Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувач кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«14» червня 2022р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи
бакалавра
(назва освітнього рівня)

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Методи та способи удосконалення сучасних алгоритмів хешування»

Виконавець: студент IV курсу, групи КБ-41

_____ **Олег БИХОВЕЦЬ**
(підпис) (ім'я, прізвище)

	Прізвище, ініціали	**Підпис**
Керівник	Яніна ШЕСТАК	

Нормоконтроль	Юрій ЩЕБЛАНІН	

Київ 2022

Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувач кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«01» листопада 2021 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студентові **КБ-41** **Олега Вікторовича БИХОВЦЯ**
(група) (прізвище ім'я по-батькові)

Тема дипломної роботи «Методи та способи удосконалення сучасних алгоритмів хешування»

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №5 від 29.10.2021 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Алгоритми хешування, технології проектування та програмування, інформаційні ресурси

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Основні поняття хеш-функцій, призначення хешування, властивості хеш-функцій, типи хеш-алгоритмів, криптографічні хеш-функції, вразливості хеш-алгоритмів, рекомендації щодо безпечного використання алгоритмів хешування.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Розробка удосконаленого алгоритму хешування та формування рекомендацій покращення безпеки використання хеш-функцій

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 01 листопада 2021 року

Завдання видав _____ Яніна ШЕСТАК
(підпис) (ініціали, прізвище)

Завдання прийняв до виконання _____ Олег БИХОВЕЦЬ
(підпис) (ініціали, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки завдання	01.11.2021 – 01.01.2022	*виконано*
2	Ознайомлення з інформаційними джерелами	02.01.2022 – 13.02.2022	*виконано*
3	Аналіз методів хешування	14.02.2022 – 01.03.2022	*виконано*
4	Дослідження хеш-функцій та їх застосування	02.03.2022 – 22.03.2022	*виконано*
5	Порівняльний аналіз сучасних хеш-функцій	23.03.2022 – 07.04.2022	*виконано*
6	Виявлення вразливостей хеш-алгоритмів	08.04.2022 – 27.04.2022	*виконано*
7	Розробка рекомендацій безпечного використання хеш-алгоритмів	28.04.2022 – 07.05.2022	*виконано*
8	Вибір та оцінка функції для удосконалення	08.05.2022 – 13.05.2022	*виконано*
9	Розробка концепції та програмна реалізація удосконаленої хеш-функції	14.05.2022 – 01.06.2022	*виконано*
10	Оформлення пояснювальної записки	02.06.2022 – 06.06.2022	*виконано*
11	Підготовка до захисту	07.06.2021 – 13.06.2022	*виконано*

Завдання видав _____ Яніна ШЕСТАК
(підпис) (ініціали, прізвище)

Завдання прийняв до виконання _____ Олег БИХОВЕЦЬ
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Дипломна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків. Основний текст займає 56 сторінок, включає в себе зміст, вступ, три розділи дипломної роботи, висновки та список джерел. Крім того, робота містить 1 додаток із кількістю сторінок 2. У дипломній роботі міститься 5 рисунків, 7 формул і 1 таблиця.

Методи дослідження дипломної роботи:

- аналіз літератури;
- аналіз документів;
- порівняльний аналіз;

Об'єктом дослідження є процес виявлення та протидії загрозам, що властиві сучасним алгоритмам хешування.

Предметом дослідження є набір технологій, що реалізують методи та способи захисту алгоритмів хешування.

Практичною цінністю отриманих результатів є поєднання та програмна реалізація засобів та методів забезпечення безпеки використання алгоритмів хешування.

Метою роботи є реалізація способів та методів удосконалення хеш-функцій, надання практичних рекомендацій для безпечного користування алгоритмами хешування.

Для досягнення зазначеної мети поставлено наступні завдання:

- дослідити алгоритми хешування
- провести аналіз найбільш поширених вразливостей, характерних для хеш-функцій
- побудувати удосконалений алгоритм хешування

Ключові слова: хеш-функція, хеш-сума, контрольна сума, хешування, колізії, алгоритм хешування, хеш, хеш-код.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ARM	–	Advanced RISC Machine
MD	–	Message Digest
RFC	–	Request for Comments
RIPEMD	–	RACE Integrity Primitives Evaluation Message Digest
RSA	–	Rivest, Shamir and Adleman
SHA	–	Secure Hash Algorithm

ЗМІСТ

ВСТУП

Актуальність даної роботи визначається тією обставиною, що у ній розглядаються хеш-функції та їх роль у забезпеченні цілісності даних. Різноманітні хеш-алгоритми незмінно є досить розповсюдженими та практично використовуваними, адже порівняння контрольних сум є найлегшим та найдієвішим способом перевірки ідентичності інформації. Також в даній роботі розглянуті проблеми та вразливості деяких хеш-функцій, скориставшись якими, зловмисники зможуть скомпрометувати дані або ж підмінити інформацію в них.

На сьогоднішній день наше суспільство перебуває в ері комп'ютерних технологій. Більша частина важливої інформації та різного роду документації перебуває в електронному вигляді, а отже безпека цієї інформації потребує найвищого рівня захисту.

З розвитком інформаційних технологій роль безпеки даних стає все більш значимою, адже все більше важливої, конфіденційної та таємної інформації потрапляє в мережу та на електронні носії, а отже і більше людей піддаються спокусі посягнути на конфіденційність, цілісність та доступність цих даних.

Саме питання цілісності інформації займає важливе місце в забезпеченні безпеки при обміні та зберіганні даних. В результаті взаємодії з іншим користувачем чи при збереженні, до прикладу, паролів, нам необхідно бути впевненими, що інформація не зазнає змін, редагувань та залишиться в тому вигляді, який їй надав суб'єкт, що оперував даними.

Для цього світовими практиками використовуються хеш-функції, які дозволяють перевірити інформацію на наявність змін, звірити ідентичність даних та контролювати рівень захищеності каналів обміну інформацією.

Тому **метою роботи** є реалізація способів та методів удосконалення хеш-функцій, надання практичних рекомендацій для безпечного користування алгоритмами хешування.

Для досягнення цієї мети було визначено такі *завдання*:

- аналіз методів хешування;
- дослідження та порівняльний аналіз хеш-функцій;
- виявлення вразливостей хеш-алгоритмів;
- розробка рекомендацій безпечного використання хеш-алгоритмів;
- вибір хеш-функції для удосконалення;
- розробка концепції покращеного хеш-алгоритму;
- використання програмного забезпечення для реалізації удосконалення.

Об’єктом дослідження є процес виявлення та протидії загрозам, що властиві сучасним алгоритмам хешування.

Предметом дослідження є набір технологій, що реалізують методи та способи захисту алгоритмів хешування.

Методи дослідження дипломної роботи:

- аналіз літератури;
- аналіз документів;
- порівняльний аналіз;

Рекомендації щодо використання алгоритмів хешування та обґрунтування способу удосконалення хеш-функції також приведені в дипломній роботі. Результати дослідження можуть бути корисними для фахівців, які займаються забезпеченням цілісності інформації.

РОЗДІЛ 1
ОСНОВНІ ПОНЯТТЯ ХЕШ-ФУНКЦІЇ

1.1 Визначення хешування, хеш-функція та контрольна сума

Хешування – це процес перетворення вхідних даних в бітовий рядок фіксованої довжини. При цьому, вхідні дані можуть бути представлені довільною кількістю символів, рядків, даних або файлом. Процес хешування досягається використанням певної хеш-функції.

Хеш-функція – це алгоритм, в результаті виконання якого отримується хеш-сума (яку часто називають просто «хешем») отриманих вхідних даних. Доволі важливо: не слід однозначно ототожнювати поняття «контрольна сума» та «хеш-сума».

Контрольна сума – це значення, що досягається в результаті використання певного алгоритму. При цьому, цей алгоритм може бути представлений не лише хеш-функцією, але й будь-яким іншим математичним, логічним тощо алгоритмом. До прикладу, контрольними сумами також являються циклічний надлишковий код, «контрольні числа», що використовуються в нумерації товарів та документів тощо [1]. Тож, можна стверджувати, що будь-яка хеш-сума є контрольною сумою, однак не будь-яка контрольна сума представляє собою хеш-суму.

1.2 Призначення хешування

Хешування даних активно використовується у всіх галузях інформаційних технологій. Створення та порівняння хеш-сум з технічної точки зору є досить незатратним та легким процесом, який не використовує значних обчислювальних потужностей та дозволяє замінити, оптимізувати та полегшити значну кількість інших процесів.

До основних завдань аналізу за допомогою хеш-сум відноситься:

- Перевірка на ідентичність інформації;
- Прискорення пошуку даних;
- Перевірка на наявність повторів в базах даних;
- Перевірка захищеності каналу передавання або обміну інформації;
- Перевірка на наявність потенційних загроз.

1.2.1 Перевірка на ідентичність інформації

Перевірка на ідентичність інформації є базовою функцією хешування. За допомогою неї можливо перевірити відповідність вмісту отриманого документу відповідно до його оригінального примірника, без доступу до оригіналу та без необхідності перегляду вмісту відповідного документу.

Для цього з отриманого зразка отримується хеш-сума того ж типу (див. 1.5 Криптографічні хеш-функції), що і хеш-сума з оригінального документу. Далі ці два значення порівнюються, і в разі повного співпадіння цих контрольних сум можна стверджувати, що отриманий зразок повністю відповідає оригіналу документу. Якщо ж ці хеш-значення відрізняються, то стає зрозумілим, що даний документ піддавався редагуванню, яке призвело до зміни його структури (видалення, додавання або зміни принаймні 1 біта інформації), що і породило відмінності в хеш-сумах початкового і отриманого файлів. Варто зазначити, що контрольна сума оригінального документу має перебувати окремо від файлу, що передається, тобто міститися в базі даних, бути переданою окремим способом або шляхом, щоб в разі підміни інформації в ході передачі було неможливо підмінити і оригінальну хеш-суму, з якою буде звірений переданий документ.

1.2.2 Прискорення пошуку даних

Прискорення пошуку даних також є однією з переваг використання хешування. Проте, для використання цього методу, наприклад, в своїй базі даних чи

сховищі інформації обов’язково необхідно налагодити та нормувати хешування всієї інформації, яка буде брати участь в цій процедурі.

Наприклад, при додаванні інформації в базу даних за певним принципом може вираховуватись її хеш-значення. За даним хешем інформація буде потрапляти в певний розділ бази даних, що в майбутньому значно полегшить її пошук в разі необхідності. Так, для пошуку можна буде використовувати лише хеш-код даних. Такий спосіб можна порівняти з принципом словника, де словами слугує інформація, яку ми заносимо в базу даних, а літерами – розділи, куди ця інформація потрапляє. Таким чином, щоб відновити чи знайти необхідну нам інформацію не обов’язково буде знати її повний точний зміст, а достатньо лише знати, в якому розділі вона знаходиться, на що і вказує отримане хеш-значення з неї [2].

1.2.3 Перевірка на наявність повторів в базах даних

Перевірка на наявність повторів в базах даних є подібною до попереднього пункту, проте має за основу різну ціль та результат.

До прикладу, нам необхідно додати певну інформацію до бази даних. Проте ми не впевнені, чи міститься вже ця інформаціє в ній, що, в такому разі, позбавить нас необхідності дублювати її туди ще раз. Для цього ми за допомогою певної хеш-функції отримуємо контрольну суму даної інформації і по ній здійснюємо пошук по базі. В результаті ми отримаємо місцезнаходження вже існуючого елементу з шуканою інформацією або відсутність результату пошуку, що свідчить, що дана інформація відсутня в базі даних, тож вона не буде дублем при її додаванні.

Слід зазначити, що для виконання і цієї задачі також необхідна функціонуюча система додавання і розміщення хеш-значень до записів в базі даних. Вона буде особливо ефективною при однотипних (коли це полегшує пошук, у порівнянні з візуальним пошуком) або ж при коротких записах у базі (наприклад, коли хеш-значення береться лише з заголовку статті – тоді і пошук можна здійснювати лише по хеш-значенню заголовка). Це пояснюється тим, що чим більші дані ми конвертуємо за допомогою хеш-функції в контрольну суму, тим більша є

ймовірність помилки в тексті при повторному конвертуванні для здійснення пошуку. А, як відомо, зміна хоч 1 біта в ньому приведе до повної зміни хеш-суми, що приведе до відсутності результатів пошуку.

1.2.4 Перевірка захищеності каналу передавання або обміну інформацією

За допомогою хешування також може відбуватися перевірка захищеності каналу передавання або обміну інформацією. Таким чином можливо перевірити канал на наявність атаки типу «людина посередині» та інших можливих порушень конфіденційності каналу передавання або обміну інформацією.

Для цього каналом, що перевіряється, передається тестова інформація, а іншим конфіденційним каналом – хеш-код даної інформації. Після отримання тестової інформації, користувач, використовуючи ту саму хеш-функцію, що була використана при отриманні хеш-суми, яка передавалася конфіденційним каналом, робить новий зразок хешу отриманої інформації і порівнює його з даним. Якщо хеш-суми ідентичні – це означає, що принаймні явних ознак прослуховування каналу немає. Коли ж хеш-код отриманої інформації різниться з хеш-кодом, що був отриманий до її відправки каналом – це може свідчити, що підчас прямування каналом інформація була перехоплена та зазнала змін. Таким чином, канал можна вважати не конфіденційним та не використовувати його для передачі чи обміну інформацією.

Варто зазначити, що така перевірка хоч і є показовою, проте не завжди може гарантувати вірний результат. До прикладу, є ймовірність, що зловмисник, при перехопленні інформації, ніяк не вплине на її зміст, таким чином хеш-сума інформації залишиться без змін, що може створити ілюзію захищеності каналу. Проте у випадку, коли хеш отриманих даних все ж різниться з початковим значенням – це явна причина ставити під сумнів конфіденційність каналу.

1.2.5 Перевірка на наявність потенційних загроз

Ще однією розповсюдженою задачею, яка вирішується з використанням хешування є перевірка файлів на наявність потенційних загроз.

Основою будь-якого антивірусного захисту є сигнатурний аналіз файлів, в якому і використовується порівняння хеш-сум файлу з хеш-сумами відомих загроз. В процесі сигнатурного аналізу антивірус використовує окремі частини файлу (також може брати файл повністю), отримує їх контрольні суми та проводить порівняння отриманих зразків з контрольними сумами відомих загроз, що містяться в базах даних антивірусів. Таким чином, якщо хеш-сума хоча б однієї частини файлу співпаде з хеш-сумою, що міститься в базі даних антивірусів, то можна стверджувати, що даний файл містить шкідливий програмний код, про що і сповістить антивірусний засіб.

Перевагою такого аналізу є вкрай мала ймовірність хибних спрацювань, адже співпадіння хеш-сум означає повне співпадіння з вже раніше виявленим шкідливим кодом. Над поповненням баз контрольних сум шкідливого програмного коду практично в кожній компанії займається група експертів в області комп’ютерної вірусології [3], яка спеціалізується на вивченні нових та модифікації старих загроз. Тож, даний метод є досить дієвим способом виявлення потенційно небезпечних файлів.

Недоліком такого аналізу можна виокремити нестійкість до нових або невідомих видів шкідливого коду, контрольні суми якого ще не були додані до баз даних антивірусного захисту.

1.3 Властивості хеш-функцій

Будь-яка хеш-функція має дотримуватися певних вимог, щоб мати змогу виконувати свої основні функції та не піддавати ризику дані, з якими вона працює. Ці вимоги носять радше рекомендаційний характер, проте без них функція стає

надто вразливою та має істотно менший шанс використання та розповсюдження, у порівнянні з іншими використовуваними алгоритмами.

В загальному, існує 2 основні вимоги до хеш-функцій:

- Мінімальна вірогідність виникнення колізій.

Колізії – це випадок в хешуванні, коли вихідні хеш-значення функції від двох різних вхідних даних збігаються. Тобто два різних рядки дають один результат. Це є основною алгоритмічною проблемою хеш-функцій, вирішення якої і є ключем до створення досконалого алгоритму хешування.

- Незначне навантаження на обчислювальні потужності системи.

Хеш-функція має містити такий алгоритм та кількість раундів, щоб не завдати критичного навантаження системам, при тому повністю виконувати свої функції. Ця вимога носить технічний, а від того і більш гнучкий характер, адже існує безліч систем з різними характеристиками та обчислювальними можливостями. Проте в ідеальному випадку алгоритм має виконуватись навіть на найслабших системах та комп’ютерах.

Проте, крім цього, існують і інші вимоги, що властиві «гарним» хеш-функціям.

- Відкритий алгоритм хешування.

Ця властивість буде корисна спеціалістам, які працюють з різними алгоритмами хешування. Так, фахівці зможуть самостійно оцінити криптостійкість алгоритму хешування та самостійно доповнювати та редагувати функцію.

- Приведення вихідних даних до заданого розміру.

Алгоритм має незалежно від кількості та довжини вхідних даних вміти конвертувати їх в код заданого розміру. Довжина вихідних даних залежить від алгоритму, що використовується.

- Будь-яка зміна має повністю змінювати хеш-код.

В хороших алгоритмах хешування зміна, видалення або додавання будь-якого біта інформації має приводити до абсолютної зміни хеш-суми. Це необхідно для того, щоб потенційний зловмисник не зміг підібрати ключ функції, використовуючи метод подібностей при переборі.

- Стійкість до зворотного перетворення.

Хеш, на відміну від шифрування, не служить методом передачі даних, а радше використовується для перевірки цілісності та конфіденційності даних. Тому алгоритм повинен забезпечувати односторонність функції хешування для того, щоб хеш-код можна було використовувати як орієнтир та відкриту інформацію про дані.

- Детермінованість.

При подачі на вхід функції двох однакових даних в результаті ми маємо постійно отримувати те саме хеш-значення. Цей пункт також є базовим поняттям для будь-якого хеш-алгоритму і виключає рандомні операції та дії, що не пов’язані з випадковістю масиву вхідних стрічок [4, 5, 6].

1.4 Типи хеш-функцій

Різновидів хеш-функцій існує дуже багато. Пов’язано це з тим, що кожен спеціаліст, який прагне створити ідеальну хеш-функцію, по-своєму бачить способи вирішення основних проблем хешування, а саме забезпечити не громіздку в плані обчислювальних спроможностей, функцію, а також забезпечити її криптостійкість і мінімалізувати можливість виникнення колізій, тобто співпадіння значень хеш-сум у двох різних випадках вхідних даних.

Для того, щоб виділити типи хеш-функцій та продемонструвати їх основний принцип дії скористаємося загальноприйнятими позначеннями, а саме:

- K – це кількість можливих ключів до хеш-функції, тобто кількість вхідних даних, які ми «передаємо» для хешування;
- h(k) – це безпосередньо хеш-функція, яку ми використовуємо до вхідних даних.
- M – це кількість значень, що можуть утворитися в результаті хешування, тобто кількість можливих вихідних значень для хеш-функції. До прикладу, якщо для функції вихідним значенням слугує 3 байта (24 біт) цифрової інформації (тобто результат виражено цифрами), то множина всіх значень M буде рівна 1000, адже результат будь-якої функції буде виражений числом від 000 до 999.

Тож, користуючись цими позначеннями, розглянемо основні типи хеш-функцій.

1.4.1 Хеш-функції основані на діленні

В загальному випадку, хеш-функція, в основі якої лежить метод ділення, використовує залишок від ділення вхідного ключа (K) на кількість можливих значень (M). У виді формули це можна зобразити так:

$$h(K) = K\ mod(M). \tag{1.1}$$

Цей метод є найбільш «легким» в плані обчислювальних можливостей, адже його реалізація заключається у виконанні однієї простої математичної дії. З іншого боку, інший важливий компонент, а саме кількість співпадінь хеш-значень для різних вхідних даних, буде нівелювати всі його потенційні переваги.

Також слід зазначити, що даний метод потребує додаткової умови: ступінь парності K має збігатися зі ступенем парності M, адже в протилежному випадку це може призвести до зсуву даних в файлі [2].

1.4.2 Хеш-функції основані на множенні

Даний вид хеш-функцій в плані реалізації більш складний, ніж перший. Для його теоретичної реалізації нам необхідно ввести декілька нових позначень:

- ω – це кількість варіантів вихідних даних у вигляді машинного слова. Так, для 32-розрядних процесорів в комп’ютерах IBM PC воно рівне 2^{32}.
- A – обрана константа, яка має ціле значення та є взаємно простою до ω (тобто не маю взаємних дільників, окрім 1).

Таким чином отримуємо формулу:

$$h(K) = \left[M \left\lfloor \frac{A}{\omega} \times K \right\rfloor\right]. \quad (1.2)$$

З цієї формули можна визначити, що у випадку з двійковою системою обчислення, M буде виступати степенем двійки, а функція h(K) складатиметься зі старших бітів добутку A×K.

Цей метод, подібно до попереднього, має свої переваги. Вони дозволяють зберігати відносну послідовність у створенні хеш-значень. Тобто, якщо вхідними даними обрати певну арифметичну прогресію, то результатом таких типів хешування будуть хеш-значення, які також мають певний взаємозв'язок, який можна відслідкувати. Проте це зменшує стійкість такої функції до взлому, адже зловмисник також зможе відслідкувати та використати такий взаємозв'язок.

1.4.3 Хешування рядків змінної довжини

Розглянуті вище методи спеціалізуються на ситуаціях, коли довжина вхідних даних нам чітко відома та заздалегідь визначена. Звісно, деякі з них здатні обробляти і рядки змінної довжини за допомогою різноманітних операцій, проте для таких цілей існують окремі функції.

Найпростішою з таких є, так зване, хешування Пірсона – хеш-алгоритм, який дозволяє перетворювати рядки будь-якої довжини в хеш. Така функція здатна приймати будь-яке слово, що складається з n символів та конвертувати його в хеш-код довжиною від 0 до 255 байт.

Можна виділити такі етапи роботи алгоритму:

1. Визначається початкове значення змінної h (зазвичай, h=0).
2. Запускається цикл, в якому змінна c «перебирає» всі символи рядка W, який було передано на вхід.
3. В циклі використовується таблиця перестановок T. Для початку виконується операція xor між h і c. Далі змінній h присвоюється значення, що перебуває в таблиці T саме на тому індексі, значення якого ми отримали при операції xor.

4. Завершується дія циклу та повертається значення h в алгоритм.

Перевагою цього алгоритму є те, що він використовує всі символи, які подаються на вхід функції, отже будь-яка зміна інформації призведе до зміни значення з хеш-суми. Також слід зазначити, що цей метод досить простий у використанні і може опрацювати дані будь-якої довжини (ймовірність колізії завжди є сталою).

1.4.4 Ідеальне хешування

Ідеальна хеш-функція – це хеш-функція, яка здатна без колізій відобразити різні елементи з множини об'єктів на множині ключів за певний період часу.

Ідеальне хешування використовується в задачах зі статичною (сталою) множиною ключів (тобто після того, як всі ключі збережені в таблиці, їх множина ніколи не змінюється) для забезпечення гарної асимптотики навіть у гіршому випадку. Також можливий варіант застосування ідеального шифрування у випадку, коли розмір таблиці лінійно залежатиме від кількості ключів.

У такому хешуванні для доступу до даних потрібно лише обчислення хеш-функцій (однієї або кількох), що робить даний підхід найшвидшим для доступу до статичними даними. Дана технологія застосовується в різних словниках та базах даних, в алгоритмах зі статичною (відомою заздалегідь) інформацією.

Розглянемо принцип дії ідеального хешування детальніше. В загальному випадку, такий алгоритм використовує дворівневу схему хешування з універсальним хешуванням на кожному рівні.

Нехай в словнику міститься n кількість різних слів. Тоді розмір таблиці також буде відповідати значенню n. В такому випадку створення хеш-функції, яка б кожен ключ відображала б на індекс масиву, де він знаходиться є виконуваною, проте важкореалізованою задачею. В більш загальних випадках реалізація даного типу хешування обмежується застосуванням наступної формули:

$$h(k) = (A \times k + B)\ mod\ P\ mod\ N \qquad (1.3)$$

де k – хеш-значення «ідеального» ключа,

P – константа, яка більша за будь-яке значення можливого ключа k,

N – розмір хеш-таблиці, тобто кількість всіх ключів.

Реалізація даного алгоритму використовує двоетапне хешування, де на першому етапі ми знаходимо «квартиру» (комірку, в якій знаходиться хеш-значення), а на другому етапі – «кімнату» (значення в підтаблиці, зв'язаної з коміркою).

На першому етапі нашою головною задачею є підбір глобальних коефіцієнтів A та B з формули, представленої вище. Ця функція вираховуватиме номер комірки, в якій буде знаходитись значення для заданого ключа. При цьому, нашою основною задачею є досягнення результату, при якому максимальна кількість «кімнат» (колізій) в одній «квартирі» було мінімальним. Це досягається простим підбором коефіцієнтів A та B. Так, за 100 різних ітерацій підбору для словника з 10000 ключів максимальна кількість колізій для одного елементу не буде перевищувати 10.

Другий етап має на меті резервування K^2 «кімнат» для розміщення K елементів в одній «квартирі». Така система дозволить без проблем підібрати коефіцієнти A та B для розміщення K елементів без жодної колізії, тому що значення всіх ключів відоме заздалегідь, бо хеш-коди всіх ключів будуть різними. Ці коефіцієнти підбираються для кожної «квартири» окремо і зберігаються в кожному елементі першої хеш-таблиці разом з кількістю «кімнат».

Для пошуку будь-якого елементу в ідеальній хеш-таблиці необхідно виконати наступні дії:

1. Знайти номер «квартири» по хеш-коду ключа k:

$$i = (A \times k + B) \ mod\ P\ mod\ N \tag{1.4}$$

2. Обрати коефіцієнти для обчислення номеру «кімнати»: A_i, B_i, K^2.
3. Знайти номер «кімнати» за формулою:

$$n = (A_i \times k + B_i) \ mod\ P\ mod\ K^2 \quad (1.5)$$

4. Повернути значення знайденої кімнати.

В будь-якому випадку (як в найкращому, так і найгіршому) на пошук елемента знадобиться стала кількість дій, яка не залежатиме від загальної кількості елементів.

Може виникнути враження, що реалізація такого алгоритму може ставити під сумнів виконання іншої важливої задачі – оптимізації роботи обчислювальних потужностей, за рахунок складності використання K^2 різних «кімнат» (значень в підтаблиці) для кожної «квартири» (комірки). Проте практичні досліди показують, що витрати пам’яті в такому випадку не перевищують лінійну складність [7, 8].

Ідеальне хешування застосовується, якщо потрібно присвоїти унікальний ідентифікатор ключа без збереження будь-якої інформації про ключ. Приклад використання ідеального (або швидше k-ідеального) хешування: розміщення хешей, пов'язаних з даними, що зберігаються у великій та повільній пам'яті, у невеликій та швидкій пам'яті. Розмір блоку можна вибрати таким, щоб дані зчитувалися з повільної пам'яті за один запит. Подібний підхід використовується, наприклад, в апаратних маршрутизаторах . Також ідеальне хешування використовується для прискорення роботи алгоритмів на графах, якщо уявлення графа не міститься в основній пам'яті [9].

1.4.5 Універсальне хешування

Універсальне хешування – це тип хешування, при якому використовується не одна конкретна хеш-функція, а відбувається вибір із певної кількості доступних хеш-функцій за певним випадковим алгоритмом. Даний тип хешування вирізняється низькою вірогідністю колізій.

Припустимо, що хочемо відобразити ключі з простору U в числа [m]. На вході алгоритм отримує деякий вибір даних S∈U та розмірністю n, причому невідомий заздалегідь. Як правило, метою хешування є отримання найменшого числа колізій, чого важко досягти, використовуючи якусь певну хеш-функцію. Саме рандомізація

використання функції дозволяє зменшити число колізій. Вибір функкції з певного набору називається універсальним сімейством та реалізується формулою [10]:

$$H = \{h: U \to [m]\}. \quad (1.6)$$

1.4.6 Подвійне хешування

Подвійне хешування — це тип хешування, який заснований на використанні двох хеш-функцій для побудови різних послідовностей використання хеш-таблиці.

Техніка подвійного хешування використовує одне хеш-значення як індекс у таблицю, а потім повторює крок вперед на інтервал, поки не буде знайдено потрібне значення, не буде досягнуто порожнє місце або не буде здійснено пошук у всій таблиці; але цей інтервал встановлюється другою незалежною хеш-функцією. На відміну від альтернативних методів роздільної здатності зіткнень лінійного та квадратичного зондування, інтервал залежить від даних, тому значення, які відображаються в одному місці, мають різні послідовності сегментів; це мінімізує повторні зіткнення та наслідки кластеризації.

Описати принцип подвійного хешування можна за допомогою формули:

$$h(i,k) = \big(h_1(k) + i \times h_2(k)\big) mod\ |T|. \quad (1.7)$$

Загалом, $h_1(k)$ та $h_2(k)$ вибираються з набору універсальних хеш-функцій; , $h_1(k)$ вибирається, щоб мати діапазон {0,|T|-1} і $h_2(k)$ лежатиме в діапазоні {1,|T|-1} Подвійне хешування наближається до випадкового розподілу; точніше, попарно незалежні хеш-функції дають ймовірність $(\frac{n}{|T|})^2$, що будь-яка пара ключів буде виконувати ту саму послідовність.

Вторинна хеш-функція $h_2(k)$ повинна мати кілька характеристик:

- вона ніколи не повинна давати нульовий індекс.
- вона повинна циклізувати всю таблицю.
- вона має бути швидкою для обчислення.

- вона повинна бути попарно залежною з $h_1(k)$.
- вона має бути абсолютно не прогнозованою.
- всі $h_2(k)$ мають бути відносно простим до |T|.
- Якщо для обох функцій використовується хешування методом ділення, то дільники обирають простими числами.
- Якщо T є степенем 2, то функція $h_2(k)$ має повертати не парне число.

Подвійне хешування корисно, якщо програмі потрібна менша хеш-таблиця, оскільки воно ефективно знаходить вільний слот. Хоча обчислювальні витрати можуть бути високими, подвійне хешування може знайти наступний вільний слот швидше, ніж підхід лінійного зондування.

1.5 Криптографічні хеш-функції

Одними із найбільш розповсюджених і найвикористовуваніших різновидів хеш-алгоритмів є криптографічні хеш-функції. Від інших видів хеш-алгоритмів дані функції відрізняються своєю криптостійкістю, що досягається дотриманням таких трьох основних правил:

- Незворотність: для відомого значення хеш-суми m певної функції має бути неможливим знайти блок даних X, для якого h(X) = m.
- Стійкість до колізій першого роду: для відомого повідомлення M має бути обчислювально неможливо підібрати інше повідомлення N, для якого h(N)=h(M).
- Стійкість до колізій другого роду: має бути обчислювально неможливо підібрати пару повідомлень (M, M`), що мають однаковий геш.

Однак, варто додати, що на практиці не доведене існування жодного прикладу, коли хеш-функцію можна було б назвати повністю незворотньою. Зазвичай, знаходження прообразу є лише обчислювально складним завданням, яке потребує багато часу та обчислювальних можливостей. Тому n-бітну хеш-функцію прийнято вважати криптостійкою, коли обчислювальна складність знаходження колізій в ній близька до значення $2^{\frac{n}{2}}$.

Ще однією важливою властивістю, яку повинна мати криптографічно стійка хеш-функція – це властивість, звана «лавинним ефектом». Вона полягає в тому, що при найменшій зміні принаймні біта інформації у вхідному повідомлення вигляд вихідної хеш-суми повинен докорінно змінюватись. Так, значення хешу не повинно дозволяти витоку інформації навіть щодо окремих бітів вхідних даних. Ця вимога є запорукою криптостійкості та надійності алгоритмів хешування в криптографічних хеш-функціях.

Криптографічні хеш-функції призначені для запобігання можливості повернення контрольної суми, яку вони створюють, до початкових текстів. Однак, хоча їх практично неможливо повернути назад, вони не гарантують на 100 відсотків захист даних. Хакери можуть використовувати райдужну таблицю, щоб зрозуміти простий текст контрольної суми. Райдужні таблиці – це словники, що містять тисячі, мільйони або навіть мільярди контрольних сум поряд із відповідними значеннями простого тексту. Хоча це технічно не скасовує криптографічний хеш-алгоритм, це може бути і з огляду на те, що це так просто зробити. Насправді, оскільки жодна райдужна таблиця не може перерахувати всі можливі контрольні суми, що існують, вони, як правило, корисні лише для простих фраз, таких як слабкі паролі.

Криптографічні хеш-функції мають багато додатків для захисту інформації, зокрема в цифрових підписах , кодах аутентифікації повідомлень та інших формах аутентифікації. Їх також можна використовувати як звичайні хеш-функції, для індексації даних у хеш-таблицях, для відбитків пальців, для виявлення повторюваних даних або однозначної ідентифікації файлів, а також як контрольні суми для виявлення випадкового пошкодження даних. Дійсно, в контекстах інформаційної безпеки криптографічні хеш-значення іноді називають (цифровими) відбитками пальців, контрольними сумами або просто хеш-значеннями, хоча всі ці терміни означають більш загальні функції з досить різними властивостями та цілями.

Всі хеш-функції, які на сьогодні є найбільш використовуваними та розповсюдженими обов’язково відповідають всім вимогам криптостійкості

криптологічних хеш-функцій. Розглянемо детальніше основи роботи розповсюджених хеш-алгоритмів [11, 12].

1.5.1 Хеш-функція MD4

MD4 (Message Digest 4) – хеш-функція, розроблена професором Рональдом Рівестом у 1990 році. Для довільного повідомлення функція генерує 128-розрядне хеш-значення, зване дайджестом повідомлення. Цей алгоритм використовується в протоколі автентифікації MS-CHAP, розробленому корпорацією Microsoft для виконання процедур перевірки автентичності віддалених робочих станцій Windows.

Нижче (рис. 1.1) показаний принцип роботи однієї операції алгоритму MD4. Хешування з MD4 складається з 48 таких операцій, згрупованих у 3 раунди по 16 операцій. F – нелінійна функція; у кожному раунді функція змінюється. M_i означає 32-бітний блок вхідного повідомлення, а K_i - 32-бітна константа, різна для кожної операції.

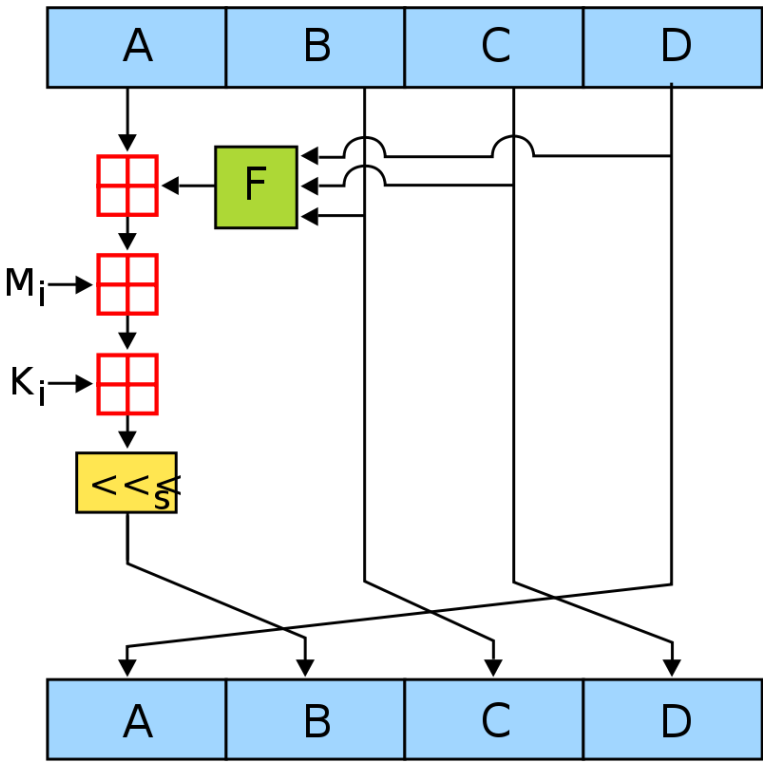

Рисунок 1.1 – Схематичне представлення принципу роботи однієї операції алгоритму MD4

Алгоритм MD4 складається з наступних кроків:

1. Додавання відсутніх бітів.
2. Додавання довжини повідомлення.
3. Ініціалізація MD-буфера.
4. Обробка повідомлення блоками 16 слів.
5. Формування хешу.

Рівень безпеки, що закладався в MD4, був розрахований створення досить стійких гібридних систем електронного цифрового підпису, заснованих на MD4 і криптосистемі з відкритим ключем. Рівест вважав, що алгоритм хешування MD4 можна використовувати і для систем, що потребують сильної криптостійкості. Але в той же час він зазначав, що MD4 створювався насамперед як дуже швидкий алгоритм хешування, тому він може бути недостатньо хорошим у плані криптостійкості. Що підтвердилося подальшими дослідженнями. Тому для додатків, де важлива перш за все криптостійкість, став використовуватися алгоритм MD5 [13].

1.5.2 Хеш-функція MD5

Алгоритм хешування MD5 (алгоритм дайджесту повідомлень) є односторонньою криптографічною функцією, яка приймає повідомлення будь-якої довжини як вхідні дані та повертає як вихідне значення дайджеста фіксованої довжини, яке буде використано для аутентифікації вихідного повідомлення.

Безпосередньо автор алгоритму, засновник RSA Data Security LLC і професор Массачусетського технологічного інституту Рональд Рівест писав про свій алгоритм так: «Алгоритм приймає як вхідні дані повідомлення довільної довжини і видає 128-бітний «відбиток пальця» або «дайджест повідомлення» вхідних даних. Передбачається, що обчислювально неможливо створити два повідомлення з однаковим дайджестом повідомлень або створити будь-яке повідомлення з заданим заздалегідь цільовим дайджестом повідомлення. Алгоритм MD5 призначений для

програм цифрового підпису, де великий файл повинен бути «стиснутий» безпечним способом перед шифруванням за допомогою приватного (секретного) ключа в криптосистемі з відкритим ключем, наприклад RSA».

Хоча MD5 спочатку був розроблений як алгоритм шифрування коду аутентифікації повідомлень для використання в Інтернеті, хешування MD5 більше не вважається надійним для використання в якості контрольної суми в криптографії, оскільки експерти з безпеки продемонстрували методи, здатні легко створювати зіткнення (колізії) MD5 на комерційних стандартних комп'ютерах. Зіткнення шифрування означає, що два файли мають однаковий хеш. Хеш-функції використовуються для захисту повідомлень, захисту паролів, комп'ютерної криміналістики та криптовалюти.

Алгоритм хешування повідомлень MD5 обробляє дані в 512-бітових рядках, розбитих на 16 слів, що складаються з 32 біт кожне. Вихід з MD5 є 128-бітним значенням дайджесту повідомлення.

Обчислення значення дайджесту MD5 виконується на окремих етапах, які обробляють кожен 512-бітовий блок даних разом із значенням, обчисленим на попередньому етапі. Принцип роботи однієї операції в алгоритмі MD5 представлено нижче (рис. 1.2).

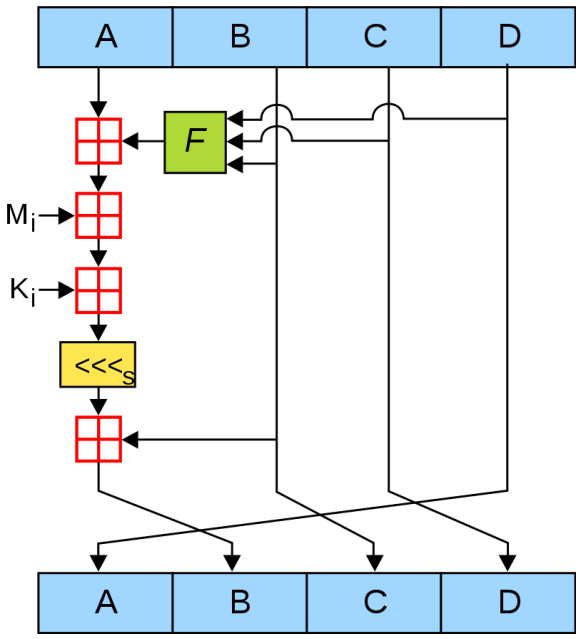

Рисунок 1.2 – Схематичне представлення принципу роботи однієї операції алгоритму MD5

Перший етап починається зі значень дайджесту повідомлення, ініціалізованих за допомогою послідовних шістнадцяткових числових значень. Кожен етап включає чотири проходи дайджесту повідомлень, які маніпулюють значеннями в поточному блоці даних і значеннями, обробленими з попереднього блоку. Остаточне значення, обчислене з останнього блоку, стає дайджестом MD5 для цього блоку [15].

Порівнюючи алгоритми MD4 та MD5 можна виокремити такі відмінності:

- MD4 використовує три цикли по 16 кроків кожен, тоді як MD5 використовує чотири цикли по 16 кроків кожен.
- У MD4 додаткова константа у першому циклі не застосовується. Аналогічна додаткова константа використовується для кожного з кроків другого циклу. Інша додаткова константа використовується для кожного кроку в третьому циклі. У MD5 різні додаткові константи, T[i], застосовуються для кожного з 64 кроків.
- MD5 використовує чотири елементарні логічні функції, по одній на кожному циклі, в порівнянні з трьома MD4, по одній на кожному циклі.
- У MD5 на кожному кроці поточний результат складається з попереднього кроку. MD4 це додавання не включає.
- Алгоритм MD5 вразливий до деяких атак, наприклад, створення двох повідомлень з однаковою хеш-сумою [14].

1.5.3 Хеш-функція SHA-1

Алгоритм безпечного хешування 1 (SHA-1) – алгоритм криптографічного хешування, описаний RFC 3174. Отримуючи вхідне повідомлення довільної довжини, алгоритм генерує 160-бітове хеш-значення, яке подається на вихід алгоритму. SHA-1 використовує велика кількість криптографічних додатків та протоколів. Попри наявність удосконалених версій (SHA-2, SHA-3), алгоритм входить до переліку рекомендованих алгоритмів хешування для установ США. Заснований на принципах, покладених в алгоритмі MD4 Рональдом Рівестом.

Алгоритм роботи SHA-1 також нагадує алгоритм MD5: вхідний текст розбивається на блоки по 512 біт. В останньому блоці даних відбувається таке доповнення: спочатку додається одиниця, а потім нулі, щоб довжина блоку стала розміром 448 біт. У решту 64 біти, яких не вистачатиме до значення в 512 біт, додається довжина повідомлення в бітах. Таке доповнення є обов'язковим навіть тоді, коли вхідне повідомлення навіть спочатку має необхідний розмір (в такому разі розмір доповнення складає 512 біт). Також даний алгоритм ініціалізує такі допоміжні змінні:

- A = a = 0x67452301
- B = b = 0xEFCDAB89
- C = c = 0x98BADCFE
- D = d = 0x10325476
- E = e = 0xC3D2E1F0

Головний цикл F протягом всього алгоритму оброблює кожен 512-бітовий блок. Алгоритм SHA-1 виконується в чотири етапи по двадцять операцій у кожному [16]. Принцип роботи однієї операції в алгоритмі SHA-1 представлено нижче (рис. 1.3).

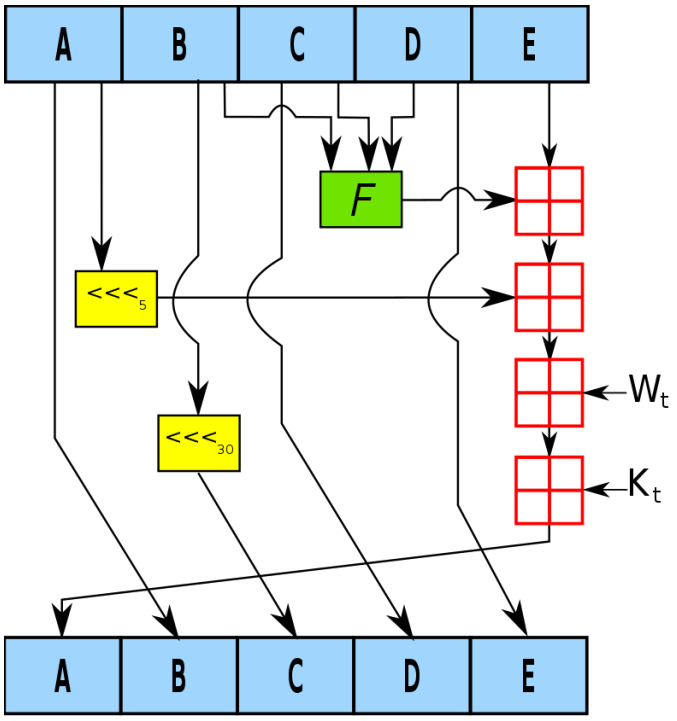

Рисунок 1.3 – Схематичне представлення принципу роботи однієї операції алгоритму SHA-1

1.5.4 Хеш-функція SHA-2

Алгоритм безпечного хешування 2 (SHA-2) — це криптографічний алгоритм комп'ютерної безпеки. Він був створений Агентством національної безпеки США (АНБ) у співпраці з Національним інститутом науки і технологій (NIST) як удосконалення алгоритму SHA-1. SHA-2 має шість різних варіантів, які відрізняються пропорційно розміром бітів, що використовуються для шифрування даних:

- SHA-224
- SHA-256
- SHA-284
- SHA-512
- SHA-512/224
- SHA-512/256

Число в кожному варіанті представляє значення бітів. SHA-2 забезпечує кращу захист від зіткнень, тобто ті самі вхідні дані завжди мають різне хеш-значення. SHA-2 використовує від 64 до 80 раундів криптографічних операцій і зазвичай використовується для перевірки та підпису цифрових сертифікатів безпеки та документів[17].

Сімейство алгоритмів хешування SHA-2 є найбільш поширеними функціями хешування. SHA-256 особливо широко поширений . Ці хеш-функції часто задіяні в основних механізмах безпеки, які допомагають захистити наше повсякденне життя. Можливо, ви ніколи цього не помічали, але SHA-2 є скрізь.

Наприклад, SHA-2 бере участь у багатьох протоколах безпеки, які допомагають захистити більшу частину нашої технології та виконує всі призначення, властиві хеш-функціям (див. 1.2 Властивості хешування) [18].

В основі хеш-функцій сімейства SHA-2 лежить структура Меркла-Демґарда. Після отримання вхідного повідомлення, алгоритм розбиває дані на блоки по 16 слів. Кожен блок алгоритм пропускає через цикл з 64 або 80 раундів (в залежності від різновиду). Кожен раунд перетворює 2 слова в даному алгоритмі, що

відбувається за участі інших слів. В результаті всіх перетворень сума всіх слів і дає остаточне хеш-значення повідомлення.

SHA-2 використовує такі прості бітові операції:

- ‖ — Конкатенація,
- + — Додавання,
- And — Побітове «І»,
- Or — Побітове «АБО»,
- Xor — Виключне «АБО»,
- Shr (Shift Right) — Логічний зсув вправо,
- Rotr (Rotate Right) — Циклічний зсув вправо [19].

1.5.5 Хеш-функція RIPEMD

RIPEMD (RACE Integrity Primitives Evaluation Message Digest) — це група хеш-функцій, розроблена Гансом Доббертіном, Антоном Босселаерсом і Бартом Пренілом у 1992 році. Ідея розробки RIPEMD заснована на MD4, який сам по собі є слабкою хеш-функцією. Він розроблений для роботи з 32-розрядними процесорами. Типи RIPEMD:

- RIPEMD-128
- RIPEMD-160
- RIPEMD-256
- RIPEMD-320

Перший RIPEMD не вважався хорошою хеш-функцією через деякі недоліки дизайну, які призводять до серйозних проблем із безпекою, одна з яких полягає у розмірі виводу, що становить 128 біт, який занадто малий і його легко зламати. У наступній версії RIPEMD-128 недолік дизайну усунено, але вихідний результат все ще 128 біт, що робить його менш безпечним.

RIPEMD-160 - це наступна версія, яка збільшує вихідну довжину до 160 біт і підвищує рівень безпеки хеш-функції. Ця функція призначена для роботи як заміна 128-бітним хеш-функціям MD4, MD5 і RIPEMD-128.

RIPEMD-256 і RIPEMD-320 є розширенням RIPEMD-128, які забезпечують таку ж безпеку, як RIPEMD-160 і RIPEMD-128, що розроблені для додатків, які віддають перевагу більшому хеш-значенню, а не більшому рівню безпеки.

Розглянемо підблок хеш-алгоритму RIPEMD-160, що зображено нижче (рис. 1.4). Повідомлення обробляється функцією стиснення в блоках по 512 біт і передається через два потоки цього підблоку за допомогою 5 різних версій, у яких значення константи 'k' також відрізняється [20].

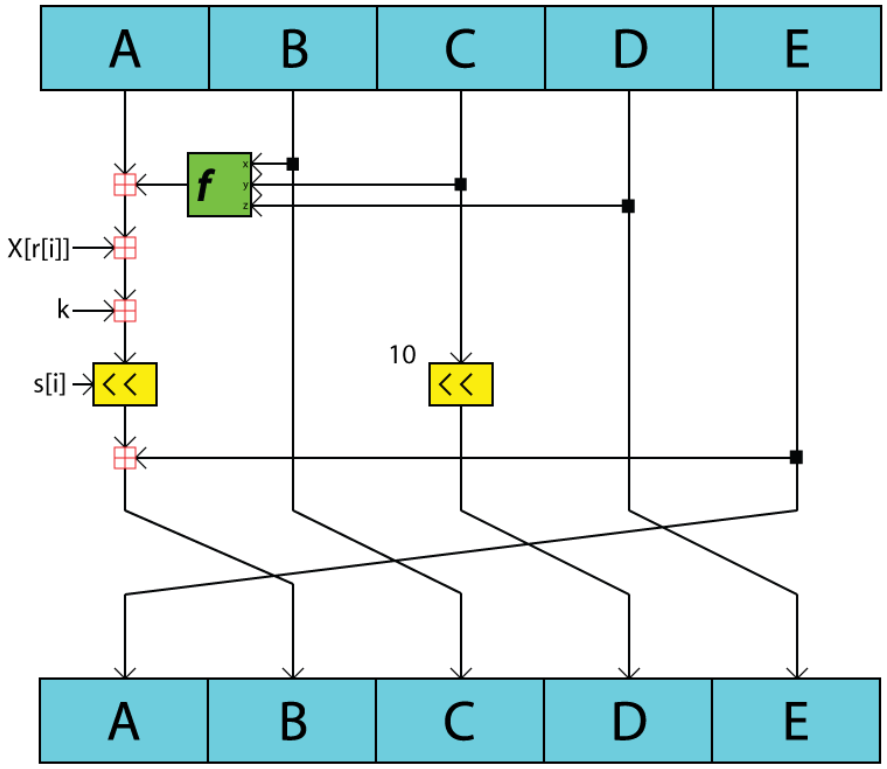

Рисунок 1.4 – Схематичне представлення принципу роботи однієї операції алгоритму RIPEMD-160

1.5.6 Хеш-функція BLAKE2

BLAKE2 — це криптографічна хеш-функція, яка швидша за MD5, SHA-1, SHA-2 і SHA-3 , та принаймні настільки ж безпечна, ніж останній стандарт SHA-3. Завдяки високій швидкості, безпеці та простоті BLAKE2 було використано багатьма проектами.

BLAKE2 представлений у двох варіантах:

- BLAKE2b (або просто BLAKE2) оптимізовано для 64-розрядних платформ, включаючи ARM з підтримкою NEON. Він створює дайджести будь-якого розміру від 1 до 64 байт.
- BLAKE2s оптимізовано для 8-32 -розрядних платформ і створює дайджести будь-якого розміру від 1 до 32 байт.

BLAKE2b швидше за такі популярні алгоритми, як MD5, SHA-1, SHA-2 і SHA-3, на 64-розрядних архітектурах x86-64 і ARM, що продемонстровано нижче(рис. 1.5).

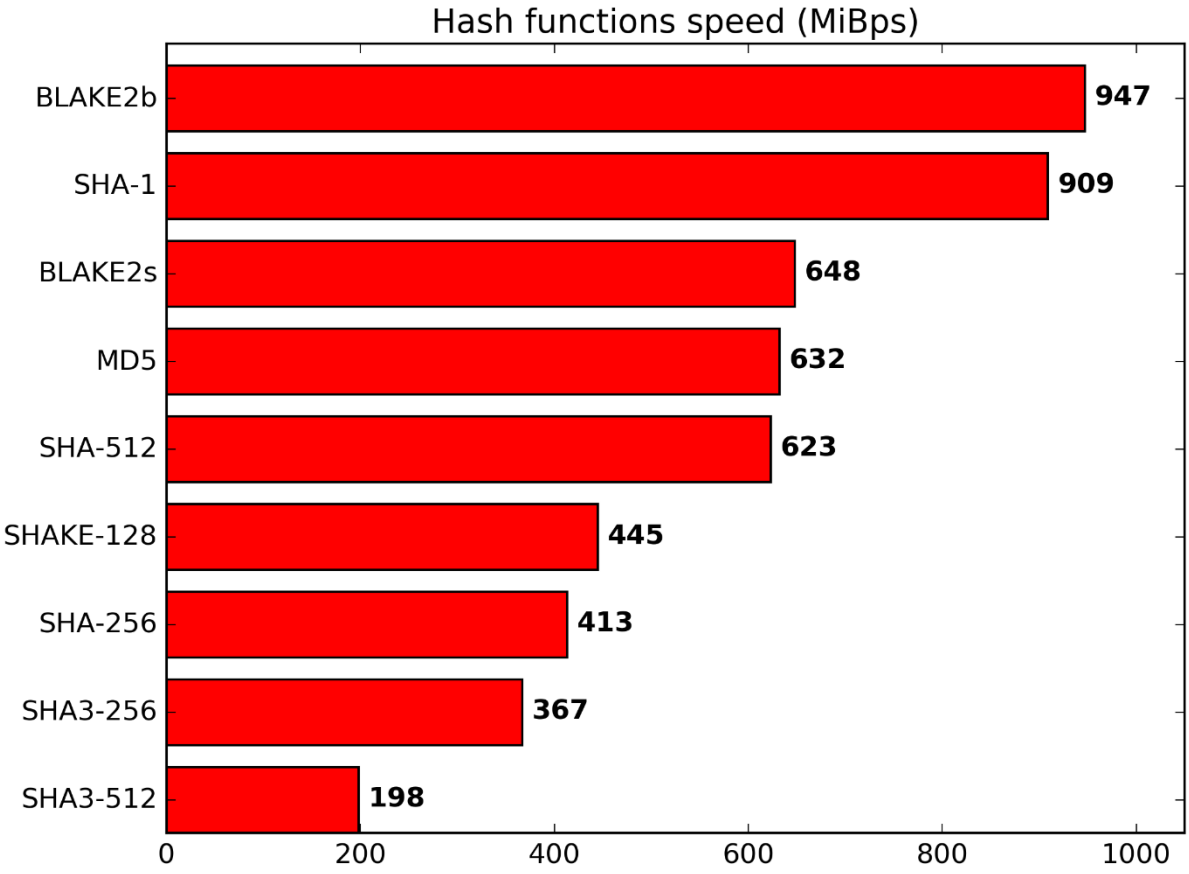

Рисунок 1.5 – Швидкість популярних хеш-алгоритмів (в MiBps)

BLAKE2 включає підтримку режимів набору ключів, додавання солі, персоналізації алгоритму та використання хеш-дерев. Існують декілька різновидів BLAKE2, що були розроблені для підвищення продуктивності на багатоядерних процесорах, серед яких BLAKE2bp і BLAKE2sp.

Алгоритм цієї функції розроблений на основі стурктури ChaCha. BLAKE2 видаляє додавання констант до слів повідомлень із функції раунду BLAKE, змінює

дві константи обертання, спрощує заповнення, додає блок параметрів, який об’єднується XOR за допомогою векторів ініціалізації, і зменшує кількість раундів з 16 до 12 для BLAKE2b (наступника BLAKE-512) та з 14 до 10 для BLAKE2 (наступник BLAKE-256).

Також існує удосконалена версія BLAKE3, яка була презентована в 2020 році [21].

Висновки за розділом 1

Хешування – це процес перетворення вхідних даних в хеш-значення фіксованої довжини. Існує безліч варіантів використання хеш-сум: від перевірки інформації ідентичність до оригіналу до прискорення пошуку по базам даних та перевірки на наявність потенційних загроз.

Будь-яка хеш-функція має задовольняти 2 основні вимоги виконання алгоритму: забезпечувати як можна меншу ймовірність виникнення колізій та мінімалізувати навантаження на обчислювальні можливості.

Існує безліч різновидів хеш-функцій, проте більшість відомих алгоритмів намагаються відповідати стандартам криптографічних хеш-функцій, що, окрім двох вищезазначених вимог, мають забезпечувати незворотність та стійкість до колізій першого та другого родів свого алгоритму. До таких хеш-функцій відносяться такі алгоритми як MD4, MD5, SHA-1, SHA-2, RIPEMD, BLAKE2 та інші.

РОЗДІЛ 2
ВРАЗЛИВОСТІ ХЕШ-ФУНКЦІЙ

2.1 Атаки на хеш-алгоритми

Як вже було вказано в розділі 1, двома основними вимогами, що повинні забезпечуватись хеш-алгоритмами є якомога менше навантаження на «систему» та мінімалізація ймовірності виникнення колізій. Саме мінімалізація, адже практика показує, що поки ні одна з відомих хеш-функцій не змогла досягти відсутність колізій взагалі. І якщо показник навантаження на обчислювальні пристрої залежить здебільшого від технічних компонентів, що менше враховується при побудові хеш-функцій, то саме ймовірність виникнення колізій стала основою оцінки якості хеш-функцій.

І не даремно. Виникнення колізій в хеш-функції може викликати не лише певні незручності в плані ідентифікації даних та заплутаність при перевірці хеш-значень, але й слугувати плацдармом для потенційних атак на алгоритми хешування. А, як показує практика, до таких та інших типів атак можуть бути вразливими навіть найбільш криптостійкі хеш-функції.

Крім колізійних атак, також часто використовуються аналітичні та загальні атаки.

До аналітичних атак відносяться: атака "зустріч посередині", атака з корекцією блоку, атака з фіксованою точкою, атака на базовий алгоритм шифрування, диференціальний аналіз. Проте такі атаки не направлені на пошук ключа, а найчастіше беруть за умову, що ми вже маємо вхідне повідомлення або доступ до хеш-функції в момент роботи алгоритму.

Загальні атаки – це атаки типу «брутфорс», атаки словникового перебору, що здатні перебирати всі значення паролів до моменту отримання потрібного значення.

Основною ціллю більшості атак на хеш-функції є визначення є виявлення ключа (вхідного повідомлення) функції.

2.1.1 Атака «днів народження»

Дана атака побудована на основі парадоксу днів народження, який твердить, що ймовірність збігу числа та місяця народження принаймні у 2 людей в групі з 23 чоловік рівна приблизно 50%. Цей парадокс побудований на математичних обрахунках ймовірностей та статистично доводить свою істинність.

У криптоаналізі під атакою «днів народження» розуміють метод злому шифрів або пошуку колізій хеш-функцій на основі парадоксу днів народження. Для заданої хеш-функції f метою атаки є пошук колізії другого роду. Для цього обчислюються значення для випадково вибраних блоків вхідних даних до тих пір, поки не будуть знайдені два блоки, що мають один і той самий хеш. Таким чином, якщо f маєм N різних рівноймовірних вихідних значення, і N є досить великим, то з парадоксу про дні народження випливає, що, в середньому, після перебору $1{,}25\times\sqrt{N}$ різних вхідних значень, буде знайдена колізія.

До цієї атаки, наприклад, може бути вразливим електронний цифровий підпис[22]. Припустимо, що 2 людини А і Б хочуть підписати контракт, але А хоче підсунути Б підроблений варіант контракту. Тоді А складає справжній договір та підроблений договір. Далі, внесенням допустимих змін не змінюють сенс тексту (розстановкою ком, переносів, відступів), А добивається, щоб обидва контракти мали однаковий хеш. Після чого А надсилає Б справжній контракт, Б його підписує, а його підпис також показує, що він підписав і підроблений контракт, так як обидва контракти мають однаковий хеш.

Для запобігання вразливості такого роду достатньо збільшити довжину хешу настільки, щоб стало обчислювально складно знайти 2 контракти з однаковими хешами.

Зокрема, потрібна вдвічі більша довжина хешу, щоб забезпечити обчислювальну складність, порівнянну зі складністю повного перебору. Іншими словами, якщо за допомогою повного перебору зловмисник може вирахувати 2^n хеш-значень, то він почне шукати хеш-колізії для всіх хешів довжиною менше 2n біт [23].

2.1.2 Атака за допомогою райдужної таблиці

Райдужна таблиця – це різновид таблиць пошуку, який побудований на механізму розумного компромісу між часом пошуку по таблиці. Райдужні таблиці використовуються для розкриття паролів, перетворених за допомогою незворотної хеш-функції, а також для атак на симетричні та асиметричні шифри на основі відомого відкритого тексту.

Основою створення райдужних таблиць є побудова ланцюжків потенційних паролів. Будь-який ланцюг в такій таблиці складається з випадкового можливого пароля, його хеш-значенню після дії хеш-функції та функції редукції. Ця функція перетворює результат хеш-функції на певний можливий пароль (наприклад, якщо ми припустимо, що пароль має довжину 64 біта, то функцією редукції може бути взято перші 64 біт хеша, побітове складання всіх 64-бітних блоків хешу і тому подібне). Проміжні паролі в ланцюжку відкидаються і таблицю записуються лише перший і останній елементи ланцюжків. Створення таких таблиць вимагає більше часу, ніж потрібно для створення звичайних таблиць пошуку, але значно менше пам'яті (аж до сотень гігабайт, при обсязі для звичайних таблиць у N слів для райдужних потрібно всього порядку N 2/3). При цьому вони вимагають хоч і більше часу (порівняно зі звичайними методами) на відновлення вихідного пароля, але на практиці більш реалізовані, адже містять на 1/3 частину блоків менше.

Для відновлення пароля значення хеш-функції піддається функції редукції і шукається в таблиці. Якщо не було знайдено збігу, то знову застосовується хеш-функція та функція редукції. Ця операція триває доки буде знайдено збіг. Після знаходження збігу ланцюжок, що містить його, відновлюється для знаходження відкинутого значення, яке буде шуканим паролем.

У результаті виходить таблиця, яка може з високою ймовірністю відновити пароль за короткий час.

Доцільність побудови такої таблиці визначається найкращим співвідношенням таких параметрів:

- вірогідність знаходження пароля за отриманими таблицями;

- часу побудови таблиць;
- час підбору пароля за таблицями;
- місце, що займає таблиця.

В свою чергу, ці параметри змінюватимуться в залежності від налаштувань побудови таблиці:

- символів, що дозволяються у варіантах паролю;
- максимальна кількість символів в паролі;
- максимальна довжина ланцюжка;
- кількість створюваних таблиць.

Незважаючи на те, що використання райдужних таблиць має ряд переваг, відносно застосування методу грубої сили, проте навіть в такому випадку при певних налаштуваннях побудови таблиці обчислювальних потужностей буде недостатньо, щоб згенерувати всі ланцюги та виконати пошук по таблиці. До прикладу, якщо довжина паролю, що може складатися з букв, цифр та спецсимволів, більше за 8 символів і його захешовано алгоритмом MD5, то таблиця може мати такі параметри:

- довжина ланцюжка 1400
- кількість ланцюжків 50 000 000
- кількість таблиць 800

Ймовірність знаходження паролю за такою таблицею буде рівна 0.7542 (75.42%), розмір таблиць перевищуватиме 596 Гб, генерація таблиць на комп'ютерах рівня Пентіум-3 1ГГц буде займати 3 роки, а для знаходження 1 паролю по цих таблицях буде необхідно 22 хвилини.

Захиститись від атаки подібного типу можна шляхом використання в хеш-функціях солі. Для підбору пароля такого типу зловмиснику необхідні таблиці для всіх можливих значень солі. Однак слід сказати, для того, щоб використовувати цей метод необхідна сіль довжиною 6-8 символів, яка буде індивідуальною для кожного паролю. Такий метод дозволить зробити пароль більшим та стійкішим. Якщо таблиця розрахована на деяку довжину або на певний обмежений набір символів, сіль може запобігти відновленню пароля [24, 25].

2.1.3 Атака грубої сили

Брутфорс – це тип атаки, при якому перебором співставляються всі можливі значення ключа або хеш-значення. Такий тип атаки ефективний, коли, до прикладу, атака здійснюється на простий хешований пароль або у випадку, коли у функції існує відносно невелика кількість можливих ключів.

В загальному випадку, ця атака є досить громіздкою в обчислювальному плані та може займати дуже багато часу при умові, що пароль є достатньо надійним.

Тож, найкращим захистом від цієї атаки, в разі хешування паролю, є вибір надійного, великого за розміром та несловарного паролю, що змусить атаку займати дуже багато часу та майже унеможливить підбір значення таким чином.

2.1.4 Атака по словнику

Атака по словнику – це тип атаки, в якому перебираються не всі можливі значення функції, а ті, які внесені до певного словника (набору можливих аргументів). Цей тип атаки, в порівнянні з атакою грубої сили, більше побудований на аналізі паролів та дозволяє зменшити час перебору можливих значень за рахунок відсікання малоймовірних варіантів.

Мінусом такого типу атак, у порівнянні з брутфорсом, є не достовірність отримання результату. Якщо в атаці грубої сили в ідеальному випадку (нехтуючи часом виконання перебору) ми в 100% випадків отримаємо значення ключа, то в атаці по словнику необхідне значення може бути не внесено до переліку аргументів, тому така атака результату не дасть.

Ще однією умовністю такої атаки є створення словника паролів. Для цього необхідно мати можливість оцінки ймовірності використань певних значень паролів жертвою. Звісно, на просторах Інтернету існує безліч базових словників паролів, в які внесені значення, що зустрічаються найбільш часто. Проте таких значень існує дуже багато, тож зібравши словник з їх усіх така атака уподібнюється до атаки грубої сили, що означає потенційно значний час виконання атаки. І, знову ж таки,

навіть такі словники не дають нам стовідсоткової впевненості, що пароль буде знайдено.

Захистом від такого типу атаки є використання великих та надійних паролів з використанням несловникових слів та спецсимволів, що фактично унеможливлює підбір паролю таким чином.

2.2 Зберігання та використання хешів

Вразливості для багатьох хеш-сум можуть виникнути не лише в процесі хешування вхідного повідомлення, а й в процесі використання або зберігання вже готових хеш-значень. Ніколи не можна бути впевнений, що твої дані знаходяться в безпеці та що до них ніхто не зможе отримати доступ.

Одним із найбільш недоцільних способів зберігання хеш-суми є її зберігання або надсилання разом з оригінальним вхідним повідомлення, з якого була й «знята» хеш-сума. В такому разі під потенційну загрозу потрапляє не лише хеш-сума, яка зможе в майбутньому піддатися криптоаналізу для знаходження можливих колізій, а й безпосередньо вхідне дані, до яких зловмисник намагається отримати доступ. Таким чином, це значно полегшить задачу атакуючій стороні, дозволяючи їй виконувати будь-які операції: зміну оригінального повідомлення з подальшою заміною значення хеш-суми, заміну оригінального повідомлення методом підбору колізії без зміни хеш-суми, крадіжку даних та «злиття» їх хеш-суми і таке інше. Тож, зберігати та надсилати хеш необхідно роздільно від основного повідомлення (якщо це стосується конфіденційної інформації), щоб не давати можливість зловмиснику дістати доступ та редагувати дані, не видаючи своєї присутності.

Використання хешів також може мати вразливості. Здебільшого, це знову пов’язано з потенційним знаходженням колізій. До прикладу, багато систем аутентифікації для перевірки паролю використовують порівняння значення не оригінального паролю з введеним, а порівняння їх хеш-сум. Таким чином, зловмиснику достатньо мати не оригінальний ключ, а лише «його рисунок», що

також дозволить йому пройти процес аутентифікації та отримати доступ до інформації.

Системи з такими технологіями мають використовувати хеш-алгоритми з мінімальною ймовірністю виникнення колізій та безпечним зберігання хеш-значень. До того ж, доцільно б виглядало порівняння не лише повних хеш-сум оригінального та введеного паролів, а й, подібно до принципу дії сигнатурного аналізу в антивірусних програмах, порівняння окремих частин паролю, що значно б підвищувало безпеку та якість такої системи порівняння паролів та удосконалювало сам алгоритм перевірки.

2.3 Методи захисту алгоритмів хешування

Захист алгоритмів хешування може бути пов’язаний як із захищеністю хеш-функцій, які застосовуються в процесі хешування, так із захистом вхідних і вихідних даних. Річ в тому, що у випадку хешування паролів, який би алгоритм хешування ми не вибрали, ненадійний та простий пароль може стати легкою ціллю для зловмисників. Проте, це ж стосується і зворотньої ситуації, коли ми використовуємо складний пароль, проте хешуємо його за допомогою алгоритму з великою ймовірністю виникнення колізій. Тому лише комплексність методів та технологій забезпечення загальної безпеки дозволить гарантувати криптостійкість функції хешування та отримати надійну хеш-суму.

Таким чином, слід дотримуватись таких кроків та рекомендацій щодо використання хеш-функцій та захисту вхідного та вихідного повідомлень:

- Використовувати криптографічні хеш-функції.

Як ми вже переконалися, криптостійкість та надійність алгоритму хешування є запорукою не лише складної для криптоаналізу вихідної хеш-суми, але й безпечності даних в цілому. Не даремно абсолютна більшість популярних хеш-функцій дотримуються властивостей криптографічних хеш-алгоритмів. Це дозволить убезпечити себе від більшості популярних атак на хеш-алгоритми або ж зробити час їх виконання непомірно великим.

- Оптимізувати безпеку зберігання хеш-сум.

Звісно, деякі хеш-суми можуть слугувати, до прикладу, для порівняння інформації або для пошуку по базі даних. Такі хеш-суми не завжди потребують безпеки зберігання, адже вони не згенеровані з конфіденційної інформації. Проте є і інший поширений випадок: хеш-суми, що отримані в результаті хешування паролю. Зберігання таких хеш-сум також має виконувати необхідні норми безпеки.

Деякі програмні засоби, що спеціалізуються генерації та збереженні паролів, зберігають їх не у відкритому вигляді, а додатково шифрують хеш-суми або розбивають їх для зберігання в різних місцях окремих частин. Такий спосіб є досить дієвим, адже він зробить задачу зловмисника скомпрометувати таке хеш-значення набагато складнішою та, в плані часу, затратнішою. В такому випадку, атакуючій стороні, крім безпосереднього отримання вхідного повідомлення з хеш-значення або пошуку колізій до нього (що також в більшості випадків є досить важкою задачею) необхідно розшифрувати та з’єднати значення хешу, що потребує значних зусиль.

Також слід згадати про зберігання хеш-сум важливої інформації окремо від безпосередньо даних, з яких вона була згенерована. В іншому випадку, це значно полегшить завдання зловмиснику при отриманні доступу до такого місця зберігання (дивись пункт 2.2 Зберігання та використання хешів).

- Обирати надійний пароль, що буде згенерований в хеш-значення.

Цей пункт також заслуговує уваги, адже ми вияснили, що яким би надійним не був алгоритм хешування, однак здійснити взлом хешів по ньому є можливою задачею. Особливо реалістична вона для «слабких» паролів. У відкритому доступі містить безліч баз даних, де представлені хеш-значення популярних паролів різних алгоритмів хешування. Таким чином, зловмисник може за лічені хвилини, використовуючи таку базу даних, компрометувати пароль такої хеш-функції.

Тож, надійність вхідних даних в хеш-функцію не менш важлива, аніж надійність самої функції. Тому треба дотримуватись основних правил забезпечення надійності паролю: він має містити несловникові значення, літери верхнього та нижнього регістрів, спецсимволи та довжину від 8 символів. Таким чином можна

забезпечити безпеку не лише безпосередньо паролю, а й хеш-значення, що буде згенеровано з нього.

- Використовувати подвійне хешування.

Цей метод подібний до шифрування хеш-суми, про яке було згадано вище (див. маркер «Оптимізувати безпеку хеш-сум»). Він також дозволить підвищити надійність та рівень безпеки хеш-функції та вихідного хеш-значення. Найкраще для подвійного хешування використовувати дві різні функції, що будуть виконані послідовно. Таким чином, отриманий хеш-код все ще можна використовувати для всіх необхідних завдань (див. пункт «1.2 Призначення хешування»), а отримати вихідні значення за допомогою криптоаналізу стане щонайменше в 2 рази важче.

- Використовувати динамічну сіль в хеш-функціях.

Використання солі в алгоритмі хешування дозволить зробити хеш-функцію надійнішою та зменшити ймовірність виникнення колізій. При цьому, це досягається краще за рахунок солі, що залежатиме від вхідних даних, а не її статичного значення. Однак в такому випадку слід пам’ятати про детермінованість функції хешування: для одного і того ж значення ключа алгоритм повинен генерувати те ж саме значення хеш-коду в будь-який момент виконання операції. Тому необхідно виключити будь-яку неконтрольовану випадковість при генерування солі для хеш-функції.

Висновки за розділом 2

Існує безліч потенційних атак на хеш-алгоритми. Більшість з них націлені на отримання вхідного повідомлення (ключа), яке було передано для хешування. Всі ці атаки можна поділити на 3 типи:

- Загальні: атаки по словнику, атаки грубої сили, повний перебір значень.
- Аналітичні атаки: атака "зустріч посередині", атака з корекцією блоку, атака з фіксованою точкою, атака на базовий алгоритм шифрування, диференціальний аналіз.

- Колізійні атаки: атака «днів народження», атака за допомогою райдужних таблиць та інші.

Також слід замислюватися над безпекою хеш-значень не лише підчас, а й після процесу хешування. Неправильне зберігання або передавання хеш-сум може полегшити задачу зловмисникам та скомпрометувати дані.

Щоб цього не сталося необхідно дотримуватись основних кроків та рекомендацій щодо забезпечення захисту алгоритмів хешування:

- Використовувати криптографічні хеш-функції.
- Оптимізувати безпеку зберігання хеш-сум.
- Обирати надійний пароль, що буде згенерований в хеш-значення.
- Використовувати подвійне хешування.
- Використовувати динамічну сіль в хеш-функціях.

РОЗДІЛ 3
ПОБУДОВА УДОСКОНАЛЕНОГО АЛГОРИТМУ ХЕШУВАННЯ НА ОСНОВІ MD5

3.1 Постановка завдання

MD5 – алгоритм, який широко використовується в наш час для перевірки цілісності інформації та гарантування правильної передачі даних. Попри це, ще з 2011 року в RFC6151 хеш-функція MD-5 отримала статус не надійною з міркувань безпеки та не придатної до використання в завданнях та продуктах, де потрібна стійкість до колізій (зокрема, в цифрових підписах) [26].

Це, перш за все, пояснюється надмірною чутливістю алгоритму до атак зіткнення та високою ймовірністю колізій, як для криптографічних хеш-функцій. Сучасні комп'ютери здатні за лічені секунди знаходити пари однакових хеш-сум MD5, використовуючи різні вхідні повідомлення. Попри це, відмова від алгоритму відбувається досить повільно та неохоче, незважаючи на всі його недоліки.

Таким чином, виникає завдання удосконалити даний алгоритм, зберігаючи його зручність та практичність, однак зменшуючи ймовірність утворення колізій, не завдаючи великої шкоди криптостійкості алгоритму. Саме таке завдання було поставлено при побудові удосконаленого алгоритму на основі MD5.

3.2 Опис програмної реалізації

Реалізація програмного алгоритму удосконаленої хеш-функції була реалізована на мові програмування Python версії 3.10.3. Тип застосунку – консольний додаток, який приймає значення тексту від користувача та виводить його хеш-значення. За середовище програмування було обрано програмний продукт PyCharm 2021.3.3 (Community Edition), встановлений на операційній системі Windows 11 Home збірки 22000.258. Повний код програмного застосунку представлено в Додатку А.

Після запуску коду спочатку оголошуються основні змінні та функції, які будуть використовуватись в процесі виконання програми. Стосовно виконання, то найперше програма нам повідомляє, що найефективніше та найбільш безпечно використовувати цей алгоритм для паролів та повідомлень від 12 символів, виводячи це повідомлення в консоль. Далі застосунок пропонує ввести нам значення вхідного повідомлення, яке буде конвертовано в хеш.

Після введення значення і підтвердження дії, застосунок звертається до функції «md5». У ній вираховується довжина текстового повідомлення в бітовому значенні, додається до нього одиничний біт та необхідну кількість нульових бітів, щоб довжина потоку стала рівна 448 по модулю 512. Надалі в цій функції відбуваються раундові операції, що відповідають операціям алгоритму MD5. В результаті них в головну функцію повертається строка, що відповідає результату алгоритму md5 в числовому десятирічному значенні.

Наступним кроком необхідно конвертувати це значення в шістнадцятирічну систему для отримання значення хеш-коду MD5. Для цього відбувається звернення до функції «md5_to_hex», де наше значення спочатку подається у вигляді масиву байтів довжиною 16 символів, після чого повертається у вигляді hex-значення.

Далі алгоритм має за ціль додавання солі на початок та кінець хеш-значення. Це відбувається за допомогою звернення до функції «salt». В ній ми спочатку, при наявності, транслітеруємо символи кирилиці у вхідному повідомленні в латиницю та замінюємо в ньому всі символи пунктуації на «0». Додатково, ми змінюємо значення другого та передостаннього символу вхідного тексту наступним чином: спочатку переводимо символ в числове значення за таблицею ASCII, додаємо до нього довжину вхідного тексту, після чого, переконавшись, що отримане значення в ASCII відповідає значення малої літери латиниці чи цифри, конвертуємо його назад та змінюємо оригінальний символ повідомлення на отримане значення. Тоді алгоритм обчислює довжину вхідного повідомлення після чого виконує наступні дії:

- Якщо довжина повідомлення рівна 0 символів (на вхід було подано пусте повідомлення), то на початку і в кінці хеш-значення буде додано по два «0».

- Якщо довжина повідомлення рівна 1 символ, то його двічі буде додано на початок та в кінець хеш-значення.
- Якщо довжина повідомлення 2 і більше символів, то на початок додаються перший та останній символ цього повідомлення, а в кінець – другий та передостанній (символи, що ми змінювали за допомогою таблиці ASCII).

Після завершення додавання солі, оновлене значення хеш-функції повертається в основну функцію програми та виводиться користувачу в консоль, по завершенні чого застосунок припиняє свою роботу.

3.3 Обґрунтування техніки удосконалення

Як вже було зазначено (див. пункт «3.1 Постановка завдання»), основною проблемою алгоритму MD5 є висока ймовірність колізій, а основною перевагою – легка програмна реалізація та висока практичність застосування. Тож, доцільним було рішення зберегти незмінною основну функцію алгоритму, таким чином не нівелюючи його переваги, а для зменшення випадків колізії додати до алгоритму 4 символи, що виконують роль динамічною солі, тобто солі, значення якої напряму залежить від вхідного повідомлення.

Сіль буде виступати своєрідним ідентифікатором хеш-коду, додаючи на його початок перший та останній символ вхідного повідомлення, а в кінець – другий та передостанній символи, які за допомогою перетворення в числове значення таблиці ASCII, сумі цього значення з довжиною вхідного повідомлення та зворотнього перетворення, замінені на інші символи. Таким чином, перші два символи слугуватимуть своєрідним «оригінальним» ідентифікатором, які, хоч і розкриють ці символи в хеш-коді, проте, через відсутність додаткових операцій з ними, «відсікають» ймовірність виникнення додаткових колізій. Останні ж символи створюють залежність підсумкового хеш-значення від довжини коду, що також зменшить ймовірність виникнення колізій.

Проте таке рішення викликає додаткові умови. Так, при хешуванні цим удосконаленим методом паролів рекомендована довжина паролю має складати не

від 8 символів, а від 12, адже перші та останні 2 символи будуть представлені в хеш-сумі, тому не будуть представляти конфіденційний зміст.

Також було додано фільтри, що форматують символи вхідного повідомлення, що будуть додані до хеш-значення. Так, якщо значення першого, другого або хоча б якогось з двох останніх символів повідомлення є знаком пунктуації, то його місце в хеш-значенні буде займати «0».

Ще одним фільтром було додано транслітерацію символів кирилиці в латиницю для збереження загального вигляду хеш-суми на випадок, коли вхідне повідомлення задане кирилицею. Всі пари символів кирилиці та відповідні їм символи латиниці, що призначає застосунок, наведено нижче (табл. 3.1).

Таблиця 3.1

Транслітерація застосунком символів кирилиці в латиницю

Літера кирилиці (незалежно від регістру)	**Відповідне значення на латиниці**
«а»	«a»
«б»	«b»
«в»	«v»
«г»	«g»
«ґ»	«g»
«д»	«d»
«е»	«e»
«є»	«e»
«ж»	«j»
«з»	«z»
«и»	«y»
«і»	«i»
«ї»	«i»
«й»	«y»
«к»	«k»

продовження табл. 3.1

«л»	«l»
«м»	«m»
«н»	«n»
«о»	«o»
«п»	«p»
«р»	«r»
«с»	«s»
«т»	«t»
«у»	«u»
«ф»	«f»
«х»	«h»
«ц»	«c»
«ч»	«c»
«ш»	«w»
«щ»	«w»
«ь»	«q»
«ю»	«u»
«я»	«a»

Можна помітити, що деякі літери латиниці дублюються для різних значень кирилиці. Тож, з цього слідує висновок, що ймовірність виникнення колізій при використанні кирилиці в даному алгоритмі є дещо вищою, аніж за використання латиниці. Проте такий дуалізм деяких символів підвищує загальний рівень стійкості утвореної хеш-суми до криптоаналізу.

Висновки за розділом 3

MD5 – популярний алгоритм хешування, який використовується в наш час в багатьох процесах. Та попри свою популярність, він має дуже вагомий недолік –

високу ймовірність виникнення колізій, через що його навіть визнали «ненадійним» для використання в цифрових підписах.

Найкращий спосіб вирішення цієї проблеми, не руйнуючи загальний алгоритм хеш-функції – додавання додаткових символів (солі) до вже згенерованої хеш-суми, які будуть слугувати своєрідними ідентифікаторами хешу.

Саме таку технологію удосконалення було вибрано та реалізовано в програмному застосунку.

ВИСНОВКИ

Хешування є досить практичним елементом перевірки на цілісність та ідентичність даних, що незмінно використовується вже багато років. Дуже відповідально треба ставитися до вибору хеш-функції, адже від її надійності може залежити не лише безпека інформації, що хешується, а й вплив на інші процеси, через застосування технології хешування в цифрових підписах.

Будь-яка хеш-функція має відрізнятися легкістю виконання для обчислювальних пристроїв, криптостійкістю та мінімальною ймовірністю виникнення колізій.

В рамках даної роботи, було проведено аналіз алгоритмів хешування, розглянуто різноманітні хеш-функції, оцінено базові атаки на них, сформульовано список рекомендацій по вибору та застосуванню хеш-функцій та програмно реалізовано вдосконалений алгоритм хешування на основі популярної хеш-функції MD5.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Контрольна сума [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/Контрольна_сума
2. Хешування [Електронний ресурс]: Wiki ТНТУ. Відкриті знання та навчання. – Режим доступу: https://wiki.tntu.edu.ua/Хешування
3. Сигнатурний аналіз [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим доступу: https://ru.wikipedia.org/wiki/Сигнатурный_анализ
4. Хеш [Електронний ресурс]: Лайкни – Режим доступу: https://www.likeni.ru/glossary/khesh/
5. Хешування [Електронний ресурс]: UTMAG – Режим доступу: https://utmagazine.ru/posts/21321-heshirovanie
6. Хешування [Електронний ресурс]: Skill Factory. Блог. – Режим доступу: https://blog.skillfactory.ru/glossary/heshirovanie/
7. Ідеальне хешування [Електронний ресурс]: Wiki. Університет ІТМО. – Режим доступу: https://neerc.ifmo.ru/wiki/index.php?title=Идеальное_хеширование
8. Вергунова І.М. Основи комп’ютерних алгоритмів. Лекція. – Вінниця :ТВОРИ, 2021. – 228 с.
9. Djamal Belazzougui, Fabiano C. Botelho, Martin Dietzfelbinger. Hash, displace, and compress . — Springer Berlin / Heidelberg, 2009.
10. Універсальне хешування [Електронний ресурс]: StudAll – Режим доступу: https://studall.org/all4-3004.html
11. Cryptographic hash function [Electronic resource]: Wikipedia. The Free Encyclopedia. – Access: https://en.wikipedia.org/wiki/Cryptographic_hash_function
12. Криптографічна хеш-функція [Електронний ресурс]: Tebapit. – Режим доступу: https://tebapit.com/криптографічна-хеш-функція/
13. Прохорова О.В. Інформаційна безпека і захист інформації: підручник / О.В. Прохорова. – 2014 – 114 с.

14. Алгоритми хешування MD4 і MD5 [Електронний ресурс]: Ozlib. – Режим доступу: https://ozlib.com/974098/tehnika/algoritmy_heshirovaniyaMD5 [Electronic resource]: TechTarget. SearchSecurity. – Access: https://www.techtarget.com/searchsecurity/definition/MD5

15. SHA-1 [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/SHA-1

16. Security Hash Algorithm 2 (SHA-2) [Electronic resource]: techopedia. – Access: https://www.techopedia.com/definition/30571/secure-hash-algorithm-2-sha-2

17. What is SHA-2 and how does it work? [Electronic resource]: comparitech. – Access: https://www.comparitech.com/blog/information-security/what-is-sha-2-how-does-it-work/

18. SHA-2 [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/SHA-2

19. RIPEMD Hash Function [Electronic resource]: GeeksforGeeks. – Access: https://www.geeksforgeeks.org/ripemd-hash-function/

20. BLAKE2 — fast secure hashing [Electronic resource]: BLAKE2. – Access: https://www.blake2.net/

21. Знаходження ключа [Електронний ресурс]: Студопедія. – Режим доступу: https://studopedia.info/2-54388.html

22. Атака «днів народження» [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим доступу: https://ru.wikipedia.org/wiki/Атака_«дней_рождения»

23. Снегуров А.В., Чакрян В.Х. Аналіз стійкості до взлому сучасних механізмів парольного захисту операційних систем / Снегуров А.В., Чакрян В.Х. // Східно-Європейський журнал передових технологій. – 2011. - № 2. – С. 28

24. Райдужна таблиця [Електронний ресурс]: StudFiles. – Режим доступу: https://studfile.net/preview/2861883/page:2/

25. Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms. Internet Engineering Task Force (march 2011). – Access: https://datatracker.ietf.org/doc/html/rfc6151

ДОДАТОК А

ПРОГРАМНИЙ КОД РОЗРОБЛЕНОГО ЗАСТОСУНКУ

```
import math
import string

Transliteration = {
        'а':'a',
        'б':'b',
        'в':'v',
        'г':'g',
        'ґ':'g',
        'д':'d',
        'е':'e',
        'є':'e',
        'ж':'j',
        'з':'z',
        'и':'y',
        'і':'i',
        'ї':'i',
        'й':'y',
        'к':'k',
        'л':'l',
        'м':'m',
        'н':'n',
        'о':'o',
        'п':'p',
        'р':'r',
        'с':'s',
        'т':'t',
        'у':'u',
        'ф':'f',
        'х':'h',
        'ц':'c',
        'ч':'c',
        'ш':'w',
        'щ':'w',
        'ь':'q',
        'ю':'u',
        'я':'a'
    }

rotate_amounts = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
                  5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,
                  4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
                  6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]

constants = [int(abs(math.sin(i + 1)) * 2 ** 32) & 0xFFFFFFFF for i in range(64)]

init_values = [0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476]

round_func = 16 * [lambda b, c, d: (b & c) | (~b & d)] + \
             16 * [lambda b, c, d: (d & b) | (~d & c)] + \
             16 * [lambda b, c, d: b ^ c ^ d] + \
             16 * [lambda b, c, d: c ^ (b | ~d)]

index_func = 16 * [lambda i: i] + \
                  16 * [lambda i: (5 * i + 1) % 16] + \
                  16 * [lambda i: (3 * i + 5) % 16] + \
                  16 * [lambda i: (7 * i) % 16]

def left_rotate(x, amount):
    x &= 0xFFFFFFFF
```

```
    return ((x << amount) | (x >> (32 - amount))) & 0xFFFFFFFF


def md5(text):
    text = bytearray(text)
    bits_len = (8 * len(text)) & 0xFFFFFFFF
    text.append(0x80)
    while len(text) % 64 != 56:
        text.append(0)
    text += bits_len.to_bytes(8, byteorder='little')

    hash_md5 = init_values[:]

    for chunk_ofst in range(0, len(text), 64):
        a, b, c, d = hash_md5
        chunk = text[chunk_ofst:chunk_ofst + 64]
        for i in range(64):
            f = round_func[i](b, c, d)
            ind = index_func[i](i)
            to_rotate = a + f + constants[i] + int.from_bytes(chunk[4 * ind:4 * ind +
4], byteorder='little')
            new_b = (b + left_rotate(to_rotate, rotate_amounts[i])) & 0xFFFFFFFF
            a, b, c, d = d, new_b, b, c
        for i, value in enumerate([a, b, c, d]):
            hash_md5[i] += value
            hash_md5[i] &= 0xFFFFFFFF
    return sum(x << (32 * i) for i, x in enumerate(hash_md5))


def md5_to_hex(hashing):
    raw = hashing.to_bytes(16, byteorder='little')
    return '{:032x}'.format(int.from_bytes(raw, byteorder='big'))

def salt(text, inhex):
    text = text.lower()
    x = 0
    for i in text:
        if x == 0 or x == 1 or x == (len(text)-2) or x == (len(text)-1):
            if Transliteration.get(i) != None:
                text = text.replace(i, Transliteration.get(i))
            if i in string.punctuation:
                text = text.replace(i, '0')
            if x == 1 or x == (len(text)-2):
                symbol = ord(text[x]) + len(text)
                t = 0
                while t==0:
                    if symbol > 57 and symbol < 97:
                        symbol = 96 + (symbol - 57)
                    elif symbol > 122:
                        symbol = 46 + (symbol - 122)
                    else:
                        text = text[:x] + str(chr(symbol)) + text[x + 1:]
                        break
        x+=1
    if len(text) == 0:
        newhash = '00' + inhex + '00'
    elif len(text) == 1:
        newhash = str(text*2) + inhex + str(text*2)
    else:
        newhash = str(text[0]) + str(text[len(text)-1]) + inhex + str(text[1]) +
str(text[len(text)-2])
    return newhash

if __name__ == '__main__':
```

```
    print('Увага! Даний застосунок рекомендовано використовувати з паролями, довжина
яких не менше 12 символів!')
    text = bytearray(input('Введіть текст: '), 'utf-8')
    hashing = md5(text)
    inhex = md5_to_hex(hashing)
    update = salt(text.decode(), inhex)
    print(update)
```