МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп’ютерних інформаційних технологій
Кафедра комп’ютерної інженерії

Троць Ігор Іванович

Алгоритм реалізації китайської теореми про залишки для асиметричних криптосистем / The algorithm of Chinese theorem implementation about residuals for asymmetric cryptosystems

спеціальність:123 – Комп’ютерна інженерія
освітньо-професійна програма – Комп’ютерна інженерія

Випускна кваліфікаційна робота

Виконав студент групи КІм-21
І. І. Троць

____

Науковий керівник:
к.т.н., Батько Ю. М.

____

ТЕРНОПІЛЬ - 2019

РЕЗЮМЕ

Випускна кваліфікаційна робота «Алгоритм реалізації китайської теореми про залишки для асиметричних криптосистем» містить 88 сторінок основного тексту, 33 рисунки, 3 додатки.

Метою даної кваліфікаційної роботи є дослідження існуючих методів протидії аналізу криптосистем, за інформацією, що несуть сторонні канали, а також дослідження цієї інформації, а завданням дослідження є розробка та створення власного програмного додатку на основі власного алгоритму.

В роботі було розроблено програмний додаток на основі власного алгоритму для реалізації китайської теореми про залишки для асиметричних криптосистем.

Розроблений алгоритм є унікальним методом для рішення завдань, раніше такий алгоритм ніде не використовувався та в порівнянні з аналогами які існують на сьогоднішній день створений алгоритм є кращим майже за всіма критеріями.

Практична значення отриманих результатів роботи проявляється в тому, що був розроблений програмний додаток для вирішення китайської теореми про залишки, який працює за унікальним алгоритмом, який має декілька кроків для вирішення завдання.

КЛЮЧОВІ СЛОВА: ПРОГРАМНИЙ ДОДАТОК, АЛГОРИТМ, КРИПТОСИСТЕМА, ТЕОРЕМА.

RESUME

The master's thesis «The algorithm of Chinese theorem implementation about residuals for asymmetric cryptosystems» contains 88 pages, 33 drawings, 3 appendices.

The purpose of this thesis is to investigate existing methods of countering the analysis of cryptosystems, information carried by third-party channels, as well as the study of this information, and the task of the research is to develop and create their own software application based on their own algorithm.

In the master's thesis, a software application was developed based on its own algorithm to implement the Chinese residual theorem for asymmetric cryptosystems.

The developed algorithm is a unique method for solving problems, such an algorithm has never been used before and compared to the analogues that exist today, the algorithm is the best by almost all criteria.

The practical significance of the results obtained is that a software application was developed to solve the Chinese residue theorem, which works on a unique algorithm that has several steps to solve the problem.

KEYWORDS: SOFTWARE, ALGORITHM, CRYPTOSYSTEM, THEOREM.

ЗМІСТ

Вступ 10
1 Асиметричні криптосистеми та основи китайської теореми та алгоритму Евкліда 12
1.1 Аналіз найбільш поширених асиметричних криптосистем 12
1.2 Теоретичні основи китайської теореми про залишки 20
1.3 Система залишкових класів 24
1.4 Висновок до першого розділу 30
2 Аналіз криптосистем та методи протидії 31
2.1 Алгоритм Евкліда 31
2.2 Аналіз по стороннім каналам та його види 33
2.3 Опис розробки та створення власного алгоритму 44
2.4 Висновок до другого розділу 50
3 Створення власного алгоритму обчислення китайської теореми про залишки 51
3.1 Структура та опис створенної програми 51
3.2 Тестування роботи створенного програмного додатку 55
3.3 Порівняння з існуючими аналогами 60
3.4 Висновок до третього розділу 69
Висновки 70
Список використаних джерел 71
Додаток А Лістинг коду створенного програмного додатку 76
Додаток Б Довідка про використання 78
Додаток В Світлокопії виданих публікацій 79

ВСТУП

Актуальність теми. Комп’ютери та інші електронні пристрої на сьогодні тісно пов’язані з людським життям. Насправді вони супроводжують нас усюди і несуть велику кількість інформації про свого власника. Такий великий попит техніки та використання її як в робочих потребах так і у власних вплинуло на розвиток механізмів захисту приватної інформації користувачів, збереженню її конфіденційності.

Головні механізми у захисті інформації у наш час спрямовані на забезпечення двох основних моментів: цілісності та конфіденційності. Починаючи з середини минулого століття цими ідеями опікувалась наука – криптографія. Її розвиток вилився у створення багатьох алгоритмів шифрування, які з рокам удосконалювались розвитком технологій та математичного апарату і на сьогодні досягли можливості виконати вимоги до збереження даних.

Мета і завдання дослідження. Метою даної дипломної роботи є дослідження існуючих методів протидії аналізу криптосистем, за інформацією, що несуть сторонні канали, а також дослідження цієї інформації, а завданням дослідження є розробка та створення власного прогрманого додату на основі власного алгоритму.

Об’єкт дослідження. Об’єктом дослідження було обрано алгоритми вирішення китайської теореми про залишки в асиметричних криптосистемах.

Предмет дослідження. Предметом дослідження в даній роботі є методи протидії аналізу інформації, їх вплив на реалізацію алгоритму, його швидкодію та надійність, яку вони гарантують.

Методи досліджень. Проведення аналізу інформації зібраної при роботі алгоритму, а також методів протидії аналізу цієї інформації та перевірка їх ефективності при використанні в порівнянні з існуючими аналогами.

Наукова новизна одержаних результатів. Розроблений алгоритм є унікальним методом для рішення завдань, раніше такий алгоритм ніде не

використовувався та в порівнянні з аналогами які існують на сьогоднішній день створений алгоритм є кращим майже за всіма критеріями.

Практичне значення отриманих результатів. Практична значення отриманих результатів роботи проявляється в тому, що був розроблений програмний додаток для вирішення китайської теореми про залишки, який працює за унікальним алгоритмом, який має декілька кроків для вирішення завдання.

Публікації та апробація ВКР. Результати наукового дослідження опубліковано в матеріалах науково – практичних конференцій молодих вчених та студентів «Інтелектуальні комп’ютерні системи та мережі» (Тернопіль, 2019) [3,4].

У першому розділі проведено аналіз найбільш поширених асиметричних криптосистем, аналіз теоретичних основ китайської теореми про залишки, аналіз системи залишкових класів та зроблено відповідні висновки.

У другому розділі описано алгоритм Евкліда, було проведено аналіз по стороннім каналам та описані його види, також був опис розробки та створення власного алгоритму та зроблено відповідні висновки.

У третьому розділі здійснена програмна реалізація алгоритму, було наведено опис структури створенної програми, проведено тестування роботи створенного програмного додатку, було наведено порівняння з існуючими аналогами та зроблено відповідні висновки.

У додатках наведено код розробленного програмного додатку та блок-схему роботи створенного алгоритму.

1 АСИМЕТРИЧНІ КРИПТОСИСТЕМИ ТА ОСНОВИ КИТАЙСЬКОЇ ТЕОРЕМИ ТА АЛГОРИТМУ ЕВКЛІДА

1.1 Аналіз найбільш поширених асиметричних криптосистем

Відомо, що усім симетричним криптосистемам, у яких шифрування та розшифровування відбувається за допомогою одного і того самого ключа, притаманні такі основні недоліки:

– принциповою є надійність каналу передачі ключа другому учаснику секретних переговорів. Інакше кажучи, ключ повинен передаватися по секретному каналу;

– до служби генерації ключів пред'являються підвищені вимоги, обумовлені тим, що для j абонентів при схемі взаємодії "кожен з кожним" потрібно $j(j-1)/2$ ключів, тобто залежність кількості ключів від кількості абонентів є квадратичною.

Для вирішення перерахованих вище проблем симетричного шифрування призначені системи з асиметричним шифруванням або шифруванням з відкритим ключем рисунок 1.1, що використовують властивості функцій з секретом, розроблених Діффі і Хеллманом.

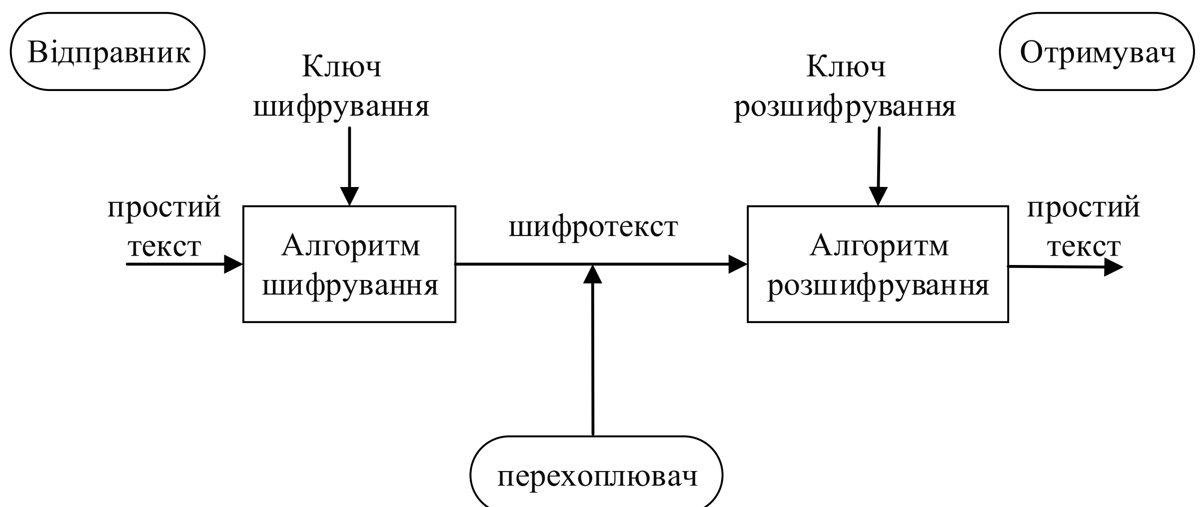

Рисунок 1.1 – Схема асиметричних криптосистем

Ці системи характеризуються наявністю у кожного абонента двох ключів: відкритого і закритого тобто секретного [1]. При цьому відкритий ключ відкрито передається всім учасникам секретних переговорів. Таким чином, вирішуються дві проблеми:

– немає потреби в секретній доставці;

– відсутня квадратична залежність кількості ключів від кількості користувачів - для *j* користувачів потрібно 2*j* ключів.

Першим шифром, розробленим на принципах асиметричного шифрування, є шифр RSA. Він названий так за першими літерами прізвищ його винахідників: Рона Райвеста, Аді Шаміра і Леонарда Елдемана - засновників компанії RSA Data Secutity. RSA - не тільки найпопулярніший з асиметричних шифрів, але, мабуть, взагалі найвідоміший.

Математичне обгрунтування RSA таке: пошук дільників дуже великого натурального числа, яке є добутком двох простих чисел – дуже трудомістка процедура. Відповідно, за допомогою відкритого ключа дуже складно обчислити відповідний йому таємний ключ. Схема криптоалгоритму RSA представлена на рисунку 1.2.

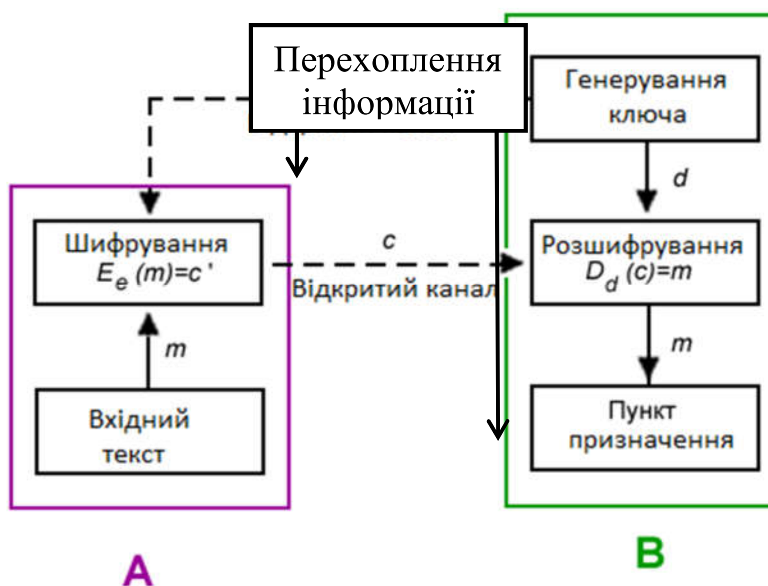

Рисунок 1.2 – Криптоалгоритм RSA

Шифр RSA всебічно вивчений і визнаний стійким при достатній довжині ключів. Наприклад, 512 біт для забезпечення стійкості не вистачає, а 1024 біти вважається прийнятним варіантом [2]. З ростом потужності процесорів при даній довжині ключа RSA втрачає стійкість до атаки повним перебором, однак це дозволяє застосувати більш довгі ключі, що в свою чергу підвищить стійкість шифру.

Шифр працює за алгоритмом, який включає в себе генерацію ключів, шифрування та дешифрування.

Для того, щоб згенерувати пари ключів виконуються такі дії:

– вибираються два великі прості числа p і q;

– обчислюється їх добуток $n=p \cdot q$;

– обчислюється функція Ейлера $\varphi(n)=(p-1)(q-1)$;

– вибирається ціле число e таке, що$1<e<\varphi(n)$ та e - взаємно просте з $\varphi(n)$, тобто НСД(e, $\varphi(n)$)=1;

– за допомогою розширеного алгоритму Евкліда знаходиться число d таке, що$ed \pmod{\varphi(n)}=1$ або $d=e^{-1} \pmod{\varphi(n)}$.

Число n називається модулем, а числа e і d – відкритою та секретною експонентами відповідно. Пара чисел (n,e) є відкритою частиною ключа, а пара (n, d) – секретною. Числа p і q після генерації ключів можуть бути знищені, але в жодному разі не повинні бути розкриті.

Для шифрування повідомлення припустимо, що перший абонент хоче відправити другому повідомлення A. Для початку він за допомогою узгодженого протоколу перетворення, відомого як доповняльні схеми або таблиці перетворення, перетворює A в ціле число a так, щоб $0\leq a\leq n$. Опісля він обчислює зашифрований тест A', використовуючи відкритий ключ другого абонента e, за допомогою рівняння:

$$A' = a^e \bmod n. \tag{1.1}$$

Це може бути зроблено досить швидко, навіть у випадку, коли текст перевищує 500-бітну розрядність.

Розшифрування відбувається наступним чином:

$$A = (A')^d \bmod n. \qquad (1.2)$$

Відповідно при виконанні даної операції відновлюєтсья вихідне повідомлення:

$$(A')^d \equiv (a^e)^d \equiv a^{ed} \pmod n, \qquad (1.3)$$

З умови

$$ed \equiv 1 \pmod{\varphi(n)}, \qquad (1.4)$$

випливає твердження, що $ed \equiv k_0\varphi(n) + 1$ для деякого цілого k_0, отже:

$$a^{ed} \equiv a^{k0\varphi(n)+1} \pmod n. \qquad (1.5)$$

Згідно з теоремою Ейлера:

$$a^{\varphi(n)} \equiv 1 \pmod n, \qquad (1.6)$$

тому

$$a^{k0\varphi(n)+1} \equiv a \pmod n,\ (A')^d \equiv a \pmod n. \qquad (1.7)$$

Для шифрування необхідно знати пару чисел *e, n*, для дешифрування - *d, n*. Перша пара – відкритий ключ, друга – закритий. Знаючи відкритий ключ, можна обчислити значення закритого ключа. Необхідною проміжною дією

цього перетворення є знаходження множників *p* і *q*, для чого потрібно розкласти *n* на множники – ця процедура займає дуже багато часу. Саме з величезною обчислювальною складністю пов'язана криптостійкість RSA.

Криптосистема Рабіна стала першою асиметричною криптосистемою, яка грунтується на складності знаходження квадратичного лишку. Вона, як і будь-яка асиметрична криптосистема, використовує відкритий і закритий ключі. Відкритий ключ використовується для шифрування повідомлень і може бути опублікований для загального огляду. Закритий ключ необхідний для розшифровки і повинен бути відомий тільки одержувачам зашифрованих повідомлень. Схема шифрування згідно криптосистеми Рабіна наведена на рисунку 1.3.

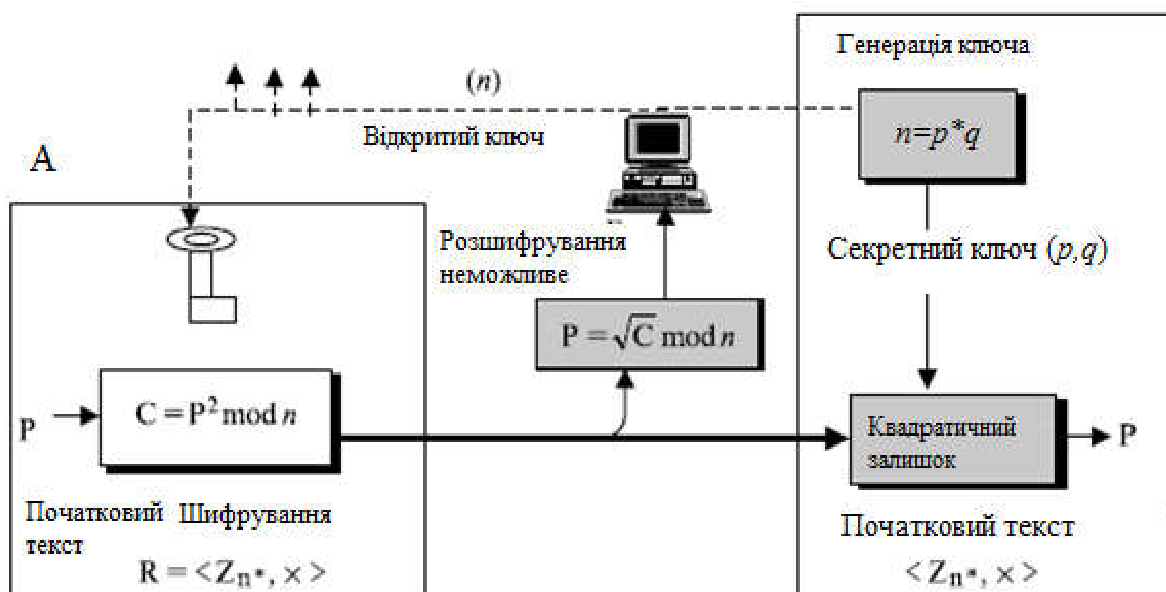

Рисунок 1.3 – Схема шифрування згідно криптосистеми Рабіна

Для генерації ключів вибираються два випадкових числа *p* і *q* з урахуванням таких вимог:

– числа повинні бути великими;

– числа повинні бути простими;

– повинна виконуватися умова: $p \equiv q \equiv 3 \bmod 4$.

Виконання цих вимог сильно прискорює процедуру знаходження коренів за модулем p і q. Далі обчислюється число $n=p\cdot q$, тоді число n – відкритий ключ; числа p і q – закритий.

Початкове повідомлення A (текст) шифрується за допомогою відкритого ключа - числа n за такою формулою:

$$A' = A^2 \bmod n. \tag{1.8}$$

Завдяки використанню операції піднесення до квадрату за модулем швидкість шифрування системи Рабіна більша, ніж швидкість шифрування за методом RSA, навіть якщо в останньому випадку вибрати невелике значення експоненти.

При дешифруванні криптограми A' для зручності вводяться додаткові допоміжні величини c_1 і c_2:

$$c_1 = A' \bmod p; \quad c_2 = A' \bmod q. \tag{1.9}$$

Для знаходження A необхідно знайти квадратичні лишки c_1, c_2 відповідно за модулями p і q:

$$x^2 \equiv c_1 (\mathrm{mod}\ p),\ y^2 \equiv c_2 (\mathrm{mod}\ q). \tag{1.10}$$

В результаті можна записати чотири системи порівнянь:

$$\begin{cases} A_1 \equiv x(\mathrm{mod}\ p); \\ A_1 \equiv y(\mathrm{mod}\ q); \end{cases} \begin{cases} A_2 \equiv x(\mathrm{mod}\ p); \\ A_2 \equiv -y(\mathrm{mod}\ q); \end{cases} \begin{cases} A_3 \equiv -x(\mathrm{mod}\ p); \\ A_3 \equiv y(\mathrm{mod}\ q); \end{cases} \begin{cases} A_4 \equiv -x(\mathrm{mod}\ p); \\ A_4 \equiv -y(\mathrm{mod}\ q). \end{cases} \tag{1.11}$$

Одне з рішень, яке грунтується на використанні китайської теореми про залишки, буде шуканим повідомленням A.

Розшифрування тексту, крім правильного, приводить ще до трьох хибних результатів. У цьому і полягає головна незручність криптосистеми Рабіна і одним з факторів, який перешкоджав тому, щоб вона знайшла широке практичне використання.

Коли вихідний текст буде являти собою текстове повідомлення, то визначення правильного тексту не буде важким. Однак якщо повідомлення є потоком випадкових бітів, наприклад, для генерації ключів або цифрового підпису, то таким чином визначення потрібного тексту стає реальною проблемою. Одним із способів уникнути цього недоліку є додавання до повідомлення перед шифруванням відомого заголовку [5].

Однак у класичній криптосистемі Рабіна блок відкритого тексту обмежується величиною відкритого ключа. Тому для дуже довгих повідомлень потрібно кожен блок шифрувати окремо.

Приблизно такою ж обчислювальною складністю, як і факторизація, володіє операція дискретного логарифмування у скінченному полі, на якій ґрунтується асиметрична криптосистема Ель-Гамаля (Elgamal). Вона включає в себе алгоритм шифрування та алгоритм цифрового підпису.

Схема була запропонована у 1985 році і являється одним із удосконалених варіантів алгоритму Діффі-Хеллмана, які використовуються для шифрування та забезпечення аутентифікації.

Алгоритм виконується в наступній послідовності:

Крок 1. Генерується випадкове просте число.

Крок 2. Вибирається ціле число та первісний корінь.

Крок 3. Вибирається випадкове ціле число яке задовільнить нерівність.

Крок 4. Обчислюється модуль цого числа.

Крок 5. Відкритий ключ становлять три числа та таємний ключ.

Повідомлення, яке повинно бути менше від числа, шифрується таким чином:

– вибирається сесійний ключ – тобто випадкове ціле число таке, що задовільнить заду нерівність;

– вираховуються числа.

Одним з недоліків криптосистеми Ель-Гамаля є те, що довжина шифротексту вдвічі більша від вихідного повідомлення A.

Знаючи закритий ключ , вихідне повідомлення можна обчислити з шифротекста (c_1, c_2) за такою формулою:

$$m=c_2(c_1^{-1})^a \bmod p. \qquad (1.12)$$

При цьому дуже легко перевірити, що $(c_1^{-1})^a \bmod p=g^{-ra} \bmod p$ і тому $c_2(c_1^{-1})^a \bmod p=(h^r A)g^{-ra} \bmod p=(g^{ra}A)g^{-ra} \bmod p=A \bmod p$.

Для практичних обчислень більше підходить наступна формула:

$$A= c_2(c_1^{-1})^a \bmod p= c_2(c_1)^{p-1-a} \bmod p. \qquad (1.13)$$

Схема шифрування на основі криптоалгоритму Ель-Гамаля представлена на рисунку 1.4.

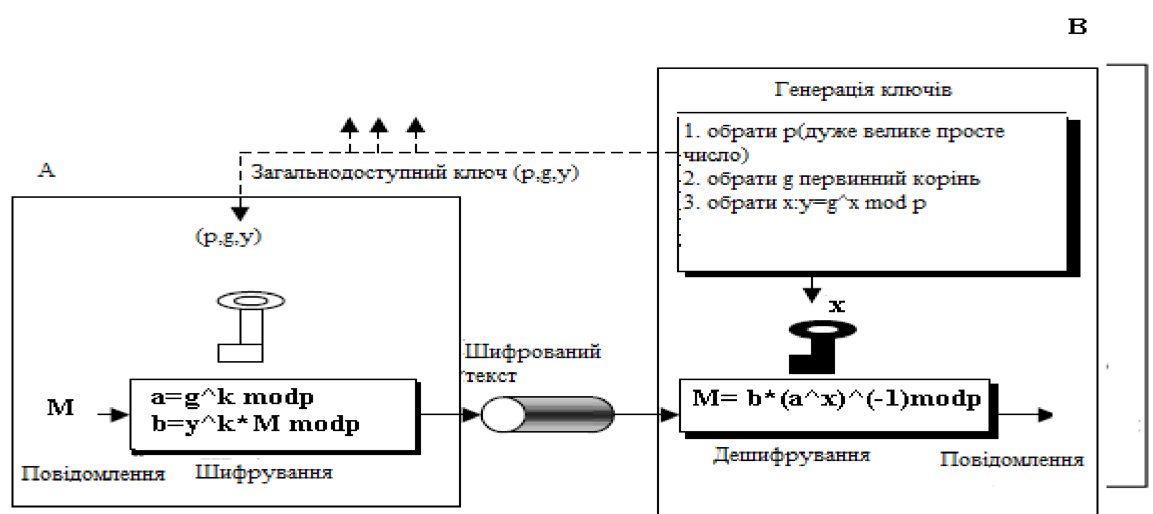

Рисунок 1.4 – Схема шифрування на основі криптоалгоритму Ель-Гамаля

Так як в схему Ель-Гамаля вводиться випадкова величина *r*, то шифр Ель-Гамаля можна назвати шифром багатозначної заміни. Через випадковість вибору числа *r* таку схему ще називають схемою імовірнісного шифрування. Імовірнісний характер шифрування є перевагою для криптоалгоритму Ель-Гамаля, так як у схем імовірнісного шифрування спостерігається більша стійкість в порівнянні зі схемами з певним процесом шифрування. Недоліком схеми шифрування Ель-Гамаля є подвоєння довжини зашифрованого тексту в порівнянні з початковим текстом. Для схеми імовірнісного шифрування саме повідомлення і ключ не визначають шифротекст однозначно. У схемі Ель-Гамаля необхідно використовувати різні значення випадкової величини для шифровки різних повідомлень і . Якщо використовувати однакові , то для відповідних шифротекстів (c_1, c_2) і (c_1', c_2') виконується співвідношення $c_2(c_2')^{-1} = A(A')^{-1}$. З цього виразу можна легко обчислити A', якщо відомо .

На даний час криптосистеми з відкритим ключем вважаються найбільш перспективними. До них належить і схема Ель-Гамаля, криптостійкість якої ґрунтується на обчислювальній складності проблеми дискретного логарифмування [6]. Крім того, існує велика кількість інших алгоритмів, які базуються на схемі Ель-Гамаля, зокрема: DSA, ECDSA, KCDSA, схема Шнорра тощо. Однак усі операції у проаналізованих криптоалгоритмах відбуваються в ПСЧ (двійковій або десятковій системі числення), у яких відсутня можливість розпаралелення процесу обчислень.

1.2 Теоретичні основи китайської теореми про залишки

Теорема про ділення з остачею: розділити число $a \in Z$ на число $b \in Z$, $b \neq 0$, з остачею, означає знайти пару цілих чисел *q* і *r*, таких, що виконуються наступні умови:

$$a = b\gamma + r, 0 \le r < |b|. \quad (1.14)$$

Легко доводиться, що для будь-яких цілих чисел a та $b \neq 0$ ділення з остачею можливо і числа γ і r визначаються однозначно.

Один з методів виконання арифметичних операцій над даними цілими числами грунтується на простих положеннях теорії чисел. Ідея цього методу полягає в тому, що цілі числа представляються в одній з непозиційних систем – СЗК. А саме: замість операцій над цілими числами оперують із залишками від ділення цих чисел на заздалегідь обрані взаємно прості числа – модулі $p_1, p_2, ..., p_j$. Найчастіше числа $p_1, p_2, ..., p_j$ вибирають із множини простих чисел.

Нехай $A \equiv \alpha_1 (\bmod p_1)$, $A \equiv \alpha_2 (mod\ p_2), \ldots,$ $A \equiv \alpha_j (\bmod p_j) \ldots$

Важливо відзначити, що при цьому немає ніякої втрати інформації за умови, що $A < p_1 p_2 ... p_j = P$, тому що завжди, знаючи $(\alpha_1, \alpha_2, ..., \alpha_j)$ можна відновити саме число A. Тому кортеж $(\alpha_1, \alpha_2, ..., \alpha_j)$можна розглядати як один зі способів подання цілого числа A в комп'ютері – модулярне подання або подання в СЗК.

Мультиплікативно оберненим елементом до числа a у модулярній арифметиці є таке число b, що виконується конгруенція:

$$ab \bmod z = 1. \quad (1.15)$$

Умовою існування мультиплікативно оберненого елемента є рівність 1 найбільшого спільного дільника (НСД) чисел a і b, тобто числа a і b повинні бути взаємно прості. Якщо ця умова не виконується, то мультиплікативно обернений елемент до a не існує [7].

Методи пошуку мультиплікативно оберненого елемента можна розділити на дві великі категорії: методи, що не ґрунтуються на методах пошуку НСД, і методи, які є похідними від методів пошуку НСД.

Найбільш поширеними методами, які відносяться до першої групи, є повний перебір всіх можливих варіантів або брутальна атака та метод на основі функції Ейлера.

Під брутальною атакою розуміється метод рішення математичних задач, при якому складність повного перебору, брутальної атаки, залежить від кількості всіх можливих варіантів вирішення задачі. Цей метод відноситься до класу методів пошуку рішення задач із вичерпуванням можливих варіантів розв'язку системи, є найпростішим і водночас найзатратнішим. Він характеризується високою обчислювальною складністю, оскільки повний перебір вимагає значних часових затрат [8].

Функція Ейлера $\varphi(z)$, де z – натуральне число, це цілочисельна функція, яка рівна кількості натуральних чисел, не більших за z і взаємно простих з ним. Функцію Ейлера можна подати у вигляді так званого добутку Ейлера:

$$\varphi(z) = z \prod_{p_0 | z} \left(1 - \frac{1}{p_0} \right), \quad (1.16)$$

де p_0 – просте число.

При використанні теореми Ейлера $a^{\varphi(z)} \bmod z = 1$ отримується: $a^{\varphi(z)-1} \bmod z = a^{-1} \bmod z$. Дана процедура передбачає виконання операції модулярного експоненціювання, що може привести до переповнення розрядної сітки процесора і ускладнює пошук оберненого елемента для багаторозрядних чисел [9].

Найбільш ефективними та поширеними є методи другої категорії, зокрема пошук мультиплікативного оберненого елемента за модулем на основі розширеного алгоритму Евкліда. Для цього спочатку потрібно записати прямий алгоритм Евкліда, згідно якого для будь–якого $z>a=r_0$, де z і a – цілі числа, виконується така система рівнянь:

$$
\begin{aligned}
&z = r_0 \cdot q_1 + r_1,\ q_1 = a,\ 0 \le r_1 < r_0;\\
&r_0 = r_1 \cdot q_2 + r_2,\ 0 \le r_2 < r_1;\\
&\dots\dots\dots\dots\dots\dots\\
&r_{j-3} = r_{j-2} \cdot q_{j-1} + r_{j-1},\ 0 \le r_{j-1} < r_{j-2};\\
&r_{j-2} = r_{j-1} \cdot q_j + r_j,\ 0 \le r_j < r_{j-1};\\
&r_{j-1} = r_j \cdot q_{j+1} + 0.
\end{aligned} \tag{1.17}
$$

Оскільки a і z є взаємно простими, то r_j=1. Далі для реалізації розширеного алгоритму Евкліда описану процедуру необхідно повторити в зворотньому порядку:

$$
\begin{gathered}
1 = r_j = r_{j-2} - q_j r_{j-1} = r_{j-2} - q_j \left(r_{j-3} - q_{j-1} r_{j-2}\right) = \\
r_{j-2} - q_j r_{j-3} + q_j q_{j-1} r_{j-2} = \\
= -q_j r_{j-3} + \left(1 + q_j q_{j-1}\right) r_{j-2} = -q_j r_{j-3} + \left(1 + q_j q_{j-1}\right)\left(r_{j-4} - q_{j-2} r_{j-3}\right) = \dots
\end{gathered} \tag{1.18}
$$

Даний процес продовжується до тих пір, поки не отримається вираз $v \cdot z + t \cdot r_0 = 1$, де величина $b = t \bmod z = a^{-1} \bmod z$ і буде шуканим оберненим елементом.

Даний метод характеризується великою кількістю ділень з остачею, перемножень і підстановок, хоча він володіє найменшою часовою складністю в порівнянні з іншими двома.

Методи пошуку оберненого елемента на основі алгоритму Евкліда можна розділити на два класи:

– методи, що ґрунтуються на класичному алгоритмі Евкліда;

– методи, що засновані на бінарному алгоритмі Евкліда.

Методи другого класу є більш ефективними, оскільки не використовують обчислювально-витратних операцій ділення на довільне число. Ці методи використовують лише елементарні операції, такі як додавання, віднімання і ділення на 2, що еквівалентне зсуву на один двійковий розряд праворуч [10].

1.3 Система залишкових класів

Дослідження, які проводились багатьма групами вчених з метою пошку шляху який підвищить продуктивність обчислювальних методів, методів для ефективної системи знаходження та виправлення помилок, а також для побудови кращих обчислювальних комплексів, які дають можливість стверджувати, що в межах позиційних систем числення не можна очікувати глобальних зрушень в цих напрямках без великого збільшення робочих частот та вдосконалення апаратної частини. Причина полягає в тому, що системи числення, в яких представляється і обробляється інформація мають важливий недолік – наявність міжрозрядних зв’язків. Таким чином буде ефективнішим використання непозиційних систем числення, які не мають цього недоліку.

Математика це наука абстрактна і в своєму чистому вигляді ніякого відношення до реального світу взагалі не має. Багато, звичайно спробують цю думку оскаржити. Однак змушує задуматися хоча б той факт, що різні народи в різні періоди часу навіть вважали по-різному. Варто тільки згадати вкрай неефективну систему числення стародавніх римлян. Греки замість чисел використовували літери власного алфавіту, як і слов'яни. У Вавилоні з'явилася шістдесяткова система числення, яка знайшла відображення в підрахунку часу 60 хвилин в годині, 60 секунд в хвилині і кутів. І єгиптяни звичайно ж відзначилися, першими придумавши десяткову, хоча і непозиційних, систему числення. Іншими словами – хто як придумав, той так і вважав [11].

Справжню революцію справила придумана в Індії не пізніше V століття позиційна десяткова система числення. Основним досягненням індійців стало використання в якості однієї з цифр нуля. Саме цей факт робить дану систему числення настільки зручною - тепер можна записати будь-яке число, використовуючи всього 10 різних символів і не вигадуючи новий для кожного

нового розряду. Величина числа залежить від позиції в ньому цифр і їх кількості.

Винахід сучасної позиційної системи числення, арабських цифр, дало серйозний імпульс розвитку математики. Була фактично вирішена проблема арифметичних операцій – їх тепер можна було виконувати інтуїтивно зрозумілим, швидким і ефективним способом, який не змінюється вже більше п'ятнадцяти століть. Подальший розвиток ідеї позиційності зробило принципово можливим створення сучасної обчислювальної техніки - вона функціонує на основі двійкової системи числення [12].

Однак, як показав досвід останніх десятиліть, в деяких випадках саме позиційний підхід до запису чисел викликає певні незручності. Основна проблема полягає в обробці чисел - раз від позиції цифри залежить величина числа, то в процесі розрахунків через перенесення вони повинні оброблятися послідовно, від молодшого до старшого. Очевидний висновок, знайоме з дитинства: спочатку складаємо одиниці, потім десятки, сотні, тисячі. На перший погляд нічого страшного в такій звичній картині немає. Але варто звернути увагу на можливості обчислювальної техніки.

Сучасні обчислювальні системи, незважаючи на всю свою міць, все-таки обмежені. І справа навіть не в тому, що вони не можуть, наприклад, зробити точний прогноз погоди або розшифрувати ДНК. Обмеженість проявляється в самій незначній, здавалося б, частини – діапазоні роботи з числами. Просто комп'ютер здатний працювати лише з обмеженим відрізком чисел, який визначається розрядної сіткою процесора. Багато пристроїв "розуміють" числа розміром 32 біта (двійкових розряди), інші – 64. Вкрай мало 128-ми бітних обчислювальних елементів, все інше – взагалі одиничні випадки. Мала розрядність чисел призводить до невисокої точності обчислень, а багато алгоритми і зовсім вимагають позамежної розрядності. Так наприклад схема шифрування RSA в сучасному варіанті вважається досить безпечною при довжині ключа 2048 біт, що викликає певні труднощі при реалізації.

Першу проблему, що виникає перед розробником в даній ситуації, можна сформулювати наступним чином: як робити обчислення з числами великої розрядності, якщо архітектура обчислювальних систем їх просто не підтримує. Але при найближчому розгляді це питання навіть не є проблемою. Відповідь на нього прихований в самій ідеї позиційних систем числення [13].

Раніше ми прийшли до висновку, що позиційні не завжди добре. Розглянемо тепер непозиційних підхід до систем числення. Під непозиційних розуміється відсутність очевидного зв'язку між розрядами або, іншими словами, відсутність переносів. Однією з найбільш використовуваних непозиційних систем числення є Система залишкових класів (СОК, Residue Number System – RNS). СЗК грунтується на теорії порівнянь і була запропонована радянським математиком Ізраїлем Яковичем Акушський в 50-ті роки двадцятого століття. Теорію обчислень в СЗК іноді називають модулярной арифметикою, основною теоремою якої є Китайська теорема про залишки (КТО, Chinese remainder theorem – CRT), одне з формулювань якої наведена нижче.

Переваги такого підходу очевидні: ми виконуємо операції одночасно по різних підставах так як вони не залежать один від одного. Іншими словами, крім зниження розрядності ми додаємо паралелізм, який принципово неможливий в позиційній системі. Це ідеально лягає під сучасні обчислювальні засоби: паралельні обчислення розвиваються останнім часом максимально можливими темпами, проявляючись не тільки в архітектурі комп'ютерів, а й при розробці вбудованих рішень, заснованих, наприклад, на ПЛІС архітектури FPGA.

Проведені дослідження показують, що в багатьох задачах використання СОК спільно з паралельними обчисленнями призводить до значного збільшення продуктивності [14]. Наприклад, операція піднесення до степеня в СОК для великих чисел може бути реалізована вкрай ефективно, що призведе до сверхлінейному прискоренню.

Загальною проблемою модулярної арифметики і СЗК зокрема є існування так званих немодульність операцій. До такого класу операцій відносяться, наприклад, порівняння та ділення чисел. Дані операції мають позиційну природу і не можуть бути виконані без обчислення будь-якої характеристики числа, що визначає його позицію в числовому ряді, що призводить до відновлення позиційного представлення числа в тому чи іншому роді. Роботи багатьох дослідників спрямовані на максимальну оптимізацію немодульность операцій в СЗК. Однак найбільш ефективними залишаються алгоритми, що зводять використання таких операцій до мінімуму. Доброю областю для застосування СЗК таким чином є криптографія, що зводиться до множення, додавання і зведення в ступінь.

Незважаючи на всі свої переваги СЗК, як було зазначено раніше, не може бути панацеєю від усіх бід в довгій арифметиці, проте конкретні проблемно-орієнтовані завдання в ній реалізуються вкрай ефективно. Найбільший позитивний ефект СОК вже зіграла в таких напрямках, крім уже описаного:

СЗК дозволяє спростити і зменшити архітектуру обчислювальних електронних пристроїв, за рахунок чого підвищується не тільки швидкість, але і енергетична ефективність продуктів.

СЗК часто використовується як засіб для синтезу пристроїв високої надійності, її "паралельна" природа дозволяє за рахунок введення надлишкових підстав будувати високоефективні перешкодостійкі коди.

Криптографічні програми не обмежуються одним лише прискоренням роботи з довгими числами. СЗК є хорошою базою для побудови схем поділу секрету, що володіють вигідними щодо аналогів властивостями.

У системах цифрової обробки сигналів (ЦОС, DSP) СЗК показала себе як вигідний інструмент підвищення ефективності цифрових фільтрів, про що свідчить велика кількість робота в даній області.

Деякі додатки знайшла СЗК в системах зв'язку, прикладом чого є оптимізація технології CDMA за рахунок впровадження модулярних перетворень.

Всі переваги, що виникають при застосуванні СЗК, отримані завдяки тому, що ми змінили свій погляд на системи числення. Іноді нетривіальний погляд на математичні об'єкти може дати перспективні результати.

Сама сутність сучасних персональних комп'ютерів робить їх залежними від систем числення і від правил, які визначають зв'язок між числами. Однак, залишкова система, або система залишкових класів, яка є темою цієї статті, являє додаткове відхилення від двійковій і десяткової систем чисел, ніж остання один від одного, так як деякі основні концепції, такі як використання фіксованих підстав (наприклад, 10 або 2) . В результаті цих основних відмінностей залишкова арифметика, яка є маніпуляцією чисел, виражених в системі залишкових класів, пропонує незвичайний набір характеристик.

Наприклад, залишкова арифметика забезпечує можливість підсумувати, віднімати або множити в один етап, незважаючи на протяжність чисел, не вдаючись до проміжних цифр перенесення або внутрішнім затримок. Таким чином, швидше за зміною в системі чисел, ніж додаванням логічних схем і устаткування, можливо в принципі змусити комп'ютер працювати швидше. На жаль, залишкова арифметика також має атрибути, які є скрутними і збитковими настільки, що на практиці технологія використання не завжди можлива.

Границя числової системи визначається як інтервал, над яким кожне ціле число може бути представлено системою через брак двох чисел з одним і тим самим поданням [15]. Очевидно, що десяткова система має невизначені кордони, проте в практиці застосовуються усічені системи чисел, тому що фізичні системи можуть вмістити лише обмежену кількість цифр. Так в комп'ютерах числовий межа визначається точно становищем довжини слова і типом використовуваної системи числення.

Подання числа називається унікальним, якщо кожне число в системі має тільки одне подання. Наприклад, двійкова система числення, часто використовувана в комп'ютерах, є унікальною. Однак багато які зазвичай використовуються системи чисел не є чимось унікальним. Наприклад,

десяткова система числення, розширена для включення негативних чисел, зазвичай має два подання для нуля: + 0 і - 0. Очевидно, що будь-яка система з поданням знака і величини модуля також не унікальна. Існують деякі системи числення, які використовують поперемінні уявлення для всіх чисел. Одна така система була придумана для того, щоб обмежити знак перенесення до певного числа цифр, в той час як інша використовується для того, щоб досягти корекції помилок в обчисленнях [16]. Основний збиток цих уявлень в тому, що числа мають бути стандартної специфічної форми для деяких арифметичних операцій.

Система числення буде надлишковою, якщо існує менше чисел, ніж комбінацій цифр. Альтернативно різні комбінації можуть стосуватися одного і того ж числа. Очевидно, неунікальність увазі надмірність. Наприклад, двійкова система з парними цифрами, додані в останню позицію цифр, має унікальну виставу для кожного числа, і так як половина комбінацій заборонені, це уявлення є надмірною.

Однією з непозиційних систем числення є код Грея, який широко використовується в аналого-цифровому перетворенні і перерахункових пристроях. Він дозволяє істотно зменшити час перетворення сигналу за рахунок спрощення кодує логіки, що, в свою чергу, дозволяє підвищити ефективність захисту при переходах вхідного сигналу від небажаних збоїв. Даному коду властива важлива риса: два суміжних цілих числа мають в усьому ідентичні уявлення на одну цифрову позицію. Алгоритм переходу довічного сигналу до коду Грея можна представити у формі наступного правила: старші розряди двійкового сигналу збігаються, а будь-який наступний розряд визначається сумою по модулю 2 відповідного і попереднього розряду коду двійкового сигналу.

Так як модулярні системи пропонують кілька цікавих альтернатив, можна брати до уваги ефект заміни традиційної системи на систему залишкових класів. Як виділялося раніше, даний використання систем числення з фіксованим підставою в комп'ютерах, як правило, має ряд

негативних рис. Множення і ділення більш складні для виконання, ніж додавання і віднімання. Більш того, додавання, яке саме по собі є основною операцією і яке також використовується у виконанні множення, не завжди легко виконати, так як кожна цифра суми – це функція всіх менш значних цифр операндів. Хоча це властивість може не бути негативною рисою в обчисленнях вручну або серійних обчисленнях, в паралельній обробці мається на увазі або подовжене час виконання, або необхідність у великій кількості обладнання.

1.4 Висновок до першого розділу

В даному розділі було поставлене завдання про створення власного алгоритму вирішення китайської теореми про залишки та реалізація цього алгоритму. Під час створення алгоритму потрібно буде знайти не тільки новий метод пошуку невідомої, але й зробити так щоб знайдене значення задовольняло усі рівняння та час виконання створеного алгоритму був менший ніж у алгоритмів що уже існують.

2 АНАЛІЗ КРИПТОСИСТЕМ ТА МЕТОДИ ПРОТИДІЇ

2.1 Алгоритм Евкліда

Поширеним застосуванням розширеного алгоритму Евкліда є китайська теорема про залишки – один з найдавніших, але досить важливий на даний час обчислювальний алгоритм.

Ще в першому столітті нашої ери китайський математик Сунь–Цзи придумав цікаву загадку, якою було покладено початок модулярній арифметиці: потрібно було знайти число, яке при діленні на 3 дасть в остачі 2, на 5 – 3, на 7 – 2. Крім того, він показав у частковому випадку еквівалентність розв’язку системи модулярних рівнянь і розв’язку одного модулярного рівняння.

Протягом майже двох тисяч років китайська теорема про залишки постійно вдосконалювалася та розвивалася. Зокрема, в XIII столітті інший китайський математик Цань Цзю–шао розв’язав наведену вище задачу. У XVIII столітті німецький математик Л.Ейлер навів загальне формулювання та доведення КТЗ, а К.–Ф. Гаус істотно розвинув його в своїх знаменитих „Арифметичних дослідженнях” [17].

І, нарешті, в середині XX століття чеські учені М.Валах та А.Свобода запропонували використати древню китайську ідею на новому технічному рівні, створивши перші модулярні електронно–обчислювальні машини „Епос” та „Епос–2”.

Слід зазначити, що на даний час існує декілька еквівалентних формулювань КТЗ. Найбільш поширене з них таке: якщо натуральні числа p_1, p_2, …, p_j попарно взаємно прості, то для будь–яких цілих r_1, r_2, …, r_j, таких що $0 \le r_i < p_i$ існує число A, яке при діленні на p_i дає залишок r_i при всіх $i=1, 2, ..., j$; більше того, якщо існує два таких числа A_1 та A_2.

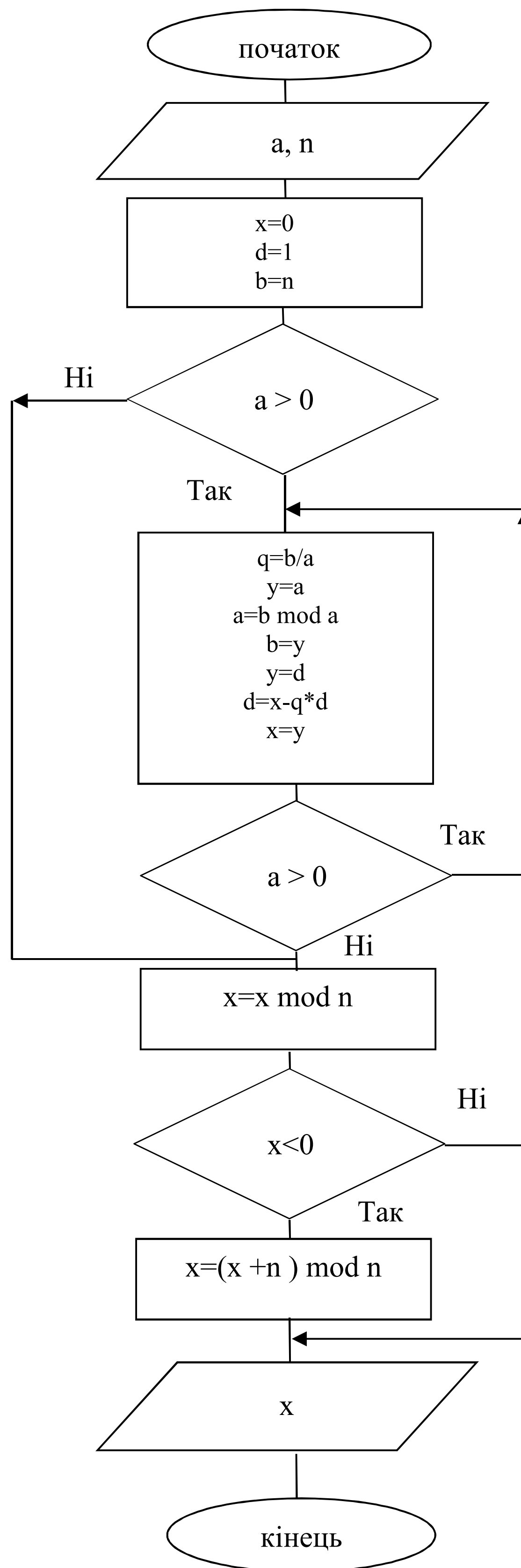

Рисунок 2.1 – Блок–схема розширеного алгоритму Евкліда

Дана теорема представляється переважно у вигляді такої системи порівнянь:

$$\begin{cases} A \bmod p_1 = r_1 \\ A \bmod p_2 = r_2 \\ \dots\dots\dots\dots\dots\dots \\ A \bmod p_i = r_i \\ \dots\dots\dots\dots\dots\dots \\ A \bmod p_j = r_j. \end{cases} \qquad (1.19)$$

Шукане число обчислюється за формулою:

$$A = \left(\sum_{i=1}^{j} M_i m_i r_i \right) \bmod P, \qquad (1.20)$$

де $M_i = \dfrac{P}{p_i} = p_1 p_2 \dots p_{i-1} p_{i+1} \dots p_j$, $m_i = M_i^{-1} \bmod p_i$.

Як було сказано вище, пошук мультиплікативного оберненого елемента, необхідний для реалізації КТЗ, характеризується значною обчислювальною складністю в зв'язку з неможливістю розпаралелення процесу обчислень. Причому всі ці операції повинні виконуватися над дуже великими числами, що може призвести до переповнення розрядної сітки.

2.2 Аналіз по стороннім каналам та його види

Всі техніки злому криптографічних систем поділяють на дві великі групи: використовують недоліки самих алгоритмів шифрування і їх фізичних

реалізацій. У цій статті ми розглянемо останні, які називають SCA повна назва side-channel attacks – атаки по стороннім або побічним канала [18].

На відміну від абстрактної математичної моделі, будь-яка фізична реалізація шіфрсістеми не може бути повністю ізольована. Вона завжди складається з якихось серійно випускаються компонентів, що мають свої особливості роботи. Наприклад, криптомодуль неоднаково споживає електроенергію під час різних бітових операцій, створює характерні радіочастотні перешкоди, відчуває відрізняються в залежності від вхідних даних затримки, сильніше нагрівається в одному випадку і слабкіше в іншому. Все це – непрямі дані, які дозволяють дізнатися секретну інформацію, не маючи до неї прямого доступу.

На практиці SCA в різних варіантах використовуються дуже широко – від підслуховування паролів до зчитування захищених областей пам'яті в обхід ізоляції адресного простору, причому необов'язково на локальній машині.

Суть SCA полягає в тому, щоб замість лобової атаки перехопити якісь побічні сигнали, що виникають при обробці ізольованих або зашифрованих даних. Потім по цих сигналах намагаються відновити секретну інформацію (пароль, хеш, ключ шифрування, текст повідомлення) без прямого звернення до захищених даних [19].

Найпростіший приклад: ти хочеш дізнатися пароль свого колеги, але не можеш підглянути його під час набору. При цьому на слух вдається визначити його довжину і одноразове використання пробілу – ця клавіша звучить дуже своєрідно. Без будь-якого обладнання і хитрих програм ти вже дізнався багато про пароль, просто насторочивши вуха.

Якщо ж зробити кілька аудіозаписів того, як логинится колега, а потім застосувати статистичний аналіз, то ти зміг би відновити весь пароль або його більшу частину. Поодинокі невпевнено розпізнані символи все одно підбираються по масці за короткий час.

Такий же по суті, але більш складний в плані реалізації, метод атаки по стороннім каналах використовувало британське агентство GCHQ в середині

шістдесятих. Воно встановило мікрофони в єгипетському посольстві і записували звуки, які видає механічна шифрувальну машину. За ним з'ясували початкові положення двох символьних дисків, а далі розкрили схему шифрування звичайним перебором.

Сьогодні SCA рідко виконуються по акустичному каналу – хіба що це робиться в якості академічного дослідження. Наприклад, відомий криптограф Аді Шамір поставив собі за мету відновити ключ RSA по диктофонного запису звукових паттернів шифрування. (Так, електроніка теж по-різному скрипить, коли виконує типові операції.) Йому в результаті вдалося це зробити, правда в абсолютно нетипових умовах: комп'ютер безперервно шифрував одним і тим же ключем протягом години, всі інші процеси були вивантажені.

Куди частіше на практиці вимірюють електромагнітне випромінювання. Воно теж змінюється в залежності від того, який скан-код відправляється комп'ютера з клавіатури і які інструкції виконують різні чіпи. В українській мові це називається реєстрація ПЕМВН (побічних електромагнітних випромінювань і наведень), а в англомовній літературі – TEMPEST (Transient Electromagnetic Pulse Emanation Standard).

Абревіатура TEMPEST була взята з однойменної секретної програми США сімдесятих років. До теперішнього часу на її основі був розроблений цілий набір стандартів, де описані вимоги для захисту обладнання різного класу від демаскируючих ЕМ-випромінювань. У нульових роках термін TEMPEST стали використовувати для позначення будь-якої атаки, заснованої на реєстрації побічної ЕМІ.

У сучасному варіанті для аналізу ПЕМВН найчастіше використовуються програмовані радіосистеми. Наприклад, в роботі Stealing Keys from PCs using a Radio автори показують, як за допомогою ресивера FUNcube Dongle Pro можна розкрити RSA ключі, перебуваючи за півметра від обробного їх ноутбука.

Ще одна цікава модифікація цієї атаки – безперервне вимірювання електричного потенціалу на корпусі ноутбука під час шифрування або

дешифрування. Зіставивши графік з відомим шифртекст і алгоритмом, можна обчислити ключ навіть дуже великої довжини.

Безпека є проблемою в обчислювальних і комунікаційних системах і значна дослідницька робота призначена для збільшення. Криптографічні алгоритми, симетричні шифри, шифри з відкритими ключами і хеш-функції утворюють безліч примітивів, які використовуються у вигляді блоків для побудови механізмів безпеки, та мають цільовий характер. Наприклад, протоколи безпеки мережі, такі як SSH і TLS, об'єднують ці примітиви для забезпечення аутентифікації між взаємодіючими суб'єктами і забезпечують конфіденційність і цілісність повідомляються відомостей. На практиці ці механізми безпеки тільки вказують, які функції повинні бути виконані, незалежно від того, як ці функції реалізовані [20]. Наприклад, специфікація протоколу безпеки зазвичай не залежить від того, чи реалізуються алгоритми шифрування в програмному забезпеченні, що працює в центральному процесорі, або за допомогою спеціальних апаратних компонентів, і чи знаходиться пам'ять, яка використовується для зберігання проміжних даних під час цих обчислень, на одному і тому ж чіпі в обчислювально пристрої або на окремому чіпі.

Це свого роду "поділ завдань" між механізмами забезпечення безпеки і їх реалізацією стало можливо (і навіть необхідно) завдяки суворому теоретичному аналізу та розробки криптосистем і протоколів безпеки. Проте, в процесі реалізації механізмів безпеки були зроблені різні припущення. Наприклад, як правило, передбачається, що реалізація криптографічних обчислень – це ідеальний "чорний ящик", чиє внутрішнє зміст не може бути переглянуто і не може постраждати від втручання будь-якого шкідливого об'єкта. За допомогою цих припущень, рівень безпеки широко визначається кількістю з точки зору математичних властивостей криптографічних алгоритмів і їх ключових розмірів.

Проте, на практиці ці механізми безпеки в поодинці далеко не повні рішення для забезпечення безпеки. Неможливо припустити, що зловмисники

спробують безпосередньо порахувати обчислювальну складність злому шифрувальних примітивів, які використовуються в механізмах безпеки. Цікаву аналогію можна провести в зв'язку з цим між сильними криптографічними алгоритмами і вельми надійним замком на вхідних дверях будинку. Грабіжники, які намагаються увірватися в будинок, рідко будуть перебирати всі комбінації, необхідні, щоб відімкнути замок. Вони можуть вдертися через вікна, зірвати двері з петель, або відняти у власника ключі, так як вони намагаються потрапити в будинок [21]. Точно так же майже всі відомі атаки на безпеку на криптографічні системи націлені на слабкі місця в реалізації і розміщенні механізмів і їх криптографічних алгоритмів. Ці недоліки можуть дозволити зловмисникам повністю обійти, або значно послабити, теоретичну міцність рішень в області безпеки.

Для криптографічного системи, щоб залишатися в безпеці, важливо, щоб секретні ключі, що використовуються для виконання необхідних служб безпеки, ні в якому разі не були виявлені. Так як самі криптографічні алгоритми вивчалися протягом довгого часу великою кількістю експертів, хакери частіше намагаються атакувати апаратні засоби і систему, всередині яких розміщена криптографічний блок. Новий клас атак розроблявся останні кілька років Кохером. Ці атаки діють, тому що існує кореляція між фізичними вимірами, зробленими в різних точках в процесі обчислення, і внутрішнім станом пристрою обробки, яка сама по собі пов'язане з секретним ключем.

Насправді, криптографічні алгоритми завжди реалізуються в програмному або апаратному забезпеченні на фізичних пристроях, які взаємодіють з навколишнім середовищем і знаходяться під її впливом. Ці фізичні взаємодії можуть бути спровоковані і контрольовані супротивниками, такими як Єва, і можуть дати інформацію, корисну в криптоанализе [22]. Цей тип інформації називається побічним каналом інформації, і атаки, що використовують побічні канали інформації називаються атаками по побічним каналам інформації, згодом SCA sidechannel attacks. Основна ідея атак по

побічним каналам – подивитися, як реалізовані криптографічні алгоритми, а не на сам алгоритм.

Не важко помітити, що традиційний криптоаналіз розглядає криптографічні алгоритми, як чисто математичні об'єкти, в той час як криптоаналіз по побічним каналам також бере до уваги реалізацію алгоритмів [23]. Таким чином, атака по побічним каналам також називається реалізаційної атакою. Навіть будь-який криптографічний алгоритм повинен бути закодований, щоб функціонувати належним чином, такі закодовані алгоритми не повинні розкривати відомості про закрите ключі, не дивлячись на те, що противник здатний спостерігати і маніпулювати запущеним алгоритмом.

На відміну від абстрактної математичної моделі, будь-яка фізична реалізація шіфрсістеми не може бути повністю ізольована. Вона завжди складається з якихось серійно випускаються компонентів, що мають свої особливості роботи. Наприклад, криптомодуль неоднаково споживає електроенергію під час різних бітових операцій, створює характерні радіочастотні перешкоди, відчуває відрізняються в залежності від вхідних даних затримки, сильніше нагрівається в одному випадку і слабкіше в іншому. Все це – непрямі дані, які дозволяють дізнатися секретну інформацію, не маючи до неї прямого доступу.

На практиці SCA в різних варіантах використовуються дуже широко – від підслуховування паролів до зчитування захищених областей пам'яті в обхід ізоляції адресного простору, причому необов'язково на локальній машині.

Суть SCA полягає в тому, щоб замість лобової атаки перехопити якісь побічні сигнали, що виникають при обробці ізольованих або зашифрованих даних. Потім по цих сигналах намагаються відновити секретну інформацію (пароль, хеш, ключ шифрування, текст повідомлення) без прямого звернення до захищених даних [19].

Найпростіший приклад: ти хочеш дізнатися пароль свого колеги, але не можеш підглянути його під час набору. При цьому на слух вдається визначити його довжину і одноразове використання пробілу – ця клавіша звучить дуже

своєрідно. Без будь-якого обладнання і хитрих програм ти вже дізнався багато про пароль, просто насторочивши вуха.

Якщо ж зробити кілька аудіозаписів того, як логинится колега, а потім застосувати статистичний аналіз, то ти зміг би відновити весь пароль або його більшу частину. Поодинокі невпевнено розпізнані символи все одно підбираються по масці за короткий час.

Такий же по суті, але більш складний в плані реалізації, метод атаки по стороннім каналах використовувало британське агентство GCHQ в середині шістдесятих. Воно встановило мікрофони в єгипетському посольстві і записували звуки, які видає механічна шифрувальну машину. За ним з'ясували початкові положення двох символьних дисків, а далі розкрили схему шифрування звичайним перебором.

Сьогодні SCA рідко виконуються по акустичному каналу – хіба що це робиться в якості академічного дослідження. Наприклад, відомий криптограф Аді Шамір поставив собі за мету відновити ключ RSA по диктофонного запису звукових паттернів шифрування. (Так, електроніка теж по-різному скрипить, коли виконує типові операції.) Йому в результаті вдалося це зробити, правда в абсолютно нетипових умовах: комп'ютер безперервно шифрував одним і тим же ключем протягом години, всі інші процеси були вивантажені.

Куди частіше на практиці вимірюють електромагнітне випромінювання. Воно теж змінюється в залежності від того, який скан-код відправляється комп'ютера з клавіатури і які інструкції виконують різні чіпи. В українській мові це називається реєстрація ПЕМВН (побічних електромагнітних випромінювань і наведень), а в англомовній літературі – TEMPEST (Transient Electromagnetic Pulse Emanation Standard).

Абревіатура TEMPEST була взята з однойменної секретної програми США сімдесятих років. До теперішнього часу на її основі був розроблений цілий набір стандартів, де описані вимоги для захисту обладнання різного класу від демаскирующих ЕМ-випромінювань. У нульових роках термін

TEMPEST стали використовувати для позначення будь-якої атаки, заснованої на реєстрації побічної ЕМІ.

У сучасному варіанті для аналізу ПЕМВН найчастіше використовуються програмовані радіосистеми. Наприклад, в роботі Stealing Keys from PCs using a Radio автори показують, як за допомогою ресивера FUNcube Dongle Pro можна розкрити RSA ключі, перебуваючи за півметра від обробного їх ноутбука.

Ще одна цікава модифікація цієї атаки – безперервне вимірювання електричного потенціалу на корпусі ноутбука під час шифрування або дешифрування. Зіставивши графік з відомим шифртекст і алгоритмом, можна обчислити ключ навіть дуже великої довжини.

Безпека є проблемою в обчислювальних і комунікаційних системах і значна дослідницька робота призначена для збільшення. Криптографічні алгоритми, симетричні шифри, шифри з відкритими ключами і хеш-функції утворюють безліч примітивів, які використовуються у вигляді блоків для побудови механізмів безпеки, та мають цільовий характер. Наприклад, протоколи безпеки мережі, такі як SSH і TLS, об'єднують ці примітиви для забезпечення аутентифікації між взаємодіючими суб'єктами і забезпечують конфіденційність і цілісність повідомляються відомостей. На практиці ці механізми безпеки тільки вказують, які функції повинні бути виконані, незалежно від того, як ці функції реалізовані [20]. Наприклад, специфікація протоколу безпеки зазвичай не залежить від того, чи реалізуються алгоритми шифрування в програмному забезпеченні, що працює в центральному процесорі, або за допомогою спеціальних апаратних компонентів, і чи знаходиться пам'ять, яка використовується для зберігання проміжних даних під час цих обчислень, на одному і тому ж чіпі в обчислювально пристрої або на окремому чіпі.

Це свого роду "поділ завдань" між механізмами забезпечення безпеки і їх реалізацією стало можливо (і навіть необхідно) завдяки суворому теоретичному аналізу та розробки криптосистем і протоколів безпеки. Проте, в процесі реалізації механізмів безпеки були зроблені різні припущення.

Наприклад, як правило, передбачається, що реалізація криптографічних обчислень – це ідеальний "чорний ящик", чиє внутрішнє зміст не може бути переглянуто і не може постраждати від втручання будь-якого шкідливого об'єкта. За допомогою цих припущень, рівень безпеки широко визначається кількістю з точки зору математичних властивостей криптографічних алгоритмів і їх ключових розмірів.

Проте, на практиці ці механізми безпеки в поодинці далеко не повні рішення для забезпечення безпеки. Неможливо припустити, що зловмисники спробують безпосередньо порахувати обчислювальну складність злому шифрувальних примітивів, які використовуються в механізмах безпеки. Цікаву аналогію можна провести в зв'язку з цим між сильними криптографічними алгоритмами і вельми надійним замком на вхідних дверях будинку. Грабіжники, які намагаються увірватися в будинок, рідко будуть перебирати всі комбінації, необхідні, щоб відімкнути замок. Вони можуть вдертися через вікна, зірвати двері з петель, або відняти у власника ключі, так як вони намагаються потрапити в будинок [21]. Точно так же майже всі відомі атаки на безпеку на криптографічні системи націлені на слабкі місця в реалізації і розміщенні механізмів і їх криптографічних алгоритмів. Ці недоліки можуть дозволити зловмисникам повністю обійти, або значно послабити, теоретичну міцність рішень в області безпеки.

Для криптографічного системи, щоб залишатися в безпеці, важливо, щоб секретні ключі, що використовуються для виконання необхідних служб безпеки, ні в якому разі не були виявлені. Так як самі криптографічні алгоритми вивчалися протягом довгого часу великою кількістю експертів, хакери частіше намагаються атакувати апаратні засоби і систему, всередині яких розміщена криптографічний блок. Новий клас атак розроблявся останні кілька років Кохером. Ці атаки діють, тому що існує кореляція між фізичними вимірами, зробленими в різних точках в процесі обчислення, і внутрішнім станом пристрою обробки, яка сама по собі пов'язане з секретним ключем.

Насправді, криптографічні алгоритми завжди реалізуються в програмному або апаратному забезпеченні на фізичних пристроях, які взаємодіють з навколишнім середовищем і знаходяться під її впливом. Ці фізичні взаємодії можуть бути спровоковані і контрольовані супротивниками, такими як Єва, і можуть дати інформацію, корисну в криптоанализе [22]. Цей тип інформації називається побічним каналом інформації, і атаки, що використовують побічні канали інформації називаються атаками по побічним каналам інформації, згодом SCA sidechannel attacks. Основна ідея атак по побічним каналам – подивитися, як реалізовані криптографічні алгоритми, а не на сам алгоритм.

Не важко помітити, що традиційний криптоаналіз розглядає криптографічні алгоритми, як чисто математичні об'єкти, в той час як криптоаналіз по побічним каналам також бере до уваги реалізацію алгоритмів [23]. Таким чином, атака по побічним каналам також називається реалізаційної атакою. Навіть будь-який криптографічний алгоритм повинен бути закодований, щоб функціонувати належним чином, такі закодовані алгоритми не повинні розкривати відомості про закрите ключі, не дивлячись на те, що противник здатний спостерігати і маніпулювати запущеним алгоритмом. Ще одна цікава модифікація цієї атаки – безперервне вимірювання електричного потенціалу на корпусі ноутбука під час шифрування або дешифрування. Зіставивши графік з відомим шифртекст і алгоритмом, можна обчислити ключ навіть дуже великої довжини.

Це свого роду "поділ завдань" між механізмами забезпечення безпеки і їх реалізацією стало можливо (і навіть необхідно) завдяки суворому теоретичному аналізу та розробки криптосистем і протоколів безпеки. Проте, в процесі реалізації механізмів безпеки були зроблені різні припущення. Наприклад, як правило, передбачається, що реалізація криптографічних обчислень – це ідеальний "чорний ящик", чиє внутрішнє зміст не може бути переглянуто і не може постраждати від втручання будь-якого шкідливого об'єкта. За допомогою цих припущень, рівень безпеки широко визначається

кількістю з точки зору математичних властивостей криптографічних алгоритмів і їх ключових розмірів.

Проте, на практиці ці механізми безпеки в поодинці далеко не повні рішення для забезпечення безпеки. Неможливо припустити, що зловмисники спробують безпосередньо порахувати обчислювальну складність злому шифрувальних примітивів, які використовуються в механізмах безпеки. Цікаву аналогію можна провести в зв'язку з цим між сильними криптографічними алгоритмами і вельми надійним замком на вхідних дверях будинку. Грабіжники, які намагаються увірватися в будинок, рідко будуть перебирати всі комбінації, необхідні, щоб відімкнути замок. Вони можуть вдертися через вікна, зірвати двері з петель, або відняти у власника ключі, так як вони намагаються потрапити в будинок. Точно так же майже всі відомі атаки на безпеку на криптографічні системи націлені на слабкі місця в реалізації і розміщенні механізмів і їх криптографічних алгоритмів. Ці недоліки можуть дозволити зловмисникам повністю обійти, або значно послабити, теоретичну міцність рішень в області безпеки.

Для криптографічного системи, щоб залишатися в безпеці, важливо, щоб секретні ключі, що використовуються для виконання необхідних служб безпеки, ні в якому разі не були виявлені. Так як самі криптографічні алгоритми вивчалися протягом довгого часу великою кількістю експертів, хакери частіше намагаються атакувати апаратні засоби і систему, всередині яких розміщена криптографічний блок.

Новий клас атак розроблявся останні кілька років Кохером. Ці атаки діють, тому що існує кореляція між фізичними вимірами, зробленими в різних точках в процесі обчислення, і внутрішнім станом пристрою обробки, яка сама по собі пов'язане з секретним ключем.

Насправді, криптографічні алгоритми завжди реалізуються в програмному або апаратному забезпеченні на фізичних пристроях, які взаємодіють з навколишнім середовищем і знаходяться під її впливом. Ці фізичні взаємодії можуть бути спровоковані і контрольовані супротивниками,

такими як Єва, і можуть дати інформацію, корисну в криптоанализе. Цей тип інформації називається побічним каналом інформації, і атаки, що використовують побічні канали інформації називаються атаками по побічним каналам інформації, згодом SCA sidechannel attacks. Основна ідея атак по побічним каналам – подивитися, як реалізовані криптографічні алгоритми, а не на сам алгоритм.

Не важко помітити, що традиційний криптоаналіз розглядає криптографічні алгоритми, як чисто математичні об'єкти, в той час як криптоаналіз по побічним каналам також бере до уваги реалізацію алгоритмів. Таким чином, атака по побічним каналам також називається реалізаційної атакою.

Навіть будь-який криптографічний алгоритм повинен бути закодований, щоб функціонувати належним чином, такі закодовані алгоритми не повинні розкривати відомості про закрите ключі, не дивлячись на те, що противник здатний спостерігати і маніпулювати запущеним алгоритмом.

Ще одна цікава модифікація цієї атаки – безперервне вимірювання електричного потенціалу на корпусі ноутбука під час шифрування або дешифрування. Зіставивши графік з відомим шифртекст і алгоритмом, можна обчислити ключ навіть дуже великої довжини.

2.3 Опис розробки та створення власного алгоритму

Розробка власного алгоритму обчислення Китайської теореми про залишки розпочалась з перебору різних можливих алгоритмів вирішення цієї теореми, щоб відповідь задовільняла рівняння у всіх можливих значеннях уже відомих змінних [32]. Багато варіантів створення цього алгоритму працювали лише з певними значеннями змінних, але мені потрібно щоб алгоритм вирішував рівняння з будь-якими значеннями.

Перебравши безліч можливих рішень цього завдання було створено алгоритм обчислення Китайської теореми про залишки наведеним алгоритмом на рисунку 2.2.

```cpp
34
35    __int64 cof1 = p3+b3;
36
37    __int64 cof2 = cof1;
38
39    if (cof2%b2==p2){
40    }
41    else {
42        for (int i=0; cof2%b2!=p2; i++){
43        cof2 = cof2+b3;
44    }
45    }
46
47    __int64 cof3 = cof2;
48
49    if (cof3%b1==p1){
50    }
51    else {
52        for (int i=0; cof3%b1!=p1; i++){
53        cof3 = cof3+(b3*b2);
54
55    }
56    }
```

Рисунок 2.2 – Програмна реалізація алгоритму обчислення Китайської теореми про залишки на мові С++

В представленному алгоритмі обчислення Китайської теореми про залишки невідоме значення змінної знаходиться шляхом перебору значень та зміною цього значення з врахування кожної з нерівності так, щоб значення невідомої задовільняло усі нерівності [24]. Для того щоб задовільнити усі нерівності ми беремо відомі значення та шукаємо їх суму, а якщо сума цих чисел не задовільняє наше рівняння ми ще раз додаємо змінну що іде до знаку рівності то тих пір поки нерівність не буде вирішино.

Точно таким же чином алгоритм підшукує значення до усіх наступних нерівностей.

Реалізація цього алгоритму розпочинається з введення значень нерівності. Як реалізованно присвоє значень у нерівності показано на рисунку 2.3.

```
11
12  cout << "Mod 1 = " << endl;
13  cin >> b1;
14  cout << "Сума 1 = " << endl;
15  cin >> p1;
16
17  cout << "Mod 2 = " << endl;
18  cin >> b2;
19  cout << "Сума 2 = " << endl;
20  cin >> p2;
21
22  cout << "Mod 3 = " << endl;
23  cin >> b3;
24  cout << "Сума 3 = " << endl;
25  cin >> p3;
26
```

Рисунок 2.3 – Програмна реалізація присвоєння значень у нерівності на мові C++

Після того як буде введено значення усіх змінних запускається алгоритм підбору першого значення невідомої та шляхом додавання двох відомих нам змінних ми отримуємо значення невідомої яке задовольнить нашу нерівність, на рисунку 3.3 наведено алгоритм підбору першого значення невідомої для першої нерівності.

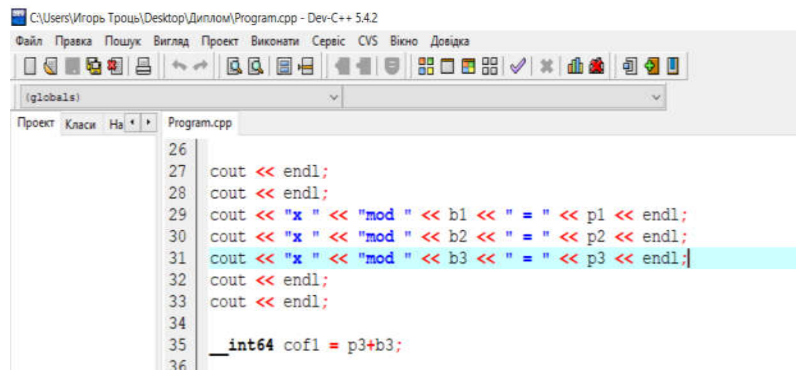

Рисунок 2.4 – Програмна реалізація алгоритму підбору першого значення невідомої для першої нерівності на мові C++

Після того як знайдено перше значення невідомої запускається наступний алгоритм який спочатку перевіряє чи не підійде перше значення невідомої до другої нерівності, якщо це значення підходить то алгоритм переходить до третього етапу, якщо ні виконується ще один підбор другого значення невідомої яке підійде до другої нерівності [25].

Цей підбор виконується шляхом додавання до першого значення невідомої значення відомої змінної з попередньої нерівності що стоїть перед знаком рівності і так до тих пір поки друге значення невідомої не задовільнить другу нерівність, приклад такого підбору наведено на рисунку 2.5.

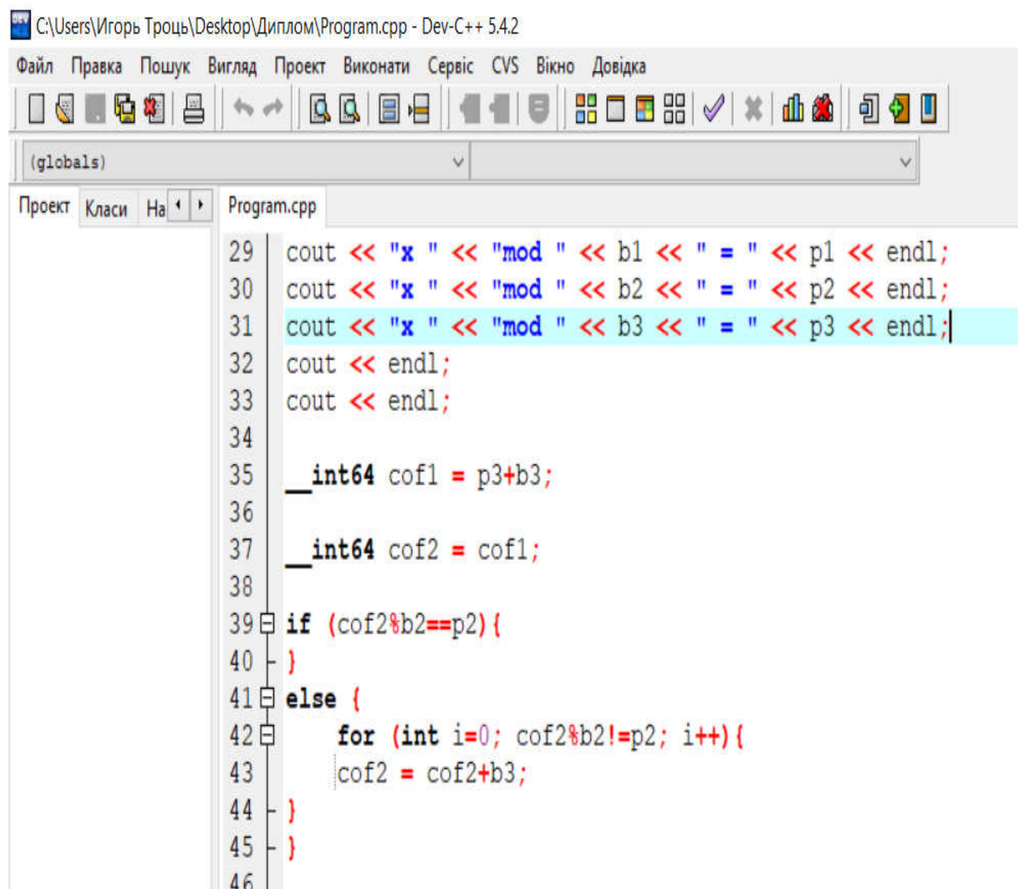

Рисунок 2.5 – Програмна реалізація алгоритму підбору другого значення невідомої для другої нерівності на мові C++

Знайшовши друге значення невідомої яке задовільняє і першу і другу нерівність приступає до роботи третій етап пошуку невідомої [31].

Він прицює за рпинципом алгоритму наведенного в попередньому етапі, тільки замість того щоб додавати значення змінної з першої нерівності ми додаєму значення змінної яка стоїть перед знаком рівності з попередньої нерівності, цей процес також буде тривати до тих пір поки шляхом постійного додавання змінної ми не отримаємо значення невідомої що задовольнить третю

нерівність, алгоритм підбору третього значення невідомої для третьої нерівності наведено на рисунку 2.6.

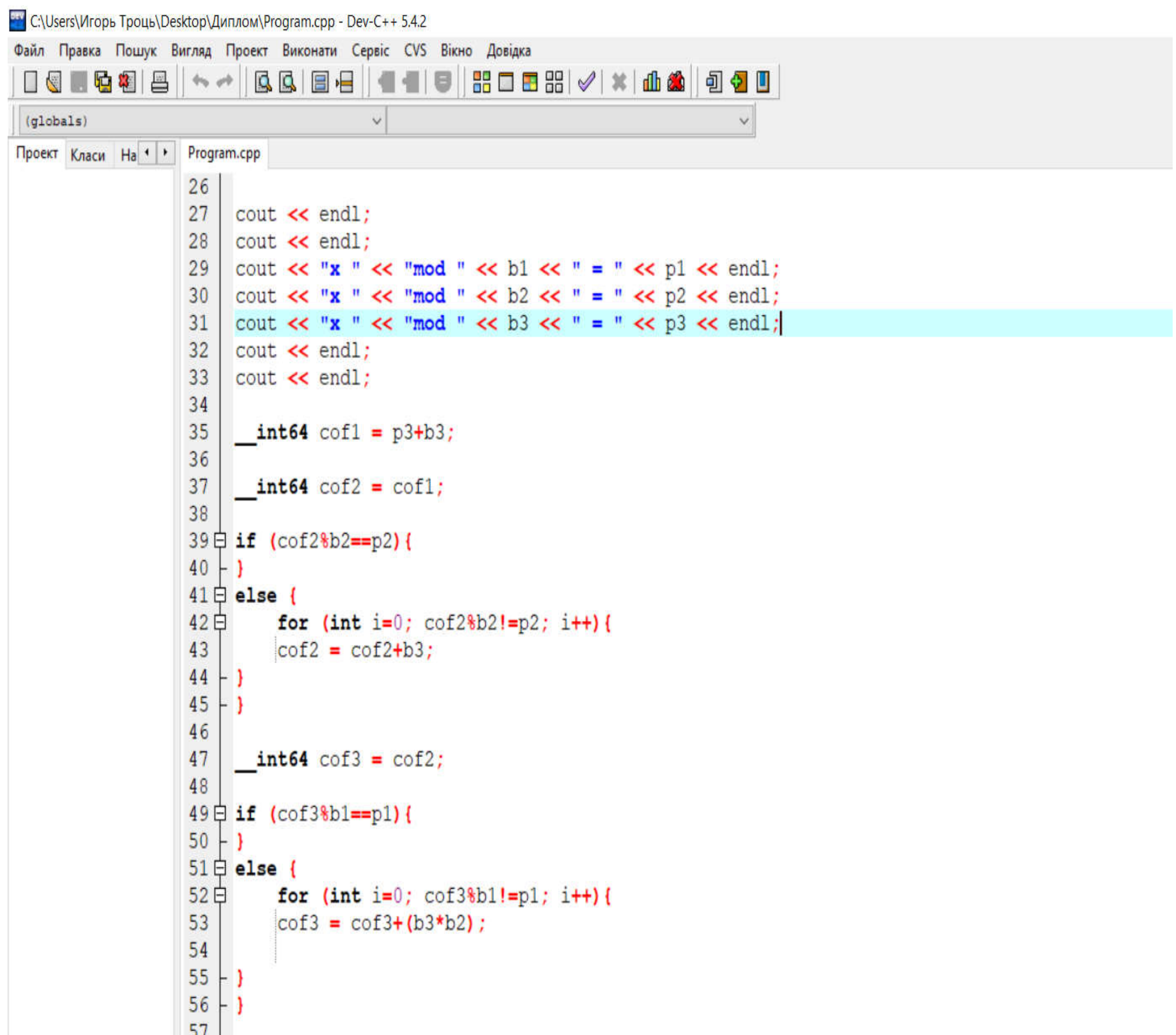

Рисунок 2.6 – Програмна реалізація алгоритму підбору третього значення невідомої для третьої нерівності мові С++

Після того як алгоритм знайшов третє значення невідомої потрібно провести перевірку чи справді остаточне значення невідомої задовільняє усі три нерівності, алгоритм перевірки наведено на рисунку 2.7.

```
}
else {
    for (int i=0; cof2%b2!=p2; i++){
    cof2 = cof2+b3;
}
}

__int64 cof3 = cof2;

if (cof3%b1==p1){
}
else {
    for (int i=0; cof3%b1!=p1; i++){
    cof3 = cof3+(b3*b2);

}
}

// перевірка всіх x

if (cof3%b3==p3 && cof3%b2==p2 && cof3%b1==p1){
cout << "x = " << cof3 << endl;
cout << "x = " << cof2 << endl;
cout << "x = " << cof1 << endl;
}
else{
cout << "x != " << cof3 << endl;
cout << "x != " << cof2 << endl;
cout << "x != " << cof1 << endl;
}
```

Рисунок 2.7 – Програмна реалізація алгоритму перевірки значення невідомої мові С++

Якщо значення невідомої підходить до усіх нерівностей то це значення вивиодиться на екран і може вважатись цілком правильним, але якщо алгоритму все ж таки не вдалось знайти правильне значення невідомої то алгоритм вивиде значення невідомої зі знаком оклику перед знаком рівності, алгоритм виведення значення невідомої наведено на рисунку 2.8.

```
cout << "x = " << cof3 << endl;
cout << "x = " << cof2 << endl;
cout << "x = " << cof1 << endl;
}
else{
cout << "x != " << cof3 << endl;
cout << "x != " << cof2 << endl;
cout << "x != " << cof1 << endl;
}
```

Рисунок 2.8 – Програмна реалізація алгоритму виведення значення невідомої на мові С++

Після виконання усіх етапів наведених раніше можна знайти значення невідомої за цілком новим алгоритмом який раніше не був представлений, а був розроблений самостійно.

2.4 Висновок до другого розділу

В даному розділі було проаналізувано декілька алгоритмів та принцип їх роботи, після чого стало зрозуміло, що створення власного алгоритму, який буде працювати за простішою схемою є найкращим рішенням. Під час програмної реалізації видно що кількість кроків до рішення рівняння є значно меншою ніж у аналогів і створений алгоритм є простішим для розуміння.

3 СТВОРЕННЯ ВЛАСНОГО АЛГОРИТМУ ОБЧИСЛЕННЯ КИТАЙСЬКОЇ ТЕОРЕМИ ПРО ЗАЛИШКИ

3.1 Структура та опис створенної програми

Після успішного створення алгоритму потрібно весь алгоритм перевести в програмний код, щоб провести тестування та перевірку правильності роботи алгоритму.

Для початку весь алгоритм переводиться в програмний код, середовище для програмування було взято C++ адже це середовище якнайкраще допоможе спростити опис алгоритмів та доможе оптимізувати роботу алгоритму.

Після запуску програми відкривається консольне вікно в якому буде запропоновано ввести значення усім змінним послідовно . Зразок стартового вікна після запуску програми зображено на рисунку 3.1.

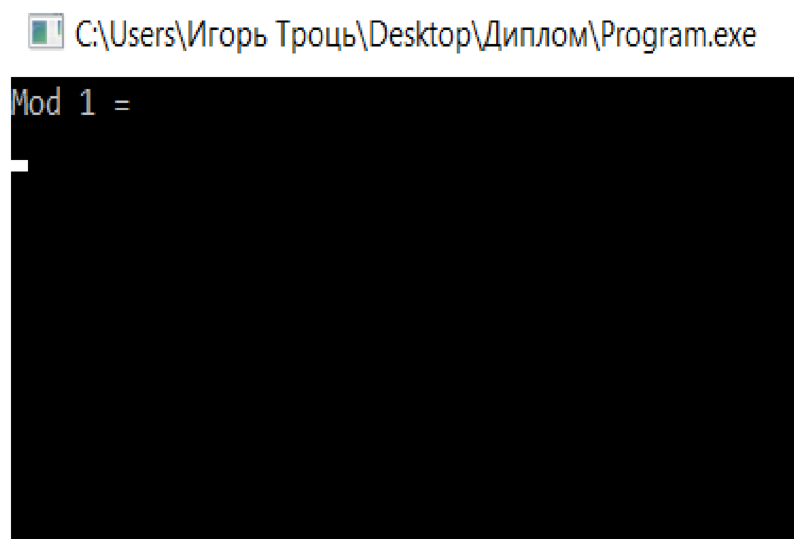

Рисунок 3.1 – Зразок стартового вікна програми

Після запуску програми потрбіно ввести значення першої змінної що відповідає за значення першого модульного числа. Приклад введеня значення першої змінної зображено на рисунку 3.2.

C:\Users\Игорь Троць\Desktop\Диплом\Program.exe

```
Mod 1 =
5
Сумма 1 =
```

Рисунок 3.2 – Приклад введеня першого модульного числа

Після введеня першого значення модульного числа потрбіно ввести значення суми для першого рівняння. Приклад введеня значення суму зображено на рисунку 3.3.

C:\Users\Игорь Троць\Desktop\Диплом\Program.exe

```
Mod 1 =
5
Сумма 1 =
2
Mod 2 =
```

Рисунок 3.3 – Приклад введеня суми для першого рівняння

Після введеня усіх значень першого рівння потрібно ввести значення модульної змінної для другого рівняння. Приклад введення значення модульної змінної для другого рівнянн зображено на рисунку 3.4.

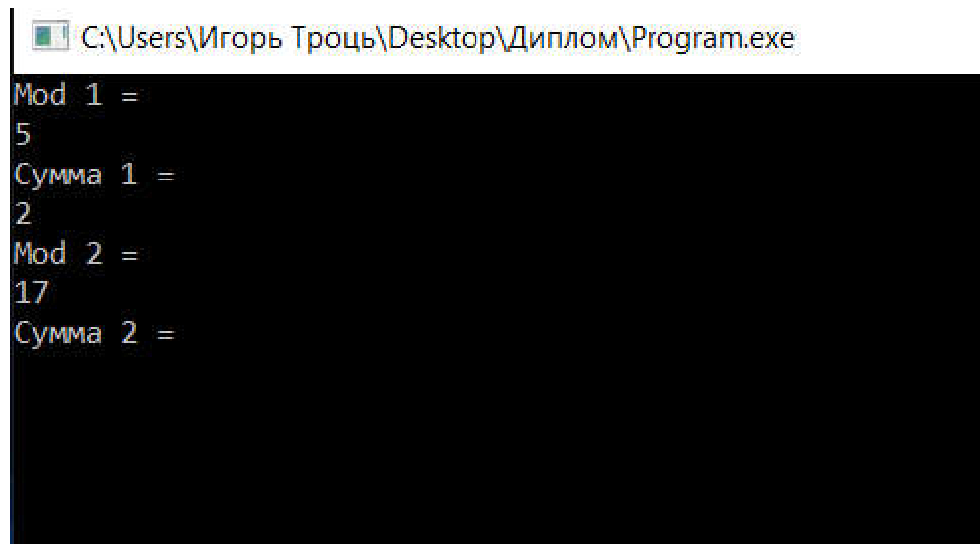

Рисунок 3.4 – Введення значення модульної змінної для другого рівняння

Після введення значення другог модульного числа потрібно вказати значення суми для другого рівняння. На рисунку 3.5 зображено введення значення суму для другого рівняння.

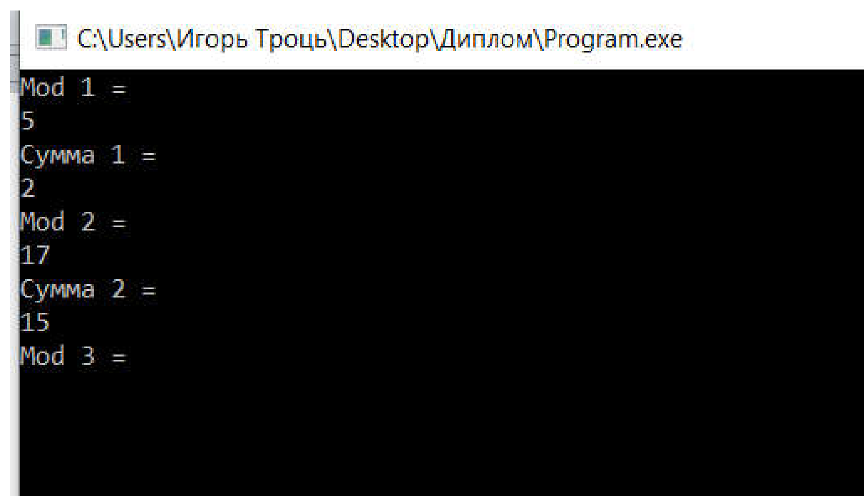

Рисунок 3.5 – Приклад введення значення суму для другого рівняння

Після заповнення значень для двох рівнянь залишається заповнити значення для третього рівняння. Потрібно ввести значення першого модульного числа для третього рівняння. На рисунку 3.6 зображенно приклад введення модульного числа для третього рівняння.

```
C:\Users\Игорь Троць\Desktop\Диплом\Program.exe
Mod 1 =
5
Сумма 1 =
2
Mod 2 =
17
Сумма 2 =
15
Mod 3 =
5
Сумма 3 =
```

Рисунок 3.6 – Приклад введення модульного числа для третього рівняння

Після заповення третього модульного числа для третього рівняння залишається заповнити останню змінну яка відповідає за суму третього рівняння [26]. Після надання значення цій змінній програма автоматично розпочне розрахунки для знаходження значення невідмої. Приклад введеня значення суму для третього рівняння зображено на рисунку 3.7.

```
C:\Users\╚уюЁN№ ╥ЁюЎN№\Desktop\─шяыюь\Program.exe
Mod 1 =
5
Сумма 1 =
2
Mod 2 =
17
Сумма 2 =
15
Mod 3 =
12
Сумма 3 =
5
```

Рисунок 3.7 – Приклад введеня значення суму для третього рівняння

Надавши значень усім змінним програма автоматичноививодить візуальне представлення усіх рівнянь, які були заповнені раніше. Як це виглядає зображено на рисунку 3.8.

```
x mod 5 = 2
x mod 17 = 15
x mod 12 = 5
```

Рисунок 3.8 – Візуальне представлення усіх рівнянь

Під час знаходження невідомої програма за алгоритмом підберає значення по черзі до кожного рівняння, таким чином програма видє значення невідомої три рази, де перше значення підходить до пергого рівняння, друге значення підходить до першого і другого рівняння, а третє значення невідомої уже задовільняє усі три рівняння і є правильною відповіддю. Приклад знаходження правильного значення невідомої в три етапи зображено на рисунку 3.9.

```
x mod 5 = 2
x mod 17 = 15
x mod 12 = 5

x = 797
x = 185
x = 17

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Рисунок 3.9 – знаходження правильного значення невідомої в три етапи

Дивлячись на попередній рисунок видно, що правильна відповідь до першого рівняння є 17, на другому етапі відповідь яка задовольнить і перше і друге рівння є 185, ну а кінцева правильна відповідь яка задовільняє усі три рівняння є 797.

3.2 Тестування роботи створенного програмного додатку

Більшість методів тестування в поточному використанні були розроблені до 1980 року. Тоді значущі програми становили менш 10 000 заяв. У комерційних середовищах, в яких розвинулася велика теорія тестування, програми були написані на мові COBOL, мовою, розробленому для розуміння непрограмістів. У цих умовах експерт по предмету може служити в якості тестувальника, читати програму, ідентифікувати всі змінні і їх найбільш цікаві

взаємодії і відстежувати основні шляхи через програму. Ефективно проведені вручну випробування, ретельно документовані, були найкращою практикою дня.

Сьогодні прийнято знаходити споживчі товари з декількома мільйонами рядків коду. Повторне використання компонентів (які, можливо, ніколи не були перевірені з урахуванням нового використання), дозволяє одночасно зібрати продукти на порядок більше. Навіть якщо вихідний код для компонентів був доступний (програмісти часто працюють з бібліотеками, які роблять видимими тільки інтерфейс прикладного програмування, а не основне джерело), жодна людина не може впоратися з внутрішніми деталями додатка цієї складності.

Ефективність тестування дещо покращилася за останні тридцять років (наприклад, сервіс QA ПО в Одесі), але практично не залежить від ефективності програмування. Найпоширеніший метод автоматизації, автоматизація регресійного тестування, автоматизує тільки виконання тесту і просте порівняння результатів - проектування, документація і обслуговування цих тестів - це дорогі людські завдання. Системи управління тестовими прикладами як і раніше вимагають покрокової тестової документації. Деякі тестувальники життєво важливих додатків повідомляють, що вони витрачають до 90% часу тестування на пов'язані з документацією завдання, тільки 10% на виконання тесту і аналіз результатів.

Спрощені показники охоплення як і раніше широко поширені. Їх можна легко вирахувати, але вони є поганими індикаторами. Розглянемо проблему забезпечення наших кордонів - запобігання ввезення бомб, небезпечних хімічних або біологічних агентів або наркотиків. Навіть якби ми могли забезпечити кордон, щоб кожен вхід (суду, автомобілі, літаки, пішоходи) приходив на укомплектований контрольно-пропускний пункт, скільки ми могли б шукати? Досягнення повного охоплення заяв - це питання про те, щоб просити кожного в'їжджає автомобіля або людини для ідентифікації. Ви знайдете деякі проблеми таким чином, але це не те ж саме, що шукати

корабель. Ми не можемо все ретельно вивчити, ми не можемо повністю перевірити всі, але ми повинні ретельно перевірити деякі речі. Вибір правильних вимагає оцінки людини і оцінки ризику.

Розпачати тестування подрібно з перевірки правильності написання програмного коду, а щоб це зробити потрібно відкомпілювати створенний код. Адже якщо код не пройде компіляцію значить була допущена помилка, або ряд помилок які не дадуть можливості запустити створенний алгоритм.

Під час компіляції не було виявлено жодних помилок у написанні програмного коду, що свідчить про те що алгоритм можна запустити для подальшої перевірки правильності його роботи. На рисунку 3.10 зображено результат компіляції коду створенної програми.

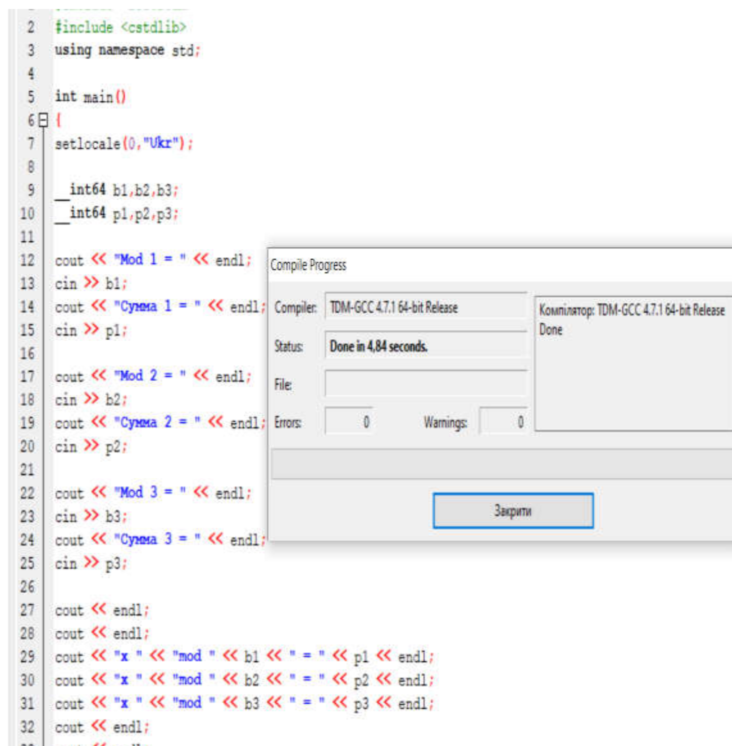

Рисунок 3.10 – Результат компіляції коду

Після компіляції коду продовжувати тестування потрібно уже з різними значеннями рівнянь. Адже чим більше рівнянь буде перевірено в алгоритмі тим з більшим відсотком можна буде сказати що алгоритм працює правильно і не дасть помилки у вирішенні нерівності. Для початку тестування правильної

роботи алгоритму потрібно запустити програму та ввести значення усіх змінних як це показано на рисунку 3.11. На рисунку видно що після того як задати значення усім змінним то алгоритм миттєво переходить до обчилень та знаходженя правильного значення невідомої [29].

Таким чином після проходження декілької етапів обчисленя значення невідомої алгоритм знаходить правильне значення невідомої яке підходить то усіх нерівностей та виводить правильну відповідь на екран. Для того щоб перевірити як можна краще роботу алгоритму кількість перевірок має бути як можна більшою.

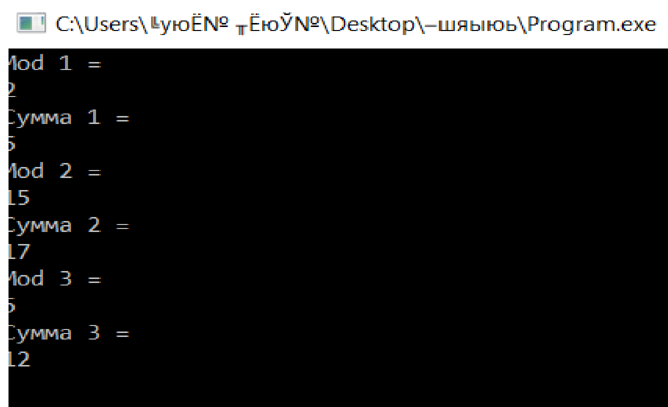

Рисунок 3.11 – Присвоєння значень змінним

На наступному рисунку 3.12 зображено результат роботи алгоритму при першій перевірці, виведене число є правильним значенням змінної. Зображена відповідь буде підходити до усіх нерівностей які були задані на попередньому етапі.

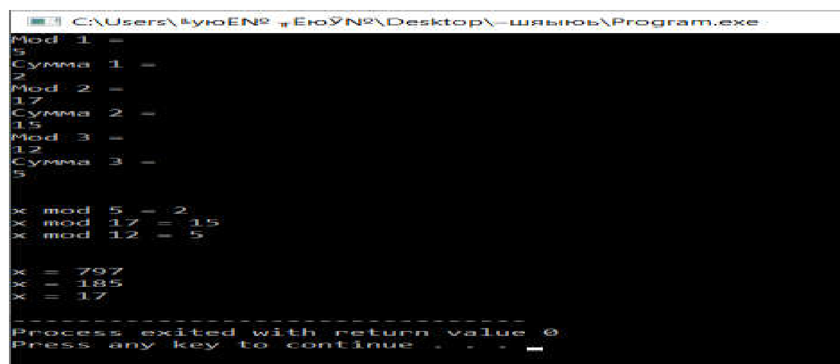

Рисунок 3.12 – Результат роботи програми при першій перевірці

Для більш поглибленного тесутвання потрібно взяти ще дкілька різних значень змінних та перевірити правильність роботи алгоритму. На рисунку 3.13 зображено результат другої перевірки роботи алгоритму.

```
C:\Users\╙уюЁN╫ ╤ЁюЎN╫\Desktop\–шяыюь\Program.exe
Mod 1 =
2
Сумма 1 =
1
Mod 2 =
3
Сумма 2 =
2
Mod 3 =
7
Сумма 3 =
6


x mod 2 = 1
x mod 3 = 2
x mod 7 = 6


x = 41
x = 20
x = 13

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Рисунок 3.13 – Результат другої превірки програми

Можна зробити безліч перевірок щоб впевнитися в правильності роботи алгоритму але для більш менш точного результату вистачить три перевірки. На рисунку 3.14 зображено результат третьої перевірки роботи алгоритму.

```
C:\Users\╙уюЁN╫ ╤ЁюЎN╫\Desktop\–шяыюь\Program.exe
Mod 1 =
3
Сумма 1 =
1
Mod 2 =
11
Сумма 2 =
2
Mod 3 =
17
Сумма 3 =
5


x mod 3 = 1
x mod 11 = 2
x mod 17 = 5


x = 277
x = 90
x = 22

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Рисунок 3.14 – Результат третьої перевірки роботи програми

Спираючись на результати тестування роботи алгоритму можна сказати що створений алгоритм працює правильно та не викликає підозр на правильність обрахунків, адже в усіх трьох перевірках було знайдено не тільки правильне значення невідомої, але ще й найменше можливе значення цієї невідомої.

3.3 Порівняння з існуючими аналогами

Виконання тестів необхідно, але не менш важливі і супроводжуючі дії - планування і документування процесу. В обов'язки тестувальників входить розробка тестових сценаріїв, а також підготовка тестування і оцінка його результатів. Становлення ідеї фундаментального тестового процесу на всіх рівнях тестування зайняло роки. В рамках цього процесу можна виділити ключові кроки:

– планування і управління;
– аналіз і проектування;
– впровадження та реалізації;
– оцінка критеріїв виходу і написання звітів;
– дії по завершенню тестування.

Тут дії описані в логічній послідовності, але в умовах реального проекту вони можуть накладатися, відбуватися одночасно або навіть повторюватися. Зазвичай, відбувається адаптація цих кроків під потреби конкретної системи або проекту.

Планування тестування включає дії, спрямовані на визначення основних цілей тестування і завдань, виконання яких необхідне для досягнення цих цілей.

У процесі планування ми переконуємося в тому, що ми правильно зрозуміли цілі і побажання замовника і об'єктивно оцінили рівень ризику для проекту, після чого ставимо мети і завдання для, власне, тестування.

Для більш ясного опису цілей і завдань тестування складаються такі документи як тест-політика, тест-стратегія і тест-план.

Тест-політика – високорівнева документ, що описує принципи, підходи і основні цілі компанії в сфері тестування.

Тест-стратегія – високорівнева документ, що містить опис рівнів тестування і підходів до тестування в межах цих рівнів. Діє на рівні компанії або програми одного або більше проектів.

Тест-план – документ, що описує засоби, підходи, графік робіт і ресурси, необхідні для проведення тестування. Крім іншого, визначає інструменти тестування, функціональність, яку потрібно протестувати, розподіл ролей в команді, тестове оточення, використовувані техніки тест-дизайну, критерії початку і закінчення тестування і ризики. Тобто, це докладний опис всього процесу тестування.

У будь-якій діяльності, управління не закінчується плануванням. Нам потрібно контролювати і вимірювати прогрес. Саме тому управління тестуванням - безперервний процес.

Управління тестуванням – зіставлення поточної ситуації в процесі тестування з планом і складання звітності.

У свою чергу, дані, отримані в ході контролю над процесом, враховуються при плануванні подальших дій.

Аналіз і проектування тестів – це процес написання тестових сценаріїв і умов на основі загальних цілей тестування.

У процесі аналізу і проектування ми розробляємо тестові сценарії на підставі загальних цілей тестування, визначених під час планування.

Тестовий сценарій – документ, що визначає встановлену послідовність дій при виконанні тестування.

Під час виконання тестування відбувається написання тест-кейсів, на основі написаних раніше тестових сценаріїв, збирається необхідна для проведення тестів інформація, готується тестове оточення і запускаються тести.

Тест-кейс – документ, що містить набір вхідних значень, перед- і постусловієм, а також очікуваний результат проведення тесту, розроблений для перевірки відповідності певної функціональності системи заданим для цієї функціональності вимогам.

Тестове оточення – апаратне і програмне забезпечення та інші засоби, необхідні для виконання тестів.

Критерії виходу визначають, коли можна завершувати тестування. Вони необхідні для кожного рівня тестування, оскільки нам необхідно знати, чи достатньо було проведено тестів:

– перевірити, чи було проведено достатню кількість тестів, чи досягнута потрібна ступінь забезпечення якості системи.

– переконається в тому, що немає необхідності проводити додаткові тести.

Якщо все ж таки така необхідність є, можливо, буде потрібно змінити встановлений критерій виходу.

Після закінчення тестування відбувається написання звіту, який буде доступний всім зацікавленим сторонам. Адже не тільки тестувальники повинні знати результати виконання тестів, – ця інформація може бути необхідна багатьом учасникам процесу створення ПО.

При завершенні тестування ми збираємо, систематизуємо і аналізуємо інформацію про його результати. Вона може стати в нагоді пізніше - під час випуску готового продукту. Можуть бути й інші причини для згортання тестування, наприклад, дострокове закриття проекту або завершення певного етапу розробки.

Перебір від довільного числа – стартова точка дослідження. Ідея алгоритму дуже проста: женемо змінну циклу від першого числа до 1. Якщо

обидва числа діляться на змінну циклу без залишку, значить змінна циклу і дорівнює найменшому спільному дільрнику та цикл можна завершити достроково. Якщо цикл пройшов до кінця, значить для цих чисел найменшому спільному дільрнику дорівнює 1.

Тестування програмного забезпечення - креативна і інтелектуальна робота. Розробка правильних і ефективних тестів - досить непросте заняття. Принципи тестування, представлені нижче, були розроблені в останні 40 років і є загальним керівництвом для тестування в цілому.

Тестування може показати наявність дефектів в програмі, але не довести їх відсутність. Тим не менш, важливо складати тест-кейси, які будуть знаходити якомога більше багів. Таким чином, при належному тестовому покритті, тестування дозволяє знизити ймовірність наявності дефектів в програмному забезпеченні. У той же час, навіть якщо дефекти не були знайдені в процесі тестування, не можна стверджувати, що їх немає.

Неможливо провести вичерпне тестування, яке б покривало все комбінації призначеного для користувача введення і станів системи, за виключенням зовсім вже примітивних випадків. Замість цього необхідно використовувати аналіз ризиків і розстановку пріоритетів, що дозволить більш ефективно розподіляти зусилля щодо забезпечення якості ПЗ.

Тестування повинне починатися якомога раніше в життєвому циклі розробки програмного забезпечення, і його зусилля повинні бути сконцентровані на певних цілях.

Різні модулі системи можуть містити різну кількість дефектів - тобто, щільність скупчення дефектів в різних елементах програми може відрізнятися. Зусилля по тестуванню повинні розподілятися пропорційно фактичній щільності дефектів. В основному, більшу частину критичних дефектів знаходять в обмеженій кількості модулів. Це прояв принципу Парето: 80% проблем містяться в 20% модулів.

Проганяючи одні і ті ж тести знову і знову, Ви зіткнетеся з тим, що вони знаходять все менше нових помилок. Оскільки система еволюціонує, багато з

раніше знайдених дефектів виправляють і старі тест-кейси більше не спрацьовують.

Щоб подолати цей парадокс, необхідно періодично вносити зміни в використовувані набори тестів, рецензувати і коригувати їх з тим, щоб вони відповідали новому станом системи і дозволяли знаходити якомога більшу кількість дефектів.

Вибір методології, техніки і типу тестування буде прямо залежати від природи самої програми. Наприклад, програмне забезпечення для медичних потреб вимагає набагато більш суворою і ретельної перевірки, ніж, скажімо, комп'ютерна гра. З тих же міркувань, сайт з великою відвідуваністю повинен пройти через серйозне тестування продуктивності, щоб показати можливість роботи в умовах високого навантаження.

Той факт, що тестування не виявило дефектів, ще не означає, що програма готова до релізу. Знаходження і виправлення дефектів будуть не важливі, якщо система виявиться незручною у використанні, і не буде задовольняти очікуванням і потребам користувача.

Очевидний недолік - несиметричність щодо аргументів. Очевидно, що найменшому спільному дільрнику менше або дорівнює меншому з двох чисел. Тому виконувати цикл від більшого числа не має сенсу. Програмну реалізацію циклу зображено на рисунку 3.15.

```
long gcd01(long a, long b) {

    long nod = 1L;
    for (long i = a; i > 0; i--) {
        if (a % i == 0 && b % i == 0) {
            nod = i;
            break;
        }
    }
    return nod;
}
```

Рисунок 3.15 – Програмна реалізація циклу перебору від довільного числа

Просто додаємо найпростішу функцію для обчислення мінімального числа для пари чисел і инициализируем змінну циклу меншим з двох чисел. У половині випадків така оптимізація працювати не буде, коли перший аргумент і так менше другого, зате в іншій половині випадків виграш за часом може бути досить значним. Програмну реалізацію циклу зображено на рисунку 3.16.

```
long min(long a, long b) {
    return a > b ? b : a;
}

long gcd02(long a, long b) {

    long nod = 1L;
    for (long i = min(a, b); i > 0; i--) {
        if (a % i == 0 && b % i == 0) {
            nod = i;
            break;
        }
    }
    return nod;
}
```

Рисунок 3.16 – Програмна реалізація циклу перебору від мінімального числа

Перший if відловлює ситуацію, коли обидва числа діляться без остачі на змінну циклу і, отже, змінна циклу є загальним простим множником для обох чисел і враховується для обчислення найменшому спільному дільрнику. Решта два if відловлюють випадки, коли тільки одне з чисел ділиться на змінну циклу; ці множники в НОД не входять.

Цикл for повинен перебирати тільки прості числа. Але знаходження простих чисел є самостійною обчислювальної завданням, яку тут вирішувати не хотілося б [28].

Можна, звичайно, використовувати таблицю простих чисел, наприклад, в межах першої тисячі, а для великих чисел, якщо буде потреба, обчислювати прості числа перевіркою на простоту, але я не став забиратися в нетрі

факторизации натуральних чисел, а просто закрив очі на свідомо холості проходи циклу for, коли змінна циклу не є простим числом, оскільки після знаходження кожного з множників хоча б для одного з чисел цикл стартує заново. Програмну реалізацію алгоритму зображено на рисунку 3.17.

```
long gcd03(long a, long b) {

    long nod = 1L;

    if (a > b) {
        long tmp = a;
        a = b;
        b = tmp;
    }

    while (a > 1L && b > 1L) {
        for (long i = 2; i <= a; i++) {
            if (a % i == 0 && b % i == 0) {
                nod *= i;
                a /= i;
                b /= i;
                break;
            }
            if (a % i == 0) {
                a /= i;
                break;
            }
            if (b % i == 0) {
                b /= i;
                break;
            }
        }
    }
    return nod;
}
```

Рисунок 3.17 – Програмна реалізація алгоритму з розкладанням на дільники

У найпростішому випадку алгоритм Евкліда застосовується до парі позитивних цілих чисел і формує нову пару, яка складається з меншої кількості і різниці між більшим і меншим числом. Процес повторюється, поки числа не стануть рівними. Знайдене число і є найбільший спільний дільник вихідної пари. Програмну реалізацію рекурсивного циклу алгоритму Евкліда зображено на рисунку 3.18.

```
long gcd04(long a, long b) {

    if (a == b) {
        return a;
    }
    if (a > b) {
        long tmp = a;
        a = b;
        b = tmp;
    }
    return gcd04(a, b - a);
}
```

Рисунок 3.18 – Програмна реалізація рекурсивного циклу алгоритму Евкліда

Вважається, що рекурсивні алгоритми менш ефективні, ніж ітераційні, за рахунок накладних витрат на виклик функції. Для перевірки робимо і ітераційний варіант. Програмну реалізацію ітераційного циклу алгоритму Евкліда зображено на рисунку 3.19.

```
long gcd05(long a, long b) {

    while (a != b) {
        if (a > b) {
            long tmp = a;
            a = b;
            b = tmp;
        }
        b = b - a;
    }
    return a;
}
```

Рисунок 3.19 – Програмна реалізація ітераційного циклу алгоритму Евкліда

Оскільки відмінною рисою бінарного алгоритму є можливість використання бітових зрушень замість повільних операцій ділення та множення, гріх такою можливістю не скористатися. Програмна реалізація алгоритму з використанням бітових зрушень зображена на рисунку 3.20.

```
long gcd07(long a, long b) {
    long nod = 1L;
    long tmp;
    if (a == 0L)
        return b;
    if (b == 0L)
        return a;
    if (a == b)
        return a;
    if (a == 1L || b == 1L)
        return 1L;
    while (a != 0 && b != 0) {
        if (a % 2L == 0L && b % 2L == 0L) {
            nod *= 2L;
            a /= 2L;
            b /= 2L;
            continue;
        }
        if (a % 2L == 0L && b % 2L != 0L) {
            a /= 2L;
            continue;
        }
        if (a % 2L != 0L && b % 2L == 0L) {
            b /= 2L;
            continue;
        }
        if (a > b) {
            tmp = a;
            a = b;
            b = tmp;
        }
```

Рисунок 3.20 – Програмна реалізація алгоритму з використанням бітових зрушень

Ця версія функції за логікою повністю збігається з попередньою, але операції ділення і множення на 2 замінені арифметичними зрушеннями, а перевірка на парність – на перевірку молодшого біта числа. Програмна реалізація алгоритму з використанням арифметичних зрушень зображена на рисунку 3.21.

```
long gcd08(long a, long b) {
    long nod = 1L;
    long tmp;
    if (a == 0L)
        return b;
    if (b == 0L)
        return a;
    if (a == b)
        return a;
    if (a == 1L || b == 1L)
        return 1L;
    while (a != 0 && b != 0) {
        if (((a & 1L) | (b & 1L)) == 0L) {
            nod <<= 1L;
            a >>= 1L;
            b >>= 1L;
            continue;
        }
        if (((a & 1L) == 0L) && (b & 1L)) {
            a >>= 1L;
            continue;
        }
        if ((a & 1L) && ((b & 1L) == 0L)) {
            b >>= 1L;
            continue;
        }
        if (a > b) {
            tmp = a;
            a = b;
            b = tmp;
        }
        tmp = a;
```

Рисунок 3.21 – Програмна реалізація алгоритму з використанням бітових зрушень

Як показали результати тестів, час роботи функцій сильно залежить від вхідних даних. Тому було вирішено генерувати випадкові пари чисел. Однак з'ясувалося, що зовсім випадкові пари чисел занадто часто є взаємно простими [27]. Тому я вирішив до пари випадкових чисел додавати випадковий множник. Вийшло, що навіть якщо спочатку в парі числа були взаємно прості, то алгоритм пошуку НСД повинен буде обчислити саме цей доданий множник.

В іншому алгоритм тестування простий: для кожного набору випадкових даних береться одна з тестованих функцій і проганяється декілька раз. Час роботи функції записується в елемент масиву, що відповідає цій функції. Після

того, як всі тести пройдені, результати з масиву діляться на кількість наборів даних – отримуємо середній час для набору.

Як і очікувалося, перший алгоритм катастрофічно неефективний. Другий алгоритм працює у два рази бистріше за перший. Третій алгоритм несподівано показав дуже гідний результат, в 50 разів швидше другого алгоритму. Четвертий і п'ятий варіанти алгоритму Евкліда, рекурсивна версія, як не дивно, обігнала ітераційну. У порівнянні з третім варіантом час покращився майже в двічі. Бінарний алгоритм Евкліда показав найкращі результати. З трьох варіантів реалізації рекурсивна версія сама некваплива.

Найкращий результат у розробленої версії з використанням власного алгоритму вирішення. Таким чином створений алгоритм працює більш ніж в 50 разів швидше, ніж існуючі алгоритми.

3.4 Висновок до третього розділу

В даному розділі було проведено тестування програмного додатку, була наведена поетапна інструкція користування програмою. Результати тестування роботи алгоритму показали що створений алгоритм працює правильно, адже в усіх перевірках було знайдено не тільки правильне значення невідомої, але й найменше можливе значення цієї невідомої та в порівнянні з існуючими аналогами алгоритм має найменший час виконання та найменшу кількість кроків до знаходження відповіді.

ВИСНОВКИ

1. В першому розділі було поставлене завдання про створення власного алгоритму вирішення китайської теореми про залишки та реалізація цього алгоритму. Під час створення алгоритму потрібно буде знайти не тільки новий метод пошуку невідомої, але й зробити так щоб знайдене значення задовольняло усі рівняння та час виконання створеного алгоритму був менший ніж у алгоритмів що уже існують.

2. В другому розділі було проаналізовано декілька алгоритмів та принцип їх роботи, після чого стало зрозуміло, що створення власного алгоритму, який буде працювати за простішою схемою є найкращим рішенням. Під час програмної реалізації видно що кількість кроків до рішення рівняння є значно меншою ніж у аналогів і створений алгоритм є простішим для розуміння.

3. В третьому розділі було проведено тестування програмного додатку, була наведена поетапна інструкція користування програмою. Результати тестування роботи алгоритму показали що створений алгоритм працює правильно, адже в усіх перевірках було знайдено не тільки правильне значення невідомої, але й найменше можливе значення цієї невідомої.

4. Отже магістерську роботу можна ввжати виконаною, адже мета роботи була досягнена та усі поставлені завдання були виконані.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп'ютерна інженерія» / І.В. Гураль, Л.О. Дубчак / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 33 с.

2. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп'ютерна інженерія» / І.В. Гураль, Л.О. Дубчак / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2018. 42 с.

3. Троць І.І., Касянчук М.М. Алгоритм реалізації китайської теореми про залишки для асиметричних криптосистем: зб. матеріалів доп. учасн. Наук. − практ. конф. Тернопіль : Тернопіль, 2019. С.48.

4. Новосад Х.В., Троць І.І. База даних модуля агрегаціїновин на сайті: зб. матеріалів доп. учасн. II Наук. − практ. конф. Тернопіль: Тернопіль, 2019. С. 22 – 42.

5. Євчук О. В. Цифрова обробка сигналів: конспект лекцій, ІваноФранківськ : ІФНТУНГ, 2010. С.135.

6. Сергиенко А.Б. Цифровая обработка сигналов, СПб.: Питер, 2003, 604 с.

7. Бабак В.П., Хандецький В.С., Шрюфер Е.К. Обробка сигналів: Підручник, Либідь, 1996. С.392.

8. Айфичер Э., Джефрис Б. Цифровая обработка сигналов: практический поход 2-е.,М:Вильямс, 2004. С.992.

9. Сойфер В.А., Сергеев В.В. и др. Теоретические основы цифровой обработки изображений, Самара, 2000. С.256.

10. Марпл С.Л. Цифровой спектральный анализ и его приложения, М:Мир, 1990. 5с.

11. Аліасинг: веб-сайт. URL: http://uk.wikipedia.org/wiki/Аліасинг (дата звернення: 28.04.2019).

12. Спектральный анализ на ограниченном интервале времени. Оконные функции: веб-сайт. URL: http://www.dsplib.ru/content/win/win.html (дата звернення: 11.11.2019).

13. G.N. Shinde, H.S. Fadewar. Faster RSA Algorithm for Decryption Using Chinese Remainder Theorem: веб-сайт. URL: http://www.techscience.com/doi/10.3970/icces.2008.005.255.pdf (дата звернення: 05.05.2019).

14. Joye, M. and F. Olivier. Side-channel analysis, Encyclopedia of Cryptography and Security, 2005. P. 571–576.

15. Peter Gutmann, David Naccache, Charles C. Palmer. Side Channel attacks on Cryptographic software, Copublished by the IEEE computer and reliability societies 1540-7993, November 2009. P. 131–133.

16. YongBin Zhou, DengGuo Feng. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing, State Key Laboratory of Information Security, Institute of Software, Chinese Academy of Sciences, Beijing, China, 1997. P. 47–51.

17. Kocher, Paul C. Cryptanalysis of Diffie-Hellman, RSA, DSS, and Other Systems Using Timing Attacks, Advances in Cryptology, CRYPTO ’95: 15th Annual International Cryptology Conference, Springer-Verlag, 1995, P. 27–31.

18. Dhem, J.F., F. Koeune, P.A. Leroux, P. Mestr´ e, J.J. Quisquater, and J.L. Willems. A practical implementation of the timing attack, Smart Card Research and Applications, Springer, 2000. P. 167–182.

19. Schindler, Werner. A Timing Attack against RSA with the Chinese Remainder Theorem, Cryptographic Hardware and Embedded Systems CHES 2000, volume 1965 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2000. P. 109–124.

20. Marc Joye, Michael Tunstall. Fault Analysis in Cryptography, SpringerVerlag Berlin Heidelberg, 2012. P. 81–85.

21. Stefan Mangard, Elisabeth Oswald, Thomas Popp. Power Analysis Attacks: Revealing the Secrets of Smart Cards, Springer Science+Business Media, LLC, 2007. P. 89-91.

22. Семенов Ю. А. Телекоммуникационные технологии. Интернет-университет информационных: веб-сайт. URL: http://book.itep.ru(дата звернення: 11.11.2019).

23. Олифер В. Г. Компьютерные сети. Принципы, технологии, протоколы: учебник для вузов: 3-е изд. / В. Г Олифер., Н. А. Олифер // СПб.: Питер, 2006. С.29-31.

24. Зайченко Ю. П. Анализ и оптимизация характеристик сетей MPLS по заданным показателям качества / Ю. П. Зайченко, Ахмед А. М. Шарадка // Вісник національного технічного університету України КПІ сер. Інформатика управління та обчислювальна техніка, 2007. С.113–123.

25. Будылдина Н. В. Разработка программного обеспечения для оптимизации мультисервисных сетей / Н. В. Будылдина., П. А. Коновалов // Открытое образование, червень 2006. С.58.

26. Зайцев Д. А. Моделирование телекоммуникационных сетей в системе NS. / Д. А. Зайцев, Т. Н. Шинкарчук // Наукові праці ОНАЗ ім. О. С. Попова, 2006. 11-15 с.

27. Кучерявый Е.А. Управление трафиком и качество обслуживания в сети Интернет / Е.А. Кучерявый // – М.: Наука и Техника, 2007. 336 с.

28. Panwar Li. Y. S. On the Performance of MPLS TE Queues for QoS Routing // Panwar Li. Y. Liu C.J. Simulation series. – Vol. 36; part 3, 2007. P. 170 – 174.

29. Аткинсон Л. Mysql. Библиотека профессионала – М Энергоатомиздат, 2002. С.496.

30. Дейт К. Руководство по реляционной СУБД DB2 – М.: Финансы и статистика, 1988. С.320.

31. Мейер М.Теория реляционных баз данных – М.: Мир, 1987, С.608.

32. Семантична модель: База даних: веб-сайт. URL: http://citforum.ru/database/advanced_intro/27.shtml(дата звернення: 09.12.2019).

33. Риккарди Г. Системы баз данных. Теория и практика использования в Internet – М.:Вильямс, 2001. С.240.

34. ДейтК. Дж. Введение в системы баз даннях СПб: Изд-во "Пітер", 2005. С.1315.

35. Роберт І.В. Сучасні інформаційні технології в освіті: дидактичні проблеми, перспективи використання М.: Школа-Пресс, 1994. С.205.

36. Розділ 2.Основи UML – діаграм: веб-сайт. URL: https://docs.kde.org/trunk4/uk/kdesdk/umbrello/uml-basics.html(дата звернення: 02.12.2019).

37. Мюллер Р.Дж. Базы данных и UML. Проектирование / Перевод. с англ. Е. Молодцова. – М.: Издательство “Лори”, 2002. С.432.

38. Острей О.Р. Діаграми класів UML як засіб моделювання інформаційної системи моніторингу освіти / М.: 2008. С.85-89.

39. Електронна енциклопедія Вікіпедія: JSON: веб-сайт. URL: https://uk.wikipedia.org/wiki/JSON(дата звернення: 05.11.2019).

40. Компонентне або модульне тестування: веб-сайт. URL: http://www.protesting.ru/testing/component.html(дата звернення: 02.12.2019).

41. Електронна енциклопедія Вікіпедія: Модульне : веб-сайт. URL: https://uk.wikipedia.org/wiki/Модульне(дата звернення: 19.10.2019).

42. Електронна енциклопедія Вікіпедія: Список кодів стану: веб-сайт. URL: https://uk.wikipedia.org/wiki/список(дата звернення: 02.12.2019).

43. Шапошников І. Web-сайт своїми руками./ І. Шапошников – СПб: Изд-во "Пітер", 2002. С.390.

44. Гаевский А. Ю. Самоучитель по созданию Web-страниц: HTML, JavaScript, Dynamic HTML / А. Ю. Гаевский, В. А.Романовский. К.: А.С.К., 2002. С.472.

45. Коннолли Т.А. Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т.А. Коннолли, К.М Бегг. М. : Вильямс, 2003. 1436 с.

46. Камер Д. Компьютерные сети и Internet М. : Вильямс, 2002. С.640.

47. Андон Ф.И. Информационные системы / Ф.И. Андон, В.П. Резниченко, У.У. Яшунин – К., 2001. С.396.

48. Дронов В. PHP 5/6, MySQL 5/6 и Dreamweaver CS4. Разработка интерактивных Web–сайтов БХВ-Петербург Москва, 2009. С.544.

49. Поломошнов О.Б. Быстро и легко создаем, программируем и раскручиваем Web-сайт. М.: Эксмо, 2011. С.352.

50. Петюшкин А.В. HTML экспресс − курс . М.: СПб: БХВ. Петербург, 2004. С.250.