Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Аналіз алгоритмів забезпечення цілісності інформації

Виконав: студент VI курсу, групи СБм-61
спеціальності 125 Кібербезпека та захист інформації
(шифр і назва спеціальності)

(підпис) Цимбрак І.С. (прізвище та ініціали)

Керівник (підпис) Карпінський М.П (прізвище та ініціали)

Нормоконтроль (підпис) Тимощук Д.І. (прізвище та ініціали)

Завідувач кафедри (підпис) Загородна Н.В. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль - 2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Загородна Н.В.
(підпис) (прізвище та ініціали)
«__» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека та захист інформації
(шифр і назва спеціальності)

Студенту Цимбраку Івану Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Аналіз алгоритмів забезпечення цілісності інформації

Керівник роботи Карпінський Микола Петрович, д.т.н., проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «20» 11 2024 року № 4/7-1112

2. Термін подання студентом завершеної роботи 22.12.2024 р.

3. Вихідні дані до роботи наукові літературні джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
1. Аналіз предметної області.
2. Реалізація, тестування та аналіз алгоритмів.
3. Дослідження стійкості криптографічних алгоритмів.
4. Охорона праці та безпека в надзвичайних ситуаціях

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Тема роботи. 2. Актуальність. 3. Мета, задачі, об'єкт, предмет дослідження, наукова новизна. 4. Аналіз предметної області. 5. Реалізація алгоритмів.
6. Тестування та проведення аналізу алгоритмів
7. Код із перевіркою на парність та код Хеммінга. 8. CRC та код Ріда-Соломона.
9. Функція хешування SHA256 та Алгоритм імітовставки HMAC-SHA256.
10. Алгоритм ЕЦП з RSA. 11. Дослідження стійкості криптографічних алгоритмів.
12. Скріншоти роботи створеного застосунку. 13. Підсумки аналізу кодів.
14. Висновки. Основні результати дослідження.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Г.М., зав. каф. КС		
Безпека в надзвичайних ситуаціях			

7. Дата видачі завдання 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	20.11 – 22.11	*Виконано*
2.	Підбір джерел про алгоритми забезпечення цілісності		
	інформації	23.11 – 26.11	*Виконано*
3.	Опрацювання джерел про алгоритми забезпечення		
	цілісності інформації	27.11 – 30.11	*Виконано*
4.	Виконання дослідження щодо аналізу алгоритмів		
	забезпечення цілісності інформації	01.12 – 06.12	
5	Розробка ПЗ для тестування алгоритмів	07.12 – 10.12	
6.	Оформлення розділу «Аналіз предметної області»	11.12 – 13.12	*Виконано*
7.	Оформлення розділу «Реалізація, тестування та аналіз алгоритмів»	14.12 – 15.12	*Виконано*
8.	Оформлення розділу «Дослідження стійкості криптографічних алгоритмів»	16.12 – 18.12	*Виконано*
9.	Виконання завдання до підрозділу «Охорона праці та		
	безпека в надзвичайних ситуаціях»	06.12 – 16.12	*Виконано*
10.	Оформлення кваліфікаційної роботи	14.12 – 19.12	*Виконано*
11.	Нормоконтроль	18.12 – 20.12	*Виконано*
12.	Перевірка на плагіат	16.12 – 19.12	*Виконано*
13.	Попередній захист кваліфікаційної роботи	17.12 – 20.12	*Виконано*
14.	Захист кваліфікаційної роботи	23.12	

Студент _____ Цимбрак І.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Карпінський М.П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Аналіз алгоритмів забезпечення цілісності інформації // Цимбрак Іван Сергійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем та програмної інженерії, кафедра кібербезпеки, група СБм-62 // Тернопіль, 2024 // с. – 73, рис. – 36, табл. – 2, слайд. – 14, бібліогр. –35.

Ключові слова: алгебраїчні коди, електронний підпис, код Ріда-Соломона, код Хеммінга, хешування SHA256, цілісність інформації, HMAC

У кваліфікаційній роботі було проаналізовано алгоритми: код з перевіркою на парність, код Хеммінга, циклічний надлишковий код, код Ріда - Соломона, алгоритм хешування SHA256, алгоритм імітовставки HMAC з використанням в якості функції хешування функцію SHA256 та алгоритм хешування електронних підписів з використанням функції SHA256 з шифруванням алгоритмом RSA.

Для їх реалізації було досліджені підалгоритми: метод Берлекемпа – Мессі та метод Пітерсена - Горенштейна – Цирлера для декодування кодів Ріда – Соломона, алгоритм тесту на простоту Міллера – Рабіна для генерації простих чисел, необхідних для створення пари ключів в алгоритмі RSA.

Отримані результати дослідження та тестування алгоритмів містять порівняння залежностей часу кодування та декодування з різними параметрами, а також отримані гістограми розподілу помилок, відсотки випадків помилок - одиночних, виявлених, коректно виправлених.

Результати проведених експериментальних досліджень показують коректність роботи усіх алгоритмів забезпечення цілісності інформації.

ANNOTATION

Analysis of algorithms for information integrity // Tsymbrak Ivan // Ternopil Ivan Pul'uj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cyber Security // Ternopil, 2024 // p. - 73, Fig. – 36, Table - 2, Slides - 14, References - 35.

Keywords: algebraic codes, electronic signature, Reed-Solomon code, Hamming code, SHA256 hashing, information integrity, HMAC

Algorithms were analyzed in the thesis: code with parity check, Hamming code, cyclic redundant code, Reed-Solomon code, SHA256 hashing algorithm, HMAC impersonation algorithm using the SHA256 function as a hashing function, and electronic signature hashing algorithm using the SHA256 function with encryption with the RSA algorithm.

For their implementation, sub-algorithms were investigated: the Berlekamp-Massey method and the Petersen-Gorenstein-Zirler method for decoding Reed-Solomon codes, the Miller-Rabin simplicity test algorithm for generating prime numbers necessary for creating a pair of keys in the RSA algorithm.

The obtained results of the research and testing of the algorithms contain a comparison of the dependencies of the encoding and decoding time with various parameters, as well as the obtained histograms of the distribution of errors, percentages of error cases - single, detected, correctly corrected.

The results of the conducted experimental studies show the correctness of the operation of all algorithms for ensuring the integrity of information.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ СКОРОЧЕНЬ І ТЕРМІНІВ

CRC (Cyclic Redundancy Check) – циклічний надлишковий код.

HMAC (Hash-based Message Authentication Code) – код автентифікації повідомлень на основі хешування.

LFSR (Linear Feedback Shift Register) – регістр зсуву з лінійним зворотним зв'язком.

SHA 256 (Secure Hash Algorithm 256 bit) – безпечний алгоритм хешування.

БМ-метод – Берлекемпа – Мессі алгоритм.

ПГЦ-метод – Пітерсена - Горенштейна – Цирлера алгоритм.

РС код – код Ріда-Соломона.

ЗМІСТ

ВСТУП

Актуальність теми. Безпека інформації на поточний день залишається актуальною темою для досліджень. З розвитком цифрових технологій та онлайн банкінгу зростає мотивація зловмисників у пошуку вразливостей у цих системах. При недостатньо стійкій криптографічній системі зловмисник може експлуатувати вразливість даної системи для досягнення своїх цілей.

Одним із значних сервісів інформаційної безпеки є цілісність інформації. Під цілісністю інформації розуміється гарантованість того, що інформація залишиться незмінною, коректною та автентичною. Таким чином, при коректній роботі сервісу користувач завжди отримуватиме лише достовірну інформацію. Для зловмисника наявність загрози цілісності дає великий спектр можливостей. Наприклад, за наявності такої вразливості порушник може виводити користувачеві недостовірну інформацію, або видавати себе цього користувача для сервера.

Для забезпечення коректної роботи підсистеми цілісності використовуються різні алгоритми, що виконують певні завдання: резервне копіювання, архівування та відновлення, забезпечення стійкості до відмови, виявлення спотворення та цілеспрямованої зміни інформації і т.д.

Мета дослідження: проаналізувати алгоритми, що виявляють спотворення та цілеспрямовану зміну інформації. Під аналізом розумітиметься виведення (теоретичне чи практичне) оцінок для аналізованих алгоритмів.

Оцінюватимуться такі аспекти: здатність виявляти помилки; здатність виправляти помилки; час виконання за різних параметрів (наприклад, довжина повідомлення); складність реалізації.

В роботі поставлено та розв’язано наступні задачі:

– вивчення різних алгоритмів забезпечення захисту від спотворення чи несанкціонованої зміни інформації;

– програмна реалізація вивчених алгоритмів із використанням мови програмування Python;

– аналіз алгоритмів, побудова візуалізованих уявлень (графіків,

рисунків тощо) з використанням загальнодоступних бібліотек (matplotlib тощо);

– формування висновку з урахуванням результатів аналізу.

Об’єкт дослідження: цілісність інформації.

Предмет дослідження: алгоритми забезпечення цілісності інформації.

Методи дослідження: наукові праці закордонних та вітчизняних учених за тематикою дослідження, фундаментальні положення кібербезпеки.

Наукова новизна отриманих результатів:

– отримав подальший розвиток механізм застосування алгебраїчних кодів та криптографічних алгоритмів для забезпечення цілісності інформації;

Практичне значення одержаних результатів. Можуть бути застосовані для виявлення спотворення та цілеспрямованої зміни інформації у реальних системах.

Апробація. Результати дослідження апробовано на XII науково-технічній конференції «Інформаційні моделі, системи та технології» у вигляді опублікованих тез.

Структура роботи. Робота складається з пояснювальної записки та графічної частини. Пояснювальна записка складається з вступу, 4 розділів, висновків, списку використаної літератури та застосунків. Обсяг роботи: пояснювальна записка – 73 арк. формату А4, графічна частина – 14 слайдів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Алгебраїчні коди

Передача інформації у найпростішому випадку має такий вигляд (рис. 1.1) Передавач надсилає по каналу зв'язку інформацію до приймача. У каналі зв'язку існує випадкова перешкода, і кожен біт, що передається, може спотворитися з деякою ймовірністю [1].

Рисунок 1.1 – Найпростіший випадок передачі

Нехай через канал передається інформаційний вектор довжини n, який позначатиметься $u = (u_1, u_2, .., u_n)$, $u_i = 0,1$. У каналі зв'язку перешкоди можуть створити певну помилку. Так як помилка може статися на будь-якому біті, можна ввести вектор помилки $e = (e_1, e_2, \ldots, e_n)$. Очевидно, якщо $e = 0$, то помилки не сталося.

Виникає питання виявлення помилки та її виправлення. Для цього інформаційний вектор, що передається, перетворюється на інший вектор довжини $n + k$, причому $n > k$. Дана процедура називається кодуванням. Кодування відбувається таким чином, щоб отриманий вектор довжини $n + k$ деяким чином зіставлявся з початковим вектором довжини n. Процедуру, зворотну кодування, називають декодуванням. Тепер передача інформації має схему, представлену на рис. 1.2.

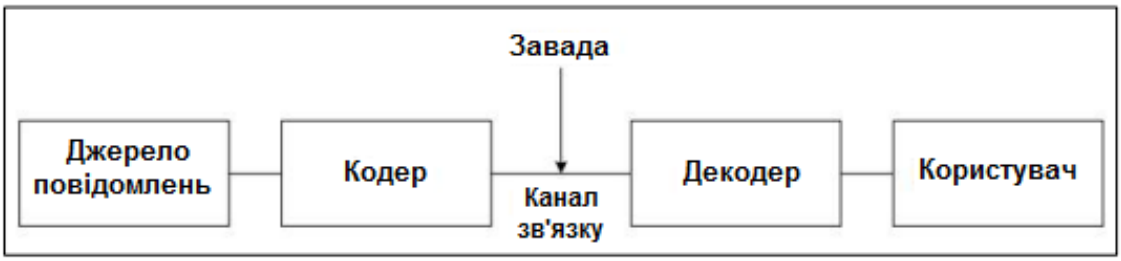

Рисунок 1.2 – Система передачі інформації

Слід також запровадити такі поняття:

– відстань Хеммінга. Відстанню Хеммінгу $d(a,b)$ двох векторів $a = (a_1, \ldots, a_n)$ і $b = (b_1, \ldots, bn)$ довжини n називають число попарно відмінних компонентів a_t і b_t цих векторів: $d(a,b) = \sum_{i=1}^{n} I(a_i = b_i)$, де I – індикаторна функція;

– кодова відстань d - мінімальна відстань Хеммінга за всіма парами векторів цього коду A:

$$a, b \in A, d = \min_{a \neq b} d(a,b).$$

За значенням кодової відстані можна судити про здатність виявляти та виправляти помилки. Нехай кодова відстань коду A $d \geq 2t + 1$. Тоді код A дозволяє виправити t та менш незалежних помилок або виявити $2t$ і менше помилок.

1.2 Поля Галуа

У цьому підрозділі буде описано метод задання поля Галуа та операції в ньому. Визначення групи, кільця, поля тут не наводяться.

Поле Галуа $GF(p^m)$ можна побудувати як примітивний многочлен $g(x)$. Зважаючи на те, що поле Галуа будується по модулю цього многочлена, то $g(a) = 0$. Таким чином, поле складається з багаточленів від a ступеня, що не перевищує m.

Елементи поля Галуа можуть представлятися у формі багаточлена, степеня та вектора. Наприклад, у полі $GF(2^3)$ багаточлен $a^2 + a$ у векторній формі

представляється як 110 (одиниці в ненульових ступенях), а в степеневій a^4. Для побудови поля необхідно знати, як відбувається операції додавання та множення.

Додавання двох елементів у полі $CP(2^q)$ порівняно з операцією «виключного або» (XOR) для двох векторних представлень цього поля. Наприклад, додавання елементів $a^2 + a$ та $a + 1$, векторна форма яких 110 і 011 відповідно, дасть результат $a^2 + 1$ (101 у векторній формі). Можна помітити, що однакові поліноми скорочуються, оскільки «виключне» або однакових векторів дає нульовий вектор.

Множення двох ступенів елементів дає елемент, ступінь якого дорівнює сумі ступенів множників за модулем $2^q - 1$. Наприклад, у GF(2^3), $a^2 * a^4 = a^6$, $a^5 * a^6 = a^{11} = a^4$. Ці дві операції дають збудувати поле Галуа. Варто зазначити, що після побудови поля бажано побудувати також таблиці додавання та множення.

Наприклад буде побудовано поле $GF(2^3)$ за модулем примітивного многочлена $g(x) = x^3 + x + 1$.

Так як $g(a) = a^3 + a + 1 = 0$, звідси $a^3 = a + 1$ (у GF(2^q) операція віднімання рівносильна додаванню). Для степенів нижче степеня примітивного многочлена значення залишаються такими ж $a^0 = 1$, $a^1 = a$, $a^2 = a^2$. Степінь a^3 виходить із примітивного багаточлена: $a^3 = a + 1$.

Таблиці додавання та множення складаються для степеневих представлень, тому що з ними зручніше працювати надалі. Їхня перевага в тому, що не відбувається перерахунок, що значно прискорює роботу програми. Недоліком є квадратичне зростання необхідної пам'яті зі збільшенням поля.

Далі наведено дослідження з деяких алгебраїчних кодів.

1.3 Код із перевіркою на парність

Є найпростішим кодом виявлення помилок. До інформаційного вектора додається так званий біт парності (зазвичай наприкінці). Значення біта дорівнює сумі по модулю два всіх інформаційних бітів. Таким чином, отримуємо, що

кількість одиниць у закодованому векторі завжди парна. Виникнення одиничної помилки призведе до зміни парності, що свідчить про виявлення помилки. Однак у разі двох помилок, парність не порушиться, отже, алгоритм виявити спотворення зможе. З вище сказаного можна також записати, що кодова відстань $d = 2$.

1.4 Код Хеммінга

Є лінійним кодом що самокоректується. Кодова відстань коду Хеммінга $d= 3$. Це означає, що він може виправити одну помилку або виявити 2 помилки. Будується код Хеммінга в такий спосіб. Нехай повідомлення має двійковий вигляд: 0111011001110111_2 . Тоді:

– вибирається довжина інформаційного вектора. Повідомлення ділиться на блоки вибраної довжини. Для прикладу довжина інформаційного вектора дорівнюватиме 8;

– до кожного блоку додаються перевірочні біти. Їхні позиції рівні степеням двійки. Таким чином отримуємо 4 перевірочних біти на позиції 1, 2, 4, 8. Перевірочний біт парності обчислюються наступним чином: біт, що стоїть на позиції i, є сумою i підряд бітів, що йдуть з кроком через i. Наочне представлення наведено на рис. 1.3.

Наприклад, перший блок повідомлення матиме вигляд 100011110111_2.

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
Parity bit coverage	p1	✓		✓		✓		✓		✓		✓	
	p2		✓	✓			✓	✓			✓	✓	
	p4				✓	✓	✓	✓					✓
	p8								✓	✓	✓	✓	✓

Рисунок 1.3 – Обчислення перевірочних бітів

Декодування відбувається так:

– в отриманому повідомленні перевірочні біти обчислюються заново (аналогічно). Тепер є отримані перевірочні біти та обчислені перевірочні біти. Вони порівнюються;

– якщо ці біти однакові - помилки не відбулося, або сталося 3 або більше помилок (тобто подолано кодову відстань та здійснено перехід з одного дозволеного коду до іншого). Повернення інформаційної складової коду з прапором «коректно»;

– якщо біти різні, то сталася помилка. Якщо сталося дві помилки, алгоритм повертає прапор «некоректно». Якщо відбулася 1 помилка, то позиції різних перевірочних бітів складаються. Їхнім результатом буде позиція помилки. Залишається лише інвертувати помилковий біт. У результаті повернеться виправлене повідомлення та прапор «коректно».

Наприклад, отриманий код Хеммінга для першого блоку повідомлення спотворився в биті з позицією 6: 100110100110_2. Повторно обчисливши перевірочні біти, отримаємо код: 110010100110_2 (різні перевірочні біти 2, 4). Відповідно, підсумувавши різні біти, знаходиться помилковий біт – під номером 6.

1.5 CRC

Ґрунтується на властивостях ділення на багаточлен, що породжує, із залишком у кінцевому полі GF(2).

Алгоритм обчислення значення циклічного надлишкового коду наступний:

– вибирається породжуючий багаточлен $g(x)$. Багато протоколів, що використовують CRC, мають стандартизовані багаточлени;

– обчислюється залишок від ділення [2, 3] $CRC(x) = u(x)\ x^n\ (mod\ g(x))$, де n - ступінь багаточлена, що породжує. У двійковому поданні тут до повідомлення приписується n нулів праворуч, а потім виконується ділення на багаточлен, що породжує, в двійковій формі;

– повідомлення, що повертається $v(x) = u(x)x^n + CRC(x)$.

Алгоритм декодування є обчисленням залишку на породжуючий

багаточлен. Якщо залишок дорівнює нулю, то повідомлення доставлено правильно, і потрібно повернути інформаційну частину. Якщо повідомлення доставлено неправильно, алгоритм просто виводить помилку. CRC не може виправляти помилки. Тому кодову відстань для нього описати не можна. Однак, можна сказати наступне про специфікацію алгоритму CRC-16: він може виявляти всі поодинокі, подвійні помилки, непарну кількість помилок і всі пакетні помилки (burst errors) завдовжки менше, ніж 16 біт. Загальний відсоток помилок 99,9984% [4].

Варто навести приклад роботи алгоритму. Нехай повідомлення "Hi!" = $010010000110100100100001_2$. Тоді:

– багаточлен, що породжує наступний: $g(x) = x^{16} + x^{12} + x^5 + 1 = 10001000000100001_2$;

– обчислений залишок від ділення 0011000111111101_2 - записаний у двійковому вигляді;

– вихідний код =$010010000110100100100001001100011111111101_2$.

Нехай трапилися помилки у таких місцях: 16, 20, 22, 26, 27. Залишок від ділення на багаточлен, що породжує, в такому випадку дорівнює - 0000000011000010_2 , що свідчить про наявність помилки.

1.6 Код Ріда-Соломона

РС коди є кодами Боуза-Чоудхурі-Хоквінгема з максимально досяжною кодовою відстанню.

Кодами Боуза-Чоудхурі-Хоквінгема є коди, породжуючий багаточлен яких своїм корінням має послідовність підряд степенів деякого елемента a кінцевого поля $GF(q^m)$ a^b , a^{b+1} , ..., $a^{b+\delta-2}$.

Максимально досяжною кодовою відстанню називають лінійний код, для мінімальної відстані якого виконується співвідношення [1]: $d = n - k + 1$, де $n = q^m - 1$, k - довжина інформаційного вектора. Інакше кажучи, цей код лежить на так званому кордоні Сінглтона.

Багаточлен коду РС, що породжує, обчислюється наступним чином:

$$g(x) = (x - a^b)(x - a^{b+1}) \dots (x - a^{b+d-2})$$

Було реалізовано код РС в кінцевому полі $GF(2)$.

Кодування РС відбувається аналогічно до обчислення коду CRC. Однак усі операції проводяться в кінцевому полі $GF(2^q)$. Тому всі функції слід реалізовувати вручну.

Алгоритм кодування:

– помножити інформаційний вектор $u(x)$ довжини n на x^d;

– поділити з залишком отримане повідомлення на багаточлен, що породжує $g(x)$: $r(x) = u(x)x^d \bmod g(x)$;

– додати отриманий залишок $r(x)$ до повідомлення $v(x) = u(x)x^n + r(x)$.

Наприклад, нехай для (n=15, k = 9) РС коду з багаточленом, що породжує, $g(x) = x^6 + a^{10}x^5 + a^{14}x^4 + a^4x^3 + a^6x^2 + a^9x + a^6$.

Інформаційний поліном - $u(x) = x^8 + a^4x^7 + a^5x^6 + a^{10}x^5 + a^7x^4 + a^{13}x^3 + a^{22} + a^{12}x + a^{11}$.

1) помножене на x^6 повідомлення $u(x)x^6 = x^{14} + a^4x^{13} + a^5x^{12} + a^{10}x^{11} + a^7x^{10} + a^{13}x^9 + a^2x^8 + a^{12}x^7 + a^{11}x^6$,

2) залишок від розподілу $r(x) = ax^5 + a^5x^4 + a^5x^3 + a^9x^2 + a^{13}x + a^{11}$,

3) закодоване повідомлення $v(x) = x^{14} + a^4x^{13} + a^5x^{12} + a^{10}x^{11} + a^7x^{10} + a^{13}x^9 + a^2x^8 + a^{12}x^7 + a^{11}x^6 + ax^5 + a^5x^4 + a^5x^3 + a^9x^2 + a^{13}x + a^{11}$.

Операції додавання, віднімання, множення, ділення відбуваються в кінцевому полі $GF(2^4)$.

Однак кодування є простою частиною. Основна складність полягає у декодуванні.

Абстрактно алгоритм декодування виглядає так:

– обчислити так звані синдроми. Якщо всі синдроми дорівнюють нулю – повідомлення закодоване коректно. Повернути інформаційну частину;

– обчислити локатор помилки. Якщо степінь багаточлена локатора більше t (з $d - 1 = 2t$), то сталося більше t помилок і вектор помилок не може бути знайдено. Повернути лише виявлення;

– за коренями локатора помилки визначити розташування помилок;

– обчислити значення помилок та сформувати вектор помилки за розташуваннями та значеннями;

– відняти з закодованого вектора з помилкою та отриманий вектор помилок (можна додати, оскільки в полі $GF(2^q)$ операція додавання рівносильна відніманню). Повернути виправлення та виявлення.

Для декодування можна використати різні алгоритми [5, 6]. Для обчислення локатора можна використовувати ПГЦ-метод, БМ-метод, алгоритм Сугіями та ін. ПГЦ метод ґрунтується на обчисленні зворотної матриці. Таким чином, він є найпростішим серед інших у реалізації, але повільним (складність обчислення зворотної матриці методом Крамера $O(n^3))$. Алгоритм Берлекемпа - Мессі складніше у реалізації, але працює досить швидко (складність $O(n^2))$. Алгоритм Сугіями ґрунтується на використанні розширеного алгоритму Евкліда для вирішення системи рівнянь. Досить складний у реалізації, але має квадратичну складність.

1.7 Функція хешування SHA256

Функція хешування - спеціальна функція, що перетворює набір даних на вході якої-небудь довжини у рядок на виході фіксованої довжини за певним алгоритмом [7]. Ефективність використання хеш-функцій у тому, що при найменшій зміні вхідних даних, вихідний рядок (дайджест) хеш-функції змінюються повністю. Це дозволяє виявляти практично будь-які помилки в коді. Однак можуть виникати випадки, коли більш ніж один масив вхідних даних перетворюється на один дайджест. Такий випадок називається колізією. У цій роботі буде розглянуто функцію хешування SHA256.

Алгоритм хешування SHA256 наступний [8]:

1) вхідні дані перетворюються на двійковий рядок;

2) до отриманого рядка на кінець приєднується одиниця;

3) далі рядок заповнюється нулями, поки його довжина стане кратна 448 бітам;

4) останні 64 біта записується довжина вхідного повідомлення. Таким чином, максимальна довжина повідомлення дорівнює 264 - 1, що досить багато;

5) ініціалізація констант. Створюються 8 значень $h_0,\ldots\ h_7$, що є 32 бітами дробових частин квадратного коріння перших 8 простих чисел (рис. 1.4). Також створюються 64 констант k_t , які вже є 32 бітами частин після коми коренів кубічних з перших 64 простих чисел (рис. 1.5);

```
h0 := 0x6a09e667
h1 := 0xbb67ae85
h2 := 0x3c6ef372
h3 := 0xa54ff53a
h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd19
```

Рисунок 1.4 – Константи h_i

6) вхідний рядок ділиться на блоки по 512 біт. Наступні кроки виконуються для кожного блоку;

7) блок розбивається на 16 слів, розмір кожного з яких дорівнює 32 біт. Створюється список *W,* званий чергою повідомлень, розміру *64* x 32, у перші 16 місць якого записується отримані раніше слова:

```
0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
```

Рисунок 1.5 – Константи *k*

1.8 Імітовставки. Алгоритм HMAC

Імітовставки [9] забезпечують імітозахову (захист від нав'язування хибних даних) у протоколах аутентифікації з учасниками, що довіряють один одному.

Для перевірки цілісності та автентичності до повідомлення прикріплюється імітовставка, вироблена секретним ключем, відомим лише відправником та одержувачем. У цій роботі використовується алгоритм HMAC, що застосовує хеш-функцію 8HA256 для вироблення імітовставки.

Алгоритм HMAC наступний [10]. На вхід подаються повідомлення m і ключ k у вигляді двійкового рядка. Для функції хешування SHA256 використовуються такі константи:

- розмір блоку в байтах $b = 64$ (у SHA256 у циклі використовуються блоки по 512 біт = 64 байти),
- розмір дайджесту в байтах $L = 32$ (256 біт),

Далі йдуть кроки алгоритму:

1) зробити довжину ключа, що дорівнює b. Якщо довжина ключа менша від b, необхідно заповнити ключ нулями праворуч. Якщо довжина ключа дорівнює b, залишити ключ незмінним. Якщо ж довжина ключа більша за b, то необхідно обчислити дайджест ключа (тим самим алгоритмом хешування), а потім доповнити ключ нулями праворуч до розміру b;

2) виконати побітове виключне або (XOR) з магічними блоками $ipad$, $opad$.. Перший блок має вигляд (0x36 0x36...0x36), де байт 0x36 повторюється b разів. Другий блок містить у собі повторювану b разів константу 0x5c. Вихідними даними будуть: $Sipad = k\ XOR\ ipad$, $Sopad = k\ XOR\ opad$;

3) обчислити такі дайджести:

$$H_1 = Hash(S_{ipad}|m),$$

$$HMAC = Hash(S_{opad}|H_1);$$

4) повернути $HMAC$.

Також цей алгоритм наочно зображено на рис. 1.6.

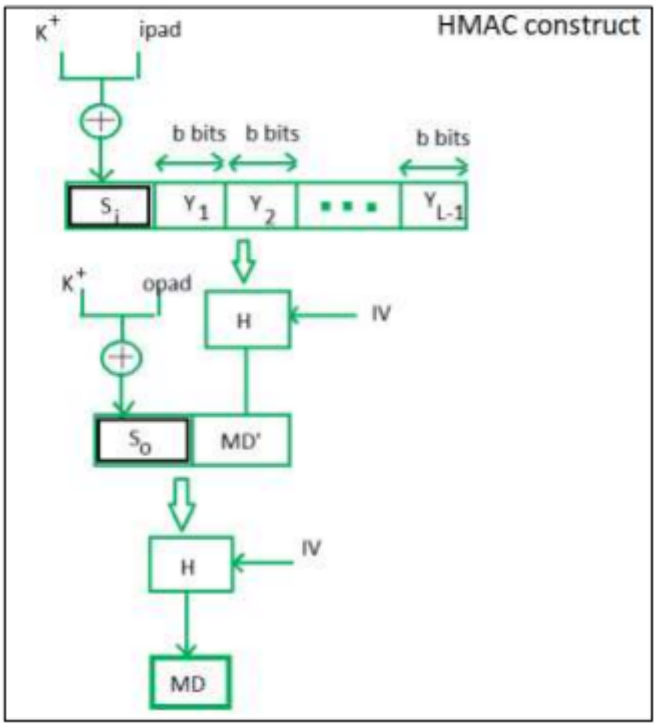

Рисунок 1.6 – Алгоритм HMAC

1.9 ЕЦП. RSA

ЕЦП [11] дозволяє підтвердити авторство повідомлення чи документа. Для цього використовуються асиметричні криптографічні алгоритми. Основна відмінність від імітовставки в тому, що підписане повідомлення або документ може перевірити на справжність будь-який охочий за наявності у нього публічного ключа (у HMAC ж перевірити на справжність можуть тільки сторони, які сформували секретний ключ). Це корисно при викладанні документів, файлів, дистрибутивів на громадський файлообмінник. Для того, щоб користувач, який завантажив файл, міг перевірити справжність, достатньо, наприклад, обчислити власне хеш-значення файлу і порівняти зі значенням, розшифрованим публічним ключем. У цій роботі обчислюватиметься ЕЦП від дайджеста хеш-функції SHA256 за допомогою пари ключів, сформованих алгоритмом RSA. Для цього слід розібратися з цим алгоритмом.

RSA [12] є алгоритмом із відкритим ключем. Його криптографічна стійкість обумовлена обчислювальною складністю факторизації великих напівпростих чисел [12, 13] - на даний момент немає ефективного алгоритму,

здатного факторизувати великі числа, що складаються з двох великих простих множників. Алгоритм формування пари ключів RSA наступний [13]:

1) генеруються два великих простих числа p і q заданого в бітах розміру (наприклад, по 256 біт кожне). Варто також відзначити, що на даний момент не існує універсального, поліноміального, детермінованого і безумовного тесту простоти, що реалізується в практиці (тобто відповідно, він реалізуємо в поточних апаратних засобах; він застосовний до всіх чисел, складність його - О-велике від деякого полінома; він визначає точно, а не з деякою ймовірністю; Тому для генерації простих чисел використовують імовірнісні тести. Один із найпопулярніших тестів – тест простоти Міллера-Рабіна [13, 14]. Його ідея полягає у перевірці деяких рівностей, що виконуються для простих чисел;

2) обчислюється добуток $n = p * q$. Він називається модулем;

3) обчислюється функція Ейлера $\phi(n) = (p - 1) * (q - 1)$;

4) вибирається число e, взаємне просте із числом $\phi(n)$;

5) обчислюється число d, мультиплікативно зворотне до e за модулем $\phi(n)$, тобто $d * e = mod\ \phi(n)$. Мультиплікативно зворотне число обчислюється за допомогою розширеного алгоритму Евкліда [13], який знаходить для чисел A, знаходить НСД(A,B) і такі числа x, y, що $Ax + By = НСД(A, B)$. Таким чином, підставивши замість чисел A,B числа $e,\phi(n)$ відповідно, буде знайдено x,y рівняння $ex + \phi(n)y = 1$. Взявши модуль $\phi(n)$, вийде $ex = 1\ mod\ \phi(n)$, що і потрібно знайти;

6) пара (e, n) утворює відкритий ключ;

7) пара (d, m) формує закритий ключ.

Далі можна перейти до алгоритму формування цифрового підпису за допомогою RSA та SHA256. На вхід дано повідомлення m. Необхідно сформувати кілька ключів (e, m) та (d,n). Після формування пар потрібно виконати наступні кроки:

1) надіслати публічний ключ до загального доступу або одержувачам;

2) обчислити дайджест повідомлення за допомогою алгоритму SHA256. Отриманий дайджест - число розміру 256 біт: $h = SHA256(m)_{числ}$;

3) за допомогою секретного ключа (d, m) зашифрувати число-дайджест повідомлення $c = h^d \ mod \ n;$

4) надіслати пару повідомлення-підпис (m, c) у спільний доступ або одержувачам.

Алгоритм перевірки підпису наступний:

1) обчислити дайджест отриманого повідомлення m' за допомогою тієї ж функції хешування SHA256: $h' = 5HA256(m')_{числ}$;

2) розшифрувати за допомогою публічного ключа (e,n) отриманий підпис повідомлення $h'' = c^e \ mod \ n = h^{de} \ modn = h$;

3) якщо повідомлення не змінено, $h' = h''$, і підпис вірний. Якщо повідомлення змінено, то підпис неправильний.

Варто зазначити, що не слід використовувати модуль n для підпису менший або близький до 256 біт, оскільки шифрування числа більшого модуля призведе до некоректного дешифрування

1.10 Висновки до першого розділу

Проаналізовано основні поняття предметної області, зокрема поля Галуа, алгебраїчні коди (з перевіркою на парність, Хеммінга, CRC, Ріда-Соломона).

Також досліджено особливості функції хешування SHA256, розглянуто імітовставки із алгоритмом HMAC. Описано застосування алгоритму RSA для ЕЦП.

2 РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ ТА АНАЛІЗ АЛГОРИТМІВ

2.1 Реалізація алгоритмів

Були реалізовані алгоритми, досліджені у першому розділі: код з перевіркою на парність, код Хеммінга, циклічний надлишковий код, код Ріда-Соломона, хеш-функція SHA256, алгоритм імітовставки HMAC-SHA256, алгоритм ЕЦП на базі RSA.

Для реалізації використано Python. Це обумовлено її зручністю та великою наявністю загальнодоступних бібліотек (таких як numpy, matplotlib, tkinter та ін), що спрощують реалізацію алгоритмів. Для простоти жертвується швидкістю роботи мови, чистий python -код (тобто код без використання бібліотек, тому що часто вони написані мовою C або C++) працює повільніше порівняно з компілюваними мовами у кілька десятків разів. Однією з переваг цієї мови у цій роботі у тому, що тип цілого числа в Python 3 необмежений (в C++ максимальне натуральне число дає тип unsigned long long рівний 2^{64}). Це дуже корисно, тому що при використанні криптографічних алгоритмів потрібно використання великих чисел, близько 2^{1024} і більше для сучасних алгоритмів.

Нижче наведено відомості про методи, реалізовані в даних алгоритмах.

2.1.1 Реалізація алгоритмів кодів алгебри

У коді з перевіркою на парність було реалізовано клас «ParityCode» з двома статичними методами (які не мають прив'язки до екземпляра класу) кодування та декодування. У функції кодування до вихідного повідомлення до кінця додається підрахована сума по модулю два. У декодуванні підрахована сума порівнюється з нулем (у коректного коду, як було сказано раніше, парне число одиниць); далі виводиться коректність чи некоректність отриманого коду.

У коді Хеммінга були реалізовані класи «hamming8» та «Encoding». У першому класі описано сам код Хеммінгу. Він містить у собі такі методи:

– підрахунок перевірочних бітів (допоміжний метод);
– кодування. Є вставкою та підрахунком перевірочних бітів;

– декодування. Здійснює перерахунок перевірочних бітів. За наявності однієї помилки, виправляє;

– функція кольорового виводу для наочнішого виведення в консоль.

Фрагмент коду класу hamming8 наведено на рис. 2.1.

```
class hamming8:
    @staticmethod
    def calculate_check_bits(code):
        c0 = (code[2] + code[4] + code[6] + code[8] +
code[10]) % 2
        c1 = (code[2] + code[5] + code[6] + code[9] +
code[10]) % 2
        c3 = (code[4] + code[5] + code[6] + code[11]) % 2
        c7 = (code[8] + code[9] + code[10] + code[11]) % 2
        c12 = np.sum(code[:12]) % 2
        return c0, c1, c3, c7, c12

    @staticmethod
    def encode(array: np.ndarray):
        if array.shape[0] != 8:
            raise Exception("False Length of an array")
        code = np.zeros(13, dtype=int)
        code[[2, 4, 5, 6, 8, 9, 10, 11]] = array
        print(code)
        c0,       c1,       c3,       c7,       c12       =
hamming8.calculate_check_bits(code)
        code[0], code[1], code[3], code[7] = c0, c1, c3, c7
        print(f"c1 = {c0}, c2 = {c1}, c4 = {c3}, c8 = {c7},
c13 = {c12}")
        code[12] = c12
        hamming8.colored_print(code)
        return code
```

Рисунок 2.1 – Фрагмент лістингу коду класу hamming8

У другому класі для простоти реалізації перераховано словник, що зберігає пари <символ, ASCII код>.

У циклічному надлишковому коді був реалізований клас «CRC», конструктор якого приймає ступінь багаточлена, що породжує, і сам породжує багаточлен. Реалізовано метод «append», який здійснює обчислення згортки за алгоритмом обчислення циклічного надлишкового коду. Як ділення за залишком використовується метод математичної бібліотеки «numpy». Метод «check» перевіряє поданий на вхід бінарний масив, виконуючи ділення із залишком, і

порівнюючи залишок, що вийшов, з нулем. Також реалізовано метод кольорового виведення.

Реалізована програма коду Ріда-Соломона складається з реалізованих модулів "GalouisField", "rs_code" та "main".

Модуль "GalouisField" містить у собі визначення класу поля Галуа. Конструктор цього поля приймає на вхід ступінь q поля Галуа $GF(2^q)$ та примітивний багаточлен. У конструкторі відбувається розрахунок таблиці степенів, таблиці множення, ділення, додавання (віднімання, як згадувалося, рівносильне доданню поля порядку 2). Реалізовано окремі методи додавання, множення, діленн, необхідні для кращої читабельності коду. Також реалізовані методи матричного множення вектора на матрицю та матриці на вектор, що використовуються при реалізації методу ПГЦ;

Модуль "rs_code" містить у собі функціональну складову самого РС коду. На конструктор подаються параметри m, n, k коду Ріда-Соломона, а також примітивний багаточлен, навколо якого будуватиметься поле Галуа, де оперуватиме сам код. Інші параметри коду обчислюються автоматично. До класу також прив'язується екземпляр класу поля Галуа, через який забезпечується доступ до операцій у цьому полі. Реалізовано методи додавання, множення, ділення багаточленів, які працюють за стандартними алгоритмами (додавання - за відповідними степенями, множення - «фонтанчиком», ділення - алгоритмом ділення в стовпчик), проте вони оперують у полі Галуа. Також реалізовані методи обчислення степеня багаточлена, обчислення значення багаточлена заданого елемента поля. Метод кодування реалізований за алгоритмом, описаним вище. Для декодування реалізовано два методи - ПГЦ-метод та БМ-метод. Для реалізації першого потрібно також реалізувати рекурсивний алгоритм розрахунку визначника матриці, розрахунку зворотної матриці та функції взяття мінору, що оперують у полі Галуа. Реалізація другого алгоритму зажадала досить великих зусиль, зважаючи на його складність і неоднозначність формулювань у різних літературах [3,4,15, 17] - для успішної та коректної реалізації потрібно написати 3 ітерації алгоритму і витратити більше двох місяців. Нарешті, було реалізовано методи пошуку позиції помилок та значення

помилок.

У модулі "main" реалізовано вхід до програми, вибір початкових значень, генерація повідомлення, генерація помилок.

2.1.2 Реалізація алгоритмів із криптографією

Було реалізовано алгоритм хешування SHA256. Він містить у собі 4 реалізовані модулі: «binary_operations», «encoding», «SHA256», «main». Їх призначення такі [17]:

– в першому модулі реалізовані операції над двійковими рядками (векторами) - операція побітового виключного або, циклічного зсуву вправо, логічного зсуву вправо, двійкового додавання двох векторів за модулем 2^{32} , операція побітового «І», операція побітового заперечення;

– у другому модулі були реалізовані методи кодування - перетворення текстового рядка в бінарний рядок кодування UTF8, переведення цілого десяткового числа в двійкове подання у вигляді рядка, переведення рядка двійкового подання у рядок шістнадцяткового подання;

– у третьому модулі було реалізовано клас, що містить функціонал хеш-функції. При створенні екземпляра класу відбувалася ініціалізація констант. Було реалізовано метод «hash_data», який приймає вхідний текстовий рядок. У ньому відбувалося початкове скидання констант (щоб забезпечити правильне хешування за нового виклику даного методу), після чого проводився алгоритм хешування, описаний у попередньому розділі. Наприкінці отриманий результат переводився в шістнадцяткову форму і повертався;

– в останньому модулі знаходиться точка входу в програму, в якій створювався рядок і подавався на вхід функції хешування.

Для перевірки коректності алгоритму було здійснено порівняння для пробного тексту, що містить 7965 символів. Як «еталонна» реалізація використовувалася реалізація SHA256 у бібліотеці ««hashlib»». Результати наведено на рис. 2.2. На ньому зображені підсумкові значення змінних у шістнадцятковій і двійковій формі. Далі вказана їхня конкатенація - результат алгоритму SHA256. В останньому рядку вказано результат хешування

««hashlib.sha256».

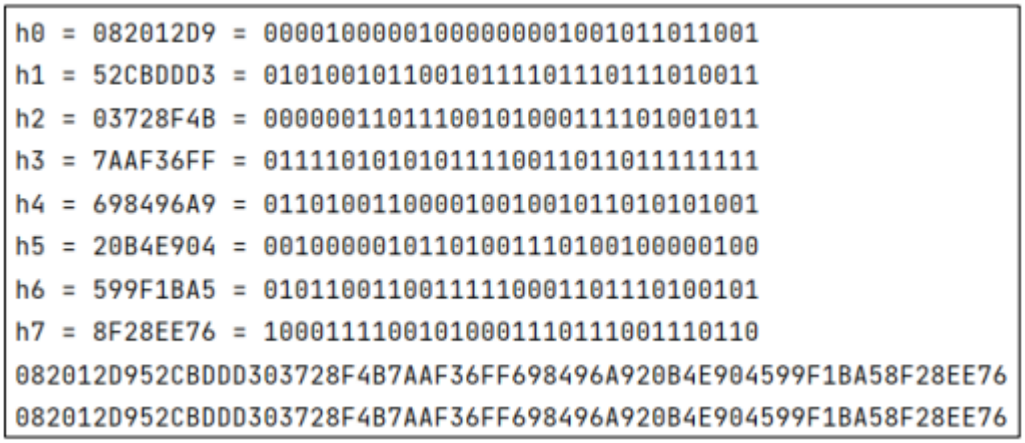

Рисунок 2.2 – Результати роботи власної реалізації SHA256 і бібліотеки "hashlib"

Далі було реалізовано алгоритм HMAC. Даний алгоритм містить у собі як пакет реалізовану вище функцію хешування SHA256. Сам же HMAC також реалізований як пакет, і містить у собі 3 модулі: «encoding», «binary_operations», «HMAC». Їх зміст такий:

– перший модуль містить у собі ті самі методи, як і SHA256, але додатково містить метод переведення байтового масиву в бінарний рядок. Він необхідний для переведення байтових блоків ipad та opad у бінарний рядок;

– другий модуль містить лише операцію XOR;

– третій модуль містить функціонал HMAC. У ньому реалізовано метод, що обчислює імітовставку та допоміжні методи: заповнення праворуч нулями, поділ на блоки, конкатенація, друк у шістнадцятковому форматі.

Як і з функцією SHA256, для перевірки коректності реалізованого HMAC було проведено порівняння з бібліотечним методом (hashlib). Було згенеровано випадковий ключ із 64 символів, а також випадковий текст із 10 тисяч символів. Результати виконання програм наведено на рис. 2.3. Як видно результат виконання ідентичний.

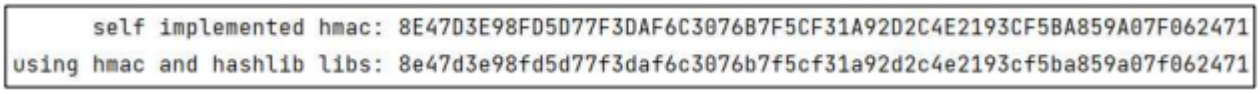

Рисунок 2.3 – Результати обчислень власної реалізації HMAC та бібліотечної

Нарешті, було реалізовано алгоритм RSA, з допомогою якого відбувається обчислення ЕЦП. Дана реалізація містить 2 пакети: cryptography, в якій міститься реалізація RSA та "hashing", в якій міститься хеш-функція SHA256. Пакет RSA містить:

– модуль «py_prime_utils», у якому реалізовані функції: алгоритм Міллера – Рабіна, генерація простих чисел (перебір та тест на простоту), розширений алгоритм Евкліда;

– модуль RSA містить клас RSA. Під час створення класу у ньому генеруються прості числа, обчислюються значення $n,\ \phi(n),e,d$. Доступ до ключів здійснюється через метод export_parameters, в аргументи якого приймається булеве значення public, що має значення за умовчанням істина. Таким чином, без вказівки параметра, буде повернуто відкритий ключ. Далі реалізовані статичні методи шифрування та дешифрування цілого числа, які приймають аргументами число та відповідний ключ. Також було реалізовано алгоритми підпису та перевірки підпису.

Таким чином, є 7 реалізованих алгоритмів. Далі буде проводитись тестування та аналіз алгоритмів.

2.2 Тестування та проведення аналізу алгоритмів

У підрозділі буде розглянуто тестування, проведене для реалізованих алгоритмів. Для проведення тестування необхідно реалізувати методи, що виконують тестування.

Для коду з перевіркою на парність, коду Хеммінга і CRC коду помилки, що відбуваються в каналі, відтворюватимуться у близькій відповідності до моделі двійкового симетричного каналу зв'язку [1], тобто кожен біт повідомлення з ймовірністю p (зазвичай досить малою) інвертується, і з ймовірністю $q = 1 - p$ залишається незмінним.

Для коду Ріда-Соломона відбуватиметься спотворення не одного біта, а одного елемента поля в цілому (генеруватиметься випадковий вектор помилок).

Для алгоритмів SHA256, HMAC-SHA256 та ЕЦП с RSA для простоти реалізації тестів будуть проводитись зміни символів повідомлення.

Таким чином, будуть отримані "емпірично" обчислені характеристики, що підтверджують теоретичні характеристики коду. Додатково буде обчислено тимчасові характеристики алгоритмів.

2.2.1 Код із перевіркою на парність

Для тестування коду з перевіркою було реалізовано метод test_detection, вхідними параметрами якого є:

– length - довжина повідомлення, що генерується. Значення за умовчанням дорівнює 50;

– times - кількість ітерацій тестування (за замовчуванням це 100 000);

– error_prob - ймовірність помилки p. Значення за умовчанням дорівнює 0.2.

Лістинг фрагменту програмного коду наведена на рис. 2.4.

```
def test_detection(length=50, times=100000, error_prob=0.2):
    detected = 0
    error_hist = np.zeros(length + 1, dtype=int)
    odd_error_count = 0
    for i in range(times):
        vector = np.random.randint(0, 2, length)
        # кодирование вектора
        encoded_vector = ParityCode.encode(vector)
        # добавление ошибок
        error_count = 0
        error_vector = np.zeros(encoded_vector.shape[0])
        for i in range(error_vector.shape[0]):
            if np.random.rand() < error_prob:
                error_vector[i] = 1
                error_count += 1
        error_hist[error_count] += 1
        if error_count % 2 == 1:
```

Рисунок 2.4 – Фрагмент лістингу коду з перевіркою на парність

Метод виводить кількість виявлених помилок, а також гістограму розподілу числа помилок (яка, очевидно, має біноміальний розподіл [18]).

Висновок методу значення за замовчуванням наступний:

– гістограма розподілу помилок представлена на рис. 2.5,

– кількість непарних помилок – 50035 (50,03% випадків),

– помилок виявлено – 50035 (50,03% випадків).

Як і очікувалося, код з перевіркою на парність виявлятиме лише непарну кількість помилок - алгоритм виявив усі 50035 помилок. Таким чином перевірено коректність реалізованого коду.

За гістограмою помилок можна також зробити невеликий висновок, що виникнення дуже великої кількості помилок дуже малоймовірно. У цьому тестуванні ймовірність помилки дорівнює 0.2 чи 20%, що дуже багато.

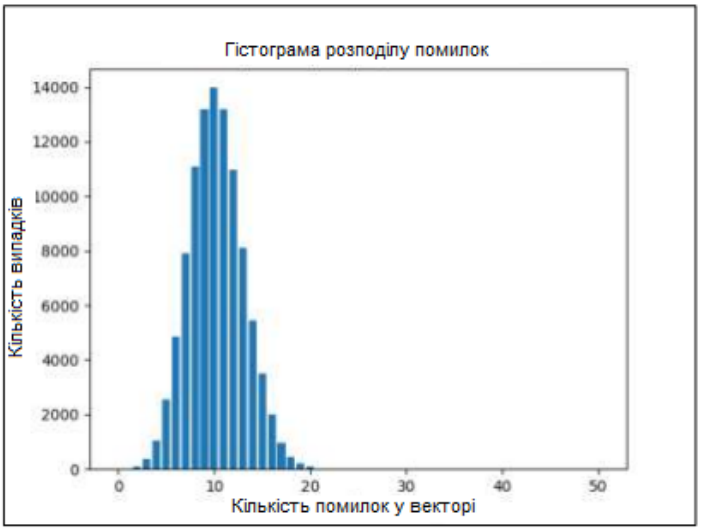

Рисунок 2.5 – Гістограма розподілу помилок коду з перевіркою на парність

Додатково було обчислено швидкості роботи алгоритму при кодуванні та декодуванні від довжини повідомлення в бітах. Графік залежності представлений на рис. 2.6. Видно, що, як і очікувалося, швидкість алгоритму лінійна, оскільки алгоритму досить пройтися по вхідному вектору лише один раз.

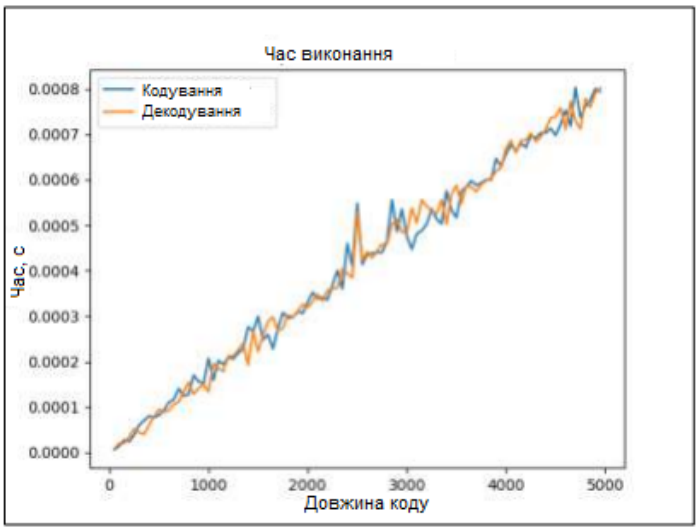

Рисунок 2.6 – Графік залежності швидкості кодування та декодування від довжини вектора

2.2.2 Код Хеммінга

Аналогічно було проведено тестування коду Хеммінга. Його результати наступні (кількість тестів – 100000):

– гістограма розподілу помилок наведена на рис. 2.7,
– кількість одиночних помилок – 17948 = 0,18% випадків,
– помилок виявлено – 94920 = 94.92% випадків,
– помилок коректно виправлено – 27893 = 27.89% випадків.

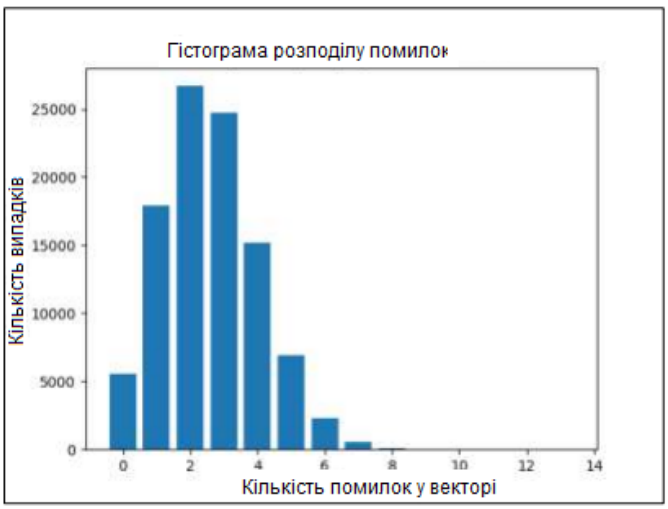

Рисунок 2.7 – Гістограма розподілу помилок коду Хеммінга

Варто зауважити, що кількість випадків коректно виправлених помилок

більша, ніж кількість одиночних помилок. Це зумовлено не малоймовірним випадком - помилки виникли в перевірочних бітах, і код Хеммінга також виправив перевірочний біт. Таким чином, інформаційні біти, що повертаються, не були доторкані, і повернулися коректними. Алгоритм виявив та виправив усі поодинокі помилки. Отже, він працює коректно.

Також варто відзначити, що код виявив практично 95% від усіх помилок, що є досить добрим показником.

Нижче наведено час виконання залежно від довжини повідомлення (рис. 2.8). Очевидно, що час виконання також лінійний, тому що тепер довжина залежить від кількості блоків. Також можна помітити, що час декодування з однією помилкою більше, ніж за інших. Це пояснюється додатковими обчисленнями для виявлення помилки.

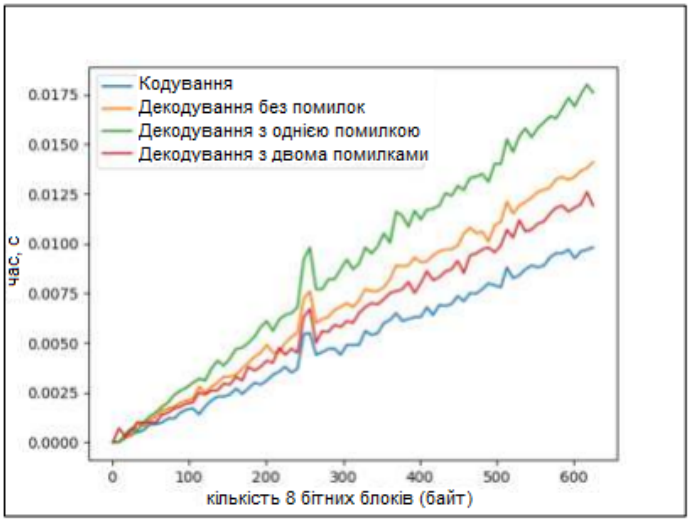

Рисунок 2.8 – Графік залежності часу кодування та декодування коду Хеммінга

2.2.3 CRC

Для цього випадку були використані такі параметри: розмір повідомлення - 32 біт, породжуючий багаточлен $g(x) = x^{16} + x^{12} + x^5 + 1$.

Результати отримані такі:

– гістограма розподілу помилок показана на рис. 2.9;

– кількість векторів без помилок - 3 випадки;

– кількість виявлених помилок – 99994 випадки.

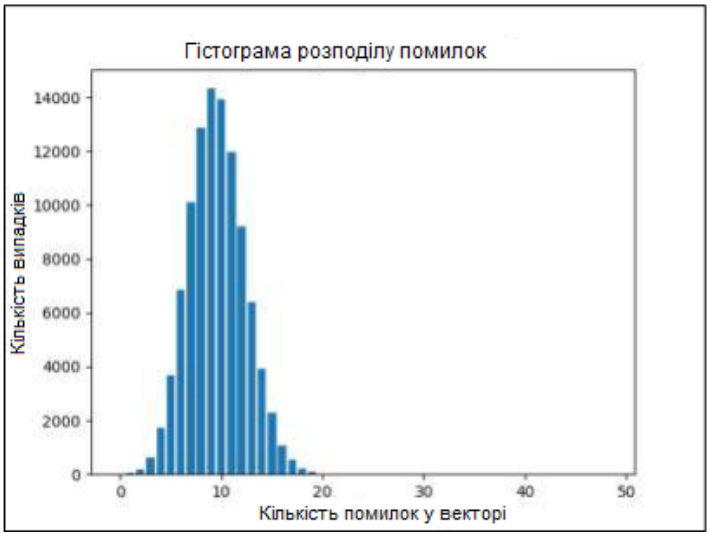

Рисунок 2.9 – Гістограма розподілу помилок CRC коду

Таким чином, було виявлено 99994/99997 = 99.997% випадків помилок, що сходиться з описаними раніше даними. Отже алгоритм працює коректно.

На рис. 2.10 наведено графік залежності часу виконання від довжини повідомлення (синім кольором відмічено кодування, помаранчевим – декодування, зеленим – декодування за наявності від 0 до 5 помилок у кожному блоці).

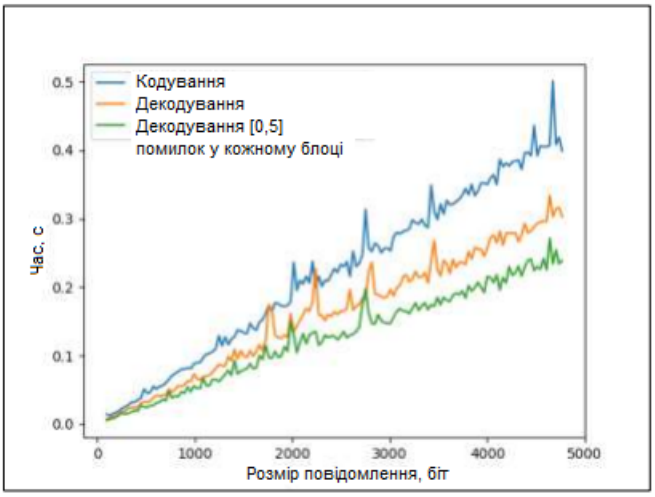

Рисунок 2.10 – Графік залежності часу виконання CRC від розміру повідомлення

Повідомлення ділиться на блоки по 32 біти. Графік також лінійний, оскільки кодування виконується блоками. Також варто зауважити, що час декодування за наявності помилок трохи нижчий. Це пояснюється тим, що

перевірка того, що залишок дорівнює нулю поверне «false» відразу ж за ненульовому значенні елемента (np.all(масив= = умова) відразу ж повертає «false» за першої невідповідності елемента умові). Таким чином нульовий залишок (масив з нулів) перевірятиметься до кінця, що значно відображається у графіку при великому розмірі вхідного повідомлення.

2.2.4 Код Ріда - Соломона

Тести проводилися для (15, 9) РС коду з багаточленом, що породжує $g(x) = x^4 + x + 1$. У висновку були додані додаткові параметри: помилок менше t, де $d = 2t + 1$; помилок менше t виправлено. Результати вийшли такими:

- гістограма розподілу помилок представлена на рис. 2.11;
- помилок всього - 96492/100000 = 96,49% випадків;
- помилок виявлено - 96492/100000 = 96.49% випадків;
- помилок менше t - 64837/100000 = 64,84% випадків;
- помилок менше t виправлено – 64837/64837 = 100% випадків.

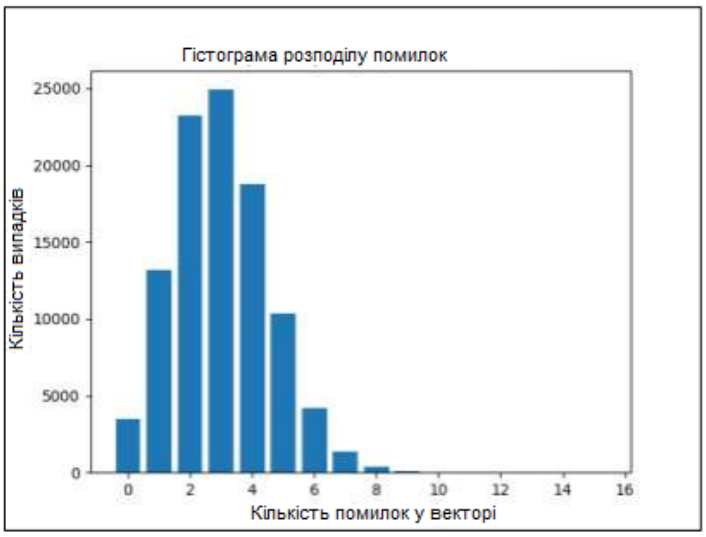

Рисунок 2.11 – Гістограма розподілу помилок для (15, 9) РС коду

Як видно, були виправлені всі помилки менше t, отже алгоритм працює коректно.

Також було отримано графіки виконання різних частин алгоритму РС коду. На рис. 2.12 представлено час декодування для методів БМ та ПГЦ.

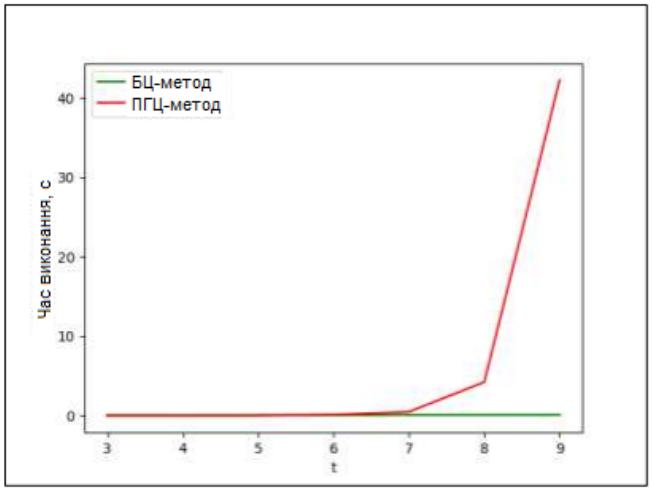

Рисунок 2.12 – Порівняння залежності часу декодування БМ-методу та ПГЦ-метода від параметра t РС коду

Як і очікувалося, ПГЦ-метод почав дуже швидко зростати за великого розміру системи рівнянь. Таке велике зростання обумовлено також тим, що весь алгоритм написаний на чистому Python, що сильно сповільнило його роботу. БМ-метод навіть при великому значенні t виконується менше секунди.

На рис. 2.13 представлено час виконання кодування, декодування без помилок, декодування з помилками t (15, 9) коду Ріда-Соломона.

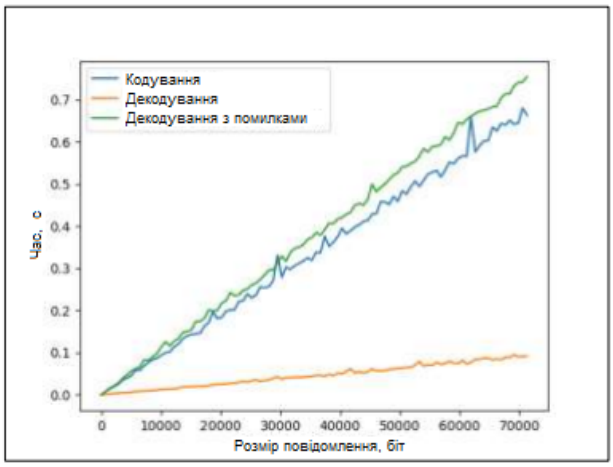

Рисунок 2.13 – Графік залежності часу виконання (15, 9) РС коду за різної довжини повідомлення

Очевидно, якщо помилок не спостерігається, то декодування відбуватиметься дуже швидко. Однак за наявності помилок доведеться

проводити повну процедуру декодування. Слід зазначити, що зручніше використовувати РС код над полем $GF(2^8)$, щоб можна було кодувати байти.

На рис. 2.14 представлено час роботи (255, 249) коду РС. На ньому вже помітна швидкість роботи БМ-методу. Однак можна помітити, що під час кодування час впав. Швидше за все це обумовлено тим, що функція ділення з залишком також написана на чистому Python, і, як відомо, операція ділення в порівнянні з іншими операціями (оскільки вона включає всі інші операції) дуже повільна [2, 3]. Однак на думку автора, цей код можна прискорити в десятки разів, використовуючи компілювані мови програмування.

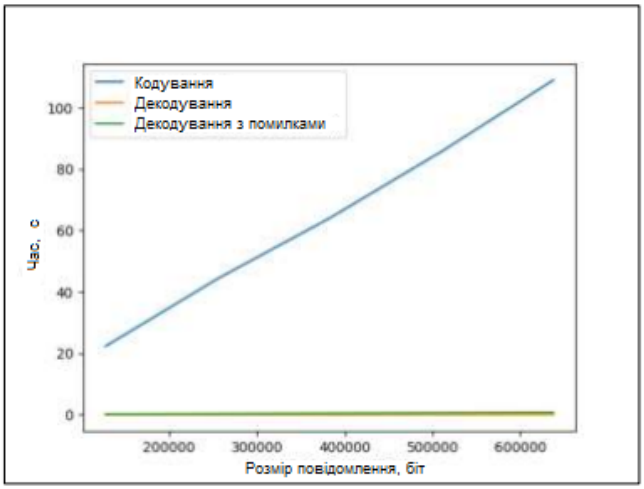

Рисунок 2.14 – Час виконання (255, 249) РС коду за різної довжини повідомлення

2.2.5 Функція хешування SHA256

У першому розділі було проведено перевірку на коректність реалізованого коду SHA256 на пробному тексті. Як згадувалося раніше, у цьому розділі буде проводитися тестування з внесенням помилок у випадково згенерований текст.

Проведено тестування для 10000 різних випадково згенерованих рядків розміром 50 символів. Можливість помилки p також дорівнює 0,2. Були отримані такі результати:

– гістограма розподілу помилок наведена на рис. 2.15;
– помилок всього 10000/10000 = 100% випадків;
– помилок виявлено: 10000/10000 = 100% випадків.

Отже, функція хешування SHA256 працює коректно.

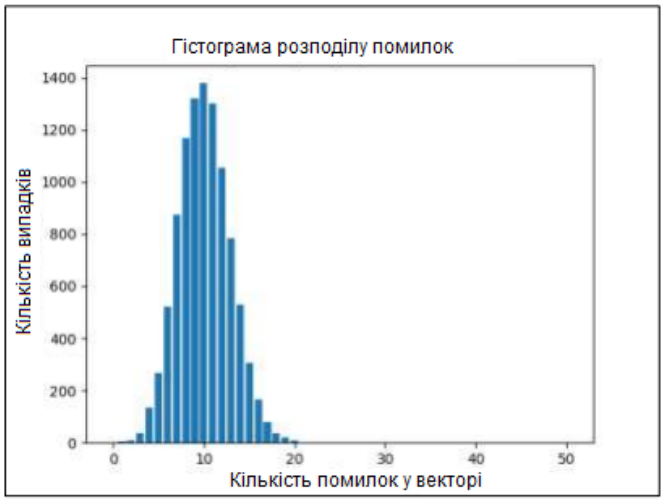

Рисунок 2.15 – Гістограма розподілу помилок SHA256

Також було отримано час виконання функції хешування від довжини повідомлення (рис. 2.16).

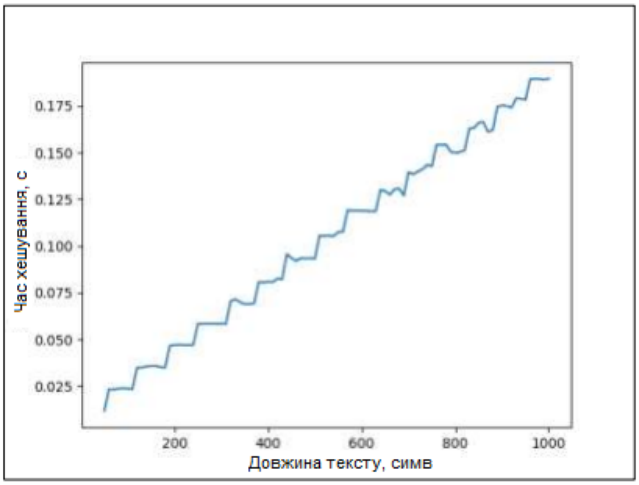

Рисунок 2.16 – Графік залежності часу хешування функції SHA 256 від довжини вхідного повідомлення

На графіку помітна ступінчастість графіка часу. Це пояснюється тим, що алгоритм хешує блоки, відповідно кожні 512 біт (тут, кожні 64 символи, зважаючи на використання кодування ASCII при реалізації виведення тимчасових графіків) час функції зросте на постійний порядок - час кодування одного блоку (був окремо проведений тест, середній час кодування одного блоку

0,012 секунд).

Функціональна частина хешування була реалізована з нуля, за винятком переведення в бінарний рядок. Таким чином, реалізований алгоритм дуже повільний, тому що виконує всі бінарні операції над бінарними рядками в циклі for. Використовуючи вбудовані бінарні операції, можна домогтися прискорення коду, проте поведінка графіка залежності хешування збережеться.

2.2.6 Алгоритм імітовставки HMAC-SHA256

Для реалізованого алгоритму імітовставки HMAC-SHA256 (далі - HMAC) були обрані наступні параметри тестування: довжина ключа - 64 символи, довжина повідомлення - 50 символів, число ітерацій - 10000, ймовірність помилки 0,2. Також було проведено експеримент із заміною ключа (у кожній ітерації). Були отримані такі результати:

– гістограма розподілу помилок показана на рис. 2.17;

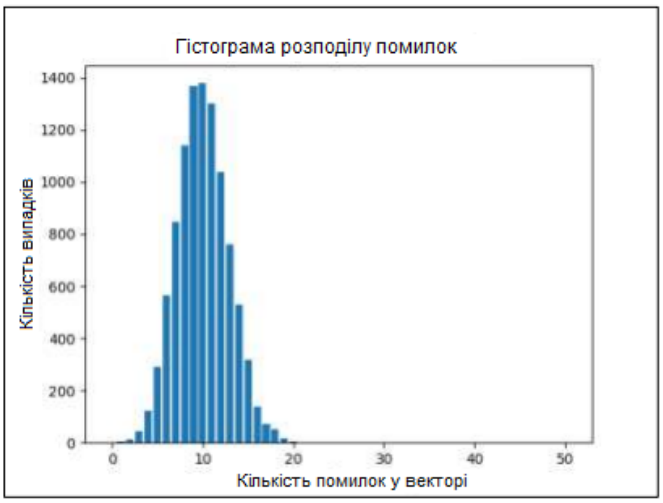

Рисунок 2.17 – Гістограма розподілу помилок HMAC

– помилок всього 10000/10000 = 100% випадків;

– помилок виявлено 10000/10000 = 100% випадків;

– випадків заміни ключа 10000;

– випадків виявлення підміни ключа (неправильний результат імітівставки) - 10000/10000 = 100% випадків.

Отже, функція імітовставки HMAC також працює коректно.

Також отримано час виконання алгоритму хешування HMAC від довжини повідомлення (рис. 2.18). Довжина ключа скрізь дорівнювала 64 символам

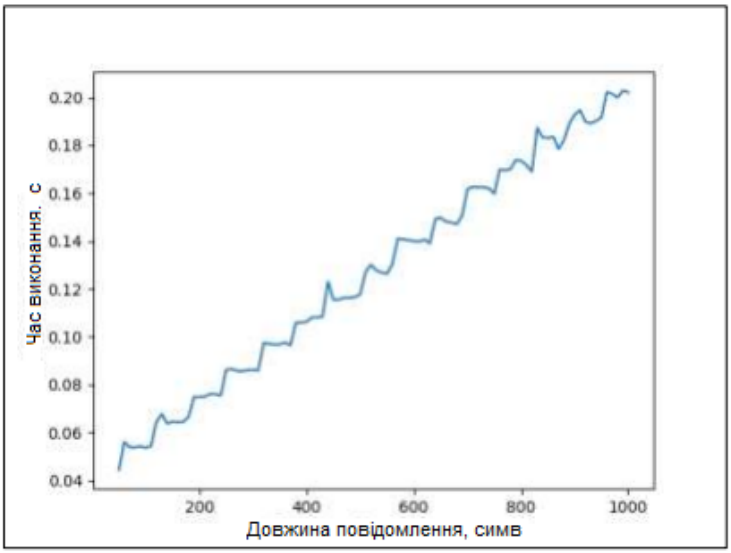

Рисунок 2.18 – Графік залежності часу виконання HMAC від довжини повідомлення

Як видно з графіка, час виконання також ступінчастий, як і у SHA256, але вище на деяке (практично постійне) число. Це пояснюється алгоритмом роботи HMAC [10]: під час його виконання функція хешування викликається двічі: розмір вхідного повідомлення для першого виклику функції хешування може змінюватись (він дорівнює сумі довжини повідомлення та довжини S_{ipad}); розмір повідомлення для другого виклику постійний (рівний сумі довжини дайджесту і S_{opad}). Таким чином, загальний час дорівнює сумі часу роботи функції хешування від динамічної довжини повідомлення, часу роботи функції хешування від постійної довжини (постійна накладка на час) та часу роботи інших (не надто витратних за часом) алгоритмів.

2.2.7 Алгоритм електронного цифрового підпису з RSA

Для реалізованого алгоритму ЕЦП з криптографічним алгоритмом обміну ключами RSA були використані такі параметри: розмір повідомлення – 50 символів, число ітерацій – 10000, ймовірність помилки – 0,05, розміри простих

чисел RSA – 256 біт. Також були проведені додаткові експерименти: експеримент із заміною ключа (у кожній ітерації), експеримент із заміною підпису (у кожній ітерації), експеримент де вірні всі параметри. Були отримані такі результати:

– гістограма розподілу помилок наведена на рис. 2.19;
– помилок всього 9276/10000 = випадків;
– помилок виявлено 9276 / 9276 = 100% випадків;
– підмін ключа 10000 випадків;
– підмін ключа виявлено 10000/10000 = 100% випадків;
– підмін підпису 10000 випадків;
– підмін підпису виявлено 10000/10000 = 100% випадків.

Отже, алгоритм ЕЦП із RSA працює коректно.

Для алгоритму цифрового підпису було проведено кілька порівнянь часу.

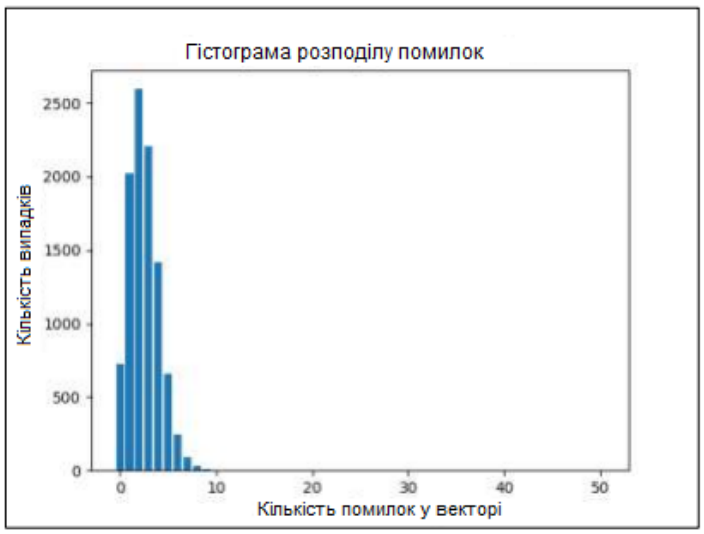

Рисунок 2.19 – Гістограма розподілу помилок в ЕЦП з RSA

Спочатку було проведено тести алгоритму генерації чисел. Результати вийшли такими. На рис. 2.20 представлений графік залежності генерації простого числа з тестом Міллера - Рабіна (кількість ітерацій тесту скрізь далі дорівнює 500) від заданої довжини.

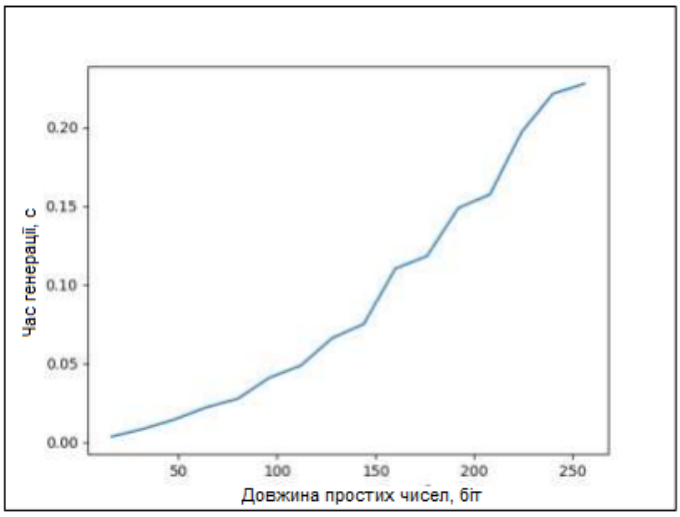

Рисунок 2.20 – Графік залежності часу генерації простого числа чистим тестом Міллера-Рабіна

На графіку видно зростання часу виконання зі збільшенням довжини повідомлень. Такий факт пояснюється зростанням розмірів чисел (математичні операції працюють довше), і навіть зростанням кількості генерацій чисел, необхідних для знаходження простого (генерації великих простих чисел займають більше часу, більше генерацій необхідно провести). На рис. 2.21 представлений графік залежності числа генерацій чисел до знаходження простого числа із тестом Міллера - Рабіна від заданої довжини. Як говорилося раніше, кількість зростає.

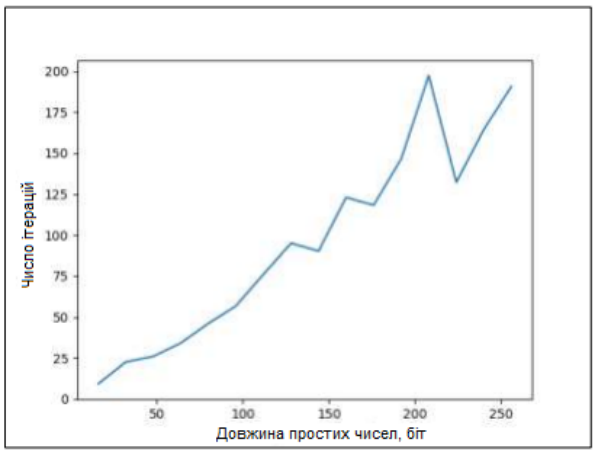

Рисунок 2.21 – Графік залежності кількості перебраних випадкових чисел до знаходження першого простого в чистому тесті Міллера - Рабіна

Тут можна зробити невелику оптимізацію, використовуючи попереднє просіювання. У ньому згенероване число перед тим як вступати до самого тесту Міллера - Рабіна проходить подільність на список заздалегідь встановлених простих чисел. Якщо число ділиться на хоча б одне з чисел із цього списку і не є ним самим, воно складне.

Були зроблені тести попереднього просіювання згенерованих чисел (перевірка подільності на невеликий масив простих чисел). Результати попереднього просіювання зображені на рис. 2.22, 2.23 (синьою лінією зображено тест без попереднього просіювання, решта вказана в легенді графіка).

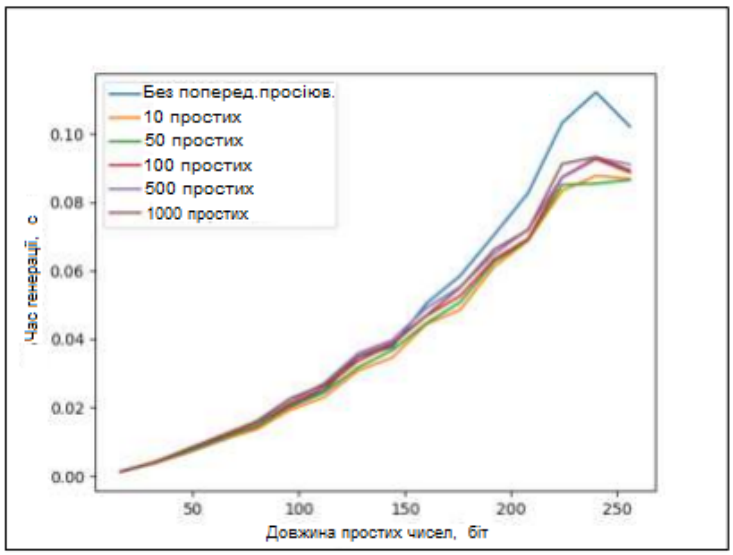

Рисунок 2.22 – Графік залежності часу виконання тестів Міллера - Рабіна з попереднім просіюванням

Час виконання зменшився при малому розмірі списку простих чисел для просіювання. Це зумовлено тим, що великі числа не дають особливого приросту до швидкості (наприклад, 997 буде відкидати кожне 997), а час проходження по масиву простих чисел збільшується. Тому оптимальним варіантом вважатимемо використання попереднього просіювання з 10-50 числами.

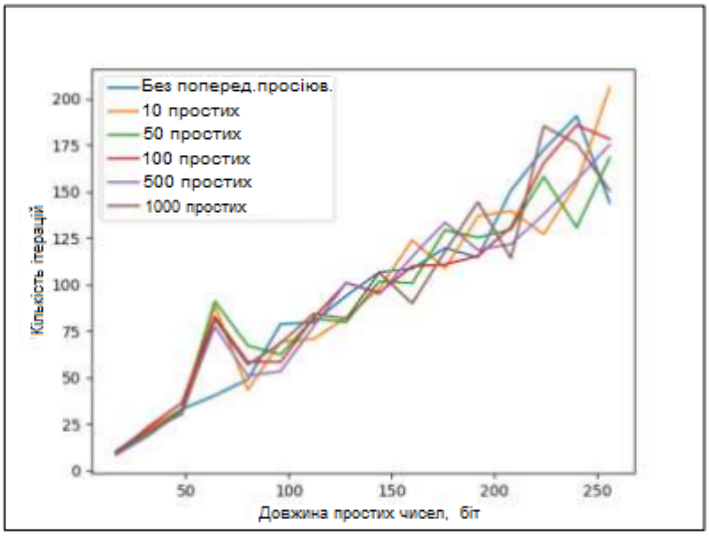

Рисунок 2.23 – Графік залежності кількості ітерацій тесту Міллера-Рабіна з попереднім просіюванням

Далі було досліджено час формування підпису та його перевірки для алгоритму RSA з ключами розміру 256 біт. Результати показані на рис. 2.24.

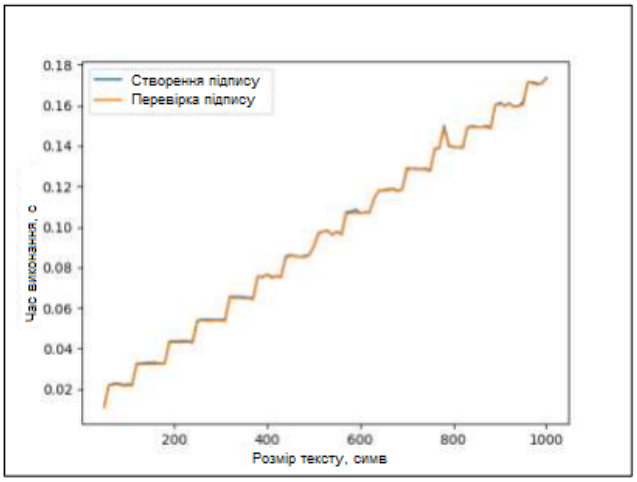

Рисунок 2.24 – Графік залежності часу створення і перевірки підпису ЕЦП з RSA

Із графіка видно, що час формування підпису і час порівняння підпису однакові, тому що для першого випадку потрібно розрахувати дайджест значення рядка і після його зашифрувати, а в другому випадку потрібно розрахувати дайджест, розшифрувати зашифроване повідомлення, отримавши оригінальний дайджест і порівняти їх. Також видно, що графік також має ступінчастий вигляд, тому що від довжини повідомлення залежить лише час

обчислення хеш-функції, а алгоритм шифрування RSA шифрує постійного розміру (256 біт).

2.3 Висновки до другого розділу

У цьому розділі було реалізовано всі алгоритми, котрі були попередньо проаналізовані у попередньому розділі, зокрема: код з перевіркою на парність, код Хеммінга, CRC, код РС, хеш-функція SHA256, алгоритм імітовставки HMAC-SHA256, алгоритм ЕЦП на базі RSA.

Всі реалізовано коди були програмно протестовані, отримані гістограми розподілу помилок, % випадків помилок - одиночних, виявлених, коректно виправлених. Побудовані графіки залежності часу виконання кодування, декодування без помилок, декодування з помилками.

Результати проведених експериментів свідчать про коректність роботи усіх алгоритмів забезпечення цілісності інформації.

3 ДОСЛІДЖЕННЯ СТІЙКОСТІ КРИПТОГРАФІЧНИХ АЛГОРИТМІВ

Варто окремо провести криптографічний аналіз алгоритмів SHA256, HMAC і RSA, оскільки звичайні тести не відображають, наскільки ефективними є дані алгоритми.

3.1 Криптографічна стійкість SHA256

SHA256 як та інші хеш-функції схильний до колізій, тобто значення хеша однакове від двох різних аргументів. Чим менша частота колізій, тим якіснішою вважається хеш-функція. Очевидно, що для хеш-функцій, кількість можливих аргументів яких перевищує кількість можливих значень, колізії обов'язково існують (за тим самим принципом Діріхле [19]).

Для криптографічних хеш-функцій наявність колізій може, наприклад, дозволити підробляти підпис. Тому складність обчислення колізії вважається мірою криптографічної стійкості хеш-функцій.

Варто зазначити, що існує два роди колізій:

– стійкість до колізій першого роду (або стійкість до знаходження прообразу) означає високу обчислювальну складність знаходження повідомлення m' таке, що для заздалегідь відомого повідомлення m та його згортки $H(m)$ буде виконуватись $H(m') = H(m)$. Грубо кажучи, така стійкість передбачає неможливість для одного повідомлення підібрати друге з таким самим хешем;

– стійкість до колізій другого роду (або стійкість до знаходження колізій) означає високу обчислювальну складність знаходження двох пар повідомлень m і m' таких, що $H(m') = H(m)$. Грубо кажучи, неможливо підібрати пару повідомлень із однаковими хешами.

Варто зазначити, що стійкість до колізій другого роду, стійка і до колізій першого роду [20].

Одна з найпростіших атак на знаходження колізій – атака «днів народження» [21]. Назва походить на честь знаменитого парадоксу дня

народжень [22]. Суть його полягає в тому, що у кімнаті, де є 23 особи, ймовірність того, що дві особи мають один день народження більше 0.5. Вивід ймовірності тут не наводиться, проте результуюча формула така:

$$P(n) = 1 - \frac{365!}{(365-n)! * 365^n}.$$

Для наглядності графік від ймовірності від *n* наведено на рис. 3.1.

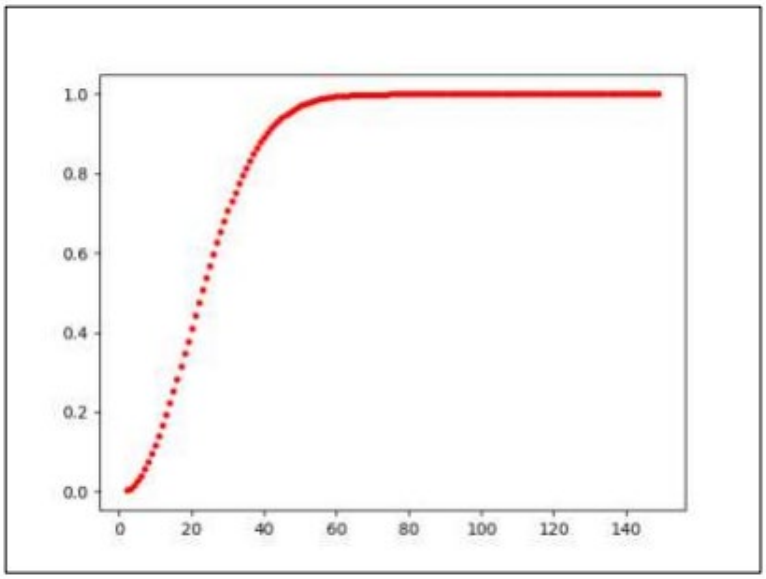

Рисунок 3.1 – Графік залежності функції розподілу для парадоксу дня народжень від кількості осіб *n*

Виконавши аналогічний вивід для *N* можливих випадків (замість 365 можливих днів), вийде $P(n) = 1 * \left(1 - \frac{1}{N}\right) * \left(1 - \frac{2}{N}\right) * \ldots * \left(1 - \frac{n-1}{N}\right).$

Використавши перші два члени у розкладанні ряду Тейлора функції $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots$ отримаємо $P(n) = 1 * e^{-\frac{1}{N}} * e^{-\frac{2}{N}} * \ldots * e^{-\frac{n-1}{N}} = e^{-\frac{(1+2+\cdots+(n-1))}{N}} = e^{\frac{-n(n-1)}{2N}} \approx e^{\frac{-n^2}{2N}}.$

Припустивши, «високою» ймовірність $P\ (n) \geq 1/2$, отримаємо рівняння

$$\frac{1}{2} \geq e^{\frac{-n^2}{2N}}.$$

Взявши логарифм з обох боків

$$-\ln 2 \geq -\frac{n^2}{2N}.$$

Звідси виходить

$$n \geq \sqrt{2 \ln 2 * N}.$$

Таким чином, використовуючи $N = 2^k, n \geq \sqrt{2 \ln 2} * 2^{\frac{k}{2}}.$, тобто для функції хешування, хеш якої - k -бітне число, необхідно перебрати приблизно $2^{k/2}$ повідомлень.

Наприклад, для SHA256 необхідно перебрати 2^{128} повідомлень. Це все ще дуже багато. Наприклад, за деякими оцінками [23], кількість розраховуваних хешів SHA256 біткоїном, в 2022 році перевищувала 200 квінтильйонів хешів (200 екзахешів) за секунду. Щоб перебрати 2^{128} хешів з такою швидкістю, потрібно $\frac{2^{128}}{200*10^{18}} \approx 1.7 * 10^{18}$ секунд або 54 мільярди років. Таким чином, криптографічна стійкість SHA256 є дуже високою.

3.2 Криптографічна стійкість HMAC

Так як HMAC двічі використовує функцію хешування, наявність колізій для даного алгоритму ще менше. Алгоритм також використовує секретний ключ, який відіграє певну роль «солі» для функції хешування, що ускладнює заміну даних.

Подвійне використання функції хешування також захищає від атаки подовження повідомлення, що виробляється більш слабкі хеш-функції (наприклад, SHA-1).

В даний час немає відомих знайдених атак проти функції HMAC, оскільки

застосування зовнішньої функції хешування приховує її вміст її внутрішнього хешу. Використання магічних блоків $ipad$ та $opad$ було спроектовано, щоб вони мали велику відстань Хеммінга між один одним. Таким чином, внутрішній і зовнішній ключі мають невелику кількість загальних бітів.

Спільною атакою на цей алгоритм залишається перебір ключа.

3.3 Криптографічна стійкість RSA

Оскільки криптографічна стійкість ЕЦП залежить переважно від використовуваного протоколу її формування, у цьому підрозділ буде проводитись аналіз для RSA.

Однією з атак є атака Хастеда [24], яка проводиться на алгоритм RSA, який використовує невелику експоненту. Наприклад, якщо кілька користувачів передають те саме повідомлення C з парами ключів $(N_i, e),\ C < N_i,$ то зловмиснику достатньо отримати $k \geq e$ повідомлень, щоб знайти M, де M - оригінальне повідомлення, $C = M^e mod N_i$.

Захистом від цієї атаки є вибір більшої експоненти. Так як атака відома, нині ні вона, ні інші атаки на малу експоненту не є актуальними.

Іншим видом атак є факторизація модуля N на прості множники p і q, $N=p*q$. Найбільш ефективними алгоритмами на даний момент є метод квадратичного решета (N до 10^{100} біт) [25] та загальний метод решета числового поля (N від 10^{100} біт) [26].

У ході виконання роботи було реалізовано алгоритм квадратичного решета. Вікно застосунку для введення даних представлено на рис. 3.2.

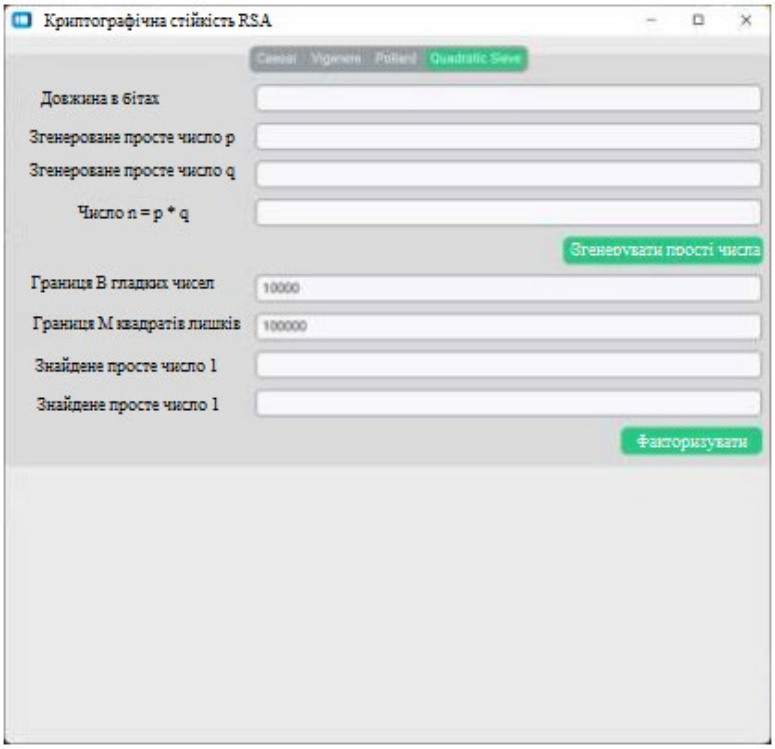

Рисунок 3.2 – Вікно для введення даних для квадратичного решета

Для демонстрації буде наведено час факторизації числа *N,* що складається з 50-бітних простих чисел. Згенероване число наведене на рис. 3.3.

Згенеровані параметри N, p та q

Рисунок 3.3 – Згенеровані параметри N, p та q

Параметри $B = 10000$, $M = 100000$ не дають рішення через лінійну незалежність отриманої матриці (рис. 3.4), тому їх потрібно збільшити

```
Пошук базисних змінних та вільних змінних... Помилка,
матриця лінійно незалежна.
0.9931373596191406 seconds...
```

Рисунок 3.4 – Вивід в консолі програми

Параметри B = 50000, M = 1000000 дали рішення за 56 с (рис. 3.5, 3.6):

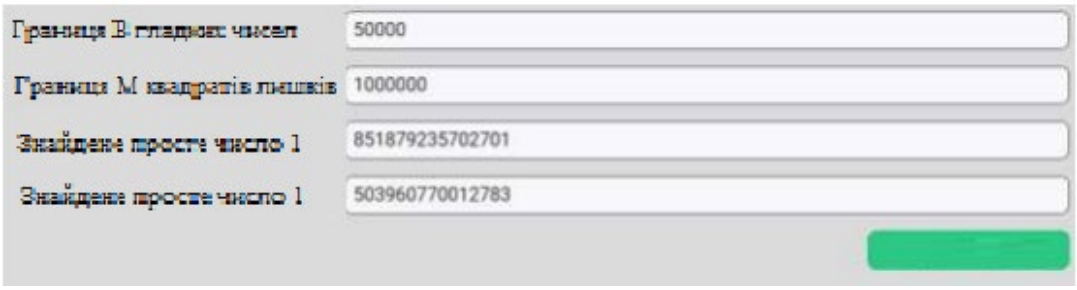

Рисунок 3.5 – Вивід результату у вікні застосунку

```
Пошук базисних змінних та вільних змінних... Спроба
одержання рішення: 1 55.93138122558594 seconds...
```

Рисунок 3.6 – Вивід часу виконання в консолі

Таким чином, було знайдено прості числа p і q, за якими можна знайти решту параметрів RSA, просто рухаючись за алгоритмом його ініціалізації (знаходження $\phi(n)$, обчислення d).

Одна з перспективних атак на RSA - використання квантового комп'ютера для факторизації простих чисел (використовуючи алгоритм Шора [27]). Основна ідея полягає в знаходженні r такого, що $a^r = 1 \bmod N$, де a - випадкове число, $2 \leq a < N$. Перетворивши це рівняння використовуючи різницю квадратів, вийде $\left(a^{\frac{r}{2}} - 1\right)\left(a^{\frac{r}{2}} + 1\right) = 0 \bmod N$, тобто N ділиться на обидва множники. Тим самим було знайдено дільники N. На звичайних комп'ютерах даний алгоритм зводиться до перебору r і стає дуже тривалим за часом. На квантових комп'ютерах використання таких алгоритмів можливе, зважаючи на використання кубитів. Однак на даний момент, кількість кубітів, що використовуються в існуючих прототипах комп'ютерів недостатньо для порушення стійкості поточної криптосистеми. Варто також відзначити, що ведуться розробки алгоритмів, захищених від квантових атак. Їх називають постквантовими [28].

3.4 Підсумки аналізу та приклади застосування алгоритмів

У цьому підрозділі представлені будуть представлені підсумки аналізу за реалізованими методами, а також деякі їх застосування. Спочатку будуть представлені результати за кодами алгебри.

3.4.1 Алгебраїчні коди

У цьому підрозділі будуть наведені підсумки аналізу та приклади застосування для реалізованих алгоритмів кодів алгебри.

Код із перевіркою на парність. Як уже говорилося раніше, цей код здатний лише виявляти одиночну помилку (кодова відстань $d = 2$). Його перевагою є дуже висока швидкість. Варто також відзначити, що апаратна реалізація також дуже проста - необхідно реалізувати суматор необхідної кількості біт (наприклад, можна використовувати 7- бітний суматор, щоб отримати байт інформації). Зважаючи на дані властивості, код з перевіркою на парність застосовується в місцях, де мала ймовірність помилки і важлива швидкість передачі. Тому він використовується в L -кешах процесорів, PCI шинах, де канал зв'язку дуже короткий (L- кеш ставиться або «впритул» до процесора, або - на сучасних процесорах - всередині кристала процесора (integrated circuit die); PCI(-E) шини розташовані якомога ближче до процесора), і ймовірність перешкоди в каналі дуже мала (забезпечується під час виробництва материнських плат).

Код Хеммінгу. Цей код дозволяє виявити дві або виправити одну помилку. Цей код теж виконується швидко (але повільніше за код з перевіркою на парність). Так як код дозволяє виправляти помилку, він вважається більш надійним, ніж код з перевіркою на парність, і також дозволяє працювати в комбінації з цим кодом. Одним з популярних застосувань коду Хеммінга - ECC пам'ять. Така пам'ять забезпечує більш надійний захист шляхом додавання окремого чіпа, що зберігає ECC коди на деякий блок пам'яті. Застосовується така пам'ять у місцях із сильними перешкодами (наприклад, у космічних апаратах, де потік нейтронів більший), або у системах із більш вимогливим захистом (наприклад, у файлових серверах).

CRC. Цей код, як згадувалося раніше, гарантовано дозволяє виявити всі одинарні, подвійні, і пакетні помилки (помилки з провалом декількох бітів поспіль) довжиною менше ступеня породжуючого багаточлена. Однак працює він значно повільніше. Основне його використання – передача інформації через мережу. Наприклад, цей код використовується в пакетах фізичного рівня сімейства технологій Ethernet [29] (поле FCS – Frame Check Sequence), який використовується у LAN, MAN, WAN мережах.

Код Ріда – Соломона. Цей код дозволяє виправити велику кількість помилок залежно від встановленого значення. Цей код в основному використовується у місцях, де неможливо повторно надіслати інформацію. Раніше найпопулярнішим місцем використання були компакт-диски. При виникненні на них подряпин, зчитування з компакт-дисків було б неможливим без цього коду. В даний час найбільш популярним місцем використання кодів Ріда – Соломона – QR -коди. До основної інформації додається надлишкова інформація коду, що кодується в двійковий код. Варто зазначити, що в QR - кодах використовуються більш удосконалені коди Ріда – Соломона, котрі мають змогу детектувати «стирання» та виправляти їх [30].

Підсумки аналізу представлені у табл. 3.1. У ній наведено результати аналізу наступних характеристик:

– кодова відстань *d;*

– кількість помилок, що гарантовано виявляються / виправляються;

– середня кількість помилок виявлених помилок у відсотках. Вказано результати тестів;

– швидкість роботи щодо інших алгоритмів;

– складність реалізації. Визначається на думку автора, як узагальнення таких критеріїв як складність теорії, складність алгоритму, обсяг коду.

Таблиця 3.1 – Підсумки аналізу кодів алгебри

Код	Кодова відстань d	Гарантов. виявлен / виправл. помилок	Середня кількість виявлен. помилок %	Швидкість роботи	Складність реалізаціїі	Захист від навмисного спотворення
Код з перевіркою на парність	2	1/0	50	дуже швидка	дуже легка	ні
Код Хеммінга	3	2/1	>94	швидка	легка	
Код CRC	*	2/2**	>99	середня	легка	
Код Ріда Соломона	***	$2t/t$, $t = \lfloor \frac{d}{2} \rfloor$	>99	повільна	дуже складна	

У табл. 3.1 "*" означає, що кодова відстань залежить від специфікації. алгоритму (вибраного породжуючого багаточлена), «**» - мінімальна гарантована кількість помилок, що виявляються/виправляються, «***» - кодова відстань залежить від зазначеного параметра i в параметрах РС коду.

3.4.2 Криптографічні алгоритми

SHA256. Даний алгоритм дозволяє виявляти практично всі спотворення (за винятком колізії). Однак, виправити він їх не може (оскільки виправлення не є метою використання хеш-функцій). Варто відзначити, що використання хеш-функцій у чистому вигляді зазвичай не мається на увазі - вона використовується в ролі криптографічного примітиву, тобто як складова більш складних алгоритмів. Приклади застосування хеш-функцій були представлені у цій роботі. Хеш-функція використовується практично у всіх алгоритмах аутентифікації (перевірки справжності), оскільки одним з важливих критеріїв безпеки при аутентифікацією є можливість отримати реквізитні дані (наприклад, пароль) при

компрометації ключа.

HMAC. Також дозволяє виявляти практично всі спотворення. Слід зазначити, що з даного алгоритму під цілісністю розуміється відсутність несанкціонованої зміни. Таким чином, цілісність та справжність у криптографічних системах взаємопов'язані (тому їх поєднують в одну підсистему «забезпечення цілісності»). HMAC використовують у JWT (JSON Web Token, вимовляється як «джот») токенах [31]. Даний токен складається з трьох закодованих в base64 рядків (header, в якому вказані параметри алгоритму; payload , в якому вказано необхідне навантаження в JSON форматі; (header) + "." + base64(payload) + "." + [секретний ключ]), записаних через точку.

ЕЦП із алгоритмом RSA. Забезпечує незмінність і автентичність інформації, що передається, оскільки використовує пару відкритого і приватного ключа (причому приватний ключ знаходиться тільки у його власника) разом з функцією хешування. Основне застосування цифрового підпису – підпис цифрових документів. Кваліфікований ЕЦП створюється в акредитованих центрах, що засвідчують, за допомогою сертифікованого обладнанням і алгоритмами. Однак замість RSA в них використовується більш надійний алгоритм Діффі - Хеллмана на еліптичних кривих (DH ECDSA)).

Підсумки аналізу представлені у табл. 3.2. У ній наведено результати аналізу наступних характеристик:

– криптографічна стійкість алгоритму Базується на результатах аналізу, проведеного раніше;

– час виконання щодо інших алгоритмів;

– складність реалізації. Процес оцінювання описаний раніше;

– захист від навмисного спотворення.

Таблиця 3.2 – Підсумки аналізу криптографічних алгоритмів

Алгоритм	Криптографічна стійкість	Швидкість роботи	Складність реалізації	Захист від навмисного спотворення
SHA256	висока	середня	середня	ні
HMAC-SHA256		середня	середня	так
ЕЦП з RSA		повільна	складна	

3.5 Висновки до третього розділу

Було виконано криптографічний аналіз алгоритмів SHA256, HMAC, RSA та алгебраїчних кодів, оскільки звичайні тести не дають змоги відобразити їх ефективність для забезпечення цілісності інформації.

Алгоритми були проаналізовані за такими параметрами: криптографічна стійкість; час виконання у порівнянні з іншими алгоритмами; обчислювальна складність реалізації; захист від навмисного спотворення.

Підсумки проведеного аналізу показують високий ступінь забезпечення цілісності інформації із використанням досліджених алгоритмів.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Метою кваліфікаційної роботи магістра є аналіз алгоритмів, що виявляють спотворення та цілеспрямовану зміну інформації. Під аналізом розумітиметься виведення (теоретичне чи практичне) оцінок для аналізованих алгоритмів. Оскільки, проведення робіт з розробки та використання ПЗ передбачає використання комп’ютерної техніки, зокрема ПК та периферійних пристроїв, то обов’язковим є дотримання вимог з охорони праці і техніки безпеки.

Для ефективної і безпечної роботи колективу працівників із аналізу аналіз алгоритмів, що виявляють спотворення та цілеспрямовану зміну інформації, необхідно організувати безпечні умови праці. При цьому керівник організації несе безпосередню відповідальність за порушення нормативно-правових актів з охорони праці [32]. Окрім цього, на робочих місцях працівників необхідно забезпечити дотримання вимог, затверджених Наказом Мінсоцполітики від 14.02.2018 за № 207 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями». Згідно Вимог приміщення, де розміщені робочі місця операторів, крім приміщень, у яких розміщені робочі місця операторів великих ЕОМ загального призначення (сервер), мають бути оснащені системою автоматичної пожежної сигналізації відповідно до цих вимог;

– переліку однотипних за призначенням об'єктів, які підлягають обладнанню автоматичними установками пожежогасіння та пожежної сигналізації, затвердженого наказом Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 22.08.2005 N 161, зареєстрованого в Міністерстві юстиції України 05.09.2005 за N 990/11270 (НАПБ Б.06.004-2005);

– Державних будівельних норм "Інженерне обладнання будинків і споруд. Пожежна автоматика будинків і споруд", затверджених наказом Держбуду України від 28.10.98 N 247 (далі - ДБН В.2.5-56:2014, з димовими пожежними

сповіщувачами та переносними вуглекислотними вогнегасниками.

В інших приміщеннях допускається встановлювати теплові пожежні сповіщувачі. Приміщення, де розміщені робочі місця операторів, мають бути оснащені вогнегасниками, кількість яких визначається згідно з вимогами ДСТУ 4297:2004 «Пожежна техніка. Технічне обслуговування вогнегасників». Загальні технічні вимоги і з урахуванням граничнодопустимих концентрацій вогнегасної рідини відповідно до вимог НАПБ А.01.001-2014. Приміщення, в яких розміщуються робочі місця операторів сервера загального призначення, обладнуються системою автоматичної пожежної сигналізації та засобами пожежогасіння відповідно до вимог ДБН В.2.5-56:2014, ДБН В.2.5-56:2010, НАПБ А.01.001-2014 і вимог нормативно-технічної та експлуатаційної документації виробника. Проходи до засобів пожежогасіння мають бути вільними.

Лінія електромережі для живлення комп'ютера та периферійних пристроїв повинні бути виконаними як окрема групова трипровідна мережа шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів. Не допускається використовувати нульовий робочий провідник як нульовий захисний провідник. Нульовий захисний провідник прокладається від стійки групового розподільного щита, розподільного пункту до розеток електроживлення. Не допускається підключати на щиті до одного контактного затискача нульовий робочий та нульовий захисний провідники.

Площа перерізу нульового робочого та нульового захисного провідника в груповій трипровідній мережі має бути не менше площі перерізу фазового провідника. Усі провідники мають відповідати номінальним параметрам мережі та навантаження, умовам навколишнього середовища, умовам розподілу провідників, температурному режиму та типам апаратури захисту, вимогам НПАОП 40.1-1.01-97.

У приміщенні, де одночасно експлуатуються понад п'ять комп'ютерів, на помітному, доступному місці встановлюється аварійний резервний вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення. Комп'ютери повинні підключатися до електромережі тільки за

допомогою справних штепсельних з'єднань і електророзеток заводського виготовлення.

У штепсельних з'єднаннях та електророзетках, крім контактів фазового та нульового робочого провідників, мають бути спеціальні контакти для підключення нульового захисного провідника. Їхня конструкція має бути такою, щоб приєднання нульового захисного провідника відбувалося раніше, ніж приєднання фазового та нульового робочого провідників. Порядок роз'єднання при відключенні має бути зворотним. Не допускається підключати комп'ютери до звичайної двопровідної електромережі, в тому числі – з використанням перехідних пристроїв. Електромережі штепсельних з'єднань та електророзеток для живлення комп'ютерної техніки повинні бути виконаними за магістральною схемою, по 3-6 з'єднань або електророзеток в одному колі. Штепсельні з'єднання та електророзетки для напруги 12 В та 42 В за своєю конструкцією мають відрізнятися від штепсельних з'єднань для напруги 127 В та 220 В. Штепсельні з'єднання та електророзетки, розраховані на напругу 12 В та 42 В, мають візуально (за кольором) відрізнятися від кольору штепсельних з'єднань, розрахованих на напругу 127 В та 220 В.

При підвищенні ефективності контролю доступу в приміщення, де для забезпечення безпеки мешканців, співробітників і збереження майна використовуються ДС, важливим, з точки зору охорони праці, є забезпечення достатньої величини природного та штучного освітлення, які визначені у НПАОП 0.00-7.15-18. Організація робочого місця фахівця із дослідження методів та програмно-апаратних засобів оптимізаційних процесів на основі ГА повинна забезпечувати відповідність усіх елементів робочого місця та їх розташування ергономічним вимогам ДСТУ 8604:2015 «Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги». Відстань від екрана до ока фахівців, які працюють за комп'ютером визначається згідно з вимогами ДСанПіН 3.3.2.007-98.

Розміщення принтера або іншого пристрою введення-виведення інформації на робочому місці має забезпечувати добру видимість екрана комп'ютера, зручність ручного керування пристроєм введення-виведення

інформації в зоні досяжності моторного поля згідно з вимогами ДСанПіН 3.3.2.007-98.

Таким чином, у результаті аналізу вимог щодо охорони праці користувачів комп'ютерів, визначено особливості організації робочих місць, вимог з електробезпеки, природного та штучного освітлення для ефективної і безпечної роботи фахівців з дослідження та аналізу алгоритмів, що виявляють спотворення та цілеспрямовану зміну інформації.

4.2. Комп'ютерне забезпечення процесу оцінки радіаційної та хімічної обстановки

Екологічне співтовариство розробило сімейство інструментів комплексної екологічної оцінки. Програмне забезпечення і послуги (ESS), комерційна група IIASA, включаючи AirWare (для повітряних проблеми якості), WaterWare (для якості води), CityWare (якість повітря і води в контексті великих міст) і EIAxpert (для надання допомоги із загальним впливом на навколишнє середовище). Функціональність в цілому схожа на RAISON, хоча з великим акцентом на моделювання і меншим акцентом на керування даними. Знову ж таки, інструменти ESS розроблені як модульні набори інструментів (доступні спеціальні системи для вирішення конкретних завдань). Компоненти включають стандартні імітаційні моделі, включаючи моделі ISC і PBM Агентства з охорони навколишнього середовища США, управління даними, в тому числі ГІС, аналіз даних (наприклад, аналіз часових рядів даних спостережень), візуалізація, а також оптимізація [33].

Іноді немає готових моделей, придатних для конкретного застосування, але тягар розробки нової програми на Фортрані або C / C ++ є надмірним. Розробка моделі оточення може відносно легко реалізувати власні моделі комп'ютерів і не турбуватися про включення процедур для вирішення рівнянь, візуалізації і т. д. Як правило, за допомогою цих інструментів користувач просто повинен вказати свою модель, використовуючи або математичні рівняння, або спеціальні графічні символи або значки, які безпосередньо представляють поведінку системи.

На даний момент є розроблені моделі комп’ютерного забезпечення процесу для оцінки радіаційної та хімічної обстановки.

GEMS – це система на основі моделей, яка підтримує оцінки схильності і ризику, надаючи доступ до одиночних і мультимедійних моделям експозиції, фізико-хімічні властивості методи оцінки, статистичний аналіз, графічні та картографічні програми з відповідними даними на навколишнє середовище, джерела, рецептори і популяції. У розробці з 1981 року, GEMS надає аналітикам 84 84 інтерактивний, легко досліджуваний інтерфейс для різних моделей, програм і даних, які необхідні для оцінки хімічного впливу і ризику [33].

HSPF – це комплексний пакет для моделювання кількості і якості стоків з багатоцільових водозборів і процесів радіації, що відбуваються в потоках або повністю змішаних озерах. Це дозволяє інтегроване моделювання землі і грунту, процесів забруднення при гідравлічній і осадово-хімічній взаємодії. Результатом моделювання є тимчасові дані витрати стоку, концентрація поживних речовин і пестицидів, а також дані кількості і якості води в будь-якій точці водозбору. Алгоритми якості води включають динаміку BOD / DO, вуглець, азот і фосфор. Процеси трансформації, які включені в модель це: гідроліз, фотоліз, окислення, випаровування, сорбція і биодеградація. Вторинні або «дочірні» хімічні речовини також моделюються.

Вимоги до даних для моделі можуть бути досить широкими в залежності від конкретного застосування.

Модель MMSOILS – це методологія оцінки впливу на людину і ризику для здоров'я, пов'язаних з викидами забруднень з небезпечних відходів. Мультимедійна модель, що стосується перенесення хімічної речовини в ґрунтові води, поверхневі води, атмосферу і накопичення в їжі. Шляхи впливу на людину, які розглянуті в методології включають: потрапляння в грунт, вдихання летких речовин в повітря і тверді частинки, шкірний контакт, прийом питної води і т.д. Ризик, пов'язаний із загальною дозою опромінення, розраховується на основі хімічної токсичності [33].

4.3 Висновки до четвертого розділу

В цьому розділі проаналізовано важливі питання охорони праці та безпеки в надзвичайних ситуаціях, висвітлено питання комп’ютерного забезпечення процесу оцінки радіаційної та хімічної обстановки.

ВИСНОВКИ

При виконані дослідження було проаналізовано та втілено такі алгоритми: код з перевіркою на парність, код Хеммінга, CRC, код Ріда - Соломона, алгоритм хешування SHA256, алгоритм імітовставки HMAC з використанням в якості функції хешування функцію SHA256 та алгоритм хешування підписи з використанням функції SHA256 як формування унікальної (до колізій) згортки повідомлення та шифрування алгоритмом RSA.

Для аналізу необхідно отримати необхідні теоретичні знання, пов'язані з роботою даних алгоритмів. Для кодів алгебри були отримані теоретичні знання про кінцеві поля *GF,* операцій додавання, множення, ділення кінцевих полях $GF(2^q)$. Для криптографічних алгоритмів було отримано знання причин криптографічної стійкості даних алгоритмів.

Для реалізації деяких алгоритмів необхідно було дослідити та реалізувати роботу підлеглих алгоритмів (далі – підалгоритмів). Були досліджені наступні підалгоритми: метод БМ та метод ПГЦ для декодування кодів Ріда – Соломона, алгоритм тесту на простоту Міллера – Рабіна для генерації простих чисел, необхідних для створення пари ключів в алгоритмі RSA.

В результаті аналізу було отримано теоретичні та практичні результати. Теоретичні результати отримані в ході дослідження теорії алгоритмів та їх подальшого теоретичного аналізу. Вони містять такі параметри як: здібності, які виявляють та коригують, захищеність від навмисного спотворення і стійкість для криптографічних алгоритмів. Практичні результати було отримано в результаті аналізу криптографічних алгоритмів. Отримані результати включають:

– результати тестування алгоритмів, у ході яких було порівняно теоретичні характеристики алгоритмів та практичні;

– результати залежностей виконання підалгоритмів для деяких алгоритмів від різних параметрів алгоритму;

– результати залежностей часу виконання різних функціональних частин алгоритмів за різних вибраних параметрів алгоритмів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Venkatesan Guruswami, Atri Rudra, Madhu Sudan. Essential Coding Theory. University at Buffalo, 2022. 473 p.
2. Жива математика / За ред. В.О. Тадеєва. Тернопіль: Навчальна книга – Богдан, 2011. 240 с.
3. Аналіз алгоритмів множення в полях Галуа для криптографічного захисту інформації | Наукові журнали та конференції. Academic Journals and Conferences |. URL: https://science.lpnu.ua/uk/sisn/vsi-vypusky/vypusk-13-2023/analiz-algorytmiv-mnozhennya-v-polyah-galua-dlya-kryptografichnogo (дата звернення: 14.12.2024).
4. Checksums & Integrity Checks. Electronics Research Group University of Aberdeen. URL: https://www.erg.abdn.ac.uk/users/gorry/eg3576/crc.html (date of access: 14.12.2024).
5. Karnaukhov, A., Tymoshchuk, V., Orlovska, A., & Tymoshchuk, D. (2024). USE OF AUTHENTICATED AES-GCM ENCRYPTION IN VPN. Матеріали конференцій МЦНД, (14.06. 2024; Суми, Україна), 191-193.
6. Decoding Generalized Reed-Solomon Codes and Its Application to RLCE Encryption Scheme. IACR Cryptology ePrint Archive. URL: https://eprint.iacr.org/2017/733 (date of access: 14.12.2024).
7. Zagorodna, N., Skorenkyy, Y., Kunanets, N., Baran, I., & Stadnyk, M. (2022). Augmented Reality Enhanced Learning Tools Development for Cybersecurity Major. In ITTAP (pp. 25-32).
8. Алгоритм хешування SHA-256: що це таке і як він працює. URL: https://portalcripto.com.br/uk/словник/хеш-алгоритм-sha-256%2C-що-це-таке-і-як-він-працює/ (дата звернення 11.11.2024).
9. 2 в 1: шифрування з імітозахистом. URL: https://habr.com/articles/497828/ (дата звернення 12.11.2024).
10. HMAC | Working of Hash Based Message Authentication Code. EDUCBA. URL: https://www.educba.com/hmac (date of access: 14.12.2024).

11. Що таке електронний цифровий підпис (ЕЦП)? URL: https:// https://www.kmu.gov.ua/usi-pitannya-po-e-poslugam/sho-tak-elektronnij-cifrovij-pidpis-ecp (дата звернення 12.11.2024).

12. Tymoshchuk, V., Karnaukhov, A., & Tymoshchuk, D. (2024). USING VPN TECHNOLOGY TO CREATE SECURE CORPORATE NETWORKS. Collection of scientific papers «ΛΌГОΣ», (June 21, 2024; Seoul, South Korea), 166-170.

13. Prime Numbers Library. Prime Numbers. URL: https://prime-numbers.info/#numberTypes (date of access: 14.12.2024).

14. Kharchenko, O., Raichev, I., Bodnarchuk, I., & Zagorodna, N. (2018, February). Optimization of software architecture selection for the system under design and reengineering. In 2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET) (pp. 1245-1248). IEEE.

15. How to interpret Berlekamp-Massey algorithm?. Cryptography Stack Exchange. URL: https://crypto.stackexchange.com/questions/88040/how-to-interpret-berlekamp-massey-algorithm (date of access: 14.12.2024).

16. Fryz, M., Mlynko, B., Mul, O., & Zagorodna, N. (2010). Conditional Linear Periodical Random Process as a Mathematical Model of Photoplethysmographic Signal. Rigas Tehniskas Universitates Zinatniskie Raksti, 45, 82.

17. Цимбрак І.С. Реалізація криптоалгоритму хешування SHA256// Інформаційні моделі, системи та технології: Праці XII наук.-техн. конф. (Тернопіль, ТНТУ ім. І. Пулюя, 18-19 грудня 2024 р.) С. 126.

18. Филипов Микита. Що таке розподіл. robot_dreams - онлайн-курси для фахівців у сфері big data, machine learning, data science | Робот Дрімс. URL: https://robotdreams.cc/uk/blog/267-chto-takoe-raspredeleniya (дата звернення: 14.12.2024).

19. Принцип Діріхле та його застосовування. URL: https://vseosvita.ua/library/princip-dirihle-ta-jogo-zastosovuvanna-181809.html (дата звернення 20.11.2024).

20. Tymoshchuk, D., Yasniy, O., Mytnyk, M., Zagorodna, N., Tymoshchuk, V., (2024). Detection and classification of DDoS flooding attacks by machine learning methods. CEUR Workshop Proceedings, 3842, pp. 184 - 195.

21. Math Forum: Ask Dr. Math FAQ: The Birthday Problem URL: http://mathforum.org/dr.math/faq/faq.birthdayprob.html (дата звернення 21.11.2024).

22. Lypa, B., Horyn, I., Zagorodna, N., Tymoshchuk, D., Lechachenko T., (2024). Comparison of feature extraction tools for network traffic data. CEUR Workshop Proceedings, 3896, pp. 1-11.

23. 1.1 quintillion operations per second: US has world's fastest supercomputer. Ars Technica. URL: https://arstechnica.com/information-technology/2022/05/1-1-quintillion-operations-per-second-us-has-worlds-fastest-supercomputer/ (date of access: 14.12.2024).

24. Алгоритм шифрування RSA, види атак на нього. Реалізація мовою Python. URL: https://dou.ua/forums/topic/43026/ (дата звернення 23.11.2024).

25. C. Pomerance, "The quadratic sieve factoring algorithm", in Proc. of EUROCRYPT 84. A Workshop on the Theory and Application of Cryptographic Techniques, Paris, 1984. pp. 169-182.

26. ТИМОЩУК, Д., & ЯЦКІВ, В. (2024). USING HYPERVISORS TO CREATE A CYBER POLYGON. MEASURING AND COMPUTING DEVICES IN TECHNOLOGICAL PROCESSES, (3), 52-56.

27. Shor's algorithm. Home. URL: https://www.qutube.nl/quantum-algorithms/shors-algorithm (date of access: 14.12.2024).

28. Post-quantum cryptography URL: https://csrc.nist.gov/projects/post-quantum-cryptography (date of access: 24.11.2024).

29. ТИМОЩУК, Д., ЯЦКІВ, В., ТИМОЩУК, В., & ЯЦКІВ, Н. (2024). INTERACTIVE CYBERSECURITY TRAINING SYSTEM BASED ON SIMULATION ENVIRONMENTS. MEASURING AND COMPUTING DEVICES IN TECHNOLOGICAL PROCESSES, (4), 215-220.

30. ISI/IEC 18004 International Standard. Official edition. ISO/IEC/ 2015. 117p.

31. Orobchuk, O. (2019). Methodology of development and architecture of ontooriented system of electronic learning of Chinese image medicine on the basis of training management system. Вісник Тернопільського національного технічного університету, 92(4), 83-90.

32. Заікіна Д., Глива В. Основи охорони праці та безпека життєдіяльності. 2019. URL: https://doi.org/10.31435/rsglobal/001 (дата звернення: 14.12.2024).

33. Кучма М.М. Цивільна оборона (цивільний захист). Навчальний посібник. Львів : Магнолія 2006 . 2024. 296 с.

ДОДАТОК А

Тези конференції