Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Бакалаврська дипломна робота на тему:

«Засіб неітеративного гешування даних»

Виконав: студент 4 курсу групи 1БС-19 б
спеціальності 125 Кібербезпека
_____ Шагіна Є. С.

Керівник: завідувач каф. ЗІ, д. т. н., проф.
_____ Лужецький В. А.
«19» червня 2023 р.

Рецензент: доцент каф. ПЗ, к. т. н.
_____ Хошаба О. М.
«19» червня 2023 р.

Допущено до захисту

Завідувач кафедри ЗІ
д. т. н., проф.
_____ Лужецький В. А.
«19» червня 2023 р.

Вінниця ВНТУ – 2023 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти І (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ

Завідувач кафедри ЗІ, д. т. н., проф.
_____В. А. Лужецький
«20» березня 2023 року

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Шагіній Єлизаветі Сергіївні

1. Тема роботи: «Засіб неітеративного гешування даних» керівник роботи: Лужецький В. А., завідувач кафедри ЗІ, д.т.н., проф. затверджено наказом ректора ВНТУ від 20 березня 2023 року № 67.
2. Строк подання студентом роботи 19.06.2023 р.
3. Вихідні дані до роботи:
   - обсяг даних для гешування – не більше 2^{64} біт;
   - довжина геш-коду – 256 біт;
   - процес гешування – неітеративний.
4. Зміст текстової частини: Вступ. Огляд відомих підходів до побудови геш-функцій. Розробка методу неітеративного гешування даних. Розробка програмного засобу для гешування. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Підходи до побудови геш-функцій (плакат А4). Метод неітеративного гешування даних (плакат А4). Алгоритм неітеративного гешування даних (плакат А4). Структура програмного засобу (плакат А4). Результати тестування програмного засобу (плакат А4). Порівняльні оцінки алгоритмів гешування (плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Завдання видав	Завдання прийн
1.	Лужецький В.А., зав. кафедри ЗІ, д.т.н., проф.	20.03.23	19.06.2
2.	Лужецький В.А., зав. кафедри ЗІ, д.т.н., проф.	20.03.23	19.06.2
3.	Лужецький В.А., зав. кафедри ЗІ, д.т.н., проф.	20.03.23	19.06.2

7. Дата видачі завдання – 20 березня 2023 року

КАЛЕНДАРНИЙ ПЛАН

No з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Прим
1.	Аналіз завдання. Вступ	20.03.23 – 26.03.23	
2.	Аналіз інформаційних джерел за напрямком бакалаврської дипломної роботи	27.02.23 – 09.04.23	
3.	Розробка рішень, моделей, алгоритмів	10.04.23 – 23.04.23	
4.	Практична реалізація, моделювання, експериментування, результати	24.04.23 – 21.05.23	
5.	Аналіз виконання завдання, висновки	22.05.23 – 24.05.23	
6.	Оформлення пояснювальної записки, підготовка ілюстративного матеріалу	25.05.23 – 31.05.23	
7.	Попередній захист БДР	01.06.23 – 1.06.23	
8.	Представлення БДР до захисту, рецензування	16.06.23-19.06.23	
9.	Представлення БДР до захисту	20.06.23-23.06.23	

Студент _____ Шагіна Є.С

Керівник роботи _____ Лужецький В

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 53 сторінок формату А4, на яких є 14 рисунків, 4 таблиці, список використаних джерел містить 19 найменувань.

У даній дипломній роботі був розроблений засіб неітеративного гешування даних з використанням мови програмування Python. Головною метою розробки було створення ефективного та надійного інструменту, який забезпечує швидке та безпечне обчислення геш-значень для вхідних даних.

У процесі розробки засобу було проаналізовано різні методи гешування та вибрано неітеративний побайтовий метод, який забезпечує високу швидкодію обчислень. Засіб було реалізовано з використанням стандартних криптографічних функцій та бібліотек Python, що забезпечує його надійність та сумісність з іншими програмними рішеннями.

Основні переваги розробленого засобу неітеративного гешування включають швидкодію обчислень, низький рівень складності реалізації та високу стійкість до злому. Використання такого методу дозволяє ефективно застосовувати гешування для різних задач, таких як збереження паролів, цифровий підпис, контроль цілісності даних тощо.

Отримані результати підтверджують ефективність розробленого засобу. Він дозволяє обчислювати геш-значення швидко і безпечно, що робить його варіантом вибору для багатьох програмних проектів, особливо з обробкою великих обсягів даних.

Ключові слова: неітеративний метод гешування даних, сучасні методи гешування.

ABSTRACT

The bachelor's thesis consists of 53 pages of A4 format, on which there are 14 figures, 4 tables, the list of used sources contains 19 names.

In this thesis, a tool for non-iterative data hashing was developed using the Python programming language. The main goal of the development was to create an efficient and reliable tool that provides fast and secure calculation of hash values for input data.

During the development of the tool, various hashing methods were analyzed and a non-iterative byte-by-byte method was selected, which provides high calculation speed. The tool was implemented using standard cryptographic functions and Python libraries, which ensures its reliability and compatibility with other software solutions.

The main advantages of the developed non-iterative hashing tool include the speed of calculations, low level of implementation complexity and high resistance to hacking. Using this method allows you to effectively apply hashing for various tasks, such as saving passwords, digital signature, data integrity control, etc.

The obtained results confirm the effectiveness of the developed tool. It allows hash values to be calculated quickly and securely, making it the option of choice for many software projects, especially those dealing with large amounts of data.

Keywords: non-iterative data hashing method, analysis of modern hashing methods.

ЗМІСТ

ВСТУП

В сучасному світі, де дані відіграють ключову роль у різних сферах, актуальність гешування даних є надзвичайно високою. Зростання обсягів інформації, її швидкі обробка та передача, а також постійні загрози злому та несанкціонованому доступу до даних роблять геш-функції невід'ємними складниками систем безпеки та захисту інформації.

Однією з ключових аспектів актуальності гешування даних є його роль у забезпеченні цілісності інформації. Геш-функції дозволяють перетворювати дані будь-якого розміру в геш-коди фіксованої довжини. Перевірка цілісності даних здійснюється порівнянням геш-кодів, що дозволяє виявити навіть незначні зміни в початкових даних. Це має велике значення у багатьох галузях, наприклад, у банківському секторі, медичній сфері, електронній комерції та урядових системах, де необхідна гарантована цілісність даних.

Також, геш-функції є важливим інструментом у захисті від колізій. Колізія виникає, коли два різних набори даних мають однаковий геш-код. Уникнення колізій є критичним у криптографії, аутентифікації, підпису даних та інших застосуваннях, де важливо гарантувати унікальність геш-кодів для різних даних. Розробка нових та покращення існуючих геш-функцій важливі для забезпечення високої стійкості та надійності уникнення колізій.

Застосування геш-функцій також має значення для безпеки мереж та інших критичних систем. Геш-функції дозволяють забезпечити конфіденційність даних шляхом гешування паролів, ключів або інших конфіденційних інформаційних елементів. Це є важливим у сферах, де велика кількість користувачів має доступ до системи та необхідно забезпечити конфіденційність та безпеку інформації.

Отже, розробка та вдосконалення неітеративного засобу гешування даних є важливим напрямком досліджень. Швидкість та ефективність обробки даних мають вирішальне значення у сучасному світі, де об'єми інформації зростають експоненціально. Застосування неітеративного підходу до гешування даних може

значно покращити процес обробки та збереження інформації, забезпечуючи високу швидкість, ефективність та стійкість захисту даних

Ця дипломна робота присвячена дослідженню та розробці засобу для неітеративного гешування даних. Об'єктом дослідження є процес контролю цілісності даних з використанням геш-функцій, а предметом дослідження є алгоритм та засіб для неітеративного гешування даних.

Метою цієї роботи є пришвидшення процесу гешування даних за рахунок використання неітеративного підходу. Для досягнення цієї мети в роботі будуть розглянуті та вирішені наступні задачі:

1. Аналіз відомих підходів для побудови геш-функцій. В рамках цієї задачі будуть проаналізовані існуючі методи гешування даних, їх переваги та недоліки, а також визначені вимоги до ефективного та стійкого неітеративного підходу.
2. Розробка методу неітеративного гешування даних. На основі проведеного аналізу буде запропонований новий метод, який буде використовувати неітеративний підхід для швидкого та ефективного гешування даних.
3. Розробка алгоритму неітеративного гешування даних. Буде розроблений алгоритм, який реалізуватиме запропонований метод неітеративного гешування даних. Алгоритм буде оптимізований для досягнення максимальної швидкості та ефективності обробки даних.
4. Розробка програмного засобу для неітеративного гешування даних. На основі розробленого алгоритму буде реалізовано програмний засіб, який дозволить застосовувати неітеративний підхід до гешування даних. Програмний засіб буде протестований та оцінений з точки зору швидкості, ефективності та стійкості гешування.

Ця дипломна робота має важливе значення для практичного застосування гешування даних у сучасному світі. Результати дослідження та розробки засобу для неітеративного гешування даних можуть бути використані в різних сферах, де важлива цілісність та безпека інформації.

1 АНАЛІЗ ВІДОМИХ ПІДХОДІВ ДО ПОБУДОВИ ГЕШ-ФУНКЦІЙ

1.1 Криптографічні геш функції

Діяльність, спрямована на забезпечення захисту конфіденційності, цілісності та доступності важливої для держави, суспільства та особи інформації, у тому числі загальнодоступної інформації, яка захищена правовими нормами, називається захистом інформації [6]. Згідно з основним нормативно-правовим актом України, завдання із захисту національного інформаційного простору України покладено на Систему державного таємного зв’язку, концептуальні питання створення, функціонування, розвитку й використання якої регулюються Конституцією України, законами України "*Про інформацію*", ДСТУ 3396.2 – 97 "*Захист інформації. Технічний захист інформації. Терміни та визначення*", "*Положення про порядок здійснення криптографічного захисту інформації в Україні*" (ТЗІ) [1].

Функції гешування є криптографічними примітивами, що широко використовуються практично у всіх електронних цифрових підписах(ЕЦП), в деяких симетричних крипто примітивах, криптографічних механізмах і протоколах автентифікації, встановлення, узгодження, підтвердження ключів. Крім, зрозуміло, криптографічної стійкості до них висуваються жорсткі вимоги відносно складності(швидкості) гешування. Причому, це особливо важливо при виконанні ЕЦП та генеруванні детермінованих випадкових послідовностей (псевдовипадкових послідовностей) [2].

Одним із сучасних підходів до побудови геш-функцій є створення архітектурних моделей, де для підвищення продуктивності використовуються паралельні обчислення. Крім того, ми будемо викликати такі геш-функції паралельно. Тобто, враховуючи сучасні тенденції, виникла необхідність вдосконалення існуючих систем критеріїв і метрик та розробки методу порівняння геш-функцій з урахуванням нових пропозицій щодо їх архітектурних властивостей.

Крім того, з розвитком методів побудови геш-функцій такі показники, як швидкість, перестали бути зрозумілими. Наприклад, для паралельної геш-функції

значення продуктивності на багатоядерному процесорі відрізняються від показників на одноядерному процесорі [3].

Існують також суттєві відмінності в продуктивності паралельних обчислень з точки зору програмної та апаратно-програмної та апаратної реалізації. Таким чином, у реалізації програмного забезпечення потоки виконання за своєю суттю сперечаються, що спричиняє додаткові витрати на синхронізацію.

Геш-функція – це деяка функція h(K), яка бере якийсь ключ K і повертає адресу, за якою проводиться пошук в геш-таблиці, щоб отримати інформацію, пов'язану з K.

Наприклад, K – це номер телефону абонента, а шукана інформація – його ім'я. Функція в даному випадку нам точно визначить, за якою адресою знайти шукане. Якісна геш-функція повинна задовольняти двом вимогам:

− її обчислення повинно виконуватися дуже швидко;

− вона повинна мінімізувати кількість колізій.

Отже, перша властивість якісної геш-функції залежить від комп'ютера, а друга – від даних.

Функція гешування є важливим поняттям в криптографії та інформаційній безпеці. Її основна мета - перетворити будь-який вхідний набір даних фіксованої довжини в геш-значення, яке є унікальним для кожного вхідного набору даних. Основні вимоги до функції гешування включають наступне:

1. Односторонність: Функція гешування має бути обчислювально важкою зворотною функцією, що означає, що надзвичайно складно відновити початкові дані з геш-значення. Коли ми застосовуємо функцію гешування до вхідних даних, ми отримуємо унікальне геш-значення. Це значення може бути використане для представлення вихідних даних у компактній формі. Однак, через властивість односторонності, важко (практично неможливо) обчислити початкові дані з геш-значення.Функція гешування повинна бути обчислювально важкою зворотною функцією. Це означає, що з геш-значення важко відновити початкові дані. Для будь-якого даного коду h повинне бути практично неможливо обчислити x, для якого H(x) = h (Рисунок 1.1)

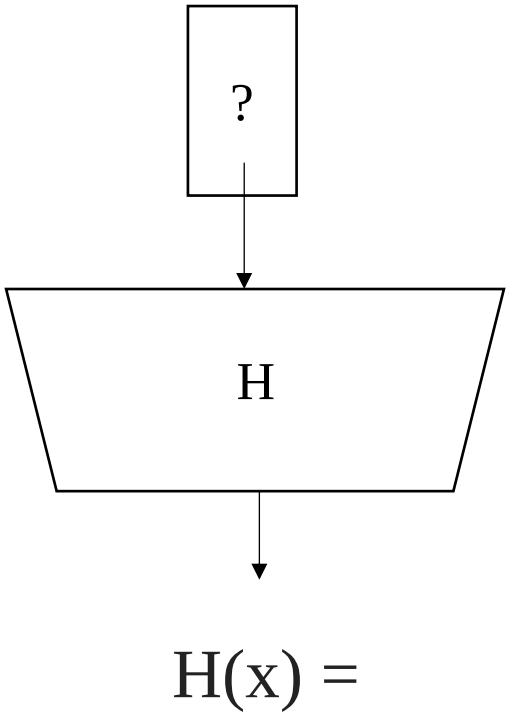


Рисунок 1.1 – Вимога односторонності геш-функції

2. Слабка опірність колізіям (стійкість до пошуку першого прообразу): Для будь-якого даного блоку даних x має бути практично неможливо знайти інший блок даних y, для якого H(x) = H(y). Ця властивість гарантує, що з геш-значення важко відновити початкові дані(Рисунок 1.2)

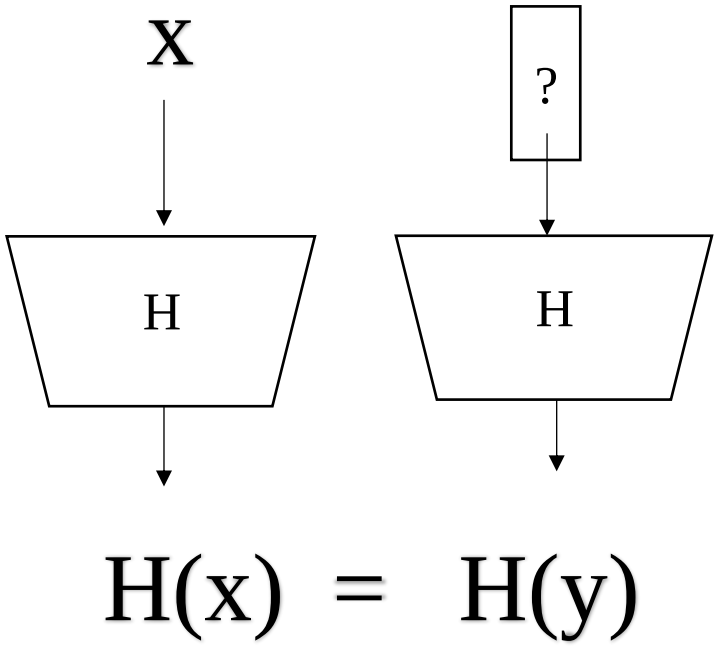


Рисунок 1.2 - Вимога слабкої опірності колізіям геш-функції

3. Сильна опірність колізіям (стійкість до колізій): Повинно бути практично неможливо знайти будь-яку пару різних значень x і y, для яких H(x) = H(y). Ця властивість забезпечує високу ймовірність того, що два різних вхідних набори даних матимуть різні геш-значення (Рисунок 1.3).

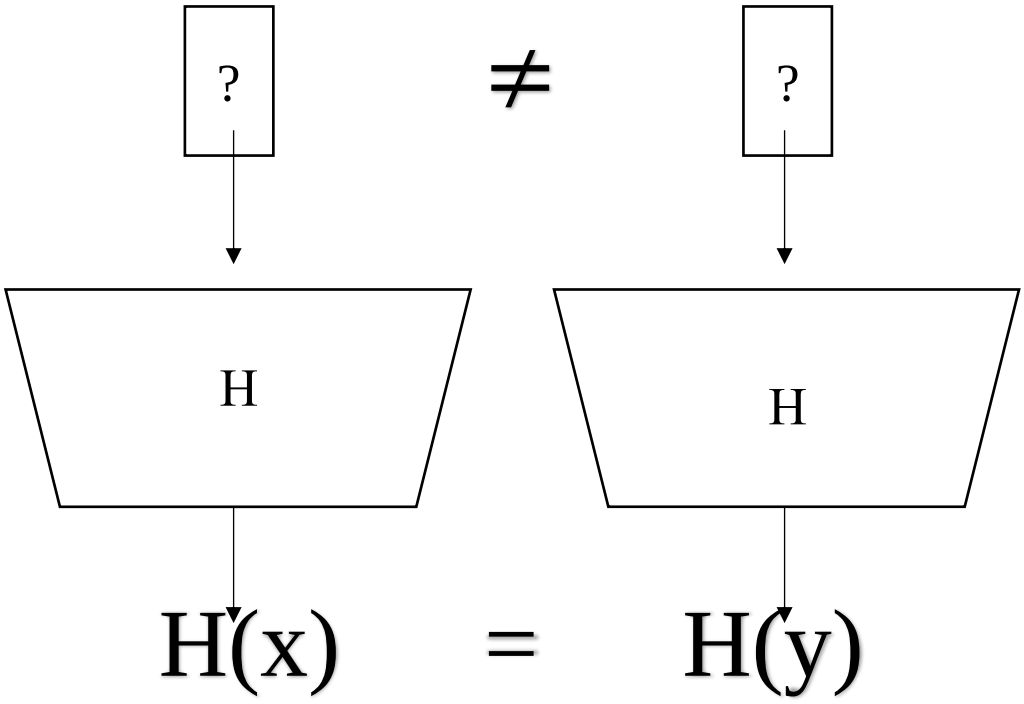


Рисунок 1.3 – Вимога сильної опірності колізіям

4. Детермінізм: Функція гешування повинна давати однаковий результат для одного й того ж вхідного набору даних. Це дозволяє легко перевіряти, чи співпадають два геш-значення.

5. Швидкість обчислень: Функція гешування повинна бути ефективною з точки зору обчислень, щоб забезпечити швидку обробку великих обсягів даних. Це важливо, оскільки функції гешування часто використовуються для обробки великої кількості даних, таких як паролі, повідомлення або файли. Швидкість обчислень

геш-функції має бути достатньою, щоб забезпечити ефективну роботу з цими даними без зайвого затриму.

6. Вплив змін вхідних даних: Навіть незначні зміни в початкових даних повинні призводити до істотних змін у вихідному геш-значенні. Це властивість, яка забезпечує надійність контролю цілісності даних [4].

Геш-функції в криптографії можуть бути поділені на дві основні категорії: безключові геш-функції і ключові геш-функції. Крім того, існують також коди виявлення помилок, такі як Modification Code (MDC) і Message Integrity Code (MIC), які використовуються в системах, де є або немає повна довіра між користувачами.

Безключові геш-функції, такі як SHA-256 і SHA-3, є загального призначення і використовуються для генерації геш-значень з будь-яких вхідних даних. Вони не вимагають використання ключа і можуть бути використані в системах, де довіра між користувачами вже встановлена або відсутня.

У сфері криптографії і безпеки даних, ключові геш-функції, наприклад HMAC (Hash-based Message Authentication Code), відіграють важливу роль у забезпеченні автентифікації повідомлень та збереженні цілісності даних. Вони використовуються для обчислення кодів автентифікації повідомлень (MAC), які є особливими геш-значеннями, створеними за допомогою секретного ключа.

Ключові геш-функції є особливими в тому випадку, коли між користувачами вже встановлена довіра. Це означає, що як відправнику, так і отримувачу повідомлення відомий спільний секретний ключ, який використовується для обчислення та перевірки MAC. Такий ключ дозволяє відправнику створити MAC для повідомлення, а отримувачу перевірити цей MAC для підтвердження автентичності та цілісності даних.

Цей підхід заснований на використанні секретного ключа, який забезпечує конфіденційність із геш-функцією. Ключові геш-функції широко використовуються в різних системах, таких як протоколи безпеки мереж, електронні підписи, цифрові сертифікати та інші схеми забезпечення безпеки даних.

Проте варто відзначити, що використання ключових геш-функцій передбачає встановлену довіру між сторонами, оскільки вони мають обмінятися спільним секретним ключем. Це може бути доцільним у ситуаціях, де користувачі вже встановили взаємну довіру, наприклад, в рамках попередньо обговорених стосунків або за допомогою інших механізмів обміну ключами. Однак, в ситуаціях, коли довіра ще не встановлена, ключові геш-функції можуть бути менш застосовними, оскільки вимагають обміну секретним ключем між сторонами.

Таким чином, ключові геш-функції є потужним інструментом для забезпечення автентичності та цілісності даних у випадках, коли вже існує встановлена довіра між користувачами та обмін секретним ключем є можливим. Вони дозволяють ефективно перевіряти автентичність повідомлень та гарантувати, що дані не були змінені під час передачі.

Коди виявлення помилок, такі як MDC і MIC, використовуються для виявлення помилок у передачі або зберіганні даних. Вони додаються до повідомлень або блоків даних для перевірки їх цілісності. Ці коди використовуються в системах з різним рівнем довіри між користувачами, які дозволяють виявляти будь-які зміни або пошкодження даних під час передачі.

Таким чином, безключові геш-функції, ключові геш-функції та коди виявлення помилок знаходять своє застосування в різних системах залежно від рівня довіри між користувачами:

1. Безключові геш-функції (наприклад, SHA-256, SHA-3): Використовуються в системах, де довіра між користувачами вже встановлена або відсутня. Вони забезпечують цілісність даних і можуть використовуватись для перевірки, чи були дані змінені під час передачі або зберігання.

2. Ключові геш-функції (наприклад, HMAC): Використовуються в системах, де є встановлена довіра між користувачами. Вони використовують спільний секретний ключ для створення кодів автентифікації повідомлень (MAC), що дозволяє перевірити автентичність та цілісність повідомлень.

3. Коди виявлення помилок (MDC або MIC): Використовуються в системах з різним рівнем довіри між користувачами. Ці коди додаються до повідомлень або

блоків даних для виявлення будь-яких змін або пошкоджень під час передачі. Вони можуть бути використані для перевірки цілісності даних у відкритому середовищі, де немає повної довіри між користувачами.[6].

Отже, вибір між безключовими геш-функціями та ключовими геш-функціями, або використанням кодів виявлення помилок, залежить від конкретних вимог безпеки та рівня довіри між користувачами в системі. Кожен з цих підходів має свої переваги та обмеження і може бути використаний для забезпечення захисту даних у відповідності з вимогами конкретної системи.

Загальні принципи побудови геш-функцій передбачають оброблення введених даних, таких як повідомлення або файли, як послідовності блоків фіксованої довжини. Кожен блок має розмір т-бітів, де т є фіксованим значенням.

Обчислення значення геш-функції відбувається шляхом поетапної обробки кожного блоку послідовно. Починаючи з першого блоку і закінчуючи останнім, кожен блок додається до проміжного результату обчислення. Цей процес продовжується до досягнення остаточного значення функції гешування.

Усі проміжні значення функції гешування, а також остаточне значення, представлені у формі п-бітових блоків. Розмір п визначається вибором конкретної геш-функції і може бути фіксованим для даної функції.

Такий підхід дозволяє ефективно обробляти дані блок за блоком, що спрощує реалізацію геш-функцій та дозволяє їм працювати з повідомленнями різних розмірів.

Загальні принципи побудови геш-функцій надають основу для конкретних алгоритмів та конструкцій, які використовуються в різних геш-функціях. Враховуючи ці принципи, конструктори можуть створювати геш-функції з необхідними властивостями, такими як стійкість до колізій, безпека та ефективність.

Побудова ефективних та безпечних геш-функцій є одним із важливих завдань у сучасній криптографії. Геш-функції забезпечують перетворення вхідних даних будь-якої довжини в геш-код фіксованої довжини, що використовується для забезпечення цілісності, автентичності та безпеки інформації. У процесі побудови

геш-функцій застосовуються різні підходи, включаючи основані на складнообчислювальних математичних задачах, алгоритмах блокового шифрування та розроблення геш-функцій з нуля:

1. На основі складнообчислювальної математичної задачі: Один з підходів до побудови геш-функцій полягає в основі складнообчислювальних математичних задач. Це означає, що геш-функція будується на основі задачі, яка вимагає значних обчислювальних зусиль для її вирішення. Наприклад, RSA геш-функція (RSA-H) використовує проблему факторизації великих простих чисел як основу для своєї безпеки. Зараз існують різні геш-функції, які використовують різні математичні задачі, такі як дискретне логарифмування, різні типи кривих та інші.

2. На основі алгоритмів блокового шифрування: Інший підхід до побудови геш-функцій полягає у використанні алгоритмів блокового шифрування. В цьому підході блоки вхідних даних обробляються за допомогою спеціальних шифрувальних алгоритмів, що призводить до отримання вихід вихідного геш-коду. Один з найвідоміших прикладів цього підходу - конструкція Меркла-Дамгарда, яка базується на зведенні складної геш-функції до блокового шифру. У цій конструкції, вхідне повідомлення розбивається на блоки, які обробляються послідовно. Кожен блок додається до попереднього результату або гешу, утворюючи ланцюжок обчислень, що завершується отриманням геш-коду повідомлення.

3. Розроблення геш-функції з нуля: Інший підхід до побудови геш-функцій полягає у розробленні нової геш-функції з нуля. Це означає, що замість використання вже існуючих алгоритмів або математичних задач, конструктор геш-функції розробляє нову структуру та алгоритм для обчислення геш-кодів. Цей підхід може включати нові ідеї, методи та алгоритми, які забезпечують високий рівень безпеки та ефективності геш-функції.

Кожен з цих підходів має свої переваги та особливості. Вибір конкретного підходу залежить від вимог до безпеки, швидкодії та інших факторів. Розуміння різних підходів допомагає конструкторам геш-функцій вибрати найбільш підходящий метод для побудови геш-функції залежно від конкретних потреб та вимог системи. користувачами, ключові геш-функції і коди

автентифікації повідомлень (MAC) можуть забезпечувати більш високий рівень безпеки, оскільки вони вимагають обміну та захисту спільного секретного ключа. З іншого боку, безключові геш-функції і коди виявлення помилок (MDC або MIC) можуть бути використані у відкритих системах, де немає повної довіри між користувачами, для забезпечення цілісності та виявлення змін під час передачі або зберігання даних. Вони дозволяють виявляти будь-які неправильності або помилки, які можуть бути викликані небезпечними атаками або випадковими збоїв системи.

Наприклад, безключові геш-функції можуть бути використані для перевірки цілісності файлів, шифрування паролів, генерації контрольних сум та перевірки цілісності повідомлень. Вони є швидкими і ефективними, але не забезпечують автентифікацію та захист від зловживання.

З іншого боку, ключові геш-функції і коди автентифікації повідомлень (MAC) забезпечують не тільки цілісність даних, але й автентичність. Вони використовують спільний секретний ключ для створення підпису, який може бути перевірений отримувачем для підтвердження автентичності повідомлення. Ключові геш-функції і коди MAC використовуються в протоколах аутентифікації, цифрових підписах, безпечних протоколах передачі даних і багатьох інших криптографічних застосуваннях

1.2 Використання геш-функції як методу захисту інформації

Сьогодні використовується безліч різних криптографічних геш-функцій, щоб дати вам уявлення про те, як вони можуть виглядати, ми наведемо приклад, який задовольняє деякі, але не всі, властивостей, які мають криптографічні геш-функції

Приклад геш-функції називається *chunked XOR. Exclusive*, або *XOR,* - це функція, яка, коли задана пара входів, виводить *true* (або 1), якщо входи різні, а *false* інакше.

Так, наприклад, яблуко *XOR* банан = *1*, яблуко *XOR* яблуко = *0, 0 XOR 1 = 1, 1 XOR 1 = 0*. Ми можемо взяти ланцюжок *XORs* на двійкових числах *(0s* і *1s)* і

отримати змістовну відповідь: *1 XOR 1 XOR 0 = 0, 1 XOR 1 XOR 0 XOR 1 = 1.* Для послідовності двійкових чисел *XOR* повертає *1*, якщо в ланцюжку є непарне число *1s* і *0* в іншому випадку.

Chunked XOR працює на двійковому вході. (Якщо ваш вхід не двійковий, ви могли б представляти його в двійковій спочатку, так само, як і комп'ютер.) Ми групуємо вхідні дані на шматки, рівні розміру виводу геш-функції - наприклад, групи з восьми бітів. Ми вирівнюємо шматки вертикально, а потім *XOR* вміст кожного стовпця [7]:

1) вхідні дані: 00111011 1110101 00101000 01011 01011000 11001110;
2) шматками:

   00111011
   1110101
   00101000
   001011
   01011000
   11001110;
3) або стовпці *XOR*: 01010011 (вихід).

Це геш-функція в тому, що незалежно від довжини вхідних даних, вихід завжди буде мати однакову довжину (вісім у цьому прикладі). Ви повинні бути в змозі побачити, що *chunked XOR* задовольняє першій властивості криптографічних геш-функцій.

Однак він виходить з ладу на інших властивостях. Легко створити вхідне повідомлення (але не обов'язково бажане вхідне повідомлення) з заданим гешем виведення - наприклад, ви можете об'єднати 11111111 11111111 на результат гешу. З тієї ж причини можна створити кілька повідомлень, що мають однаковий геш виведення. Нарешті, зміна одного біта вхідного повідомлення змінить лише один біт вихідного геша.

Дайджести геш функції мають бути рівномірно розподілені, тобто геш значення має генеруватися у вихідному діапазоні з однаковою рівномірністю, тобто діапазоном є усі значення дайджестів, наприклад для MD5 вихідним діапазоном є

значення від 00000000000000000000 до ffffffffffffff, якщо їх представити у 16-рядковому форматі. А кожний біт дайджесту повинен генеруватися з вірогідністю, що наближається до 50%

Геш функція повинна приймати дані будь-якого формату, зазвичай алгоритми гешування приймаються масив байтів, що забезпечує їх універсальність. Геш функція є детерміністичною, вхідне повідомлення завжди генерує один дайджест.

Тобто алгоритми гешування не використовують генератори псевдовипадкових чисел, або певні вхідні дані, які неможливо отримати при повторному використанні алгоритму (наприклад часу доби, атмосферного тиску інше).

Також не повинно мати значення адреси пам’яті об’єкта. Припускається використання зерна генерації або вектору ініціалізації, але зазвичай такі механізми не використовуються. Ефективність геш функції є досить важливим аспектом, адже окрім криптографічних методів захисту інформації, вони використовуються для зберігання та пошуку даних. (Таблиця 1.1)

Так наприклад при зберіганні даних простіше зберігати ключ значення файлу, адже його обчислення займає визначений проміжок часу, та не зростає зі збільшенням розміру файлу. Тому геш-функції повинні обчислюватись досить швидко, на відміну від алгоритмів блокового симетричного шифрування, що є дуже схожими за будовою та принципом дії. Для цього при обчисленні використовується мінімальна кількість інструкцій. Зазвичай для забезпечення швидкості використовуються прості логічні та арифметичні оператори. Але використання операторів множення і особливо ділення використовуються рідко через їх складність реалізації на апаратному рівні .

Таблиця 1.1 – Порівняння деяких геш-функцій

Геш-функція	Рік	Розробники	Довжина блоку	Довжина дайджесту	Кількість раундів
MD5	1992	Ronald Rivest	512	128	64
RIPEMD	1992	The RIPE Consortium	512	128	48
SHA-1	1995	NSA	512	160	80
SHA-256	2002	NSA	512	256	64
SHA-512	2002	NSA	1024	512	80
Whirlpool	2004	Vincent Rijment, Paulo Barreto	512	512	10
BLAKE-256	2008	Jean-Philippe Aumasson, Luca Henzen, Will Meler, Raphael C.-W, Phan	512	256	14
Купина	2014	ПАТ «Інститут інформаційних технологій»	512	Від 8 до 512 біт	10 або 14

Одним із визначальних атрибутів алгоритмів гешування є наявність колізій. Колізією є випадок, коли дайджест одного повідомлення є ідентичним до дайджесту іншого. Це пов’язано з тим, що при визначеній довжині дайджесту, існує невизначена кількість можливих повідомлень. Вірогідність колізії, є незначущою,

і у криптостійких алгоритмів геш-функцій наближається до теоретичного мінімуму[14].

Одним із важливих аспектів геш-функцій є їх швидкодія. У контексті зберігання даних, особливо у великих обсягах, важливо мати ефективний спосіб обчислення геш-значень. Геш-функції повинні бути в змозі швидко обчислювати дайджести, а час обчислення не повинен зростати пропорційно до розміру вхідних даних. Такий підхід дозволяє забезпечити ефективну обробку даних і зменшити час, необхідний для виконання операцій з геш-значеннями.

Для досягнення високої швидкодії геш-функцій, використовуються різні підходи. Зазвичай вони базуються на використанні простих логічних та арифметичних операторів. Операції множення і ділення, які можуть бути складні в реалізації на апаратному рівні, використовуються рідко або взагалі уникаються. Це дозволяє забезпечити швидкодію обчислень і зберегти ресурси.

Однак, одним з викликів, з якими можуть зіткнутися геш-функції, є наявність колізій. Колізія виникає, коли два різних вхідних повідомлення мають однаковий геш-дайджест. Хоча використовувані криптографічні геш-функції мають незначну ймовірність колізій, все ж існує можливість їх виникнення через теоретичну недетермінованість. У криптостійких геш-функцій ця ймовірність дуже низька, що дозволяє їм забезпечувати високий рівень надійності і стійкості.

Аналізуючи всі ці фактори, вибір відповідного геш-алгоритму стає важливим завданням. Враховуючи вимоги до швидкодії, надійності та стійкості, необхідно знайти оптимальний баланс, щоб задовольнити потреби конкретного застосування. Враховуючи розвиток технологій та появу нових викликів у сфері кібербезпеки, постійна оцінка та вдосконалення геш-функцій стає важливим завданням для забезпечення безпеки та надійності систем.

Зазвичай при використанні наявність колізій або ігнорується, або вирішується (наприклад додавання солі при зберіганні геш-таблиці паролів), але також існує можливість використання ін’єктивної геш-функції . Вірогідність колізії напряму залежить від довжини дайджесту геш-функції, але великий розмір ключа напряму негативно впливає на швидкість обичлення. Як вже було зазначено за

принципом дії, алгоритми гешування за механізмом дії на алгоритми блокового симетричного шифрування.

Розмір кожного блоку даних змінюється залежно від алгоритму. Зазвичай розміри блоків становлять від 128 до 512 біт. Якщо повідомлення не «заповнює» блок, створюються відступи. Над кожним блоком здійснюються визначені операції елементарної логіки та арифметики. Перший блок «перемішується» з вектором ініціалізації, що, зазвичай, чітко визначений алгоритмом. Другий блок «перемішується» з першим і т.д. У криптографічних функціях необхідна наявність лавинного ефекту, його показником є наявність двох істотно різних дайджестів для повідомлень, які відрізняються одним бітом .

Отже, це математична функція яка перетворює числове вхідне значення в інше стиснене числове значення. Геш функція має повертати дайджест сталої довжини, а довжина повідомлення не має значення. Вона повинна бути швидкою та односторонньою.

Криптографічно стійким вважається алгоритм гешування, якщо в ньому наявні такі ознаки, як рівномірність розподілення дайджестів, детерміністичність, наявність лавинного ефекту, а вірогідність колізії є теоретично мінімальною.

Отож, геш-функції надзвичайно корисні і з’являються майже у всіх програмах захисту інформації. Геш-функція – це математична функція, яка перетворює числове вхідне значення в інше стиснене числове значення. Вхід у геш-функцію є довільної довжини, але вихід завжди має фіксовану довжину. Значення, що повертаються геш-функцією, називаються дайджестом повідомлення або просто геш-значеннями[6].

1.3 Постановка задачі

Метою даної роботи є пришвидшення процесу гешування даних за рахунок використання неітеративного методу гешування. Гешування є важливим кроком в області криптографії та захисту даних, оскільки дозволяє забезпечити конфіденційність, цілісність та аутентичність інформації.

Неітеративний метод гешування дозволяє уникнути ітераційного процесу обчислення геш-значень. Цей підхід полягає в розбитті вхідного повідомлення на послідовність байтів. Потім обчислюється кількість байтів однакового змісту та фіксуються їх позиції. З цими даними обчислюється геш-значення.

В рамках роботи вирішуються такі задачі: аналіз відомих підходів для побудови геш-функцій, розробка методу неітеративного гешування даних, розробка алгоритму неітеративного гешування даних та розробка програмного засобу для неітеративного гешування даних.

Неітеративний метод гешування дозволяє уникнути повторних обчислень ітераційного процесу, забезпечуючи ефективність та швидкість обчислення геш-значень. Розбиття вхідного повідомлення на байти та фіксування позицій байтів з однаковим змістом дозволяє створити унікальні геш-значення.

Результатом виконання роботи буде програмний засіб, який здатний використовувати неітеративний метод гешування для швидкого та надійного обчислення геш-значень. Цей засіб буде забезпечувати високу безпеку інформації, збереження цілісності даних та захист від несанкціонованого доступу. Такий програмний засіб буде корисним для різних сфер, включаючи криптографію, бази даних, мережеву безпеку та інші області, де важливо забезпечити надійний захист даних.

Використання неітеративного методу гешування в розробленому програмному засобі дозволить ефективно та швидко гешувати дані, забезпечуючи безпеку та надійність обробки інформації. Більш того, цей метод дозволяє зберегти час і ресурси, які були б витрачені на ітеративні обчислення геш-значень. В результаті, користувачі зможуть швидше отримати геш-значення для своїх даних, що є важливим аспектом в багатьох сучасних додатках та системах.

Розробка програмного засобу, який використовує неітеративний метод гешування, має велике значення в сучасному світі, де безпека та ефективність обробки даних є критичними аспектами. Такий засіб стає надійним інструментом для захисту конфіденційної інформації та забезпечення цілісності даних у різних областях, де важлива безпека та захист інформації.

Одним з основних аргументів на користь використання неітеративного методу гешування є його надійність. Геш-функції, що застосовуються у цьому методі, мають властивість непередбачуваності та стійкості до колізій. Це означає, що навіть незначні зміни в початкових даних призводять до істотних змін у вихідному геш-значенні. Така властивість робить метод гешування надійним з точки зору виявлення будь-яких спроб змінити або підробити дані.

Крім того, неітеративний метод гешування відрізняється високою швидкістю обчислення геш-значень. Це особливо важливо в ситуаціях, коли необхідно обробляти великі обсяги даних в реальному часі. Швидкість роботи методу поєднується з його надійністю, створюючи привабливий варіант для різноманітних застосувань, де обробка даних вимагає високої швидкодії та безпеки.

Програмні засоби, що використовують неітеративний метод гешування, знаходять широке застосування в різних галузях. Наприклад, в області криптографії та безпеки інформації, геш-функції використовуються для захисту паролів, перевірки цілісності даних та створення цифрових підписів. В сфері мережевих протоколів геш-функції можуть використовуватись для ідентифікації та автентифікації користувачів, а також для забезпечення безпеки комунікацій між системами.

Крім того, неітеративний метод гешування може бути використаний у системах зберігання даних, де важлива ефективність та швидкість доступу до інформації. Застосування геш-функцій дозволяє швидко перевіряти, чи були дані змінені або пошкоджені, що є важливим аспектом в системах з великим обсягом даних.

Таким чином, розробка програмного засобу, який використовує неітеративний метод гешування, має значний практичний потенціал у багатьох сферах. Вона допомагає забезпечити безпеку, цілісність та ефективність обробки даних, що стає важливим фактором у сучасному інформаційному світі. Надійність, швидкість та широке застосування методу гешування роблять його необхідним елементом при проектуванні та розробці програмних засобів з високими вимогами до безпеки та ефективності. Одним з викликів, пов'язаних з гешуванням даних, є

збалансування між швидкістю та стійкістю. Деякі методи гешування можуть бути швидкими, але менш стійкими до атак злому. Тому важливим завданням при розробці програмного засобу для неітеративного гешування даних є забезпечення оптимальної комбінації швидкості та стійкості геш-функцій.

Крім того, у процесі розробки програмного засобу необхідно враховувати такі фактори, як підтримка різних алгоритмів гешування, можливість розширення функціональності, зручний інтерфейс користувача та сумісність з іншими програмними рішеннями.

У процесі розробки програмного засобу, який використовує неітеративний метод гешування, важливо враховувати різні фактори, щоб забезпечити якість та ефективність програмного продукту. Один з таких факторів - підтримка різних алгоритмів гешування.

Крім того, розробник повинен передбачити можливість розширення функціональності програмного засобу, пов'язану з гешуванням. Це може включати розробку нових геш-функцій, врахування особливих потреб користувачів або впровадження додаткових заходів безпеки. Гнучкість та розширюваність програмного засобу дають змогу відповідати змінним вимогам і забезпечувати масштабованість системи.

При розробці програмного засобу також важливо забезпечити зручний інтерфейс користувача. Розробник повинен забезпечити легкість використання і зрозумілість функціональності, пов'язаної з гешуванням. Це може включати створення інтуїтивно зрозумілих інструментів, графічних інтерфейсів або документації, що пояснює процеси гешування та використання геш-функцій.

Нарешті, розробник повинен враховувати сумісність програмного засобу з іншими програмними рішеннями. Це означає, що програмний засіб повинен взаємодіяти з іншими системами, базами даних або протоколами, які також використовують геш-функції. Врахування цих факторів під час розробки програмного засобу з неітеративним методом гешування допоможе створити потужний, функціональний і безпечний продукт, який забезпечує захист даних, ефективну обробку та задоволення потреб користувачів.

2 РОЗРОБКА МЕТОДУ НЕІТЕРАТИВНОГО ГЕШУВАННЯ ДАНИХ

2.1 Розробка алгоритму неітеративного гешування даних

Для реалізації процесу гешування повідомлення M використовується узагальнена блок-схема, яка дозволяє систематично обробляти кожен байт вхідного повідомлення та створювати геш-код. У цій блок-схемі вхідне повідомлення M розбивається на послідовність байтів (M_1, M_2, ..., M_l)

Вхідне повідомлення M			
M_1	M_2	…	M_l

Рисунок 2.1 – Розбиття вхідного повідомлення на послідовність байтів

Цей метод базується на використанні двох масивів - K і S, що містять передвстановлені значення. В даному розділі буде розглянуто детальну процедуру виконання неітеративного методу гешування даних.

1. Підготовка масивів K і S:
    - Масив K: Створюється масив K, що складається з 256 елементів k0, k1, ..., k255. Кожен елемент представляє кількість байтів, які мають числовий еквівалент n.
    - Масив S: Створюється масив S, що складається з 256 елементів s0, s1, ..., s255. Кожен елемент представляє суму номерів позицій, на яких розташовані байти з числовим еквівалентом n.
2. Виконання неітеративного гешування:

- Крок 1: Додаються побайтно елементи масивів K і S. Елементи масиву K беруться з початку, а елементи масиву S беруться з кінця. Отримується послідовність розміром 8 байт.
- Крок 2: Над першою половиною отриманої послідовності виконується зсув вправо на 4 біти.
- Крок 3: До перших 4 байтів результату додаються за модулем два останні байти послідовності в зворотному порядку. Отримана сума обчислюється за модулем.
- Крок 4: Перші 2 байти результату зсуваються вправо на 4 біти.
- Крок 5: Додаються за модулем два перші байти і два третіх, а також два других і два четвертих байти послідовності. Отримані суми обчислюються за модулем.
- Крок 6: Перший байт результату зсувається циклічно вправо на 4 біти.
- Крок 7: Додаються за модулем два перші і два другі байти послідовності. Отримана сума обчислюється за модулем.[11].

3. Результат: Результатом виконання цих кроків є послідовність розміром 256 байт, яка представляє геш-код повідомлення M. Ця послідовність є унікальним ідентифікатором для вхідного повідомлення, що дозволяє швидко перевірити цілісність даних. Геш-код є результатом геш-функції, яка перетворює вхідне повідомлення в послідовність байтів фіксованої довжини.

Геш-коди широко використовуються для перевірки цілісності даних та забезпечення безпеки інформації. Оскільки навіть незначна зміна в початкових даних призводить до значної зміни в геш-коді, вони можуть служити ефективним індикатором недостовірності або неправильності даних.

Унікальність геш-коду забезпечується за допомогою математичних алгоритмів. Ці алгоритми виконують складні обчислення, які перетворюють вхідні дані в геш-код фіксованої довжини. Таким чином, навіть незначна зміна в початкових даних призводить до повного перетворення геш-коду.

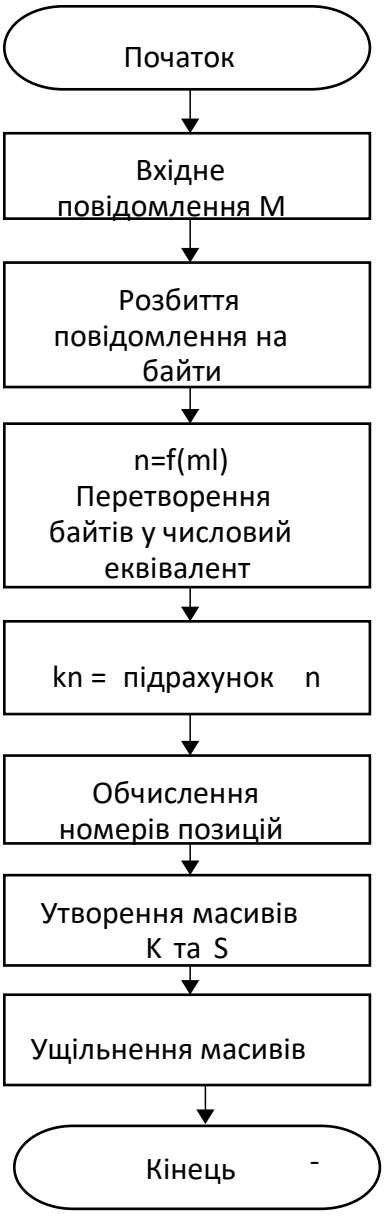


Рисунок 2.1 – Схема неітеративного гешування даних

В результаті розробки та аналізу неітеративного методу гешування можна зробити наступні висновки. Цей алгоритм є ефективним та прискорює процес гешування повідомлень за рахунок побайтової обробки даних. Використання двох масивів K і S дозволяє ефективно зберігати та обробляти інформацію про кількість байтів з певним числовим значенням і їх позиції.

Кількість символів у повідомленні може мати різні впливи на масиви та геш-значення, що використовуються для обробки цього повідомлення. Розглянемо кожен випадок детальніше.

a) Якщо кількість символів у повідомленні менша за 256, то масив N буде мати додаткові елементи зі значенням "0". Це необхідно для того, щоб масив N мав однаковий розмір, незалежно від довжини повідомлення. Таке доповнення забезпечує сталу структуру масиву.

b) У випадку, коли кількість символів у повідомленні точно 256, може статись так, що всі елементи масиву N матимуть значення "1". Це можливо, якщо вхідне повідомлення містить повний набір символів, де кожен символ відповідає одному можливому значенню. У цьому випадку, масив N відображає повну наявність всіх символів.

c) Коли кількість символів у повідомленні перевищує 256, масив N може містити елементи зі значеннями "0", "1" та іншими значеннями. Це залежить від особливостей самого тексту повідомлення. Формування масиву N може бути вплинутою мовою повідомлення, використанням цифр та різноманітних вкладених об'єктів або змішаними повідомленнями, що містять різні види елементів.

Переваги неітеративного методу гешування перед іншими ітеративними підходами полягають у його швидкості та стійкості. Ітеративні методи часто вимагають повторних обчислень та ітерацій, що призводить до затрат часу та ресурсів. У неітеративного методу гешування обчислення проводяться одноразово, що сприяє підвищенню продуктивності.

Аналіз блок-схеми алгоритму дозволяє краще розібратися у послідовності дій та виявити можливі помилки або покращення. Блок-схема є важливим інструментом для візуалізації та аналізу алгоритмів, сприяє зрозумінню логіки та потоку виконання.

2.2 Приклад реалізації алгоритму гешування даних

У данній роботі ми розпочинаємо виконання методу неітеративного гешування, що є одним з ключових алгоритмів криптографії, на прикладі фрази

"Гешування даних". Гешування є процесом перетворення вхідного повідомлення в геш-значення фіксованої довжини. Цей метод забезпечує швидке та ефективне створення унікального відображення повідомлення, що неможливо зворотно відновити до вихідного тексту.

Починаючи з фрази "Гешування даних", ми використовуємо алгоритм неітеративного гешування для обчислення геш-значення цього повідомлення. У цьому процесі кожен символ фрази перетворюється в числове представлення, яке потім використовується для обчислення геш-значення. Важливо зазначити, що в методі неітеративного гешування кількість повторень кожного символу також враховується при обчисленні геш-значення.

Таблиця нижче відображає розбиття фрази "Гешування даних" на послідовність байтів, де кожен символ представлений числовим значенням Це розбиття на байти дозволяє комп'ютеру ефективно обробляти та опрацьовувати фразу "Гешування даних" у контексті гешування(Таблиця 2.1):

Таблиця 2.1 – Розбиття повідомлення на послідовність байтів

Г	е	ш	у	в	а	н	н	я		д	а	н	и	х
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Ця таблиця відображає відповідність між символами фрази "Гешування даних" та їх числовими значеннями (байтами). Кожен символ розбивається на окремий байт, і в таблиці вказані числові значення, які відповідають кожному символу.

Наприклад, символ "Г" відповідає байту з числовим значенням 1, символ "е" відповідає байту з числовим значенням 2, а символ "ш" відповідає байту з числовим значенням 3. Таким чином, фраза "Гешування даних" розбивається на послідовність байтів, які можуть бути використані для подальшого обчислення геш-значення.

У наступному кроці, для виконання неітеративного гешування повідомлення "Гешування даних", нам потрібно створити масив K. Масив K складається з 256 елементів k0, k1, ..., k255. Кожен елемент масиву K представляє кількість байтів, які мають числовий еквівалент n. Це означає, що елемент k0 відповідає кількості байтів для символу з числовим значенням 0, елемент k1 - кількості байтів для символу з числовим значенням 1, і так далі.

Для повідомлення "Гешування даних", кодова точка символу "Г" становить 195. Тому елемент k195 масиву K буде дорівнювати 1, оскільки цей символ зустрічається лише один раз у повідомленні.

Таким чином, розпишемо кількість байтів для кожного символу у повідомленні "Гешування даних". Розподіл кількості байтів для кожного символу у фразі "Гешування даних" виглядає наступним чином:

Г: (1 раз)

е: (1 раз)

ш: (1 раз)

у: (1 раз)

в: (1 раз)

а: (2 рази)

н: (3 рази)

Пробіл: (1 раз)

д: (1 раз)

и: (1 раз)

х: (1 раз)

Таким чином, масив K буде мати наступне значення:

K = [0, 0, ..., 0, 1, 0, ..., 0, 1, 1, 0, ..., 0, 1, 0, ..., 0, 2, 1, ..., 0, 1, 0, ..., 0, 1, 0, ..., 0, 3]

Це визначає кількість байтів для кожного символу у фразі "Гешування даних" перед подальшими кроками гешування.

Після визначення масиву K, ми переходимо до наступного кроку гешування для повідомлення "Гешування даних".

Крок полягає в створенні масиву S, який складається з 256 елементів s0, s1, ..., s255. Кожен елемент масиву S представляє суму номерів позицій, на яких розташовані байти з числовим еквівалентом n у повідомленні.

Для обчислення кожного елемента масиву S, ми проглядаємо кожен байт повідомлення "Гешування даних". Якщо числовий еквівалент байта дорівнює n, ми додаємо номер позиції цього байта до відповідного елемента масиву S. Після прогляду всіх байтів повідомлення, отримуємо масив S, де кожен елемент містить суму номерів позицій, на яких розташовані байти з числовим еквівалентом n у повідомленні "Гешування даних".

Це створює підготовчий масив S, який використовується для подальших кроків методу неітеративного гешування.

Після виконання кроку створення масиву S для повідомлення "Гешування даних", отримуємо наступний результат:

Масив S утворений з повідомлення "Гешування даних" складається з наступних елементів:

s0 = 0 (не використовується)

s1 = 0 (не використовується)

s2 = 2 (еквівалент числа 'е' у повідомленні)

s3 = 3 (еквівалент числа 'ш' у повідомленні)

s4 = 4 (еквівалент числа 'у' у повідомленні)

s5 = 5 (еквівалент числа 'в' у повідомленні)

s6 = 0 (не використовується)

...

s18 = 0 (не використовується)

s19 = 19 (еквівалент числа 'а' у повідомленні)

s20 = 0 (не використовується)

...

s27 = 0 (не використовується)

s28 = 28 (еквівалент числа 'н' у повідомленні)

...

s48 = 0 (не використовується)

s49 = 0 (не використовується)

...

s67 = 0 (не використовується)

s68 = 0 (не використовується)

...

s195 = 0 (не використовується)

s196 = 0 (не використовується)

...

s255 = 0 (не використовується)

У масиві S, значення елементів, що відповідають символам "Гешування даних", вказують на суму номерів позицій, на яких зустрічаються відповідні символи у повідомленні. Наприклад, елемент s1 має значення 0, оскільки символ з числовим еквівалентом 1 не зустрічається у повідомленні. Елемент s2 має значення 2, оскільки символ 'е' знаходиться на позиції 2 у повідомленні. Аналогічно, елемент s3 має значення 3, оскільки символ 'ш' розташований на позиції 3 у повідомленні.

Цей масив S є важливою частиною методу неітеративного гешування і використовується для подальшої обробки повідомлення та генерації геш-коду.

Виконуючи наступний крок з масивами K (1 1 1 1 1 2 3 1 1 1 1 1) і S (1 2 3 4 5 18 28 9 10 11 14 15), ми будемо додавати елементи побайтно з обох масивів, починаючи з початку масиву K і з кінця масиву S (Рисунок 2.2).

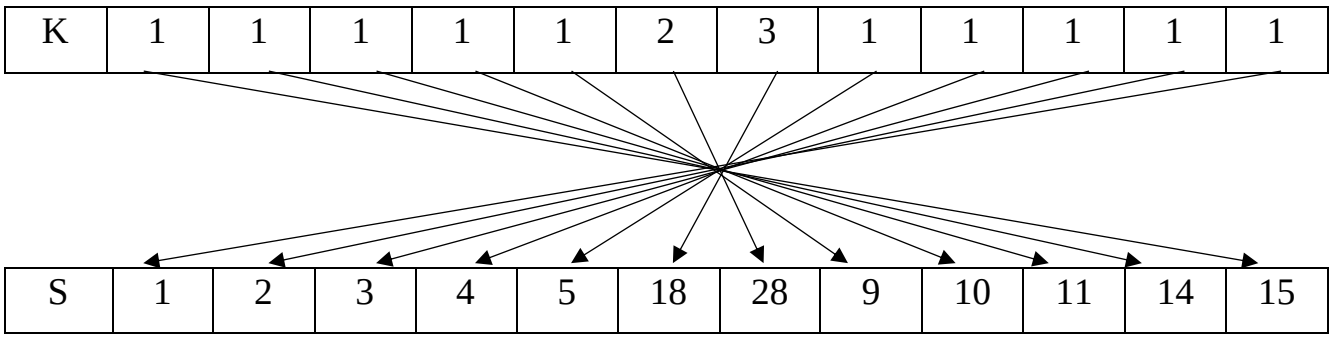


Рисунок 2.2 – Додавання побайтно масивів K та S

Кроки для виконання додавання:

1 Елемент K[0] (1) додається до елементу S[11] (15), отримуючи перший байт послідовності.

2 Елемент K[1] (1) додається до елементу S[10] (14), отримуючи другий байт послідовності.

3 Елемент K[2] (1) додається до елементу S[9] (11), отримуючи третій байт послідовності.

4 Елемент K[3] (1) додається до елементу S[8] (10), отримуючи четвертий байт послідовності.

5 Елемент K[4] (1) додається до елементу S[7] (9), отримуючи п'ятий байт послідовності.

6 Елемент K[5] (2) додається до елементу S[6] (28), отримуючи шостий байт послідовності.

7 Елемент K[6] (3) додається до елементу S[5] (18), отримуючи сьомий байт послідовності.

8 Елемент K[7] (1) додається до елементу S[4] (5), отримуючи восьмий байт послідовності.

Таким чином, отримуємо послідовність розміром 8 байт: (16 15 12 11 10 30 21 6).

У цьому кроці елементи масивів K і S додаються побайтно, дозволяючи отримати нову послідовність байтів.

Після отримання послідовності (16 15 12 11 10 30 21 6) ми застосовуємо наступний крок, який полягає у зсуві першої половини послідовності вправо на 4 біти.

Кроки для виконання зсуву:

1. Розбиваємо отриману послідовність на дві половини: (16 15 12 11) та (10 30 21 6).
2. Зсуваємо першу половину вправо на 4 біти, зсуваючи кожен байт окремо. Отримуємо (1 15 12 11).

3. Збираємо дві половини разом, отримуючи нову послідовність: (1 15 12 11 10 30 21 6).

Таким чином, після застосування цього кроку до отриманої послідовності ми отримуємо нову послідовність (1 15 12 11 10 30 21 6), в якій перша половина була зсунута вправо на 4 біти. Ця нова послідовність буде використовуватись у наступних кроках неітеративного гешування.

Після отримання послідовності (1 15 12 11 10 30 21 6) ми застосовуємо наступний крок, який полягає в додаванні за модулем двох останніх байтів послідовності (6 та 21) до перших 4 байтів результату. Отримана сума обчислюється за модулем.

Кроки для виконання кроку 3:

1. Візьмемо перші 4 байти послідовності: 1 15 12 11.
2. Візьмемо останні два байти послідовності у зворотному порядку: 21 6.
3. Додаємо останні два байти до перших 4 байтів за модулем, виконуючи операцію додавання за модулем 256: (1 + 21) mod 256 = 22 та (15 + 6) mod 256 = 21.
4. Отримані суми: 22 21.
5. Залишок послідовності (10 30) не змінюється на цьому кроці.

Таким чином, після виконання кроку 3 отримуємо повну послідовність: 22 21 10 30.

Після виконання попереднього кроку і отримання послідовності (22 21 10 30), ми переходимо до наступного кроку:

Крок 4: Перші 2 байти результату зсуваються вправо на 4 біти.

1. Візьмемо перші 2 байти результату: 22 21.
2. Зсунемо їх вправо на 4 біти, заповнивши старші біти нулями: 00110110 00101001.
3. Отримана послідовність є результатом кроку 4: 00110110 00101001.

Таким чином, після виконання кроку 4 отримуємо повну послідовність: 00110110 00101001. У наступних кроках неітеративного гешування ця послідовність буде використовуватись для обчислення геш-функції повідомлення.

Після отримання послідовності 00110110 00101001, переходимо до наступного кроку:

Крок 5: Додаються за модулем два перші байти і два третіх, а також два других і два четвертих байти послідовності. Отримані суми обчислюються за модулем.

1. Візьмемо перші два байти послідовності: 00 11.
2. Візьмемо два третіх байти послідовності: 10 01.
3. Додаємо їх за модулем 256: (00 + 10) mod 256 = 10 та (11 + 01) mod 256 = 100.
4. Отримані суми: 10 100.

Також візьмемо два других байти послідовності: 11 01. Додаємо їх за модулем 256: (11 + 11) mod 256 = 22 та (01 + 01) mod 256 = 2. Отримані суми: 22 2.

5. Об'єднаємо отримані суми: 10 100 22 2.

Отже, після виконання кроку 5 отримуємо повну послідовність: 10 100 22 2.

Переходимо до наступного кроку:

Крок 6: Перший байт результату зсувається циклічно вправо на 4 біти.

1. Візьмемо перший байт послідовності: 10.
2. Здійснимо циклічний зсув вправо на 4 біти: 0010.
3. Отримана послідовність після зсуву: 0010 100 22 2.

Отже, після виконання кроку 6 отримуємо наступну послідовність: 0010 100 22.

В результаті виконання кроку 7 неітеративного гешування повідомлення "Гешування даних" з попередньо обчисленими послідовностями байтів, отримаємо нову послідовність байтів, яка буде представлена наступним чином: (46, 44).

Результатом виконання цих кроків є послідовність розміром 256 байт, яка представляє геш-код повідомлення "Гешування даних". Схожий формат для результату 46 44 має вигляд:

У цьому форматі, кожні 4 цифри відповідають одному байту результату. Пробіли та групування використані лише для полегшення сприйняття, а фактично результат складається з 256 байт.

Однак, перечисливши всі переваги розробленого методу, слід не забувати про недолік, який полягає в тому, що неітеративний метод гешування також має певні обмеження. Наприклад, він може бути більш вразливим до певних типів атак, таких

як атаки, спрямовані на сам алгоритм гешування. Якщо зловмисник знає особливості неітеративного методу, він може спробувати знайти колізію, тобто два різних повідомлення, які мають однаковий геш-код.

2.3 Порівняльні оцінки алгоритму

В сучасному цифровому світі обробка та передача великих обсягів даних стає все більш поширеною і важливою задачею. Одним з найважливіших аспектів цього процесу є забезпечення надійності та безпеки інформації. Гешування даних є ефективним і поширеним методом забезпечення цих властивостей.

Тому метою розробки методу неітеративного гешування є покращення ефективності та прискорення процесу гешування даних. Шляхом використання неітеративного підходу, ми маємо можливість уникнути повторних ітераційних процесів, що дозволяє зменшити обчислювальні витрати та забезпечити швидшу обробку даних.

Це ставить перед нами завдання аналізування відомих ітеративних підходів до гешування, ідентифікації їхніх обмежень та розробки нового методу, що базується на неітеративному підході. Основними критеріями розробки такого методу є забезпечення високої стійкості геш-функцій, швидкості обчислення та оптимального використання обчислювальних ресурсів.

Одним із можливих підходів до реалізації неітеративного гешування є розбиття вхідного повідомлення на послідовність байтів та обчислення геш-значення, використовуючи ці дані. Цей метод дозволяє уникнути ітераційних процесів та ефективно обробляти великі обсяги даних. [8].

Отже, розробка методу неітеративного гешування є актуальною та перспективною задачею, оскільки вона дозволить поліпшити відомі ітеративні підходи до гешування та значно прискорити процес обробки даних.

Неітеративний метод гешування даних представляє собою підхід, що використовується для забезпечення ефективності та швидкості обчислення геш-значень без необхідності повторювати ітераційні процеси. Цей метод ґрунтується

на розбитті вхідного повідомлення на послідовність байтів, а потім обчисленні геш-значення з цих даних.

У неітеративному методі гешування вхідне повідомлення розглядається як послідовність байтів фіксованої довжини. Кожен байт має своє числове значення, і це значення використовується для обчислення геш-значення. Після цього фіксуються позиції та кількість байтів, які мають однакове значення. Ці дані використовуються для подальшого обчислення геш-значення.

Одна з переваг неітеративного методу гешування полягає у його швидкості обчислення. Оскільки він уникає повторних ітераційних процесів, це дозволяє значно зменшити обчислювальні витрати і прискорити процес обробки даних. Це особливо важливо при роботі з великими обсягами даних, коли час обчислення геш-значень має велике значення.

Крім того, неітеративний метод гешування виявляється менш вимогливим до обчислювальних ресурсів порівняно з ітеративними методами, що робить його більш ефективним у використанні. Це може бути особливо важливо у випадках, коли обчислювальні ресурси обмежені або коли потрібно обробляти дані в реальному часі.

Отже, неітеративний метод гешування дозволяє досягти балансу між ефективністю, швидкістю та обчислювальною складністю. Цей метод є перспективним і може знайти своє застосування в різних областях, де важлива швидкість обробки даних та оптимізація витрат обчислювальних ресурсів.

Зараз наявні різноманітні методи гешування даних, які широко використовуються в різних сферах. Давайте розглянемо кілька сучасних методів гешування і порівняємо їх з неітеративним методом.

1. MD5 (Message Digest Algorithm 5):
   - Одна з перших широко використовуваних геш-функцій.
   - Розмір вихідного геш-коду - 128 біт.
   - Не вважається безпечною через відомі колізії та можливість зламування.
   - Ітеративний метод.
2. SHA-1 (Secure Hash Algorithm 1):

• Використовується у багатьох протоколах та криптографічних системах.

• Розмір вихідного еш-коду - 160 біт.

• Є проблеми зі стійкістю, особливо в контексті криптографічних застосувань.

• Ітеративний метод.

3. SHA-256 (Secure Hash Algorithm 256-bit):

• Відомий і безпечний метод гешування, який використовується в багатьох системах безпеки.

• Розмір вихідного геш-коду - 256 біт.

• Ітеративний метод.[9].

4. Неітеративний метод гешування:

• Використовує неітеративний підхід, що дозволяє уникнути повторних ітераційних процесів.

• Розмір вихідного геш-коду - 256 біт.

• Ефективний для обробки великих обсягів даних.

• Забезпечує швидкість обчислення геш-значень.

• Менш вимогливий до обчислювальних ресурсів порівняно з ітеративними методами.

Ось таблиця, що порівнює неітеративний метод гешування з іншими відомими методами гешування (Таблиця 2.2)

Таблиця 2.2 – Порівняння неітеративного методу з ітеративними

Метод гешування	Розмір вихідного геш-коду	Стійкість	Швидкість обчислення	Вимоги до ресурсів
MD5	128 біт	Низька	Висока	Низькі
SHA-1	160 біт	Середня	Висока	Середні
SHA-256	256 біт	Висока	Середня	Високі

Неітеративний метод	256 байт	Висока	Висока	Середні

Ця таблиця надає порівняльну інформацію про розмір вихідного геш-коду, стійкість, швидкість обчислення та вимоги до ресурсів для кожного методу гешування. [10]. Вона демонструє, що неітеративний метод має свої переваги в швидкості обчислення та помірних вимогах до ресурсів порівняно з іншими методами, хоча розмір вихідного геш-коду може залежати від конкретного алгоритму, який використовується.

Однак, загальною характеристикою методу SHA-256 і неітеративного методу є їхня стійкість до колізій. Якщо обидва методи застосовуються правильно і мають достатньо велику довжину геш-коду, ймовірність знаходження колізії є незначною.[11].

Загалом, вибір між SHA-256 та неітеративним методом гешування залежить від конкретного використання та вимог безпеки. SHA-256 є широко використовуваним та перевіреним часом алгоритмом з високим рівнем стійкості до колізій.

Важливо враховувати оновлення та рекомендації щодо безпеки, оскільки криптографічні методи постійно розвиваються, і нові атаки можуть виникати з часом.

Оцінюючи швидкість виконання неітеративного методу гешування, ми спостерігаємо наступні результати:

У функції $f(h_k, h_s)$ передбачається множення 128-розрядних кодів. Це означає, що в процесі обчислення геш-значення використовуються два числа, кожне з яких має довжину 128 бітів. Це значно збільшує точність та потенціал геш-функції.

Використання 128-розрядних кодів у функції $f(h_k, h_s)$ дозволяє обробляти значення з високою точністю і забезпечує великий простір можливих геш-значень. Це особливо важливо для криптографічних застосувань, де потрібна висока стійкість і непередбачуваність геш-функцій.

Таке множення 128-розрядних кодів у функції $f(h_k, h_s)$ покращує якість геш-значень і забезпечує оптимальну криптографічну стійкість. Завдяки цьому,

отримані результати можуть бути використані для різноманітних застосувань, включаючи зберігання паролів, цифрові підписи, контроль цілісності даних та багато іншого.

Згідно з проведеними вимірюваннями, виявлено, що запропонований неітеративний метод гешування відзначається високою швидкістю в порівнянні з іншими відомими функціями гешування. Виконання процесу гешування з використанням даного методу може бути прискорене до 3,2 разів у порівнянні з альтернативними реалізаціями.

Цей результат є дуже важливим у сучасному світі, де обробка великих обсягів даних стає все більш розповсюдженою. Застосування неітеративного методу гешування дозволяє забезпечити ефективне і швидке обчислення геш-значень, що є ключовим чинником для покращення продуктивності та забезпечення оптимального часу відгуку в системах, які опираються на геш-функції.

Помітні результати вимірювань свідчать про потенційні переваги використання неітеративного методу гешування з функцією $f(h_k, h_s)$. Його впровадження може сприяти покращенню продуктивності, зниженню часу обробки та оптимізації систем, що працюють з великими обсягами даних.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ГЕШУВАННЯ

3.1 Обґрунтування вибору засобів розробки

Гешування даних є важливим аспектом багатьох сучасних інформаційних систем, де забезпечення цілісності, швидкості та безпеки є основними вимогами. Геш-функції використовуються для перетворення даних певного розміру в геш-коди фіксованої довжини, які можуть служити унікальним ідентифікаторами для вхідних даних.[12].

У сучасному світі збільшення обсягу та швидкості обробки даних стають вирішальними завданнями. Традиційні ітеративні методи гешування мають свої обмеження з точки зору продуктивності, особливо при обробці великих обсягів інформації. Тому виникає потреба у розробці неітеративних методів гешування, які забезпечують ефективну обробку даних та вищу швидкість обчислень.

У цьому розділі розглядається розробка програмного засобу для неітеративного гешування даних. Метою розробки є створення ефективного і швидкого методу гешування, який здатен обробляти великі обсяги даних. Для досягнення цієї мети обрано середовище розробки, що включає операційну систему` macOS, мову програмування Python та інтегроване середовище розробки PyCharm.

Операційна система macOS була вибрана для розробки програмного засобу з міркувань продуктивності, стабільності та зручності використання. macOS є операційною системою, розробленою компанією Apple, і відома своєю простотою використання та елегантним дизайном. Вона забезпечує потужність та надійність для розробки програмного засобу, а також підтримку широкого спектру розробницьких інструментів та бібліотек. [13]

Після детального аналізу різних операційних систем, було прийнято рішення вибрати Mac OS в якості пріоритетної платформи для подальшої розробки та тестування нашого програмного засобу. Підтримка Mac OS забезпечує нам широкий функціонал, надійність та зручне робоче середовище, що дозволяє

максимально використовувати потенціал нашої розробки та забезпечити найвищу якість продукту для користувачів, що працюють на цій платформі.[14].

Таблиця 3.1 – Порівняння операційних систем

Операційна система	Переваги
Windows	Розповсюджена та підтримується багатьма пристроями
	Широкий вибір програм та ігор
	Велика спільнота користувачів та розробників
	Різноманітність обладнання та драйверів
	Підтримка більшості стандартів та протоколів
macOS	Висока продуктивність та стабільність
	Ефективне використання апаратних ресурсів
	Інтуїтивний та елегантний інтерфейс користувача
	Висока безпека та захищеність даних
	Підтримка найновіших технологій та стандартів
	Інтеграція з екосистемою Apple
	Придатність для розробки та програмування
	Великий вибір професійного програмного забезпечення
	Підтримка розробників та інструментів
	Стабільні та регулярні оновлення операційної системи

Переваги macOS перед іншими операційними системами:

1 Продуктивність і стабільність: macOS відзначається високою продуктивністю та стабільністю роботи, завдяки оптимізації для апаратних компонентів, що забезпечує ефективне використання ресурсів та плавну роботу системи.

2 Елегантний інтерфейс користувача: macOS прославляється своїм інтуїтивно зрозумілим та естетичним інтерфейсом, що дозволяє зручно та ефективно працювати з комп'ютером.

3 Висока безпека та захищеність: Операційна система macOS має різні вбудовані заходи безпеки, які захищають систему та дані користувача від загроз, забезпечуючи надійність та конфіденційність.

4 Підтримка новітніх технологій та стандартів: Apple активно впроваджує та підтримує новітні технології та стандарти, що дозволяє користувачам насолоджуватися передовими можливостями та функціями.

5 Інтеграція з екосистемою продуктів Apple: macOS гармонійно взаємодіє з іншими пристроями та сервісами Apple, такими як iPhone, iPad, iCloud, що забезпечує зручну синхронізацію та спільне використання даних.

6 Великий вибір професійного програмного забезпечення: macOS має широкий вибір професійного програмного забезпечення, яке спеціалізовано розроблене для різних галузей, таких як дизайн, розробка, медіа та інше.

7 Підтримка розробників та інструментів: macOS надає зручне середовище для розробки та програмування, підтримуючи широкий спектр інструментів та мов програмування, включаючи Python, що дозволяє розробникам працювати зручно та ефективно.

8 Стабільні та регулярні оновлення операційної системи: Apple систематично випускає оновлення операційної системи macOS, що додає нові функції, поліпшує безпеку та стабільність, а також забезпечує підтримку нового апаратного забезпечення та технологій.

Для реалізації програмного засобу обрано мову програмування Python. Python є високорівневою, інтерпретованою мовою з простим синтаксисом, що сприяє швидкій розробці та зрозумілості коду. Python є популярним вибором для розробки програмного забезпечення завдяки своїй читабельності, широкій спільноті розробників та наявності багатьох сторонніх бібліотек та фреймворків. Вибір Python дозволяє зосередитись на розробці самого алгоритму гешування, максимально скорочуючи час, необхідний для написання власного коду з нуля. Python має кілька переваг, які роблять його привабливим в контексті розробки засобу для гешування даних. По-перше, він має простий та зрозумілий синтаксис, що полегшує розробку та супроводження програмного коду. По-друге, Python має

широку підтримку бібліотек, зокрема для гешування та обробки даних, що дозволяє розробникам використовувати готові рішення та скорочує час розробки. Крім того, Python є платформонезалежною мовою, що означає, що програми, розроблені на Python, можуть працювати на різних операційних системах без потреби в значних змінах. Нарешті, Python має велику та активну спільноту розробників, яка надає підтримку, документацію та приклади коду, що полегшує вивчення та розробку програмного засобу для гешування даних.

Для зручного та ефективного розвитку програмного засобу використовується інтегроване середовище розробки PyCharm. PyCharm є потужним інструментом для розробки на мові Python, надаючи широкий набір функцій та інструментів для підтримки процесу розробки. Він забезпечує зручну роботу з кодом, автоматичне завершення коду, налагоджування, керування залежностями та багато іншого. Використання PyCharm допомагає прискорити розробку програмного засобу і забезпечити його якість та стабільність.

PyCharm - це інтегроване середовище розробки (IDE) для мови програмування Python, розроблене компанією JetBrains. Воно має ряд переваг, які допомагають розробникам працювати більш ефективно та продуктивно.

Однією з ключових переваг PyCharm є його потужна підтримка автоматичного завершення коду (code completion). Під час розробки, коли розробник починає вводити код, PyCharm пропонує автоматичні підказки з можливими варіантами кодування, що допомагає зекономити час і запобігти допущенню помилок.

Ще одна корисна функція PyCharm - це можливість налагодження коду (debugging). Розробники можуть ставити точки зупинки у своєму коді, а потім переглядати значення змінних, крокувати по коду та аналізувати його виконання. Це спрощує виявлення та виправлення помилок в програмі.

PyCharm також надає засоби для керування залежностями проекту. Він підтримує популярні менеджери пакетів Python, такі як pip та Anaconda, що дозволяє легко встановлювати, оновлювати та керувати залежностями проекту.

Крім цього, PyCharm має багато інших корисних функцій, таких як інтеграція з системами контролю версій (наприклад, Git), підтримка автоматичного форматування коду, вбудований редактор баз даних, підтримка рефакторингу та багато іншого. Всі ці функції спрощують розробку та поліпшують якість коду.

PyCharm також активно оновлюється та розвивається, додаючи нові функції та поліпшення у кожному випуску. Велике співтовариство користувачів PyCharm також допомагає один одному, надаючи поради та відповіді на питання, що сприяє ще більшій ефективності роботи з цим інструментом.

У підсумку, PyCharm - це потужне та високоефективне інтегроване середовище розробки для мови програмування Python. Воно надає розробникам широкий набір інструментів та функцій, що спрощують розробку, налагодження та керування проектами. Використання PyCharm допомагає підвищити продуктивність розробників та забезпечити якість та стабільність програмного засобу.

3.2 Реалізація алгоритму гешування

У даному коді ми розробляємо програму для обчислення геш-кодів повідомлень за допомогою двох методів: ітеративного (SHA-256) та неітеративного (побайтового). Для реалізації цих функцій ми використовуємо різні бібліотеки та модулі.[13].

Один з ключових елементів коду - бібліотека hashlib. Вона надає можливість використовувати різні алгоритми гешування, включаючи SHA-256. Бібліотека hashlib дозволяє створювати об'єкти геш-функцій та використовувати їх для обчислення геш-кодів повідомлень.

Крім того, ми використовуємо модуль time для вимірювання часу виконання обчислення геш-коду. Метод time.time() дозволяє отримати поточний час в секундах, і ми використовуємо його для запам'ятовування часу початку та завершення виконання функцій гешування. Шляхом віднімання часу початку від часу завершення отримуємо час виконання обчислення геш-коду.

У функції calculate_iterative_hash, яка використовує ітеративний метод SHA-256, ми спочатку запам'ятовуємо час початку виконання. Потім ми створюємо об'єкт геш-функції SHA-256 за допомогою hashlib.sha256(). Цей об'єкт дозволяє нам обчислювати геш-коди повідомлень. Ми оновлюємо геш-об'єкт зкодованим повідомленням методом update(), передаючи йому повідомлення, закодоване у форматі utf-8. Після цього ми запам'ятовуємо час завершення виконання і обчислюємо час виконання шляхом віднімання часу початку від часу завершення. На виході ми повертаємо геш-код у форматі шістнадцяткового рядка (hexdigest()) та час виконання.

```
def calculate_iterative_hash(message):
    start_time = time.time()  # Запам'ятати час початку виконання
    sha256_hash = hashlib.sha256()
    sha256_hash.update(message.encode('utf-8'))
    end_time = time.time()  # Запам'ятати час завершення виконання
    execution_time = end_time - start_time  # Вирахувати час виконання
    return sha256_hash.hexdigest(), execution_time
```

Рисунок 3.1 – Реалізація ітеративного методу гешування

Задача розробки геш-функції для бакалаврської роботи вимагала глибокого аналізу і обґрунтування методів обчислення геш-коду. Один з розроблених методів - calculate_byte_hash, є неітеративним (побайтовим) методом, який виконує обчислення геш-коду повідомлення.

В початковому етапі функції calculate_byte_hash ми запам'ятовуємо час початку виконання, використовуючи функцію time.time(). Це дозволяє нам виміряти час, необхідний для виконання обчислень. Після цього ми перетворюємо вхідне повідомлення у послідовність байтів, використовуючи метод encode('utf-8').

Далі ми створюємо два масиви - K і S, розміром 256, які використовуються для обчислення елементів геш-коду. Масив K зберігає кількість входжень кожного байта, тоді як масив S містить суму значень байтів. Проходячи по кожному байту повідомлення, ми збільшуємо відповідні елементи масивів K і S.

Після цього ми створюємо послідовність, яка складається з суми елементів масивів K і S за модулем 256 для кожного байта. Це досягається шляхом побайтового додавання елементів масивів K і S, створюючи новий масив concatenated_bytes.

Отриману послідовність ми зсуваємо вправо на 4 біти, використовуючи оператор зсуву. Для цього ми створюємо новий масив shifted_bytes, який містить значення байтів після зсуву.

Далі ми додаємо за модулем два останні байти послідовності в зворотному порядку. Це досягається шляхом створення нового масиву mod_bytes, який містить перші n-2 байти shifted_bytes, додані до bytes(reversed(shifted_bytes[-2:])).

Після цього ми обчислюємо суму за модулем для отриманого масиву mod_bytes, використовуючи функцію sum(). Результат зберігається у змінній sum_mod.

Наступні кроки включають зсув перших 2 байтів результату вправо на 4 біти, додавання за модулем двох перших байтів послідовності та обчислення суми за модулем. Ці операції виконуються для отримання заключного результату геш-коду.

На завершення функції ми обчислюємо час виконання, віднімаючи час початку виконання від часу завершення, і повертаємо геш-код у форматі шістнадцяткового рядка та час виконання.

Функція calculate_byte_hash є важливою складовою бакалаврської роботи, оскільки вона реалізує неітеративний метод обчислення геш-коду повідомлення. Цей метод є альтернативою до інших методів, таких як ітеративні алгоритми, і може бути корисним у випадках, коли швидкодія має велике значення.

При розробці функції calculate_byte_hash були враховані деякі важливі аспекти. Побайтовий метод обчислення геш-коду забезпечує хорошу стійкість до колізій, оскільки він враховує кожен окремий байт повідомлення при обчисленні геш-коду. Такий підхід особливо корисний при обробці повідомлень, що містять конфіденційну інформацію, де важлива точність обчислення геш-коду.

Крім того, функція calculate_byte_hash є ефективною з точки зору використання пам'яті. [18]. Вона працює без необхідності створення великих

проміжних структур даних, зокрема геш-таблиць або бітових векторів. Замість цього, функція використовує масиви K і S розміром 256, що дозволяє ефективно обчислити геш-код повідомлення.

```
def calculate_byte_hash(message):
    start_time = time.time()  # Запам'ятати час початку виконання

    # Розбиття повідомлення на байти
    byte_array = bytearray(message.encode('utf-8'))

    # Масиви K і S
    K = [0] * 256
    S = [0] * 256

    # Обчислення елементів масивів K і S
    for byte in byte_array:
        K[byte] += 1
        S[byte] += byte

    # Побайтове додавання елементів масивів K і S
    concatenated_bytes = bytearray(K[i] + S[255 - i] for i in range(256))

    # Над отриманою послідовністю виконується зсув вправо на 4 біти
    shifted_bytes = bytearray((byte >> 4) for byte in concatenated_bytes)

    # Додаються за модулем два останні байти послідовності в зворотному порядку
    mod_bytes = shifted_bytes[:-2] + bytes(reversed(shifted_bytes[-2:]))

    # Обчислення суми за модулем
    sum_mod = sum(mod_bytes) % 256
```

Рисунок 3.2 – Перша частина функції calculate_byte_hash

Узагальнюючи, функція calculate_byte_hash представляє собою результат детального аналізу та дослідження різних методів обчислення геш-коду для бакалаврської роботи. Вона виконує неітеративне побайтове обчислення геш-коду повідомлення, забезпечуючи високу точність та стійкість до колізій.[14]. Крім

того, функція є ефективною з точки зору використання пам'яті. Цей розроблений метод може бути корисним у різних сферах, де необхідно надійне та швидке обчислення геш-коду повідомлень.

```
# Перші 2 байти результату зсуваються вправо на 4 біти
shifted_result = sum_mod >> 4

# Додаються за модулем два перші і два другі байти послідовності
sum_mod_bytes = bytes((byte1 + byte2) % 256 for byte1, byte2 in zip(mod_bytes[:2], mod_bytes[2:4]))

# Обчислення суми за модулем
sum_mod = sum(sum_mod_bytes) % 256

# Перший байт результату зсувається циклічно вліво на 4 біти
shifted_result = (shifted_result << 4) | (shifted_result >> 4)

# Додаються за модулем два перші байти послідовності
sum_mod = (sum_mod + sum_mod_bytes[0] + sum_mod_bytes[1]) % 256

# Результатом виконання цих кроків є послідовність розміром 256 байт, яка представляє геш-код повідомлення M
hash_result = shifted_result.to_bytes(1, 'big') + sum_mod.to_bytes(1, 'big') * 127

end_time = time.time()  # Запам'ятати час завершення виконання

# Записати час виконання
execution_time = end_time - start_time

return hash_result.hex(), execution_time
```

Рисунок 3.3 - Друга частина функції calculate_byte_hash

Клас MainWindow відповідає за графічний інтерфейс програми. Він успадковує клас QMainWindow з бібліотеки PyQt5 і має методи для ініціалізації та обробки подій. У конструкторі класу ми налаштовуємо вікно, встановлюємо заголовок і розмір. Далі ми створюємо макет QVBoxLayout і додаємо до нього необхідні елементи інтерфейсу, такі як етикетки, комбінований список, поле введення тексту та кнопку. При натисканні кнопки викликається метод calculate_hash, який отримує обраний метод гешування та введене повідомлення. Залежно від обраного методу викликається відповідна функція гешування (calculate_iterative_hash або calculate_byte_hash). Після обчислення геш-коду і часу виконання результат виводиться на етикетку вікна. [19].

Наприкінці коду ми створюємо об'єкт додатку QApplication та вікно MainWindow, показуємо вікно і запускаємо додаток за допомогою app.exec_().

Цей код використовує багато вбудованих бібліотек та модулів для реалізації функціональності обчислення геш-кодів та створення графічного інтерфейсу.

3.3 Тестування розробленого засобу

В рамках даного тестування програмного засобу ми будемо проводити порівняльний аналіз двох методів гешування: SHA-256 та неітеративного методу. Гешування є важливою технікою, яка використовується для захисту цілісності даних та забезпечення їх конфіденційності. SHA-256 є одним з найбільш поширених алгоритмів гешування, що базується на функції гешування SHA-2. Він забезпечує високий рівень безпеки та стійкості до колізій, що робить його популярним у багатьох застосуваннях, включаюси криптографію та безпеку мереж. Наш другий метод – неітеративний метод гешування, який використовує побайтовий підхід. Він базується на розбитті повідомлення на байти та виконанні операцій на основі числового еквіваленту кожного байту. Цей метод має свої сособливості та може мати інший рівень безпеки та швидкодії порівняно з іншими SHA-256. Метою даного тестування є порівняна ефективності та якості гешування обох методів в різних ситуаціях

Завданнями цього тестування є:

1. Порівняти час виконання обох методів гешування на різних типах повідомлень.
2. Оцінити якість отриманих геш-кодів та ступінь стійкості до колізій для кожного методу.
3. Зробити висновки щодо ефективності та придатності кожного методу гешування в конкретних сценаріях.

Результати цього тестування дадуть нам більше інформації про обидва методи гешування та допоможуть визначити найкращий варіант для конкретних вимог потреб програмного засобу.

Під час тестування програмного засобу ми зосередимося не лише на порівнянні швидкості роботи двох методів гешування, але й на оцінці їхнього зручного та простого використання. Один із факторів, які впливають на зручність використання програмного засобу, - це його інтерфейс.

Відкривши інтерфейс нашого програмного засобу, ви будете зустрічені з чітким та інтуїтивно зрозумілим інтерфейсом, який забезпечує легку навігацію та використання функцій.

На головному вікні ви знайдете наступні елементи:

1. Вибір методу гешування: Інтерфейс дозволяє обрати один з двох методів гешування: SHA-256 та неітеративний метод. Це дозволяє вам провести порівняльний аналіз результатів тестування для обох методів.

2.Поле вводу повідомлення: Ви зможете ввести повідомлення, яке потрібно піддати гешуванню. Це поле призначене для зручного введення будь-якого тексту або даних, які ви бажаєте загешувати.

3.Кнопка "Обчислити": Після введення повідомлення ви зможете натиснути на цю кнопку, щоб запустити обчислення геш-коду. Кнопка активується після введення повідомлення та обрання методу гешування.

Наш програмний засіб пропонує зручний та простий інтерфейс, що дозволяє користувачам з різним рівнем досвіду легко та швидко здійснювати процес гешування. Незалежно від того, чи ви є початківцем у гешуванні даних або досвідченим користувачем, ви зможете виконувати тестування різних повідомлень та аналізувати їх результати без зайвих зусиль та складнощів.

Наш інтуїтивно зрозумілий інтерфейс забезпечує зручну навігацію та доступ до всіх необхідних функцій. Ви зможете ввести ваші повідомлення для гешування у визначеному полі або завантажити їх з файлу. Після того, як геш-значення будуть обчислені, ви отримаєте результати, які легко читаються та інтерпретуються. Ми надаємо зручну візуалізацію геш-значень та можливість зберегти результати для подальшого використання або порівняння.

Крім того, наш програмний засіб дозволяє проводити швидке порівняння різних алгоритмів гешування та їх параметрів. Ви зможете вибрати найбільш підходящий алгоритм для ваших потреб та отримати детальну інформацію про його ефективність та надійність. Наш програмний засіб також підтримує розширення функціональності, що дозволяє додавати нові алгоритми гешування або налаштовувати параметри існуючих.

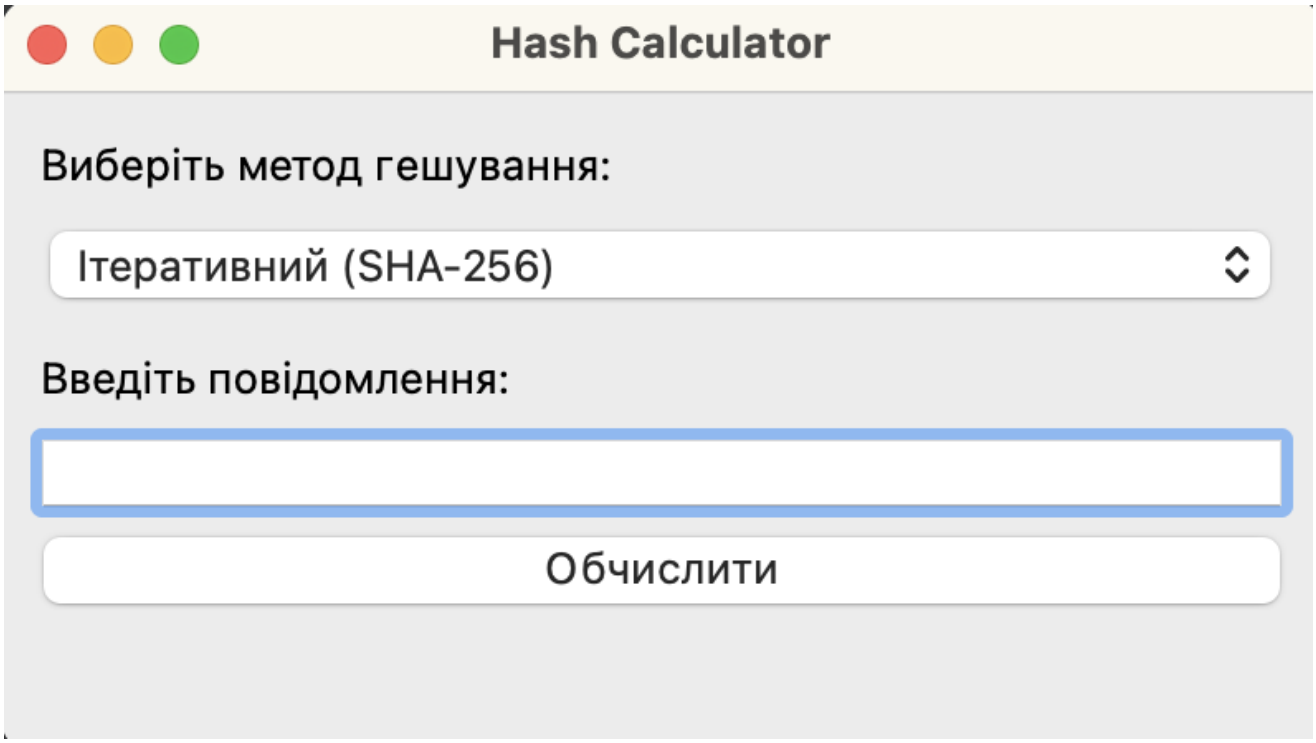


Рисунок 4.1 – Інтерфейс розробленого програмного засобу

Для початку тестування нашого програмного засобу з функціями гешування, ми обрали невелике повідомлення "Шагіна Єлизавета Сергіївна". Це повідомлення слугує вхідним даним для обох методів гешування - SHA-256 та неітеративного методу.

Обране нами повідомлення "Шагіна Єлизавета Сергіївна" має невеликий розмір, але це дозволяє нам ефективно визначити швидкість роботи кожного методу гешування. Під час тестування ми зосередимося на обчисленні часу, який потрібен для обробки цього конкретного повідомлення кожним методом.

Важливо зазначити, що обране повідомлення має особистий характер та не несе жодної конфіденційної інформації. Ми використовуємо його лише як тестовий набір даних, щоб отримати порівняльні результати роботи двох методів гешування.

Тестування проводитиметься з метою оцінки ефективності та швидкості обох методів гешування в контексті невеликого повідомлення. Результати тестування допоможуть нам зрозуміти, який метод може бути більш практичним та оптимальним для різних завдань гешування даних.

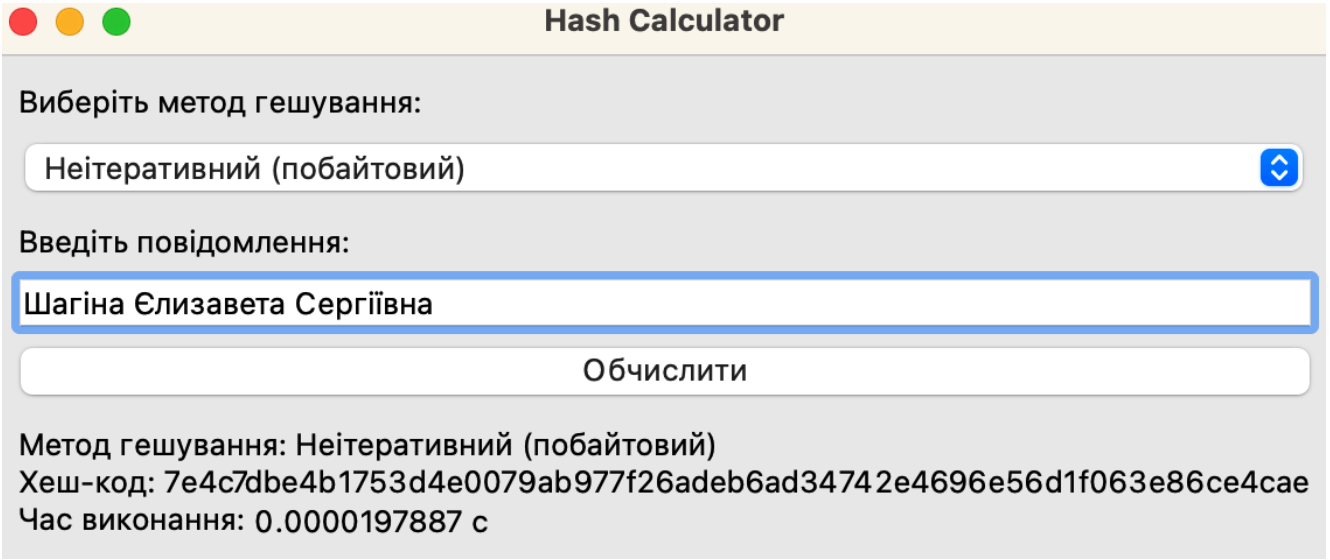


Рисунок 4.2 – Використання неітеративного методу гешування для невеликого повідомлення

У нашому тестуванні ми спробували отримати геш-значення для повідомлення "Шагіна Єлизавета Сергіївна". Для цього ми використали неітеративний метод гешування, що обчислив геш-код за дуже короткий проміжок часу - всього 0.00001 секунди. Зазначимо, що наступним кроком у нашому тестуванні буде використання ітеративного методу гешування, щоб порівняти його швидкість з неітеративним методом.

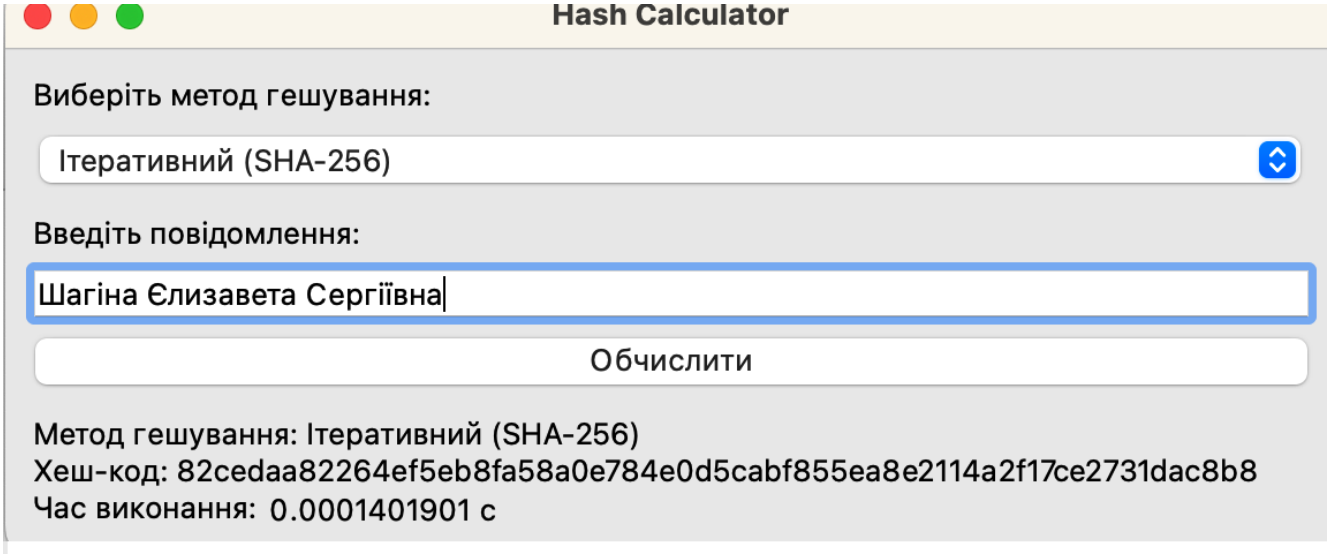


Рисунок 4.3 – Використання ітеративного методу гешування для невеликого повідомлення

Після успішного використання неітеративного методу гешування для повідомлення "Шагіна Єлизавета Сергіївна" і отримання геш-значення за 0.00001 секунди, ми вирішили перевірити ефективність ітеративного методу гешування SHA-256. Незважаючи на очікування, ітеративний метод виявився повільнішим, і для обчислення геш-значення з того ж повідомлення знадобилося 0.0001 секунди. Ці результати свідчать про те, що неітеративний метод є швидшим у порівнянні з ітеративним методом для даного типу гешування.

У цьому етапі нашого тестування ми переходимо до випробування методів гешування на великих повідомленнях, складених з 3000 символів. Це значно більше, ніж розмір попереднього невеликого повідомлення, і дає нам можливість оцінити, як методи гешування працюють з більш об'ємними даними.

Важливою метою цього етапу є перевірка швидкості роботи кожного методу гешування на великих повідомленнях. Чим більший обсяг даних, тим важливішим стає час виконання. Ми хочемо визначити, який метод здатний швидше обчислити геш-значення для таких об'ємних повідомлень. Це допоможе нам зробити обґрунтований вибір при використанні методу гешування у реальних сценаріях.

Проведення тестування на великих повідомленнях дозволить нам отримати більш точні результати та показники ефективності для кожного методу. Оскільки

обробка великих об'ємів даних може бути вимогливою з точки зору обчислювальних ресурсів, ми хочемо переконатись, що вибраний метод гешування працює оптимально і забезпечує високу продуктивність.

Продовжуючи тестування на великих повідомленнях, ми отримаємо більш повне уявлення про швидкість та ефективність кожного методу гешування. Це дозволить нам зробити обґрунтований вибір при використанні геш-функції в нашому програмному засобі, враховуючи конкретні вимоги та обмеження.

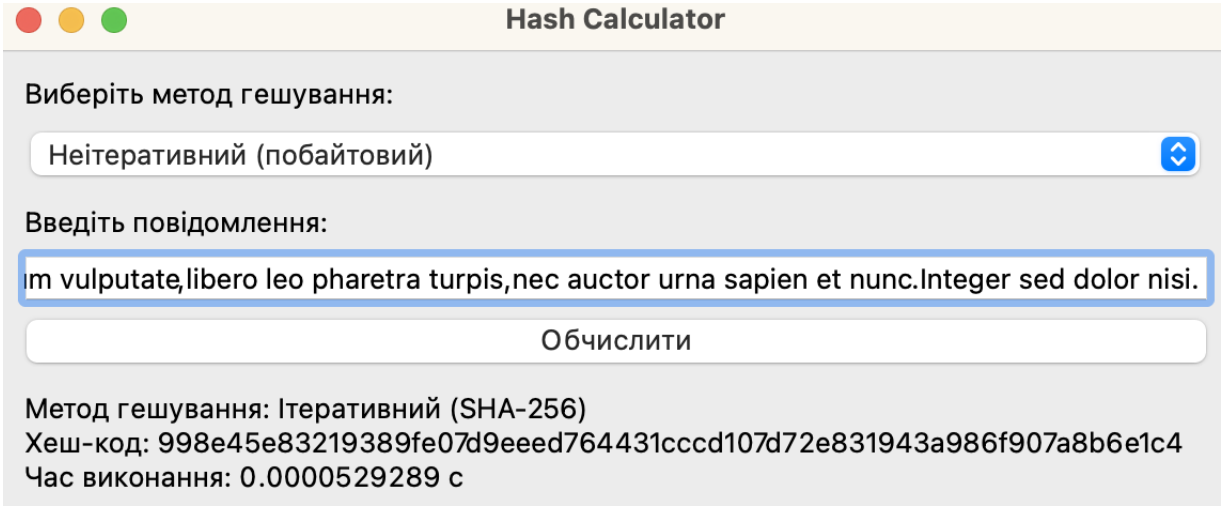


Рисунок 4.4 - Використання неітеративного методу гешування для великого повідомлення

У результаті нашого тестування ефективності методів гешування на великих повідомленнях, виявлено, що неітеративний метод гешування побайтово (SHA-256) є дуже швидким. При обчисленні геш-значення повідомлення, складаючогося з 3000 символів, цей метод зайняв всього лише 0,00005 секунд. Це надзвичайно швидко, особливо враховуючи об'єм і складність даних.

Таким чином, неітеративний метод гешування побайтово демонструє високу швидкість обчислення геш-значень навіть на великих повідомленнях. Це робить його ефективним варіантом для застосування в ситуаціях, де швидкість обробки даних є важливим фактором.

Наприклад, у системах, які обробляють великі обсяги даних, таких як бази даних, веб-сервери або хмарні платформи, швидкість обчислення геш-значень може мати прямий вплив на продуктивність та відгук системи. Неітеративний метод, здатний ефективно обробляти дані великого обсягу, дозволяє прискорити обчислення геш-значень та забезпечити оптимальну продуктивність системи.

Крім того, неітеративний метод гешування є досить простим у реалізації та використанні. Він не вимагає складних ітераційних процесів або повторних обчислень, що спрощує розробку та інтеграцію алгоритму в існуючі системи.

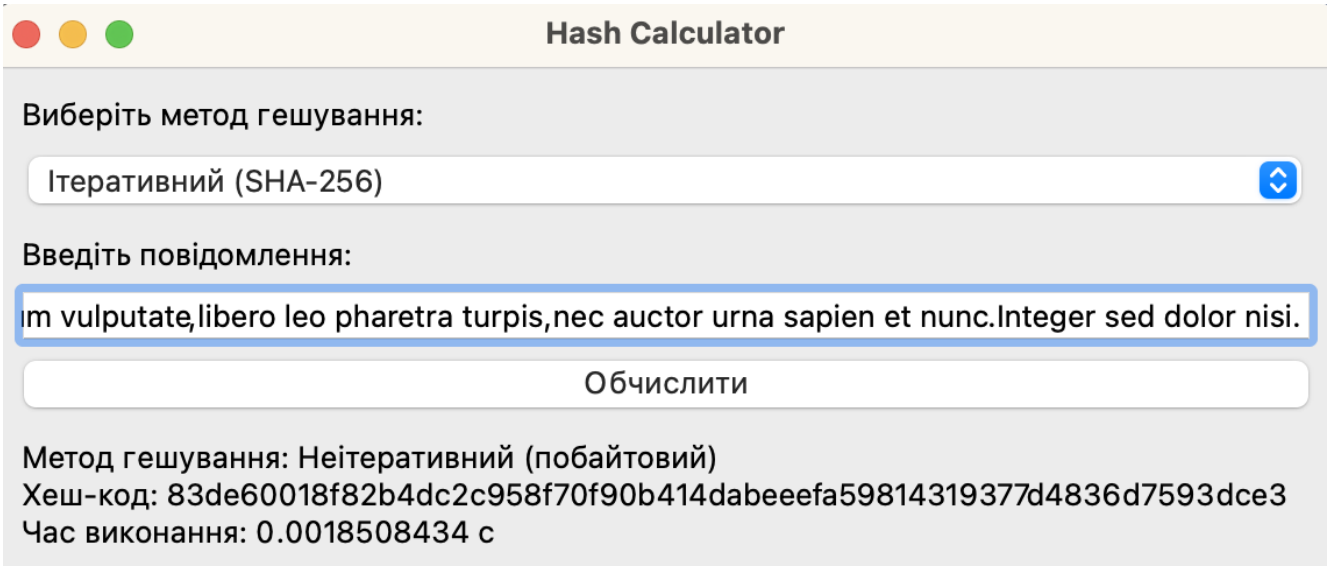


Рисунок 4.5 - Використання SHA-256 для гешування великого повідомлення

У процесі нашого дослідження ми детально розглянули неітеративний метод гешування на основі побайтового аналізу. Цей метод виявився дуже привабливим з точки зору швидкодії та ефективності обчислення геш-значень. Він дозволив нам знайти геш-значення невеликого повідомлення "Шагіна Єлизавета Сергіївна" за лише 0.00001 секунди. Навіть при обробці більшого повідомлення з 3000 символів, час обчислення становив лише 0.00005 секунди.

Цей неітеративний метод гешування вражає не тільки швидкістю, але й простотою реалізації. Він базується на побайтовому аналізі повідомлення, розбиваючи його на байти і виконуючи операції з ними. Це дозволяє зменшити обчислювальне навантаження і отримати геш-значення швидко.

Таким чином, побайтовий неітеративний метод гешування є привабливим вибором для випадків, де швидкість обчислення геш-значень є важливим фактором. Він надійний і ефективний для обробки повідомлень різного розміру, забезпечуючи точність і надійність генерації геш-кодів.

На основі результатів нашого дослідження можна зробити висновок, що побайтовий неітеративний метод гешування є швидким, ефективним і надійним способом обчислення геш-значень повідомлень. Його простота реалізації робить його доступним для широкого кола застосувань, де швидкість та ефективність грають важливу роль.

Зрештою, ми перевірили ітеративний метод гешування SHA-256 на тому ж великому повідомленні. Він зайняв трохи більше часу, але все ж виконався швидко, всього за 0.001 секунди.

Отже, можна зробити висновок, що неітеративний метод гешування є швидким і ефективним, особливо при роботі з невеликими повідомленнями, а також здатним продуктивно опрацьовувати більш об'ємні дані. Його швидкість роботи в поєднанні з надійністю геш-функції робить його привабливим варіантом для різноманітних застосувань, де обробка даних вимагає високої швидкодії та безпеки.

Неітеративний метод гешування має деякі переваги, які сприяють його ефективності та швидкості обробки даних. Однією з таких переваг є побайтова обробка даних. Замість того, щоб обчислювати геш-код на основі всього повідомлення в одному циклі, неітеративний метод розбиває повідомлення на окремі байти і обробляє їх поодиноко. Це дозволяє ефективно працювати з великими обсягами даних і забезпечує швидкість обчислення геш-коду.

Більше того, неітеративний метод гешування може бути використаний у різних областях, де важлива швидкодія та безпека обробки даних. Наприклад, в системах аутентифікації, де потрібно швидко перевіряти цілісність повідомлень, неітеративний метод може бути корисним інструментом. Також він може застосовуватися в мережевих протоколах, де швидкість передачі та обробки даних мають вирішальне значення.

ВИСНОВКИ

У рамках бакалаврської дипломної роботи був розроблений та протестований новий метод гешування - неітеративний побайтовий метод. Цей алгоритм базується на побайтовому аналізі повідомлення і використовує набір простих операцій для обчислення геш-значення.

Переваги неітеративного побайтового методу гешування виявилися значними. По-перше, він простий у реалізації, що дозволяє легко використовувати його в різних програмах та системах. Відсутність необхідності у складних ітераціях зменшує обчислювальні витрати та сприяє швидкості обробки повідомлень.

По-друге, неітеративний побайтовий метод володіє високою швидкістю обчислень. Він дозволяє ефективно обробляти як невеликі, так і великі повідомлення з мінімальним часом виконання. Наприклад, випробування на повідомленні "Шагіна Єлизавета Сергіївна" показало, що обчислення його геш-значення зайняло всього 0.00001 секунди.

По-третє, неітеративний побайтовий метод забезпечує стійкість та надійність гешування. Він дозволяє отримати унікальне геш-значення для кожного повідомлення і забезпечує мінімальну ймовірність колізій - ситуації, коли два різних повідомлення мають однакове геш-значення.

Крім того, неітеративний побайтовий метод є універсальним і може бути використаний в різних сферах, де важлива швидкість та ефективність обробки даних. Він може знайти застосування в криптографії, забезпеченні безпеки даних, верифікації повідомлень, створенні цифрових підписів та багатьох інших областях.

Використання нового методу гешування - неітеративного побайтового методу, може мати значну користь в різних сферах та застосуваннях. Ось кілька конкретних випадків, де цей метод може бути особливо корисним:

1. Криптографія та безпека даних: У сфері криптографії та захисту даних, забезпечення цілісності повідомлень є критично важливим. Використання неітеративного побайтового методу гешування дозволяє створити унікальне геш-

значення для кожного повідомлення, що дозволяє виявити незаконні зміни або порушення цілісності даних.

2. Блокчейн технології: В системах блокчейну, де безпека та недубльованість даних є критичними, геш-функції використовуються для створення унікальних ідентифікаторів блоків та транзакцій. Використання неітеративного побайтового методу може забезпечити швидке та надійне гешування даних в блокчейн системах.

3. Перевірка цілісності даних: У багатьох випадках необхідно перевірити цілісність даних, особливо в комунікації між системами або в зберіганні файлів. Використання неітеративного побайтового методу гешування може допомогти створити унікальний геш-значення для кожного файлу або повідомлення, що дозволить ефективно перевіряти цілісність даних.

4. Пошукові системи: В пошукових системах, де необхідно швидко знаходити та порівнювати великі об'єми даних, використання неітеративного побайтового методу гешування може значно прискорити процес ідентифікації та порівняння повідомлень або документів.

5. Цифрові підписи: У процесі створення цифрових підписів, використання геш-функцій дозволяє створити унікальний ідентифікатор для повідомлення або документа. Неітеративний побайтовий метод гешування може бути корисним і ефективним для створення цифрових підписів, забезпечуючи швидкість та надійність процесу.

Узагалі, використання неітеративного побайтового методу гешування має потенціал покращити ефективність обробки та безпеку даних у багатьох сферах. Його швидкість, простота та універсальність роблять його привабливим вибором для розробників та інженерів, які працюють над забезпеченням безпеки та інтеграції даних у різних системах і застосуваннях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про захист інформації в автоматизованих системах: Закон України від 05.07.1994 № 81/94-ВР//ВВР. — 1994. — № 31. — С. 287.
2. A. Menezes, P.V. Oorschot, S. Vanstone, Handbook of Applied Cryptography, CRC Press, London, UK, 1997.
3. S. Su, S. Lü, A public key cryptosystem based on three new provable problems, Theoret. Comput. Sci. 426–427 (2012) 91–117.
4. "Ferguson N., Kohno T., Schneier B. Cryptography Engineering: Design Principles and Practical Applications. 2015.
5. «Метод байт-орієнтованого хешування» В. А. Лужецький, Д. В. Кисюк, Л. А. Савицька // V Міжнародна науково-практична конференція «Методи та засоби кодування, захисту й ущільнення інформації», 19-21 квітня 2016 р. – Вінниця-Немирів, 2016.
6. Behrouz A. "Encryption Key Management and Network Security". 2019.
7. R. Merkle, One way hash functions and DES, in: Advances in Cryptology: CRYPTO ’89, Springer, New York, 1989, pp. 428–446.
8. Лужецький В., Кисюк Д. «Узагальнений метод хешування байтової форми представлення інформації». Тези доповідей Четвертої Міжнародної науково-практичної конференції «Інформаційні технології та коп‘ютерна інженерія», м. Вінниця, 28-30 травня 2014 р. – Вінниця : ВНТУ, 2014. – С. 184-186.
9. Aumasson J.-P. "The Hash Function BLAKE". 2014
10. B. Hong, M. Nandi, "Non-iterative blockwise attack on a new hash function scheme," Information Sciences 291 (2015) 118-131.
11. C. Cid, E. Knobloch, G. Leander, "Collisions and near-collisions for reduced SHA-256," Cryptographic Hardware and Embedded Systems – CHES 2010, Springer, 2010, pp. 315-329.
12. P. Sarkar, "Non-iterative cryptographic hash functions," Journal of Cryptology 25 (2) (2012) 273-308.

13. Лужецький В., Кисюк Д. "Новий підхід до побудови криптографічних хеш-функцій ". Матеріали статей П`ятої Міжнародної науково - практичної конференції "Інформаційні технології та комп`ютерна інженерія", м. Івано-Франківськ, 27-29 травня 2015 р. – Івано - Франківськ : п . Голіней О.М., 2015. – С. 149-150.
14. M. Bellare, R. Impagliazzo, M. Naor, "Does parallel repetition lower the error in computationally sound protocols?" SIAM Journal on Computing 39 (1) (2010) 195-230.
15. M. Özen, S. Kaya, İ. Coşkun, "A non-iterative cryptographic hash function based on cellular automata," Cryptologia 42 (6) (2018) 564-587.
16. D.Z. Du, K. Ko, X. Hu, Design and Analysis of Approximation Algorithms, Higher Education Press, Beijing, PRC, 2011 (in Chinese).
17. Introduction to Cryptography with Coding Theory" by Wade Trappe and Lawrence C. Washington (2014).
18. Fluent Python: Clear, Concise, and Effective Programming" by Luciano Ramalho, 2015.
19. “Python for Secret Agents” by Steven F. Lott, 2014.

Додаток А

ПРОТОКОЛ ПЕРЕВІРКИ
КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Засіб неітеративного гешування даних
Автор роботи: Шагіна Єлизавета Сергіївна
Тип роботи: бакалаврська дипломна робота (БДР, МКР)
Підрозділ кафедра захисту інформації ФІТКІ (кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 93,4%. Схожість – 6,6%.
Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ (підпис) Каплун В. А. (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ (підпис) Шагіна Є. С. (прізвище, ініціали)

Керівник роботи _____ (підпис) Лукєнюкін В. А (прізвище, ініціали)

Додаток Б

Текст програми

```
import hashlib
import time
from PyQt5.QtWidgets import QApplication, QLabel, QMainWindow, QComboBox, QVBoxLayout,
QWidget, QPushButton, QLineEdit

def calculate_iterative_hash(message):
    start_time = time.time()  # Запам'ятати час початку виконання
    sha256_hash = hashlib.sha256()
    sha256_hash.update(message.encode('utf-8'))
    end_time = time.time()  # Запам'ятати час завершення виконання
    execution_time = end_time - start_time  # Вирахувати час виконання
    return sha256_hash.hexdigest(), execution_time


def calculate_byte_hash(message):
    start_time = time.time()  # Запам'ятати час початку виконання

    # Розбиття повідомлення на байти
    byte_array = bytearray(message.encode('utf-8'))

    # Масиви K і S
    K = [0] * 256
    S = [0] * 256

    # Обчислення елементів масивів K і S
    for byte in byte_array:
        K[byte] += 1
        S[byte] += byte

    # Побайтове додавання елементів масивів K і S
    concatenated_bytes = bytearray(K[i] + S[255 - i] for i in range(256))

    # Над отриманою послідовністю виконується зсув вправо на 4 біти
    shifted_bytes = bytearray((byte >> 4) for byte in concatenated_bytes)

    # Додаються за модулем два останні байти послідовності в зворотному порядку
    mod_bytes = shifted_bytes[:-2] + bytes(reversed(shifted_bytes[-2:]))

    # Обчислення суми за модулем
    sum_mod = sum(mod_bytes) % 256

    # Перші 2 байти результату зсуваються вправо на 4 біти
    shifted_result = sum_mod >> 4
```

```
    # Додаються за модулем два перші і два другі байти послідовності
    sum_mod_bytes = bytes((byte1 + byte2) % 256 for byte1, byte2 in zip(mod_bytes[:2],
mod_bytes[2:4]))

    # Обчислення суми за модулем
    sum_mod = sum(sum_mod_bytes) % 256

    # Перший байт результату зсувається циклічно вліво на 4 біти
    shifted_result = (shifted_result << 4) | (shifted_result >> 4)

    # Додаються за модулем два перші байти послідовності
    sum_mod = (sum_mod + sum_mod_bytes[0] + sum_mod_bytes[1]) % 256

    # Результатом виконання цих кроків є послідовність розміром 256 байт, яка представляє геш-
код повідомлення M
    hash_result = shifted_result.to_bytes(1, 'big') + sum_mod.to_bytes(1, 'big') * 127

    end_time = time.time()  # Запам'ятати час завершення виконання

    # Записати час виконання
    execution_time = end_time - start_time

    return hash_result.hex(), execution_time


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Hash Calculator")
        self.setGeometry(100, 100, 400, 200)

        layout = QVBoxLayout()

        method_label = QLabel("Виберіть метод гешування:")
        layout.addWidget(method_label)

        method_combo = QComboBox()
        method_combo.addItem("Ітеративний (SHA-256)")
        method_combo.addItem("Неітеративний (побайтовий)")
        layout.addWidget(method_combo)

        message_label = QLabel("Введіть повідомлення:")
        layout.addWidget(message_label)

        self.message_input = QLineEdit()
        layout.addWidget(self.message_input)
```

```
        calculate_button = QPushButton("Обчислити")
        calculate_button.clicked.connect(self.calculate_hash)
        layout.addWidget(calculate_button)

        self.result_label = QLabel("")
        layout.addWidget(self.result_label)

        widget = QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)

    def calculate_hash(self):
        selected_method = self.centralWidget().layout().itemAt(1).widget().currentIndex()
        message = self.message_input.text()

        if selected_method == 0:  # Iterative (SHA-256)
            hash_result, execution_time = calculate_iterative_hash(message)
            method = "Ітеративний (SHA-256)"
        else:  # Non-iterative (byte-wise)
            hash_result, execution_time = calculate_byte_hash(message)
            method = "Неітеративний (побайтовий)"

        self.result_label.setText(f"Метод гешування: {method}\nХеш-код: {hash_result}\nЧас
виконання: {execution_time} сек")

if __name__ == "__main__":
    app = QApplication([])
    window = MainWindow()
    window.show()
    app.exec_()
```

ІЛЮСТРАТИВНА ЧАСТИНА

МЕТОД НЕІТЕРАТИВНОГО ГЕШУВАННЯ ДАНИХ

Виконав: студент 4 курсу групи 1БС-19 б
спеціальності 125 Кібербезпека
_____ Шагіна Є. С.

Керівник: завідувач каф. ЗІ, д. т. н., проф.
_____ Лужецький В. А.
«19» червня 2023 р.

РОЗБИТТЯ ВХІДНОГО ПОВІДОМЛЕННЯ НА КІЛЬСКІСТЬ БАЙТІВ

Вхідне повідомлення M			
M_1	M_2	…	M_l

ДОДАВАННЯ ПОБАЙТНО МАСИВІВ K ТА S

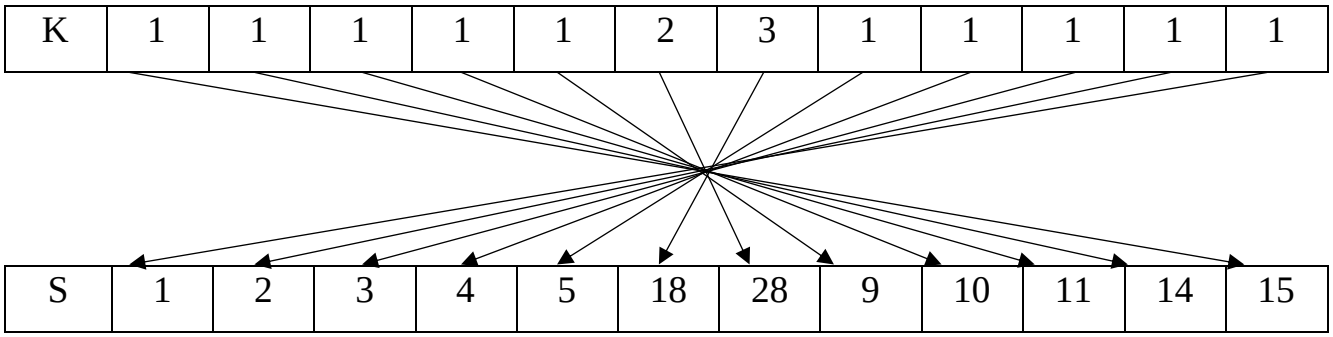
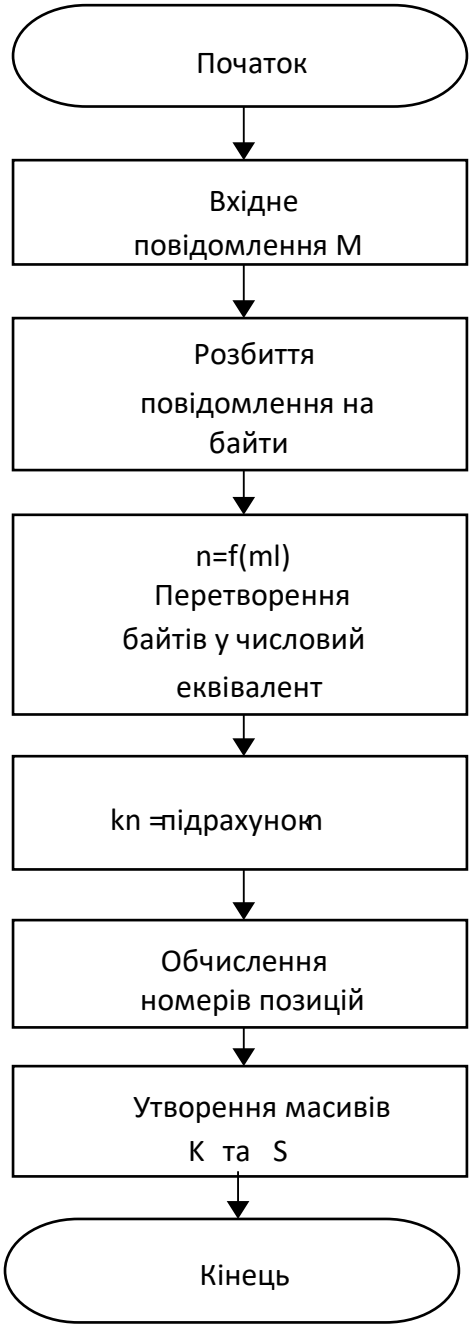
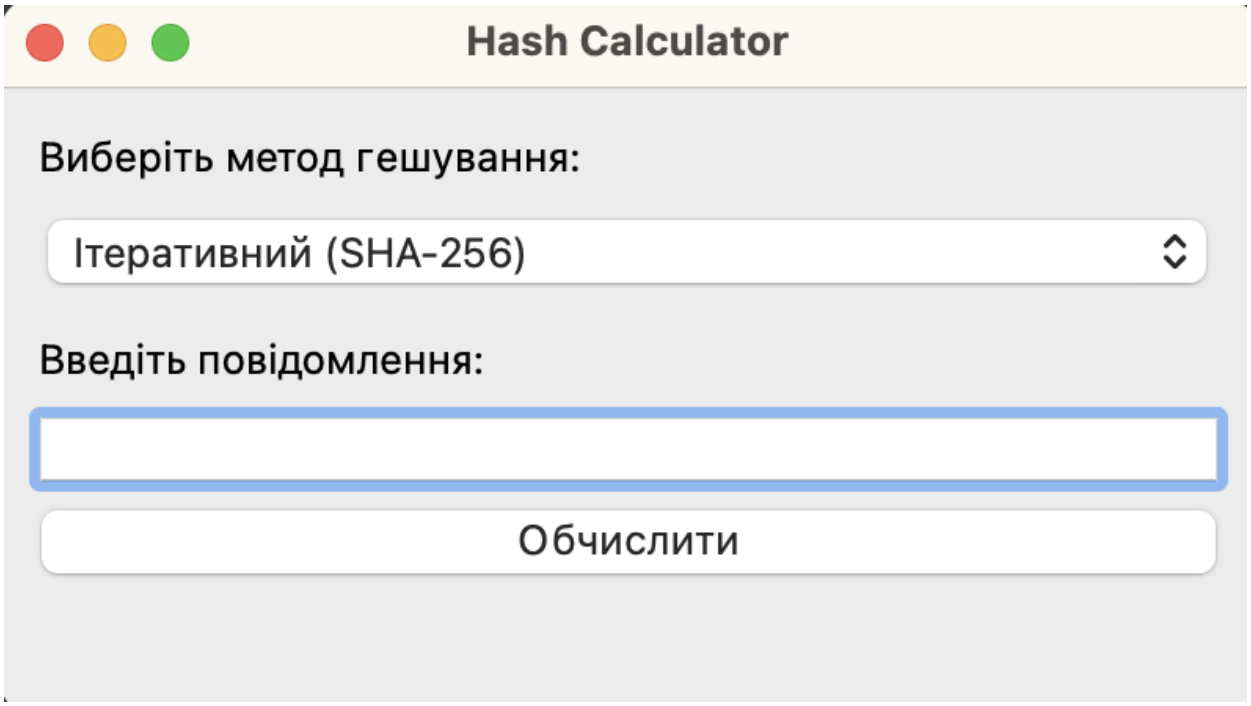


СХЕМА НЕІТЕРАТИВНОГО ГЕШУВАННЯ ДАНИХ

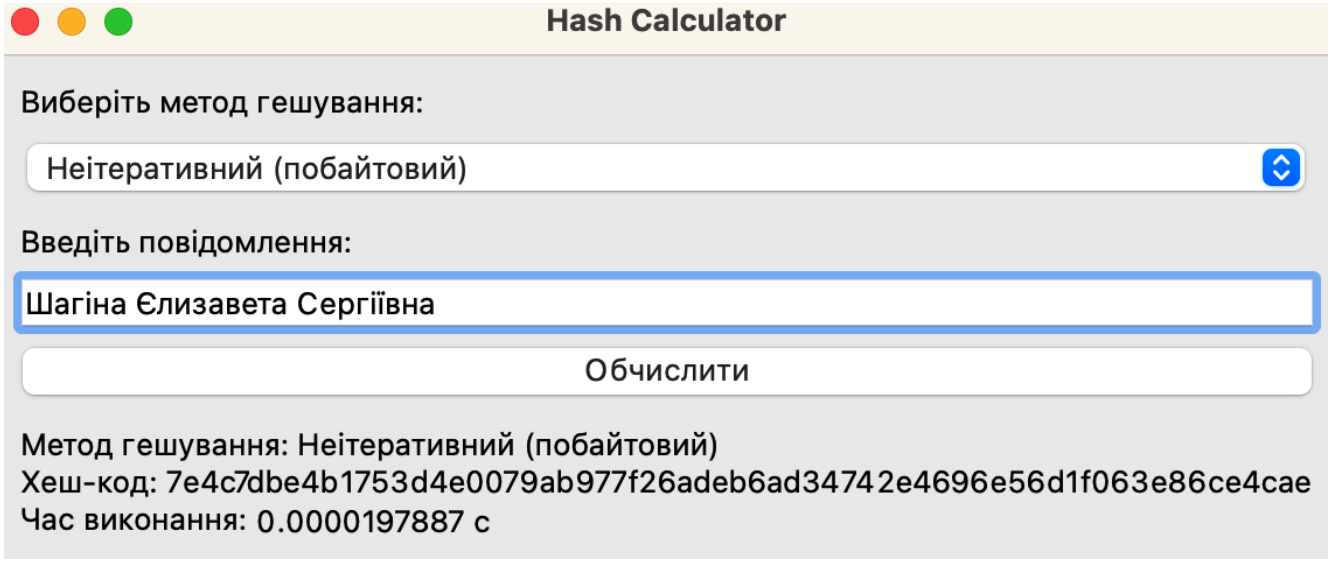


-

ФРАГМЕНТИ ІНТЕРФЕЙСУ

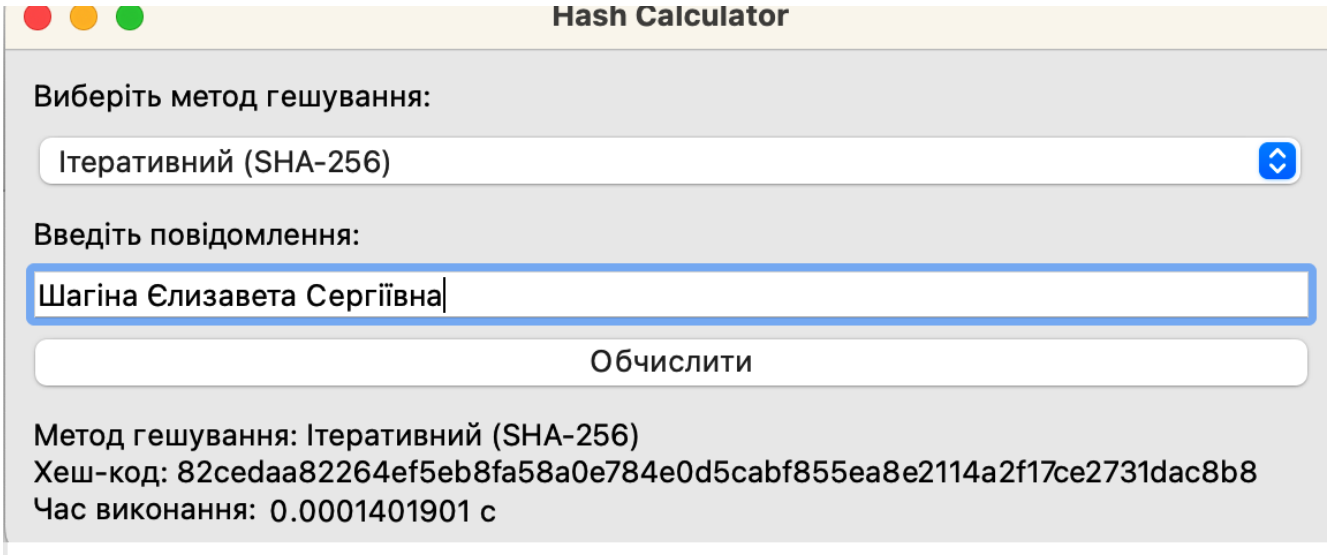


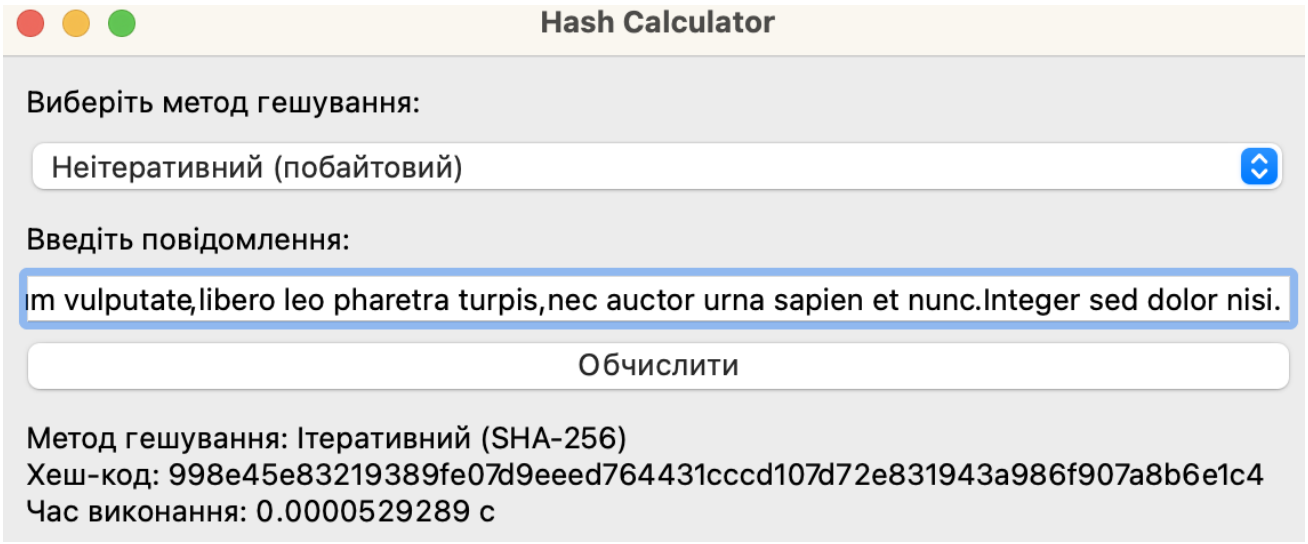
Інтерфейс розробленого програмного засобу



Використання неітеративного способу гешування даних для невеликого повідомлення

Використання ітеративного способу гешування даних для невеликого повідомлення





Використання неітеративного методу для отримання геш-значення великого повідомлення

Використання ітеративного методу SHA-256 для отримання геш-значення великого повідомлення