НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»
УДК _____

До захисту допущено:
Завідувач кафедри
_____Сергій СТІРЕНКО
«___»_____2023 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Інженерія програмного забезпечення комп'ютерних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Спосіб криптографічно-строгої ідентифікації на основі хеш-перетворень з програмованими колізіями»

Виконала:
студентка ІІ курсу, групи ІМ-11мн
Стеблевець Тетяна Олександрівна _____

Керівник:
доцент каф. ОТ, к.т.н.
Марковський Олександр Петрович _____

Консультант з нормоконтролю:
проф. каф. ОТ, д.т.н.
Жабін Валерій Іванович _____

Рецензент:
декан ФПМ, д.т.н., проф.
Дичка Іван Андрійович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність *121. Інженерія програмного забезпечення*
(код і назва)

Спеціалізація *121. Інженерія програмного забезпечення комп'ютерних систем*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Сергій СТІРЕНКО
(підпис) (ініціали, прізвище)
« » 2023 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Стеблевець Тетяні Олександрівні
(прізвище, ім'я, по батькові)

1. Тема дисертації «*Спосіб криптографічно-строгої ідентифікації на основі хеш-перетворень з програмованими колізіями*»
Науковий керівник дисертації *Марковський Олександр Петрович, доцент каф. ОТ, к.т.н.*
затверджені наказом по університету від «20» березня 2023 р. № 1275-С

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження: *процеси криптографічно строгої ідентифікації учасників віддаленої інформаційної взаємодії на основі незворотних хеш-перетворень з програмованими колізіями*.

4. Предмет дослідження: *методи побудови незворотних хеш-перетворень з програмованими колізіями для криптографічно строгої ідентифікації учасників віддаленої інформаційної взаємодії.*

5. Перелік завдань, які потрібно розробити: *виконати огляд існуючих методів ідентифікації віддалених користувачів, розробити та дослідити метод побудови*

хеш-перетворення з програмованими колізіями ,реалізувати програмні засоби відповідно запропонованого методу

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	*Жабін В.І., проф.*		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	*Огляд предметної області та вивчення літератури*	*16.01.2023-05.02.2023*	
2	*Складання і узгодження технічного завдання*	*06.02.2023-18.02.2023*	
3	*Написання вступної частини та огляд рішень*	*20.02.2023-26.02.2023*	
4	*Моделювання розробленого способу*	*27.02.2023-02.04.2023*	
5	*Оформлення документації*	*03.04.2023-06.05.2023*	
6	*Подання ДП рецензенту*	*16.05.2023*	
7	*Захист*	*22.05.2023*	

Студент _____ Стеблевець Т.О.
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Марковський О. П.
(підпис) (ініціали, прізвище)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Спосіб криптографічно-строгої ідентифікації на основі хеш-перетворень з програмованими колізіями

студенткою: Стеблевець Тетяною Олександрівною

Робота складається зі вступу та чотирьох розділів. Загальний обсяг роботи: 90 аркушів основного тексту, 25 ілюстрацій, 2 таблиці, додатки. Перелік використаних джерел налічує 70 найменувань.

Актуальність. Динамічний прогрес Інтернет та пандемія COVID-19 стимулюють швидкий розвиток технологій віддалених форм інформаційної взаємодії у всіх сферах людської діяльності. На сьогоднішній день Інтернету став основним засобом обміну даними в системах комп'ютерного моніторингу об'єктами реального світу та управління ними, зокрема в Інтернеті речей. Розширення використання технологій віддаленої Інтернет взаємодії в банківській сфері, оборот платіжних коштів, а також зростаюча комерціалізація надання інформаційних послуг стимулює активізацію комп'ютерних злочинів.

В зазначених умовах об'єктивно зростає значення контролю потоків даних і розмежування прав доступу до них. Ключову роль при цьому відіграє забезпечення ефективної ідентифікації та автентифікації суб'єктів віддаленої інформаційної взаємодії.

Динамічне розширення використання систем віддаленої інформаційної взаємодії в усіх сферах людської діяльності педалює активізацію протиправних дій, направлених на втручання в процеси віддаленої інформаційної взаємодії. Особливо активно в останні роки вдосконалюються методи атак на процеси ідентифікації учасників віддаленої інформаційної взаємодії. Це об’єктивно вимагає вдосконалення механізмів ідентифікації з метою підвищення їх захисних

властивостей. Добре відомо, що найбільш повний спектр захисних функцій реалізується при використанні криптографічного строгих механізмів ідентифікації.

Водночас розширення застосування мережевих технологій віддаленої інформаційного взаємодії в системах реального часу, Інтернеті речей, а також динамічне зростання кількості користувачів диктують необхідність підвищення швидкості засобів ідентифікації.

Таким чином, особливості сучасного етапу розвитку технологій віддаленої взаємодії вимагають вирішення наукової задачі пошуку нових шляхів підвищення ефективності засобів ідентифікації та автентифікації суб'єктів такої взаємодії.

Мета і завдання дослідження. Мета магістерського дослідження полягає в підвищенні ефективності криптографічного строгої ідентифікації на основі хеш-перетворень з програмованими колізіями за рахунок збільшення кількості сеансових паролів доступу.

Для досягнення поставленої в магістерській дисертації мети дослідження сформульовано і вирішено такі задачі :

1. Аналіз сучасного стану використання механізмів ідентифікації учасників віддаленої інформаційної взаємодії, формулювання вимог до засобів ідентифікації цих учасників. Огляд, з позицій сформульованих вимог, існуючих методі та засобів ідентифікації учасників віддаленої інформаційної взаємодії, виявлення їх недоліків в сучасних умовах та перспективі. Визначення можливих шляхів підвищення ефективності засобів ідентифікації учасників віддаленої інформаційної взаємодії, обґрунтування найбільш перспективних з них.

2. Розробка та дослідження методу побудови хеш-перетворення з програмованими колізіями для криптографічно строгої ідентифікації, який відрізняється тим, що ідентифікатором учасника віддаленої взаємодії виступає не фіксований код, а певна сукупність кодів, біти яких пов’язані певними криптографічними відношеннями, які залежать від секретного ключа, за рахунок чого збільшується кількість колізій хеш-перетворення і, тим самим на значною міною знімається

обмеження на кількість можливих сеансових ключів криптографічно строгої ідентифікації.

3. Створення програмних засобів для експериментального дослідження ефективності запропонованого методу побудови хеш-перетворення з програмованими колізіями, а також визначення залежностей між показниками ефективності криптографічно строгої ідентифікації та параметрами хеш-перетворення.

Об'єкт дослідження – процеси криптографічно строгої ідентифікації учасників віддаленої інформаційної взаємодії на основі незворотних хеш-перетворень з програмованими колізіями.

Предмет дослідження – методи побудови незворотних хеш-перетворень з програмованими колізіями для криптографічно строгої ідентифікації учасників віддаленої інформаційної взаємодії.

Методи досліджень базуються на базових засадах теорії незворотних перетворень, булевої алгебри, теорії рекурсивних функцій, теорії ймовірності, а також на основних положеннях статистичного та імітаційного моделювання.

Наукова новизна одержаних результатів роботи полягає у наступному:

1. Вперше запропоновано метод побудови хеш-перетворення з програмованими колізіями для криптографічно строгої ідентифікації, який відрізняється тим, що ідентифікатором учасника віддаленої взаємодії виступає не фіксований код, а певна сукупність кодів, біти яких пов'язані певними криптографічними відношеннями, які залежать від секретного ключа, за рахунок чого збільшується кількість колізій хеш-перетворення і, тим самим на значною міною знімається обмеження на кількість можливих сеансових ключів криптографічно строгої ідентифікації.

Практична значимість отриманих результатів визначається тим, що запропонований метод побудови хеш-перетворення з програмованими колізіями для криптографічно строгої ідентифікації дозволяє значно прискорити процес

синтезу зазначеного перетворення при збільшенні кількості можливих сеансових паролів доступу. Збільшена кількість сеансових паролів, яка забезпечується запропонованим методом дозволяє збільшити число циклів ідентифікації до переналаштування системи.

Особистий внесок здобувача полягає в теоретичному обґрунтуванні одержаних результатів, їх експериментальній перевірці та дослідженні, а також у створенні програмних продуктів для практичного використання одержаних результатів.

Апробація результатів дисертації. Основні результати дисертації доповідались та обговорювались на міжнародниій науково-технічній конференції “The International Conference on Security, Fault Tolerance, Intelligence”, м.Київ, 12-13 травня 2021 р.

Публікації. Основні положення магістерської дисертації опубліковані в науковій праці, що належить до матеріалів науково-технічної конференції.

Ключові слова: криптографічно строга ідентифікація, ідентифікації на основі концепції нульового розголошення, незворотні хеш-перетворення.

Пояснювальна записка

до магістерської дисертації

на тему: «Спосіб криптографічно-строгої ідентифікації на основі хеш-перетворень з програмованими колізіями»

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АНФ	Алгебраїчна нормальна форма
БФП	Булеве функціональне перетворення
ЗСР	Зворотний зсув рядків матриці стану
ПЗ	Пряме заміщення байту мультиплікативною інверсією
ПЗР	Прямий зсув рядків матриці стану
ФП	функціональне перетворення
ХП	Хеш-перетворювач
SHA	Збірна назва сімейства хеш-функцій (Secure Hash Algorithm)
AES	Покращений стандарт шифрування (Advanced Encryption Standard)
DES	Стандарт шифрування даних (Data Encryption Standard)
FFSIS	Схема ідентифікації Фейга-Фіата-Шаміра (Feige Fiat Shamir Identification Scheme)
GF	Поля Галуа (Galoise Fields)
SAC	Строгий лавинний критерій (Strict Avalanche Criterion)
JVM	Віртуальна машина Java (Java Virtual Machine)
XOR	Сума за модулем два (eXclusive OR)

ВСТУП

В сучасних умовах інформаційна інтеграція стала головним фактором у досягненні якісно нового рівня обробки інформації в усіх галузях людської діяльності. Наразі, мережа Інтернет є не лише засобом міжкористувацької взаємодії, а і основним інструментом обміну даними в системах комп'ютерного моніторингу об'єктів реального світу та управління ними, зокрема в Інтернеті речей. Динамічне розповсюдження хмарних технологій, що надають можливості доступу до значних обчислювальних потужностей, розширення використання технологій віддаленої взаємодії в банківській сфері, обіг платіжних коштів і зростаюча комерціалізація надання інформаційних послуг – все це надало нового потужного імпульсу зростанню важливості питання інформаційної безпеки та встановлення якісного контролю доступу до інформаційних ресурсів, адже динамічне розширення можливостей використання систем віддаленої взаємодії відповідним чином викликає розширення зони ризиків, пов’язаних з можливостями встановлення несанкціонованого доступу до інформаційних та обчислювальних ресурсів цих систем.

Одним з ключових елементів контролю доступу до даних у віддалених системах є процедура ідентифікації віддаленого абонента. Ця процедура повинна гарантувати, що віддалений користувач системи колективного доступу до інформаційних ресурсів має усі необхідні права для доступу до них. Це викликає потребу застосування ефективних та безпечних механізмів ідентифікації віддалених користувачів. Для цього можуть бути використані різноманітні методи, такі як автентифікація з використанням паролів, біометричні методи (відбиток пальця, сканування обличчя), криптографічні методи (електронні підписи, шифрування) тощо.

Ключовими критеріями оцінки ефективності систем, направлених на реалізацію процесу ідентифікації віддалених користувачів є рівень їх захищеності

від потенційних ризиків встановлення несанкціонованого доступу до системи та власне час, необхідний для виконання операції ідентифікації деякого віддаленого користувача. Окрім того, з огляду на різке прискорення темпів зростання обсягів застосування систем віддаленої взаємодії, ще одним важливим показником ефективності таких механізмів є їх здатність до масштабування, тобто можливості виконання поставлених перед ними завдань в повній мірі навіть в умовах високого навантаження.

Наразі, більшість протоколів ідентифікації користувачів у віддалених системах ґрунтуються на концепції «нульових знань», відповідно якій для підтвердження автентичності абонента, необхідно, аби він неявно довів факт володіння деякою конфіденційною інформацією, яка системі, в свою чергу, не є відомою, проте наявність якої у абонента система може перевірити. Це означає, що жодна секретна інформація, яка може бути застосована для відновлення ідентифікаційних даних абонента, не зберігається в системі, що дозволяє гарантувати захист від втручання в процес ідентифікації як від зовнішніх, так і внутрішніх загроз.

Проте, так як існуючі методи реалізації вказаної концепції мають у своїй основі незворотні перетворення теорії чисел, це означає значну обчислювальну складність цих операцій, які, в свою чергу, призводить до потреби залучення додаткових обчислювальних та часових ресурсів, що значним чином зменшує їх ефективність. Особливо гострою ця проблема є для портативних вбудованих мікроконтролерів, які часто виконують роль термінальних пристроїв, адже є обмеженими в продуктивності та можливостях енергоспоживання.

Отже, завдання підвищення ефективності механізмів криптографічно строгої, заснованої на основі концепції «нульових знань», ідентифікації віддалених абонентів у багатокористувацьких системах та системах з обмеженими обчислювальними потужностями наразі є надзвичайно важливою та актуальною.

РОЗДІЛ 1

АНАЛІЗ СУЧАСНИХ ВИМОГ ДО КОНТРОЛЮ ДОСТУПУ ДО ДАНИХ ТА ОГЛЯД ІСНУЮЧИХ МЕТОДІВ ІДЕНТИФІКАЦІЇ

Швидкий технологічний прогрес глобальних мереж спричинив значне розширення впливу систем віддаленої інформаційної взаємодії. Кількість систем колективного доступу та їх користувачів різко зросла. Це викликало особливу увагу до проблем контролю доступу до інформаційних ресурсів в усіх основних областях, пов’язаних з збереженням та обробкою даних. У таких системах механізм ідентифікації віддалених користувачів є ключовим засобом контролю доступу до інформації. Внаслідок цього, вимоги до надійності захищеності та швидкості виконання процедури ідентифікації віддалених користувачів невпинно зростають.

Отже, розробка ефективних засобів та методів ідентифікації користувачів віддалених систем для гаранту захищеного доступу до інформаційних ресурсів потребує попереднього аналізу сучасного ринку технологій віддаленої взаємодії, визначення та формулювання вимог до сучасних засобів контролю доступу до даних, а також, з урахуванням цих вимог, критичного огляду існуючих засобів та методів ідентифікації віддалених користувачів.

1.1 Аналіз засобів незаконного доступу до інформаційних ресурсів

Швидкий розвиток телекомунікацій та Інтернет-технологій призвів до широкого розповсюдження розподілених систем, що надають користувачам віддалений доступ до інформаційних, програмних та апаратних ресурсів інформаційних систем. У загальній структурі логічної взаємодії системи, яка надає інформаційні ресурси користувачам, ключовою роллю є розподілення прав доступу користувачів до ресурсів та послуг системи, що забезпечує виключення можливості

отримання незаконного доступу до інформаційних ресурсів. Один з основних інструментів для розподілу прав доступу - механізми ідентифікації. Структура логічної взаємодії системи, яка забезпечує доступ користувачів до інформаційних ресурсів, узагальнена на рисунку 1.1.

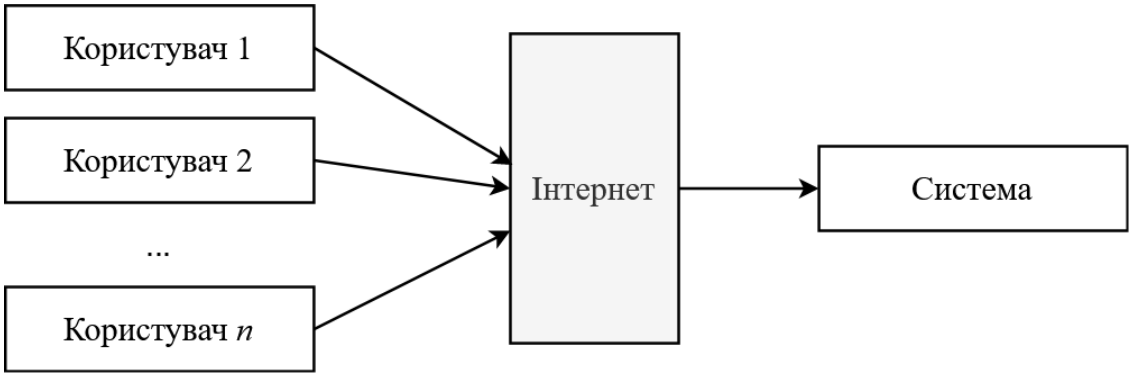

Рис.1.1. Структура взаємодії віддалених користувачів та системи

Ідентифікація є процесом, що полягає у визначенні особи в системі за допомогою попередньо встановленого ідентифікатора або інших попередніх даних, що система може розпізнати. Перед початком роботи в системі користувач повинен виконати процедуру реєстрації, після якої, користувачеві надається унікальний ідентифікатор або пароль, який він повинен використовувати при вході в систему. Ідентифікація віддалених користувачів забезпечує захист ресурсів багатокористувацьких систем від незаконного доступу та є однією з трьох основних задач захисту інформації.

Ефективність ідентифікації віддалених користувачів віддалених систем зумовлена двома ключовими показниками:

- мірою захисту інформаційних чи обчислювальних ресурсів деякої віддаленої комп’ютерної системи від спроб здобуття незаконного доступу;

- величиною об’ємів обчислюваних ресурсів, які необхідні до застосування користувачем та власне системою при виконанні механізмів ідентифікації.

Таким чином, ефективність рівня захищеності ідентифікації визначається мірою складності встановлення незаконного доступу зловмисником до ресурсів відданої системи та межею витрат ресурсів на забезпечення механізму ідентифікації самою системою і користувачем для виконання цієї операції. Звідси, серед залучених ресурсів розглядаються:

- обчислювальні потужності системи (час обробки та кількість процесорів),

- об’єми використання пам’яті комп’ютерної системи.

Один із найбільш складних елементів оцінки ефективності процесу ідентифікації – це критерій захисту, який визначає ступінь захищеності системи. З теоретичної точки зору, рівень захисту може бути визначений шляхом вимірювання об’єму ресурсів, необхідних для зламу системи захисту. Проте, отримання такої оцінки складається потребує значних зусиль у зв’язку з широким різноманіттям методів, які зловмисник може застосувати для зламу систем захисту. Таким чином, для оцінки рівня захищеності, що забезпечують конкретні засоби ідентифікації віддалених користувачів, потрібно ретельно досліджувати відповідні можливі типи атак, направлені на систему контролю доступу до інформаційних даних.

З огляду на розташування зловмисника відносно системи ідентифікації при спробі встановлення несанкціонованого доступу до системи можна виділити наступні класи атак:

- атаки зі сторони легального користувача. У такому разі, володіючи певною інформацією щодо власне механізму ідентифікації, зловмисник пробує досягти доступу до ресурсів системи шляхом підбору одного із паролів легальних користувачів. Нехай в системі є n користувачів, а середній пароль має розрядність у k-біт, тоді ймовірність підбору зловмисником пароль за q спроб P можна обчислити за формулою:

$$P = (1 - 2^{-k})^{n \cdot q}. \qquad (1.1)$$

Звідси є очевидним, що ймовірність підбору паролю легко і ефективно може бути значно зменшеною шляхом збільшення його розрядності. Варто зауважити,

що зловмисник, як правило, не виконує перебір всі можливих паролів, зосереджуючи увагу лише на найбільш ймовірних.

- атаки зі сторони зловмисника, що володіє доступом до каналу, призначеного для обміну даних між користувачами і системою. Атаки з боку зловмисника, який має несанкціонований доступ до каналу обміну даними між системою та користувачами, є серйозною загрозою для безпеки інформаційних ресурсів системи. Зловмисник може отримати доступ до одного або кількох центрів комутації мережі Інтернет і прослуховувати повідомлення, які обмінюються система та користувачі в процесі ідентифікації. Шляхом аналізу такого прослуховування зловмисник може отримати пароль та використовувати його для незаконного доступу до інформаційних ресурсів системи. Описана атака належить до категорії пасивних атак (див. Рис. 1.2). Зловмисник також може використовувати активну тактику, що полягає в перехопленні взаємодії системи та легітимного користувача після того, як останній успішно пройшов процес ідентифікації від імені системи.

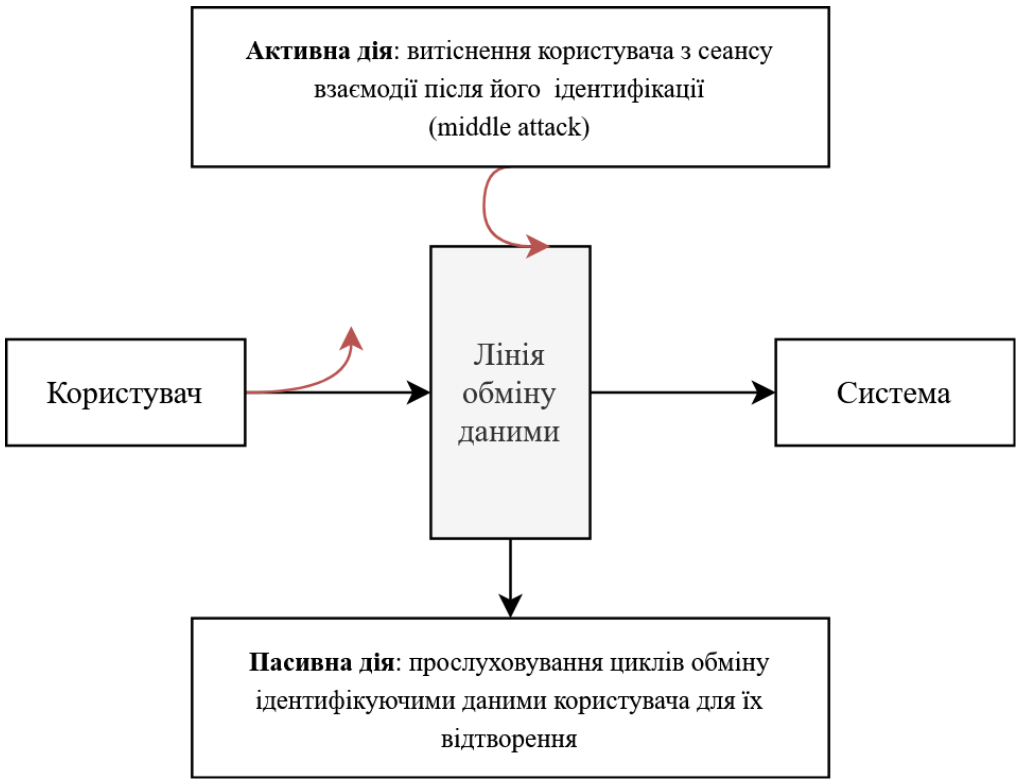

Рис.1.2. Схема порушення прав доступу стороннім зловмисником

- атаки зі сторони зловмисника всередині системи. Можливість проникнення в систему може бути спричинена економічними мотивами, що полягають у намаганні зловмисника отримати доступ до ресурсів системи за рахунок легальних користувачів. За допомогою вірусних програм зловмисник може здобути доступ до ідентифікаційної інформації, що зберігається в системі. Також це можливо, якщо він виступає в якості співробітника системи.

Захист інформації можна здійснювати за допомогою різних методів ідентифікації, серед яких можна традиційно виділяють такі типи як криптографічно слабка та строга ідентифікація. У випадку з криптографічно слабкою ідентифікацією розглядається небезпека отримання незаконного доступу з боку зловмисника ззовні системи, який володіє можливостями організації пасивного або активного доступ до деякого каналу обміну інформацією між віддаленим користувачем та системою. При такому підході, система, з якою відбувається взаємодія, вважається користувачем захищеною від зловмисника. У свою чергу, строга ідентифікація базується на збереженні інформації в механізмі контролю самої системи, яка може бути застосована для отримання доступу до інформаційних ресурсів від імені користувача. Фактично, цей тип ідентифікації уособлює важливий принцип єдиного власника деякої секретної інформації.

Очевидно, що кожна з можливих атак на систему, спрямованих на незаконне отримання доступу до її інформаційних або обчислювальних ресурсів, потребує витрат певних ресурсів з боку зловмисника. Кожен вид атаки вимагає від системи надійного захисту від проникнення та несанкціонованого доступу до її ресурсів. Система повинна бути захищена від спроб несанкціонованого доступу до її ресурсів, наприклад, шляхом повторного використання одного і того ж пароля або ж його зашифрованої форми, яку можна отримати шляхом перехоплення при прослуховуванні каналу передачі даних. Це можливо забезпечити за допомогою широкого спектру засобів протидії незаконному доступу до ресурсів системи,

зокрема, використанням часових міток, які включаються в пароль, та шифруванням пароля відкритим ключем віддаленої системи.

Отже, розгляд сучасних можливостей отримання незаконного доступу до інформаційних та обчислювальних ресурсів віддаленої системи зловмисником, можна виділити наступні вимоги до систем ідентифікації віддалених користувачів:

- щоб попередити виконання несанкціонованого доступу до інформаційних ресурсів шляхом перехоплення ідентифікаційних даних під час пасивного прослуховування каналу обміну даних між користувачем і системою, при кожному наступному сеансі ідентифікації пароль користувача повинен змінюватися;
- ідентифікація віддалених користувачів повинна відбуватися таким способом, щоб передані по каналу взаємодії ідентифікаційні дані не були достатніми для відтворення ідентифікації віддаленого користувача. З цього слідує, що в процесі звірки доступу користувача до ресурсів системи, повинна бути залучена деяка секретна інформація, що має зберігатися як на стороні користувача, так і системи;
- власне сама система не може зберігати дані чи інформацію, застосування яких надає змогу відновити деякий сеансовий пароль віддаленого користувача в повній мірі;
- механізм ідентифікації віддалених користувачів повинен бути орієнтований на високе навантаження і відповідним чином оптимізованим з вимогою одночасного обслуговування великої кількості користувачів, таким чином це породжує потребу забезпечення високу швидкодію під час виконання процесу ідентифікації абонентів.

Оцінюючи ефективність схем ідентифікації віддалених користувачів, важливо враховувати складність обчислювальної реалізації. Обчислення, пов'язані з ідентифікацією, зазвичай виконуються на потужних комп'ютерних платформах, що належать системі. Однак, на стороні віддаленого користувача, виконуються обчислення, пов'язані з реалізацією протоколу ідентифікації, які можуть

виконуватися на різних пристроях, включаючи вбудовані термінальні контролери, які є більш простими і менш потужними. У такому випадку, ефективність реалізації обчислень, пов'язаних з ідентифікацією, є критичною проблемою, яка повинна бути врахована при виборі схеми ідентифікації.

Аналіз сучасних засобів ідентифікації віддалених користувачів показує, що зростання комерційної складової віддаленого надання доступу до інформаційних та обчислювальних ресурсів для широкого кола користувачів збільшує значимість надійного контролю за доступом до цих ресурсів. Це зумовлює використання криптографічно строгої ідентифікації як більш перспективного рішення, яке забезпечує високий рівень захисту інформації.

1.2 Огляд та аналіз сучасних схем криптографічно строгої ідентифікації

На сьогоднішній день існує ряд схем, які призначені забезпечувати вимоги криптографічно строгої ідентифікації, більшість з яких базується на застосуванні криптографічної концепції «нульових знань». Основна ідея цієї концепції полягає у єдиному власнику механізму утворення паролів, таким чином користувач має певний секретний механізм генерації валідних сеасових паролів, водночас як система має лише механізм перевірки «правильності» цих паролів. Важливою умовою є використання незворотних математичних перетворень в механізмі перевірки цих паролів. У сучасній криптографії використовуються два типи незворотних математичних перетворень, які відносяться до задач теорії чисел та булевої алгебри, які не можуть бути розв’язані аналітичним шляхом.

Перші розроблювані схеми криптографічно строгої ідентифікації були засновані на незворотних математичних перетвореннях з розділу теорії чисел, зокрема на задачі дискретного логарифмування.

Однією з таких схем є FFSIS, що є відносно простою і, водночас, ефективною схемою ідентифікації віддалених користувачів. Для реалізації цієї схеми, системою

генерується деяке 1024-розрядне число m, яке є добутком двох простих чисел p та q, тобто $m = p \cdot q$. Далі, це число m розповсюджується серед користувачів системи. Для створення відкритого та закритого ключів для конкретного користувача, застосовується стандартна процедура генерації ключів криптографічного алгоритму RSA.

В подальшому, протокол ідентифікації віддаленого користувача в системі передбачає виконання послідовності дій наступного характеру:

1. Віддалений користувач генерує деяке випадкове число t, менше m: $t<m$. Далі обчислюється $z = (t^2) \bmod m$ і число z передається до системи.

2. Система надсилає користувачеві випадково згенерований біт β.

3. Якщо отриманий користувачем біт β є рівним нулю, у такому разі користувач надсилає системі попередньо обране число t: $u = t$. В протилежному випадку, тобто якщо $\beta = 1$, тоді користувач виконує обчислення значення u за формулою $u = (t \cdot \chi) \bmod m$ та надсилає його до системи, де χ – це мультиплікативна інверсія знаяення t^2, тобто $\chi \cdot t^2 \bmod m = 1$.

4. Далі система приймає значення, надіслане на попередньому кроці користувачем та виконує його аналіз за наступним алгоритмом: якщо біт β, раніше згенерований системою, є рівним нулю, тоді виконується перевіряється виконання умови $u = t^2 \bmod m$, що можу бути задоволеною лише якщо користувачу відомо значення $\sqrt{t}$. Інакше, якщо біт $\beta = 1$, тоді система виконує перевірку справедливості умови $\chi = (t^2 \cdot u) \bmod m$. Вона задовольняється лише у разі, коли мультиплікативна інверсія квадрату числа t, знайдена користувачем є правильною.

Описана схема ідентифікації FESIS при практичному застосуванні потребує повторення наведеної вище послідовності операцій n раз, до тих пір, доки система з попередньо вказаною ймовірністю не буде впевненою, що закритий ключ системи є відомий користувачу. Значення генерованого біту β дозволяє випадковим винок виконати перевірку одночасно двох важливих вимог для гаранту захисту доступу

до системи: при $\beta = 0$ система гарантує відсутність підміни користувача, а при $\beta = 1$ – володіння користувачем закритого ключа системи.

Ще одна схема криптографічно строгої ідентифікації Guillou-Quisquater набула широкого розповсюдження у значній кількості багатокористувацьких систем. Алгоритм роботи цієї схеми в узагальненому вигляді наступний: кожен легітимний користувач системи володіє деяким відкритим паролем P, в цей час система має відкритий ключ E. Ключ E системи містить в собі число m, що є добутком двох простих чисел p та q, тобто $m=p\cdot q$, та деяке секретне число t, тоді закритим ключем u системи виступає мультиплікативна інверсія відкритого ключа користувача, тобто $P\cdot u^t \bmod m = 1$.

У такому разі схема ідентифікації користувача передбачає послідовного виконання таких дій:

1. Користувач випадковим чином утворює число s таке, щоб $s < m$. Далі, користувач обраховує значення $F = s^t \bmod m$ та передає обчислене значення F до системи.

2. Отримавши від користувача значення коду F, система виконує генерацію значення w випадковим чином так, щоб $w < m$, та передає його користувачу.

3. Після одержання користувачем від системи значення w, він виконує обчислення значення $Q = s\cdot u^w \bmod m$ та надсилає його до системи.

4. Система отримує значення Q та здійснює обчислення значення числа $V = P^w\cdot F^s \bmod m$, після чого відбувається перевірка відповідності його значення числу Q, тобто виконується істинності твердження $V = Q$. У разі ствердної відповіді, користувач вважається легітимним, що передбачає надання йому права доступу до інформаційних ресурсів системи.

У схемі ідентифікації Guillou-Quisquater, надійність захисту і неможливість підробки права надання доступу до інформаційних та обчислювальних ресурсів забезпечується шляхом генерації випадкового числа w системою в кожному сеансі ідентифікації, яке не повторюється в інших сеансах ідентифікації. Однак, порівняно

з попередньою схемою, її недоліком є більший обсяг обчислень, необхідних для її реалізації. Крім ідентифікації віддалених користувачів, дана схема активно використовується для ідентифікації інформаційних повідомлень, які надходять до системи.

Ще однією з активно використовуваних на практиці схем ідентифікації віддалених користувачів інтегрованих систем є схема Шнорра. Ця схема також ґрунтується на математично-аналітичному завданні дискретного логарифмування. Для генерації ключів спочатку обирають два прості числа p і q, такі, що q є дільником $p - 1$. Далі обирається довільне число w, яке не дорівнює нулю, і таке, щоб $w \cdot q \bmod p = 1$. Зазначені числа є загальними для групи користувачів системи та є відкритими всередині цієї групи.

Для виконання генерації пари ідентифікуючих ключів користувача, для початку обирається довільне випадкове число $t < q$, яке представляє собою приватний (секретний) ключ користувача. Далі виконується обчислення публічного (відкритого) ключа користувача за формулою $P = w \cdot t \bmod p$. Подальший механізм ідентифікації віддаленого користувача послідовного виконання кроків наступного алгоритму дій:

1. Користувач випадковим чином генерує число u таке, щоб відбувалося виконання умова $u < q$, після чого він обчислює число $z = t \cdot u \bmod p$. Далі, користувач передає код z системі по відкритому каналу зв'язку.

2. У відповідь система генерує деяке випадкове число $h < 2 \cdot t$ та передає його користувачеві.

3. Користувач обраховує значення $y = (u + z \cdot h) \bmod q$ та передає його системі.

4. Система виконує перевірку легітимності користувача шляхом обчислення значення $y^u \cdot P^h \bmod p$ та подальшого цього значення з попереднього одержаним значенням z. Користувачеві надається доступ до інформаційних ресурсів системи у разі якщо ці значення є рівними між собою.

Загальним недоліком криптографічно строгої ідентифікації, яка ґрунтується на незворотних математичних перетвореннях теорії чисел, є значна обчислювальна складність. Це унеможливлює проведення швидкої ідентифікації великої кількості користувачів, при значному навантаженні, системою в режимі реального часу.

Такий же недолік характерний для схем криптографічно строгої ідентифікації, що базуються на незворотних математичних перетвореннях алгебри кінцевих полів Галуа. Проте, такі схеми гарантують значно більшу швидкодію у порівнянні з традиційною алгеброю. Використання алгебри полів Галуа для криптографічних застосувань має перевагу у тому, що можна генерувати безліч алгебраїчних базисів за рахунок вибору утворюючого поліному поля, результати операцій в яких різні. Крім того, порівняно з модулярною арифметикою, обчислювальна реалізація полів Галуа є значно простішою. Це дозволяє виконувати криптографічні перетворення значно швидше, особливо при апаратній реалізації.

Для криптографічних застосувань, як для модулярної арифметики, так і полів Галуа, ключовою операцією є експоненціювання, що є базовою складовою для багатьох схем криптографічної ідентифікації. Особливо важливою є її роль у ключових протоколах криптографічної ідентифікації, які використовують алгебру кінцевих полів Галуа. При обробці чисел з розрядністю *n*-біт, обчислювальна складність експоненціювання на кінцевих полях Галуа становить $O(n^3)$. Це означає, що при подвоєнні розрядності, обсяг необхідних обчислень зростає у вісім разів. В сучасних умовах, коли злом криптографічного захисту можливий за допомогою обчислювально потужних хмарних технологій, для забезпечення безпеки даних необхідно збільшувати розрядність чисел. Однак, це призводить до збільшення часу обчислення криптографічних протоколів ідентифікації віддалених користувачів.

Для того, щоб підвищити швидкість криптографічної ідентифікації віддалених користувачів, можна застосовувати альтернативні технології, які ґрунтуються на незворотних задачах булевої алгебри.

Зокрема, було запропоновано використання послідовності сеансових паролей, які сформовані з використанням ланцюжка стандартизованих хеш-перетворень. Стандартизований хеш-перетворювач є сертифікованим алгоритмом, який використовується для незворотного перетворення блоку інформації довільної довжини на код хеш-сигнатури фіксованої довжини *h*. SHA-1 та RIPEMD-160 є найбільш значущими прикладами таких хеш-перетворювачів.

Головна особливість стандартизованих хеш-перетворювачів полягає у забезпеченні незворотності перетворення. Це означає, що практично неможливо знайти сукупність вхідних векторів, які призводять до отримання результату хеш-перетворення. Наприклад, для хеш-перетворення RIPEMD-160, при 512-бітовому вхідному коді та 160-бітовому вихідному коді, існує 10^{105} вхідних кодів, які дають однаковий результат хеш-перетворення, проте ймовірність їхнього знаходження методом перебору складає близько 10^{-14}. Відповідні методи криптографічно строгої ідентифікації передбачають виконання комбінації двох операцій: початкової реєстрації користувача та механізм сеансової ідентифікації.

Механізм реєстрації віддаленого користувача у системі передбачає послідовного виконання алгоритму дій:

1. Користувач випадковим чином генерує *m*–тий сеансовий пароль Q_m.

2. Індекс сеансового паролю *j* користувача встановлюється рівним *m-1*, тобто $j = m-1$.

3. Користувач виконує обчислення *j*-того сеансового пароль Q_j шляхом стандартизованого хеш-перетворення над попереднім відомим сеансовим паролем, тобто $Q_j = H(Q_{j+1})$. В подальшому, утворений пароль зберігається користувачем в області захищеної пам'яті його обчислювальної платформи.

4. Виконується декремент індексу паролю, тобто $j = j-1$. У разі якщо $j > 0$, то виконується перехід до повторного виконання п.4.

5. Останній із утворених результуючих паролів Q_0 надсилається користувачем до системи. В подальшому, система зберігає цей пароль в якості ідентифікуючого коду користувача G, тобто $G = Q_0$.

Зрозуміло, що система, яка має ідентифікуючий код віддаленого користувача, не може сама відтворити код Q1. Це пов'язано з тим, що виконання цієї задачі еквівалентне злому стандартизованого хеш-перетворення, яке вважається практично нереалізованим.

Варто зауважити, що стандартизовані хеш-перетворення пройшли багаторічну перевірку та тестування в тисячах організацій та відповідають всім необхідним вимогам. Використання стандартизованого хеш-перетворення практично виключає можливість використання будь-яких методів підбору вхідних даних за заданим вихідним кодом, крім повного перебору. Проте, реалізація повного перебору вимагає значних ресурсів, які перевищують технічні можливості та не є комерційно доцільними.

В подальшому, процедура ідентифікації сеансу віддаленого користувача потребує виконання послідовного виконання алгоритму дій:

1. Користувач передає системі віддаленого доступу пароль Q_j, що відповідає j–тому сеансу ідентифікації, з тих, що були попередньо збережені в захищеній області пам'яті на його обчислювальній платформі.

2. Система отримує сеансовий пароль Q_j та здійснює над ним обчислення результату стандартизованого хеш-перетворення $Y = H(Q_j)$. У разі, якщо обчислене значення Y є рівним поточному коду G користувача, тобто $Y = G$, то відбувається надання доступу до інформаційних ресурсів системи. Після цього відбувається заміна значення поточного ідентифікаційного коду користувача G, тобто $G = Q_j$.

Використання подібного методу дозволяє досягти значно вищих показників швидкодії в порівнянні з методом з застосуванням модулярного експоненціювання чисел великої розрядності, оскільки обчислення хеш-перетворення здійснюється швидше на 3-4 порядки. Ще більший ефект прискорення можна отримати за

рахунок використання кращих апаратних засобів. Переважна більшість криптопроцесорів, які наразі використовуються у персональних комп’ютерах, здатні апаратно реалізувати стандартизовані хеш-перетворення.

Однак, ця схема ідентифікації має суттєвий недолік – вона є вразливою до порушень синхронізації, які можуть бути викликані різними факторами, такими як збої та перешкоди в роботі каналу передачі даних або зловмисний вплив.

Отже, можна зробити висновок, що існуючі схеми ідентифікації, які базуються на нерозв'язних задачах теорії чисел і полів Галуа, наразі не досягають балансу достатніх швидкості і рівня захищеності при ідентифікації віддалених користувачів.

Проведений аналіз існуючих методів побудови хеш-перетворень без колізій для криптографічно строгої ідентифікації віддалених користувачів показав, що основним недоліком існуючих схем таких перетворень є обмежена кількість можливих вхідних векторів.

Ціль досліджень полягає полягає в підвищенні ефективності криптографічного строгої ідентифікації на основі хеш-перетворень з програмованими колізіями за рахунок збільшення кількості сеансових паролів доступу.

Висновки до розділу 1

Результати здійсненого аналітичного огляду існуючих засобів криптографічно строгої ідентифікації, який складає перший розділ магістерської дисертації можуть сформульовані у вигляді наступних висновків:

1. Виконано аналіз сучасного стану використання механізмів ідентифікації учасників віддаленої інформаційної взаємодії, який показав, що розширення сфери використання віддалених форм інформаційної взаємодії та комерціалізація віддаленого надання інформаційних послуг диктують необхідність радикального підвищення рівня захищеності процесів взаємної ідентифікації учасників віддаленої взаємодії, що може бути досягнуто тільки в рамках використання криптографічно строгих різновидів механізмів ідентифікації.

2. Огляд, з позицій сучасних вимог, існуючих методів та засобів криптографічно строгої ідентифікації учасників віддаленої інформаційної взаємодії показав, що ті їх різновиди, які базуються на незворотних перетвореннях модулярної арифметики мають недостатню швидкодію. Більш швидкісні методи криптографічно строгої ідентифікації, що базуються на використанні незворотних перетворень булевої алгебри мають обмежену кількість сеансових паролів.

3. В якості найбільш перспективного шляху підвищення ефективності криптографічно строгої ідентифікації на основі хеш-перетворень з програмованими колізіями є пошук можливостей радикального збільшення числа сеансових паролів.

РОЗДІЛ 2

РОЗРОБКА МЕТОДУ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ КРИПТОГРАФІЧНО СТРОГОЇ ІДЕНТИФІКАЦІЇ НА ОСНОВІ ХЕШ-ПЕРЕТВОРЕННЯ З ПРОГРАМОВАНИМИ КОЛІЗІЯМИ

Швидкий розвиток глобальних мереж та віддаленої інформаційної взаємодії призвів до значного збільшення кількості систем колективного доступу та їх користувачів. У зв'язку з цим, контроль доступу до даних став критично важливою проблемою в нових областях віддаленої взаємодії. Ідентифікація віддалених користувачів стала основним засобом контролю доступу до інформації, і вимоги до її рівня захисту та швидкості виконання процедури зросли в останні роки.

Концепція криптографічно строгої ідентифікації, інша її назва «нульових знань», в теоретичній площині найбільш повно відповідає наведеним раніше вимогам відносно механізму ідентифікації віддалених абонентів. Наразі, необхідним для можливості практичного застосування рівень надійності та захищеності ідентифікації може бути забезпечений лише методами, що спираються на концепцію «нульових знань». Концепція «нульових знань» передбачає використання незворотних математичних перетворень. Це означає, що існує алгоритм перетворення в прямому напрямку, але принципово неможливим є аналітичне віднаходження алгоритму зворотного перетворення. Одним з можливих варіантів криптографічно строгої ідентифікації з застосуванням принципів «нульових знань» є використання спеціальних незворотних хеш-перетворень з програмованими колізіями.

У другому розділі дослідження розробляється структура хеш-функції з програмованими колізіями та метод визначення значень нелінійних булевих перетворювачів. Це дозволяє утворювати сеансові паролі, що гарантують виконання вимог щодо криптографічно строгої ідентифікації. Основною метою

дослідження є розробка методу побудови незворотних булевих перетворень, які мають властивості неоднозначного зворотного перетворення. Цей метод дозволить покращити рівень захисту та швидкість виконання процедури ідентифікації віддалених користувачів.

2.1 Визначення структури хеш-перетворювача

Для досягнення зазначеної мети є необхідним формування функціонального перетворення $F(X)$, що визначене на множині значень від 0 до 2^n для n-бітного бінарного вектору $X=\{x_1, x_2, \ldots, x_n\}$, де $\forall i \in \{1,\ldots,n\}$: $x_i \in \{0,1\}$. Результатом перетворення вхідного вектору X є n-бітовий вихідний вектор $Y=F(X)$, $Y=\{y_1,y_2,\ldots,y_n\}$, де $\forall i \in \{1,\ldots,n\}$: $y_i \in \{0,1\}$. Кожна i-та бінарна складова y_i вихідного вектору Y може бути оцінена як значення булевої функціонального перетворення $f_i(X)$, визначеної на множині значень X. Таким чином, можна стверджувати, що перетворення $F(X)$ є агрегацією n булевих функціональних перетворень $f_1(X)$, $f_2(X)$, $\ldots$,$f_n(X)$ та можу бути записано у вигляді $F(X)=\{f_1(X), f_2(X), \ldots, f_n(X)\}$.

Для відповідності властивості незворотності, утворене функціональне перетворення $F(X)$, важливо, щоб кожне з булевих перетворень $f_1(X)$, $f_2(X)$, …, $f_n(X)$, що є складовими перетворення $F(X)$, було нелінійним за своєю структурою та було залежним від кожної зі змінних-складових вектору $X=\{x_1, x_2, \ldots, x_n\}$.

Для досягнення відповідності властивості неоднозначності функціональним перетворенням $F(X)$, є необхідним існування множини Ω допустимих вхідних векторів, після застосування яких функціональне перетворення $F(X)$ приймає однакові (рівні) значення. У формальному записі ця умова виглядає наступним чином: $\forall Q, G \in \Omega, Q \neq G$, де $F(Q) = F(G) = U$.

Широко відомо, що булеві функціональні перетворення можуть бути представлені у трьох різних формах: у вигляді таблиць істинності, в алгебраїчних нормальних формах та в процедурному вигляді, а саме у вигляді алгоритму

обчислення вихідного вектору Y за значенням вхідного вектору X. Зокрема, для захисту інформації найчастіше використовується саме третя форма, оскільки булеві функції, на яких здійснюється захист, можуть містити сотні змінних. Тому, з метою формування булевих перетворень, рекомендується використовувати процедурний підхід.

Можливі альтернативні схеми внутрішньої структури хеш-перетворення, що відрізняються між собою способом виконання функції перемішування. Основна мета цього перемішування полягає у потребі встановлення залежності кожного розряду вихідного коду, одержаного після застосування хеш-перетворення, від кожного розпяду вхідного коду, який є сесійний паролем користувача. Одним з таких хеш-перетворень є структура, зображена на рисунку 2.1.

Наведена на рисунку 2.1 структура потребує виконання обчислень функціонального перетворення $F(X)$ з попередньою розбивкою даних на m-розрядні фрагменти. Числа n і m є степенями 2, число фрагментів (або ж секцій) k визначається за співвідношенням $k = n/m$. Переважна більшість структурних утворювань, що використовуються для реалізації булевих перетворень в межах алгоритмів, направлених на забезпечення захисту доступу до інформації, відповідно схема перетворювача, наведена на рис. 2.1, утворена з послідовних шарів нелінійного перетворення вхідних фрагментів та шарів їх подальшого перемішування. Нелінійне перетворення в межах цих функціональних перетворень здійснюється з застосуванням табличних булевих функціональних перетворень $\varphi_1, \varphi_2, \ldots, \varphi_k$, де k – кількість фрагментів. Кожне j-те ($j \in \{1,\ldots,k\}$), фрагментарне перетворення $\varphi_j(V)$ є визначеним на множині 2^m всіх допустимих значень вектору $V=\{v_1,v_2,\ldots,v_m\}$, де $\forall j \in \{1,\ldots,m\}$: $v_j \in \{0,1\}$, а – кількість бітів вектору V. Для перемішування поміжних результатів перетворення окремих фрагментів реалізується операція виконання сумування результатів проміжних нелінійних перетворень сусідніх фрагментів i та $(i+1) \bmod k$ по модулю 2. Для забезпечення залежності кожного біта, що є складовою вихідного вектору Y, від кожного з бітів

вхідного вектору *X*, число повторюваних шарів перемішування в структурі перетворювача зобов'язане бути рівним значенню, яке є не меншим значення числа фрагментів *k*.

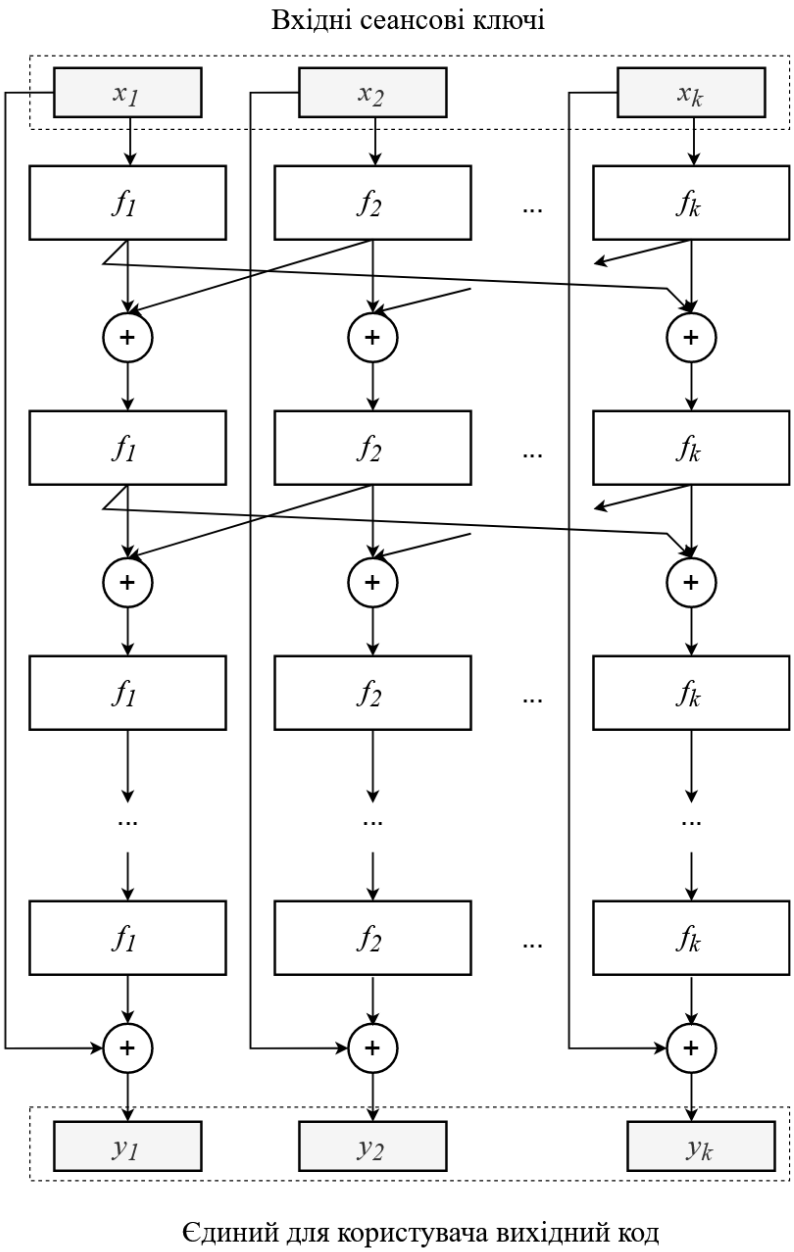

Рис. 2.1. Внутрішня структура хеш-перетворювача з програмованими колізіями

Альтернативна схема перемішування, що могла б бути застовонава на шарах хеш-перетворювача, наведена на рис. 2.2.

В схемі, наведеній на рис. 2.1, перемішування регультатів фрагментарних перетворень виконується в межах одного шару хеш-перетворювача. На противагу, схема, зображена на рис. 2.2. вимагає перемішування «по діагоналі», тобто відбувається перемішування між суміжними рівнями.

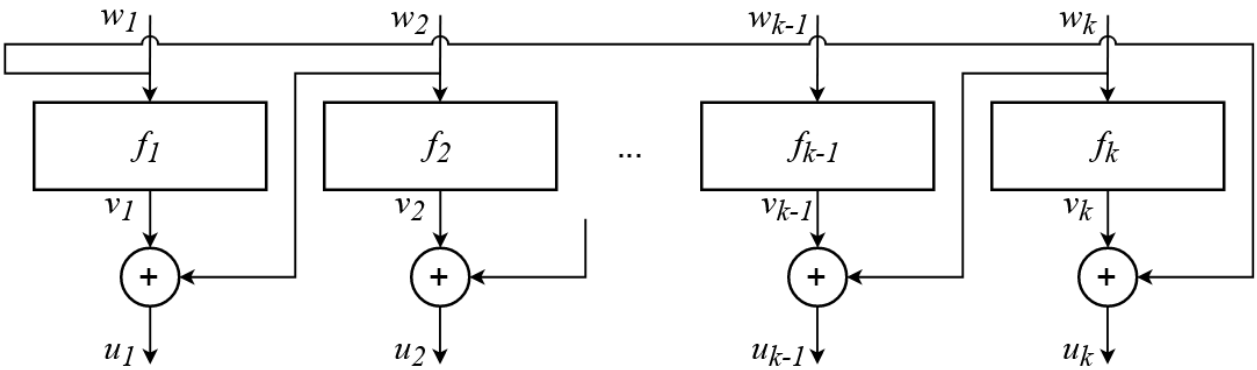

Рис. 2.2. Структура міжрівневого перемішування

Позначимо Z_{qj} як m-бітний вектор, який є j-тим фрагментом вхідного n–бітного коду $W_q=\{Z_{q1}, Z_{q2}, \ldots, Z_{qk}\}$, що є результатом виконання q-того циклу, де $W_0 = X$ та $W_k = Y$. Через запис $W_q(Q)$ позначимо результат проміжних обрахунків на виході q-го шару перетворювача для вхідного вектору $X=Q$. Звідси, алгоритм обчислення значень вихідного вектору хеш-перетворювача, схема якого є представлена на рис. 2.1, може бути записано у наступному аналітичному вигляді:

$$\begin{aligned}
&\forall j \in \{1,\ldots,k\} : Z_{0,j} = \{x_{(j-1)\cdot m+1}, x_{(j-1)\cdot m+2}, \ldots, x_{j\cdot m}\};\\
&\forall q \in \{1,\ldots,k\} :\\
&\forall g \in \{1,\ldots,g-1\} : Z_{qg} = \varphi_j(Z_{q-1,g}) \oplus Z_{q-1,g+1}\\
&Z_{qk} = \varphi_k(Z_{q-1,k}) \oplus Z_{q-1,1}
\end{aligned} \quad . \qquad (2.1)$$

Розглянемо деякий один внутрішній шар хеш-перетворювача, структуру якого наведено на рисунку 2.3. Його дослідження дозволяє встановити лінійні залежності між складовими вихідного коду цієї структури.

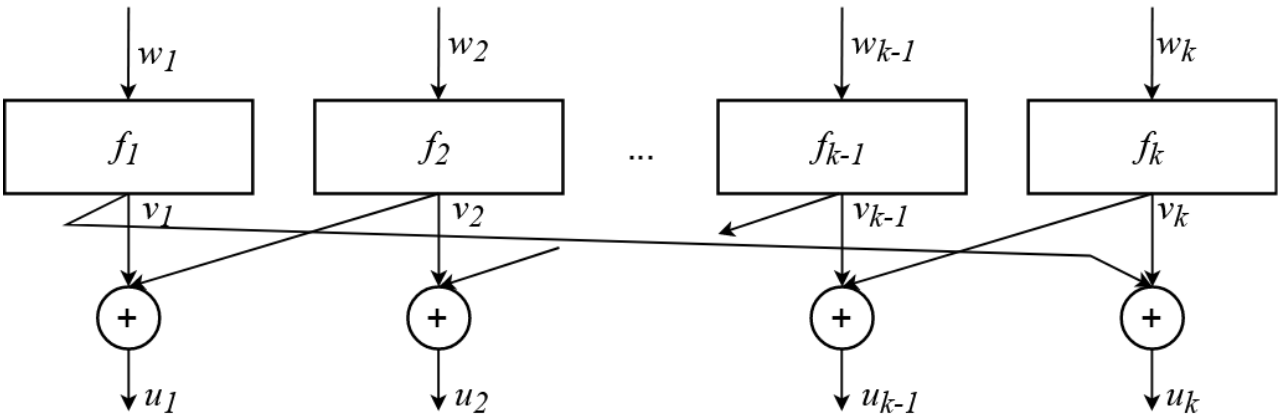

Рис. 2.3. Структура одного внутрішнього шару перетворювача

Залежності між вихідними кодами нелінійних перетворень і кодами на виході шару змішування визначаються архітектурою однорівневого змішування і можуть бути формально представлені наступною системою лінійних рівнянь:

$$\begin{cases} v_1 \oplus v_2 = u_1 \\ v_2 \oplus v_3 = u_2 \\ v_3 \oplus v_4 = u_3 \\ \ldots \\ v_k \oplus v_1 = u_k \end{cases}. \qquad (2.2)$$

Ще однією альтернативною схемою внутрішньої структури хеш-перетворювача є трапецевидна структура, зображена на рис. 2.4.

Властивість незворотності булевих функціональних перетворень базується на тому, що аналітично неможливо знайти корені систем нелінійних булевих рівнянь. Тому перебір єдиний метод розв’язання цих задач. Ця властивість є важливою в більшості сучасних алгоритмів захисту інформації, і проблема забезпечення захисту цих алгоритмів від зламу є еквівалентною рішенню тотожної системи нелінійних булевих рівнянь. За останні роки було запропоновано ряд підходів для скорочення обсягів перебору під час розв’язання цієї математичної проблеми, таких як лінійний та диференційний криптоаналіз. Таким чином, залежність властивостей

булевих функцій та можливості скорочення перебору, що складають перетворення, є глибоко дослідженою.

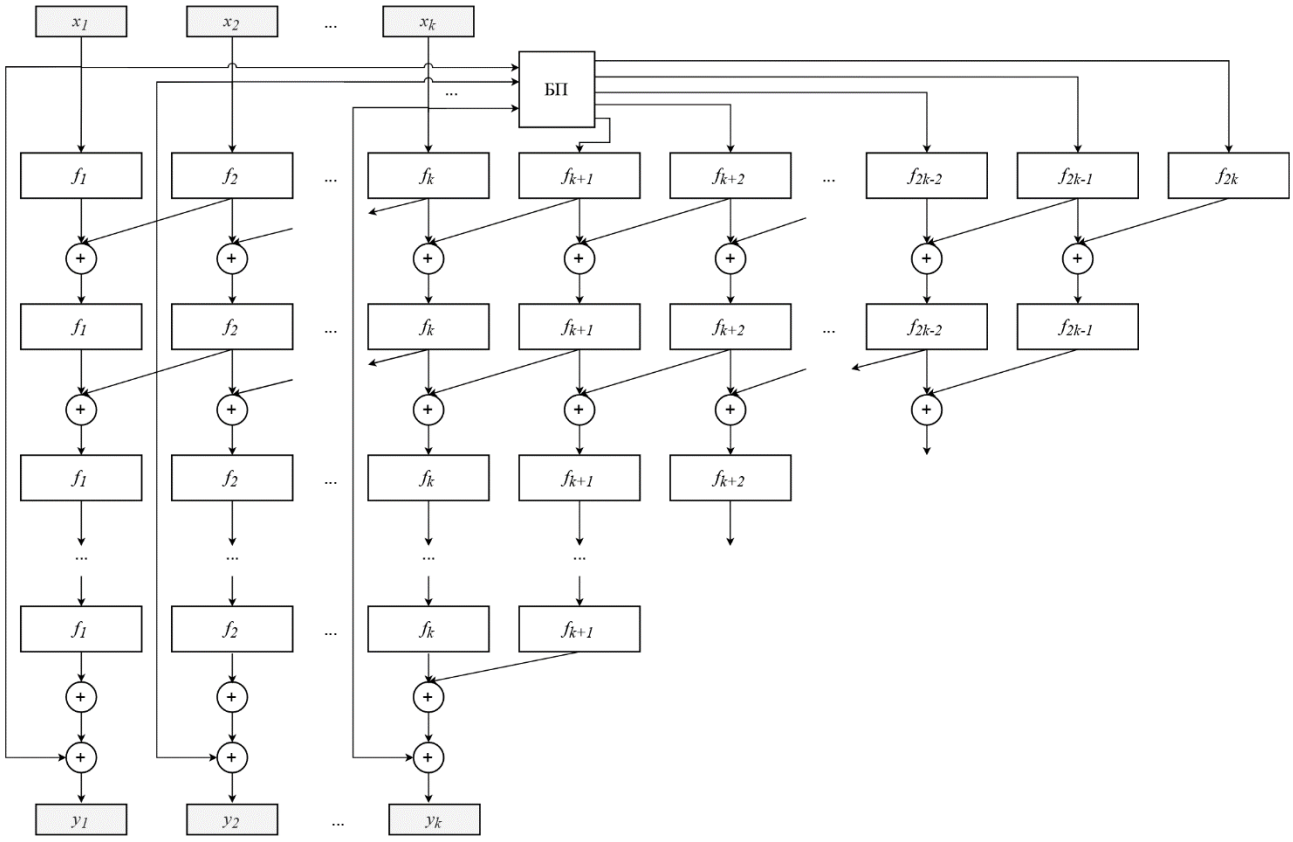

Рис. 2.4. Трапецевидна структура хеш-перетворювача.

Основними властивостями усіх булевих перетворень є притаманна їм нелінійність та диференціальні характеристики, які визначають обсяг перебору значень направлених на вирішення завдання знаходження коренів деякої системи булевих рівнянь.

Булева функціональне перетворення $f(x_1,\ldots,x_n)$ від n булевих змінних є визначеною на множині 2^n допустимих наборів значень множини Z та досягає вихідних значень на множині B={0, 1}. Як наслідок, хеммінгова вага $W(f(x_1,\ldots,x_n))$ цього функціонального перетворення обчислюється як сумарне число одиничних значень, яких досягає вона досягає на всій множині своїх значень:

$$W(f(x_1,..., x_n)) = \sum_{x_1,..., x_n \in Z} f(x_1,..., x_n). \quad (2.3)$$

Булева функціональне перетворення $f(x_1,\ldots,x_n)$ є балансним, адже задовольняє критерій максимуму повної ентропії, тобто з однаковою ймовірністю досягає значень нуля та одиниці:

$$W(f(x_1,..., x_n)) = 2^{n-1}. \quad (2.4)$$

Позначимо літерою G множину n булевих функціональних перетворень, $G=\{f_1(x_1,\ldots,x_n), \ f_2(x_1,\ldots,x_n), \ \ldots \ , \ f_k(x_1,\ldots,x_n)\}$. Ця множина є досягає властивості ортогональності, адже сума будь-якої підмножини її функцій по модулю 2 є балансною функцією:

$$\forall \vartheta \subseteq G : W(f_p(x_1,...,x_n) \oplus f_l(x_1,...,x_n)) = 2^{n-1}. \quad (2.5)$$

Властивість нелінійності булевих функцій є важливою для застосування булевих перетворень в задачах захисту інформації, оскільки вона визначає стійкість до зламу алгоритмів захисту інформації методами лінійного криптоаналізу. Такі атаки використовують лінійні залежності між вхідними та вихідними бітами булевих функцій для злому шифрування. Нелінійність булевих функцій забезпечує відсутність лінійних залежностей та ускладнює проведення таких атак. Тому при проектуванні криптографічних алгоритмів велика увага приділяється вибору нелінійних булевих функцій з високими значеннями нелінійності.

Мінімальна хеммінгова відстань перетворення до її лінійних функцій дозволяє обчислити критерій нелінійності $N(f(x_1,\ldots,x_n))$ булевого функціонального перетворення $f(x_1, \ldots, x_n)$:

$$N(f(x_1,..., x_n)) = \min_{a_p \in \{0,1\}, p=0,...,n} W(f(x_1,..., x_n) \oplus (a_0 \bigoplus_{j=1}^{n} a_j \cdot x_j)). \quad (2.6)$$

У відношенні до систем булевих функцій, поняття нелінійності позначає значення критерію, який відображає мінімальну лінійність лінійних комбінацій булевих функцій, які при об'єднанні утворюють вищенаведену систему *G*:

$$N(G) = \min_{b_p \in \{0,1\},\, p=1,\ldots,n} N(\bigoplus_{p=1}^{n} b_p \cdot f_p(x_1,\ldots, x_n)),\ f_p \in G. \qquad (2.7)$$

У системи булевих функцій *G*, критерій нелінійності визначається за допомогою побудови лінійного профілю, який складається з таблиці, де стовпці відображають значення лінійних комбінацій булевих функцій, які утворюють систему, а рядки відповідають лінійним функціям. На перетині кожного стовпця і кожного рядка зазначається хеммінгова відстань між *i*-тою лінійною комбінацією функцій, що складають систему, та відповідною їй *j*-тою лінійною функцією. Критерій нелінійності системи булевих функцціональних перетворень є мінімальним елементом в розробленому лінійному профілю системи *G*. У контексті застосування булевих функцій у задачах захисту інформації, нелінійність є важливою характеристикою, яка визначає стійкість до атак методом лінійного криптоаналізу.

Для ефективного застосування булевих перетворень у механізмах захисту інформації важливо враховувати їх диференціальні характеристики. Ці характеристики визначають захищеність алгоритмів від зламів методами диференціального криптоаналізу та виключення змінних. Диференціальна ентропія є найважливішою диференціальною характеристикою булевих функцій. Вона вказує, як саме змінюється булева функція при зміні вхідних змінних, на яких ця функція базується. Чим вища диференціальна ентропія, тим складніше провести атаку диференціальним криптоаналізом на булеву функцію, тому використання функцій з високим значенням диференціальної ентропії є бажаним при проектуванні захисту інформації.

Особливе значення мають булеві функції, які задовольняють критерію максимальної диференціальної ентропії по одній змінній. Це означає, що вони

змінюють своє значення з ймовірністю 0,5 при зміні будь-якої з k вхідних змінних. Для задач хеш-адресації та захисту інформації ключове значення мають саме такі функції. Такі функції називають PC(k)-функціями (Propagation Criterion on k variables). Значення критерію диференціальної ентропії булевого функціонального перетворення $f(x_1,\ldots,x_n)$ залежить від ймовірності P_k зміни її значення при модифікації k вхідних змінних з n, на яких вона визначена. Це означає, що булеве функціональне перетворення $f(x_1,\ldots,x_n)$ належить до класу PC(k)-функцій та відповідає критерію максимуму диференціальної ентропії по k змінним у разі, якщо виконується наступна умова:

$$W(f(X)\oplus f(X\oplus A))=2^{n-1}, A=\{a_1,...,a_n\}, a_p\in\{0,1\}, \sum_{p=1}^{n} a_p \le k. \qquad (2.8)$$

Булеві функції, які максимізують значення диференціальної ентропії по одній змінній, мають особливе значення. Такі функції належать до класу функцій, які відповідають критерію максимальної диференціальної ентропії, і мають характеристику лавинного ефекту. Завдяки цим властивостям вони грають ключову роль у багатьох задачах, пов'язаних з захистом інформації та хеш-адресацією.

Критерій максимуму умовної ентропії відносно однієї змінної є застосовним до булевого функціонального перетворення $f(x_1,\ldots,x_n)$ у разі, якщо заміна будь-якої з її вхідних змінних продукує зміну результуючих значень виконання функції з ймовірністю 0,5:

$$\forall p\in\{1,..,n\}: W(f(x_1,...,x_p,...,x_n)\oplus f(x_1,...,\bar{x}_p,...,x_n))=2^{n-1}. \qquad (2.9)$$

Бент-функцією називають PC(n)-функцію, що змінює своє вихідне значення зі сталою ймовірністю рівною 0,5 при інверсії абсолютно всіх її вхідних змінних. Бент-функції притаманне максимально можливе значення нелінійності, але при цьому вона не є балансною. Таким чином, термін бент-функції може бути застосовано лише для значень n, які є парними. У такому разі показник її нелінійності становить:

$$N(f_{bt}(x_1,..., x_n)) = 2^{n-1} - 2^{n/2-1}. \qquad (2.10)$$

У разі якщо $n > 3$, то значення показника нелінійності $N(f)$ для балансних булевих перетворень від n змінних $f(x_1,\ldots,x_n)$ буде обмежено зверху границями:

$$N(f) \le 2^{n-1} - 2^{n/2-1} - 2 \text{ (для парних } n), \qquad (2.11)$$

$$N(f) \le \left\lfloor 2^{n-1} - 2^{n/2-1} \right\rfloor \text{ (для непарних } n).$$

Для виконання ідентифікації відповідно до концепції «нульових знань» з використанням необоротних булевих функціональних перетворень, потрібно передавати інформацію від абонента A до системи. Для цього можна використовувати вихідний вектор U_A разом з описом функціонального перетворення $F(X)$ в процедурній формі. Ці дані не дозволяють отримати значення вхідного вектору X або самого функціонального перетворення, тому їх можна вважати «нульовими знаннями».

Нехай секретний разовий «пароль» користувача A – це вхідний вектор X, для якого формується вихідний вектор U_A при виконанні на ним функціонального перетворення $F(X)$. Використання незворотних булевих функціональних перетворень гарантує надійність системи ідентифікації на основі концепції «нульових знань». Безаперечно, що завдяки властивості незворотності перетворення, за присутнім в системі вихідним вектором U_A та інкапсульованим функціональним перетворенням $F(X)$ відновлення значення вхідного вектору X є практично неможливим.

На жаль, подібна схема є коректною лише для єдиного циклу ідентифікації. Для реалізації багаторазової ідентифікації з використанням змінюваного вхідного ключа необхідно, щоб необоротне булеве функціональне перетворення $F(X)$ володіло властивістю неоднозначності. Це означає потребу існування деякої множини $\Omega_A=\{X_1,X_2,\ldots,X_s\}$, де s – кількість можливих вхідних векторів, для кожного з яких обов'язково виконується умова $\forall X \in \Omega$: $F(X)=U_A$.

Таким чином, отримання булевих функцій, які володіють властивістю неоднозначності, для застосування в схемі ідентифікації на основі концепції «нульових знань», потребує розробки спеціальних методів для їх представлення в процедурному вигляді.

2.2 Розробка механізму формування хеш-перетворення

Структура, представлена на рис. 2.1, визначає задачу утворення неоднозначного і незворотного функціонального булевого перетворення $F(X)$ в межах бажаної структури при організації фрагментарних булевих функціональних перетворень $\varphi_1,\varphi_2, \ldots,\varphi_k$, таким чином, щоб вони гарантували забезпечення єдиного результуючого значення вихідного вектору для заздалегідь обраної множини допустимих вхідних бінарних векторів $\Omega = \{X_1, X_2 ,\ldots, X_s\}$. Вихідне значення перетворювача обчислюється відповідно до формули (2.10). Кожне j-те фрагментарне функціональне перетворення $\varphi_j(V)$, де $j \in \{1,\ldots,k\}$, складається з k булевих функціональних перетворень $\varphi_j = \{\phi_{j1}(V), \phi_{j2}(V), \ldots,\phi_{jk}(V)\}$ та може набувати значень з інтервалу від 0 до 2^m. Для реалізації цієї функціональності пропонується метод, який полягає в послідовному обрахуванні значень фрагментарних функціональних перетворень протягом перетворення вхідних векторів з множини Ω. Наведений метод використовує особливу множину Θ, до якої входить список наборів аргументів кожного з фрагментарних перетворень $\varphi_1,\varphi_2,\ldots\varphi_k$. Для цих аргументів значення перетворення визначаються випадковим чином протягом обчислення значення поточного вхідного вектору X_c.

Запропонований метод передбачає виконання наступної послідовності дій:

1. Значення всіх k фрагментарних перетворень $\varphi_1,\varphi_2,\ldots\varphi_k$ на всіх 2^m можливих бінарних наборах значень їх аргументів вважаються рівними -1, тобто вважаються невизначеними. При цьому множина $\Omega=\varnothing$.

2. Вибирається довільний вектор X_1 розрядністю n, який вноситься до множини Ω.

3. У відповідності до формули (2.10) виконується обчислення значення $U = F(X_1)$. На кожному q-тому з k шарів перетворювача, для кожного j-го фрагменту (j=1,…k), виконується наступна перевірка: якщо для попереднього фрагменту значення результату фрагментарного булевого функціонального перетворення $z_{q-1,j}(X_1)$ не є визначеним, є рівним -1, то воно встановлюється рівним випадковому значенню з інтервалу від 0 до 2^m-1.

4. Довільним чином обирається вектор X_c, після чого встановлюється $W_0(X_c)=X_c$. Множина Θ визначається порожньою, тобто Θ=∅. Випадковим чином генерується номер N_c рівня, де N_c={1,…,k}, на якому виконується злиття проміжних фрагментарних значень вектору X_c разом з одним з векторів множини Ω, у якого порядковий номер є меншим c. Далі необхідно встановити номер поточного шару q рівним одиниці: q = 1. Якщо N_c = 1, то виконати перехід до п.7.

5. У разі, якщо q < N_c-1, то обчислення проміжних вихідних значень перетворень для кожного j-го фрагменту виконується наступним чином: в разі $\varphi_j(z_{q-1,j})\neq$-1, то обчислення проводиться відповідно до (1). Якщо значення фрагментарного функціонального перетворення φ_j не є визначеним на наборі $z_{q-1,j}$, тобто $\varphi_j(z_{q-1,j})$= -1, то на наборі $z_{q-1,j}$ випадковим чином формується значення φ_j, що належить інтервалу від 0 до 2^m-1, після чого обчислення здійснюється згідно (1). При цьому, номер j довизначеного перетворення φ_j, разом з набором $z_{q-1,j}$, заносяться до множини Θ: $\Theta=\Theta\cup<j,z_{q-1,j}>$. Після того, як таким чином було обчислено значення всіх k фрагментів проміжних результатів q-шару для коду X_c здійснюється перехід до наступного шару, шляхом присвоєння q=q+1 і повернення на повторне виконання пп. 5.

6. У разі, якщо q = N_c - 1, і $\varphi_j(z_{q-1,j})$ не є визначеним, то значення булевого перетворення φ_j на наборі $z_{q-1,j}$, обирається таким чином, щоб на наборі $z_{q,j}$ = $\varphi_j(z_{q-}$

$_{1,j}$)$\oplus z_{q-1,j+1}$ значення фрагментарного перетворення φ_j було невизначеним, тобто виконувалася умова $\varphi_j(z_{q,j}) = -1$. Далі, номер j перетворення φ_j, для якого було довизначено значення, заноситься до множини Θ разом з набором $z_{q-1,j}$: $\Theta=\Theta\cup<j,z_{q-1,j}>$. У разі, якщо $\varphi_j(z_{q-1,j}) \neq -1$, тобто є визначеним, обчислення $z_{q,j}$ здійснюється відповідно до формули (2.10). Врешті, після обрахування таким чином значення всіх k фрагментів проміжних вихідних результатів q-шару для коду X_t, відбувається перехід до наступного шару перетворювача шляхом встановлення $q=q+1$.

7. Для кожного наступного j-го фрагменту виконується перевірка виконання умови визначеності значення, а саме $\varphi_j(z_{q-1,j}) = -1$. У разі виконання цієї умови, випадковим чином обирається $d\in\{1,\ldots,c\text{-}1\}$ та здійснюється подальший перехід до пп.8. Далі, для кожного j-го фрагменту виконується перевірка виконання умови визначеності значення, а саме $\varphi_j(z_{q-1,j})\neq -1$, з урахуванням існування такого $d\in\{1,\ldots,c\text{-}1\}$, щоб $\varphi_j(z_{q-1,j})\oplus z_{q-1,j+1} = z_{qj}(X_d)$. У разі якщо ця умова виконується, то відбувається перехід до пп.8. Якщо ж зазначені вище вимоги не виконуються, то вважається, що виконаний підбір значень фрагментарних булевих функціональних перетворень викликає виникнення конфліктів для вхідного коду X_c. Всі визначені значення фрагментарних булевих перетворень при перетворенні X_c повертаються до попереднього стану і знову вважаються невизначеними: $\forall<j,z>\in\Theta$: $\varphi_j(z)= -1$. Виконується повернення до пп.4.

8. Для кожного наступного j-го фрагменту перетворення виконується: у разі якщо $\varphi_j(z_{q-1,j})=-1$, тоді відповідне значення фрагментарного булевого перетворення на означеному наборі визначається наступним чином: $\varphi_j(z_{q-1,j}(X_c)) = z_{q-1,j+1}(X_c) \oplus z_{q-1,j}(X_c)$, якщо $j<k$, та $\varphi_j(z_{q-1,j}(X_c)) = z_{q-1,1,}(X_c) \oplus z_{q-1,j+1}(X_d)$, якщо $j = k$.

9. У разі, якщо $c < k$, тоді виконується інкремент номеру поточного вектору X з множини Ω: $c = c +1$. Відбувається повернення до пп. 4.

10. На наборах, які не були визначеними раніше, для всіх k фрагментарних функціональних булевих перетворень $\varphi_1,\varphi_2,\ldots,\varphi_k$, встановити як значення випадкові

цілі числа з інтервалу від 0 до 2^m–1, тобто $\forall j \in \{1,\ldots,k\}$, $\forall z \in \{0,\ldots,2^m\text{-}1\}$: $\varphi_j(z)$= -1, встановити $\varphi_j(z)$ = rand(0, 2^m-1).

В результаті виконання вищенаведеної послідовності дій отримуються k таблиць фрагментарних булевих перетворень $\varphi_1,\varphi_2,\ldots\varphi_k$. Разом із структурою, наведеною на рис. 2.1, вони повністю визначають процедуру обчислення необоротного функціонального перетворення *F(X).*

Імовірність того, що при застосуванні перетворення до випадкового вхідного коду X_e буде отримано значення, яке збігається з U, становить 2^{-n}. Оскільки на практиці значення n зазвичай складає сотні, то ймовірність того, що будь-яка зовнішня особа зможе підібрати ідентифікуючий код, використовуючи запропоноване перетворення, практично дорівнює нулю.

Застосування описаного методу для отримання нелінійних і неоднозначних булевих перетворень можна проілюструвати наступним прикладом. Нехай n = 16, тобто розрядність ключів на вході і виході перетворювача $F(X)$ є рівною 16 біт. Далі, встановимо що процедура перетворення складається з 4-х шарів, тобто оброблювані перетворенням $F(X)$ вхідні коди поділяються на 4 рівні фрагменти (k = 4) розрядністю в 4 біт кожен ($m = n/k = 4$). Таким чином, фрагментарні табличні булеві перетворення $\varphi_1,\varphi_2,\varphi_3,\varphi_4$ є визначеними на множині 4-бітних змінних, тобто приймає 16 значень від 0 до 2^4. Нехай множина ключів Ω містить три 16-бітних ключів, записані з використанням шістнадцядкової системи числення у вигляді: X_1=5E69h, X_2 = B2C6p, X_3 = 7F48h.

Таблиця 2.1 ілюструє динамічне заповнення таблиць істинності для фрагментарних булевих функціональних перетворень, а саме $\varphi_1,\varphi_2,\varphi_3,\varphi_4$, під час обробки ключів, що належать до множини Ω. Порожні клітинки таблиці означають, що значення фрагментарних булевих перетворень не є заповненими на відповідних наборах вхідних ключів.

Таблиця 2.1

Заповнення функціональних перетворень $\varphi_1,\varphi_2,\varphi_3,\varphi_4$ значеннями

X	Таблиці значень фрагментарних булевих перетворень							
	Після обробки X_1				В остаточному вигляді			
	φ_1	φ_2	φ_3	φ_4	φ_1	φ_2	φ_3	φ_4
0			6		11	4	6	7
1					3	14	9	2
2			3		0	0	6	7
3	4				4	1	4	11
4	8				8	2	6	5
5	14				10	7	3	8
6				0	3	10	7	2
7			10	15	13	15	12	5
8					12	6	15	6
9	3		6		3	9	8	8
10		15			5	15	12	10
11		13		12	15	13	10	12
12			12		8	2	12	11
13		6		1	5	6	10	0
14	8				8	2	12	9
15					1	0	15	3

В результаті, за допомогою запропонованого методу було отримано таблиці істинності значень фрагментарних функціональних булевих перетворень $\varphi_1,\varphi_2,\varphi_3,\varphi_4$. Після обробки всіх трьох кодів X_1, X_2, X_3, значення функцій $\varphi_1,\varphi_2,\varphi_3,\varphi_4$ були визначені на 64% наборів значень їх вхідних змінних. Залишаються 36%

наборів, де значення функцій визначаються випадковим чином, згідно з пунктом 10 методу, описаного вище.

Згідно з викладеним методом формування фрагментарних булевих функціональних перетворень, процес виконується з використанням проб. Якщо при обробці чергового вхідного вектору X, в процесі якого використовується випадкове визначення значень перетворень, на певних наборах не вдається уникнути конфлікту з раніше визначеними значеннями, то здійснюється повернення на повторну обробку вектору X. Час формування нелінійного функціонального перетворення залежить від кількості проб заповнення таблиць фрагментарних функціональних перетворень.

Отже, важливим елементом запропонованого методу є здатність отримання неоднозначного і незворотного булевого функціонального перетворення в процедурній формі є оцінка залежності числа проб від її параметрів. Це може включати параметри, такі як розмір таблиці функціональних перетворень, кількість можливих варіантів значень перетворень та кількість можливих конфліктів між раніше визначеними та поточними значеннями перетворень.

У процесі формування фрагментарних булевих функціональних перетворень методом, що був описаний вище, важливо дотримуватися правил обробки вхідних векторів, щоб уникнути можливість конфліктів між раніше визначеними та поточними значеннями перетворень. Це може забезпечити ефективне формування нелінійного функціонального перетворення та зменшення кількості проб заповнення таблиць фрагментарних функціональних перетворень.

Нехай відбувається обробка деякого g-того вхідного вектору X_g з множини всіх можливих вхідних векторів Ω. Так як при перетворенні вектору значення фрагментарних перетворень, в середньому, встановлюється на половині шарів, отже, середній показник числі значень одного функціонального перетворення, які значення для яких визначаються при виконанні перетворення одного вектору X становить близько $k/2$. Звідси, до завершення обробки g-го вхідного вектору X_g в

кожному з k булевих фрагментарних перетворень буде визначено, в середньому, $k \cdot g/2$ значень. З представленого опису методу побудови перетворювача (пп.7) випливає, що конфліктна ситуація здатна виникнути у разі, якщо при обробці j-го фрагменту $z_{q,j}$ деякого проміжного значення деякого перетворення на q-тому шарі, для відповідного перетворення φ_j значення були раніше визначені водночас на наборах $z_{q,j}$, та $\varphi_j(z_{q,j}) \oplus z_{q-1,j+1}$ для $j<k$ або наборі $\varphi_j(z_{q,j}) \oplus z_{q-1,1}$ для $j=k$. Це означає, що у випадку коли значення для двох фіксованих наборів деякого функціонального перетворення φ_j були раніше визначені, це призводить до виникнення конфліктної ситуації. Враховуючи факт того, що загальне число наборів, на яких перетворення φ_j набуває значень рівне 2^m, то, в середньому, на $k \cdot g/2$ з них, деяке значення перетворення φ_j було визначено раніше, звідси ймовірність того, що на деяких двох фіксованих наборах бажане значення функціонального перетворення φ_j уже було визначено раніше, становить $(k \cdot g)^2/2^{2 \cdot (m+1)}$. Враховуючи, що для виникнення потреби у повторній обробці вхідного вектору достатня наявність конфліктної ситуації в одному з його фрагментів, то ймовірність P_g успіху спроби випадкового заповнення всіх наявних k фрагментарних перетворень $\varphi_1, \varphi_2, \ldots, \varphi_k$ при перетворенні деякого вхідного вектору становить:

$$P_g = (1 - \frac{h^2 \cdot g^2}{2^{2 \cdot (k+1)}})^h . \qquad (2.11)$$

Таким чином, середнє число спроб заповнення таблиць фрагментарних булевих перетворень при обробці g-го вхідного вектору становить $t_g = 1/P_g$. Звідси, середнє число спроб T_k, направлених на заповнення таблиць значень фрагментарних булевих перетворень при обробці множини всіх s можливих вхідних кодів з множини Ω, обраховується як сума середнього числа спроб, необхідних для перетворення кожного з s можливих вхідних кодів:

$$T_s = \sum_{j=1}^{s} \frac{1}{(1 - \frac{k^2 \cdot j^2}{2^{2 \cdot (m+1)}})^k} . \qquad (2.12)$$

Загальний об’єм пам’яті таблиць значень фрагментарних нелінійних булевих перетворень V_T:

$$V_T = k \cdot 2^{\frac{n}{k}}. \qquad (2.13)$$

Для перевірки теоретичної залежності часу формування функціонального перетворення від параметрів процедурної форми та кількості можливих вхідних векторів були проведені статистичні дослідження за допомогою програмної моделі. Результати експериментів показали, що теоретична оцінка залежностей відповідає емпіричним даним. Наприклад, при використанні довжини коду доступу n=256, параметрах процедурної форми k=16 та m=6, побудова функціонального перетворення потребує близько 40000 спроб для утворення 4000 кодів доступу, що займає не більше години роботи персонального комп'ютера. Множина вхідних векторів містить близько 4000 елементів, для яких результат перетворення $F(X)$ однаковий. Заповнено близько 47% таблиць фрагментарних перетворень, а решта заповнюється випадковим чином згідно з пунктом 10 методу.

2.3 Розробка способу прискорення побудови хеш-перетворення

Дослідження показали, що розроблений метод побудови хеш-перетворень з програмованими колізіями, який був розроблений в попередньому підрозділі, вимагає залучення значних часових ресурсів, причому їх об’єм активно зростатиме зі збільшенням очікуваної кількості сеансових паролів. Отже, необхідно розробити більш швидкодіючі варіанти цього методу. Аналіз показав, що головним джерелом втрат часу є рекурсія - тобто повернення на попередні кроки викладеного вище алгоритму під час заповнення таблиць нелінійних функціональних перетворень.

Під час побудови хеш-перетворення з програмованими колізіями необхідно знайти компроміс між двома вимогами: забезпечення якомога більшої траєкторії та практично унеможливлення їх віднаходження зловмисником. Очевидно, що ці

вимоги суперечать одна одній, тому в алгоритмі необхідно дотримуватися певної пропорції між детермінованим та випадковим.

Також важливо зауважити, що при постановці задачі проектування хеш-перетворення з програмованими колізіями немає чіткого визначення оптимальності. Отже, досягнення лише певного наближення до оптимального результату на інтуїтивному рівні буде досяжним. Структурно, процедура побудови хеш-перетворення з програмованими колізіями може бути описана наступним чином.

Випадковим чином оберемо перший вхідний вектор і проведемо траєкторію, без урахування потреби економії значень при заповненні. Цей процес дасть вихідний вектор, що виступатиме одним з кодів ідентифікації віддаленого користувача системи. Випадкове заповнення значень функціональних перетворень при першому проході є необхідним для виключення можливості виникнення детермінованості стану перетворювача.

Побудову другої траєкторії варто розпочати з обрання такого вхідного вектору, що дозволив би використати половину визначених верхніх значень та половину нижніх у своєму представленні. Далі почнемо проводити траєкторію одночасно зверху і знизу, забезпечуючи виникнення стику на одному з середніх шарів перетворювача. Якщо стик зробити не вдалося внаслідок виникнення конфліктних значень, виконується рекурсивне повернення на крок назад або більше. Побудову наступної траєкторії починаємо знизу, причому оберемо такий вхідний вектор, щоб при кожному наступному обранні відсоток визначених раніше значень функціональних перетворень у вхідному ключі щораз збільшувався.

Для прикладу розглянемо перетворювач розрядністю 3. Позначимо через x_1,x_2,x_3 компоненти вхідного вектору X, а y_1,y_2,y_3 – вихідного Y. На початку нової ітерації, зафіксуємо вже відомі значення, щоб їх не змінювати. Оберемо новий вхідний вектор для виконання перетворення. Для заощадження кількості вільних значень функціональних перетворень, віддаймо пріоритет обранню вхідних

компонентів з множини співставних (зайнятих) вхідних ключів відповідних функціональних перетворень. Для цього, наприклад, залишимо значення x_1 незмінним відносно попереднього, а значення x_3 змінимо так, щоб відповідати переставному закону додавання, при цьому x_2 залишимо невідомим. Це дозволить уникнути повторення уже існуючих маршрутів. Далі, після заповнення простору перетворювача даними, шо є визначеними для такого вхідного вектору, отримаємо значну кількість наразі невідомих співставлень, тобто маршрут не завершено. Тепер задача побудови маршруту зводиться до пошуку таких співставлень між вхідними та вихідними значеннями функціональних перетворень, для досягнення коректного вихідного коду без конфліктів на шарах перетворень з застосуванням якомога меншої кількості нових значень.

Якщо значення, отримане після застосування першого функціонального перетворення на вхідному значенні x_1 уже є відомим, то для знаходження інших значень необхідно підібрати числа в точках 1 та 2 (див. Рис. 2.5), для забезпечення виконання всіх рівності в системі булевих рівнянь перетворювача.

Припустимо, що після проведення першої траєкторії в хеш-перетворювачі було отримано вихідний ключ $Y = \{2,4,6\}$. Надалі відкриті до застосуванні такі стратегії:

- заповнення проходом вниз: обрати новий вхідний вектор $X=\{x_1,x_2,x_3\}$ який би відрізнявся від створених раніше і використовував раніше визначені значення функціональних перетворень. Для цього потрібно отримати значення на виході першого шару перетворень, далі пробувати йти низхідним маршрутом.

- заповнення проходом наверх: обрати вхідний вектор $X=\{x_1,x_2,x_3\}$ таким чином, щоб вихідний код на після застосування операції зворотнього зв'язку $\{x_1 \oplus y_1, x_2 \oplus y_2, x_3 \oplus y_3\}$ можна було отримати взагалі без заповнення функціональних перетворень новими значеннями. У такому разі необхідно, щоб код $\{x_1,x_2,x_3\}$ адресував максимум незаповнених значень – це необхідно для здійснення стику на першому шарі перетворювача без колізій. Далі рухаємося шлязом знизу наверх: на

кожному наступному j-тому шарі, знаючи значення суми s_{ji}, потрібно визначити такі значення r_{ji} та $r_{j,i+1}$, щоб $u_{ji} = v_{ji} \oplus v_{j,i+1}$, і v_{ji} було попередньо визначено в якості вихідного значення i-го функціонального перетворення, а $v_{j,i+1}$ – в якості вихідного коду (i+1)-го функціонального перетворення (рис. 2.5).

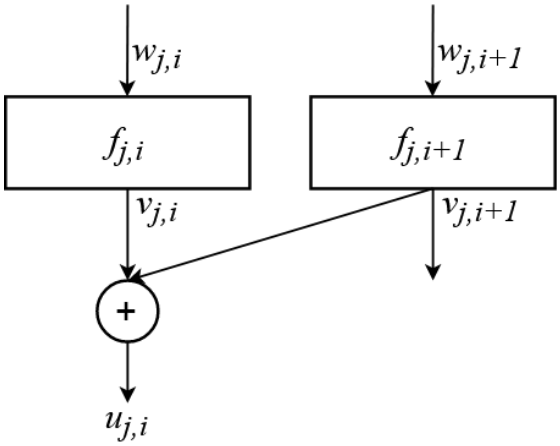

Рис. 2.5. Структура визначення значення функціонального перетворення

- комбінація проходу вниз та вверх: рухатись одночасно у низхідному і висхідному напрямках, а стик робити десь посередині перетворювача.

Ще одна стратегія полягає в наступному: скласти загальну велику систему рівнянь з урахуванням поточної наповненості функціональних перетворень. Далі, виконувати доповнення системи до однозначного розв'язку шляхом пошуку точок мінімального заповнення. Очевидно, що для проведення якомога більшої кількості траєкторій в перетворювачі, потрібно виконати якомога жадібніше використання раніше визначених значень виконання функціональних перетворень f_1, f_2, f_3, f_4. При цьому, логічна сума попарного перемішування компонент $w_1 \oplus w_2 \oplus w_3 \oplus w_4$ має дорівнювати нулю.

Так як $u_1 \oplus u_2 \oplus u_3 \oplus u_4 = 0$, то система (2.1) завжди має однозначний розв'язок. Наприклад: $u_1 = 5$, $u_2 = 3$, $u_3 = 7$, $u_4 = 1$. Встановимо $v_1 = 4$, тоді $v_2 = v_1 \oplus$

$u_1 = 1$. Відповідно, $v_3 = v_2 \oplus u_2 = 1 \oplus 3 = 2$, звідси $v_4 = v_3 \oplus u_3 = 2 \oplus 7 = 5$, в той час як, з іншого боку, $v_4 = u_4 \oplus v_1 = 1 \oplus 4 = 5$.

Застосуємо стратегію прокладення маршруту шляхом руху наверх. На першій ітерації оберемо деяку початкову комбінація значень, одержаних після застосування останнього шару перетворень в межах хеш перетворювача $\{y_1, y_2, y_3\}$. Саме це комбінація чисел стане опорною точкою при побудові траси для руху по хеш-перетворювачу. Далі, після виконання залучення операції зворотного зв'язку з вхідним вектором $\{x_1, x_2, x_3\}$, ми одержимо вектор $\{x_1 \oplus y_1,\ x_2 \oplus y_2,\ x_3 \oplus y_3\}$, який і є бажаним вихідним вектором.

Для коректного руху перетворювачем критично важливим є не порушити уже визначені атрибути, тобто на початку кожної наступної ітерації необхідно виконувати фіксацію уже відомих значень в межах відповідних функціональних перетворень, вважаючи їх константними у подальших ітераціях. Це означає, що при побудові нових трас у перетворювачі, дозволяється маніпулювати значеннями одержаними на поточній ітерації.

На кожній наступній ітерації при проході хеш-перетворювачем є необхідним згенерувати такі значення результатів застосування проміжних функціональних перетворень, які б в кінці проходу перетворювачем утворювали бажаний кінцевий вектор $\{x_1 \oplus y_1,\ x_2 \oplus y_2,\ x_3 \oplus y_3\}$, для цього на всіх наступних ітераціях після першої визначаємо вхідний вектор $X' = \{x_1', x_2', x_3'\}$ та вихідний $Y' = \{y_1', y_2', y_3'\}$, одержаний внаслідок застосування кінцевого етапу перетворень за допомогою відповідних функціональних перетворень таким способом, щоб після застосування механізму зворотного зв'язку, вони утворювали вихідний вектор $Y = \{x_1 \oplus y_1,\ x_2 \oplus y_2,\ x_3 \oplus y_3\}$, визначений як вихідний ключ після виконання першої ітерації проектування хеш-перетворювача.

При виборі вхідних даних слід намагатися використовувати значення, які вже передбачені функціональним перетворенням, що допоможе оптимізувати

використання значень функцій. У випадку, якщо це неможливо, необхідно визначити нові значення вхідних даних.

Наприклад, під час наступної ітерації ми зосередимося на фіксації вхідних та вихідних значень на конкретній колонці. Це досягається шляхом симетричного відображення значень в інших колонках. Наприклад, припустимо, що ми обрали вхідний вектор $\{x_1,y_2,y_3\}$. Такий вибір дозволить нам зберегти значення кінцевого вектору незмінними. Використання такого методу обрання вхідних даних не є обов'язковим, однак воно дозволяє отримати опорну точку для заповнення перетворювача та більш оптимально перебирати можливі варіанти значень вхідних векторів. Аналогічно, наступні ітерації будуть передбачати фіксацію вхідних та вихідних значень інших колонок.

Далі заповнення перетворювача виконується шляхом поєднання низхідного та висхідного проходжень хеш-перетворювачем, використовуючи відомі проміжні перетворення. Спочатку встановлюються відомі дані проміжних перетворень, після чого проводиться аналіз центральної частини перетворення за допомогою розв'язання систем лінійних рівнянь. Застосовуються можливі варіанти вхідних та вихідних даних функцій, що поступово підставляються в систему, поки не будуть знайдені всі необхідні значення змінних. Для цього використовується центральний блок перетворень функцій як інструмент для їх з'єднання. У разі виникнення суперечливих значень аргументів відносно функцій перетворення після підстановки всіх можливих варіантів, потрібно змінити початкові дані вхідного вектору.

Виконаємо ілюстрацію прикладу заповнення функціонального перетворення значеннями на етап виконання деякої ітерації проектування перетворювача при використанні повністю унікальних вхідних та вихідних ключів. Припустимо, що значення u_1 та u_2, були обрані як вхідні та вихідні значення на деякій колонці перетворень та не є визначеними в межах відповідного булевого функціонального перетворення.

Очевидно, що у такому разі гарантовано будуть використані принаймні два нових значення функціонального перетворення, тому, для реалізації задачі мінімізації заповнення значень, використаємо в якості вхідних значень v_1 та v_2 деякі ключі, які є уже визначеними в межах відповідних функціональних перетворень. У разі, якщо кожне з цих значень неможливо використати для детермінації аргументів поточної ітерації проектування хеш-перетворювача, тоді припустимо повторення блоків, а саме блок №2 повторює блок №1, тобто $u_1 = v_1$.

У разі, якщо це також не є можливим, тоді виконаємо спробу представлення блоку вхідних вихідних даних поточної ітерації блоку №2 як копію відповідних даних блоку №3, тобто $u_1 = v_2$. Якщо при використанні таких даних відбувається виникнення конфліктів, можливим рішенням є лише залучення нових значень функцій, або ж повернення на ітерацію назад для проби застосування інших параметрів, і саме такий підхід є пріоритетним, адже використання нових даних на поточній ітерації не запобігає виникнення конфліктних станів у майбутньому.

Якщо детальніше розглянути спосіб перемішування змінних між послідовними шарами перетворювача, то легко помітити, що підбір потрібних значень відповідних функціональних перетворень зводиться до розв’язання циклічних систем рівнянь, що є однією з ключових мірників оцінки параметрів функціональних перетворень при їх проектуванні. Приклад перемішування для трьох секцій наведено на рисунку 2.6.

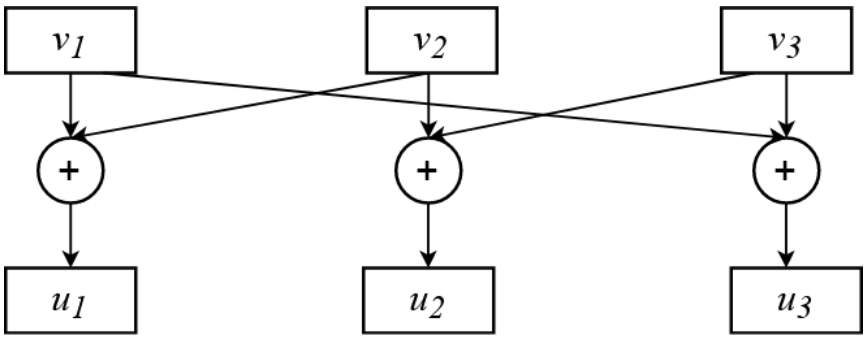

Рис. 2.6. Приклад перемішування даних для трьох секцій

З огляду на наведені вимоги до хеш-перетворювача опишемо загальні правила проектування його функціональних перетворень:

1. Повторне використання послідовних перетворень на одній колонці може призвести до колізій значень функцій або дублювання послідовності. Створення нової комбінації вхідного вектору на основі такого методу неможливе. Якщо використовувати ті ж послідовні вхідні та вихідні дані, центральна частина перетворення повинна бути змінена.

2. Використання пар відповідностей ключів та значень функцій у зворотному порядку (v, u) та (u, v) призводить до конфліктів значень.

3. Заміна значень на вході лише одного функціонального блоку вхідного вектору призводить до вимоги відновлення оригінального значення послідовності.

Очевидно, що зменшення кількості вхідних та вихідних значень функції на початкових етапах проектування забезпечує більш широку зону можливості розширення на наступних ітераціях підбору. Таким чином, важливим завданням при розробці ефективного побудови є визначення стратегії, яка дозволить зменшити використання значень функції та оптимізувати їх розподілення.

Для виконання цих вимог оберемо деяку початкову комбінацію значень $\{v_1,v_2,v_3\}$ та виконаємо їх попарне перемішування шляхом їх попарного сумування за модулем двійки. На вихід отримаємо комбінацію чисел $\{u_1,u_2,u_3\}$, яка стане базовою точкою для подальшої побудови перетворювача.

Для виконання першої ітерації циклу проектування, проведемо співставлення відповідних значень функціональних перетворень наступним чином: v_1 -> u_1, v_2 -> u_2, v_3 -> u_3. Надалі заповнимо ланцюжки перетворень дублюючими зонами перетворень та отримаємо вхідний та вихідний вектори, використовуючи лише одне значення табличної нелінійної булевої функції.

Далі, використовуючи визначені співставлення v_1 -> u_1, v_2 -> u_2, v_3 -> u_3 за базис, виконаємо підбір значення іншого вхідного вектора. Для цього, продублюємо одну з компонентів базового вхідного вектору, симетрично

відтворюючи інші значення. Наприклад, якщо вхідним вектором є $\{u_1,v_2,v_3\}$, то одним з його відповідних вихідних векторів є $\{v_1,u_2,u_3\}$.

Доречно зауважити, що у разі якщо встановлення відповідностей деяких вхідних або вихідних значень функціонального перетворення ще не відбулося, то ці значення залишаються невизначеними, до тих пір, доки не буде виконано побудову деякого маршруту, який потребує задоволення вимоги встановлення саме цього значення. Підбір здійснюється методом пошуку першого відповідного екземпляру, який не викликає виникнення конфліктних значень в межах будь-якого з функціональних перетворень. Якщо такого значення не існує, то необхідно здійснити рекурсивне повернення на крок назад до попереднього розгалуження маршруту, аж до повної заміни вхідного вектору.

На кожній наступній ітерації підбору вхідного вектору деякого проміжного шару хеш-перетворювача, підбір значень відбувається відносно функціонального перетворення, що є найбільш заповненим у напрямку на функціонального перетворення з меншою кількістю заповнених значень функції у разі якщо існує декілька функціональних перетворень з однаковою межею заповнення (див. приклад на рис. 2.7).

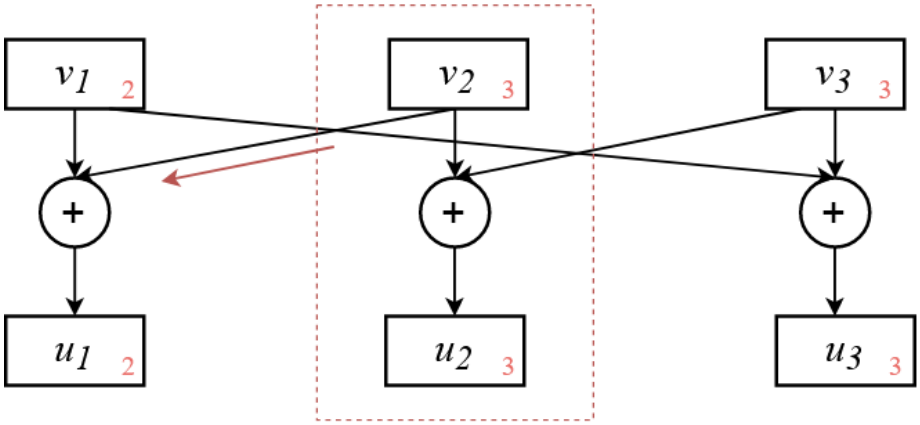

Рис. 2.7. Схема обрання керуючого функціонального перетворення при виконанні підбору значень

Для ефективної мінімізації заповнення функціонального перетворення значеннями необхідно дотримуватись важливої вимоги – на кожній наступній ітерації проектування використовувати не більше двох нових значень. Це можна досягти методом повторення блоків перетворень, тобто шляхом використання значень ключів функціональних перетворювачів, які є визначеними, як базових. Це дозволить забезпечити поповнення кожного функціонального перетворення 1-2 значеннями за одну ітерацію. При цьому, навіть якщо під час ітерації вдається встановити не встановити нових значень для деякого функціонального перетворення, вважати це успіхом можна тільки в тому випадку, якщо різниця між граничними кількостями заповнених значень у всіх булевих перетворювачах не перевищує 2. Застосування більшої кількості значень на одну функцію за ітерацію, навіть при абсолютній відповідності, слід вважати провалом. У такому разі необхідно повторити ітерацію для пошуку кращого рішення. Така стратегія дозволить не лише скоротити об’єми використання незаповнених значень функціональних перетворень при побудові, а й здійснювати балансування їх розподілення між функціями.

Для ілюстрації роботи запропонованого методу порівняємо його ефективність з проектуванням без системи оптимізації, тобто способом пасивного вибору на вимогу. На рис. 2.8 наведено приклад послідовності заповнення функціональних перетворень значеннями відповідно до встановлених вимог, тоді як на рис. 2.9 – заповнення випадковим чином. Протиставлення цих прикладів дозволяє легко помітити, що навіть після чотирьох послідовних проходів хеш-перетворювачем, запропонований метод виявляє значно кращу ефективність у порівнянні з заповненням функціональних перетворень випадковим чином навіть при меншій кількості виконаних ітерацій побудови.

Структура включає шари перемішувань вихідних значень сусідніх блоків, кількість яких визначена кількістю секцій, що дозволяє виконати повне перемішуванням компонентів вхідного сеансового паролю, що означає, що кожен

біт вихідного ідентифікаційного ключа, отриманого після виконання проходу хеш-перетворювачем, явно залежить від всіх бітів вхідного ключа, що виступає в якості сеансового паролю користувача.

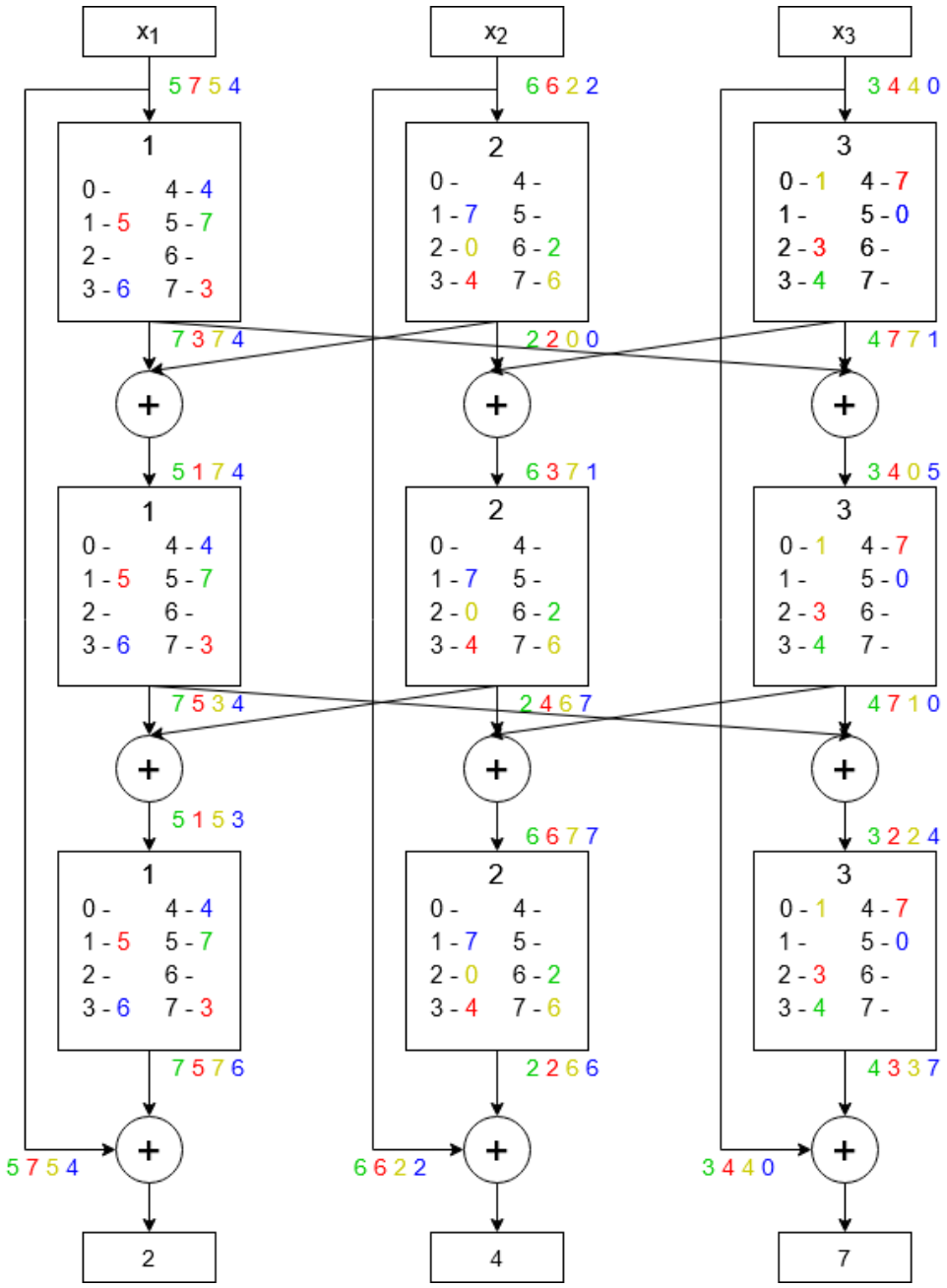

Рис. 2.8. Приклад заповнення таблиць при оптимізованому проході

Алгоритм побудова хеш-перетворення з програмованими колізіями полягає в послідовному заповненні таблиць значень нелінійних булевих функціональних

перетворень для виконання однозначного співставлення значень на області визначення останніх. Кількість таких таблиць, тобто кількість секцій перетворювача, має бути якомога меншою, при цьому їх об'єм має бути якомога більшим для збільшення можливостей диференціації маршрутів, які можна провести в межах перетворювача за рахунок залучення компонент більшої розрядності.

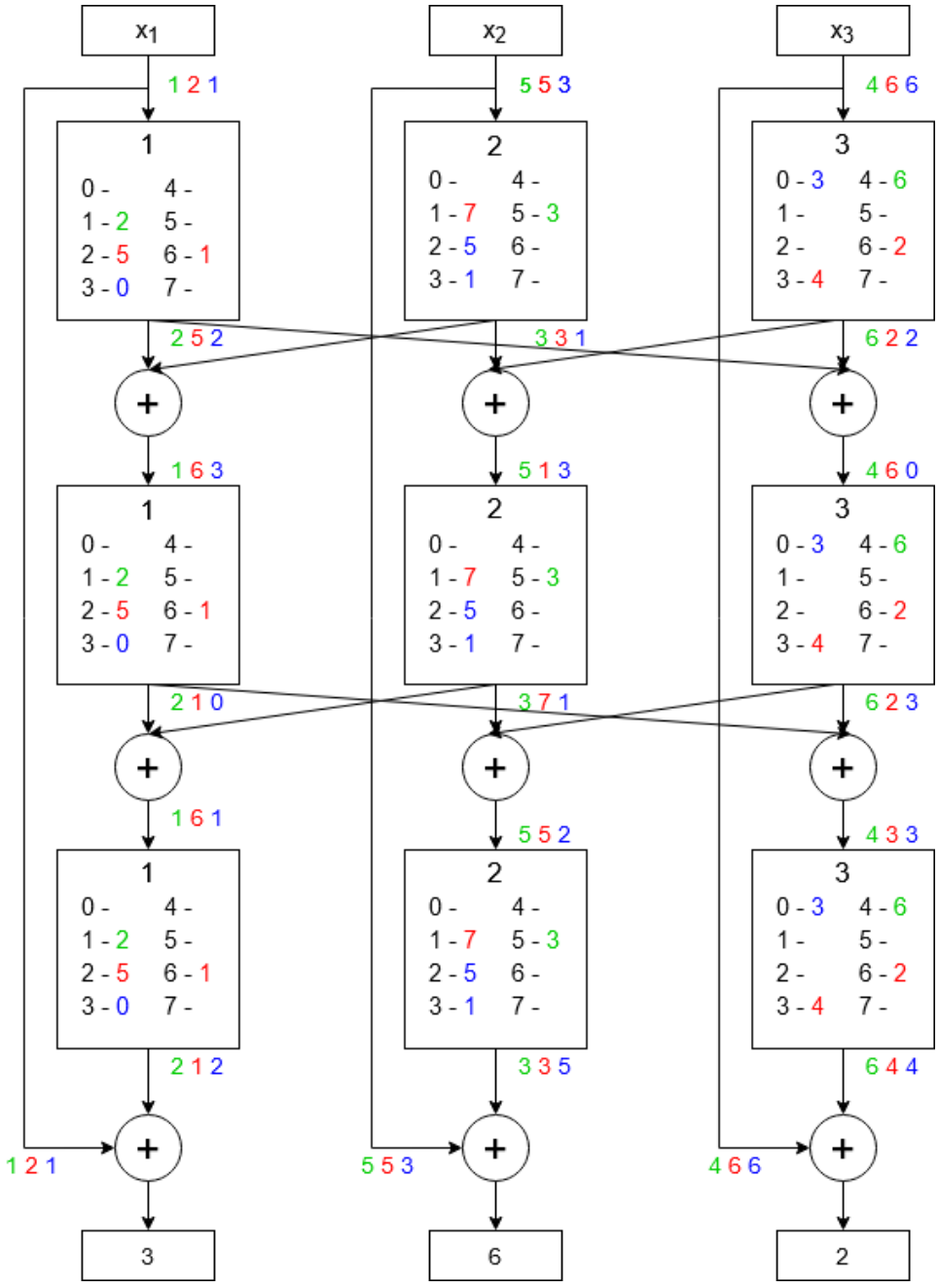

Рис. 2.9. Приклад заповнення таблиць при пасивному проході

Ключовим критичним моментом при побудові траєкторії русі хеш-перетворювачем є реалізація визначення значень нелінійних булевих функціональних перетворень при стику – тобто при зустрічі східного та низхідного проходів.

Схему стику двох шарів перетворень представлено на рис. 2.10.

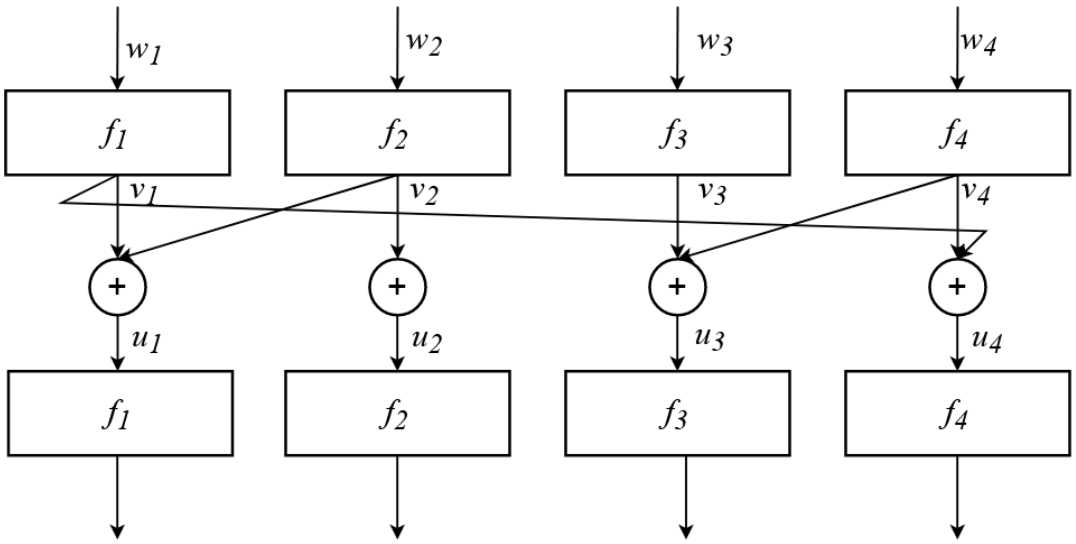

Рис. 2.10. Організація стику траєкторій східного і низхідного проходів

Припустимо, що значення w_1,w_2,w_3,w_4 та u_1,u_2,u_3,u_4 є визначеними та відомими, водночас як значення $f_1(w_1),f_2(w_2),f_3(w_3),f_4(w_4)$ функціональних перетворень вважаються невідомими, тобто є невизначеними.

Аналогічним чином значення змінних v_1,v_2,v_3,v_4 повинні задовольняти системі рівнянь (2.2). Кількість рівнянь системи (2.2) рівна k-1 та є на одиницю менше за число змінних k, що означає, що її кількість можливих розв'язків є відповідною максимальній допустимій кількості можливих значень кожної зі змінних $v_1,v_2,\ldots,v_k$, при цьому завжди існує їх однозначна комбінація. Максимальна допустима кількість можливих значень змінних $v_1,v_2,\ldots,v_k$, в свою чергу, визначається числом $\eta_1,\eta_2,\ldots,\eta_k$ значень функціональних перетворень $f_1,f_2,\ldots,f_k$ які не є визначеними. Нехай, $j\in\{1,2,\ldots,n\}$ – індекс стовпця функціонального перетворення f_j, для якого

кількість невизначених значень є максимальною: $\eta_j = max\{\eta_1,\eta_2,\ldots,\eta_k\}$. У такому разі система (2.2) має щонайбільше η_j можливих рішень.

Після обрання одного із η_j допустимих значень змінної v_j та розв'язку оновленої системи рівнянь (2.2) отримаємо значення змінних $v_1,v_2,\ldots,v_k$.

Для того, щоб використати отриманий розв'язок системи (2.2) для довизначення функціональних перетворень $f_1,f_2,\ldots,f_k$ на вхідних наборах $w_1,w_2,\ldots,w_k$, необхідно, щоб для кожного наступного функціонального перетворення вихідні значення $v_1,v_2,\ldots,v_k$ не були визначеними раніше. Якщо позначити ймовірність того, що для i-того функціонального перетворення вихідне значення не було визначене раніше через $p_i = \eta_i/2^n$, де $i \in \{1,2,\ldots,k\}$, тоді ймовірність можливості використання розв'язку системи для довизначення значень функціональних перетворень може бути обрахована за формулою:

$$Q = \prod_{i=1}^{k} p_i = \prod_{i=1}^{k} \frac{\eta_i}{2^n}. \qquad (2.14)$$

Отже, ймовірність P того, що в результаті розв'язку системи (2.2) принаймні при одному із η_j допустимих значень змінної v_j буде отримано значення змінних $v_1,v_2,\ldots,v_k$, які в подальшому можуть бути використані для довизначення значень функціональних перетворень $f_1,f_2,\ldots,f_k$ на вхідних значеннях $w_1,w_2,\ldots,w_k$, може бути обрахованою за наступною формулою:

$$P = 1-(1-Q)^{\eta_j} = 1-(1-\frac{1}{2^{k\cdot n}}\cdot\prod_{i=1}^{k}\eta_i)^{\eta_j}. \qquad (2.15)$$

У випадку якщо кількість невизначених значень для всіх булевих функціональних перетворень $f_1,f_2,\ldots,f_k$ є приблизно однаковою, тобто $\eta_1 \approx \eta_2 \approx \ldots \approx \eta_k$, у такому разі відношення числа невизначених значень функціональних перетворень до їх загальної кількості рівне $\gamma = \eta/2^k$ та приблизно однаковим всіх функціональних перетворень. Як наслідок, формула (2.15) може бути записана у вигляді:

$$P = 1-(1-\gamma^k)^{\gamma\cdot 2^k}. \qquad (2.16)$$

В якості ілюстрації в таблиці 2.2. наведені значення ймовірностей здатності побудови траєкторії в хеш-перетворювачі, обчислених за формулою (2.16).

Таблиця 2.2

Залежність ймовірності здатності побудови траєкторії без використання рекурсії залежно від ступеню заповнення таблиць

m	n	Завантаженість - (1- γ)		
		0.5	0.6	0.7
4	4	0.41	0.19	0.08
	5	0.79	0.29	0.95
	6	0.87	0.53	0.12
5	4	0.23	0.07	0.06
	5	0.41	0.14	0.22
	6	0.62	0.25	0.47

Для оцінки розробленого методу побудови хеш-перетворювача з програмованими колізіями були виконані експериментальні дослідження для встановлення залежності часу побудови хеш-перетворювача від його основних характеристичних параметрів: доступної розрядності сеансових ключів (n) та їх загальної кількості. Візуалізацію отриманих експериментальним шляхом залежностей показане у графічній формі на рис. 2.11.

Відповідним чином, за використанням розроблених програмних засобів було проведено експериментальне дослідження залежності ймовірності підбору паролю зловмисником відносно кількості можливих паролів. Візуалізація відповідної залежності представлена на рис. 2.12.

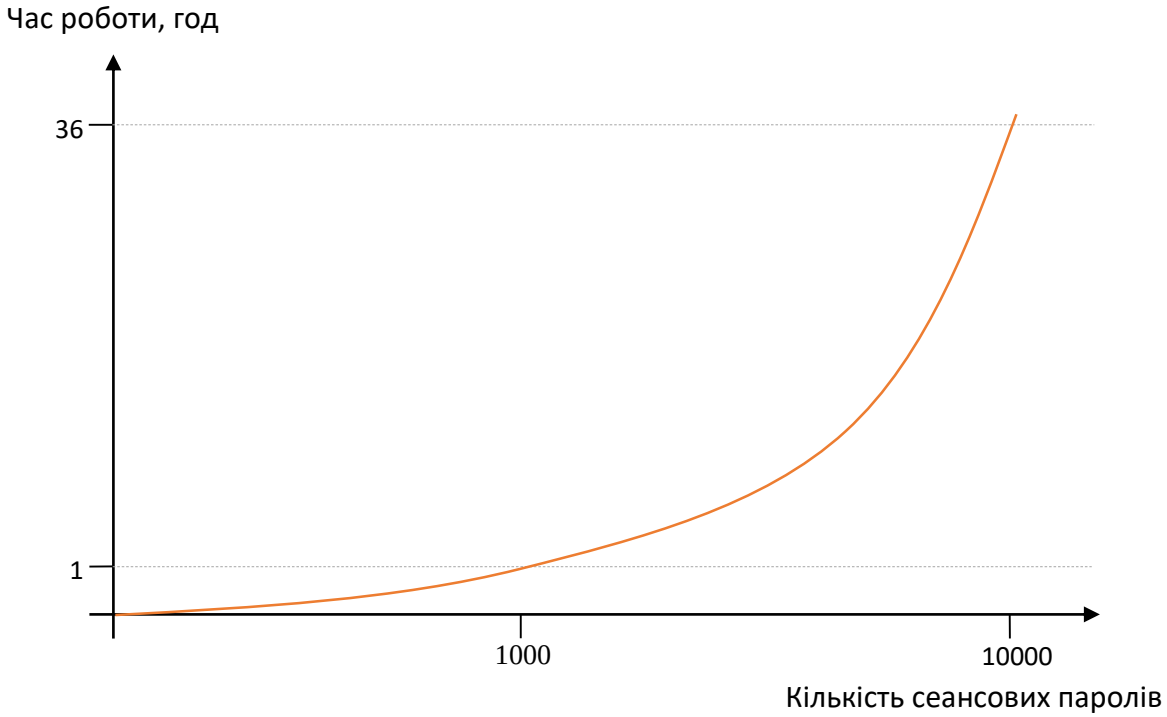

Рис. 2.11. Залежність часу побудови сеансових паролів відносно їх кількості на типовому персональному комп'ютері для n=1024.

Аналіз залежностей, які представлені на рисунках 2.11 та 2.12, дозволяє зробити висновок про те, що розроблений метод забезпечує достатню кількість сеансових паролів для практичного застосування на доступних для широкого кола користувачів обчислювальних платформах. З іншого боку, було показано, що при застосуванні 10^4 різних сеансових паролів, ймовірність виконання їх успішного підбору зловмисником залишається критично низькою, що свідчить про ефективність захисту, який є достатнім для практичного застосування.

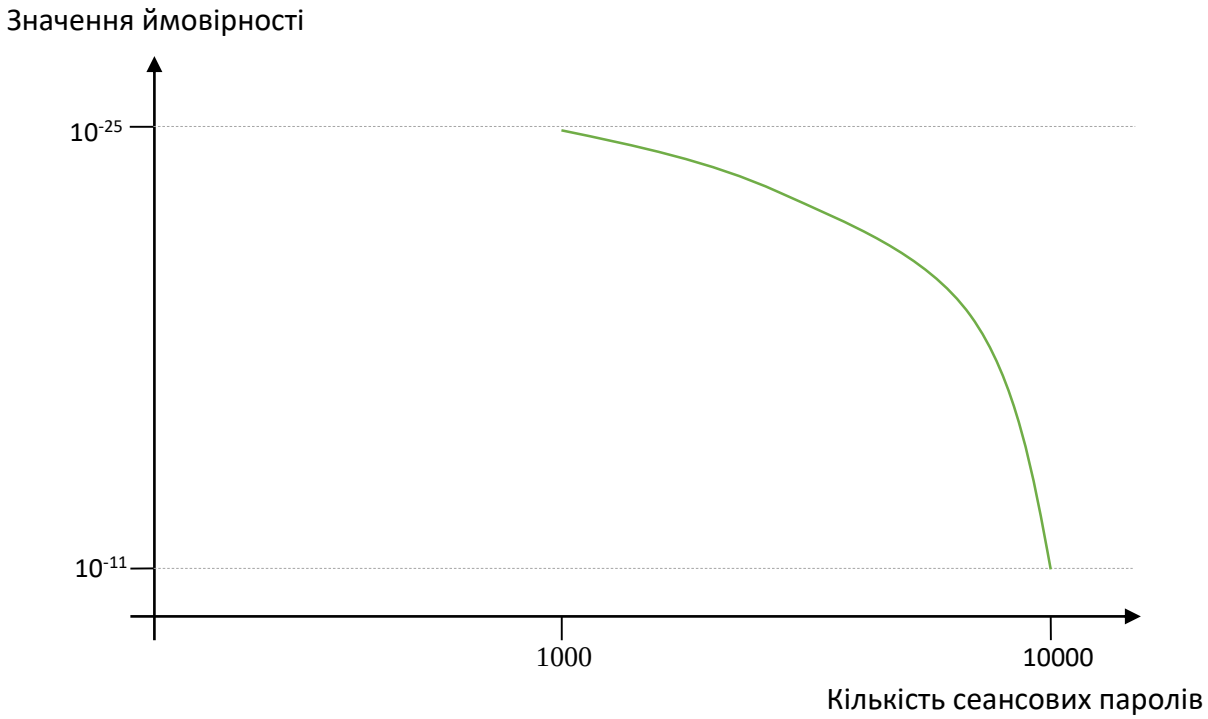

Рис. 2.12. Залежність ймовірності підбору сеансового паролю зловмисником відносно кількості можливих паролів

2.4 Розробка методу збільшення кількості сеансових паролів криптографічно строгої ідентифікації

Проведений аналіз існуючих методів побудови хеш-перетворень без колізій для криптографічно строгої ідентифікації віддалених користувачів показав, що основним недоліком існуючих схем таких перетворень є обмежена кількість можливих вхідних векторів.

Ціль досліджень полягає в підвищенні ефективності криптографічного строгої ідентифікації на основі хеш-перетворень з програмованими колізіями за рахунок збільшення кількості сеансових паролів доступу.

Для досягнення поставленої цілі досліджень пропонується збільшити кількість ступенів свободи при формуванні траси обчислення коду доступу з кодів сеансових паролів за рахунок розширення множини можливих кодів доступу.

В відомих схемах хеш-перетворювачів з програмованими колізіями формування трас здійснюється за умови, що існує для кожного із користувачів лише один можливий код доступу. В розробленому методі пропонується використовувати для ідентифікації конкретного віддаленого користувача не один, а багато кодів доступу. Це дозволить суттєвим чином збільшити кількість можливих трас обчислення коду доступу.

Множину коректних *n*-розрядних кодів $Y_1,Y_2,..,Y_\eta$ доступу кожного із віддалених користувачів пропонується формувати наступним чином. Молодші *n*/2 розрядів кожного із кодів $Y_1,Y_2,..,Y_\eta$ для кожного із віддалених користувачів можуть приймати довільні значення. Старші *n*/2 розряди кожного із кодів $Y_1,Y_2,..,Y_\eta$ пропонується формувати в результаті шифрування молодших *n*/2 розрядів кодів доступу з використанням секретного ключа К, яким володіють тільки користувач та система. Схема формування коду доступу користувача показана на рис.2.13.

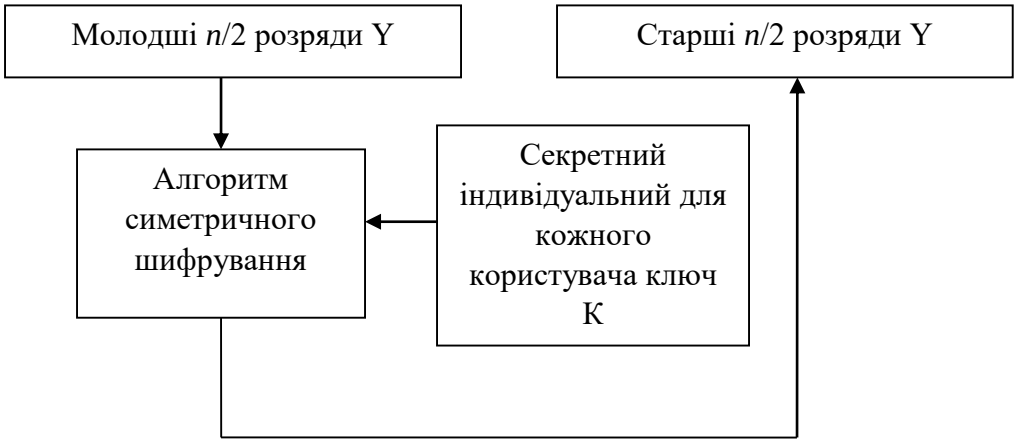

Рис. 2.13. Схема формування ключа доступу користувача

В принципі, для формування множини кодів доступа користувача можуть бути використані і стандартизовані хеш-перетворювачі типу SHA-256 або Ripemd-160. Відповідна схема представлена на рис.2.14. Важливою перевагою використання для формування ключів доступу стандартизованих алгоритмів

симетричного шифрування типу DES або AES, а також стандартизованих хеш-алгоритмів типу SHA-256 є те, що їх захисні властивості пройшли всебічно перевірку, а також те, що ці алгоритми реалізуються на апаратному рівні в усій криптопроцесорах, які входять в склад практично всіх обчислювальних платформ крім мобільних пристроїв. За забезпечує високі характеристики безпеки та швидкодії при виконання передбаченим запропонованим методом операції формування кодів доступу.

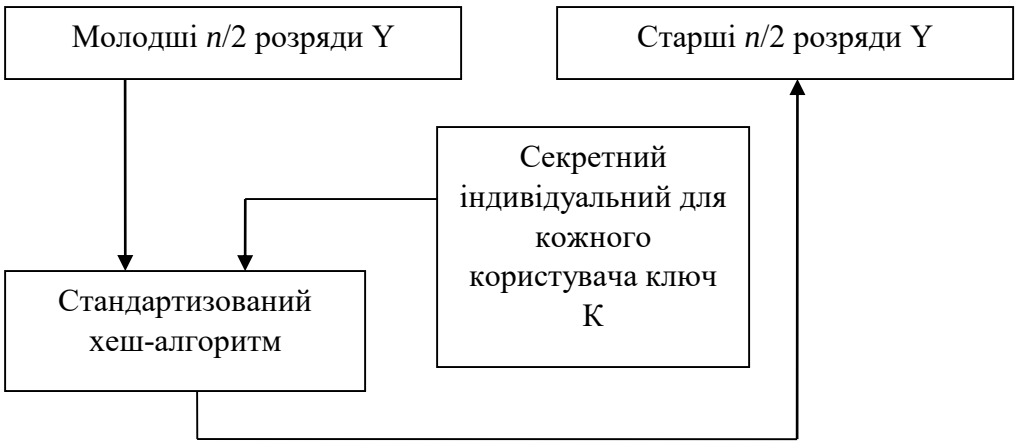

Рис. 2.14. Схема формування ключа доступу користувача на основі стандартизованого хеш-перетворення

Вважається, що хеш-перетворювач з програмованими колізіями має розрядність n, містить k секцій і, відповідно, таку ж кількість шарів. Розрядність кожної секції становить $m=n/k$. Це означає, що вхідний n бітовий вектор X складається із k m-розрядних слів: $X=\{x_1,x_2,\ldots,x_k\}$, де $\forall j=1,2,\ldots,k$: $x_j \in \{0,1,\ldots,2^m-1\}$. Відповідно, кожен із k функціональних перетворювачів $f_1,f_2,\ldots,f_k$ має m входів та m виходів, тобто його об'єм становить $m\cdot 2^m$. Якщо значення функції для аргументу x не визначено, то воно вважається рівним -1.

Нижче наведено процедуру формування векторів X, крім першого.

1. Обираються компоненти $x_1,x_2,\ldots,x_k$ вхідного вектору X такі, щоб для них були визначені значення відповідних функцій, тобто, щоб $f_1(x_1) \neq -1$, $f_2(x_2) \neq -1$, …, $f_k(x_k) \neq -1$.

2. Довільним чином обираються перші k компонент вихідного вектору $Y = \{y_1, y_2, \ldots, y_k\}$, $\forall j=1,2,\ldots,k$: $x_j \in \{0,1,\ldots,2^m-1\}$ таким чином, щоб для логічних сум $x_1 \oplus y_1$, $x_2 \oplus y_2$, …, $x_k \oplus y_k$ були визначені значення відповідних функцій, тобто, щоб $f_1(x_1 \oplus y_1) \neq -1$, $f_2(x_2 \oplus y_2) \neq -1$, …, $f_k(x_k \oplus y_k) \neq -1$.

3. Далі здійснюються низхідна і висхідна процедури побудови траєкторії формування вихідного вектору так, щоб мінімізувати кількість визначення значень функцій $f_1, f_2, \ldots, f_k$.

4. Виконується процедура стику низхідної і висхідної траєкторій формування вихідного вектору.

Запропонований метод формування хеш-перетворення з програмованими колізіями для криптографічно строгої ідентифікації віддалених учасників інформаційної взаємодії передбачає виконання наступної послідовності дій на етапі реєстрації системою нового користувача.

1. Користувач довільним чином обирає секретний ключ К.

2. В межах виділеного часового інтервалу виконується формування сеансових ключів з формуванням траєкторій їх обчислення до отримання на виході хеш-перетворювача одного із допустимих кодів доступу. При цьому, в процесі трасування здійснюється заповнення таблиць булевих функціональних перетворень.

3. Після закінчення обраного часового інтервалу формування множини вхідних сеансових паролів здійснюється заповнення таблиць булевих функціональних перетворень випадково обраними значеннями.

4. Обраний користувачем секретний ключ К формування розширення коду доступу шифрується відкритим ключем системи і надшиється в систему.

5. Надсилаються користувач в систему сформовані таблиці булевих функціональних перетворень.

6. Система зберігає отриманий від користувача секретний ключ К в спеціальній захищеній пам'яті, а таблиці функціональних перетворень зберігаються системою на диску в розділі, відведеному для даного користувача.

На цьому виконання процесу реєстрації користувач системою вважається закінченою.

Розроблений метод формування хеш-перетворення з програмованими колізіями для криптографічно строгої ідентифікації віддалених учасників інформаційної взаємодії передбачає виконання наступної послідовності дій на етапі ідентифікації користувача.

1. Користувач обирає один із сформованих сеансових паролів X, який раніше не використовувався і надсилає його системі.

2. Система читає із дискової пам'яті таблиці булевих функціональних перетворень даного користувача і з їх використанням обчислює вихідний код Y хеш-перетворення, на вхід якого подається отриманий системою сеансовий пароль X користувача. Вихідний код Y розділяється на дві частини: $Y_1 = \{y_1, y_2, \ldots, y_{n/2}\}$ та $Y_2 = \{ y_{n/2+1}, y_{n/2+2}, \ldots, y_n\}$.

3. Обчислюється значення бінарного вектору $Y' = E(Y_1, K)$, де E- функція симетричного шифрування коду Y_1 ключем К. Якщо $Y' = Y_2$, то користувачеві надаються права доступу до ресурсів віддаленої системи.

Основний ефект наведеної процедури криптографічно строгої ідентифікації в плані підвищення ефективності визначається тим, що запропонований метод побудови хеш-перетворення з програмованими колізіями для криптографічно строгої ідентифікації дозволяє значно прискорити процес синтезу зазначеного перетворення при збільшенні кількості можливих сеансових паролів доступу. Збільшена кількість сеансових паролів, яка забезпечується запропонованим методом дозволяє збільшити число циклів ідентифікації до переналаштування системи.

Висновки до розділу 2

За результатами досліджень, які складають другий розділ магістерської дисертації і направлені на теоретичне обґрунтування та розробку методу підвищення ефективності криптографічно строгої ідентифікації на основі хеш-перетворень з програмованими колізіями можна зробити наступні висновки:

1. Одним із важливих напрямків підвищення ефективності криптографічно строгої ідентифікації на основі хеш-перетворень з програмованими колізіями є збільшення кількості сеансових паролів. Реалізація такого збільшення може досягнута за рахунок збільшення витрат ресурсів пам'яті або за рахунок надання більшої степені свободи процедурі трасування обчислень. Другий шлях є більш перспективним в силу того, що він потребує менше ресурсів.

2. Теоретично обґрунтовано, розроблено та досліджено метод побудови хеш-перетворення з програмованими колізіями для криптографічно строгої ідентифікації, який відрізняється тим, що ідентифікатором учасника віддаленої взаємодії виступає не фіксований код, а певна сукупність кодів, біти яких пов’язані певними криптографічними відношеннями, які залежать від секретного ключа, за рахунок чого збільшується кількість колізій хеш-перетворення і, тим самим на значною міною знімається обмеження на кількість можливих сеансових ключів криптографічно строгої ідентифікації.

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ ДЛЯ ПОБУДОВИ ХЕШ-ПЕРЕТВОРЕННЯ З ПРОГРАМОВАНИМИ КОЛІЗІЯМИ

Програмний продукт створюється для експериментального дослідження запропоновано і теоретично дослідженого в другому розділі методу криптографічно строгої ідентифікації на основі незворотних перетворень з програмованими колізіями.

Програмний продукт має вирішувати задачі, як побудови хеш-перетворення з програмованими колізіями для практичного використання, так і експери-ментального визначення показників ефективності криптографічно строгої ідентифікації, що використовують такі незворотні хеш-перетворювачі. В якості основних критеріїв ефективності в рамках розробки використовуються:

- число сеансових паролів користувача, який генерується системою і які він без повторень може використовувати для організації сеансу віддаленої взаємодії з системою до її пере налаштування;

- час генерування незворотного хеш-перетворення, програмовані колізії при цьому утворюють сеансові паролі користувача на обчислювальній платформі користувача;

- об’єм пам'яті, який використовується в хеш-перетворення для реалізації властивості незворотності, основаній на нелінійності булевих функціональних перетворень, які реалізуються табличними булевими перетворювачами; об’єм пам’яті табличних булевих перетворювачів не є критичним для користувача, але є критичним для системи колективного доступу, яка обслуговує в реальному часі десятки і сотні віддалених користувачів.

Оскільки програмний продукт використовується окремо кожним із віддалених користувачів системи масового обслуговування, його архітектура має

бути орієнтована на технічні особливості обчислювальної платформи користувача, в якості якої в переважній більшості випадків виступає персональний комп’ютер. .

3.1 Організація структур і даних

Одним з ключових супутніх компонентів програмного комплексу є клас ConversionFunction, що є високорівненим представленням структури функціонального перетворення в межах розроблюваного хеш-перетворювача. В класі реалізовано функціональність зберігання пар вхідних і вихідних значень перетворення та доступі до цих значень за допомогою методів, які забезпечують зручний інтерфейс для роботи з функціональним перетворенням.

Клас ConversionFunction допомагає виконувати базові операції, пов'язані з функціональними перетвореннями, і забезпечує зручний доступ до вхідних та вихідних значень функцій. Використовуючи методи класу, можна легко визначити наявність відповідності між вхідним та вихідним значеннями, отримати списки вільних і зайнятих ключів та значень, отримати статистичні відомості про кількість використовуваних значень тощо.

Однією з проблем організації даних для реалізації представлення функціонального перетворення є необхідність забезпечення швидкого доступу до необхідних даних при помірних витратах пам’яті. Для гнучкого та ефективного зберігання і співставлення значень функціонального перетворення, в класі ConversionFunction використовується колекція типу HashMap, яка забезпечує якісний доступ до значень шляхом швидкого пошуку ключа за допомогою хеш-функції. Ключова перевага HashMap полягає в тому, що часова складність вставки та отримання значення в ній в середньому становить O(1). Ключем у колекції є значення вхідного параметра, а значенням – відповідне значення функції на цьому параметрі. Значення функції зберігаються у колекції у вигляді пар вхідного та

вихідного параметрів, забезпечуючи швидкодію та продуктивність доступу до даних.

Кожне вхідне значення функціонального перетворення має однозначний відповідник на своїй області визначення, обсяги якої встановлюються на етапі ініціалізації екземпляру класу за допомогою параметру rate – величину розрядності вхідних та вихідних значень функціонального перетворення. Якщо ж деякий момент часу певний ключ не має відповідника, тобто не є визначним, то його значення вважається рівним константі ValueConstants.UNMAPPED, яка є рівною -1.

Кожне значення функціонального перетворювача представлено числовим значенням типу даних long, яке є 64-бітним знаковим цілим числом. Найбільша степінь двійки, яка може бути представлена в типі long, може бути обчислена за формулою: 2^{63}, адже один біт відведено для знака. Таким чином, найбільше можливе значення змінної rate становить 63, а області визначення та значень функціональних кожного функціонального перетворення обмежено числами відповідної розрядності.

Для забезпечення збережень значень більшої розрядності замість тип значень внутрішнього стану функціонального перетворювача варто замінити на тип BigInteger. Цей клас дозволяє представляти довільно великі цілі числа та не має фіксованого обмеження на розмір числа. Максимальний розмір числа обмежується тільки обсягом пам'яті, яка доступна для програми, яка використовує клас BigInteger. Важливо зауважити, що об’єкти такого типу, у зв’язку з внутрішньою організацією, використовують значно більші об’єми пам’яті у порівняння з аналогами-примітивами для представлення однакових числових значень. Таким чином використання типу даних long або класу BigInteger залежить від потреб програми в роботі з великими цілими числами, та від обсягу пам'яті, яку програма може використовувати.

Імплементацію ключа даних хеш-перетворювача реалізовано в класі DataKey. Цей клас представляє собою контейнер, що містить набір послідовних числових

значень у вигляді масиву типу long, визначаючи вектор даних, отриманий після розбиття вхідного ключа у вигляді єдиного числа на фрагменти визначеної розрядності. Використання статичної адресації для збереження представлення ключа дозволяє зберігати та отримувати інформацію з масиву ключів з високою швидкістю та ефективністю.

Клас містить конструктори для створення об'єкту DataKey та методи управління ключами даних, а саме для створення, збереження, зчитування, збереження та перетворення ключів. Клас також містить статичні методи zip() і unzip(), що надають можливість стиснути та розгорнути масив ключів у вигляд єдиного числового значення. Метод zip() приймає на вхід об'єкт типу DataKey і повертає рядок, що містить стиснуті значення масиву ключів у вигляді послідовності цифр без розділових знаків. При перетворенні цього рядка у число, отримаємо кінцеве упаковане значення вектору даних. Метод unzip() виконує логічно протилежну операцію – розбирає переданий на вхід єдине числове значення на фрагменти заданої розрядності та повертає об'єкт типу DataKey з розгорнутими значеннями ключів.

Для збереження стану проміжних обчислень на шарах перетворювача, що є необхідним при прокладенні кожного нового маршруту та його візуалізації, додатково реалізовано клас ConversionPair, який є простою структурою даних, що зберігає пару чисел для використання у конвертації. Цей клас містить два поля типу long - in та out, що відповідають вхідному та вихідному значенням відповідно.

3.2 Реалізація програмного модулю хеш-перетворювача

Клас HashConverter є реалізацією описаного методу утворення хеш-перетворювача з програмованими колізіями та здійснює інтеграцію всіх основних дій, передбачених методом побудови, інкапсулюючи в собі структуру хеш-перетворювача з програмованими колізіями в цілому.

Внутрішня організація класу HashConverter вимагає збереження копії стану функціональних перетворень, що складають основу його шарів перетворень, у вигляді набору послідовних екземплярів класу ConversionFunction. Так як звернення до певного функціонального перетворення вимагає вибору стовпця визначеного індексу, збереження цього набору у вигляді масиву дозволяє використати усі переваги статичної адресації даних.

Кожен екземпляр класу HashConverter містить регістр згенерованих вхідних ключів, які при застосуванні надають доступ до системи. Цей регістр реалізовано за у вигляді колекції типу ArrayList, що дозволяє виконувати необмежене розширення його розмірів на вимогу. Важливо зауважити, що кожен ключ є одноразовим до застосування, тобто після активації ключ вважається недійсним та вноситься до чорного списку, який також зберігається в межах кожного хеш-перетворювача.

Для створення екземпляру класу HashConverter, клас містить конструктори, які дозволяють створювати об'єкти з різними конфігураційними параметрами. Ключовими атрибутами при ініціалізації екземпляру хеш-перетворювача є цілочисельні параметри sectionsAmount та rate. Параметр sectionsAmount визначає кількість функцій та рівнів перетворення у хеш-перретворювачі, а параметр rate – розрядність функціональних перетворень у його межах. Окрім налаштування базових параметрів, конструктор ініціалізує всі необхідні супутні об’єкти програми відносно означеної топології.

Перед виконанням побудови траєкторії в межах хеш-перетворювача, після ініціалізації об’єкту HashConverter, необхідно виконати розігрів його внутрішнього стану, а саме виконати ініціалізацію функціональних перетворень, задіяних на внутрішніх шарах перетворювача. Це можна виконати в автоматичному режимі за допомогою методу init() – у такому разі буде виконано генерація випадкового вхідного ключа розрядності, відповідної розрядності хеш-перетворювача, після чого буде виконано побудову деякого випадкового маршруту для цього ключа. При цьому вихідні значення для кожного нового вхідного обираються серед множини

вільних до застосування значень довільним чином. Це дозволяє досягти достатнього рівню розкиду значень функціональних перетворень для уникнення детермінованості внутрішнього стану перетворювача та запобіганню перебору зловмисником.

Також, у класі HashConverter передбачена можливість мануального встановлення бажаних значень функцій. У такому разі необхідно виконати імпорт значень за допомогою методу importState(ConversionFunction[] functions), який на вхід приймає масив об’єктів типу ConversionFunction. Оберненою операцією є exportState() – експорт поточних станів функціональних перетворень хеш-перетворювача.

Для побудови траєкторії у межах хеш-перетворювача та генерації нового вхідного ключа, який би надавав доступ до системи при його застосуванні, реалізовано метод buildTrace(). При виклику цього методу, виконується підбір нового унікального вхідного вектору з обсягу визначених значень відповідних функціональних перетворень, після чого виконується побудова низхідної траєкторії по вибраному вхідному вектору з мінімізацією кількості визначень значень функцій. Після виконання методу, на виході користувач отримує згенерований вхідний ключ який надає можливість одноразового доступу до системи.

Виконати перевірку валідності ключа можна лише при його активації за допомогою методу activateKey(DataKey inputKey). У випадку, якщо ключ уже було застосовано раніше або його застосування призводить до утворення некоректного вихідного ключа, виконується викид помилки типу InvalidKeyException, яка є нащадком IllegalArgumentException. Таким чином, InvalidKeyException є помилкою, яка повідомляє про те, що переданий ключ не може бути застосований до поточного стану хеш-перетворювача, оскільки стан останнього не дозволяє його коректне застосування. У разі якщо застосування ключа дозволяє утворити коректний вихідний вектор, він вилучається зі списку активних ключів inputKeys та вноситься

до чорного списку ключів inputKeyBlackList, що запобігає його подальше застосування. Цей процес є незворотнім.

Ще одним важливим методом класу HashConverter є estimateTracesAmount(Duration duration). Він дозволяє виконати оцінку поточного стану хеш-перетворювача шляхом визначення межі його варіативності в залежності від заданого часу роботи програми. На вихід метод повертає об’єкт обгортку для результатів оцінювання перетворювача та містить такі атрибути:

- загальна кількість унікальних вхідних ключів, які можуть бути утворені на поточний момент часу;

- кількість унікальних вхідних ключів, застосованих при переборі загальної кількості можливих шляхів у перетворювачі, які можуть бути застосовані за проміжок часу duration;

- загальна кількість можливих шляхів у перетворювачі, яка може бути побудована за проміжок часу duration;

- загальна кількість унікальних вихідних ключів, які були згенерованих перетворювачем за проміжок часу duration.

Метод виконує оцінку цих властивостей шляхом рекурсивного перебору всіх траєкторій, які можуть бути проведені у хеш-перетворювачі на поточний момент часу за заданий часу роботи.

Висновки до розділу 3

В результаті виконання робіт, які складають зміст третього розділу магістерського дослідження отримані наступні результати:

1. Визначено і обґрунтовано технічне завдання на розробку програмного продукту, який призначено для побудови незворотних хеш-перетворень з програмованими колізіями. Програмний продукт має вирішувати задачі, як побудови хеш-перетворення з програмованими колізіями так і експериментального визначення показників ефективності засобів криптографічно строгої ідентифікації, що використовують такі незворотні хеш-перетворювачі.

2. Виходячи з визначеного технічного завдання обґрунтовано, розроблено та протестовано програмний продукт, здатний синтезувати незворотне хеш-перетворення з програмованими колізіями по заданим базовим параметрів: розрядність сеансового паролю та коду доступу, кількості секцій хеш-перетворення, секретний ключ формування розширення коду доступу, граничний час формування сеансових паролів користувача. Результатом роботи програмного продукту в режимі побудови незворотного хеш-перетворення є набір кодів сеансових паролів користувача, які зберігаються ним в захищеній пам'яті.

РОЗДІЛ 4

ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РОЗРОБЛЕНОГО МЕТОДУ ПОБУДОВИ ХЕШ-ПЕРЕТВОРЕННЯ З ПРОГРАМОВАНИМИ КОЛІЗІЯМИ

4.1 Постановка задачі експериментальних досліджень

Задача експериментального дослідження розробленого методу побудови хеш-перетворення з програмованими колізіями полягає в визначенні основних показників ефективності запропонованого методу синтезу хеш-перетворення з програмованими колізіями. пр. застосуванні синтезованого хеш-перетворювача для криптографічно строгої ідентифікації віддалених користувачів колізії відіграють роль одноразових сеансових паролів ідентифікації користувача системою.

В якості базових критеріїв ефективності, по яким потрібно провести порівняння запропонованого методу та відомих виступають:

- число сеансових паролів користувача, які можуть бути утворені в процесі побудови незворотного хеш-перетворення і які повторень може використовувати користувач для отримання сеансу віддаленої взаємодії з системою до пере налаштування;

- час формування незворотного хеш-перетворення та сеансових паролів користувача на його обчислювальній платформі;

- час ідентифікації віддаленого користувача системою, тобто для представленої розробки – це час виконання хеш-перетворення на обчислювальній платформі системи, обчислення коду доступу; цей параметр ефективності є критичним для широкого кола система колективного доступу, надання інформаційних послуг, тобто для практичних застосувань, для яких кількість користувачів вимірюється десятками і сотнями тисяч;

- об’єм пам'яті, який використовується в хеш-перетворення для реалізації властивості незворотності, основаній на нелінійності булевих функціональних перетворень, які реалізуються табличними булевими перетворювачами; об’єм памяти табличних булевих перетворювачів не є критичним для користувача, але є критичним для системи колективного доступу, яка обслуговує в реальному часі десятки і сотні віддалених користувачів.

4.2 Дослідження залежності кількості сеансових паролів від числа блоків хеш-перетворення

Основним джерелом підвищення ефективності криптографічно строгої ідентифікації в представленій розробці виступає збільшення кількості сеансових паролsd за рахунок зростання кількість ступенів свободи при формуванні траси обчислення коду доступу з кодів сеансових паролів за рахунок розширення множини можливих кодів доступу.

В відомих схемах хеш-перетворювачів з програмованими колізіями формування трас здійснюється за умови, що існує для кожного із користувачів лише один можливий код доступу. В розробленому методі пропонується використовувати для ідентифікації конкретного віддаленого користувача не один, а багато кодів доступу. Це дозволить суттєвим чином збільшити кількість можливих трас обчислення коду доступу.

Проте кількість сеансових паролів суттєвим чином залежить і від об’єму пам'яті блоків хеш-перетворення: чим менше блоків, тим більший їх об’єм і тим більше можна провести трас обчислення з кодів сеансових паролів вихідного коду доступу користувача. Тому, при проведенні експериментальних досліджень враховувалися всі можливі значення кількості блоків, які, на практиці являють собою ступені двійки. При цьому, вибір мінімальної кількості блоків диктується

допустимим об’ємом пам'яті для зберігання таблиць хеш-перетворень на обчислювальній платформі системі колективного доступу.

Крім того, в ході експериментів досліджувалась кількість можливих паролів для різних значень їх розрядності n : від 128 до 512. Зокрема, результати досліджень залежності кількості згенерованих сеансових паролів при відсутності часових обмежень для n=128 і для різної кількості блоків функціональних перетворень наведено на рис.4.1.

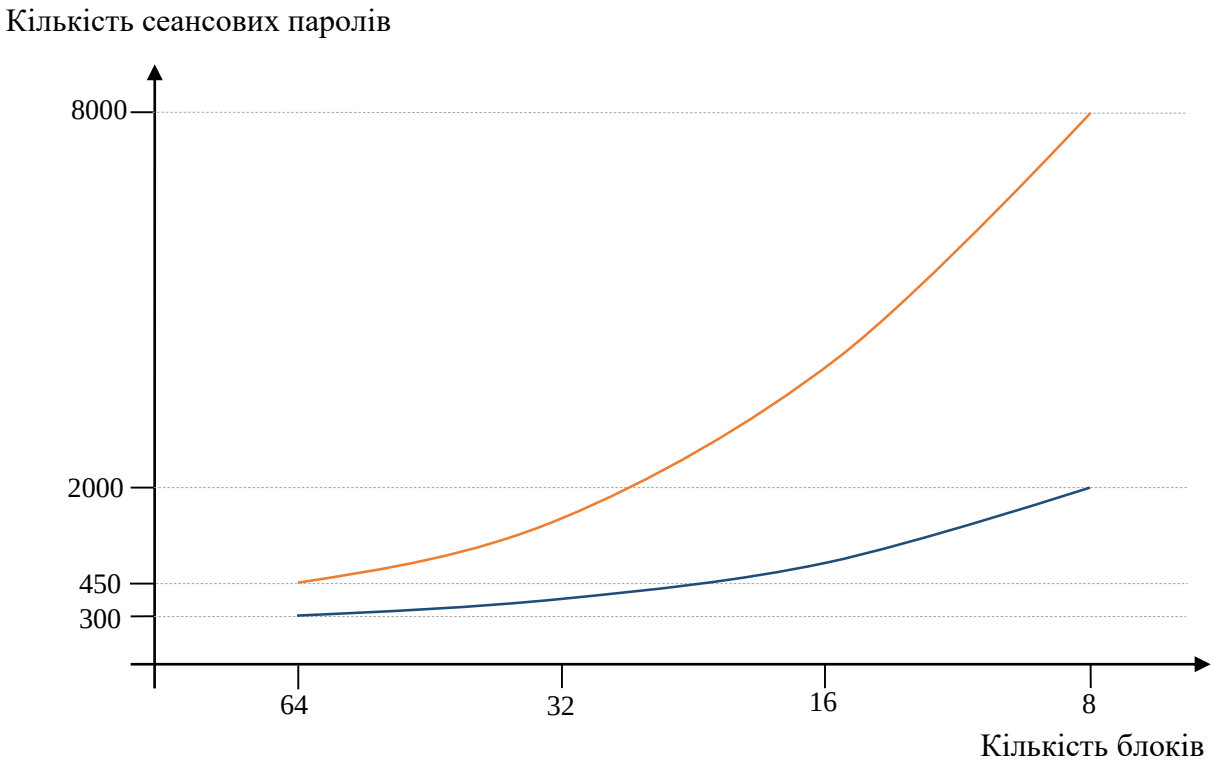

Рис. 4.1. Графік залежності кількості ключів від кількості блоків при розрядності n = 128

Синя крива на графіку, представленому на рис.4.1 показує залежність максимальної кількості сеансових паролів для відомих схем побудови хеш-адресації без колізій, тобто за умови, що код доступу для кожного із віддалених користувачів системи колективного доступу фіксований.

Зокрема, ця кількість мінімальна при числі блоків 64, при цьому об'єм пам'яті блоків булевого функціонального перетворення мінімальний і, відповідно, кількість

сеансових паролів близька до 300. При використання мінімально-доцільної кількості блоків функціональних перетворень – 8 число сеансових паролів, за рахунок збільшення витрат ресурсів пам'яті кількість сеансових паролів, що можуть бути згенеровані зростає до 2-х тисяч.

При використанні розробленого методу в рамках якого пропонується використовувати для ідентифікації конкретного віддаленого користувача не один, а 2^{n-1} кодів доступу кількість можливих трас обчислення коду доступу і, відповідно, число отриманих сеансових паролів помітно зростає. Відповідний графік залежності число сформованих сеансових паролів показано на рис.4.1 лінією червого кольору. За рахунок збільшення числа ступенів свободи обчислювачі хеш-сигнатури зявилась можливість провести більшу кількість трас обчислення коду доступу користувача при різній кількості блоків функціонального перетворення.

Аналіз представленого на рис.4.1 графіку, побудованого на основі отриманих експериментальним щляхом результатів дозволяє зробити такі висновки. При максимально-доцільному числі блоків булевого функціонального перетворення 64, кількість сеансових паролів за рахунок можливості використання багатьох кодів доступу користувача зріс до 450, тобто в півтора рази в порівнянні з відомими схемами хеш-перетворювачів з програмованими колізіями. При використання мінімально-доцільної кількості блоків функціональних перетворень – 8 число сеансових паролів за рахунок можливості використання багатьох кодів доступу користувача кількість сеансових паролів, що можуть бути згенеровані за запропонованим методом збільшується, за експериментальними даними з 2-х до до 8-ми тисяч, тобто в чотири рази.

Таким чином, можна констатувати, що застосування запропонованого методу синтезу хеш-перетворення з програмованими колізіями і 2^{n-1} кодами доступу для кожного з зареєстрованих користувачів дозволяє значно збільшити число колізій, або, що те ж саме, число сеансових паролів віддаленого користувача. Причому, цілком очевидним є те, що ефективність запропонованого синтезу хеш-

перетворення з програмованими колізіями і 2^{n-1} кодами доступу для кожного з зареєстрованих користувачів зростає зі зменшенням числа блоків функціональних перетворень, а бо, що є тим самим, зі зростанням об’єму блока функціонального булевого перетворення. Цей ефект може бути пояснено тим, що збільшення кількості ступенів сподоби при проектуванні хеш-перетворювача більш ефективно використовується при великих об’ємах пам'яті хеш-перетворення.

На рис.4.2. приведено графіки залежності кількості згенерованих сеансових паролів при відсутності часових обмежень для n=256 і для різної кількості блоків функціональних перетворень. Зі збільшенням розрядності ключів збільшується до 16-ти значення мінімально-доцільної кількості блоків функціонального перетворення.

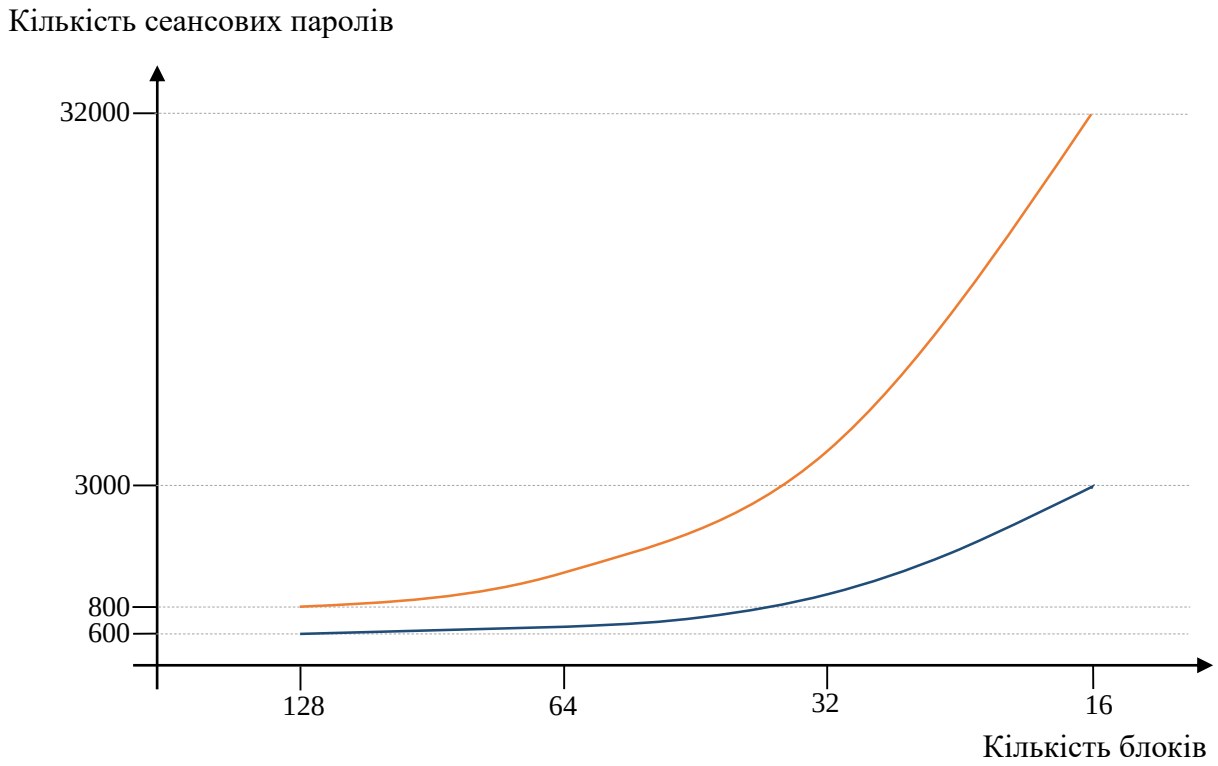

Рис. 4.2. Графік залежності кількості сеансових паролів від числа блоків при розрядності n = 256

Синя крива на графіку, показаному на рис.4.2 відображає залежність максимальної кількості сеансових паролів за умови, що код доступу для кожного із віддалених користувачів системи колективного доступу фіксований. Таке рішення характерне для відомих методів побудови незворотних хеш-перетворень з програмованими колізіями, орієнтованих на використання для криптографічно строгої ідентифікації віддалених учасників інформаційної взаємодії.

Цілком очевидно, що при числі блоків функціонального перетворення, що дорівнює 128 об'єм пам'яті блоків булевого функціонального перетворення мінімальний і, відповідно, кількість сеансових паролів також мінімальні. За експериментальними даними, вона становить близько 600. При використання мінімально-доцільної для $n = 256$ кількості блоків функціональних перетворень – 16 число сеансових паролів, за рахунок збільшення витрат ресурсів пам'яті кількість експериментально сформованих сеансових паролів зростає до 3-х тисяч.

При використанні розробленого методу в рамках якого пропонується використовувати для ідентифікації конкретного віддаленого користувача не один, а 2^{n-1} кодів доступу кількість можливих трас обчислення коду доступу і, відповідно, число отриманих сеансових паролів радикально зростає, особливо при великих об’ємах блоків табличних функціональних перетворень. Отриманий експериментальним шляхом графік залежності кількості отриманих при застосуванні запропонованого методу сеансових паролів показано на рис.4.2 лінією червого кольору. За рахунок збільшення числа ступенів свободи вибору траси обчислення коду доступу експериментально доведено суттєве зростання кількості сеансових паролів при всіх різній значеннях кількості блоків табличного функціонального перетворення.

Аналіз представленого на рис.4.2 графіку, побудованого на основі отриманих експериментальним щляхом результатів, доводить, що при максимально-доцільному числі блоків булевого функціонального перетворення 128, кількість сеансових паролів за рахунок можливості використання множини кодів доступу

користувача збільшився із 600 до 800, тобто в 1.33 рази в порівнянні з відомими схемами хеш-перетворювачів з програмованими колізіями і фіксованим одним кодом доступу для кожного з віддалених користувачів. При використання мінімально-доцільної кількості для n = 256 блоків функціональних перетворень – 16 число сеансових паролів за рахунок можливості використання множини із 2^{n-1} кодів доступу користувача, число сеансових паролів, що можуть бути отримані при використанні розробленого методу збільшується, за експериментальними даними з 3-х до до 32-х тисяч, тобто практично на порядок. Це свідчить про те, що при збільшенні розрядності сеансових паролів користувачів, зростає ефективність запропонованого методу.

Для n = 256 можна відмітити, що застосування розробленого методу побудови незворотних хеш-перетворень з програмованими колізіями і множиною кодів доступу за результатами проведених експериментів забезпечує помітне збільшення числа сеансових паролів в порівнянні з одним кодом доступу для кожного користувача.

Аналіз графіків, наведених на рис.4.2 свідчить про те, що ефективність розробленого методу побудови хеш-перетворення з програмованими колізіями і багатьма кодами доступу для кожного з зареєстрованих користувачів зростає зі зростанням об’єму блока функціонального булевого перетворення.

Результати аналогічних досліджень для n = 512 представлені на рис.4.3. Ці результати відображають залежність числа реально сформованих сеансових паролів від кількості блоків функціональних перетворень при використанні одного та багатьох кодів доступу. Цілком очевидно, що для n = 512 значення мінімально-доцільної кількості блоків функціонального перетворення дорівнює 16-ти, а максимально-доцільне – 256.

На графіку, представленому на рис. 4.3 кривою синього кольору відображено залежність максимальної кількості сеансових паролів отриманих при використанні одного коду доступу для кожного із віддалених користувачів системи віддаленої

інформаційної взаємодії. З наведеного графіку видно, що максимально-доцільному числі блоків функціонального перетворення їх об'єм мінімальний і, відповідно, кількість сеансових паролів також мінімальні. За експериментальними даними, вона для $n = 512$ становить 1250. При використання мінімально-доцільного для $n = 512$ числа блоків булевих функціональних перетворень – 32, кількість сеансових паролів користувача, за рахунок використання більшого об’єму пам'яті число отриманих сеансових паролів збільшується до 12500.

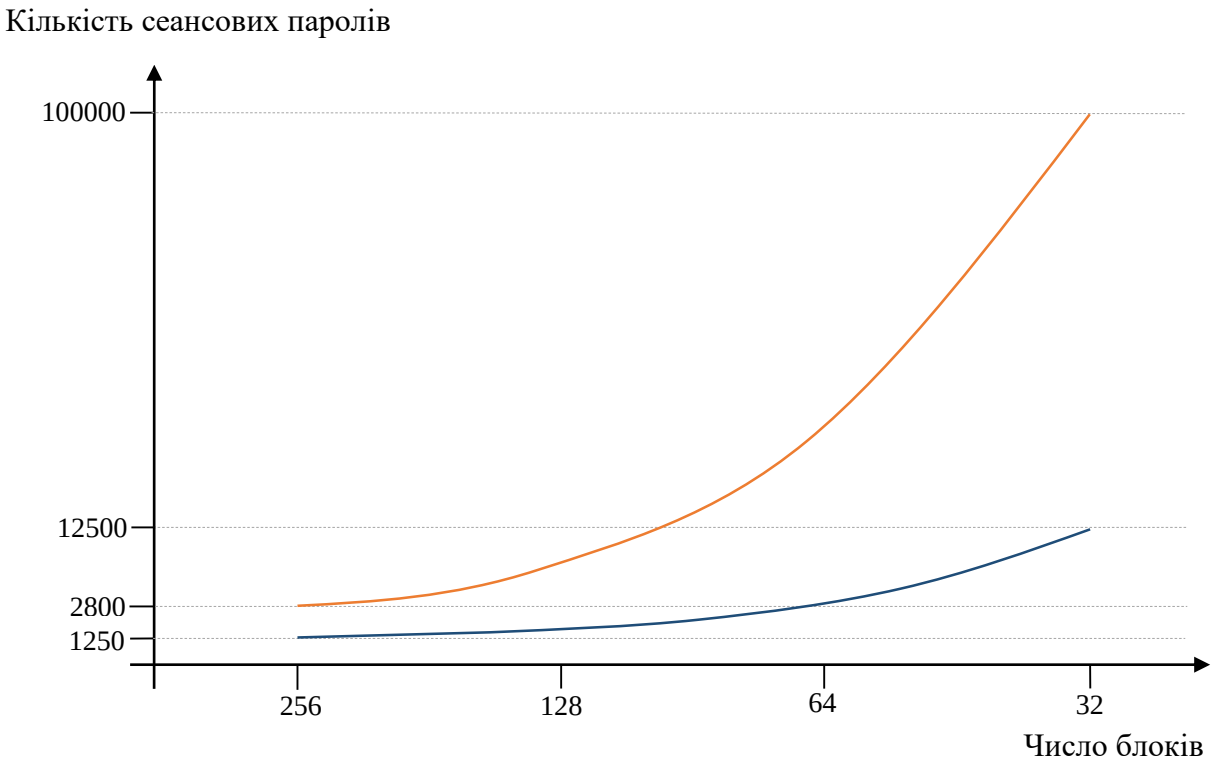

Рис. 4.3. Графік залежності кількості сеансових паролів від кількості блоків при розрядності $n = 512$

При використанні замість одного фіксованого коду доступу користувача 2^{n-1} кодів доступу кількість можливих трас обчислення коду доступу, що реалізовано в рамках запропонованого методу, кількість експериментально отриманих сеансових паролів суттєвим чином зростає, причому це зростання збільшується зі ростом об’єму блоків табличних функціональних перетворень. Отриманий експериментальним шляхом графік залежності числа отриманих сеансових паролів

при використанні розробленого запропонованого методу побудови хеш-перетворення демонструється на рис.4.3 червоною кривою. За рахунок збільшення числа ступенів свободи вибору траси обчислення коду доступу експериментально показано багатократне зростання кількості сеансових паролів при всіх різній значеннях кількості блоків табличного функціонального перетворення.

З наведених на рис.4.3 залежностей очевидно, що при максимально-доцільному числі табличних функціональних перетворювачів , кількість сеансових паролів за рахунок запропонованого підходу зріс з 1250 до 2800, тобто в 2.44 рази в порівнянні з відомими схемами хеш-перетворювачів з програмованими колізіями і фіксованим єдиним кодом доступу. При використання мінімально-доцільної кількості для n = 512 кількості табличних функціональних перетворювачів, кількість сеансових паролів при використанні 2^{n-1} можливих кодів доступу, зросло за результатами експериментів з 12500 до 100000, тобто практично в 8 раз. Для n = 256 цілком очевидним є те, що застосування множинного коду доступу забезпечує зростання числа сеансових паролів в порівнянні з одним кодом доступу для кожного користувача.

Аналіз графіків, наведених на рис.4.3 свідчить про те, що ефективність запропонованого підходу, що базується на використання великої кількості кодів доступу для кожного з користувачів зростає зі зростанням об’єму табличного функціонального перетворювача.

4.3 Дослідження залежності часу формування хеш-перетворення від числа блоків хеш-перетворення

Як зазначалося вище, важливим критерієм ефективності засобів криптографічно строгої ідентифікації є час формування незворотного перетворення. Використання саме часу для оцінки трудоємкості побудови незворотного перетворення зумовлене тим, що ця операція здійснюється на обчислювальній

платформі користувача, в якості якої найбільш часто виступає персональний комп’ютер.

Відповідно, для отримання експериментальних результатів. які дозволяють об’єктивно оцінити час формування незворотного хеш-перетворення з програмованими колізіями за запропонованим методом, використовувалась обчислювальна платформа у вигляді персонального комп’ютера.

На рис. 4.4. наведено графіки залежності кількості сеансових паролів від часу роботи програми при їх розрядності $n = 128$.

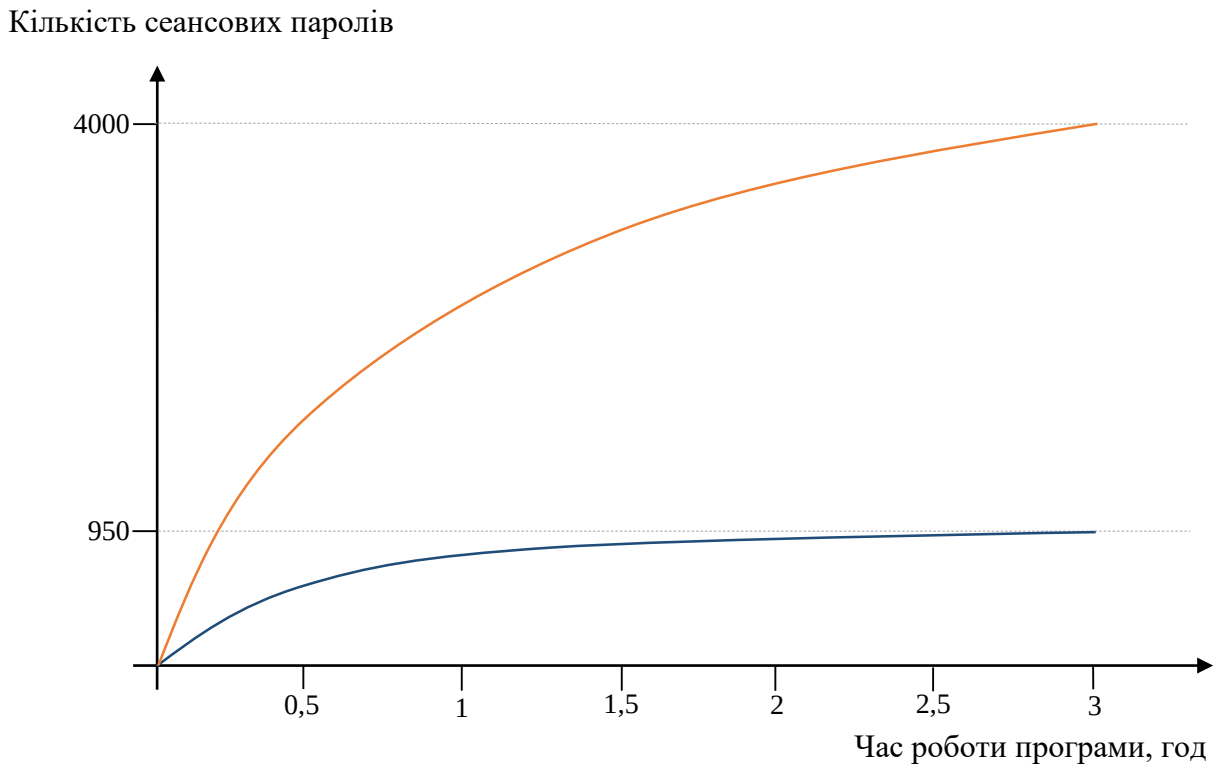

Рис. 4.4. Графік залежності кількості сеансових паролів від часу роботи програми при розрядності $n = 128$

На графіку, показаному на рис.4.4. крива синього кольору показує отриману експериментальним шляхом залежність кількості отриманих сеансових паролів користувачів, за умови, що кожен з останніх має лише один фіксований код доступу. Характер графіку показує на очевидний факт, який полягає в тому, що трасування

обчислення кожного наступного сеансового паролю потребує більше часу, ніж попереднього. Лінією червоно кольору на графіку на рис.4.4. представлена отримана експериментальним шляхом залежність кількості отриманих сеансових паролів користувачів, за умови, що кожен з останніх має 2^{n-1} кодів доступу. Аналіз наведених на рис.4.4 залежностей показує, що використання розробленого методу, що базується на застосування множини з 2^{n-1} кодів доступу дозволяє при будь-якому фіксовано часі роботи програми отримати суттєво більшу кількість сеансових паролів в порівнянні з технологіями, що використовують один фіксований код доступу. Наприклад, за експериментальними даними, при роботі програми 10 хвилин, використання запропонованого методу дозволяє сформувати близько 950 сеансових паролів, в той час, як при застосуванні методів з фіксованим кодом доступу за цей же проміжок часу формується близько 310 сеансових паролів, тобто практично втричі менше.

Аналогічні експериментальні дослідження проводилися для $n = 256$. Отримані результати представлені у вигляді графіків на рис.4.5.

На графіках, показаних на рис.4.5 синім кольором позначена крива залежності залежність кількості отриманих сеансових паролів користувачів, за умови, що кожен з останніх має лише один фіксований код доступу. Червона крива на рис. 4.5 відображає отриману експериментально залежність кількості сформованих сеансових паролів користувачів, за умови, що кожен з останніх має 2^{n-1} кодів доступу. Аналіз представлених на рис.4.5 залежностей для $n = 256$ показує, що використання розробленого методу, що базується на застосування множини з 2^{n-1} кодів доступу дозволяє при будь-якому фіксовано часі роботи програми отримати значно більшу кількість сеансових паролів в порівнянні з технологіями, що мають за основу використання одного фіксованого коду доступу.

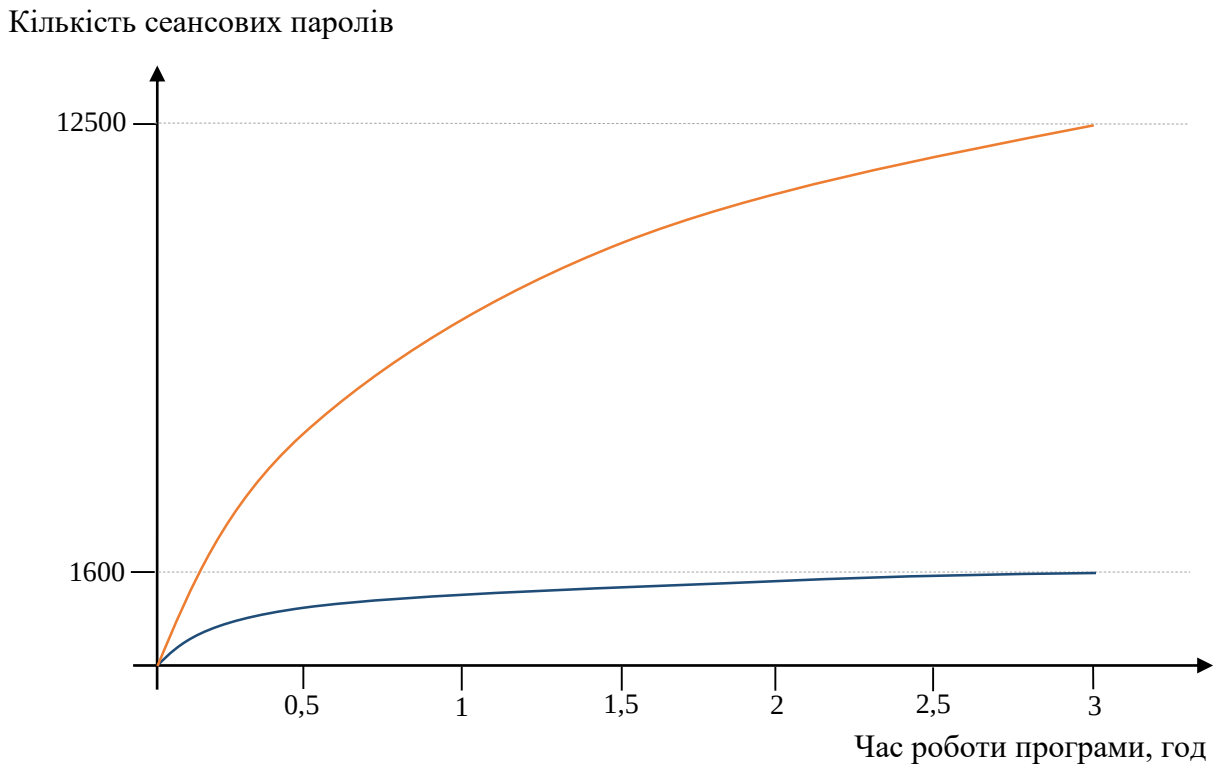

Рис. 4.5. Графік залежності кількості сеансових паролів від часу роботи програми при розрядності n = 256

Подібні експериментальні дослідження проводилися для n = 512. Отримані результати представлені у вигляді графіків на рис.4.6.

Як і на попередніх заледностях, на графіках представлених на рис.4.6 лінією синього кольору позначена залежність кількості отриманих сеансових паролів користувачів, за умови, що кожен з останніх має лише один фіксований код доступу, а лінією червоно кольору - залежність кількості сформованих сеансових паролів користувачів, за умови, що кожен з останніх має 2^{n-1} кодів доступу. Аналіз представлених на рис.4.6 залежностей для розрядності сеансових паролів n = 512 однозначно свідчить про те, що використання розробленого методу, який має за основу застосування множини з 2^{n-1} можливих кодів доступу дозволяє при будь-якому часі роботи програми отримати в 2-3 рази більшу кількість сеансових паролів в порівнянні з відомими методами побудови хеш-перетворень з програмованими колізіями, що базуються на використанні єдиного коду доступу користувача.

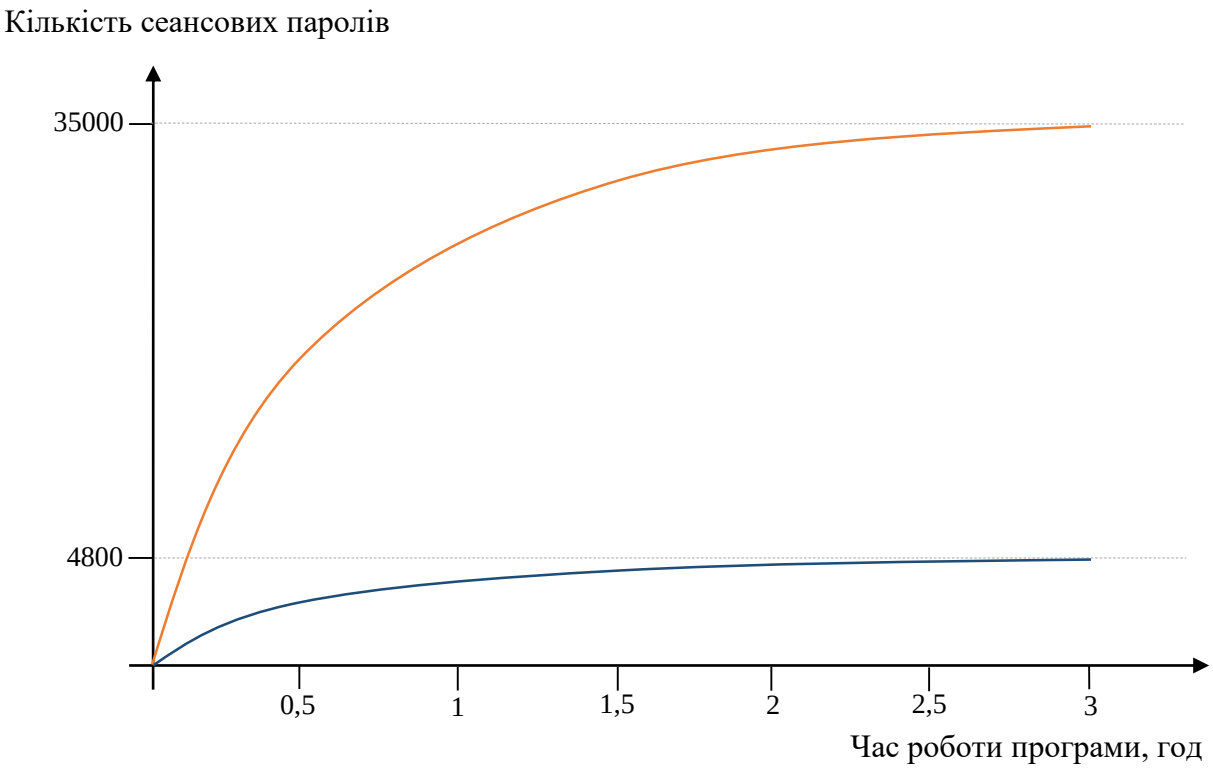

Рис. 4.6. Графік залежності кількості сеансових паролів від часу роботи програми при розрядності $n = 512$

Разом з тим, детальний аналіз результатів отриманих експериментальних результатів не виявив чіткої залежності різниці в часі формування визначеної кількості сеансових паролів від їх розрядності.

4.4 Дослідження залежності потрібних ресурсів пам'яті від числа блоків хеш-перетворення

Як зазначалося вище, для широкого класу практичних систем віддаленої взаємодії коли одна із сторін взаємодіє з великою кількістю віддалених користувачів, важливим критерієм ефективності засобів криптографічно строгої ідентифікації виступає об’єм ресурсів пам'яті для реалізації незворотних перетворень. Стосовно хеш-перетворень з програмованими колізіями ресурси пам'яті оцінюються через об’єм пам'яті, який використовується в хеш-перетворення

для реалізації властивості незворотності, основаній на нелінійності булевих функціональних перетворень, які реалізуються табличними булевими перетворювачами.

В рамках проведених експериментальних досліджень визначався об’єм памяті для реалізації табличних функціональних перетворень, які використовуються в структурі хеш-перетворювача з програмованими колізіями.

Отримані в результаті цих досліджень залежності показані на рис. 4.7 для розрядності сеансових паролів n = 128.

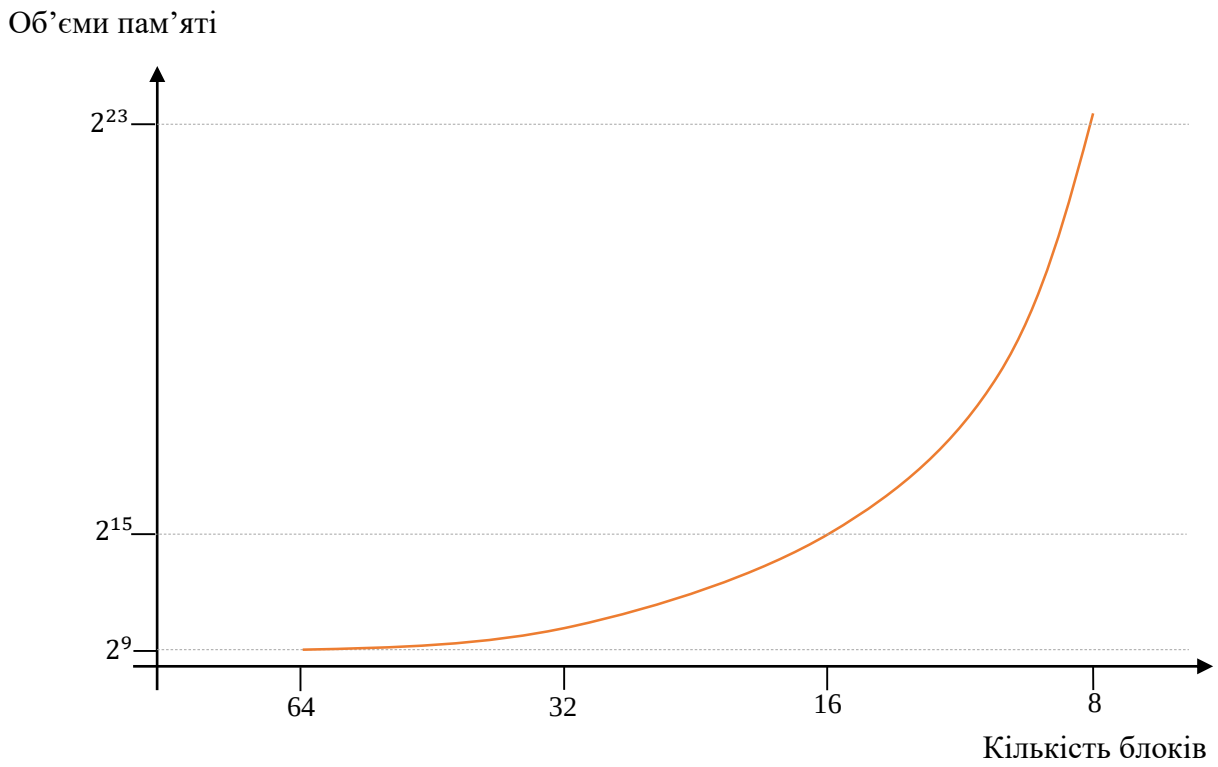

Рис. 4.7. Графік залежності об’ємів використання пам’яті від розміру одного блоку при розрядності n = 128

Аналіз отриманої залежності цілком очевидно вказує на екпоненційний її характер. Отриманий графік вкупі з залежностями, показаними на рис. 4.1 можуть бути використанні для оптимізації структури хеш-перетворювача з програмованими колізіями на основі множини кодів доступу. Оптимізація полягає

в виборі найбільш доцільного з точки зору кількості отриманих сеансових паролів користувача та задіяних для цього ресурсів пам'яті числа блоків табличних функціональних перетворень.

На рис.4.8 і рис.4.9 наведені аналогічні залежності для $n = 256$ і $n = 512$ відповідно.

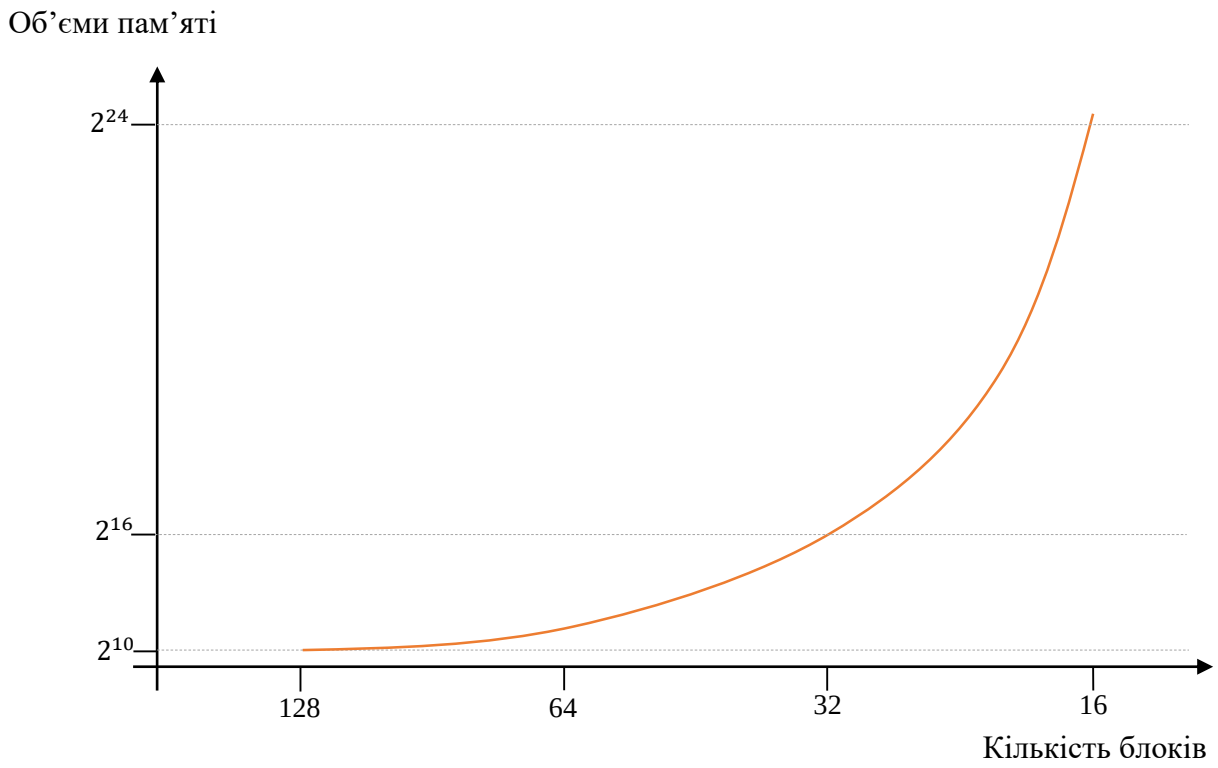

Рис. 4.8. Графік залежності об’ємів використання пам’яті від розміру одного блоку при розрядності $n = 256$

На практиці задача оптимального вибору кількості блоків табличного функціонального перетворення вирішується виходячи з заданого з умов практичного застосування граничного об’єму пам'яті незворотних перетворень всіх користувачів системи колективного доступу. Відповідно, визначається граничний об’єм пам'яті, що може бути використаний системою для реалізації незворотного перетворення одного користувача. По наведеним на рис.4.7 – 4.9 графікам, в залежності від розрядності сеансового пароля визначається кількість блоків

табличного функціонального перетворення, сумарний об’єм пам'яті яких менший за визначений ліміт.

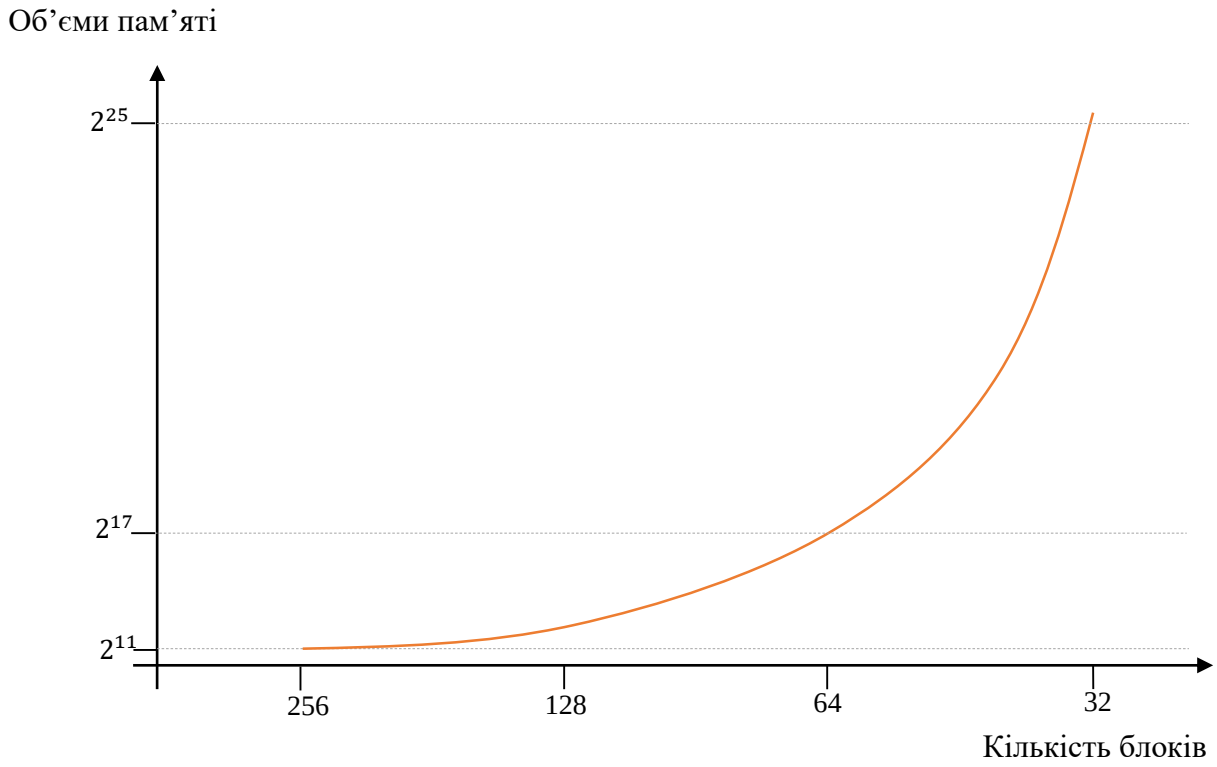

Рис. 4.9. Графік залежності об’ємів використання пам’яті від розміру одного блоку при розрядності n = 512

Висновки до розділу 4

В результаті проведених з використанням розроблених програмних засобів експериментальних досліджень, направлених на визначення характеристик ефективності запропонованого методу побудови незворотного хеш-перетворення з програмованими колізіями та порівняння їх з аналогічними характеристиками ефективності відомих методів можу зробити наступні висновки:

1. Для отримання достовірних та коректних оцінок ефективності розробленого методу застосовано методики статистичного моделювання зі статистичним відхиленням отриманих результатів не більше 3%. Оцінювалися параметри ефективності запропонованого методу, який має за основу застосування множини з 2^{n-1} можливих кодів доступу та відомих методів побудови хеш-перетворень з програмованими колізіями, що базуються на використанні єдиного коду доступу користувача.

2. Експериментально доведено, що запропонований метод побудови хеш-перетворень з програмованими колізіями і 2^{n-1} кодами доступу для кожного з зареєстрованих користувачів дозволяє значно збільшити число колізій, або, що те ж саме, число сеансових паролів віддаленого користувача. Причому, цілком очевидним є те, що ефективність запропонованого синтезу хеш-перетворення з програмованими колізіями і 2^{n-1} кодами доступу для кожного з зареєстрованих користувачів зростає зі зменшенням числа блоків функціональних перетворень, а бо, що є тим самим, зі зростанням об’єму блока функціонального булевого перетворення. Цей ефект може бути пояснено тим, що збільшення кількості ступенів сподоби при проектуванні хеш-перетворювача більш ефективно використовується при великих об’ємах пам'яті хеш-перетворення.

3. Проведеними експериментальними дослідженніми доведено, що використання розробленого методу автоматизованого синтезу хеш-перетворень з програмованими колізіями, що базується на застосування множини з 2^{n-1} кодів

доступу дозволяє при будь-якому фіксованому часі роботи програми отримати суттєво більшу кількість сеансових паролів в порівнянні з відомими технологіями побудови незворотних хеш-переторень, що використовують один фіксований код доступу для кожного учасника віддаленої взаємодії.

ВИСНОВКИ

В результаті виконання магістерської дисертації, метою якої є підвищення ефективності криптографічного строгої ідентифікації на основі хеш-перетворень з програмованими колізіями за рахунок збільшення кількості сеансових паролів доступу, отримані певні результати, які можуть бути сформульовані наступним чином:

1. Виконано аналіз сучасного стану використання механізмів ідентифікації учасників віддаленої інформаційної взаємодії, який показав, що розширення сфери використання віддалених форм інформаційної взаємодії та комерціалізація віддаленого надання інформаційних послуг диктують необхідність радикального підвищення рівня захищеності процесів взаємної ідентифікації учасників віддаленої взаємодії, що може бути досягнуто тільки в рамках використання криптографічно строгих різновидів механізмів ідентифікації.

2. Огляд, з позицій сучасних вимог, існуючих методів та засобів криптографічно строгої ідентифікації учасників віддаленої інформаційної взаємодії показав, що ті їх різновиди, які базуються на незворотних перетвореннях модулярної арифметики мають недостатню швидкодію. Більш швидкісні методи криптографічно строгої ідентифікації, що базуються на використанні незворотних перетворень булевої алгебри мають обмежену кількість сеансових паролів.

3. В якості найбільш перспективного шляху підвищення ефективності криптографічно строгої ідентифікації на основі хеш-перетворень з програмованими колізіями є пошук можливостей радикального збільшення числа сеансових паролів.

4. Одним із важливих напрямків підвищення ефективності криптографічно строгої ідентифікації на основі хеш-перетворень з програмованими колізіями є збільшення кількості сеансових паролів. Реалізація такого збільшення може досягнута за рахунок збільшення витрат ресурсів пам'яті або за рахунок надання

більшої степені свободи процедурі трасування обчислень. Другий шлях є більш перспективним в силу того, що він потребує менше ресурсів.

5. Теоретично обґрунтовано, розроблено та досліджено метод побудови хеш-перетворення з програмованими колізіями для криптографічно строгої ідентифікації, який відрізняється тим, що ідентифікатором учасника віддаленої взаємодії виступає не фіксований код, а певна сукупність кодів, біти яких пов’язані певними криптографічними відношеннями, які залежать від секретного ключа, за рахунок чого збільшується кількість колізій хеш-перетворення і, тим самим на значною міною знімається обмеження на кількість можливих сеансових ключів криптографічно строгої ідентифікації.

6. Визначено і обґрунтовано технічне завдання на розробку програмного продукту, який призначено для побудови незворотних хеш-перетворень з програмованими колізіями. Програмний продукт має вирішувати задачі, як побудови хеш-перетворення з програмованими колізіями так і експериментального визначення показників ефективності засобів криптографічно строгої ідентифікації, що використовують такі незворотні хеш-перетворювачі.

7. Виходячи з визначеного технічного завдання обґрунтовано, розроблено та протестовано програмний продукт, здатний синтезувати незворотне хеш-перетворення з програмованими колізіями по заданим базовим параметрів: розрядність сеансового паролю та коду доступу, кількості секцій хеш-перетворення, секретний ключ формування розширення коду доступу, граничний час формування сеансових паролів користувача. Результатом роботи програмного продукту в режимі побудови незворотного хеш-перетворення є набір кодів сеансових паролів користувача, які зберігаються ним в захищеній пам'яті.

8. Для отримання достовірних та коректних оцінок ефективності розробленого методу застосовано методики статистичного моделювання зі статистичним відхиленням отриманих результатів не більше 3%. Оцінювалися параметри ефективності запропонованого методу, який має за основу застосування

множини з 2^{n-1} можливих кодів доступу та відомих методів побудови хеш-перетворень з програмованими колізіями, що базуються на використанні єдиного коду доступу користувача.

9. Експериментально доведено, що запропонований метод побудови хеш-перетворень з програмованими колізіями і 2^{n-1} кодами доступу для кожного з зареєстрованих користувачів дозволяє значно збільшити число колізій, або, що те ж саме, число сеансових паролів віддаленого користувача. Причому, цілком очевидним є те, що ефективність запропонованого синтезу хеш-перетворення з програмованими колізіями і 2^{n-1} кодами доступу для кожного з зареєстрованих користувачів зростає зі зменшенням числа блоків функціональних перетворень, а бо, що є тим самим, зі зростанням об’єму блока функціонального булевого перетворення. Цей ефект може бути пояснено тим, що збільшення кількості ступенів сподоби при проектуванні хеш-перетворювача більш ефективно використовується при великих об’ємах пам'яті хеш-перетворення.

10. Проведеними експериментальними дослідженніми доведено, що використання розробленого методу автоматизованого синтезу хеш-перетворень з програмованими колізіями, що базується на застосування множини з 2^{n-1} кодів доступу дозволяє при будь-якому фіксованому часі роботи програми отримати суттєво більшу кількість сеансових паролів в порівнянні з відомими технологіями побудови незворотних хеш-переторень, що використовують один фіксований код доступу для кожного учасника віддаленої взаємодії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Steblevets T. Cryptographically strict user identification based on hash transformation with programmed collision / Tetyana Steblevets, Ihor Daiko Oleksandr Markovskyi // Proceeding of the International Conference on Security, Fault Tolerance, Intelligence" (ICSFTI-2021).- 2021.- 12 May, Kyiv.- P.12-20.

2. I. Daiko, V. Selivanov, M. Chernyshevych, O. Markovskyi. Zero-knowledge identification of remote users by utilization of pseudorandom sequences // Information, Computing and Intelligent systems.- № 3.-2022. – P. 42-48.

3. Schneier B. Applied Cryptography. Protocols. Algorithms and Source codes in C. Ed.John Wiley, 1996 - 758 p.

4. Schneier B. "A primer on authentication and digital signatures". Computer Security Journal, v.10, n.2,1994, pp.38-40.

5. Menezes Alfred, Handbook of Applied Cryptography. / Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone // CRC Press. – 2001. – 780 p.

6. Kittichokenai K., Care G. Secret Key-based Identification and Authentication with a Privacy Constraint. IEEE Trans. Inf. Theory.- Vol. 62.- 2016.- № 11.- P. 6189-6203.

7. Aquilar C., Gaborit P., Schrec J. A new zero-knowledge code based identification scheme with reduced communication. IEEE Information Theory Workshop 16-20 Oct. 2011 .-2011.-P.648-652. DOI: 10.1109/ITW.2011.6089577

8. Asimi Y., Amghar A., Asimi A., Sadgi Y. Strong zero-knowledge authentication based on the session keys (SASK). International Journal of Network Security & Its Applications (IJNSA).-2015.- Vol.7, - No.1,- P.51-66.

9. Bardis N., Doukas N., Markovskyi O. Fast subscriber identification based on the zero knowledge principle for multimedia content distribution. International Journal of Multimedia Intelligence and Security.-2010.- No.4,- P. 363-377.

10. Bardis N., Doukas N. Markovskyi O. A Method for strict remote user authentication using non-revesible Galois field transformations // MATEC Web of

Conferences 125, 05017 (2017)- P.243-249. DOI: 10.1051/matecconf/20171250 CSCC 2017 5017

11. Bardis N.G., Markovskyi O.P., Doukas N., Drigas F. Fast implementation zero knowledge identification schemes using the Galois Fields arithmetic, Proceeding of IX. International Symposium IEEE on Telecommunications - BIHTEL-2012, October 25-27, 2012, Sarajevo, Bosnia and Herzegovina. – P.54-62.

12. Bardis N.G., Doucas N., Markovskyi O. Zero-Knowledge Identification Method Based on Block Ciphers. Proceeding of 2017 International Conference on Control, Artificial Intelligence, Robotic & Optimization (ICCAIRO). May 2017. DOI: 10.1109/ICCARO.2017.63.

13. Stavroulakis P., Markovskyi O., Bardis N., Doucas N. Efficient Zero Knowledge identification based on one way Boolean transformations. IEEE Globecom Workshops. 5-9 Dec. 2011.- 2011.-P.275-280. DOI: 10.1109/GLOCOMW.2011.6162452.

14. Schnorr C.P. Method for Identification Subscribers and for Generating and Verificating Electronic Signatures in data Exchange System.- US Patent #4995,083.19-1991.

15. Schnorr C.P. “Efficient signature generation for smart cards”. Journal of Cryptology, v.4, n.3, 1991, pp.161-174.

16. Guillou L.C., Quisquater J.J. A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memore. Proceeding of Advances of Cryptology – Eurocrypt-88.- Springer-Verlag.- 1988.- P.123-128.

17. Mu Han, Zhikun Yin, Pengzhou Chen, Xing Zhang, Shidian MA. Zero-knowledge identity authentication for internet of vehicles: Improvement and application. PLoS ONE.- 2020.- Vol.15 –No.9.-P.217-247. DOI.ORG/10.137/journal.pone.0239043

18. Feige U.,Fiat A.,Shamir A. Zero knowledge proofs of identity. Journal of Cryptology, - 1988.- Vol.1.- №.2 . – P.77-94.

19. Fiat A., Shamir A. How to prove yourself: practical solutions to identification and signature problems // Proceeding of Workshop ”Crypto’86”.- Santa Barbara, USA, August 11-15, 1986.- Berlin: Springer-Verlag.- LNCS-263 – 1986.- P.186-194.

20. Shamir A. On the generation of cryptographically strong pseudorandom sequenses. ACM Transaction on Computer Systems. – 1983.- Vol.1.- № 1.- P.38-44. DOI: 10.1145/357353.357357

21. Biham E., Shamir A. Differential cryptanalysis of DES-like cryptosystems.// Journal of Cryptology.- 1991.- Vol. 4 -№ 1.- P.3-72

22. Haller N., Atkinson R. On Internet Authentication. RFC 1704, Bell Communications Research and Naval Research Laboratory, October 1994.

23. Stern J. Designing identification schemes with keys of short size // Proceeding of Workshop ”Crypto’94”.- Santa Barbara, USA, August 21-25, 1994.- Berlin: Springer-Verlag.- LNCS-839 – 1994.- P.164-173.

24. Conti, M., Dragoni, N., Lesyk, V. A Survey of Man in the Middle Attacks. IEEE Communications Surveys and Tutorials.- 2016. -Vol.18.- № 3.- P.2027-2051. DOI:10.1109/COMST.2016.2548426

25. Lamport L.Password Authentication with Insecure Communication. Communications of the ACM. -1981.- Vol.24.- № 11,- P.770-772

26. Burmester M.V.D. A remark on efficiency of identifications schemes // Proceeding of Workshop ”Eurocrypt’90”.- Aarhus, Denmark, May 21-24, 1990.- Berlin: Springer-Verlag.- LNCS-473 – 1991.- P.493-495.

27. Ohta K. “Identity-based authentication schemes using the RSA cryptosystem”. Transactions of the institute of electronic, Information and communication engineers, v. J72-D-II, n.8, 1989, pp.612-620.

28. Okamoto T. Provably secure and practical identification schemes and corresponding signature schemes // Proceeding of Workshop ”Crypto’92”.- Santa Barbara, USA, August 16-20, 1992.- Berlin: Springer-Verlag.- LNCS-740 – 1992.- P.31-53.

29. Aziz A., Daffie W. “Privacy and authentication foe wireless local area networks” IEEE Personal Communications, v.1, n.1,1994, pp.25-31.

30. San Ling, Khoa Nguyen, Damien Stehle, Huaxiong Wang. Improved Zero-Knowledge Proofs of Knowledge for the ISIS Problem, and Applications. International Workshop on Public Key Cryptography PKC 2013. – P.107-124. DOI: 10.1007/978-3-642-36362-7_8

31. Soo Yun Hwang, Gi Yoon Parkm Dae Ho Kim, Kyong Son Jhang. Efficient Implementation of a Pseudorandom Sequence Generator for High-Speed Data Communications. ETRI Journal,-2010.- Vol.32,- № 2,-P.222-229.

32. Wang X., Wu Y., Caron B. Transmitted identification using embedded pseudorandom sequences. IEEE Transaction on Broadcasting.-2004. Vol.50. - № 3.- P.244-252. DOI: 10.1109/TBC.2004.834027.

33. Wang W. On the Relation Between Identifiability , differential Privacy and Mutual Informational Privacy / Wang W., Ying I., Zhang J. // IEEE Trans. Inf. Theory.- Vol. 62.- 2016.- № 9.- P. 5018-5029.

34. Knudsen L.R., Meier W.”Cryptoanalysis of an identification scheme based on the permuted perceptron problem”. Advances in Cryptology-EUROCRYPT ’99 Proceedings, Lecture Notes in Computer Science. Spriger-Verlag, 1999, pp.363-374.

35. Knudsen L.N. The block cipher companion / L.R. Knudsen, M.J.B. Robashaw. //Berlin: Springer-Verlag.-2011.

36. Stafford E. Preventing Weak Password Choices // Computers and security.-1992.- 3.- P.46-53.

37. Pieprzyk J., Sadeghiyan B. Design of Hashing Algorithms. LNCS 756. Berlin: Springer, 1993.- 194 p.

38. Bengio S., Brassard G., Desmedt Y.G. Goutier C.,Quisquater J.J. “Secure implementation of identification system”, Jornal of Cryptology, v.4, n.3, 1991, pp.186-192.

39. Blundo C., DeSntis A., Kurosawa K., Ogata W. “On a fallacious bound for authentication codes”. Journal of Cryptology, v.12, n.3, 1999 pp.155-159.

40. Kurosawa K., Yoshida T. “ Strongly universal hasing and identification codes via channels. IEEE Trans. Information theory, v.45, no.6, 1999. pp.2091-2095.

41. Matsumoto T., Imai H. Human identification through insecure channel // Proceeding of Workshop ”Eurocrypt’91”.- Brighton, UK, April 8-11, 1991.- Berlin: Springer-Verlag.- LNCS-547 – 1991.- P.409-421.

42. Pourand G. “A realistic security analysis of identification schemes based on combinatorial problems”. European Transactions on Telecommunications. V.8, n.5. 1997, pp.471-480.

43. Budaghyan J. Construction and Analysis of cryptographic functions. New York,:NY:Springer-Verlag.-2014.- 200 p.

44. Burrows M., Abadi M., Needham R. A logic for authentication // Proceeding of the 12-th ASM Symposium on Operating Systems Principles. N.Y. – 1989.- P.39-48.

45. Camenisch J., “Efficient and Generalized Group Signatures,” Advances in Cryptology-EUROCRYPT ’97 Proceedings, Lecture Notes in Computer Science. Spriger-Verlag, 1997, pp.464-479.

46. Chaum D., Van Heijst E., and Pfitzmann B. “Cryptographicall strong undeniable signatures, unconditionally secure for signer“ , Advances in Cryptology - Crypto ’91, v. 576 of Lecture Notes in Computer Science, 1992. pp. 470-484.

47. Chabaud F., Vaudenay S. Links between differential and linear cryptanalysis.// Proc. of International Conf. Advanced in Cryptology – Asiacrypt’94 Proceeding, LNCS 950 – 1994,- P.356-365.

48. Cusic T.W. On construction balanced correlation immune function, in sequences and their application. // Proceeding of SETA’98-Springer Discrete Mathematics and Theoretical Computer Sciences, 1999-P.184-190.

49. Davies D.W. and Price W.L.,” The Application of Digital Signatures Based on Public-Key Cryptosystems, “Proceedings of the Fifth International Computer Communications Conference, Oct 1980, pp. 525-530.

50. Davio M., Goethals J.-M., Quisquater J.-J. Authentication Procedures // Proceeding of Workshop ”Eurocrypt’82”.- Burg Feuerstein, Germany, 29 March-2 April 1982.- Berlin: Springer-Verlag.- LNCS-149 – 1983.- P.283-288.

51. Dontas K., Sarma J., Srinivasan P., Wechsler H. Fault tolerant hashing and information retrieval using back propagation // Proceeding of the 23-th Annual International Conference on System Sciences,- 1990,- Vol.4,- P.345-352.

52. Fagin R., Nievergelt J., Pippenger N., Strong H.R. Expan¬ding hashing - a fast access method for dynamic files//ASM Trans. Database Syst.- 1979,- Vol.4, № N 3 p.315-344.

53. Forre R. The strict avalanche criterion: spectral properties of Boolean functions and extend definition // Advanced in Cryptology – Crypto’88 Proceeding, Lecture Notes in Computer Sciences, 403 – 1990-P.450-468.

54. Goldreich O. On concurrent identification protocols // Proceeding of Workshop ”Eurocrypt’84”.- Paris, France, April 9-11, 1984.- Berlin: Springer-Verlag.- LNCS-209 – 1984.- P.387-398.

55. Gonnet G.H., Larson P.A. External hashing with limited internal storage //Journal ACM,- 1988,- Vol.35, № 1,- P.161-184.

56. Hauser R.C., Lee E.S. ”Verification and modeling of authentication protocols”. ESORICS’92, Proceedings of the second European symposium on research in computer security. Springer-Verlag,1992, pp.131-154.

57. Hellman M.E. The mathematics of public-key cryptography. Scientific American, v.241 n. 8 1979 . pp. 146-157.

58. Hannesey J.L., Patterson D.A. Computer architecture: quantitative approach. California: Morgab Kaufman Publishers.-1999.- 239 p.

59. ISO DIS 10118 DRAFT, “Information Technology Security Techniques Hash Functions,” International Organazation for Standardization, April 1991.

60. Стеблевець, Т. О. Метод побудови хеш-перетворення з програмованими колізіями та програмні засоби його реалізації : дипломний проєкт ... бакалавра : 123 Комп'ютерна інженерія / Стеблевець Тетяна Олександрівна. - Київ, 2021. - 109 с.

61. Самофалов К.Г., Тубольцев А.А., Ияд Мохд Маджид Ахмад Шахрури. Повышение эффективности идентификации удаленных абонентов многопользовательских систем // Проблеми інформатизації та управління. Збірник наукових праць: Випуск 3(14).-К.,НАУ.- 2008.- С.129-136.

62. Захарченко Н.А., Топалова К.Н. Использование булевых преобразований для быстрой идентификации абонентов на основе концепции нулевых знаний. // Матеріали XII Міжнародної науково-технічної конференції ”Системний аналіз та інформаційні технології”.- К.:НТУУ ”КПІ”.-2010.- С.441.

63. Ияд Мохд Маджид Ахмад Шахрури. Использование хеш-памяти для быстрой аутентификации абонентов многопользовательских систем // Вісник Національного технічного університету України “КПІ” Інформатика, управління та обчислювальна техніка. К.: ТОО „ВЕК+”.- № 43.- 2005.- С. 75-85.

64. Ияд Мохд Маджид Ахмад Шахрури, Магрело В.Д. Потапенко М.М. Ідентифікація віддалених абонентів на основі технології нейронних мереж // Матеріали X Міжнародної науково-технічної конференції ”Системний аналіз та інформаційні технології”.-К.:НТУУ ”КПІ”.-2008.- С.357.

65. Захариудакис Лефтерис. Метод быстрой аутентификации удаленных пользователей на основе концепции “нулевых знаний” / Наукові записки Українського науково-дослідного інституту зв’язку. 2017.- № 1 (45).– С.109-117.

66. Захаріудакіс Лефтеріс Метод строгої ідентифікації віддалених користувачів з використанням перетворень на полях Галуа / Захаріудакіс Лефтеріс, А.А.Олієвський // IV Міжнародна науково-практична конференція ”Summer

Infocom Advanced Solution 2017”,: Збірник матеріалів конференції. К.:КПІ ім. Ігоря Сікорського, 2017.- С.56-60.

67. Марковський О.П. Метод строгої ідентифікації віддалених абонентів на основі стандартизованих шифроблоків та хеш-перетворень / О.П. Марковський, Захаріудакіс Лефтеріс., М.Ф. Федотов // Вісник Національного технічного університету України “КПІ” Інформатика, управління та обчислювальна техніка. К.: ТОО „ВЕК+”. 2016.- № 64.- С. 161-165.

68. Марковський О.П. Метод строгої ідентифікації абонентів в телекомунікаційних системах / О.П. Марковський, Лефтеріс Захаріудакіс, М.Ф. Федотов // XI Міжнародна науково-технічна конференція “Проблеми телекомунікації” ПТ-2017: Збірник матеріалів конференції.К.:КПІ ім. Ігоря Сікорського, 2017.- С.334-337.

69. Марковский А.П., Абу Усбах А.Н., Аль-Омар Салех. Получение систем ортогональных булевых SAC–функций для систем защиты информации. // Вісник Національного технічного університету України ”КПІ”. Інформатика, управління та обчислювальна техніка. 2001, - № 36. - С.94-108.

70. Марковский А.П., Осадчий В.В., Аль-Омар Салех. Получение балансных булевых SAC–функций для систем защиты информации //Вісник Національного технічного університету України ”КПІ”. Інформатика, управління та обчислювальна техніка.-2001.- № 36.-С.54-60.

71. Марковский А.П., Абу Усбах А.Н., Иваненко Я.П. К вопросу об определении нелинейности булевых функций специальных классов. // Вісник Національного технічного університету України ”КПІ”. Інформатика, управління та обчислювальна техніка, -2002,- № 37. - С.14-24.

72. Марковский А.П., Азаде Герами, Ияд Мохд Маджид Ахмад Шахрури. Организация идентификации абонентов многопользовательских систем // Труды 9-й ”Международной конференции ”Современные информационные и электронные технологии”. Одесса.-2008.- С.129.

73. Марковский А.П., Зюзя А.А., Шерстюк В.Д. Получение булевых преобразований специальных классов для построения эффективных алгоритмов защиты информации // Вісник Національного технічного університету України ”КПІ”. Інформатика, управління та обчислювальна техніка. К.,"ВЕК++",- 2008.- № 49.- С.7-13.

74. Марковський О. П. Використання алгебри полів Галуа для реалізації концепції нульових знань при ідентифікації та автентифікації віддалених користувачів / О.П.Марковський, Захаріудакіс Лефтеріс., В.Р.Максимук // Электронное моделирование. 2017.- № 6.- С.96-110.

75. Марковський О.П. Метод швидкої строгої ідентифікації віддалених користувачів / О.П. Марковський, Лефтеріс Захаріудакіс, М.Ф. Федотов // X Міжнародна науково-технічна конференція “Комп’ютерні системи та мережні технології” : Тези доповідей.К.: НАУ, 2017.- С. 59-61

76. Стіренко С.Г. Спосіб прискореного обчислення модулярної експоненти / С.Г. Стіренко О.П. Марковський, Захаріудакіс Лефтеріс., Л.Д. Міщенко // Вісник Національного технічного університету України “КПІ” Інформатика, управління та обчислювальна техніка. К.: ТОО „ВЕК+”.2017.- № 65.- С. 110-115.

77. Мухін В.Є. Метод ідентифікації віддалених абонентів на основі концепції “нульових знань” / В.Є. Мухін, Лефтеріс Захаріудакіс, Ю.Н. Герасименко, М.С. Козерацький // Телекомунікаційні та інформаційні технології, 2017 –№1.– С.50-57. рг.-2005.- 286 с.

ДОДАТОК А

Спосіб криптографічно-строгої ідентифікації на основі хеш-перетворень з програмованими колізіями

Текст програми

Аркушів 16

Київ – 2023 р.

HashConverter.java

```
package converter;

import converter.dto.EstimationResWrapper;
import converter.error.InvalidKeyException;
import org.mindrot.jbcrypt.BCrypt;

import java.time.Duration;
import java.time.Instant;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Random;


public class HashConverter {

    /**
     * Size of the converter.
     * Determines the number of functions and transformation levels amount.
     */
    private final int sectionsAmount;

    /**
     * Salt of the out key hasher.
     * Auto-generated if no value is set.
     */
    private final String salt;

    /**
     * Data conversion functions.
     */
    private final ConversionFunction[] functions;

    /**
     * Register of valid input keys. Key is invalidated after activation.
     */
    private final ArrayList<DataKey> inputKeys = new ArrayList<>();

    /**
     * Register of deactivated input keys.
     */
    private final ArrayList<DataKey> inputKeyBlackList = new ArrayList<>();

    /**
     * Secure random util. Inner purposes only.
     */
    private final Random rand = new Random();
```

```
/**
 * Recorder of the converter movements estimation util.
 * Inner purposes only.
 */
private final EstimatorControl estimatorControl = new EstimatorControl();

public HashConverter(int sectionsAmount, int rate) {
   this.sectionsAmount = sectionsAmount;
   this.salt = BCrypt.gensalt();
   functions = new ConversionFunction[sectionsAmount];
   for (int i = 0; i < functions.length; i++) {
      functions[i] = new ConversionFunction(rate);
   }
   generateTraceMap();
}

public HashConverter(int sectionsAmount, int rate, String salt) {
   this.sectionsAmount = sectionsAmount;
   this.salt = salt;
   functions = new ConversionFunction[sectionsAmount];
   for (int i = 0; i < functions.length; i++) {
      functions[i] = new ConversionFunction(rate);
   }
   generateTraceMap();
}

public void init() {
   DataKey inKey = new DataKey(sectionsAmount);
   ConversionPair[][] trace = generateTraceMap();
   for (int i = 0; i < sectionsAmount; i++) {
      for (int j = 0; j < sectionsAmount; j++) {
         // get input value
         long inValue;
         if (i == 0) {
            List<Long> freeKeys = functions[j].getFreeKeys();
            inValue = freeKeys.get(rand.nextInt(freeKeys.size()));
            inKey.setValue(j, inValue);
         } else {
            inValue = trace[i - 1][j].getOut() ^ trace[i - 1][(j + 1) % sectionsAmount].getOut();
         }

         // generate output value if no mapping
         if (!functions[j].isMappingExists(inValue)) {
            List<Long> freeValues = functions[j].getFreeValues();
            functions[j].getMappings().put(inValue, freeValues.get(rand.nextInt(freeValues.size())));
         }

         // save state
```

```
            trace[i][j] = new ConversionPair(inValue, functions[j].getValue(inValue));
        }
    }
    inputKeys.add(inKey);
}

public void importState(ConversionFunction[] functions) {
    if (functions.length != this.functions.length) {
        throw new IllegalArgumentException("Conversion state cannot be imported");
    }
    if (sectionsAmount >= 0) System.arraycopy(functions, 0, this.functions, 0, sectionsAmount);
}

public ConversionFunction[] exportState() {
    return this.functions;
}

public ArrayList<DataKey> exportInputKeys() {
    return inputKeys;
}

public DataKey buildTrace() {
    DataKey inputKey = new DataKey(sectionsAmount);
    for (int j = 0; j < sectionsAmount; j++) {
        List<Long> takenKeys = functions[j].getTakenKeys();
        long inValue = takenKeys.get(rand.nextInt(takenKeys.size()));
        inputKey.setValue(j, inValue);
    }

    ConversionPair[][] trace = generateTraceMap();
    for (int i = 0; i < sectionsAmount; i++) {
        for (int j = 0; j < sectionsAmount; j++) {
            if (i == 0) {
                long inValue = inputKey.getValue(j);
                trace[i][j] = new ConversionPair(inValue, getOrGenerateAnyValue(j, inValue));
                continue;
            }
            long inValue = trace[i - 1][j].getOut() ^ trace[i - 1][(j + 1) % sectionsAmount].getOut();
            trace[i][j] = new ConversionPair(inValue, getOrGenerateAnyValue(j, inValue));
        }
    }

    DataKey outputKey = new DataKey(sectionsAmount);
    for (int i = 0; i < sectionsAmount; i++) {
        outputKey.setValue(i, inputKey.getValue(i) ^ trace[sectionsAmount - 1][i].getOut());
    }

    inputKeys.add(inputKey);
    return inputKeys.get(inputKeys.size() - 1);
```

```
}

private long getOrGenerateAnyValue(int funcIndex, long inputValue) {
    if (functions[funcIndex].isMappingExists(inputValue)) {
        return functions[funcIndex].getValue(inputValue);
    }
    List<Long> freeValues = functions[funcIndex].getFreeValues();
    long outValue = freeValues.get(rand.nextInt(freeValues.size()));
    functions[funcIndex].getMappings().put(inputValue, outValue);
    return outValue;
}

public EstimationResWrapper estimateTracesAmount() {
    return estimateTracesAmount(null);
}

public EstimationResWrapper estimateTracesAmount(Duration duration) {
    try {
        estimatorControl.setStartAt(Instant.now());
        System.out.printf(
                "Start estimation calculations, sectionsAmount=%s: %s\n",
                sectionsAmount, estimatorControl.getStartAt()
        );

        long totalInKeyAmount = Arrays.stream(functions)
                .mapToLong(ConversionFunction::getUsedSize)
                .reduce(1, (subtotal, funcSize) -> subtotal * funcSize);
        estimatorControl.setTotalInKeyAmount(totalInKeyAmount);

        moveForward(0, 0, cloneState(), generateTraceLevel(), null, duration);
        return estimatorControl.toEstimationResWrapper();
    } finally {
        resetEstimatorState();
    }
}

public EstimationResWrapper getEstimationResults() {
    return estimatorControl.toEstimationResWrapper();
}

private void moveForward(
        int level, int index,
        ConversionFunction[] func, ConversionPair[] levelTrace,
        DataKey inputKey,
        Duration maxDuration
) {
    if (isExecutionTimeExceed(maxDuration)) {
        return;
    }
```

```
        if (level == sectionsAmount - 1 && index == sectionsAmount) {
            DataKey outKey = new DataKey(sectionsAmount);
            for (int i = 0; i < sectionsAmount; i++) {
                outKey.setValue(i, inputKey.getValue(i) ^ levelTrace[i].getOut());
            }
            estimatorControl.appendOutKey(outKey);
            estimatorControl.appendInKey(inputKey);
            estimatorControl.incrementTracesAmount();
//            System.out.println("Interim traces amount: " + estimatorControl.getTotalTracesAmount());
            return;
        }

        if (index >= sectionsAmount) {
            ConversionPair[] nextLevelTrace = generateTraceLevel();
            for (int i = 0; i < sectionsAmount; i++) {
                nextLevelTrace[i].setIn(levelTrace[i].getOut() ^ levelTrace[(i + 1) %
sectionsAmount].getOut());
            }
            moveForward(level + 1, 0, func, nextLevelTrace, inputKey, maxDuration);
            return;
        }
        if (level > 0) {
            if (func[index].isMappingExists(levelTrace[index].getIn())) {
                levelTrace[index].setOut(func[index].getValue(levelTrace[index].getIn()));
                moveForward(level, index + 1, func, levelTrace, inputKey, maxDuration);
            } else {
                List<Long> freeValues = func[index].getFreeValues();
                for (long outValue : freeValues) {
                    // clone funcs state and set option
                    ConversionFunction[] funcVariation = cloneState(func);
                    funcVariation[index].getMappings().put(levelTrace[index].getIn(), outValue);

                    // clone level state and set option
                    ConversionPair[] levelStateVariation = cloneTraceLevel(levelTrace);
                    levelStateVariation[index].setOut(outValue);

                    moveForward(level, index + 1, funcVariation, levelStateVariation, inputKey,
maxDuration);
                }
            }
        } else {
            List<Long> takenKeys = func[index].getTakenKeys();
            for (Long takenKey : takenKeys) {
                // clone level state and set option
                long outValue = func[index].getValue(takenKey);
                ConversionPair[] levelStateVariation = cloneTraceLevel(levelTrace);
                levelStateVariation[index].setIn(takenKey);
                levelStateVariation[index].setOut(outValue);
```

```
            // clone input key state and set option
            DataKey inputKeyVariation = inputKey == null
                    ? new DataKey(sectionsAmount)
                    : new DataKey(inputKey.getKey());
            inputKeyVariation.setValue(index, takenKey);

            moveForward(level, index + 1, func, levelStateVariation, inputKeyVariation, maxDuration);
        }
    }
}

private boolean isExecutionTimeExceed(Duration maxDuration) {
    if (maxDuration == null) {
        return false;
    }
    Duration executionTime = Duration.between(Instant.now(), estimatorControl.getStartAt());
    return executionTime.abs().toSeconds() > maxDuration.toSeconds();
}

public void resetEstimatorState() {
    this.estimatorControl.reset();
}

private long calculateMovementsAmount(long[] levelIn) {
    long result = 1;
    for (int i = 0; i < sectionsAmount; i++) {
        result = functions[i].isMappingExists(levelIn[i]) ? 1 : functions[i].getFreeValues().size();
    }
    return result;
}


public void activateKey(DataKey inputKey) {
    if (inputKeyBlackList.contains(inputKey)) {
        throw new InvalidKeyException("Key is already used");
    }
    verifyInputKey(inputKey);

    inputKeys.remove(inputKey);
    inputKeyBlackList.add(inputKey);
}

private void verifyInputKey(DataKey inputKey) {
    DataKey outKey = calculateKey(inputKey);
    DataKey keyH1 = new DataKey(Arrays.copyOfRange(
            outKey.getKey(),
            0,
            outKey.getKey().length / 2
```

```
    ));
    DataKey keyH2 = new DataKey(Arrays.copyOfRange(
            outKey.getKey(),
            outKey.getKey().length / 2,
            outKey.getKey().length
    ));

    if (!BCrypt.checkpw(keyH1.zip(), BCrypt.hashpw(keyH2.zip(), salt))) {
        throw new InvalidKeyException("Invalid key");
    }
}

private DataKey calculateKey(DataKey inputKey) {
    ConversionPair[][] trace = generateTraceMap();
    for (int i = 0; i < sectionsAmount; i++) {
        for (int j = 0; j < sectionsAmount; j++) {
            if (i == 0) {
                long inValue = inputKey.getValue(j);
                trace[i][j] = new ConversionPair(inValue, functions[j].getVerifiedValue(inValue));
                continue;
            }
            long inValue = trace[i - 1][j].getOut() ^ trace[i - 1][(j + 1) % sectionsAmount].getOut();
            trace[i][j] = new ConversionPair(inValue, functions[j].getVerifiedValue(inValue));
        }
    }

    DataKey outputKey = new DataKey(sectionsAmount);
    for (int i = 0; i < sectionsAmount; i++) {
        outputKey.setValue(i, inputKey.getValue(i) ^ trace[sectionsAmount - 1][i].getOut());
    }
    return outputKey;
}

private ConversionFunction[] cloneState() {
    ConversionFunction[] result = new ConversionFunction[functions.length];
    for (int i = 0; i < functions.length; i++) {
        result[i] = functions[i].clone();
    }
    return result;
}

private ConversionFunction[] cloneState(ConversionFunction[] functions) {
    ConversionFunction[] result = new ConversionFunction[functions.length];
    for (int i = 0; i < functions.length; i++) {
        result[i] = functions[i].clone();
    }
    return result;
}
```

```
private ConversionPair[][] cloneTraceMap(ConversionPair[][] trace) {
   ConversionPair[][] result = new ConversionPair[sectionsAmount][sectionsAmount];
   for (int i = 0; i < sectionsAmount; i++) {
      for (int j = 0; j < sectionsAmount; j++) {
         result[i][j] = new ConversionPair(trace[i][j].getIn(), trace[i][j].getOut());
      }
   }
   return result;
}

private ConversionPair[] cloneTraceLevel(ConversionPair[] trace) {
   ConversionPair[] result = new ConversionPair[sectionsAmount];
   for (int i = 0; i < sectionsAmount; i++) {
      result[i] = new ConversionPair(trace[i].getIn(), trace[i].getOut());
   }
   return result;
}

private ConversionPair[][] generateTraceMap() {
   ConversionPair[][] trace = new ConversionPair[sectionsAmount][sectionsAmount];
   for (int i = 0; i < sectionsAmount; i++) {
      for (int j = 0; j < sectionsAmount; j++) {
         trace[i][j] = new ConversionPair();
      }
   }
   return trace;
}

private ConversionPair[] generateTraceLevel() {
   ConversionPair[] levelTrace = new ConversionPair[sectionsAmount];
   for (int i = 0; i < sectionsAmount; i++) {
      levelTrace[i] = new ConversionPair();
   }
   return levelTrace;
}

private void printState(ConversionPair[][] trace) {
   System.out.println("Trace state:");
   for (int j = 0; j < sectionsAmount; j++) {
      for (int i = 0; i < sectionsAmount; i++) {
         System.out.print(trace[j][i].getIn() + "\t");
      }
      System.out.println();
      for (int i = 0; i < sectionsAmount; i++) {
         System.out.print(trace[j][i].getOut() + "\t");
      }
      System.out.println("\n----------------------");
   }
```

```
        System.out.println("\nFunctions state:");
        for (int j = 0; j < sectionsAmount; j++) {
            System.out.printf("Func %s, used values=%s:\t", j, functions[j].getUsedSize());
            functions[j]
                    .getMappings()
                    .forEach((in, out) -> {
                        System.out.printf("[%s -> %s],\t", in, out == ValueConstants.UNMAPPED ? "-" : out);
                    });
            System.out.println();
        }
    }

public static class EstimatorControl {

        private Instant startAt = Instant.EPOCH;
        private long totalTracesAmount = 0;
        private long totalInKeyAmount = 0;
        private final List<DataKey> outKeys = new ArrayList<>();
        private final List<DataKey> inKeys = new ArrayList<>();

        public EstimatorControl() {
        }

        public void reset() {
            this.outKeys.clear();
            this.totalTracesAmount = 0;
            this.totalInKeyAmount = 0;
            this.startAt = Instant.EPOCH;
        }

        public void incrementTracesAmount() {
            this.totalTracesAmount = this.totalTracesAmount + 1;
        }

        public void appendInKey(DataKey inputKey) {
            if (!inKeys.contains(inputKey)) {
                inKeys.add(inputKey);
            }
        }

        public void appendOutKey(DataKey outputKey) {
            if (!outKeys.contains(outputKey)) {
                outKeys.add(outputKey);
            }
        }

        public EstimationResWrapper toEstimationResWrapper() {
            return new EstimationResWrapper(
                    totalTracesAmount,
```

```
                totalInKeyAmount,
                inKeys.size(),
                outKeys.size()
            );
        }

        public Instant getStartAt() {
            return startAt;
        }

        public void setStartAt(Instant startAt) {
            this.startAt = startAt;
        }

        public long getTotalTracesAmount() {
            return totalTracesAmount;
        }

        public void setTotalInKeyAmount(long totalInKeyAmount) {
            this.totalInKeyAmount = totalInKeyAmount;
        }
    }
}
```

DataKey.java

```
package converter;

import java.util.Arrays;
import java.util.Collections;

public class DataKey {

    private long[] key;

    public DataKey(long[] key) {
        this.key = new long[key.length];
        System.arraycopy(key, 0, this.key, 0, key.length);
    }

    public DataKey(int size) {
        key = new long[size];
        for (int i = 0; i < size; i++) {
            key[i] = ValueConstants.UNMAPPED;
        }
    }

    public static String zip(DataKey key) {
```

```
    StringBuilder str = new StringBuilder();
    for (long value : key.getKey()) {
        str.append(value);
    }
    return str.toString();
}

public String zip() {
    return zip(this);
}

public static DataKey unzip(String sKey, int size) {
    long[] invertedKey = Arrays.stream(new StringBuilder(sKey)
                .reverse()
                .toString()
                .split(String.format("(?<=\\G.{%s})", size)))
            .map(Long::parseLong)
            .sorted(Collections.reverseOrder())
            .mapToLong(l -> l)
            .toArray();
    return new DataKey(invertedKey);
}

public long[] getKey() {
    return key;
}

public void setKey(long[] key) {
    this.key = key;
}

public void setKey(DataKey key) {
    this.key = key.getKey();
}

public long getValue(int index) {
    return this.key[index];
}

public void setValue(int index, long value) {
    this.key[index] = value;
}

@Override
public String toString() {
    StringBuilder str = new StringBuilder();
    for (long value : key) {
        str.append(value).append(" ");
    }
```

```
        return str.toString().trim();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        DataKey dataKey = (DataKey) o;
        return Arrays.equals(key, dataKey.key);
    }

    @Override
    public int hashCode() {
        return Arrays.hashCode(key);
    }
}
```

ConversionPair.java

```
package converter;

import java.util.Objects;

public class ConversionPair {

    private long in;
    private long out;

    public ConversionPair() {
        this.in = ValueConstants.UNMAPPED;
        this.out = ValueConstants.UNMAPPED;
    }

    public ConversionPair(long in, long out) {
        this.in = in;
        this.out = out;
    }

    public long getIn() {
        return in;
    }

    public void setIn(long in) {
        this.in = in;
    }

    public long getOut() {
        return out;
```

```
    }

    public void setOut(long out) {
        this.out = out;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        ConversionPair that = (ConversionPair) o;
        return in == that.in && out == that.out;
    }

    @Override
    public int hashCode() {
        return Objects.hash(in, out);
    }
}
```

ConversionFunction.java

```
package converter;

import converter.error.IllegalMappingException;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.stream.Collectors;
import java.util.stream.LongStream;

public class ConversionFunction implements Cloneable {

    /**
     * Input and Output pair for func(input) = output.
     */
    private final Map<Long, Long> mappings = new HashMap<>();

    /**
     * Rate of the function. Defines the volume of the function values.
     */
    private final int rate;

    public ConversionFunction(int rate) {
        this.rate = rate;
        long range = (long) Math.pow(2, rate);
```

```
        for (long i = 0; i < range; i++) {
            mappings.put(i, ValueConstants.UNMAPPED);
        }
    }

    public ConversionFunction(int rate, Map<Long, Long> mappings) {
        this.rate = rate;
        setMappings(mappings);
    }

    public void clear() {
        mappings.clear();
    }

    public long getValue(long inputValue) {
        return mappings.get(inputValue);
    }

    public boolean isMappingExists(long inputValue) {
        return !Objects.equals(mappings.get(inputValue), ValueConstants.UNMAPPED);
    }

    public long getVerifiedValue(long inputValue) {
        if (!mappings.containsKey(inputValue) || mappings.get(inputValue) ==
ValueConstants.UNMAPPED) {
            throw new IllegalMappingException("No mapping exists for key=" + inputValue);
        }
        return mappings.get(inputValue);
    }

    public long getInputValue(int outputValue) {
        if (outputValue == ValueConstants.UNMAPPED) {
            return ValueConstants.UNMAPPED;
        }
        for (long in : mappings.keySet()) {
            if (mappings.get(in) == outputValue) {
                return in;
            }
        }
        return ValueConstants.UNMAPPED;
    }

    public Map<Long, Long> getMappings() {
        return mappings;
    }

    public void setMappings(Map<Long, Long> mappings) {
        this.mappings.clear();
        this.mappings.putAll(mappings);
```

```
    }

    public List<Long> getTakenKeys() {
        return mappings.keySet()
                .stream()
                .filter(key -> !mappings.get(key).equals(ValueConstants.UNMAPPED))
                .collect(Collectors.toList());
    }

    public List<Long> getFreeKeys() {
        return mappings.keySet()
                .stream()
                .filter(key -> mappings.get(key).equals(ValueConstants.UNMAPPED))
                .collect(Collectors.toList());
    }

    public List<Long> getTakenValues() {
        return mappings.values()
                .stream()
                .filter(value -> !value.equals(ValueConstants.UNMAPPED))
                .collect(Collectors.toList());
    }

    public List<Long> getFreeValues() {
        List<Long> allValues = LongStream.range(0, (long) Math.pow(2,
rate)).boxed().collect(Collectors.toList());
        allValues.removeAll(getTakenValues());
        return allValues;
    }

    public long getUsedSize() {
        return mappings.values()
                .stream()
                .filter(value -> !value.equals(ValueConstants.UNMAPPED))
                .count();
    }

    @Override
    protected ConversionFunction clone() {
        ConversionFunction cloned = new ConversionFunction(rate);
        cloned.setMappings(mappings);
        return cloned;
    }
}
```

EstimationResWrapper.class

```
package converter.dto;

public class EstimationResWrapper {

    /**
     * Total amount of possible traces
     */
    private final long tracesAmount;

    /**
     * Total amount of unique in keys
     */
    private final long totalInKeysAmount;

    /**
     * Amount of used unique in keys
     */
    private final long inKeysAmount;

    /**
     * Total amount of built unique out keys
     */
    private final long outKeysAmount;

    public EstimationResWrapper(long tracesAmount, long totalInKeysAmount, long inKeysAmount,
long outKeysAmount) {
        this.tracesAmount = tracesAmount;
        this.totalInKeysAmount = totalInKeysAmount;
        this.inKeysAmount = inKeysAmount;
        this.outKeysAmount = outKeysAmount;
    }

    public long getTracesAmount() {
        return tracesAmount;
    }

    public long getTotalInKeysAmount() {
        return totalInKeysAmount;
    }

    public long getInKeysAmount() {
        return inKeysAmount;
    }

    public long getOutKeysAmount() {
        return outKeysAmount;
    }
}
```