НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

УДК 004.78:004.056.53

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	Завідувач кафедри
Інформаційних технологій	Комп'ютерних систем, мереж та кібербезпеки
_____ Глазунова О.Г., д.пед.н, проф.	_____ Касаткін Д.Ю., к.п.н., доцент.
підпис ПІБ, вчене звання і ступінь	підпис ПІБ, вчене звання і ступінь
«___»_____2022 р.	«___»_____2022 р.

МАГІСТЕРСЬКА РОБОТА

На тему: «Дослідження параметрів універсальної хеш-функції для прикладного застосування»

Спеціальність 123 «Комп’ютерна інженерія»

Освітня програма Комп’ютерні системи та мережі

Орієнтація освітньої програми _____

Керівник дипломного проекту: _____ Сагун А.В.
підпис ПІБ

Виконав: _____ Денисюк Ю.Т.
підпис ПІБ

КИЇВ-2022

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»
завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
_____ / Лахно В.А., д.т.н., проф. /
підпис ПІБ, вчене звання і ступінь
«__» _____ 20__ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТУ

Денисюк Юрій Тарасович
(прізвище, ім'я, по батькові)

Спеціальність (напрям підготовки): 123 - комп'ютерна інженерія
Освітня програма: комп'ютерні системи та мережі
Орієнтація освітньої програми: -

Тема магістерської роботи: «Дослідження параметрів універсальної хеш-функції для прикладного застосування»

затверджена наказом ректора НУБіП України від "11" лютого 2022 р. № 289
«С»Термін подання завершеної роботи на кафедру: 17 листопада 2022 р.

Вихідні дані до магістерської роботи: алгоритми та описи роботи раундів криптографічних функцій md5, SHA-1, алгоритмічна мова, «райдужні» таблиці для функцій md5, SHA-1.

Перелік питань, що підлягають дослідженню:

1. Огляд джерел за темою дослідження.
2. Дослідження та аналіз алгоритмів роботи хеш-функцій md5 та sha-1
3. Створення та програмна реалізація алгоритму універсальної хеш-функції мовою Python

Перелік графічного матеріалу (за потреби):

Дата видачі завдання "11" лютого 2022 р.

Керівник магістерської роботи _____ Сагун А.В., к.т.н., доцент
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Денисюк Ю.Т.
(підпис) (прізвище та ініціали студента)

ВСТУП

Хеш-функції мають багато напрямів практичного використання і дуже затребувані в техніці. Такі функції, перетворюють вхідні дані будь-якого розміру в послідовність бітів фіксованого розміру. Саме за току властивість їх називають функціями згортки. Хеш-функції мають застосування у структурах даних — так званих хеш-таблицях, які широко застосовуються у програмному забезпеченні для швидкого пошуку даних. Хеш-функції також використовуються з метою оптимізації таблиць та баз даних різної структури та організації. Це зручно, тому, що в однакових записів однакові значення хеш-функції. Такий підхід пошуку дублікатів ефективний у файлах великого розміру. Особливо часто і багато хеш-функції застосовуються в криптографії та системах автентифікації.

Тобто, зараз є багато практичних застосувань хеш-функцій. Існує тенденція до збільшення областей їх застосування [1-5].

Актуальність роботи пов’язана з необхідністю створення універсальної криптостійкої хеш-функції. Така функція повинна мати просту структуру та може бути легко реалізованою програмно та апаратно за рахунок наявності в ній простих операцій бітових зсувів, XOR, бітової заміни.

Об’єктом роботи є криптографічна хеш-функція.

Предметом роботи є асиметрична криптографічна хеш-функція в бітовому відображенні.

Новизна роботи полягає в створенні та дослідженні параметрів хеш-функції, яку можливо застосувати для різних технічних задач.

Мета роботи полягає в розробці та аналізі роботи створеної хеш-функції, виявлення характеристик розподілу та можливих колізій хеш-сум через порівняння результатів роботи функції згортки на генерування колізій з програмними реалізаціями існуючих алгоритмів MD5 та SHA-1.

Для досягнення мети необхідно вирішити **задачі:**

– проаналізувати принципи роботи та виявити недоліків найбільш розповсюджених алгоритмів односторонніх хеш-функцій;

– здійснити реалізацію алгоритм швидких бітовик перетворень в межах створюваної хеш-функції та знайти шлях мінімізації кількості можливих колізій;

– сконструювати універсальну хеш-функцію, яка б оперувала значеннями в модульному числовому полі типу і базувалася б на розробленому математичному апараті;

– створити програмний засіб, що реалізує роботу створеного універсальної хеш-функції та отримати результати порівняння роботи отриманої функції з хеш-функціями типу md5 та sha-1.

Структура та обсяг роботи. Робота складається із вступу, 3 розділів, висновків, використаних джерел, що містить 20 найменувань на 3 сторінках, 23 рисунків та 5 таблиць та 1 додаток на 3 сторінках.

Загальний обсяг роботи – 45 сторінок, з них 42 сторінки основного тексту.

РОЗДІЛ 1

ОДНОСТОРОННІ КРИПТОПЕРЕТВОРЕННЯ ТА ФУНКЦІЇ, ЩО ЇХ РЕАЛІЗОВУЮТЬ

1.1 Принцип перетворень в хеш-функціях

Хеш-функція - це математичний алгоритм, який відображає дані довільного розміру в бітовий масив фіксованого розміру.

Результат, що генерується хеш-функцією, називається «хеш-сумою» або просто «хешом», а вхідні дані часто називають «повідомленням» [6]. Такі перетворення виконуються певним алгоритмом. Перетворення, яке здійснюється хеш-функцією, називається хешуванням. Вихідні дані такого перетворення називаються вхідним масивом, "ключом" або "повідомленням". Результат такого перетворення може називатися "хеш", "хеш-кодом", "хеш-сумою", "зведенням повідомлення".

Використання хеш-функцій в практичних системах досить різноманітне. Найрозповсюдженіші з них це:

Перевірка цілісності повідомлень та файлів. Порівнюючи хеш-значення повідомлень, обчислені до та після передачі, можна визначити, чи були внесені будь-які зміни до повідомлення чи файлу.

Верифікація пароля. При перевірці пароля зазвичай використовує криптографічні хеші, а не безпосередньо пароль. Збереження всіх паролів користувачів у вигляді відкритого тексту може призвести до масового і грубого порушення безпеки, адже файл з паролями у відкритому вигляді може бути скомпрометований. Одним із способів зменшення цієї небезпеки є зберігання у базі даних самих паролів, які хешей. При здійсненні хешування вихідні паролі не можуть бути відновлені зі збережених хеш-значень, тому

якщо користувач втратить свій пароль, то схема авторизації запропонує скинути його і придумати новий.

Цифровий підпис. Документи, що підписуються, мають різний обсяг, тому найчастіше в схемах ЕП підпис ставиться не на сам документ, а на його хеш. Обчислення хешу дозволяє виявити найменші зміни у документі під час перевірки підпису. Хешування не входить до складу алгоритму ЕП, тому у схемі може бути застосована будь-яка надійна хеш-функція.

Для ідеальної хеш-функції повинні виконуватися такі умови:

a) хеш-функція є детермінованою, тобто те саме повідомлення призводить до одного і того ж хеш-значенню;

b) значення хеш-функції має швидко обчислюватися для будь-якого повідомлення;

c) повинно бути неможливо знайти повідомлення, яке дає задане хеш-значення;

d) неможливо знайти два різні повідомлення з однаковим значенням хешу;

e) невелика зміна в повідомленні має змінювати хеш настільки сильно, що нове та старе значення здаються некорелюючими.

Вибір тієї чи іншої хеш-функції визначається багатьма факторами. Найпростішим прикладом хеш-функції може бути «обрамлення» даних циклічним надлишковим кодом (так званий CRC, cyclic redundancy code).

При виборі хеш-функції в загальному випадку для хеш-функції необхідно, щоб однозначної відповідності між вихідними даними і хеш-кодом не було б в силу того, що кількість значень хеш-функцій менша, ніж число варіантів значень вхідного масиву. Може існувати безліч масивів з різним вмістом, які дають однакові хеш-коди (таке явище називається **колізією)**. Імовірність виникнення колізій є дуже важливим фактором в оцінці якості будь-якої хеш-функції, тому адекватна хеш-функція повинна мати мінімальну кількістю колізій. Доведено, що хеш-функцій, які б зовсім не генерували б колізій, не існує взагалі.

1.2 Математичний апарат формування хеш-значень

Нехай H — скінченна множина хеш-функцій, які відображають простір ключів U в діапазоні {0,1,2,..,m−1}. Така множина називається універсальною, якщо для кожної пари ключів k,l ∈ U, (k≠l) кількість хеш-функцій h∈H, для яких h(k)=h(l) не перевищує $\frac{|H|}{m}$.

Іншими словами, при випадковому виборі хеш-функції H ймовірність колізії між різними ключами k,l не перевищує ймовірності збігу двох випадковим чином обраних хеш-значень з множини {0,1,2,..,m−1}, яка дорівнює $\frac{1}{|m|}$.

Побудова універсальної множини хеш-функцій виду $H_{p,m} = \{h_{a,b}: a \in Z_p^*, b \in Z_p\}$, де $h_{a,b}(k) = \big((ak+b) mod\ p\big) mod\ m, Z_p^* = \{1,2,..,p-1\}$, $Z_p = \{0,1,..,p-1\}$, а p — універсальне просте число.

Як правило, при розрахунку та дослідженні параметрів хеш-функцій користуються апаратом теорії чисел (модульна арифметика).

1.2.1 Розрахунок колізій

Розглянемо $k, l \in Zp: k \neq l$. Нехай для даної хеш-функції $h_{a,b}$ записані вирази:

$r=(ak+b) mod\ p$

$s=(al+b) mod\ p$,

причому, $r \neq s$, так як $r-s \equiv a(k-l) (mod\ p)$, а p — просте число, a і $(k-l)$ не дорівнюють нулю за модулем p. Отже, добуток r та s також відмінні від нуля за модулем p.

Таким чином, колізії за модулем p відсутні. Більш того, кожна з $p(p-1)$ можливих пар значень (a,b) призводять до отримання різних пар $(r,s): r \neq s$.

Щоб довести це, достатньо розглянути можливість однозначного визначення параметрів a та b за заданими значеннями r та s:

$a = ((r - s)((k - l)^{-1} \bmod p)) \bmod p$, $b = (r - ak) \bmod p$.

Оскільки є тільки $p(p-1)$ можливих пар (r,s): $r \neq s$, то є взаємно однозначна відповідність між парами (a,b) та парами (r,s): $r \neq s$. Таким чином, для будь-яких k,l при рівномірному випадковому виборі пари (a,b) з діапазону $Z_p^* \times Z_p$, одержувана в результаті пари (r,s) може бути з рівною ймовірністю будь-якої з пар з відмінними значеннями по модулю p.

Звідси випливає, що ймовірність того, що різні ключі k,l призводять до колізії, дорівнює ймовірності того, що $r \equiv s \pmod m$ при довільному виборі значень r і s, що відрізняються за модулем p. Для даного r існує $p-1$ можливих значень s. При цьому кількість значень s: $s \neq r$ та $s \equiv r \pmod p$, не перевищує $\left\lceil \frac{p}{m} \right\rceil - 1 \leq \frac{p+m-1}{m} - 1 = \frac{p-1}{m}$.

Імовірність того, що s призводить до колізії з r при наведенні за модулем m, не перевищує $\frac{p-1}{m} * \frac{1}{p-1} = \frac{1}{m}$.

Значить, $\forall k \neq l \in Z_p \; P(h_{a,b}(k) = h_{a,b}(l)) \leq \frac{1}{m}$, що означає, що множина хеш-функцій $H_{p,m}$ є універсальною.

1.3 Види хеш-функцій та перетворення в них

Сьогодні відомо доволі багато хеш-функцій. Наприклад:

1) алгоритми MD2/4/5/6. Автором даної серії алгоритмів є Рон Райвест, він же є автором алгоритму RSA:

- алгоритм MD5 наразі втрачає популярність, що пов'язано з недостатньою криптостійкістю.
- алгоритм MD6 - цікавий з конструктивної точки зору алгоритм. Він висувався на конкурс SHA-3, але, ще не був достатньо криптостійким.

2) Алгоритми лінійки SHA. Широко поширені алгоритми. В зв'язку з недостатньою криптостійкістю зараз відбувається активний перехід від SHA-1 до стандартів SHA-2.

- SHA-2 – збірна назва алгоритмів SHA224, SHA256, SHA384 та SHA512. SHA224 та SHA384 є по суті аналогами SHA256 та SHA512 відповідно. Після розрахунку згортки частина інформації в ній відкидається. Використовувати їх рекомендується лише для забезпечення сумісності з обладнанням старих моделей.
- SHA-3 (*Keccak*). Розмір хеша 224, 256, 384, 512 (змінний, $0<d\leq 264-1$), кількість раундів 24 (за замовченням).

3) BLAKE-256 працює з 32-бітними словами і повертає 32-байтний хеш, кількість раундів складає 16.

РОЗДІЛ 2

СТРУКТУРА ТА КРИПТОАНАЛІЗ ХЕШ-ФУНКЦІЙ

2.1 Структура та принцип проведення односторонніх криптоперетворень

Більшість сучасних універсальних хеш-функцій засновані на схожому математичному апараті, структура, якого наведена на рис.2.1. Таку конструкцію називають ітерованою функцією хешування. Дана конструкція була запропонована Меркл (Merkle). Подібні структури мають більшість функцій хешування, що використовуються сьогодні, включаючи MD5, SHA-1 і RIPEMD-160 тощо [12,13].

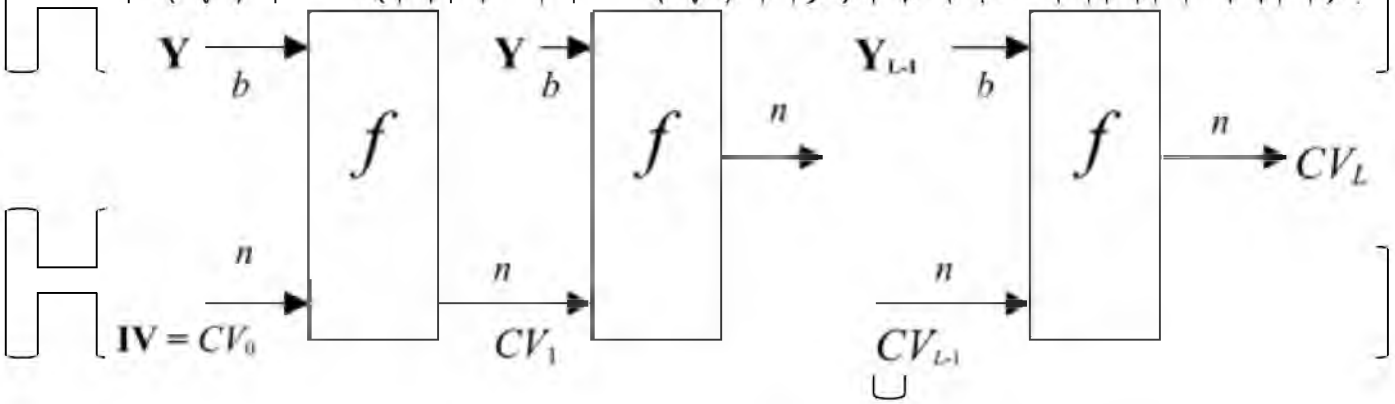

Рис. 2.1 –структура хеш-функції за арклюм

На рис. 2.1 використані позначення:

IV — початкове значення;

CV— змінна зчіплювання;

Y — *y*-й введений блок;

f — алгоритм стискання;

L — номер вхідного блоку;

N — довжина хеш-коду;

b — довжина вхідного блоку.

Показана на рис.2.1 функція хешування отримує на вхід повідомлення і ділить його на L - 1 блоків рівної фіксованої довжини по b бітів кожен. При необхідності, останній блок доповнюється додатковими бітами (для вирівнювання кратності). До останнього блоку також включається значення сумарної довжини введенної до функції хешування інформації (в бітах).

Такі перетворення ставлять перед криптоаналітиком складну задачу. Вона полягає в тому, що:

– існуватимуть два повідомлення рівної довжини, що можуть мати однакові значення функції хешування;

– можуть бути два повідомлення різної довжини, які також можуть мати однакові значення функції хешування.

В ході роботи алгоритму хешування відбувається багаторазове застосування функції стиснення f, яка отримує на вхід такі значення:

– «бітове» значення, яке отримується на попередній ітерації і називається змінною зчеплення;

– n-бітний блок повідомлення та породжуюче «бітове» вихідне значення.

На першому етапі хешування змінна зчеплення приймає початкове значення, що і є частиною алгоритму. Фінальне значення змінної зчеплення стає значенням функції хешування. Зазвичай, вважають, що $b > n$, тому говорять про те, що дана функція виконує стискання вхідної інформації при кожній ітерації. В зв'язку з цим функція хешування може бути сформульована наступним чином:

$CV_0 = IV$ = початкове n-бітове значення,

$CV_0 = f(CV_{i-1}, Y_{i-1}), 1 \le i \le L,$

де введені дані функції хешування є повідомлення M, яке складається з блоків $Yq, Y, Yl-i$.

Створення такого роду ітераційних структур є результатом досліджень Меркла і Дамгарда (Damgard). Можна стверджувати, що якщо функція стиснення має опір колізіям, то ітерована функція хешування такою ж повинна бути такою ж стійкою [11,14]. Тому, отримана структура, може використовуватися для створення практичної функції хешування, що працює з повідомленнями будь-якої довжини. Проблема створення захищеної функції хешування зводиться до проблеми пошуку функції стиснення, що має опір колізіям і функціонує з даними фіксованої довжини. А довжина таких даних може залежати від:

- продуктивності роботи готової системи
- затребуваної стійкості функції
- обмежень апаратних вимог практичної інформаційної системи.

Однією з важливих характеристик універсальних хеш-функцій є їх стійкість. Стійкість таких функції визначається методами криптоаналізу. Криптоаналіз хеш - функцій зазвичай концентрується на дослідженні внутрішньої структури та спирається на спроби знаходження ефективних методів виявлення колізій при одноразовому виконанні функції f.

Якщо таку проблему вирішено, то атакуючому залишається розглянути фіксоване початкове значення IV. Конкретний вид атаки залежить від внутрішньої структури цієї хеш-функції. Зазвичай, наприклад, коли йдеться про симетричні блокові шифри, то передбачає кілька раундів обробки даних (криптоперетворень), тому краще виконувати аналіз зміни побітової структури даних від раунду до раунду.

Необхідно приймати до уваги факт, що колізії неминуче будуть виникати в будь-якій функції хешування. Це пов’язано з тим, що кожна така функція перетворює, як мінімум, блок довжиною b в хеш-код довжини n, де виконується умова $b > n$.

Від практичної функції вимагається лише обчислювальна неможливість виявити подібні колізії. Для прикладу можна оцінити складність атак на алгоритм SHA-1 (таблиця 2.1) [14,15].

В таблиці 2.1 наведено приблизну вартість атак типу «груба сила» на хеш-функції. Вартість атаки наведена з розрахунку вартості оренди одного GPU типу GTX 1060 в 35 доларів на місяць.

Таблиця 2.1 - Складність атак на SHA-1

Функція	Тип колізії	GPU	Час, років	Складність атаки	Ціна атаки, USD
SHA-1	collision	GTX 970	22	$2^{61,2}$	11000
		GTX 1060	27	$2^{61,6}$	
		GTX 1080Ti	8	$2^{61,6}$	
	Chosen-prefix	GTX 970	99	$2^{63,4}$	45000
		GTX 1060	107	$2^{63,5}$	
		GTX 1080Ti	34	$2^{63,6}$	
MD5||SHA-1	Both (plain or CP)	GTX 970	1400	$2^{67,2}$	720000
		GTX 1060	1700	$2^{67,6}$	
		GTX 1080Ti	540	$2^{67,6}$	

2.2 Проблема колізій в хеш-функціях

Колізією у хеш-функціях називають явище, коли для двох різних вхідних значень (аргументів) функції їх хеш-сума на виході функції є ідентичною. Колізії однаково часто зустрічаються в різноманітних алгоритмах хешування. Уникнути колізій не вдається навіть теоретично. Даний факт приймається в якості норми. При розробці нової універсальної функції слід звести виникнення колізій до мінімуму.

Так, при занесенні в таблицю розміром 365 комірок лише 23 хеш-значень, ймовірність колізії може перевищити 50% (за умови, якщо кожен елемент може з однаковою ймовірністю потрапити в будь-яку комірку). Тому механізм подолання колізій є важливою складовою будь-якої хеш-функції.

На сьогодні алгоритм SHA-1 не в змозі забезпечити необхідну стійкість до криптоатак. Тому, наприклад, розробники бібліотек для протоколу SSH відключили використання за замовчанням застарілої криптофункції SHA-1 з даного набору. Наразі підбір серверного ключа при автентифікації через функцію SHA-1, тобто колізія з обраним префіксом, на орендованому кластері з GPU можлива і обійдеться у суму приблизно $45 тис (таблиця 2.1). Тому, подібна атака є доступною не тільки для державних спецслужб з достатнім бюджетом, але і для комерційних клієнтів та і для будь-кого.

Зараз алгоритм SHA-1 вже вважається потенційно безпечним. Наприклад, відключення SHA-1 за замовчанням одночасно провели розробники відкритих криптографічних програмних бібліотек OpenSSH (release notes) та libssh (зміна коду). Дана одностороння хеш-функція майже 20 років від початку свого представлення була надійною (на відміну від функції md5), але у 2017 році співробітники Google та Центру математики та інформатики м. Амстердам представили перший спосіб генерації колізій для SHA-1.

2.3 Принцип роботи алгоритму MD5

MD5 є криптографічною хеш-функцією, яка працює на базі алгоритму, також відомого, як "повідомлення-дайджест". Отримана послідовність містить ряд бітів, що генеруються через процедуру одностороннього хешування. Така хеш-функція вважається одностороннім криптографічним перетворенням. Дайджести повідомлень розроблені для того, аби забезпечити цілісність фрагментів інформації. Застосування хеш-функцій дозволяє точно знати, чи

були внесені зміни або доповнення до повідомлення. Відповідь на це питання подається у формі: «так» або «ні».

Алгоритм, який реалізує хеш-функцію MD5 повертає 128-бітну хеш-суму. Дане криптографічне перетворення реалізується, щодо будь-якого розміру вхідного повідомлення в хеш-значення. Дане перетворення реалізується у 5 кроків. Опис такого перетворення наведений у відкритих джерелах.

Крок 1: Додавання бітів заповнення

Заповнення означає додавання додаткових бітів до відкритого повідомлення. Таким чином, MD5 повідомлення доповнюється таким чином, щоб його довжина в бітах збігалася з 448 за модулем 512. Заповнення виконується так, що загальна кількість бітів на 64 менше, кратне довжині 512 біт.

Заповнення виконується, навіть якщо довжина вихідного повідомлення відповідає 448 по модулю 512. У бітах заповнення єдиний перший біт дорівнює 1, а інші біти дорівнюють 0.

Крок 2: Додавання до кратної довжини блоку

Після заповнення блоків до кратних за модулем 512 в кінці додаються 64 біти, що використовуються для запису довжини вихідного введення по модулю 2^{64}. На цьому етапі результуюче повідомлення має довжину, кратну 512 бітів.

Крок 3: Ініціалізація буфера MD функції

Буфер, що складається з чотирьох слів (A, B, C, D) використовується для обчислення значень для дайджесту повідомлення. В таблиці 2.2 значення A, B, C, D є 32-бітними регістрами ініціалізуються відповідними значеннями.

Таблиця 2.2

Приклад виконання кроку 3 перетворень функції md5

Слова дайджесту	Значення в бітах регістрів слів			
Слово A	01	23	45	67
Слово B	89	Ab	Компакт диск	Ef
Слово C	Fe	Округ Колумбія	Ba	98
Слово D	76	54	32	10

Крок 4: Обробка повідомлення у блоці з 16-ти бітових слів

Алгоритм MD5 використовує допоміжні функції, які приймають дані зі входу в якості трьох 32-бітних чисел і видають 32-бітну числову комбінацію на вихід. Ці функції використовують логічні оператори, такі як OR, XOR, NOR (таблиця 2.3).

Таблиця 2.3

Логічні операції в хеш-функції md5 на кроці 4

Крокова функція	**Логічна операція**
F(X,Y,Z)	XY ∨ не (X) Z
G(X,Y,Z)	XZ ∨ Y не (Z)
H(X,Y,Z)	X XOR Y XOR Z
Я(X,Y,Z)	Y хог (X ∨ не (Z))

Вміст чотирьох буферів зміщується з входом з використанням цього допоміжного буфера і 16 циклів виконуються з використанням 16 основних операцій.

В результаті перетворень, які відбуваються в на кроці 4, значення отримані в буферах A, B, C, D містять частини хеш-суми функції MD5, починаючи з молодшого біта A і закінчуючи старшим бітом D.

2.4 Криптоаналіз хеш-функцій

Криптоаналіз хеш-функцій спрямований на дослідження вразливості до різного виду атак. Основні з них наступні:

- знаходження колізій — ситуація, коли двом різним вихідним повідомленням відповідає те саме хеш-значення;
- знаходження прообразу - вихідного повідомлення - на його хешу
- при розв'язку методом «грубої сили»:
  - друга задача вимагає 2^{160} операцій;
  - перша задача вимагає в середньому $2^{160/2}=2^{80}$ операцій, якщо використовувати «парадокс днів народження» [15].

Від стійкості хеш-функції до знаходження колізій залежить безпека електронного цифрового підпису, який отримується за допомогою даного хеш-алгоритму. Від стійкості до знаходження прообразу залежить безпека зберігання хешів паролів з метою автентифікації.

У січні 2005 року Vincent Rijmen та Elisabeth Oswald опублікували повідомлення про атаку на усічену версію SHA-1 (53 раунди замість 80), яка дозволяє знаходити колізії менше, ніж за 280 операцій [16].

MD5, разом з іншими подібними хеш-функціями, зазвичай використовується при створенні цифрових підписів та кодів підтвердження повідомлень, індексування даних у хеш-таблицях, в якості індикаторів скопійованих даних, в якості цифрових «відбитків пальців», для сортування та ідентифікації файлів, а також виступати в якості контрольних сум, у виявленні ненавмисне пошкодження даних тощо.

Використання MD5 хеш для забезпечення цілісності даних файлів пояснюється тим, що хеш-алгоритм MD5 має унікальний спосіб отримання самих результатів, для одного і того ж набору даних.

Слід розуміти, що алгоритм MD5 не реалізує шифрування, він створює цифровий відбиток для входу. Крім того, дана хеш-функція є односторонньою.

Це означає, що користувач не може скасувати згенерований MD5 хеш, щоб відновити початковий рядок (відкритий текст).

Початково алгоритм MD5 був розроблений для зберігання одностороннього хешу пароля. Тому деякі файлові сервери також надають попередньо обчислену контрольну суму файлу MD5, щоб користувач міг порівняти з ним контрольну суму завантаженого файлу. Більшість операційних систем на основі Unix включають утиліти контрольної суми MD5 у пакети розповсюдження того чи іншого сумісного ПЗ.

MD5 хеш генерується шляхом отримання рядка будь-якої бажаної довжини та кодуванням його в 128-бітний цифровий «відбиток пальця». Набравши той самий рядок, використовуючи генератор MD5 завжди буде згенерувати той самий 128-бітовий хеш-результат. В практичних умовах алгоритм MD5 зазвичай використовуються з меншою бітністю вихідного рядку при прийнятті та збереженні паролів, номерів кредитних карт або будь-якої іншої конфіденційної інформації в базах даних, таких як популярна система зберігання даних у форматі MySQL.

Отже, можна використовувати тільки MD5 в якості базового алгоритму для обчислення контрольної суми або елемент управління в таблиці бази даних. З огляду на недостатню на сьогодні криптостійкість в ряді інших задач, даний алгоритм є цілком придатним для таких задач. MD5 вважається компактним алгоритмом – його хеш має довжину 32 цифр. Саме тому він не потребує занадто багато місця для зберігання і може обчислювати і генерувати хеш в оснастці інформаційної системи. Слід мати на увазі, що рекомендовано уникати використання MD5 для паролів, дайджестів або інших критичних систем безпеки.

2.4 Райдужні таблиці в криптоаналізі хеш-функції

Злам хеш-функцій можливий по заздалегідь сформованим таблицям їх повних значень. Такі таблиці відомі, як райдужні (веселкові).

Райдужні таблиці також відомі як довідкові таблиці зворотної хеш-функції. У цьому методі MD5 обчислює хеш-значення для даних або повідомлень, що дає змогу досить легко використовувати грубу силу для пошуку значення функції. Наприклад, можна розпочати обчислення дайджесту з восьми знаків (літерно-цифрових комбінацій) і аналітик може подивитися на таблицю паролів, щоб дізнатися, який пароль збігається та з яким хешем дане значення співставне.

Хоча, в свій час алгоритм MD5 був визнаний криптостійким (без зіткнень та колізій) та даний факт може сильно залежати від того, як і з якою метою його використовують.

Райдужна таблиця створюється шляхом побудови ланцюжків потенційних паролів. Кожен ланцюжок починається з випадкового можливого пароля, потім піддається хешування та функції редукції. Ця функція перетворює результат хеш-функції на певний можливий пароль.

Райдужна таблиця створюється побудовою ланцюжків потенційних паролів. Кожен ланцюжок починається з випадкового можливого пароля, потім піддається дії хеш-функції та функції редукції. Ця функція перетворює результат хеш-функції на певний можливий пароль.

Для відновлення пароля значення хеш-функції підлається обробці функції редукції і здійснюється пошук в таблиці. Якщо не знайдено збігу, то знову застосовується хеш-функція та функція редукції. Ця операція триває, доки не буде знайдено збігів. Після знаходження таких збігів, ланцюжок, що містить його, відновлюється для знаходження відкинутого значення, яке і буде шуканим паролем.

В результаті виходить «веселкова» таблиця, яка може з високою ймовірністю відновити пароль за короткий час.

«Веселкові» таблиці можуть зламувати тільки ту хеш-функцію, для якої вони створювалися, тобто таблиці для MD5 можуть зламати тільки хеші, отримані для функції MD5. Теорія для застосування даної технології була розроблена Philippe Oechslin [17] в якості швидкого варіанту time-memory

tradeoff [18]. Вперше технологія була використана в програмі Ophcrack для зламування хешів LanMan, які використовуються в операційній системі Microsoft Windows.

Пізніше була розроблена більш досконала програма RainbowCrack, яка може працювати з великою кількістю хеш-значень, наприклад LanMan, MD5, SHA1 та інші. Наступним кроком було створення програми The UDC, яка дозволяє будувати Hybrid Rainbow таблиці не за набором символів, а за набором словників, що дозволяє відновлювати паролі більшої довжини (фактично необмеженої довжини).

При генерації таблиць важливо знайти найкраще співвідношення взаємозалежних параметрів, таких як:

- ймовірність знаходження пароля за отриманими таблицями;
- часу генерації таблиць;
- час підбору пароля за таблицями;
- займане місце.

Наведені вище параметри залежать від налаштувань заданих при генерації таблиць:

- допустимий набір символів;
- довжина пароля;
- довжина ланцюжка;
- кількість таблиць.

При цьому час генерації хешів залежить майже виключно від бажаної ймовірності підбору, використовуваного набору символів та довжини пароля.

Займане таблицями місце залежить від бажаної швидкості підбору одного пароля по готовим таблицям.

Хоча застосування райдужних таблиць полегшує використання методу грубої сили (bruteforce) для підбору паролів, у деяких випадках необхідні для їх генерації або використання обчислювальні потужності не дозволяють одиночному користувачеві досягти бажаних результатів за прийнятний час. Наприклад для паролів довжиною трохи більше 8 символів, які з букв, цифр і

спеціальних символів !@#$%^&*()-_+= захешованих алгоритмом MD5 можуть бути генеровані таблиці з наступними параметрами [20]:

- довжина ланцюжка 1400;
- кількість ланцюжків 50000000;
- кількість таблиць 800.

При цьому ймовірність знаходження паролю за допомогою даних таблиць для випадку хеш-функції MD5 складе 0.7542 (75.42%), а розмір таблиці для даної розрядності функції складе орієнтовно 596 Гб. Генерація таких таблиць на комп'ютері CISC архітектури типу x86 при однопоточних обчисленням, з частотою CPU в 1 ГГц займе 3 роки, а пошук лише одного паролю по готових таблицях - не більше 22 хвилин.

Однак процес генерації таблиць можна розпаралелити, наприклад розрахунок однієї таблиці з наведеними вище параметрами займає приблизно 33 години. В разі, якщо в розпорядженні криптоаналітика є 100 комп'ютерів, всі таблиці можна згенерувати за проміжок часу в 11 діб.

2.4.1 Захист від «райдужних» таблиць

Дані таблиці неможливо використовувати проти незворотних хеш-функцій, які включають salt (сіль). Для прикладу, розглянемо наступну функцію для створення хешу від пароля:

хеш = MD5 (пароль + сіль), де «+» означає операцію конкатенації.

Для відновлення такого пароля криптоаналітику необхідні таблиці для всіх можливих значень «тестової суми».

По суті, «тестова сума» збільшує довжину і, можливо, складність пароля. Якщо таблиця розрахована на деяку довжину або на певний обмежений набір символів, «тестова сума» може запобігти відновленню пароля.

Практично всі дистрибутиви ОС Unix, GNU/Linux і BSD використовують хеші з сіллю для зберігання системних паролів, хоча багато програм, наприклад, інтернет-скрипти на PHP використовують простий хеш (звичайно MD5) без солі. ОС Microsoft Windows і Windows NT

використовують хеші LAN Manager і NT LAN Manager, які також використовують сіль, що робить їх одними з найпопулярніших для створення «веселкових» таблиць.

Тож, можна виділити ключові відмінності між алгоритмами (рис.2.2).

Funciton	MD5	SHA1
Block length	512 bit	512 bit
Algorithm length	128 bit	160 bit
Rotation steps	64 steps	80 steps
Initialization variables	4	5

Рис.2.2 – Відмінності алгоритмів MD5 та SHA-1

Таким чином, можна побачити, що обидва алгоритми SHA-1 і MD5, вийшли від MD4, тому мають багато спільного. В той же час для отримання об'єктивних результатів порівнянь створюваного алгоритму універсальної хеш-функції має сенс с створювати хеш-функцію однакової розрядності і засновану на однакових підходах до конструювання та здійснення ітерацій (принцип Меркле).

РОЗДІЛ 3

РОЗРОБКА ТА ТЕСТУВАННЯ КРИПТОГРАФІЧНОЇ ХЕШ-ФУНКЦІЇ

3.1 Алгоритм криптографічної хеш-функції

При розробці алгоритму хеш-функції слід виходити з того, що така універсальна функціє є функцією згортки. Для тестування прототипу отриманої функції достатньо отримати її 16 та 32-бітний варіант хеш-суми. Відповідно до описаного вище класичного підходу Меркле, вхідне повідомлення *M* розіб'ємо на 4 блоки рівної довжини, які будемо послідовно обробляти за допомогою XOR-перетворень, кількість яких співпадає з кількістю блоків поділу. Але, така обробка, за умови не застосування до отриманої суми згортки функції безумовно викликатиме наявність колізій вже при значенні повідомлення на вході функції, яке дорівнює рівному «0» (null). Таке значення на вході хеш-функції є теоретично дорпустимим.

Для того, щоб підготувати дані, необхідно змінити у повідомленні значення передостаннього біта на 1, останнього на 0. Так необхідно робити для випадків, якщо в повідомленні присутні тільки нульові або тільки одиничні значення бітів). Потім слід дописати дзеркально біти, починаючи з кінця, поки повідомлення не буде кратне 64 бітам (якщо не вистачає бітів, то дописуємо нулі):

Початкове повідомлення:

1100 0110 0001 0100 1101 1110 0101 0101 1110 1011 0110 0001 0001

Кориговане повідомлення має наступний вид:

1100 0110 0001 0100 1101 1110 0101 0101 1110 1011 0110 0001 0010 0100 1000 0110

Слід додати, що подібний прийом використовують для уникнення колізій цілий ряд існуючих хеш-функцій (наприклад, md4/md5/md6).

Підготовлене повідомлення далі розбивається на блоки по 16 біт кожен наступним чином:

b1: 1100 0110 0001 0100

b2: 1101 1110 0101 0101

b3: 1110 1011 0110 0001

b4: 0010 0100 1000 0110

Після розбивки виконуємо інверсію першого блоку:

b1: **0011 1001 1110 1011**

b2: 1101 1110 0101 0101

b3: 1110 1011 0110 0001

b4: 0010 0100 1000 0110

Виконання інверсії також є запозиченим прийом. Так досягається збільшення криптостійкості через досягнення більш рівномірної дисперсії в розподілі нульових та одиничних бітів в блоках.

Наступним кроком реалізуємо для отриманих раніше блоків b1, b2, b3, b4 операцію XOR в такій послідовності:

$b12 = b1 \oplus b2$; $b23 = b12 \oplus b3$; $b34 = b23 \oplus b4$.

Після реалізації XOR-перетворень можна говорити про те, що отримана класична хеш-сума у вигляді односторонньої згортки. А результатом реалізації ланцюга XOR перетворень будуть наступні блоки:

b12: 1110 0111 1011 1110

b23: 0000 1100 1101 1111

Після, на фінальному кроці отримуємо значення хеш-суми функції згортки:

b34: 0010 1000 0101 1001

Далі можна продемонструвати отримані параметрів результатів роботи створеної хеш-функції порівняно з реалізаціями в коді на Python хеш-функцій SHA-1 та MD5.

3.2 Програмний засіб на базі створеного алгоритму хеш-функції

Для перевірки створеного алгоритму хеш-функцій на наявність колізій та інших параметрів здійснено програму реалізацію алгоритму цієї функції мовою Python. Тут порівнюємо результати роботи створеної універсальної хеш-функції з її аналогами за призначенням та принципом проведення згортки - функціями MD5 та SHA-1. В якості основною характеристики проведення перетворень для хеш-функції, як було відмічено в розділі 2, будемо розглядати відсутність або мінімальну присутність колізій (в ідеальної хеш-функції результат на виході повинен мати більш рівномірний розподіл значень хеш-сум і не співпадати з конгруентним законом формування результатів).

Методика перевірки можливої появи колізії для хеш-функції MD5 реалізований наступним програмним кодом:

Код програми функції md5:

Вміст файлу filecheck.py наведений на рис.3.1.

```python
from collections import Counter
def filewrite(text):
    f=open("hash.txt",'a')
    f.write(text+'\n')
    f.close()
def filecheckmd5():
    kount=0
    with open('md5.txt') as f:
        c = Counter(c.strip().lower() for c in f if c.strip())  # for case-insensitive search
    for line in c:
        if c[line] > 1:
            kount+=1
        else:
            kount+=0
    if kount>0:
        print("MD5: На жаль, присутні колізії в кількості", kount)
    else:
        print("MD5: Колізій немає")
    f.close()

```

Рис.3.1 - Вміст файлу filecheck.py з програмною реалізацієї функції md5

Фрагмент програмного коду для перевірки на наявність можливих колізій в хеш-функції SHA-1 реалізовано в наступному фрагменті програмного коду (рис.3.2):

```
def filechecksha1():
    kount=0
    with open('sha1.txt') as f:
        c = Counter(c.strip().lower() for c in f if
c.strip())  # for case-insensitive search
    for line in c:
        if c[line] > 1:
            kount+=1
        else:
            kount+=0
    if kount>0:
        print("SHA-1: На жаль, присутні колізії в
кількості", kount)
    else:
        print("SHA-1: Колізій немає")
    f.close()
```

Рис.3.2 - Фрагмент програмного коду для перевірки на наявність можливих колізій в хеш-функції SHA-1

Метод перевірки можливих колізій розробленої в роботі хеш-функції user для 32 бітової версії реалізований у наступний спосіб (рис.3.3):

```
def filecheckuser():
    kount=0
    with open('user.txt') as f:
        c = Counter(c.strip().lower() for c in f if
c.strip())  # for case-insensitive search
    for line in c:
        if c[line] > 1:
            kount+=1
        else:
            kount+=0
    if kount>0:
        print("User: На жаль, присутні колізії в
кількості", kount)
    else:
        print("User: Колізій немає")
    f.close()
```

Рис.3.3 – Фрагмент програмного коду методу перевірки можливих колізій розробленої в роботі хеш-функції

Нижче наведено основний програмний код головного класу програми, який містить:

1. Розроблений алгоритм хешування.
2. Програмну реалізації хеш-функцій MD5, SHA-1 для зняття порівняльних характеристик розробленого і даних алгоритмів хешування
3. Інтерфейс виводу результатів роботи представлений в класі main.py:

```
1  import filecheck
2  from random import randrange
3  import hashlib
```

Рис.3.4 – перелік допоміжних модулів в основній програмі

В якості допоміжних модулів в основній програмі використано створені раніше на Python класи filecheck, random, randrange, hashlib (рис.3.4).

На рис.3.4 наведено етап програмний код етапу підготовки даних та інверсії бітів в блоці b12 16-ти бітового варіанту розробленої хеш-функції.

```
#user HASH
def logic_xor(x1,x2):
    res=""
    for i in range (0,16):
        xor=str(int(bool(int(x1[i])) ^ bool(int(x2[i]))))
        res=res+xor
    return res
def inversion(x):
    res=""
    for i in range (0,16):
        if x[i] == "1":
            res=res+"0"
        else:
            res=res+"1"
    return res
def hashown(x):
```

Рис.3.5 – етап програмний код етапу підготовки даних та інверсії бітів в блоці b12 16-ти бітового варіанту хеш-функції user

Зміна бітів для уникнення колізії першого роду реалізована, як показано на рис.3.6.

```
b1="1"
b2="0"
x=x[:-2]
x=x+b1
x=x+b2
```

Рис.3.6 – Зміна бітів для уникнення колізії першого роду

Корегування повідомлення для функції user для доповнення до номінального кратного розміру блоків представлено на рис.3.7.

```
lm=len(x)
zaz=lm
if lm<32:
    print("Замале число для 64-бітного хешу")
    while len(x)<zaz*2:
      x=x+x[lm-1]
      lm-=1
    if len(x)<64:
      while len(x)<64:
        ins='0'
        x=x+ins
if lm<64:
    while len(x)<64:
      x=x+x[lm-1]
      lm-=1
```

Рис.3.7 – Корегування повідомлення для функції user для доповнення до номінального кратного розміру блоків

Ділення на блоки реалізуємо у досліджуваної функції user таким чином (рис.3.8):

```
    word1=x[0:16]
    word1=inversion(word1)
    word2=x[16:32]
    word3=x[32:48]
    word4=x[48:64]
    ##XOR
    res=logic_xor(word1,word2)
    res=logic_xor(res,word3)
    res=logic_xor(res,word4)
    return res
f1=open("md5.txt",'w')
f2=open("sha1.txt",'w')
f3=open("user.txt",'w')
for i in range (0,60):
    mess=randrange(0, 2**128)

f1.write(str(hashlib.md5(str(mess).encode()).hexdigest())+
"\n")

f2.write(str(hashlib.sha1(str(mess).encode()).hexdigest())
+"\n")
    f3.write(str(hex(int(hashown(bin(mess)),2)))[2:]+"\n")
f1.close()
f2.close()
f3.close()
filecheck.filecheckmd5()
filecheck.filechecksha1()
filecheck.filecheckuser()
```

Рис.3.8 - Ділення на блоки реалізуємо у досліджуваній функції user

В результаті програмної реалізації отримано робочий програмний код для трьох досліджуваних хеш-функцій: md5, sha-1 і створеної в даній роботі функції user, яку надалі будемо позначати, як user-16 (для 16 бітового варіанту хеш-суми) і user-32 (для 32 бітового варіанту хеш-суми).

3.3 Якісні характеристики розробленої криптографічної хеш-функції

Результати роботи програмної реалізації запропонованого алгоритму створеної хеш-функції user та порівнянюваних хеш-функцій: md5, sha-1 для різної кількості вхідних повідомлень наведені на рис.3.9-3.14.

Далі наведемо результати роботи програмних реалізацій створеної хеш-функції на 6 (шести) фіксованих вхідних наборах вхідних послідовностей даних *M*. Причому для всіх трьох порівнюваних універсальних хеш-функції ці набори вхідних даних M будуть ідентичними.

```
MD5: Колізій немає
SHA-1: Колізій немає
User: На жаль, присутні колізії в кількості 2
```

Рис.3.9 – Результати роботи програми для 500 вхідних повідомлень *M*

```
MD5: Колізій немає
SHA-1: Колізій немає
User: На жаль, присутні колізії в кількості 6
```

Рис.3.10 – Результати роботи програми для 1000 вхідних повідомлень *M*

```
MD5: Колізій немає
SHA-1: Колізій немає
User: На жаль, присутні колізії в кількості 25
```

Рис.3.11 – Результати роботи програми для 2000 вхідних повідомлень *M*

```
MD5: Колізій немає
SHA-1: Колізій немає
User: На жаль, присутні колізії в кількості 162
```

Рис.3.12 – Результати роботи програми для 5000 вхідних повідомлень *M*

Рис.3.13 – Результати роботи програми для 7500 вхідних повідомлень M

Рис.3.14 – Результати роботи програми для 10000 вхідних повідомлень M

Порівнюючи результати роботи програми для програмних реалізацій різних алгоритмів можна дійти висновку, що створена користувацька хеш-функція не є надто криптостійкою в своєму 16 бітовому варіанті адже з точки зору кількості колізій порівняно з існуючими алгоритмами MD5 та SHA-1 вона програє ним. Причини появи колізії у порівнянні зі стандартними функціями MD5 і SHA-1 в створеному алгоритмі можна пояснити відомими причинами:

1. Довжина функції у 16 біт порівняно з 32 бітами у функціях MD5 і SHA-1 занадто недостатня для отримання більш/менш рівномірної дисперсії даних.

2. Відсутній механізмів корегування і нормалізації вхідних блоків повідомлення M через дописування бітів контрольних сум або криптографічної солі тощо

3. Недосконалий механізм бітової згортки хеш-функції.

В зв’язку з наведеними вище результатами необхідно розробити методи для усунення отриманих в ході тестування користувацького алгоритму універсальної хеш-функції недоліків.

3.4 Методи розв’язання проблеми колізій в хеш-функціях

Проблема наявності колізій характерна для всіх хеш-функцій і не тільки тих, що побудовані за принципами Меркле. В деяких випадках вдається

уникнути колізій на певних проміжках існування функції. Наприклад, якщо ключі елементів будуть відомі розробнику заздалегідь або дуже рідко змінюватимуться, то для таких елементів можна визначити деяку досконалу хеш-функцію, яка здійснить їх розподіл по коміркам хеш-таблиці без наявності колізій.

3.4.1 Методи ліквідації з колізіями в хеш-таблицях

Розв’язок колізій в хеш-таблиці –завдання, яке теоретично можна розв'язати декількома способами, наприклад:

- метод ланцюжків;
- відкрита адресація тощо.

При розв’язанні колізій у хеш-функціях важливо зводити їх кількість до мінімуму, адже це підвищує вимоги до часу роботи з хеш-таблицями.

В більшості робіт, що описують процес хешування, описані методи боротьби з колізіями в таблицях хеш-сум [19,20]. Перші універсальні хеш-функції в більшості випадків застосовувалися для пошуку тексту у файлах великого розміру. Описані два основних методи боротьби з такими колізіями в хеш-таблицях:

- спосіб ланцюжків (метод прямого зв'язування);
- метод відкритої адресації.
- лінійне розв’язання колізій.
- криптографічна сіль.

При використання способу ланцюжків кожна комірка з номером і масиву H буде містити покажчик на початок списку всіх елементів, хеш-код яких дорівнює і або вказуватиме на їх відсутність. Колізії спричиняють явище, коли з'являються списки розмір яких більше одного елемента.

Для універсальної функції залежно від потреби в унікальності значень операції вставки, швидкість здійснення перетворень буде різною. Якщо значення часового показник не важливе, то можна використовувати список, час формування значення для вставки в який буде в не більше ніж $\theta(1)$. В

противному випадку необхідно буде перевіряти наявність у списку даного елементу, а потім, в разі його відсутності, такий елемент необхідно буде додати. Тоді вставка елемента в найнесприятливішому випадку буде виконана за час $\theta(n)$, де n – максимально можлива кількість елементів (хеш-сум).

В разі лінійного розв'язку колізій час здійснення пошуку у найнесприятливішому випадку буде пропорційним до довжини списку всіх елементів. Якщо ж всі n ключів хешуються в одну й ту ж комірку (утворюють список довжиною n), тоді час пошуку буде дорівнюватиме $\theta(n)$ плюс час, необхідний для обчислення хеш-функції. Отриманий показник не буде кращим, ніж за умови використання зв'язкового списку для зберігання всіх n елементів.

Видалення елемента може бути виконано за час $\theta(1)$, так само, як і вставка при використанні 2-зв'язкового списку.

Для створюваної функції гарним механізмом підвищення якості хеш-сум може виявитися криптографічна «сіль». Така «сіль» використовується з метою захисту паролів та електронного цифрового підпису від підробки.

Існує декілька методів впровадження «солі» в хеш-функцію. Такі методі є дієвим захистом навіть тоді, коли криптоаналітику відома максимальна кількість способів отримання колізій для конкретної хеш-функції. Одним із даних методів є додавання до вхідної послідовності в якості криптографічної "солі" рядків випадкових даних (як правило бітових). В особливо відповідальних системах передбачено використання апаратного генератора випадкової «солі». Іноді "сіль" додається також і до хеш-коду. Додавання випадкових даних завжди значно ускладнює аналіз сформованих хеш-таблиць. А криптоаналіз з використанням «райдужних» таблиць практично завжди не виправдає себе. Такий спосіб, наприклад, використовувався широко при автентифікації та збереженні паролів користувачів у UNIX-подібних операційних системах. Тоді в якості хеш-функції використовували функцію md5.

В розробленому алгоритмі колізій можна уникнути шляхом вдосконалення наявного алгоритму, наприклад за рахунок роботи з більшими за розмірами вхідними блоками, вкладенням процесу згортки повідомлення або збільшенням кількості ітерацій і розрядності хешу. Відредагувавши відповідним чином хеш-функцію user, так, що повідомлення буде ділиться на блоки по 32 біти а хеш становитиме 32 біти, маємо відповідний результат (рис.3.15-3.20):

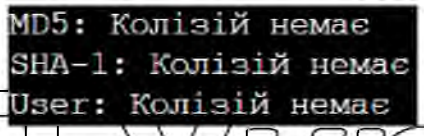

Рис.3.15 – Результати роботи програми для 500 заданих повідомлень

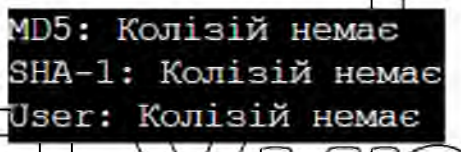

Рис.3.16 – Результати роботи програми для 1000 вхідних повідомлень

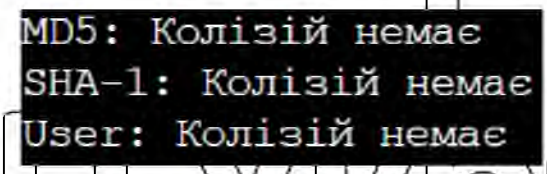

Рис.3.17 – Результати роботи програми для 2000 вхідних повідомлень

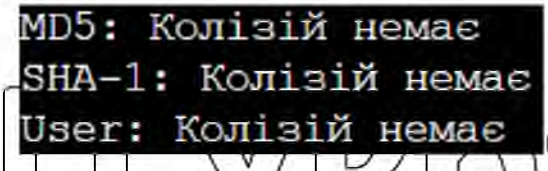

Рис.3.18 – Результати роботи програми для 5000 вхідних повідомлень

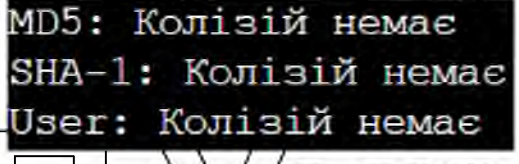

Рис.3.19 – Результати роботи програми для 7500 вхідних повідомлень

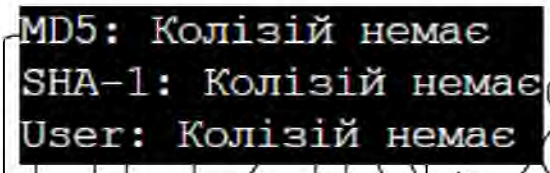

Рис.3.20 – Результати роботи програми для 10000 вхідних повідомлень

Отримані на рис. 3.15-3.20 результати свідчать про те, що застосований метод усунув колізії у функції user. В таблиці 3.1 наведено порівняння числа колізій в розробленій хеш функції (варіант user-32) та функціях MD5 і SHA-1 відповідно.

З отриманих внаслідок обчислювальних експериментів даних видно, що методи боротьби з колізіями в хеш-таблицях, такі як:

- збільшення розмірів вхідних блоків;
- ускладенням процесу згортки повідомлення;
- збільшенням кількості ітерацій, інверсії бітів та розміру самого хешу;
- модифікація процесу підготовки блоків,

дають змогу покращити якість роботи створеної хеш-функції

Таблиця 3.1

Порівняння кількості колізій в розробленій хеш функції (User) та функціях MD5 і SHA-1

Кількість повідомлень	Кількість колізій (одиниць)			
	User (16-бітний хеш)	User (32-бітний хеш)	MD5	SHA-1
500	2	-	-	-
1000	6	-	-	-
2000	25	-	-	-
5000	162	-	-	-
7500	384	-	-	-
10000	694	-	-	-

Для того аби отримати вичерпні дані для всього діапазону значень функції слід здійснити атаку методом «грубої сили», тобто перебрати 2^{32} = 4 294 967 296 наборів вхідних даних *M*. Навіть за умови повного перебору кількість знайдених колізій теоретично залишилася на рівні функцій-аналогів - SHA-1 і MD5.

З даних таблиці 3.1, можна побачити, що за умови внесення коректив в алгоритм роботи створеної функції вже при довжині вихідних значень хеш-функції у 32 біти і проведенні 10000 тестів, колізій хеш-сум не фіксуються (рис.3.20).

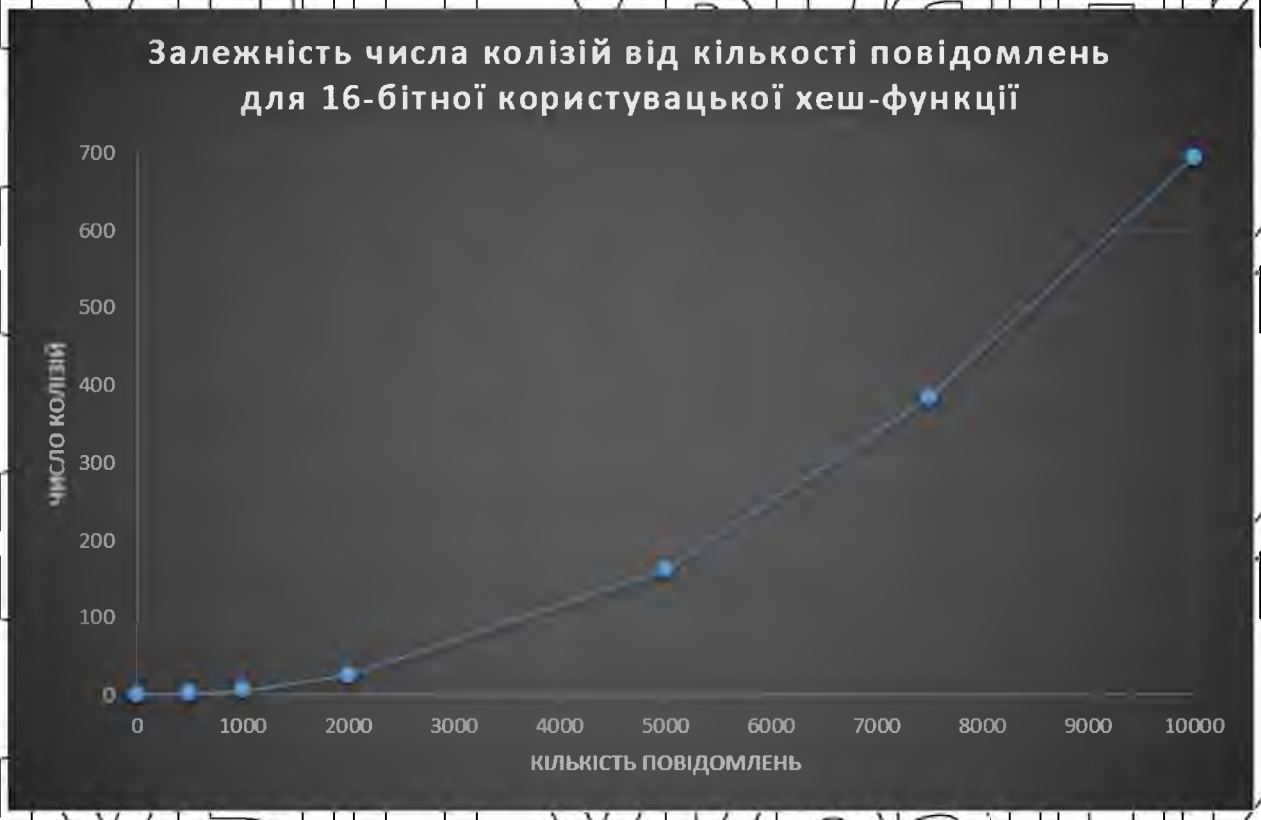

Рис.3.20 – залежність числа колізій в 16-бітовому варіанті створеної хеш-функції (user) від кількості вхідних тестових повідомлень

З графіку залежності числа колізій в 16-бітовому варіанті створеної хеш-функції (user) від кількості вхідних тестових повідомлень видно, що початкова

якість і стійкість до колізій у розробленої функції user-16 є незадовільною при довжині хеш-суми в 16 біт. Частота появи колізій хоч і перевищує лінійний показник функцій типу $y = kx$, але це недостатньо для практичного її застосування. В разі збільшення кількості тестових прикладів кутовий коефіцієнт графіку, який описує графічну залежність , як видно, збільшується за законом, близьким до експоненційного.

Далі проаналізуємо варіант створеної хеш-функції з використанням криптографічної солі. За аналогією з попередній експериментом візьмемо перші 1000 фіксованих комбінацій бітових повідомлень, потім - наступні 1000 фіксованих комбінацій в повідомленнях. Результати криптоаналізу за методом атаки «грубої сили» на хеш-функцію user-32 з сіллю показані на рис.3.22.

```
Перевірка на криптостійкість для іншої 1000 повідомлень:
Кількість зламаних хешей: 149

Перевірка на криптостійкість для іншої 1000 повідомлень:
Кількість зламаних хешей: 165
```

Рис.3.22 - Результати роботи програми при застосуванні криптоаналізу методом «грубої сили» для створеної хеш-функції user-16 з криптографічною сіллю

Як видно, сіль і збільшення розрядності, інверсія бітів та інші застосовані покращення якості створеної функції для боротьби з колізіями в хеш-таблицях здатні помітно покращити якість функції. З даних таблиці 3.1 видно, що для функції User (32-бітний хеш) на видимому проміжку тестових повідомлень (до 10000 повідомлень включно) не фіксується жодна наявна колізія. Отримані дані порівнянь говорить про те, що наближення створеної функції до ідеальної хеш-функції за показником рівномірності дисперсії вихідних значень реально досягне.

ВИСНОВКИ

Внаслідок проведених при розробці універсальної хеш-функції досліджень можна пересвідчитися в тому, що однією з базових вимог до хеш-функцій із точки зору її якості та стійкості є мінімізація кількості так званих колізій. Але повністю уникнути їх не вдасться.

При використанні методів боротьби з колізіями в хеш-таблицях, таких як:

- збільшення розмірів вхідних блоків;
- ускладненням процесу згортки повідомлення;
- збільшенням кількості ітерацій, інверсії бітів та розміру самого хешу;
- модифікація процесу підготовки блоків,

вдалося отримати більш стійку та якісну універсальну хеш-функцію для прикладного застосування.

При програмній реалізації створеного алгоритму універсальної хеш-функції з використанням універсального середовища Spider/Anaconda мовою Python можна отримати наступні висновки:

- існуючі алгоритмів хешування MD5 і SHA-1 є надійними для невідповідальних застосувань;
- підтверджується пряма залежності ймовірності появи колізій хеш-значень від розміру хешу (чим менша довжина значення, тим більша ймовірність появи колізії).
- після проведених засобів модернізації алгоритму універсальної хеш-функції, кількість зафіксованих випадків колізій впала до порівняною з аналогами MD5 і SHA-1.

Проведена модернізація хеш-функції доводить гарну якість запропонованого алгоритму функції user-32.

Через те, що для розробленої функції не може існувати жодного успішного варіанту успішної криптоатаки, то її використання можливо 1 в

більш спеціалізованих і відповідальних сферах – кібербезпеці. Дана функція через наявність в ній швидких бітових операцій обіцяє бути «легкою» в апаратній реалізації на вбудованих архітектура та на базі архітектури x86.

Загалом, за результатами тестування можна вважати розробку перспективною в рамках прикладних застосувань.

Вважається за потрібне перевірити стійкість отриманої універсальної хеш-функції до атаки по «райдужним» таблицям та розробка ad-hoc методу криптоаналізу функції user. Також правильним підходом до більш повного тестування отриманого алгоритму універсальної хеш-функції є її побудова в середовищі симуляційного моделювання типу MicroCap або аналогічних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Брюс Шнайер. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. — М.: Триумф, 2002. — ISBN 5-89392-055-4.

2. Лапонина О.Р. Криптографические основы безопасности. — М.: Интернет-университет информационных технологий - ИНТУИТ.ру, 2004. — С. 320. — ISBN 5-9556-00020-5.

3. П. В. Кравчук. Аналіз застосування функції гешування у технології BLOCKCHAIN / П. В. Кравчук, І. Д. Горбенко, А. І. Пушкарьов // Прикладная радиоэлектроника, 2018, Том 17, № 3, 4. – С. 147-151.

4. Andreas M. Antonopoulos Mastering Bitcoin: Unlocking Digital Cryptocurrencies / Andreas M. Antonopoulos – K.: NGITS, 2014. – С. 10–150.

5. Don Tapscott, Alex Tapscott Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World / Don Tapscott, Alex Tapscott Blockchain – K.: Information Systems, 2016 – С. 65–102.

6. Геш-функція. Картка даних терміну: [арх. 23.09.2017] // Українське агентство зі стандартизації. — Дата звернення: 23.09.2017.

7. Лекция 6 БСит [интернет-ресурс]. Режим доступу: https://bit.nmu.org.ua/ua/student/metod/cryptology/%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D1%8F17.pdf.м — Дата звернення: 13.04.2022.

8. Ramakrishna, M. V.; Zobel, Justin (1997). "Performance in Practice of String Hashing Functions". Database Systems for Advanced Applications '97. DASFAA 1997. pp. 215–224. CiteSeerX 10.1.1.18.7520. doi:10.1142/9789812819536_0023. ISBN 981-02-3107-5. S2CID 8250194. Retrieved 2021-12-06.

9. Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, Yarik Markov. The first collision for full SHA-1. CWI Amsterdam/Google Research.

10. D. Eastlake. US Secure Hash Algorithm 1 (SHA1), Cisco Systems. September 2001, 3d Motorola, Network Working Group. Access mode: https://www.ietf.org/rfc/rfc3174.txt

11. Криптоаналіз хеш-функцій [інтернет-ресурс]. Режим доступу: https://studref.com/403722/informatika/kriptoanaliz_funktsiy - — Дата звернення: 14.04.2022.

12. Хэш-функции, хеш и хеширование. Криптографическая соль. Інтернет-ресурс. Режим доступу:- https://intellect.icu/khesh-funktsii-khesh-i-kheshirovanie-kriptograficheskaya-sol-5748— Дата звернення: 15.04.2022.

13. R.C. Merkle. A Certified Digital Signature. In Advances in Cryptology - CRYPTO '89 Proceedings, Lecture Notes in Computer Science Vol. 435, G. Brassard, ed, Springer-Verlag, 1989, pp. 218-238.

14. I. Damgård. *A Design Principle for Hash Functions*. In Advances in Cryptology - CRYPTO '89 Proceedings, Lecture Notes in Computer Science Vol. 435, G. Brassard, ed, Springer-Verlag, 1989, pp. 416-427.

15. G. Leurent. SHA-1 is a Shambles / G. Leurent , Thomas Peyrin // Computer Science. Published 2020 Access mode: https://www.semanticscholar.org/paper/SHA-1-is-a-Shambles-Leurent-Peyrin/b3acfb1793ba495b165053e90c60030aac75e22d

16. Valeriy, L., Andrii, S., Vladyslav, K., Elena, P., Anatolii, C., Nataliia, U. (2022). Evaluation of the Probability of Breaking the Electronic Digital Signature Elements. In: Karrupusamy, P., Balas, V.E., Shi, Y. (eds) Sustainable Communication Networks and Application. Lecture Notes on Data Engineering and Communications Technologies, vol 93. Springer, Singapore. https://doi.org/10.1007/978-981-16-6605-6_48

17. Vincent Rijmen and Elisabeth Oswald. Update on SHA-1. Access mode: https://eprint.iacr.org/2005/010.pdf

18. Pasini, Sylvain ; Vaudenay, Serge. Hash-and-Sign with Weak Hashing Made Secure. Lecture Notes in Computer Science, 4586. 12th Australasian

Conference on Information Security and Privacy: ACISP '07, Townsville, Queensland, Australia, July 2-4, 2007. https:// doi:10.1007/978-3-540-73458-1_25.

19. Lu, Yi; Vaudenay, Serge; Meier, Willi; Ding, Liping; Jiang, Jianchun. Synthetic Linear Analysis: Improved Attacks on CubeHash and Rabbit. Lecture Notes in Computer Science, CISC, Seoul, Korea, November 30 - December 2, 2011. 7259. Pp. 248-260. https:// doi: 10.1007/978-3-642-31912-9_17

20. RainbowCrack. 2020, RainbowCrack Project. Access mode: http://project-rainbowcrack.com/