip home page. https://www.gzip.org/
14. Information Technology Laboratory. (2012). *Secure hash standard (SHS)* (Federal Information Processing Standards Publication 180-4).
15. Information Technology Laboratory. (2015). *Fips 202* (SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions).
16. Kumar Yadav, A., & Prasad Panday, S. (б. д.). *Burrows-Wheeler post-transformation with effective clustering and interpolative coding*. IOE Graduate Conference. http://conference.ioe.edu.np/ioegc10/papers/ioegc-10-161-10211.pdf
17. *The most popular file compression and management utility*. (б. д.). WinZip for Mac - Zip Files, Unzip Files. https://www.winzip.com/en/?alid=997335683.1717755843
18. *PeaZip free archiver utility, open extract RAR TAR ZIP files*. (б. д.). PeaZip file archiver utility, free RAR ZIP software. https://peazip.github.io/
19. Pu, I. M. (2005). *Fundamental data compression*. Elsevier Science & Technology Books.
20. Sadeghi-Nasab, A., & Rafe, V. (2022). A comprehensive review of the security flaws of hashing algorithms. *Journal of Computer Virology and Hacking Techniques*. https://research.gold.ac.uk/id/eprint/33410/1/paper_en_v1.pdf
21. Sayood, K. (2017). *Introduction to data compression*. Elsevier Science & Technology Books.
22. Stallings, W. (2006). The whirlpool secure hash function. *Cryptologia*, *30*(1), 55–67. https://www2.seas.gwu.edu/~poorvi/Classes/CS381_2007/Whirlpool.pdf
23. Taylor, P. (б. д.). *Data growth worldwide 2010-2025 | Statista*. Statista. https://www.statista.com/statistics/871513/worldwide-data-created/

24. *Teaching guide: Run-length encoding (RLE)*. (б. д.). filestore.aqa.org.uk. https://filestore.aqa.org.uk/resources/computing/AQA-8525-TG-RLE.PDF
25. *What is data compression & what are the benefits*. (б. д.). Barracuda Networks. https://www.barracuda.com/support/glossary/data-compression
26. Глинчук, Л. Я. (2014). *Криптологія*. Вежа-Друк.
27. Коваленко, А. Є. (2020). *Теорія інформації і кодування*. КПІ ім. Ігоря Сікорського 2020.
28. Мінекономрозвитку України. (2015). *Інформаційні технології. Криптографічний захист інформації. Функція хешування* (ДСТУ 7564:2014).
29. Нечипоренко, О., & Корпань, Я. (2018). *Системи збору даних та їх компактного представлення*

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 3 (23), 2024
ISSN 2663 - 4023

Andrii Fesenko
Candidate of Technical Sciences (Ph. D.), Associate professor,
Associate professor at the Department of Cyber Security and Information Protection
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine
ORCID 0000-0001-5154-5324
aafesenko88@gmail.com

Rostyslav Hapon
Cybersecurity student
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine
ORCID 0009-0004-8021-7724
rostikgapon587@gmail.com

COMPARISON OF COMPRESSION ALGORITHMS AND HASH FUNCTIONS OF READY-MADE SOFTWARE SOLUTIONS IN THE CONTEXT OF CREATING YOUR OWN APPLICATION

Abstract. With the rapid development of information technology, working with electronic data has become easy to implement and commonly used. As a result, most organizations have eventually switched entirely to electronic data storage systems. However, the amount of information is increasing exponentially every year, which requires the use of larger storage facilities and resources to process it. In addition, the consumption of information in systems creates increasing risks of data compromise. One of the ways to solve the described problems is to use a software solution for data compression with integrity control by cryptographic hash functions. The article analyzes the lossless data compression algorithms and hash functions used in off-the-shelf software solutions, and additionally compares the hash functions with the State Standard of Ukraine 7564:2014 Kupyna and analyzes the functionality of off-the-shelf solutions. Based on the results of the study, the Deflate compression algorithm was chosen for practical implementation. Compared to other options, the Kupyna hash function performs at the same level or exceeds some indicators, which shows the quality of the algorithm. Most off-the-shelf software solutions do not have a cryptographic function to check the integrity of information. The purpose of this article is to build our own software application based on the study of all components. The article proves that Deflate is better than the listed algorithms in the characteristics of compression algorithms: memory size, performance, compression ratio, number of passes, whether redundancy appears; the determining characteristic when comparing hash functions was the value of crypto-resistance against one of the attacks (by collisions), where Kupyna has better characteristics and has a high value of resistance; ready-made information compression programs were analyzed for the algorithms they use, hash functions, and advantages with disadvantages. Comparative tables with each component for the application and for the created software solutions are provided. Also demonstrated are parts of the code implementation, the results of our own application of information compression with integrity control. The effectiveness of our own tool is compared with the analyzed software.

Keywords: compression; hashing; Deflate; Kupyna hash function; software solution; integrity control; crypto resistance.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Adaptive Huffman coding. (n.d.). courses.cs.washington.edu. https://courses.cs.washington.edu/courses/csep590a/07au/lectures/lecture02small.pdf.
2. Aumasson, J.-P., Neves, S., & Wilcox-O'Hearn, Z. (b.d.). Blake2. BLAKE2. https://www.blake2.net/blake2.pdf.
3. Bandizip (n.d.). Bandisoft - Bandizip, BandiView, Honeycam, Honeyview. https://uk.bandisoft.com/bandizip/.
4. Breezip is the best free file unzip and zip utility for Windows. (n.d.). Breezip is the best free file unzip and zip utility for Windows. https://www.breezip.com/.

5. Budrani, D. (2020, September 3). How the LZ78 compression algorithm works | hackernoon. HackerNoon - read, write and learn about any technology. https://hackernoon.com/how-lz78-compression-algorithm-works-x7103tlm.
6. Crochmore, M., and Lecroix, T. (b.d.). Text data compression algorithms. Researchgate.net. https://www.researchgate.net/publication/2243831_Text_Data_Compression_Algorithms.
7. Dheemanth, H. N. (b.d.). LZW data compression. AJER. https://www.ajer.org/papers/v3(2)/C0322226.pdf.
8. Dipperstein, M. (b.d.-a). Discussion and implementation of the Burroughs-Wheeler transform. Michael Dipperstein's GitHub site. https://michaeldipperstein.github.io/bwt.html.
9. Dipperstein, M. (b.d.-b). Discussion and implementation of LZSS (LZ77). Michael Dipperstein's GitHub site. https://michaeldipperstein.github.io/lzss.html.
10. Duarte, F. (2023, March 16). The amount of data created every day (2024). Exploding Topics. https://explodingtopics.com/blog/data-generated-per-day.
11. The Future of Data: Unstructured data statistics you need to know - congruity 360. Congruity 360. https://www.congruity360.com/blog/the-future-of-data-unstructured-data-statistics-you-should-know/.
12. Global Data Breaches and Cyberattacks. (n.d.). www.itgovernance.co.uk. https://www.itgovernance.co.uk/blog/global-data-breaches-and-cyber-attacks-in-january-2024-29530829012-records-breached
13. Gzip homepage. (n.d.). Gzip home page. https://www.gzip.org/.
14. Information Technology Laboratory. (2012). Secure Hashing Standard (SHS) (Federal Information Processing Standards Publication 180-4).
15. Laboratory of Information Technology. (2015). Fips 202 (SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions).
16. Kumar Yadav, A., & Prasad Panday, S. (b.d.). Post Burroughs-Wheeler transform with efficient clustering and interpolation coding. IOE graduate student conference. http://conference.ioe.edu.np/ioegc10/papers/ioegc-10-161-10211.pdf.
17. The most popular file compression and management utility. (n.d.). WinZip for Mac - Zip Files, Unzip Files. https://www.winzip.com/en/?alid=997335683.1717755843
18. PeaZip is a free archiver utility that opens RAR TAR ZIP archives. PeaZip is a file archiver utility, free RAR ZIP software. https://peazip.github.io/.
19. Pu, I. M. (2005). Fundamental data compression. Elsevier Science & Technology Books.
20. Sadeghi-Nasab, A., & Rafe, V. (2022). A comprehensive review of security flaws in hashing algorithms. Journal of Computer Virology and Hacking Technology. https://research.gold.ac.uk/id/eprint/33410/1/paper_en_v1.pdf.
21. Sayood, K. (2017). Introduction to data compression. Elsevier Science & Technology Books.
22. Stallings, W. (2006). The secure Whirlpool hash function. Cryptology, 30(1), 55-67. https://www2.seas.gwu.edu/~poorvi/Classes/CS381_2007/Whirlpool.pdf
23. Taylor, P. (n.d.). Data growth worldwide 2010-2025 | Statista. Statista. https://www.statista.com/statistics/871513/worldwide-data-created/.
24. Tutorial: Run-length encoding (RLE). (b.d.). filestore.aqa.org.uk. https://filestore.aqa.org.uk/resources/computing/AQA-8525-TG-RLE.PDF
25. What is data compression and what are its benefits. (b.d.). Barracuda Networks. https://www.barracuda.com/support/glossary/data-compression.
26. Glinchuk, L. Ya. (2014). Cryptology. Vezha-Druk.
27. Kovalenko, A. E. (2020). Theory of information and coding. Igor Sikorsky Kyiv Polytechnic Institute. Igor Sikorsky Kyiv Polytechnic Institute 2020.
28. Ministry of Economic Development of Ukraine (2015). Information technologies. Cryptographic protection of information. Hashing function (DSTU 7564: 2014).
29. Nechyporenko, O., & Korpan, Y. (2018). Data collection systems and their compact representation