МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

На правах рукопису

Якименко Ігор Зіновійович

УДК 681.3.06

Методи та алгоритми опрацювання інформаційних потоків в комп'ютерних мережах за умови застосування еліптичних кривих

Спеціальність 05.13.05 – Комп'ютерні системи та компоненти

Дисертація на здобуття наукового ступеня
кандидата технічних наук

Науковий керівник
Николайчук Ярослав Миколайович
доктор технічних наук, професор

Тернопіль – 2011

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АЦК – аналого-цифровий кодер;

АЦП – аналого-цифровий перетворювач;

БМРД – багаторівнева модель руху даних;

БОС – багатопроцесорні обчислювальні системи;

БРДММРД–багаторівнева диференційована матрична модель руху даних;

Д – дані;

ДІ – джерело інформації;

ДЕРД –диференціація епюр руху даних;

ДФСЗК – досконала форма систем залишкових класів;

ЕК – еліптична крива;

ЕЦП – електронно-цифровий підпис;

КМ – комп'ютерні мережі;

КС – комп'ютерні системи;

КТЗ – Китайська теорема про залишки;

ММРД – матричні моделі руху даних;

МРД – модель руху даних;

НСД — найбільший спільний дільник;

О – оператори;

ОУ – об'єкти управління;

ПІ – приймач інформації;

ПКД – пам'ять колективного доступу;

ПЛІС – програмовані логічні інтегральні схеми;

ПОКС – проблемно-орієнтовані комп'ютерні системи;

Р – процесори;

РКС – розподілена комп'ютеризована система;

СД – структуризовані дані;

СЗК – система залишкових класів;

СКС – спеціалізовані комп'ютерні системи;

СО – системні об'єкти;

СОІ – середовище передавання інформації;

СПІ – середовище передавання інформації;

СПД – система передавання даних;

ФІП – формувачі інформаційних потоків;

RSA — алгоритм Рівеста-Шаміра-Адлемана;

MIPS (Million Instruction Per Second) – міліон повідомлень в секунду.

ВСТУП

Актуальність теми. Створення розвиненого і захищеного інформаційного суспільства є невід'ємною умовою розвитку держави. Глибока технологічна реформа, що проходить сьогодні в Україні, спрямована на впровадження низки важливих автоматизованих інформаційних систем і мереж зв'язку, телекомунікаційних систем, систем прийняття рішень та ін.

Важливим аспектом розвитку досліджень в цій галузі є врахування принципово нових відмінностей між комп'ютеризованими системами та комп'ютерними мережами. Виходячи з визначення комп'ютерної системи як взаємодії сукупності системних об'єктів, вона, як показано в роботах американського вченого Дж. Мартіна та українських вчених О.В.Палагіна, В. А. Стеклова, Я.М. Николайчука і т.д., включає в себе п'ять системних об'єктів: Р – процесори, Д – дані, СПД – систему передавання даних, О – оператори і ОУ – об'єкти управління. При цьому комунікаційним компонентом комп'ютеризованих систем є комп'ютерні мережі, які в склад своїх компонентів включають процесори, дані і СПД. Причому характеристики комп'ютерних мереж та функціональні характеристики операторів проектуються, виходячи з проблемної орієнтації та спеціалізації реальних об'єктів управління, для якої проектується і застосовується розподілення комп'ютеризованих систем.

При проектуванні та розвитку сучасних комп'ютерних мереж актуальною задачею є дотримання умов високоефективного захисту інформаційних потоків від несанкціонованого доступу.

Вагомий внесок у дослідження проблеми захисту інформаційних потоків комп'ютерних систем та мереж вклали такі вітчизняні та зарубіжні вчені: Задірака В.К., Широчин В.П., Горбенко І.Д., Долгов В.І., Бессалов А.В., Воєводін В.В., Головко В.А., Мельникова О.А., Качко О.Г., Р.Шуф, В. Міллер, Н. Кобліц, А. Лєнстра, Т.Ланге К. Фокс, В.Лі, Ф.Гонсалес та ін.

Аналіз стану захисту інформаційних потоків (ІП) в комп'ютеризованих системах свідчить, що в цілому стан розв'язання цієї задачі далекий від

досконалості. Тим більше, що виникає потреба у побудові стійких і продуктивних методів та алгоритмів шифрування ІП в комп'ютерних мережах з врахуванням тенденцій зростання вимог до необхідного рівня захисту різних типів ІП. До таких методів можна віднести алгоритми з використанням математичного апарату еліптичних кривих (ЕК), які, як показав світовий досвід, забезпечують високий рівень захисту в порівнянні з відомими.

Удосконалення систем захисту інформаційних потоків з використанням математичного апарату ЕК можливе шляхом підвищення ефективності мережевих технологій, алгоритмів, спеціалізованих процесорів і методів опрацювання на основі організації розподілених обчислень у комп'ютерних мережах.

Тому розробка підходів, методів, алгоритмів, криптографічних комп'ютерних засобів захисту інформації з використанням мережевих технологій та високопродуктивних спецпроцесорів, особливо для проблемно-орієнтованих (ПОКС) та спеціалізованих комп'ютерних систем (СКС) на основі різних теоретико-числових базисів (ТЧБ) є актуальною науковою задачею.

Зв'язок роботи з науковими програмами, планами і темами.

Дисертаційна робота виконувалася у рамках науково-дослідних робіт кафедри комп'ютерної інженерії Тернопільського національного економічного університету "Методи та засоби реалізації алгоритмів захисту інформації, стійких до атак на реалізацію" (Державний реєстраційний номер 0105U008181) та "Паралельні методи та засоби реалізації алгоритмів захисту інформації в комп'ютерних мережах з використанням математичного апарату еліптичних кривих" (Державний реєстраційний номер 0109U000035).

Мета і задачі дослідження.

Метою досліджень є зменшення складності та підвищення швидкодії алгоритмів та спеціалізованого програмно-апаратного опрацювання інформаційних потоків за умови застосування еліптичних кривих.

Для досягнення поставленої мети в дисертаційній роботі необхідно розв‘язати низку взаємопов‘язаних **задач**:

– дослідити системні характеристики різних класів архітектур комп’ютерних мереж (КМ) та трафіків;

– проаналізувати сучасний рівень захисту ІП в комп’ютерних мережах;

– оцінити ефективність мережевих та програмно-апаратних засобів захисту ІП;

– розробити та дослідити методи та швидкодіючі алгоритми опрацювання ІП на основі ТЧБ Радемахера-Крестенсона в задачах захисту інформації;

– розробити ефективні алгоритми генерування параметрів ЕК стійких до різного виду атак;

– розробити швидкодіючі алгоритми пошуку порядку ЕК з використанням мережевих технологій;

– дослідити особливості та оцінити часову складність основних операцій алгоритму Шуфа в задачах захисту ІП з застосуванням ЕК;

– розробити спеціалізовані програмно-апаратні засоби опрацювання ІП за умови застосування ЕК.

Об'єкт дослідження – процеси програмно-апаратного опрацювання інформаційних потоків в комп’ютерних мережах за умови застосування еліптичних кривих.

Предмет дослідження – методи, алгоритми та засоби зменшення часової складності опрацювання інформаційних потоків на основі використання ТЧБ Радемахера-Крестенсона.

Методи дослідження. Для дослідження та аналізу програмно-апаратних засобів опрацювання інформаційних потоків в комп’ютерних мережах використана теорія чисел, алгебра Евкліда та теорія Галуа, теорія інформації, теорія алгоритмів, теорія графів та теорія цифрових автоматів.

Наукова новизна отриманих результатів полягає в наступному:

Вперше:

–розроблено метод опрацювання ІП у КМ за умови застосування ЕК на основі матричних програмно-апаратних засобів які, на відміну від відомих, дали змогу шляхом застосування теоретико-числових базисів Радемахера-Крестенсона зменшити часову складність з експоненційної до лінійної або лінійно-логарифмічної;

– побудовано аналітичні вирази характеристик часової складності формування та опрацювання ІП за умови застосування ЕК з використанням системи числення залишкових класів, ТЧБ Радемахера-Крестенсона, розмежованої системи числення Радемахера-Крестенсона, які склали теоретичну основу зменшення часової складності компонентів алгоритму Шуфа, що, на відміну від існуючих, дало можливість підвищення рівня захисту ІП в сучасних та проектованих КМ;

– розроблено теоретичні засади матрично-модульних операцій модулярного множення та експоненціювання які, на відміну від існуючих, ґрунтуються на використанні теоретико-числових базисів Радемахера-Крестенсона та дають змогу зменшити часову складність алгоритму пошуку порядку ЕК.

Отримав подальший розвиток:

– метод захисту ІП з використанням ЕК на основі генерування їх параметрів, що, на відміну від класичних, дало змогу зменшити часову складність алгоритмів пошуку залишків чисел великої розрядності, знаходження найбільшого спільного дільника (НСД), модулярного множення, експоненціювання та пошуку оберненого елемента за модулем за рахунок використання ТЧБ Крестенсона та Радемахера-Крестенсона.

Практичне значення одержаних результатів.

1. Розроблено високопродуктивні алгоритми модулярного множення та експоненціювання, пошуку найбільшого спільного дільника, оберненого алгоритму за модулем шляхом використання розробленого математичного апарату матрично-модульних операцій у базисі Крестенсона-Радемахера, що

дозволило зменшити на 1-2 порядки часову складність базових операцій алгоритму Шуфа.

2. Запропоновано апаратні засоби реалізації модульних операцій над числами великої розрядності та розроблено схемотехнічні рішення відповідних спеціалізованих процесорів, які можна використати у засобах шифрування ІП.

3. Розроблено програмні пакети формування параметрів еліптичних кривих та спеціалізовані високопродуктивні програмно-апаратні засоби опрацювання інформаційних потоків в комп'ютерних мережах, які можна використати для підвищення рівня захисту.

Результати досліджень використані в навчальному процесі на кафедрах комп'ютерної інженерії та спеціалізованих комп'ютерних систем Тернопільського національного економічного університету при викладанні дисциплін: «Комп'ютерні системи», «Захист інформації в комп'ютерних системах», «Проектування спеціалізованих комп'ютерних систем», а також впроваджені на ТОВ ТИБР «Стріла» для захисту інформаційних потоків в дистрибутивних та корпоративних комп'ютерних мережах.

Особистий внесок.

Дисертаційна робота є результатом самостійної роботи автора. У друкованих працях, опублікованих у співавторстві, автору належить: Дисертаційна робота є результатом самостійної роботи автора. У друкованих працях, опублікованих у співавторстві, автору належить:

– метод перетворень КТЗ в матрично-розмежованому базисі Радемахера Крестенсона;

– методи пошуку НСД у базисі Крестенсона;

– метод шифрування на основі алгоритмів RSA та Ель-Гамаля з використанням ТЧБ Радемахера-Крестенсона;

– теорія та оптимізація опрацювання чисел великої розрядності у базисі Крестенсона;

– метод модулярного множення та експоненціювання з використанням розмежованої системи числення Радемахера-Крестенсона;

– алгоритм пошуку простих чисел та аналітика взаємнопростих чисел спеціального виду на основі використання базису Крестенсона;

– структура високопродуктивного спецпроцесора в системі СЗК для виконання операцій з числами великої розрядності;

– високопродуктивний генетичний алгоритм пошуку параметрів ЕК;

– метод генерування параметрів ЕК із застосуванням символів Якобі;

– формалізована математична модель захисту інформаційних потоків та апаратно-програмна реалізація їх опрацювання.

Апробація результатів дисертації. Основні результати дисертаційної роботи доповідались і обговорювались на:

- міжнародній конференції “Сучасні проблеми радіотехніки, телекомунікації та комп’ютерні науки”, TCSET’2004, 24–28 лютого, 2004 року;

- III Міжнародній конференції студентства та молоді "Світ інформації та телекомунікацій-2006" Україна, Київ, ДУІКТ 26–27 квітня 2006 року;

- XII науковій конференції Тернопільського державного технічного університету імені Івана Пулюя, Тернопіль, ТДТУ, 14–15 травня 2008 року;

- X міжнародній конференції «Досвід проектування і застосування САПР в мікроелектроніці», (CADSM-2009), Львів-Поляна, Україна, 19–24 лютого 2009 року;

- VII міжнародній науково-практичній конференції «Проблеми впровадження інформаційних технологій в економіці», Ірпінь, 23–24 квітня 2009 року;

- міжнародній науково-практичній конференції “Сучасні проблеми радіо-техніки, телекомунікації та комп’ютерні науки”, TCSET’2010, 23–27 лютого, 2010;

- проблемно-науковій міжгалузевій конференції «Інформаційні проблеми комп’ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління (ПНМК-2010), Україна, Бучач, 01–04 червня

2010 року;

- XI міжнародній науково-практичній конференції «Досвід проектування і застосування САПР в мікроелектроніці», (CADSM-2011), Поляна-Свалява, Україна , 23–25 листопада 2011року.

Публікації. Основні положення дисертаційної роботи в повному обсязі висвітлені у 18 працях, з яких 11 статей у науково-технічних журналах, (з них 2 одноосібних), які входять до переліку періодичних фахових видань, затверджених ВАК України, 7 публікацій у збірниках матеріалів і тезах доповідей науково-технічних конференцій

Структура дисертації. Дисертаційна робота складається зі вступу, 4 розділів, загальних висновків, списку використаних джерел та додатків. Загальний обсяг дисертації становить 201 сторінку, з них 142 сторінки основного тексту, містить 35 рисунків, 32 таблиці, 12 додатків, список використаних джерел із 122 найменувань.

РОЗДІЛ 1
СИСТЕМНІ ХАРАКТЕРИСТИКИ КОМП'ЮТЕРНИХ МЕРЕЖ ТА ЗАДАЧІ ОПРАЦЮВАННЯ ІНФОРМАЦІЙНИХ ПОТОКІВ.

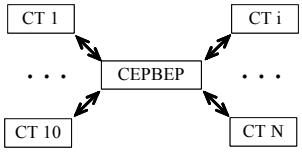
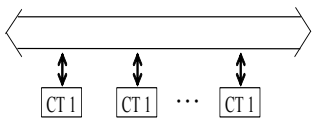
1.1 Класифікація архітектур та трафіків комп'ютерних мереж.

Дослідження та аналіз архітектур розподілених комп'ютерних систем, які використовуються в локальних, проблемно-орієнтованих та спеціалізованих комп'ютерних мережах [5,18,50,60,95-97], дозволяє визначити наступні класи їх архітектур, представлених в табл. 1.1., до яких належать: однорівнева, багаторівнева, безпровідна, з відкритим оптичним каналом. Основним показником ефективності архітектур розподілених комп'ютерних систем є коефіцієнт емерджентності, який визначається згідно рівняння 1.1:

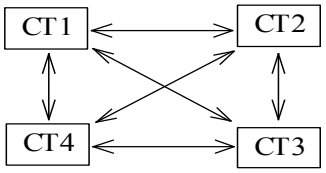
$$K_e = \frac{n_з}{n_e}, \qquad (1.1)$$

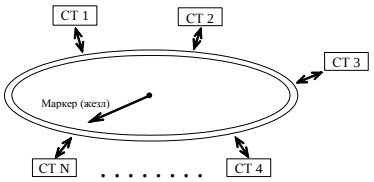
де $n_з$ - число звязків, n_e - число компонентів.

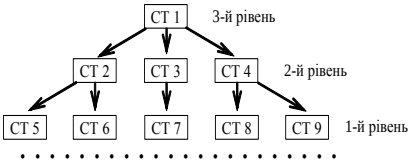
Таблиця 1.1 – Тип та архітектура комп'ютерних мереж

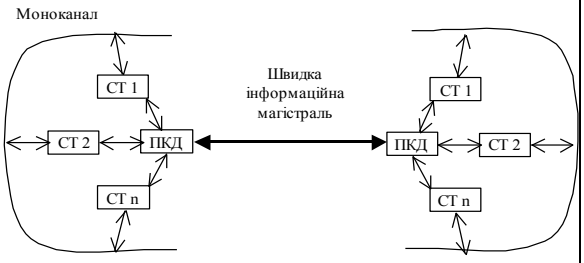
№	Тип та архітектура КМ	
Однорівневі архітектури		
1	1. Зіркова  $K_e = 3$	2. Моноканал  $K_e = 1$

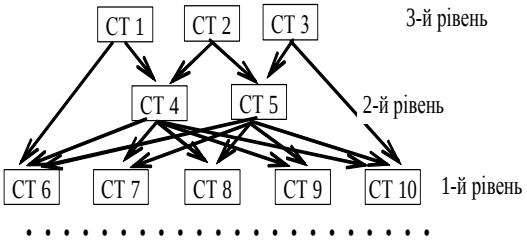
Продовження таблиці 1.1

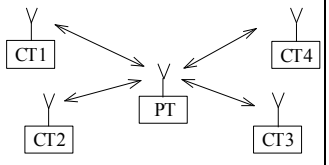
<table>
<tr><td>№</td><td colspan="3">Тип та архітектура КМ</td></tr>
<tr><td colspan="4">Однорівневі архітектури</td></tr>
<tr><td rowspan="2">2</td><td colspan="2">3. Систолічна
 

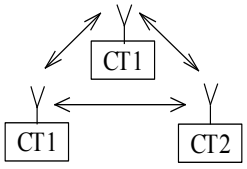
$K_e=10$</td><td colspan="2">4. Кільцева
 

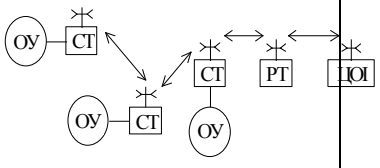
$K_e=1$</td></tr>
<tr><td colspan="4">Багаторівневі архітектури</td></tr>
<tr><td rowspan="2">3</td><td colspan="2">5. Ієрархічна
 

$K_e=1$</td><td colspan="2">6. Зірково-магістральна
 

$K_e=10$</td></tr>
<tr><td rowspan="2">4</td><td colspan="4">7. Мережно-ієрархічна
 

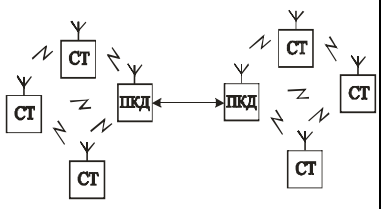
$K_e=10$</td></tr>
<tr><td colspan="4">Безпровідні архітектури</td></tr>
<tr><td rowspan="2">5</td><td>8. З пасивним ретранслятором
 

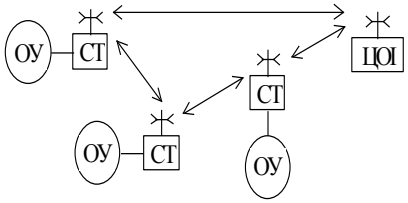
$K_e=3$</td><td>9. Безретрансляційна
 

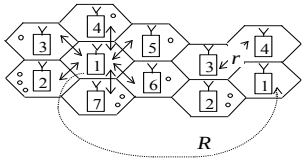
$K_e=10$</td><td colspan="2">10. З активним ретранслятором
 

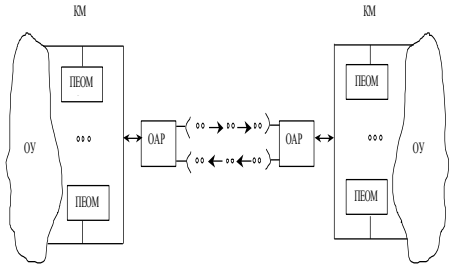
$K_e=4$</td></tr>
</table>

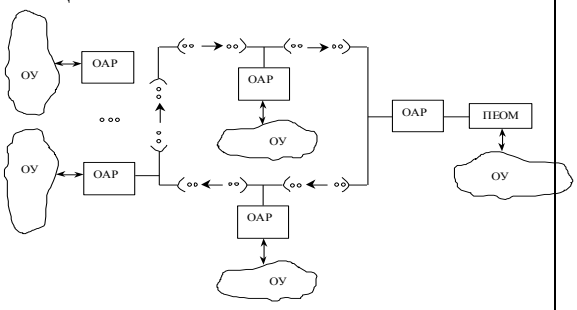
Продовження таблиці 1.1.

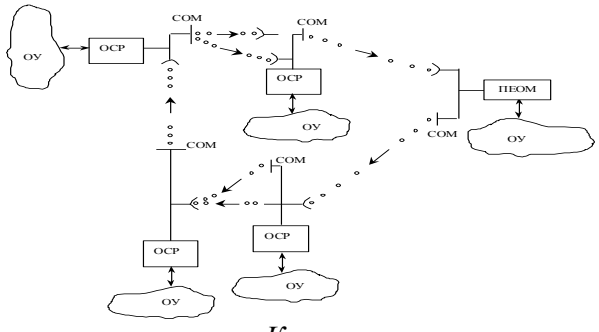
<table>
<tr><td>№</td><td colspan="3">Тип та архітектура КМ</td></tr>
<tr><td colspan="4">Безпровідні архітектури</td></tr>
<tr><td>6</td><td>11. Зірково-магістральна
 

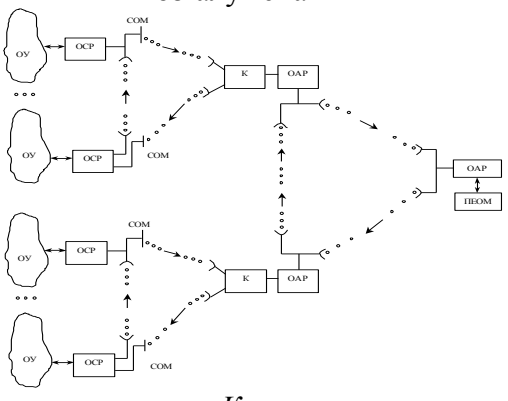
$K_e=10$</td><td>12. З активним ретранслятором та кільцевою структурою
 

$K_e=6$</td><td>13. Сотова
 

$K_e=6$</td></tr>
<tr><td colspan="4">Архітектури КМ з відкритими оптичними каналами зв’язку</td></tr>
<tr><td>7</td><td colspan="2">14. Високошвидкісна дуплексна
 

$K_e=1$</td><td>15. Середньошвидкісна кільцева
 

$K_e=3$</td></tr>
<tr><td>8</td><td>16. Низькошвидкісна кільцева
 

$K_e=3$</td><td>17. Розгалужена
 

$K_e=4$</td></tr>
</table>

З таблиці 1.1 видно, що найкращими показниками емерджентності характеризуються наступні архітектури: мережно-ієрархічна; систолічна; безпровідна безретрансляційна; безпровідна зірково-магістральна; зірково-

магістральна. На рис 1.1 показана гістограма коефіцієнтів емерджентності різних типів архітектур КМ.

Аналіз рис. 1.1 показує, що найвищим коефіцієнтом емерджентності характеризуються мультипроцесорні системи концентрованого опрацювання ІП, які відповідають автономним СКС спеціалізованих мобільних засобів, літаків, супутників і потребують високого рівня захисту інформаційних даних від несанкціонованого доступу.

Високим рівнем емерджентності характеризуються багаторівневі зірково-магістральні та мережно-ієрархічні КМ, а також безпровідні КМ з активними ретрансляторами. До класу таких систем належать СКС атомних електростанцій, нафто-газових, хімічних та інші екологічно небезпечних об'єктів.

У роботах [5,18,97] показано, що найбільш перспективною архітектурою КМ, в якій доцільно використовувати спецпроцесори, аналого-цифрові кодеки та АЦК є безпровідна зірково-магістральна архітектура. Ефективність трафіку передавання даних досліджуваним класом спеціалізованих процесорів при реалізації різних процедур доступу до віддаленого концентратора даних досліджено у роботах [103,104].

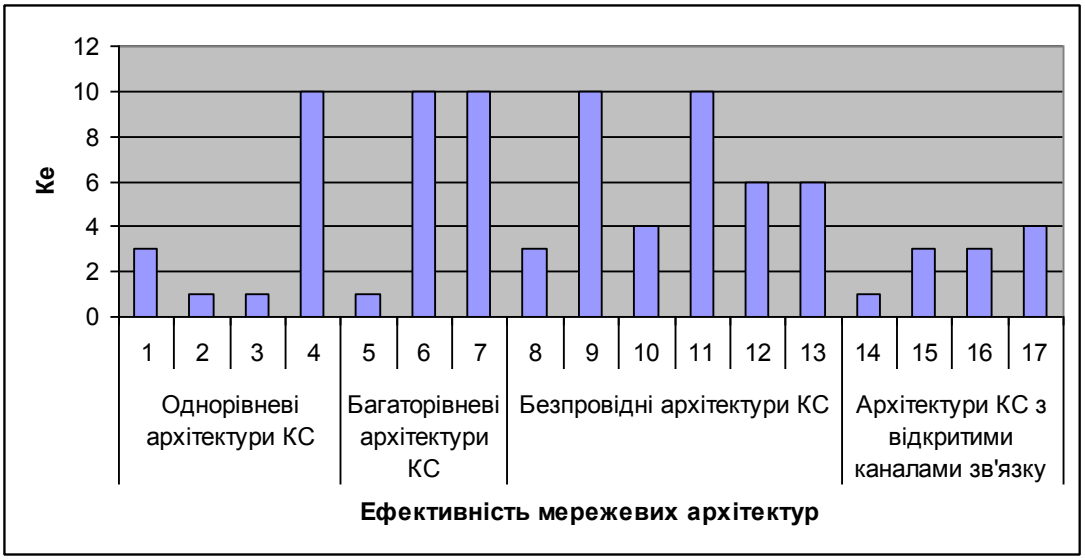


Рисунок 1.1 – Ефективність мережевих архітектур за параметром еммерджентності. Одноріневі: 1-зіркова; 2-моноканал; 3-кільцева; 4-систолічна; багаторівневі: 5-ієрархічна; 6-зіково-магістральна; 7- мережно-

ієрархічна; безпровідні: 8- з пасивним ретранслятором; 9- безретрансляційні; 10- з активним ретранслятором; 11- зірково-магістральна; 12- з активним ретранслятором та кільцевою структурою; 13- сотова; з відкритими оптичними каналами зв'язку: 14- високошвидкісна дуплексна; 15- середньошвидкісна кільцева; 16- низькошвидкісна кільцева; 17- розгалужена.

Оцінка архітектур та трафіків КМ на основі коефіцієнта емерджентності найкраще відображає структурно-організований інтелект КМ. В той же час, дана оцінка не відображає умов необхідного рівня захисту ІП КМ від несанкціонованого доступу, що потребує їх класифікації з врахуванням вразливості, умов захисту та характеру секретності ІП, які можуть циркулювати в різних архітектурах. Спроба такої систематизації виконана у роботі [118]. У табл. 1.2 на основі експертних оцінок коефіцієнта захисту $K_з$ приведена класифікація мереж по характеру секретності та необхідного рівня захисту ІП від несанкціонованого доступу.

Таблиця 1.2–Класифікація мереж по характеру секретності та захисту.

№ п/п	Класифікація СКС по характеру секретності та захисту	$K_з$
1	Державного значення: служба безпеки України, національний банк України, Система Президента України, та ін	10
2	Банківські системи	9
3	Відомчі системи(системи стратегічних обєктів, військові)	8
4	Екологічно-небезпечні(атомні, нафтогазові та ін.)	7
5	Транспортні	6
6	Інтернет	5
7	Регіональні комп'ютерні мережі	4
8	Локальні мережі	3
9	Низові комп'ютерні мережі	2
10	Інформаційно-вимірювальні керуючі системи	1

На основі оцінок коефіцієнта емерджентності КМ $K_е$ та коефіцієнта захисту ІП $K_з$ запропоновано критерій:

$$K_{зе} = \frac{K_з}{K_е}, \qquad (1.2)$$

який одночасно враховує необхідний рівень захисту ІП з врахуванням їх топологій та структурного інтелекту.

Слід зауважити, що архітектури ІП з врахуванням коефіцієнта емерджентності можна поділити на п’ять груп. До першої групи відносяться мережі з $K_е = 1$, до другої з $K_е = 2{,}5$, третьої $K_е = 4$, четвертої $K_е = 6$, п’ятої $K_е = 10$. У табл.. 1.3 приведені показники критерію $K_{зе}$ забезпечення певного рівня захисту з врахуванням інтелекту мережі.

Таблиця 1.3 – Показники критерію $K_{зе}$ забезпечення певного рівня захисту з врахуванням інтелекту мережі

$K_з$ \ $K_е$	1	2,5	4	6	10
10	0,1	0,25	0,4	0,6	1
9	0,11	0,28	0,44	0,67	1,11
8	0,125	0,3125	0,5	0,75	1,25
7	0,14	0,36	0,57	0,86	1,43
6	0,167	0,42	0,67	1	1,67
5	0,2	0,5	0,8	1,2	2
4	0,25	0,625	1	1,5	2,5
3	0,33	0,83	1,33	2	3,33
2	0,5	1,25	2	3	5
1	1	2,5	4	6	10

Проведені дослідження дозволяють встановити, що відповідно до характеристик емерджентності мережі і її коефіцієнтів захисту забезпечення оптимального рівня $K_{зе}$ досягається тоді, , коли $K_{зе}$ приймає значення, близького до 1 (рис1.2).

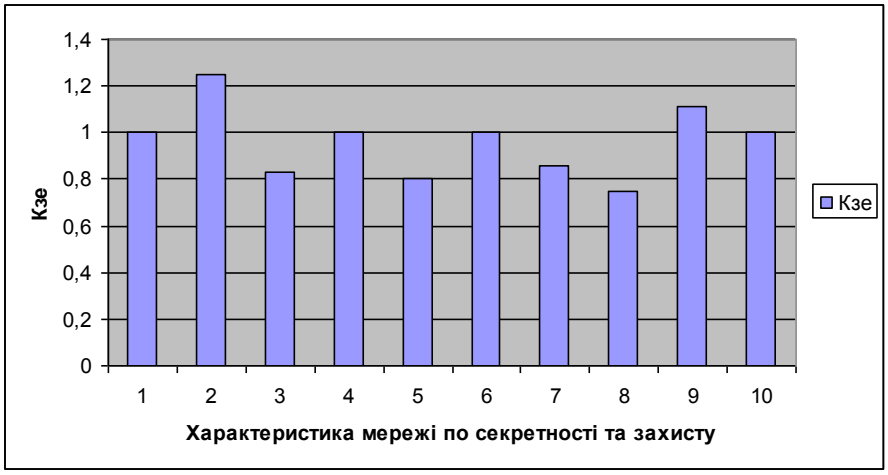


Рисунок 1.2 – Оцінка критерію $K_{зе}$ відповідно до характеристик емерджентності мережі та її коефіцієнтів захищеності.

Результати досліджень показали, що для забезпечення необхідного рівня захисту інформаційних потоків в комп'ютерних мережах необхідно вибирати такий клас архітектур, для яких співвідношення коефіцієнту захищеності та інтелектуальності мережі наближається до 1.

1.1.1. Системні об'єкти та моделі комп'ютерних мереж.

Глобальна модель комп'ютерної системи вперше була запропонована Николайчуком Я.М. у вигляді взаємодії п'ятьох типів системних об'єктів (рис.1.3) [53].

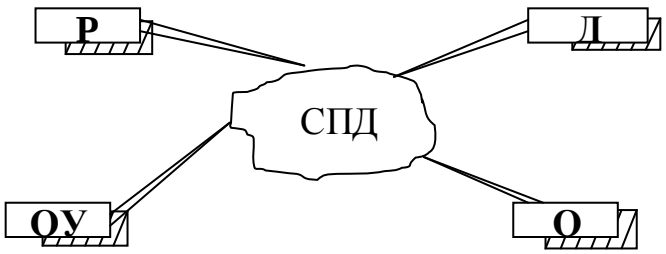


Рисунок 1.3 – Глобальна модель комп'ютерної системи: Р – процесор, Д – дані, СПД – система передавання даних, ОУ – об'єкт управління, О – оператор.

В середовищі КМ системні об’єкти виконують чотири системні функції: формування даних, передавання даних, цифрова обробка даних, приймання та зберігання даних. Тобто кожен з системних об’єктів (СО) може бути одним з функціональних об’єктів наступного типу: джерело інформації (ДІ), середовище передавання інформації (СПІ), середовище цифрової обробки інформації (СОІ), приймач інформації (ПІ).

Отже, системні об’єкти КМ характеризуються дуальними властивостями, що в значній мірі ускладнює методологію проектування та теоретичні основи оптимізації параметрів КМ.

Виходячи з класифікації п’яти системних об’єктів КМ, можна побудувати таблицю пар їх взаємодії через інтерфейсні засоби комунікацій (табл.1.4).

Таблиця 1.4.–Взаємодія системних об’єктів КМ через інтерфейсні засоби комунікацій

	Р	Д	СПД	О
Р	Р→Р	Р→Д	Р→СПД	Р→О
Д	Д→Р	Д→Д	Д→СПД	Д→О
СПД	СПД→Р	СПД→Д	СПД→СПД	СПД→О
О	О→Р	О→Д	О→СПД	О→О

Очевидно, що для вивчення названих у табл.1.4. системних взаємодій СО та використання їх при проектуванні КМ необхідно описати взаємодію 16-ти пар. У табл. 1.4 системні об’єкти Р, Д і СПД виділені, оскільки вони є базовими при формуванні та опрацюванні ІП у комп’ютеризованих системах [3, 19, 21, 47].

Аналогічні зв’язки можуть бути застосовані для інших системних об’єктів, у яких формальні параметри S будуть відрізнятися наступними системними функціями: P – процесор (апаратне, системне та прикладне програмне забезпечення); $Д$ – дані (зберігання даних в архівах, способи кодування даних, захист від помилок, захист від несанкціонованого доступу, семантичні властивості даних і.т.д.) ; $СПД$ – мережеве програмне

забезпечення, оптимізація маршрутів передавання даних, використання спецканалів, інформаційна технологія моделювання руху даних ; OV – характеристики стаціонарності, нестаціонарності, квазістаціонарності, інформаційні технології кодування станів та контролю їхнього відхилення від норми, статистичні, кореляційні та ентропійні моделі, функції керування та побудови моделей руху даних; O – система знань та професійних навиків і.т.д.

У таблиці 1.5 подано основні аналітичні моделі характеристик системних об'єктів комп'ютерної мережі, які є базою для теоретичної формалізації руху ІП.

Таблиця 1.5

№	Аналітичні моделі характеристик системних об'єктів комп'ютерної мережі.	Характеристики СО
1	2	3
1	$Eco = F(T,V,M,S)$ $0 \le T \le T_0;$ $0 \le V \le V_0;$ $0 \le M \le M_0;$ $0 \le S \le S_0,$	Загальний випадок опису ресурсних характеристик СО, та межі змін параметрів проектованої КС
2	$E_{co} = \frac{T+V+M+S}{4}$, $0 \le E_{co} \le 1$	Функція адаптивності, що забезпечує діапазон зміни від 0 до 1, та відповідає гіпотезі про статистичну незалежності ресурсних параметрів СО
3	$P_{co} = E_{co} \cdot P_0 - V_0$	Економічна собівартість руху даних з врахуванням прибутків та затрат на реалізацію функцій СО
4	$P_{co} = \frac{T+V+M+S}{4} \cdot P_0 - V_0$	
5	$Eco = F(T,V_R,V_W,M,S)$	Розширений функціонал системних об'єктів, з врахуванням параметру швидкості виконання системних операцій, а саме швидкості запису та швидкості зчитування, дозволяє врахувати асиметричність характеристик швидкодії вхідних та вихідних пристроїв системних об'єктів

де T – час використання ресурсу, V – швидкість виконання системних операцій (формування, передавання, цифрова обробка та зберігання даних), M – об'єм використовуваного ресурсу пам'яті, S – системні функції, T_0 – час

формування, передавання, цифрової обробки та зберігання даних, використання технічного засобу та інше, V_0–пропускна здатність каналу зв'язку, максимальна швидкість читання/запису, максимальна частота обміну даними та інше, M_0– максимальний об'єм пам'яті, що використовується (ОЗП, ПЗП, магнітних, оптичних, та твердих копій носіїв), S_0– максимальний ресурс системних функцій (операційні системи, пакети прикладних програм тощо), P_0, V_0 – відповідно прибутки та затрати на реалізацію функцій СО, V_R–швидкість запису (вхідний інформаційний потік), V_W–швидкість зчитування (вихідний інформаційний потік)

Аналітичні моделі характеристик системних об'єктів КМ є базою для теоретичної формалізації процесів формування та опрацювання інформаційних потоків. За допомогою моделей (табл.1.5) можна проводити аналіз ефективності та розробку проектів КМ з проблемною орієнтацією для конкретних технологічних процесів та підприємств [23, 40, 75, 83].

Ці показники є важливими для аналізу, вдосконалення, підвищення швидкодії при формуванні та опрацюванні ІП.

1.1.2. Характеристики структуризації інформаційних потоків комп'ютерних мереж.

Паралельну систему з k абонентами, з'єднаних з пам'яттю колективного доступу (ПКД), досліджено у роботі [102], окремими каналами зв'язку – [103] (рис.1.4).

На низовому рівні дистрибутивних та корпоративних відомчих КМ існує тільки зв'язок сенсор – ПКД, що не дозволяє синхронізувати процес передачі даних від формувачів інформаційних потоків (ФІП). Продуктивність такої системи можна розрахувати за наступною формулою [103]:

$$P = \frac{\sum_{i=0}^{N} \frac{K_i}{H}}{N}, \qquad (1.3)$$

де P – коефіцієнт продуктивності системи, K_i – кількість активних ФІП в і-й момент часу, H – загальна кількість формувачів інформаційних потоків системи, N – кількість відліків часу в дискретному просторі.

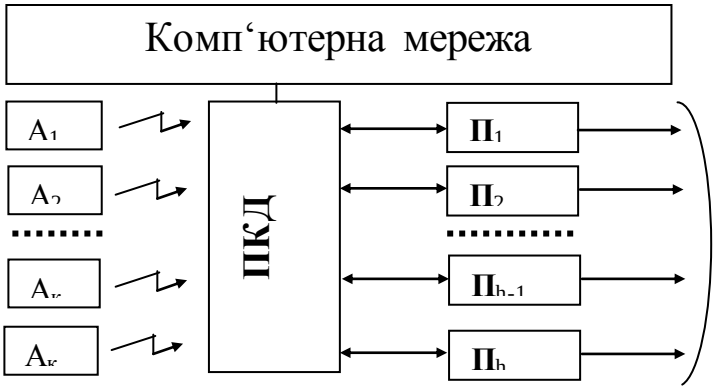


Рисунок 1.4 - Структура системи низового рівня дистрибутивних КС на базі ПКД ($А_1$-$А_к$ – автономні сенсори, ПКД – пам’ять колективного доступу, $П_1$-$П_h$ – процесори).

На основі використання математичної моделі (1.3) отримано значення продуктивності при 30000 формувачах інформаційних потоків комп’ютерної системи (рис.1.5).

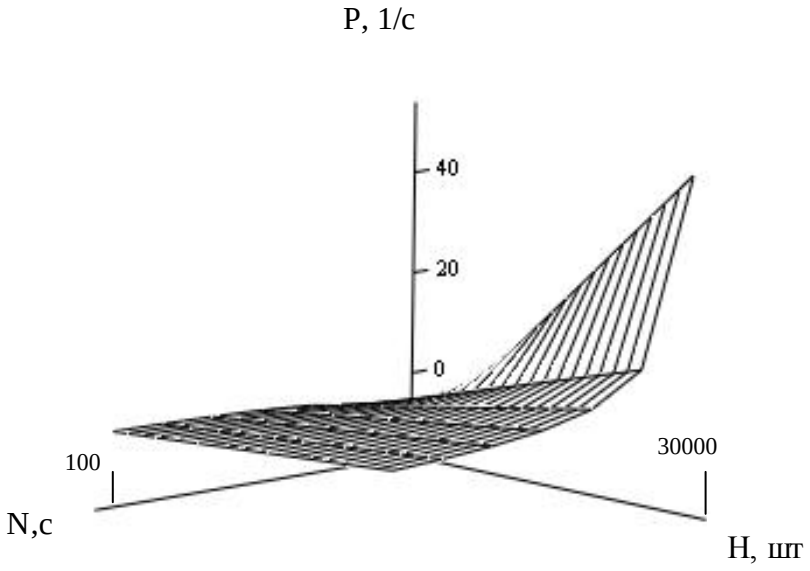


Рисунок 1.5 – Продуктивність системи.

Результати досліджень показали, що максимальна продуктивність досягає 40 % при використанні асоціативної оперативної пам'яті системи з паралельним доступом.

При збільшенні часу обміну інформацією в комп'ютерній системі спостерігається зменшення продуктивності. Тому для ефективного використання КМ потрібно максимально збільшити обсяг оперативної пам'яті системи, оскільки її ефективність в значній мірі залежить від часу обміну інформацією між формувачами ІП системи.

В результаті аналізу трафіку автономних сенсорів безпровідних архітектур КМ можна зробити висновок, що СП низових рівнів безпровідних зірково-магістральних КМ забезпечують максимальну ефективність при режимі паралельного асинхронного формування та передавання даних. Незважаючи на велику ефективність зірково-магістральних мереж, такі архітектури отримали недостатньо широке застосування, оскільки вони на стадії розвитку і використовуються як проблемно-орієнтовані КС.

Методологія побудови моделей руху даних була розроблена Дж. Мартіном [97], теорія моделей руху даних значно розвинена і вдосконалена Я.М. Николайчуком [26]. Методологія, формалізація та атрибути моделей руху даних достатньо повно викладені, оновлені і доповнені в працях І.Р.Пітуха [61,62] та Н.Я. Возної [27]. В цьому аспекті, як показано в роботі Н.Я.Возної та Я.М. Николайчука [26], найбільш актуальними є такі задачі:

- широке використання математичних основ новітніх ТЧБ: Крестенсона, Галуа та ін. для представлення даних у вигляді фреймів СД, максимально адаптованих до різних рівнів РКС та роботи в умовах різноманітних атак;
- вдосконалення методів побудови моделей руху даних та перспективні методи їх структуризації у вигляді фреймів, епюр собівартості руху даних;
- дослідження засобів формування даних на основі критеріїв оптимізації;
- методи побудови багаторівневих моделей руху даних РКС;

- вдосконалення алгоритмів реалізації стратегії проектування РКС, які забезпечують необхідний рівень захисту інформаційних потоків;
- розробки спеціалізованих апаратно-програмних засобів формування та опрацювання інформаційних потоків.

Створення моделей складних комп'ютеризованих систем, формування та опрацювання інформаційних потоків є актуальною задачею. Основні напрямки розвитку таких систем включають: вдосконалення теорії, методології та практики, проектування та діагностику функціонування в реальному масштабі часу.

Вирішення такої задачі в методологічному плані потребує: проблемної орієнтованості, цілісності, складності, невизначеності, адаптивності та універсальності комп'ютеризованої системи [91].

Основними завданнями при створенні КС є визначення ступеня цілеспрямованості та мети функціонування системи, можливості опису системи однією моделлю, оцінка ентропії, що відображає необхідну кількість керуючої інформації, можливості пристосування системи до впливу зовнішніх факторів, а також опис системи математичними моделями, що мають однакову структуру незалежно від класу об'єктів – джерел інформації.

В роботі Пітуха І.Р [61] показано суть одного з існуючих методів організації руху інформаційних потоків в однорівневих КС, який полягає у виконанні функцій у наступній послідовності виконання алгоритмів:

$$\odot \bigcirc \otimes \Rightarrow F_2\left\{\begin{array}{c} E_p = F(T,V,M,S) \\ E_d = F(T,M,V_R,V_W,S) \\ E_{OY} = F(T,M,I,S) \\ E_{СПД} = F(V_R,V_W,P_i,S) \\ E_O = F(T,V_R,V_W,S,M) \\ X_{OY} = F(X(t),M_x,D_x,\sigma_x,R_{xx},R_{xy},M_{ij},S(w),K_{ij},ЛСІМ,I_x) \end{array}\right\} \Rightarrow$$

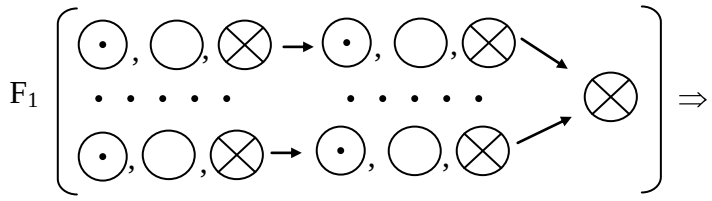
$$F_1[\quad , \quad , \quad] \tag{1.4}$$

$$\Rightarrow F_4[M_1, ТМРД, ДММРД, ..., M_n] \Rightarrow F_5\left\{\begin{array}{c} ЕРД \\ \Delta ЕРД(t_j) \\ \int \Delta ЕРД(T)dT \\ \sum_{i=1}^{n} \int \Delta ЕРД(T)dT \end{array}\right\} \Rightarrow F_6[G_{КС}]$$

$$\Rightarrow F_3[K_{ed}, P-V] \tag{1.5}$$

Опис компонентів (1.4)-(1.5) подано в додатку А.

Возною Н.Я. запропоновано модель руху інформаційних потоків в багаторівневих КС [16]:



(1.6)

$$\Rightarrow F_2\left[\left\{\begin{array}{c} E_p = F(T,V,M,S) \\ E_d = F(T,M,V_R,V_W,S) \\ E_{OY} = F(T,M,I,S) \\ E_{СПД} = F(V_R,V_W,P_i,S) \\ E_O = F(T,V_R,V_W,S,M) \\ X_{OY} = F(S_i : X(t) + ТЕД, M_x, D_x, \sigma_x, R_{xx}, R_{xy}, M_{ij}, S(w), K_{ij}, ЛСІМ, I_x, СД) \end{array}\right.\right] \Rightarrow \quad (1.7)$$

$$\Rightarrow F_3\left[K_{ed}, Д(P-V)\right] \Rightarrow$$

$$\Rightarrow F_4\left[БРМ_1, ДД, БРДММРД, \ldots, Д(M_n + ГАМ)\right] \Rightarrow$$

$$\Rightarrow F_5\left[\left\{ДЦРД, ДЕРД, \Delta ЕРД(t_j), \int \Delta ЕРД(T)dT, \sum_{i=1}^{n} \int \Delta ЕРД(T)dT\right\} \Rightarrow F_6\left[F_{KC}\right] \Rightarrow\right.$$

$$\Rightarrow \quad F_7 \left[\left\{\begin{array}{|c|c|c|c|c|c|c|c|} \hline \text{star} & \text{T} & \text{N} & \text{S} & \{\text{X} & \{\text{M} & \{\text{L} & \text{stop} \\ \hline \Phi & \text{Ci} & \text{Cj} & \text{N} & \text{S} & \text{M} & \text{L} & \text{ТЕД} & \Phi \\ \hline \text{T} & \text{N} & \text{S} & \text{X} & \text{M} & \text{L} & \text{ТЕД} & \text{I} \\ \hline \end{array}\right.\right] \Rightarrow \quad F_8 \left[\left\{\begin{array}{|c|c|c|c|c|c|} \hline \Phi & \text{C} & \text{C} & \text{Y} & \Phi & \\ \hline \Phi & \text{C} & \text{Ci} & \text{Y} & \text{G} & \Phi \\ \hline \Phi & \text{C} & \text{Ci} & \text{V} & \text{Y} & \Phi \\ \hline \Phi & \text{C} & \text{Ci} & \text{W} & \text{Y} & \Phi \\ \hline \end{array}\right.\right],$$

Основні результати щодо вдосконалення методів формування та організації руху ІП у багаторівневих РКС полягають у розширенні функцій організації руху даних на основі:

- багаторівневої матричної моделі руху даних (1.6);
- розширення функцій формування СД шляхом сумісного кодування технологічних даних (x(t)) та ТЕД (1.7) при заданих

квазістаціонарних станах ОУ (S_i:), а також реалізація апаратно-програмних засобів синтезованого вводу алфавітно-цифрових даних;

- формування мінімально надлишкових та захищених від помилок фреймів СД, які формуються, обробляються, зберігаються та передаються;
- диференціація собівартісних характеристик руху даних в активних вузлах МРД: $Д = V = \sum_{i=1}^{m} P_i - \sum_{j=1}^{n} V_j$, де m – число диференційованих компонентів прибутків, n – число диференційованих компонентів витрат в активному вузлі МРД;
- розширення класу МРД до рівня багаторівневих МРД (БМРД);
- врахування диференціації форм документів (ДД) та даних ММ по відповідних активних вузлах МРД, що відповідають відповідним об'єктам МРД;
- розширення сукупності похідних моделей руху даних шляхом включення в їх склад важливого класу граф-часових моделей (ГАМ);
- диференціації циклів руху даних з врахуванням диференціації собівартісних характеристик руху даних в активних вузлах БРДММРД (ДЦРД) та відповідна диференціація епюр руху даних (ДЕРД).

У роботах Возної Н.Я., Николайчука Я.М. [65] наведено алгоритми формування на низових рівнях КС структуризованих даних в цілочисельній формі базису Крестенсена, на низових рівнях КС у нормалізованій досконалій формі, в базизі Галуа у вигляді асинхронного алгоритму та синхронного структуризованого потоку даних рекурентних кодів Галуа з доповненням біт-орієнтованих інформаційних потоків ТЕД (табл.1.7).

Таблиця 1.7 – Алгоритми формування на низових рівнях КС структуризованих даних в різних ТЧБ.

№ п/п	Алгоритми формування структуризованого потоку даних	Аналітичне представлення алгоритмів формування структуризованого потоку даних
1	2	3

Продовження таблиці 1.7

1	2	3
1.	на низових рівнях КС структуризованих даних в цілочисельній формі базису Крестенсена	$x_1 \to x_{i1} \to p_1 \to b_1$ $x_2 \to x_{i2} \to p_2 \to b_2$ ... $x_j \to x_{ij} \to p_j \to b_j$ ... $x_m \to x_{im} \to p_{k-1} \to b_{k-1}$ $D_i \to p_k \to b_k$ $N_k = res \sum_{j=1}^{k} b_j B_j (\mathrm{mod} P);$ $B_j = \frac{P}{p_j} m_j = 1 (\mathrm{mod} P_j)$
2.	на низових рівнях КС в нормалізованій досконалій формі	$x_1 \to x_{i1} \to p_1 \to [b_1]_0$ $x_2 \to x_{i2} \to p_2 \to [b_2]_0$ ... $x_j \to x_{ij} \to p_j \to [b_j]_0$ ... $x_m \to x_{im} \to p_{k-1} \to [b_{k-1}]_0$ $D_i \to p_k \to [b_k]_0$ $[N_k]_0 = res \sum_{i=1}^{k} [b_i]_0 (\mathrm{mod} 1)$ $b_i = \mathrm{int}\, res [N_k]_0 (\mathrm{mod} 1) : P$

де $x_j(t)$ - аналогові дані телеметрії; x_{ij} - цифрові дані телеметрії; D_i – ТЕД; p_1, p_2... p_k – система взаємно простих модулів; b_1, b_2..... b_j.... B_k – набір найменших невід'ємних залишків; Bj – система ортогональних базисів СЗК.

Часова складність алгоритму формування структуризованих даних на низових рівнях КС в нормалізованій досконалій формі в базисі Крестенсона включає характеристики часової складності виконання операцій додавання, множення та модульної операції, що реалізовуються за допомогою універсальних процесорів різної розрядності n. Оцінка функціональних можливостей реалізації арифметики та базових функцій над числами в базисах Радемахера, Крестенсона представлена в табл.1.8.

Таблиця 1.8 – Функціональні можливості досліджуваних ТЧБ

№	Базові операції	Радемахер	Крестенсон
1	2	3	4
1	Додавання	$2nv$	v

Продовження таблиці 1.8

1	2	3	4
2	Зсув	v	v
3	Множення	$2v(2n+1)$	v
4	Рівності	v	v
5	Знакова(старшинства)	nv	-
6	Віднімання	$(3n+5)v$	-
7	Ділення	n^2v	-
8	Модульна	n^2v	$2nv$

У таблиці n – розрядність представлення чисел, v – тривалість спрацювання мікроелектронного вентиля. Експериментальні дослідження проводилися на основі Sempron 3000+, для якого $\nu = \frac{1 \cdot c}{2000\, млн.опер} = 5 \cdot 10^{-10} \frac{c}{опер}$.

1.1.3 Способи кодування інформаційних потоків в комп'ютерних мережах на основі теоретико числових базисів Радемахера та Крестенсона

Способи кодування інформаційних потоків визначаються ТЧБ, які застосовуються для їх представлення [9,17, 20, 22]. Найбільш поширеними ТЧБ в сучасних КС є наступні: унітарний, Хаара, Грея, Радемахера, Крестенсона та Галуа.

Світовий досвід створення процесорів для комп'ютерних систем за останні 50 років, поряд з застосуванням ТЧБ Радемахера, який породжує двійкову систему числення, демонструє тенденцію все ширшого застосування інших ТЧБ. Реалізація спеціалізованих, сигнальних, комутаційних та проблемно-орієнтованих процесорів цифрової обробки даних часто виконується на базі сумісного використання комбінацій названих ТЧБ, наприклад Радемахера-Хаара, Крестонсона-Галуа та ін [56].

Перспективним напрямком розвитку теорії та технологій побудови спеціалізованих програмно-апаратних комп'ютерних засобів є реалізація супершвидкодіючих мультибазисних RCG-процесорів на основі базисів Радемахера, Крестенсона і Галуа [56]. Відомі успішні спроби розвитку теорії

та техніки побудови матричних процесорів на основі двовимірних базисів Радемахера та Галуа, а також конвеєрних спецпроцесорів у базисі Галуа [54].

Спостережувані тенденції розвитку теорії методології та техніки процесорів комп'ютерних систем обумовлені теоретичним та ідейним насиченням можливостей застосування базису Радемахера для побудови арифметико-логічних компонентів процесорів, до яких ставляться все жорсткіші вимоги щодо швидкодії, покращення регулярності структури та розширення функціональних можливостей.

У зв'язку з цим існує проблема глибокого дослідження характеристик «нерадемахівських» ТЧБ та граничних можливостей їх застосування для реалізації компонентів як спеціалізованих, так і універсальних процесорів. При цьому перспективним, крім найбільш сьогодні масового одновимірного (векторного) представлення чисел та виконання арифметико-логічних операцій у базисі Радемахера, є застосування двовимірних систем числення, вертикальної інформаційної технології у базисі Галуа та різних форм багатовимірного представлення чисел у вигляді залишків різних форм СЗК базису Крестенсона [2, 50].

В табл. 1.9 приведені характеристики кодових матриць ТЧБ Радемахера, Крестенсона, які найширше використовуються для кодування та цифрової обробки даних в інформаційних системах, а також мають властивості мінімальної надлишковості по відношенню до наступних базисів унітарного, Хаара, Крейга, Уолша та Грея [6].

Таблиця 1.9 – Характеристики кодових матриць ТЧБ

Базис	Кодові матриці	n-число активних кодових елементів	V – об'єм кодової матриці
1	2	3	4
Радемахера	$M_{Rad} = \begin{vmatrix} 000...00 \\ 000...01 \\ 000...10 \\ 000...11 \\ ..... \\ 111...11 \end{vmatrix}$	$n = \frac{\mathrm{N} \cdot \log_2 N}{2}$	$\mathrm{V}=\mathrm{N}\cdot\log_2 \mathrm{N}$

Продовження таблиці 1.9

1	2	3	4
Крестенсона	$M_{Cres} = \begin{vmatrix} P_1 & P_2 & \dots & P_m \\ 0 & 0 & \dots & 0 \\ 1 & 1 & \dots & 1 \\ 2 & 2 & \dots & 2 \\ 0 & 3 & \dots & 3 \\ \dots & \dots & \dots & \dots \\ 2 & 4 & \dots & 6 \end{vmatrix}$	$n = \prod_{i=1}^{m} P_i$	$V = \sum_{i=1}^{m} \log_2 (P_i - 1)$

У табл. 1.9: N – діапазон представлення чисел, $p_1, p_2, \dots, p_i, \dots, p_m$ – набір взаємо простих модулів СЗК базису Крестенсона, $a_i = p_i - 1$.

Система числення залишкових класів базису Крестенсона, розроблена Акушським І.Я. та Юдіцьким Д.І. [59], особливо її цілочисельна форма, широко використовувалась, починаючи з 70-х років минулого століття для побудови швидкодіючих спеціалізованих процесорів систем повітряної оборони колишнього СРСР.

Нормалізована форма СЗК, запропонована науковою школою проф. Николайчука Я.М., найбільш поширена в телекомунікаційних процесорах інформаційних систем нафтогазової промисловості [57].

У роботі [55] запропоновано чотири аналітичні моделі прямих та зворотніх перетворень залишкових класів (табл.1.10)

Таблиця 1.10 – Аналітичні моделі прямих та зворотніх перетворень залишкових класів

№ п\п	Пряме перетворення форми СЗК	Зворотнє перетворення форми СЗК
1	2	3
1.	Цілочисельна форма СЗК	
	$N_k = (b_1 b_2 \dots b_i \dots b_k)_{(p_1 p_2 \dots p_i \dots p_k)}$ $N_k = b_i (\text{mod } p_i)$, $N_k = a_i p_i + b_i$, $P = \prod_{i=1}^{k} p_i$; $0 \le N_k \le P$.	$b_i = res N_k (\text{mod } p_i)$ $N_k = res \sum_{i=1}^{k} b_i \cdot B_i (\text{mod } P)$, $B_i = \frac{P}{p_i} \cdot m_i \equiv 1 (\text{mod } P_i)$.
2.	Нормалізована форма СЗК	

Продовження таблиці 1.10

1	2	3
	$\frac{N_k}{P} = res\sum_{i=1}^{k}\frac{b_i \cdot B_i (\mathrm{mod}\, P)}{P}$, $[N_k]_0 = res\sum_{i=1}^{k} b_i \cdot \frac{B_i}{P} (\mathrm{mod}\, 1)$, $0 \le [N_k]_0 \le P-1$; $\frac{B_i}{P} = \frac{1}{p_i}$, $\delta_p \le \frac{1}{P}$, $\frac{1}{p_i} = 0.\overbrace{gggggggg}^{n_i}\overbrace{}^{\delta_p}$,	$[N_k]_0 = res\sum_{i=1}^{k} b_i \cdot \frac{m_i}{p_i} (\mathrm{mod}\, 1)$ $[N_k]_0 = res\sum_{i=1}^{k} [b_i]_0 \cdot m_i (\mathrm{mod}\, 1)$, $[b_i]_0 = \frac{b_i}{p_i}$, $0 \le [b_i]_0 \le 1$. $N_k = \mathrm{int}[N_k]_0 \cdot P$,
3.	Досконала форма СЗК	
	$[N_k]_0 = res\sum_{i=1}^{k}[b_i]_0 (\mathrm{mod}\, 1)$.	$b_i = \mathrm{int}\, res\left[N_k \_0 \left(\mathrm{mod}\, 1\right) P_i\right.$
4.	Розмежована форма СЗК	
	$N_k = N_{1k} + N_{2k} + \ldots + N_{ik} + \ldots + N_{nk}$,	$N_k =$ → $b_1=(b_{11}+b_{21}+\ldots+b_{r1}+\ldots+b_{n1}) \mathrm{mod} p_1$; $b_2=(b_{12}+b_{22}+\ldots+b_{r2}+\ldots+b_{n2}) \mathrm{mod} p_2$; … $b_i=(b_{1i}+b_{2i}+\ldots+b_{ri}+\ldots+b_{ni}) \mathrm{mod}\ p_i$; … $b_k\, b_2=(b_{1k}+b_{2k}+\ldots+b_{rk}+\ldots+b_{nk}) \mathrm{mod} p_k$.

У табл. 1.10 N_k – число у позиційній системі числення (у базисі Радемахера); $(b_1 b_2 \ldots b_i \ldots b_k)$ – представлення числа у СЗК; $(p_1 p_2 \ldots p_i \ldots p_k)$ - набір взаємно простих модулів СЗК; b_i - найменший невід'ємний залишок; P – діапазон кодування чисел в СЗК; a_i – ранг; K – число модулів СЗК; B_i – базисні числа СЗК; res – символ операції знаходження найменшого невід'ємного залишку; int – символ операції виділення цілої частини; mod - символ операції по модулю; m_i – ранговий коефіцієнт СЗК; δр – дробова частина в нормалізованій формі СЗК; $[N_k]_0, [b_i]_0$ – відповідно число та залишок в нормалізованій формі базису Радемахера.

В даний час виконуються активні дослідження двовимірних (матричних) форм систем числення базисів Радемахера та Галуа [57].

Результати досліджень часової складності реалізації модульної операції над числами в базисах Радемахера, Крестенсона на основі Sempron 3000+ приведені на рис.1.6.

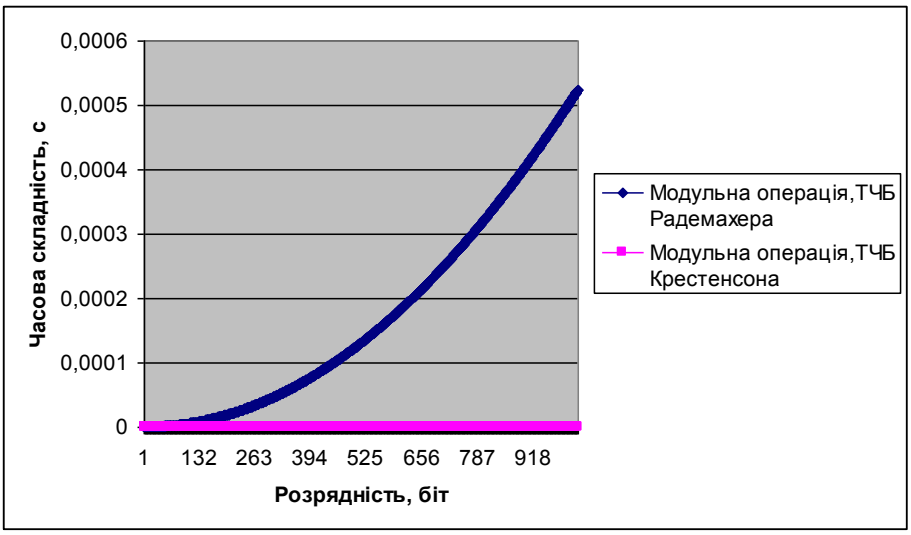


Рисунок 1.6 – Часова складність реалізації модульної операції над числами в базисах Радемахера та Крестенсона в залежності від розмірності.

З рис 1.6 видно, що алгоритм обчислення залишку у базисі Крестенсона характеризується суттєвим збільшенням швидкодії, що є важливою перевагою його застосування на низових рівнях РКС шляхом використання спеціалізованих процесорів та контролерів, які призначені для роботи в промислових умовах при дії впливів вібрації, широкого діапазону температур, вибухобезпечності, мобільного виконання та інших факторів.

Незважаючи на ефективні алгоритми формування структуризованих даних, які реалізуються на основі програмно-апаратних спецпроцесорів і забезпечують захист від помилок та певний рівень захисту інформації від несанкціонованого доступу, потрібно, для забезпечення необхідного рівня захисту, додаткове опрацювання ІП, формування захищених даних, аналіз і ефективне застосування існуючих методів захисту структуризованих даних на основі відомих алгоритмів RSA, Ель-Гамаля, а особливо алгоритмів з використанням математичного апарату ЕК. Використання ЕК є перспективним напрямком вдосконалення захисту ІП структуризованих і неструктуризованих даних на різних рівнях КМ.

1.2. Аналіз сучасного рівня захисту інформаційних потоків

Інформаційні ресурси в сучасних умовах є одним із найважливіших результатів діяльності людського суспільства. Саме тому особлива увага приділяється задачі захисту інформаційних ресурсів.

Всі криптографічні алгоритми з відкритим ключем для розподілу ключів, шифрування ІП тощо спираються на один з наступних типів незворотних перетворень [13]: розклад великих чисел на прості множники або задача факторизації; проблема дискретного логарифмування; обчислення коренів алгебраїчних рівнянь.

Складність системи захисту інформаційних потоків з використанням алгоритму шифрування RSA базується на розкладанні на множники (факторизації) великих чисел. Для надійного захисту інформаційних потоків в системі шифрування RSA використовуються ключі розрядністю не менше 1024-біт. Захищеність більшості інших алгоритмів з відкритим ключем – Ель-Гамаля, DSA і т.д. – грунтується на задачі дискретного логарифмування. Обидві задачі дуже схожі і всі сучасні алгоритми факторизації можна застосовувати до розрахунку дискретних логарифмів мультиплікативної групи в скінченному полі. При оцінці складності операції факторизації чисел визначеної довжини і розв’язання задачі дискретного логарифма багатьма вченими було показано, що ці операції вимагають приблизно однакового обсягу роботи. З цього випливає, що ключі RSA, Ель-Гамаля, DSA одинакової довжини будуть мати приблизно однакову стійкість.

Системи захисту ІП з використанням математичного апарату ЕК запропоновані В. Міллером [13] і Н. Кобліцем [14] в 1986 році й вже близько 25 років інтенсивно аналізуються вченими світу. Особливий інтерес до них обумовлений такими перевагами - швидкодія та порівняно невелика довжина ключа.

Якщо не враховувати вже відомі криптографічно слабкі криві, то стійкість алгоритмів шифрування на ЕК оцінюється як експоненційна. Складність атаки на ключ у цьому випадку експоненціально пов'язана з

довжиною ключа, тобто наростає дуже швидко й для деякої довжини ключа атака стає практично нереалізованою. Стійкість же систем шифрування RSA й Ель-Гамаля субекспоненціальна. Практично це означає, що алгоритми шифрування на ЕК за однакової стійкості має модуль, розмір якого на порядок менший, ніж у попередніх системах шифруванні. Відомо [6, 7], що ЕК із розміром поля 160 біт забезпечує ту ж стійкість, що й традиційні системи захисту розмірності модулів 1024 біт. В табл. 1.11 подані порівняльні характеристики алгоритмів шифрування.

Таблиця 1.11 – Порівняльні характеристики розмірів ключів алгоритмів шифрування RSA, Ель-Гамаля та криптографія ЕК.

Час на злом, MIPS років	Розмір ключа RSA, Ель-Гамаля, біт	Розмір ключа алгоритму шифрування на ЕК, біт	Відношення довжин ключів RSA/ алгоритму шифрування на ЕК
10^{8}	768	132	6:1
10^{11}	1024	160	7:1
10^{20}	2048	210	10:1
10^{78}	21000	600	35:1

Таким чином, як видно з табл. 1.11, алгоритми шифрування ІП на ЕК за однакової стійкості мають розмірність модуля на порядок менше, ніж у RSA та Ель-Гамаля. Тому сьогодні, по суті, немає альтернативи цим алгоритмам шифрування.

У зв'язку з радикальним зменшенням модуля й обсягу пам'яті алгоритми шифрування ІП з використання математичного апарату ЕК вперше знайшли застосування у малогабаритних терміналах, таких як пластикові смарт-карти й мобільні телефони. Не треба доводити необхідність надійності таких систем, причому захист ІП мобільного зв’язку може використатися як додаткова послуга в необхідних випадках.

1.3 Теоретичні основи перспективних методів формування інформаційних потоків захищених від несанкціонованого доступу на еліптичних кривих, постановка задачі досліджень.

ЕК Е над полем $E(F_p)$ описується наступним рівнянням [12]:

$$y^2 \equiv x^3 + ax + b(\text{mod } p), \quad p \in F_p, \qquad \Delta = -(4a^3 + 27b^2) \neq 0, \quad a, b \in F_p. \tag{1.8}$$

де $(x, y) \in E(F_p) \cup O$, O - нескінченно віддалена точка.

Найважливішим аспектом теорії ЕК є введення операції додавання точок $P= (x_1, y_1)$ і $Q = (x_2, y_2)$ як основи для побудови абелевої групи. Їх сумою буде третя точка $R = P + Q = (x_3, y_3)$, яка належить ЕК. Координати цієї точки обчислюються з наступної системи рівнянь:

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2, P \neq Q; \\ y_3 = -y_1 - \lambda(x_3 - x_1), \lambda = \dfrac{y_2 - y_1}{x_2 - x_1} \end{cases} \tag{1.9}$$

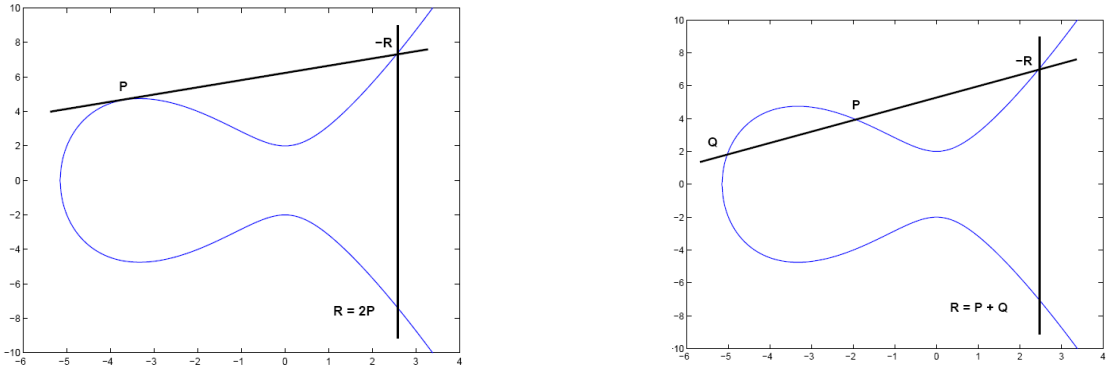
де

$$\lambda = \frac{-y_3 - y_1}{x_3 - x_1}. \tag{1.10}$$

Якщо $P=Q$, $R=2P=(x_3,y_3)$, тоді координати точки $R=(x_3,y_3)$ знаходиться згідно співвідношення:

$$\begin{cases} x_3 = \nu^2 - 2x_1, P = Q; \\ y_3 = -y_1 - \nu(x_3 - x_1), \nu = \dfrac{3x_1^2 + a}{2y_1} \cdot \end{cases} \tag{1.11}$$

Графічне представлення додавання двох точок та знаходження кратних точок на ЕК зображено на рис.1.7.



а б

Рисунок 1.7 - Геометрична інтерпретація подвоєння точки P (а) і додавання точок P и Q (б)

На сьогодні ЕК застосовують для реалізації різноманітних класів систем захисту інформації, зокрема їх можна використовувати для побудови симетричних, асиметричних криптосистем та систем електронного цифрового підпису (ЕЦП). На рис. 1.8 зображено класифікацію сучасних методів криптографії, що базуються на застосуванні еліптичних кривих. Їх аналіз показує, що математичний апарат еліптичних кривих можна застосовувати [1], [2], [84]:

1) в асиметричних системах захисту інформації;
2) в симетричних криптосистемах;
3) для реалізації електоронно цифрового підпису;
4) для електронних платежів;
5) генератори для побудови псевдовипадкових послідовностей.

Слід зауважити, що стосовно симетричних криптосистем, в літературі показано лише теоретичну можливість побудови засобів такого класу, щодо ж практичної реалізації необхідно відзначити, що продуктивність таких систем є нижчою в порівнянні з традиційними.

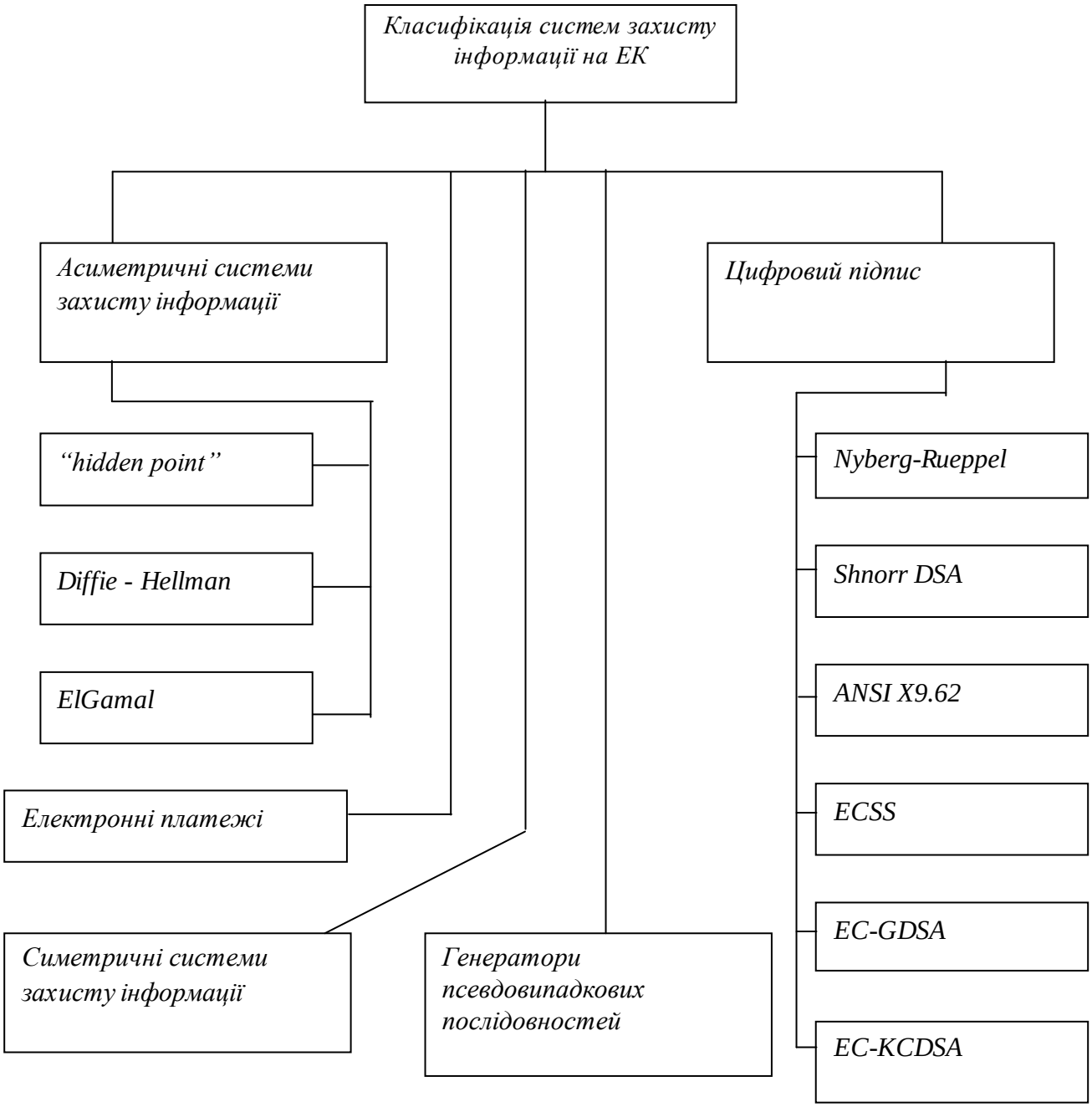


Рисунок 1.8 - Класифікація застосування еліптичних кривих для задач захисту інформаційних потоків.

Незважаючи на вагомі переваги застосування ЕК, поряд з цим існують певні проблеми та труднощі. Зокрема, виділяють такі класи задач: генерування параметрів еліптичної кривої; обчислення порядку еліптичної кривої; дискретне логарифмування.

Ці три класи задач взаємоповязані і є ключовими в системах захисту інформаційних потоків з використанням ЕК: перших два – для шифрування і аналізу стійкості, третій – дешифрування.

Проведений аналіз показує, що для розв'язання задачі знаходження кількості точок на ЕК можна застосувати наступні алгоритми: «крок гіганта-крок малюка», Шуфа, Еткіна та Еліза [1, 71] .

Часова складність першого алгоритму оцінюється як $O\left(q^{\frac{1}{4}+\varepsilon}\right)$, зокрема для обмеженого поля малої потужності, та ґрунтується на теорії Шанкса [1]. Алгоритм приречений на невдачу дуже рідко, тоді і тільки тоді, коли для кожної точки еліптичної кривої P є більш, ніж одне число m в інтервалі $(p+1-2\sqrt{p}, p+1+2\sqrt{p})$, для якої $mP=0$. Це стається тоді, коли показник степеня групи $E(F_p)$ дуже малий, тому в цьому випадку доцільно скористатися алгоритмами Шуфа, Еткіна та Еліза. Часова складність першого алгоритму ґрунтується на обчисленні високих степенів $x^p, y^p, x^{p^2}, y^{p^2}$ по модулю $f_l(x)$, має часову складність $O((\log p)M(n))$ і $O((\log p^2)M(n))$ відповідно бітових операцій, яке зводиться до декількох додавань і множень в кільці. Оскільки елементи кільця мають розмір $l^2 \log p$, то часова складність становить $O(\log p(l^2 \log(p)^2)$ і $O(l(l^2 \log p)^2)$ відповідно. Тут прийнято, що множення двох елементів розмірності n біт відбувається за час, який пропорційний до n^2. Беручи до уваги, що $l = O(\log p)$, і, здійснивши обчислення для кожного l, остаточно отримуємо вираз для роботи, яка витрачається на повне обчислення: $O(\log^8 p)$ [41,42].

Удосконалення алгоритму Шуфа показано в роботах Еткіна та Еліза [45,46]. Більша частина розрахунків зводиться до обчислення X_p і Y_p в кільці

$$F_p[X,Y]/(F(X), Y^2 - X^3 - AX - B). \qquad (1.12)$$

Беручи до уваги, що для F(X) притаманний порядок $(l-1)/2$, а не $(l^2-1)/2$, то для обчислення беруть тільки дії $O(l^2 \log^3 p + l^3 \log^2 p) = O(\log^5 p)$.

Це і є суттєвим щодо значного зменшення часу виконання $O(\log^7 p)$ відповідної дробової частини алгоритму .

Слід зазначити, що підхід Елкіза справджується та є ефективним лише для простих чисел l, для яких φ має власні значення в F_l, тобто для половини простих чисел l, тих, що розпадаються в полі $Q(\sqrt{t^2-4p}$. При цьому потрібно обчислити коефіцієнти многочлена $F(X) \in F_p[X]$. У табл. 1.12 показано складності алгоритмів пошуку порядку ЕК.

Таблиця 1.12 – Часова складність алгоритмів пошуку порядку ЕК

Криптоаналітичний алгоритм	Часова складність
Алгоритм Шуфа	$O(\log^8 p)$
Алгоритм Еткіна та Елкіза	$O(\log^7 p)$
Алгоритм „кроку гіганта – кроку малюка”	$O\left(q^{\frac{1}{4}+\varepsilon}\right)$

Для підвищення стійкості систем захисту інформації необхідно використовувати ЕК, для яких число кількості точок було б максимальним та містило б великий простий дільник. Це вимагає знання точної кількості точок ЕК. На практиці розв’язання задачі обчислення кількості точок на ЕК значно підвищує рівень стійкості системи захисту ІП.

Таким чином, для підвищення високого і необхідного в перспективі рівня захисту ІП в КМ необхідна розробка двох важливих напрямків - методів та засобів формування та опрацювання інформаційних потоків на основі мережевих, а також високопродуктивних програмно-апаратних засобів.

Незважаючи на високу ефективність мережевих технологій формування та захисту ІП від несанкціонованого доступу [72,73,74], сучасні можливості становлення теорії та алгоритмів опрацювання інформаційних потоків на основі різних ТЧБ та високопродуктивних спецпроцесорів також

створює базові основи підвищення ефективності формування, зменшення часової складності, швидкодії та захищеності ІП на основі високопродуктивних програмно-апаратних засобів, як це показано в розділі 1.2 для формування структуризованих базисів на основі ТЧБ Крестенсона, Галуа, Крейга, Уолша та ін.

ВИСНОВКИ ДО РОЗДІЛУ 1

1. Аналіз та дослідження архітектур та трафіків КМ на основі оцінки їх емерджментності показує, що найбільш перспективною архітектурою мережевих технологій опрацювання інформаційних потоків є зірково-магістральна. Архітектури існуючих КМ, типу ієрархічних та багаторівневих, характеризуються нижчим рівнем емерджментності та рівнем трафіку у порівнянні з зірково-магістральною. Однак магістральні архітектури ще не є достатньо поширеними і тиражованими у світовій практиці і розглядаються як перспективні. Таким чином, вдосконалення мережевих технологій захисту інформаційних потоків на основі сучасних архітектур КМ не можуть в повній мірі забезпечити ефективність формування та опрацювання інформаційних потоків для захисту інформації.

2. Систематизація системних об’єктів та їх аналітичних моделей, а також критеріїв оптимізації характеристик КС дозволили обґрунтувати важливість об’єктів формування структуризованих даних та методів їх захисту в КМ на основі матричних моделей руху даних, а також ефективного застосування ТЧБ Крестенсона та Галуа для зменшення часової складності формування інформаційних потоків, які включають не тільки алфавітно-цифрові, але й телеметричні, технологічні дані з елементами їх захисту від несанкціонованого доступу. Тому, захист такого класу даних на різних рівнях КМ включаючи дистрибутивні, корпоративні, відомчі, локальні та глобальні мережі, потребує розвитку спеціалізованих алгоритмів та програмно-апаратних засобів.

3. Аналіз сучасних методів та часових засобів захисту інформації на основі алгоритмів RSA, Ель-Гамаля та інших стандартів показує їх функціональну обмеженість у порівнянні з необхідним рівнем захисту інформації в КС, які розвиваються, що обґрунтовує перспективу удосконалення алгоритмів формування та опрацювання інформаційних потоків з захистом на основі ЕК. В цей же час, алгоритм пошуку порядку ЕК на основі відомого алгоритму Шуфа характеризується експоненціальною

складністю, що потребує глибокого теоретико-часового дослідження вдосконалення та розвитку нових методів формування та опрацювання інформаційних потоків за умов застосування ЕК.

Для досягнення мети вдосконалення та підвищення ефективності захисту інформаційних потоків в КС за умов застосування ЕК необхідно вирішити ряд задач:

1) дослідити системні характеристики та класифікацію архітектур комп'ютерних мереж, трафіків;

2) проаналізувати сучасний рівень захисту інформаційних потоків;

3) оцінити ефективність мережевих та програмно-апаратних засобів захисту інформаційних потоків;

4) розробити та дослідити методи швидкодіючих алгоритмів опрацювання інформаційних потоків на основі різних ТЧБ в задачах захисту інформації;

5) дослідити особливості алгоритму Шуфа в задачах захисту інформаційних потоків з застосуванням ЕК;

6) дослідити ефективність генерування параметрів та стійкість алгоритмів шифрування інформаційних потоків на еліптичних кривих;

7) створити високопродуктивні спеціалізовані програмно-апаратні засоби опрацювання інформаційних потоків за умови застосування ЕК.

РОЗДІЛ 2
РОЗРОБКА І ДОСЛІДЖЕННЯ МЕТОДІВ ТА АЛГОРИТМІВ ОПРАЦЮВАННЯ ІНФОРМАЦІЙНИХ ПОТОКІВ У ЗАДАЧАХ ЗАХИСТУ ІНФОРМАЦІЇ

2.1Теоретичні засади виконання операцій в базисі Крестенсона

Аналіз наукових тенденцій розвитку теорії та перспективних інформаційних технологій покращення ефективності опрацювання ІП в КМ, проведений на основі новітніх публікацій та досліджених автором у розділі 1, потребує поглибленого дослідження теоретичних засад базисів Крестенсона та Галуа. Слід зауважити, що найбільш фундаментально досліджено цілочисельну форму в СЗК, яка утворюється на основі прямого перетворення ТЧБ Крестенсона.

Тому є доцільним дослідити інші форми СЗК, які можуть бути використані для реалізації високопродуктивних алгоритмів опрацювання і захисту інформаційних потоків, а також виконати порівняльний аналіз названих ТЧБ з базисом Радемахера, який породжує двійкову систему числення на основі відповідних критеріїв.

Відомо, що двійкова система числення, яка використовується в сучасних КС, має певні недоліки – наявність міжрозрядних зв’язків та велику розрядність [46]. Тому актуальним є розвиток і застосування непозиційних систем числення, в яких відсутні вказані недоліки. Прикладом може бути СЗК, або, як її ще називають, представлення чисел у базисі Крестенсона [29], [47]. Хоча вона не набула значного поширення у зв’язку з необхідністю визначення умов переповнення, складністю та громіздкістю зворотнього перетворення чисел у десяткову систему числення, а також складнощами реалізації операцій ділення та порівняння, але СЗК можна ефективно використовувати у мультибазисних процесорах, спеціалізованих обчислювальних машинах для виконання операцій додавання, віднімання та множення, наприклад, у задачах лінійної алгебри (матрично–векторні

операції) тощо. Необхідно відмітити, що ця система особливо ефективна при обчисленнях з великими числами [37], [48].

Фундаментальною основою СЗК є теорія чисел [51], [52], зокрема, властивості китайської теореми про залишки. Будь–яке ціле додатнє число N у десятковій системі числення представляється в СЗК у вигляді набору найменших додатніх залишків від ділення цього числа на фіксовані цілі додатні попарно взаємно прості числа $p_1, p_2, \dots, p_n$ ($N_{10}=(b_1, b_2, \dots, b_n)_{p1, p2, \dots, pn}$, де $b_i=N \bmod p_i$), які називаються модулями (n – кількість модулів). При цьому повинна виконуватись умова $0 \le N \le P-1$, де $P=\prod_{i=1}^{n} p_i$.

На відміну від позиційних систем числення, де величина визначеного розряду суми, різниці або множення залежить не тільки від значень відповідних, але і від попередніх розрядів доданків або множників, в СЗК додавання, віднімання та множення цілих чисел виконується окремо по кожному модулю і переноси між розрядами відсутні. Отже, такі операції в СЗК є модульними [54].

Нехай два десяткові числа A і B, записані в СЗК за вибраними модулями: $A_{10}=(a_1, a_2, \dots, a_i, \dots, a_n)_{p1, p2, \dots, pi, \dots, pn}$, $B_{10}=(b_1, b_2, \dots, b_i, \dots, b_n)_{p1, p2, \dots, pi, \dots, pn}$. Тоді:

$A_{10} \pm B_{10}=C_{10}=(c_1, c_2, \dots, c_i, \dots, c_n)_{p1, p2, \dots, pi, \dots, pn}$;

$A_{10} \times B_{10}=D_{10}=(d_1, d_2, \dots, d_i, \dots, d_n)_{p1, p2, \dots, pi, \dots, pn}$,

де $c_i= a_i \pm b_i$, $d_i= a_i \times b_i$.

Останні рівності справедливі лише в тому випадку, коли результат операції не виходить за межі інтервалу $\prod_{i=1}^{n} p_i - 1$.

Зворотнє перетворення із базису Крестенсона у десяткову систему числення є досить громіздким і грунтується на використанні китайської теореми про залишки [51]:

$$N = \left(\sum_{i=1}^{n} b_i B_i \right) mod\ P , \qquad (2.1)$$

де $B_i=M_i m_i$, $M_i = \frac{P}{p_i}$, m_i шукається з виразу $(M_i m_i) \bmod p_i = 1$, при цьому повинна виконуватись умова $\left(\sum_{i=1}^{n} B_i \right) mod\ P = 1$.

Слід зазначити, що при переведенні чисел із СЗК у десяткову систему числення значну обчислювальну складність становить пошук коефіцієнтів $m_i = M_i^{-1} mod\ p_i$. У роботі [29] було запропоновано досконалу форму СЗК (ДФ СЗК), у якій підбір модулів такий, що m_i=1, тобто

$$M_i\ mod\ p_i = 1. \tag{2.2}$$

Крім того, було підібрано декілька наборів для чотирьох та п'яти модулів ДФ СЗК. Подальший розвиток ДФ СЗК отримала у роботах [63, 64], у яких було встановлено правила побудови наборів з будь–якої кількості модулів ДФ СЗК для будь–якого діапазону десяткових чисел. Шукані модулі повинні отримуватися з такої умови:

$$\begin{cases} p_1 = 2 \\ p_i = p_1 p_2 \ldots p_{i-1} + 1,\ 1 < i < n. \\ p_n = p_1 p_2 \ldots p_{n-1} - 1. \end{cases} \tag{2.3}$$

Слід зазначити, що запропонована система не вичерпує всіх можливих наборів для базису Крестенсона при заданих n. Наприклад, при n=5 набір модулів, отриманий за допомогою системи (2.3), буде P_{51} = 2, 3, 7, 43, 1805. Однак відомі також набори P_{52} = 2, 3, 7, 83, 85 та P_{53} = 2, 3, 11, 17, 59. При n=6 набір модулів, отриманий з (2.3), буде таким: P_{61} = 2, 3, 7, 43, 1807, 3263441. Усі можливі набори модулів для ДФ СЗК базису Крестенсона при n=6, відповідні їм діапазони десяткових чисел та розрядність у двійковій системі наведені у таблиці 2.1. Як видно з таблиці 2.1, набір модулів,

отриманий за допомогою системи (2.3), найоптимальніший, оскільки в цьому випадку величина P є максимальна, що дозволяє розглядати найбільший діапазон десяткових чисел. При цьому досягається зменшення розрядності вдвічі.

Крім того, у цих роботах запропонована напівдосконала форма СЗК (m_i=±1), яку зручно використовувати у випадку обмеженої кількості модулів та необхідності розгляду великих чисел. Порівняно з ДФ СЗК, це збільшує обчислювальну складність, але вона менша, ніж при пошуку оберненого елемента $m_i = M_i^{-1} \, mod \, p_i$.

Таблиця 2.1 – Можливі набори модулів при n=6 для ДФ СЗК та відповідні їм діапазони десяткових чисел (в дужках – розрядність у двійковій системі)

№	p_1, p_2	p_3	p_4	p_5	p_6	P
1	2, 3 (2)	7 (3)	43 (6)	1807 (11)	3263441 (22)	$1{,}0650050423922\times10^{13}$ (44)
2				1811 (11)	654133 (20)	$2{,}139450562578\times10^{12}$ (41)
3				1819 (11)	252701 (18)	$8{,}30151592914\times10^{11}$ (41)
4				1825 (11)	173471 (18)	$5{,}7175174245\times10^{11}$ (40)
5				1871 (11)	51985 (16)	$1{,}7565866661\times10^{11}$ (38)
6				1901 (11)	36139 (16)	$1{,}24072631634\times10^{11}$ (37)
7				1945 (11)	25271 (15)	$8{,}876868357\times10^{10}$ (37)
8				2053 (12)	15011 (14)	$5{,}5656554898\times10^{10}$ (36)
9				2167 (12)	10841 (14)	$4{,}2427359282\times10^{10}$ (36)
10				2501 (12)	6499 (13)	$2{,}9354722194\times10^{10}$ (35)
11				3041 (12)	4447 (13)	$2{,}4423128562\times10^{10}$ (35)
12				3611 (12)	3613 (12)	$2{,}3562056658\times10^{10}$ (35)
13			47 (6)	395 (9)	779729 (20)	$6{,}0797809317\times10^{10}$ (36)
14				481 (9)	2203 (12)	$2{,}091735282\times10^{9}$ (31)
15			53 (6)	271 (9)	799 (10)	$4{,}81993554\times10^{8}$ (29)
16			71 (7)	103 (7)	61429 (16)	$1{,}8867671634\times10^{10}$ (35)
17		11 (4)	23 (5)	31 (5)	47057 (16)	$2{,}214408306\times10^{9}$ (32)

Напівдосконала форма дозволяє побудувати систему з двох модулів, що неможливо у ДФ СЗК. Для цього необхідно вибрати два будь–які

послідовні числа p_1 та $p_2=p_1+1$, які завжди будуть взаємно простими, оскільки для них виконується умова:

$$\begin{cases} (p_1+1) \bmod p_1 = 1 \\ p_1 \bmod (p_1+1) = -1. \end{cases} \tag{2.4}$$

Загальна система для визначення набору з будь–якої кількості модулів напівдосконалої форми СЗК має вигляд:

$$\begin{cases} p_2 = p_1 + 1 \\ p_i = p_1 p_2 \dots p_{i-1} \pm 1, \end{cases} \tag{2.5}$$

де $i = 3, \dots, n$.

Перспективними модифікаціями СЗК, які на даний час глибоко досліджуються науковою школою професора Я.М.Николайчука, є нормалізована та розмежована форми СЗК, описані в [65].

В роботі Возної Н.Я., Николайчука Я.М [65] отримані аналітичні вирази та графіки (рис. 1.7) алгоритмічної складності різних форм формування структуризованих даних, які підтверджують високу ефективність застосування досконалої нормалізованої форми базису Крестенсона.

2.2 Розробка високопродуктивних алгоритмів множення багато розрядних чисел у базисі Крестенсона

Однією з найважливіших операцій у всіх асиметричних алгоритмах шифрування інформаційних потоків є модулярне множення багаторозрядних чисел a та b розмірності n. Для підвищення ефективності обчислення $a \cdot b (\mathrm{mod}\ p)$ багатьма авторами запропоновано різні підходи, алгоритми, але в основному з використанням десяткової системи числення з часовою складністю $O(n^2)$, що значно збільшує час виконання програми. Одним з

підходів щодо підвищення швидкодії знаходження результату модулярного множення є метод Карабуци, але він практично не використовують через складність його реалізації. В [25] запропоновано алгоритм Шенхаге – Штрассена з використанням швидкого перетворення Фур'є і потребує O(*nlog(n)log(log(n))* бітових операцій.

Тому розробка ефективного алгоритму з використанням теоретико-числового базису Радемахера-Крестенсона для зменшення часової складності формування, збільшення швидкодії на основі матричних моделей та розробка спеціалізованих програмно-апаратних засобів є актуальною задачею.

Операції модулярного множення в базисі Радемахера лежать в основі алгоритмів електронного цифрового підпису, криптографічних протоколів, розв'язування задач обчислювальної, прикладної та дискретної математики [45]. Визначення стійкості еліптичних кривих методом пошуку їх порядку за допомогою алгоритму Шуфа [42,43] тощо. Існуючі алгоритми (стандартний, швидкого множення, Blakey, Монтгомері, бінарні і т.д.) характеризуються значною часовою складністю. Запропонований алгоритм модулярного множення в базисі Крестенсона за допомогою матричних обчислень дозволить зменшити часову складність.

Оскільки операції в комп'ютерних системах виконуються в базисі Радемахера, то постає питання в прямому переході з базису Крестенсона в базис Радемахера і навпаки, що ефективно виконується за допомогою принципової розмежованої форми системи числення в базисах Радемахера-Крестенсона [55].

Запропоновано алгоритм матрично-модулярного множення n–розрядних чисел $a=a_{n-1}2^{n-1}+\ldots+ +a_i2^i+\ldots+ a_12+a_0$ та $b=b_{n-1}2^{n-1}+\ldots+ +b_i2^j+\ldots+ +b_12+b_0$, де a_i, b_j=0, 1, n–розрядність модуля p. Для знаходження результату їх множення за модулем p побудована матриця $\left\|c_{ij}\right\| = 2^{i+j} \bmod p$ (табл. 2.2). Добуток чисел a та·b отримується згідно формули:

$$a \cdot b = \left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} a_i b_j 2^{i+j} \right) \bmod \ p \,, \qquad (2.6)$$

де a_i, $b_j=1$, тобто c_{ij} знаходиться на перетині стовбця та рядка, для яких відповідні a_i та b_j дорівнюють 1

Таблиця 2.2 – Матриця множення в базисі Радемахера–Крестенсона

	b_{n-1}	…	b_j	…	b_1	b_0
a_{n-1}	$c_{n-1\ n-1}$	…	$c_{n-1\ j}$	…	$c_{n-1\ 1}$	$c_{n-1\ 0}$
…	…	…	…	…	…	…
a_i	$c_{i\ n-1}$	…	c_{ij}	…	c_{i1}	c_{i0}
…	…	…	…	…	…	…
a_1	$c_{1\ n-1}$	…	c_{1j}	…	c_{11}	c_{10}
a_0	$c_{0\ n-1}$	…	c_{0j}	…	c_{01}	c_{00}

Модулярне множення здійснюється згідно алгоритму:

Алгоритм 2.1 – Модулярне множення в базисі Радемахера–Крестенсона.

Вхід: $a=a_{n-1}2^{n-1}+\ldots+ +a_i2^i+\ldots+ a_12+a_0$, $b=b_{n-1}2^{n-1}+\ldots+ +b_i2^j+\ldots+ b_12+b_0$, a_i, $b_j=0, 1$, n–розрядність модуля p.

1. Знаходимо залишки кожного біта a і b по заданому модулю p.
2. Сумуємо залишки останньої стрічки матриці (табл..2.2) розкладу чисел a і b по модулю p. Отримуємо c_n.
3. Помножуючи c_n*2 за модулем p і пропускаємо 0 стрічки записуємо вектор стовпчик c_i. В результаті ми отримуємо вектор залишків $a \cdot b (\bmod p)$.

Вихід c_i.

Таким чином, отримано новий алгоритм заміни операції множення, яка має квадратичну часову складність $O1(n) = n^2$ - звичайний метод множення, лінійно-логарифмічну $O(n) = n \cdot \log n \cdot \log(\log n)$ - алгоритм Шонхаге-Штрасена , або $O3(n) = n^{1,585}$, $O4(n) = n^{1,465}$ - алгоритми Каратсуби та Тома-Кука відповідно, операцією додавання зі складністю $O2(n) = \begin{cases} \log n, \textit{ якщо } n \le 64 \\ \frac{n}{2} \cdot \log n, \textit{ в інших випадках} \end{cases}$. Результати дослідження часової складності запропонованого алгоритму приведена на рис. 2.1.

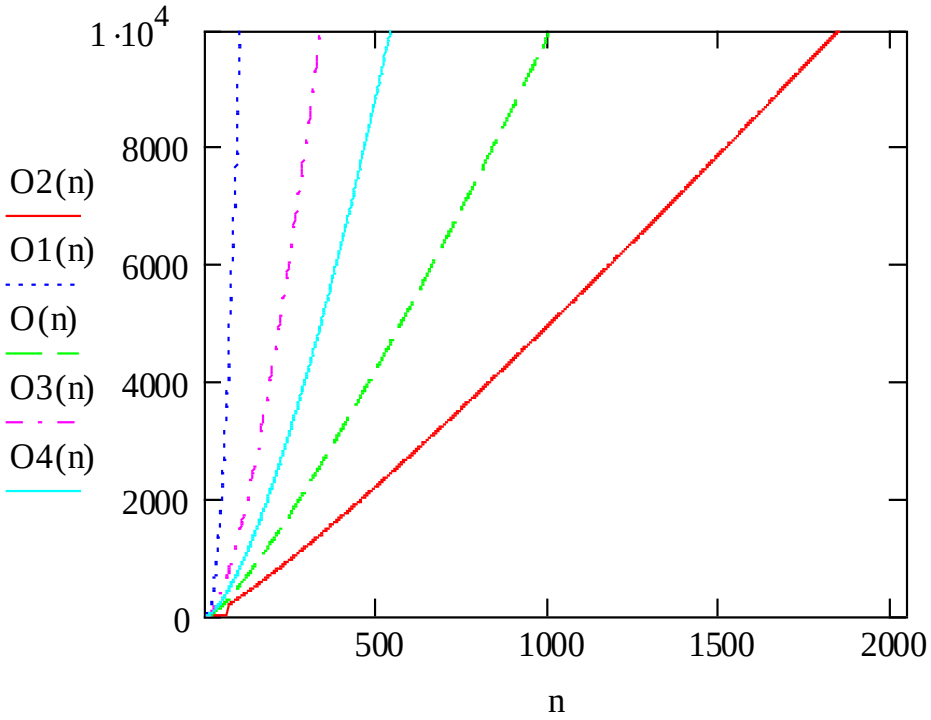


Рисунок 2.1. – Складність операції модулярного множення

Слід зазначити, що перераховані алгоритми здійснюють тільки операцію множення, а запропонований алгоритм модулярного множення шукає залишки з складністю, на один порядок меншої в діапазоні $n \le 64$ бітів і на 20% при n>64 в порівнянні з відомими алгоритмами.

З рис.2.1 видно, що при зростанні розрядності компонентів дискретного логарифма n>64 відомий алгоритм не може бути фізично реалізований з використанням сучасної мікропроцесорної техніки, особливо з

врахуванням наступної операції піднесення до степеня, яка включає операцію множення. А запропонований алгоритм з використанням базису Радемахера-Крестенсона дозволяє реалізувати можливості суттєвого підвищення захисту ІП від несанкціонованого доступу, які характеризуються ознаками незворотності, як показано в роботі Николайчука Я.М. [17], повинно бути практично нездійсненно, і поки що строго математично не доведено.

Ефективність запропонованого алгоритму модулярного множення буде $EЗ(n) = \begin{cases} n \cdot \log(\log n), \text{ якщо } n < 64 \\ \log(\log n), \text{ в інших випадках.} \end{cases}$ (рис.2.2). і дорівнює співвідношенню часових складностей.

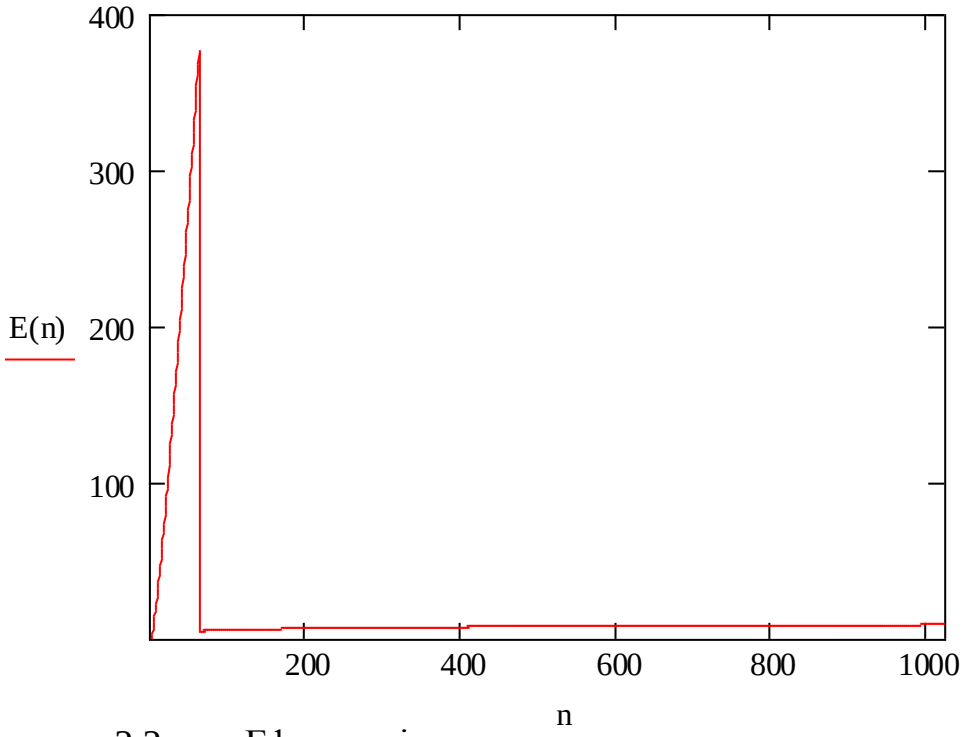


Рисунок 2.2 – Ефективність алгоритму модулярного множення розмірності n.

Результати чисельного експерименту показують, що при розмірності чисел менших 64 біт ефективність стрімко зростає за рахунок переходу від двовимірного базису Радемахера, складність операції модулярного множення в якому мають квадратичну залежність, до матрично-модульного множення в ТЧБ Радемахера-Крестенсона з логарифмічною складністю. А в діапазоні від

64 до 1024 біти складність операції множення буде лінійно-логарифмічною, тому що при реалізації запропонованого алгоритму операція додавання чисел великої розрядності більших 64 біт, яка замінює операцію множення у відомих алгоритмах, здійснюється не за один такт, як при менших розрядах, а за декілька тактів.

Отже, з використанням базису Крестенсона суттєво збільшуються перспективи щодо розробки систем з високим рівнем захисту інформаційних потоків та існує можливість пришвидшення часу виконання алгоритму едулярного множення.

2.3 Розробка і дослідження алгоритмів піднесення до високих показників степеня у розмежованій системі числення базису Крестенсона.

Для едулярного експоненіювання $a^x mod\ p$ (вважаємо, що $x \le \varphi(p)$, $\varphi(p)$ – значеня функції Ейлера від модуля p) використаємо проміжну матрицю, представлену в едул. 2.3. Її розмірність дорівнює розрядності n модуля p. В стовбцях матриці записано величини $a^{2^i} \bmod p$ в базисі Радемахера, тобто $a_{ij}=0, 1$. Тоді будь–який степінь x можна записати за степенями 2 і шуканий результат можна отримати, перемноживши відповідну кількість стовбців за допомогою едул. 2.2, та виразу:

$$a^x \bmod p = \left(\prod_{i=0}^{n-1} a^{x_i 2^i} \right) \bmod p = \prod_{k=0}^{n-1} \left(\sum_{j=0}^{n-1} \left(\sum_{i=0}^{n-1} a_i \cdot a_j \cdot 2^{i+j} \right) \bmod p \right)^{x_k 2^k} \bmod p \,. (2.7)$$

Основними перевагами такого методу є здійснення операцій в системі залишкових класів, а не з великими числами, що дозволяє пришвидшити алгоритм едулярного експоненціювання.

Таблиця 2.3–Матриця піднесення до степеня в базисі Радемахера–Крестенсона

$a_{n-1\ n-1}$	…	$a_{i\ n-1}$		$a_{1\ n-1}$	$a_{0\ n-1}$
…	…	…	…	…	…
$a_{n-1\ j}$	…	$a_{i\ j}$	…	$a_{1\ j}$	$a_{0\ j}$
…	…	…	…	…	…
$a_{n-1\ 1}$	…	$a_{i\ 1}$	…	$a_{1\ 1}$	$a_{0\ 1}$
$a_{n-1\ 0}$	…	$a_{i\ 0}$	…	$a_{1\ 0}$	$a_{0\ 0}$
$a^{2^{n-1}}$	…	$a^{2^{i}}$	…	$a^{2^{1}}$	$a^{2^{0}}$

Отже, для модулярного експоненіювання $a^x(modp)$ потрібно скористатися алгоритмом піднесення до степеня двійкового числа будь-якої розрядності за модулем p.

Алгоритм 2.2. – Модулярне експоненціювання в базисі Радемахера–Крестенсона.

Вхід: $a = a_{n-1}2^{n-1} + \ldots + + a_i 2^i + \ldots + a_1 2 + a_0, a_i = 0, 1, n$–розрядність модуля p $x \le \varphi(p)$, $\varphi(p)$ – значеня функції Ейлера від p.

1. Знаходимо залишки $a^{2^i} mod\ p$ в базисі Радемахера, $a_{ij} = 0, 1$.
2. Записуємо степінь x за степенями 2, тобто $x = \sum_{i=0}^{n-1} x_i \cdot 2^i$.

3. Знаходимо $a^x mod\ p$, перемноживши відповідну кількість стовпців за допомогою алгоритму 2.1.

Вихід: $a^x mod\ p$.

Запропонований алгоритм піднесення до степеня двійкового числа будь-якої розрядності за модулем p дозволяє зменшити часову складність за рахунок заміни операції множення додаванням, підвищити швидкодію на 2 порядки (рис.2.2), для чисел розрядності менше 64 біт, а в діапазоні від 64 біт на 1 порядок. Чисельний експеримент показав, що запропонований алгоритм піднесення до степеня двійкового числа будь-якої розрядності за модулем p в базисі Радемахера–Крестенсона дозволяє зменшити складність з

$O2 = n^3$, або $O1(n) = n^2 \log n$ (Монтгомері метод) до $O(n) = \begin{cases} n \log_2 n, якщо\ n < 64 \\ \frac{n^2}{2} \log_2 n \cdot, n \geq 64. \end{cases}$ в $E2(n) = \begin{cases} n, для\ n < 64 \\ \log_2 n\ , n \geq 64 \end{cases}$ разів, що показано на рис. 2.3.

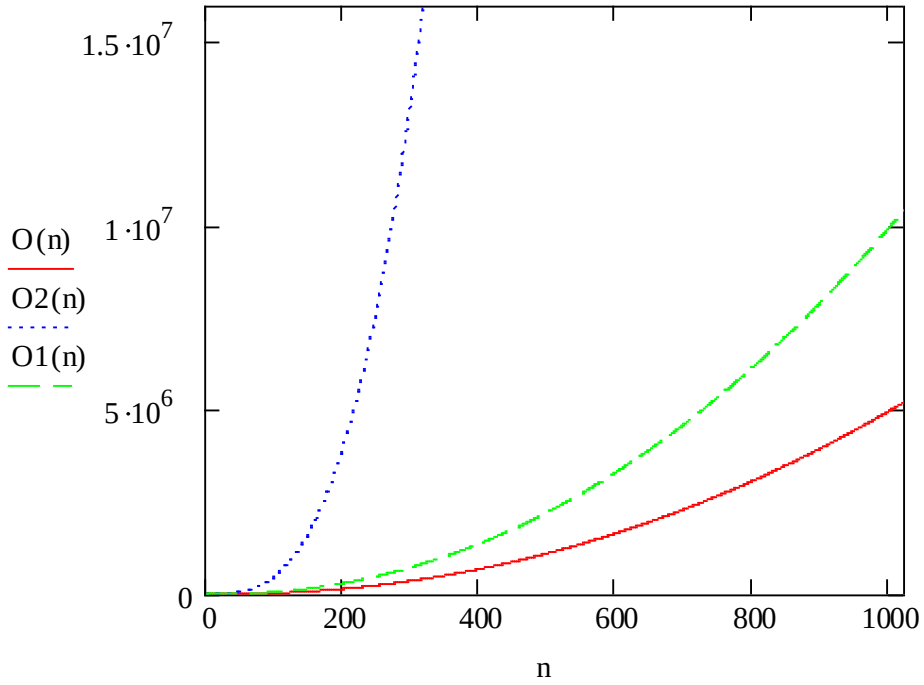


Рисунок 2.2 – Часова складність операції модулярного піднесення до степеня.

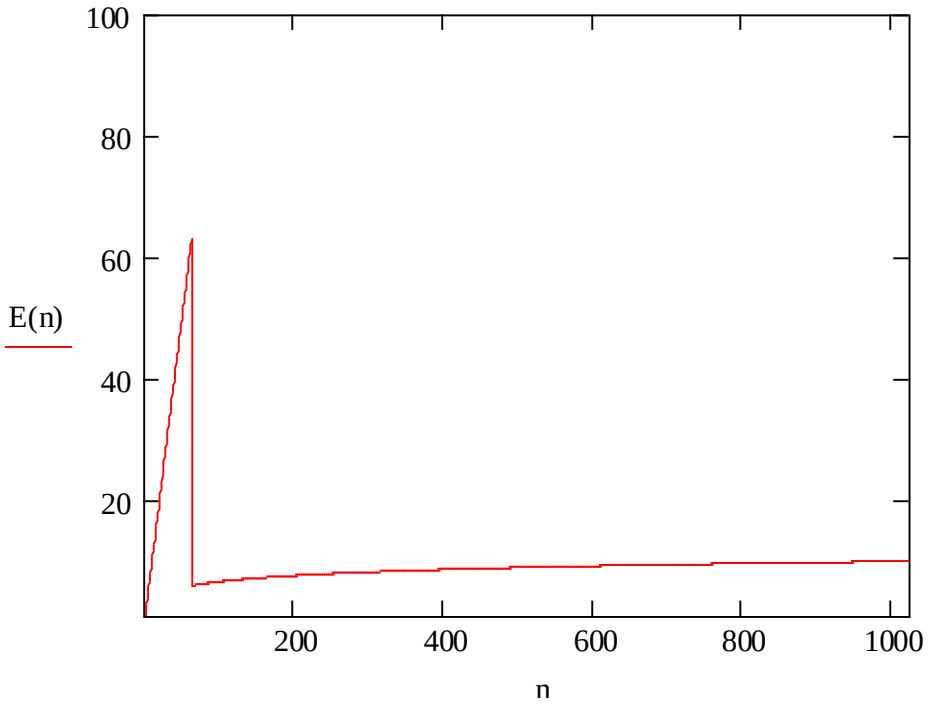


Рисунок 2.3 – Ефективність запропонованого алгоритму.

Дослідження показали, що запропонований алгоритм характеризується високою швидкодією та ефективністю для виконання операції піднесення до степеня двійкового числа будь-якої розрядності за модулем p. Слід зазначити, що при збільшенні розрядності чисел зменшується ефективність (рис 2.3), бо частина ресурсу комп'ютера буде задіяна на обробку службової інформації, що значно збільшує складність, кількість операцій та зменшує швидкодію. Оскільки операція модулярного експоненіювання є базовою в найбільш поширених системах захисту ІП з відкритими ключами (RSA, Ель–Гамаля тощо), визначення стійкості ЕК методом пошуку їх порядку за допомогою алгоритму Шуфа [2], то доцільно використовувати розроблений метод в задачах захисту ІП на практиці для вдосконалення систем захисту ІП.

2.4 Оцінка ефективності алгоритмів опрацювання інформації в задачах криптографії.

Алгоритми формування СД на основі програмно-апаратних засобів забезпечують захист від помилок та певний рівень захисту інформації від несанкціонованого доступу. Але сучасний розвиток комп'ютерної техніки потребує високого рівня захисту інформації, додаткового опрацювання ІП та формування захищених даних. Тому зроблений аналіз ефективності існуючих методів захисту СД на основі відомих алгоритмів RSA, Ель-Гамаля, а особливо алгоритмів з використанням математичного апарату ЕК, говорить, що використання ЕК є перспективним напрямком вдосконалення захисту ІП структуризованих і неструктуризованих даних.

2.4.1 Дослідження алгоритмів шифрування RSA та Ель-Гамаля в базисах Радемахера та Крестенсона

Система захисту інформаційних потоків від несанкціонованого доступу з відкритим ключем RSA базується на задачах множення і розкладу чисел на прості множники, які є обчислювально складними перетвореннями. В

системі RSA учасники процесу шифрування мають в розпорядженні відкритий і закритий (секретний) ключі, кожний з яких складається з пари цілих чисел, які генеруються наступним чином [2]:

1. Генеруються два випадкових простих числа p і q довільного розміру (чим більші, тим краще для стійкості системи захисту інформаційних потоків).

2. Обчислюється $n = p * q$, тобто модуль криптографічних перетворень з використанням матричних перетворень:

Подаємо число p і q у вигляді: $p = p_{r-1}2^{r-1} + p_{r-2}2^{r-2} + \ldots + p_1 2^1 + p_0 2^0$, $q = q_{r-1}2^{r-1} + q_{r-2}2^{r-2} + \ldots + q_1 2^1 + q_0 2^0$, де r - розрядність чисел p і q. Для знаходження результату їх множення побудуємо матрицю представлену в табл. 2.4., де $m_{ij} = 2^{i+j}$.

Добуток чисел p і q отримуємо за формулою:

$$n = p \cdot q = \sum_{s,k=1}^{r-1} m_{sk} \text{ ,} \tag{2.8}$$

де p_s, $q_k = 1$, тобто m_{sk} знаходиться на перетині стовбця та рядка, для яких відповідні p_i і q_j дорівнюють 1.

Таблиця 2.4 – Матриця знаходження модуля перетворення в базисі Радемахера–Крестенсона

	q_{r-1}	…	q_j	…	q_1	q_0
p_{r-1}	$m_{r-1\ r-1}$	…	$m_{r-1\ j}$	…	$m_{r-1\ 1}$	$m_{r-1\ 0}$
…	…	…	…	…	…	…
p_i	$m_{i\ r-1}$	…	m_{ij}	…	m_{i1}	m_{i0}
…	…	…	…	…	…	…
p_1	$m_{1\ r-1}$	…	m_{1j}	…	m_{11}	m_{10}
p_0	$m_{0\ r-1}$	…	m_{0j}	…	m_{01}	m_{00}

Це значно зменшить складність алгоритму пошуку модуля криптографічних перетворень.

3. Аналогічним чином знаходимо значення функції Ейлера від числа n, а саме $\varphi(n) = (p-1)(q-1)$.

4. Вибираємо ціле число $e, 1 < e < \varphi(n)$ - взаємнопросте з $\varphi(n)$, його доцільно вибирати з найменшою кількістю одиничних бітів у двійковій формі, рекомендуються вибирати числа Ферма 17, 257, 65537…

- Число e називають відкритою експонентою;
- час шифрування залежить від швидкодії операції піднесення числа в степінь по модулю, тому малі значення e можуть зменшити стійкість системи захисту інформаційних потоків.

4. Знаходимо секретну експоненту d, обернену до числа e за модулем $\varphi(n)$, тобто $d \cdot e \equiv 1(\mathrm{mod}\,\varphi(n))$. В класичній літературі знаходження оберненого елемента в залишкових классах обчислюється згідно розширеного алгоритму Евкліда. Для зменшення складності, доцільно скористатися матричним методом:

Розглянемо модуль $\varphi(n)$. Нехай x пробігає зведену систему найменших додатних лишків за модулем $\varphi(n)$: $r_1 = 1, r_2 = 2, r_3 = 3, \ldots, r_i = i, \ldots, r_{\varphi(n)-1} = \varphi(n) - 1$.

Тоді для числа $e < \varphi(n)$ добуток $e \cdot x$ теж пробігатиме зведену систему лишків за цим модулем:

$$\begin{cases} e \cdot r_1 \bmod \varphi(n) = c_1 \\ e \cdot r_2 \bmod \varphi(n) = c_2 \\ e \cdot r_3 \bmod \varphi(n) = c_3 \\ \ldots\ldots\ldots\ldots \\ e \cdot r_i \bmod \varphi(n) = c_i \\ \ldots\ldots\ldots\ldots \\ e \cdot r_{\varphi(n)-1} \bmod \varphi(n) = c_{\varphi(n)-1} \end{cases} . \qquad (2.9)$$

Аналогічно для числа d:

$$\begin{cases} d \cdot r_1 \bmod \varphi(n) = g_1 \\ d \cdot r_2 \bmod \varphi(n) = g_2 \\ d \cdot r_3 \bmod \varphi(n) = g_3 \\ \ldots\ldots\ldots\ldots\ldots \\ d \cdot r_i \bmod \varphi(n) = g_i \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ d \cdot r_{\varphi(n)-1} \bmod \varphi(n) = g_{\varphi(n)-1} \end{cases} . \qquad (2.10)$$

З першої системи виберемо перше рівняння, а з другого – рівняння, в якому $c_1 = r_i$:

$$\begin{aligned} & e \cdot r_1 \bmod \varphi(n) = c_1; \\ & d \cdot r_i \bmod \varphi(n) = g_i . \end{aligned} \qquad (2.11)$$

Перемножимо ці рівняння:

$$er_1 \cdot dr_i \bmod \varphi(n) = c_1 \cdot g_i . \qquad (2.12)$$

Враховуючи, що $r_1 = 1, c_1 = r_i$, отримаємо добуток за відповідним модулем:

$$e \cdot d \bmod \varphi(n) = g_i . \qquad (2.13)$$

Даним методом можна перемножувати будь–яку кількість множників, підносити до будь–якого степеня, шукати обернені елементи та квадратні корені. Для цього зручно побудувати таблицю Келі, наприклад, для $\varphi(n)$ =20, для $p = 3, q = 11$.

Таблиця 2.5.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
2	**4**	6	8	10	12	14	16	18	20	22	1	3	5	7	9	11	13	15	
3	6	**9**	12	15	18	21	1	4	7	10	13	16	19	22	2	5	8	11	
4	8	12	**16**	20	1	5	9	13	17	21	2	6	10	14	18	22	3	7	
5	10	15	20	**2**	7	12	17	22	4	9	14	19	1	6	11	16	21	3	
6	12	18	1	7	**13**	19	2	8	14	20	3	9	15	21	4	10	16	22	
7	14	21	5	12	19	**3**	10	17	1	8	15	22	6	13	20	4	11	18	
8	16	1	9	17	2	10	**18**	3	11	19	4	12	20	5	13	21	6	14	
9	18	4	13	22	8	17	3	**12**	21	7	16	2	11	20	6	15	1	10	
10	20	7	17	4	14	1	11	21	**8**	18	5	15	2	12	22	9	19	6	
11	22	10	21	9	20	8	19	7	18	**6**	17	5	16	4	15	3	14	2	
12	1	13	2	14	3	15	4	16	5	17	**6**	18	7	19	8	20	9	21	
13	3	16	6	19	9	22	12	2	15	5	18	**8**	21	11	1	14	4	17	
14	5	19	10	1	15	6	20	11	2	16	7	21	**12**	3	17	8	22	13	
15	7	22	14	6	21	13	5	20	12	4	19	11	3	**18**	10	2	17	9	
16	9	2	18	11	4	20	13	6	22	15	8	1	17	10	**3**	19	12	5	
17	11	5	22	16	10	4	21	15	9	3	20	14	8	2	19	**13**	7	1	
18	13	8	3	21	16	11	6	1	19	14	9	4	22	17	12	7	**2**	20	
19	15	11	7	3	22	18	14	10	6	2	21	17	13	9	5	1	20	**16**	

6. В результаті отримуємо пари $P = (e, n)$ - відкритий ключ, $S = (d, n)$ - таємний ключ.

Після генерування ключів шифрування ІП здійснюється наступним чином:

- Беремо $P = (e, n)$ і інформаційний потік у вигляді цілого числа $M \in D \in Z_n, 0 < M < n-1.$

- Шифрується ІП і передається по каналу зв'язку згідно співвідношення $P(M) = M^e \bmod n$, використовуючи алгоритм піднесення до степеня двійкового числа будь-якої розрядності за модулем n.

Її розмірність дорівнює розрядності *r* модуля *n*. В стовбцях матриці записано величини $M^{2^i} (\bmod n)$ в базисі Радемахера, тобто M_{ij}=0, 1 (табл.2.6). Тоді будь–який степінь e можна записати за степенями 2 і шуканий результат можна отримати, перемноживши відповідну кількість стовбців за допомогою табл. 2.2.

Таблиця 2.6 – Матриця шифротексту алгоритму шифрування RSA в базисі Радемахера–Крестенсона

$M_{r-1\ r-1}$	…	$M_{i\ r-1}$		$M_{1\ r-1}$	$M_{0\ r-1}$
…	…	…	…	…	…
$M_{r-1\ j}$	…	$M_{i\ j}$	…	$M_{1\ j}$	$M_{0\ j}$
…	…	…	…	…	…
$M_{r-1\ 1}$	…	$M_{i\ 1}$	…	$M_{1\ 1}$	$M_{0\ 1}$
$M_{r-1\ 0}$	…	$M_{i\ 0}$	…	$M_{1\ 0}$	$M_{0\ 0}$
$M^{2^{r-1}}$	…	M^{2^i}	…	M^{2^1}	M^{2^0}

Для дешифрування використовується таємний ключ $S=(d,n)$ до зашифрованного ІП $P(M)$: $S(P(M))=(P(M))^d \bmod n$.

Використання ТЧБ Радемахера-Крестенсона в алгоритмі шифрування інформаційних потоків RSA дозволяє зменшити його обчислювальну складність, яка базується на складності виконання операції $P(M)=M^e \bmod n$. Як показано в [68], обчислювальна складність цієї операції з використанням алгоритму швидкого піднесення до степеня потрібно $O(\ln e)$ операцій множень по модулю. Отже, з врахуванням того, що операція модулярного множення має складність $O(r^2)$, то складність $P(M)=M^e \bmod n$ буде $O(n^2 \cdot \ln e)$, де n - розрядність модуля n.

Оскільки в алгоритмі RSA для шифрування та дешифрування використовується одна і та ж операції, тому складність цього алгоритму оцінюється, як $O1(n^2 \cdot \ln e + 2n)$, тоді як запропонованого алгоритму з використанням ТЧБ Радемахера $O(n)=\begin{cases} \log_2 n\left(3\log_2 n+\dfrac{n}{2}\right), \textit{якщо } n<64 \\ n\cdot\left(3\log_2 n+\dfrac{n}{2}\right), \textit{в інших випадках.} \end{cases}$

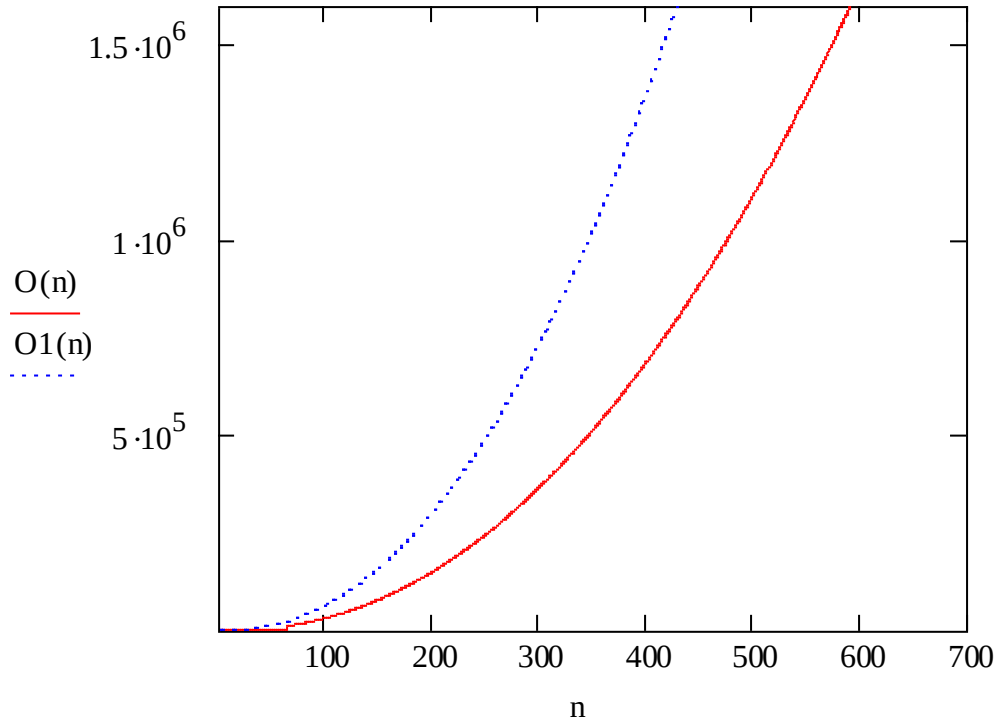


Рисунок 2.4 – Складності $O(n)$ - алгоритму шифруваня RSA з використанням розмежованої системи числення Радемахера-Крестенсона, $O1(n)$ - класичного алгоритму RSA.

Основною трудомісткою операцією при шифруванні та дешифруванні ІП в алгоритмі RSA є операція модулярного множення та експоненціювання, тому використання запропонованих алгоритмів дозволить зменшити складності на порядок для параметрів, менших 64 біт, і на 20-40 % при параметрах від 64 біт.

Для реалізації алгоритму шифрування Ель-Гамаля потрібно першочергово згенерувати ключі згідно наступної послідовності дій:

1. Генерується випадкове просте число p довжини n.
2. Вибирається довільне ціле число g, яке є первісним коренем по модулю p, та будь-яке число $x \in (1, p)$, взаємно просте з $p-1$.

Обчислюємо відкритий ключ $y = g^x mod\, p$ з використанням матричного методу піднесення до степеня в базисі Радемахера–Крестенсона описаного в пункті 2.2. Записуємо в стовпцях матриці величини $g^{2^i} (\mathrm{mod}\, n)$ в базисі Радемахера, тобто g_{ij}=0, 1(табл.2.7) та подамо степінь x за степенями 2, тобто $x = x_{n-1}2^{n-1} + \ldots + x_i 2^i + \ldots + x_1 2 + x_0$ і шуканий результат можна отримати,

перемноживши відповідну кількість стовбців за допомогою табл. 2.2. Отже, за допомогою матричного методу експоненціювання ми знаходимо відкритий ключ алгоритму шифрування Ель-Гамаля ІП.

Таблиця 2.7 – Матриця знаходження відкритого ключа алгоритму шифрування Ель -Гамаля в базисі Радемахера–Крестенсона

$g_{n-1\ n-1}$	…	$g_{i\ n-1}$		$g_{1\ n-1}$	$g_{0\ n-1}$
…	…	…	…	…	…
$g_{n-1\ j}$	…	$g_{i\ j}$	…	$g_{1\ j}$	$g_{0\ j}$
…	…	…	…	…	…
$g_{n-1\ 1}$	…	$g_{i\ 1}$	…	$g_{1\ 1}$	$g_{0\ 1}$
$g_{n-1\ 0}$	…	$g_{i\ 0}$	…	$g_{1\ 0}$	$g_{0\ 0}$
$g^{2^{n-1}}$	…	g^{2^i}	…	g^{2^1}	g^{2^0}

Відкритим ключем в алгоритмі шифрування інформаційних потоків Ель-Гамаля є трійка *(p,g,y)*, закритим ключем – число *x*. Після генерування ключів, шифрування повідомлення M здійснюється згідно алгоритму:

1. Вибирається випадково секретне число k, взаємно просте з p − 1.
2. Обчислюється $a = g^k mod\ p$, $b = y^k M mod\ p$, де M — ІП, який шифрується. Для обчислення $b = y^k M mod\ p$ потрібно послідовно застосувати алгоритм модулярного експоненціювання та модулярного множення в розмежованій системі числення Радемахера – Крестенсона.

Пара чисел *(a,b)* називається шифротекстом, причому довжина шифротексту в алгоритмі шифрування ІП Ель-Гамаля вдвоє більша від повідомлення M.

Таким чином, знаючи секретний ключ *x*, вихідне повідомлення можна обчислити з шифротексту *(a,b)* за формулою: $M = b \cdot (a^x)^{-1} \bmod p$.

Як і в алгоритмі шифрування інформаційних потоків RSA, так і в Ель-Гамаля основною операцією є модулярне експоненціювання. Тому часова складність алгоритму шифрування Ель-Гамаля, з врахуванням генерування

ключів, буде $O1(n^2(3n+1))$. Використання розмежованої системи числення Радемахера – Крестенсона дозволяє зменшити складність з $O1(n^2(3n+1))$ до

$$O3(n)=\begin{cases}\log_2 n\cdot\left(\log_2 n+\frac{n}{2}\log_2 n+1\right), \textit{якщо } n<64\\ n\cdot\left(\log_2 n+\frac{n}{2}\log_2 n+1\right), 64\le n\le 1024\end{cases}$$

, де n - розрядність модуля p (рис.2.5).

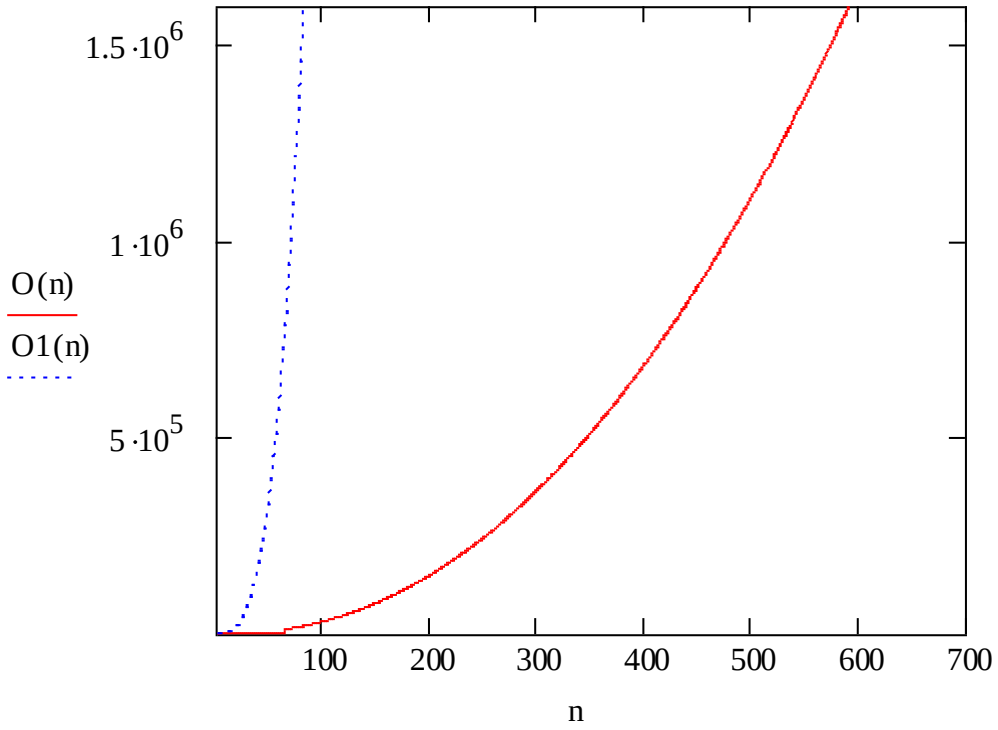


Рисунок 2.5 – Складності $O(n)$ - алгоритму шифруваня Ель-Гамаля з використанням розмежованої системи числення Радемахера-Крестенсона, $O1(n)$ - класичного алгоритму Ель-Гамаля.

Отже, використання розмежованої системи числення Радемахера – Крестенсона в задачах захисту ІП на основі алгоритмів RSA та Ель-Гамаля дозволяє ефективно застосовувати матричні методи при побудові мультибазисних процесорів шифрування, а заміна операцій множень операціями додавання на етапах генерування ключів, шифрування та дешифрування дозволяє зменшити часову складність від 20 до 40 % в залежності від розрядності параметрів алгоритмів RSA та Ель-Гамаля.

2.5 Розробка системи показників ефективності функціонування алгоритмів шифрування на еліптичних кривих та оцінки їх стійкості до атак.

В [36] наведено класифікацію показників і критеріїв ефективності функціонування схем потокового шифрування. Опираючись на підхід, запропонований в [36], а також аналізуючи сучасний стан області застосування ЕК, доцільно скористатися системним підходом для визначення оцінки стійкості та ефективності функціонування паралельних алгоритмів шифрування на ЕК .

На рисунку 2.6 зображено показники оцінки ефективності функціонування алгоритмів шифрування на ЕК, які розділені на чотири категорії:

– показники оцінки стійкості алгоритмів на ЕК;
– програмно-реалізаційні показники;
– апаратно-реалізаційні показники;
– конструктивно-технологічні показники.

Система оцінки ефективності функціонування паралельних алгоритмів шифрування на ЕК $R_{пр}$ описується наступним чином:

$$\begin{cases} R_{пр} = \alpha_1 \cdot П_{ст} + \alpha_2 \cdot П_{прр} + \alpha_3 \cdot П_{ар} + \alpha_4 \cdot П_{кт} \\ \sum_{i=1}^{4} \alpha_i = 1 \\ П_{ст} = \left\langle R_{дл} \left\langle R_{гбт}, R_{гп} \right\rangle, R_{засв} \left\langle R_{го}, R_{ро} \right\rangle \right\rangle \\ П_{прр} = \left\langle R_{овп}, R_{шші}, R_{квао} \right\rangle \\ П_{ар} = \left\langle R_{аз} \left\langle R_{ле}, R_{еп}, R_{вв} \right\rangle, R_{отч}, R_{оп} \right\rangle \\ П_{кт} = \left\langle R_{пк}, R_{ппа}, R_{п}, R_{зс} \right\rangle \end{cases} \quad (2.14)$$

де $П_{ст}$ - показники стійкості алгоритмів шифрування на ЕК, які характеризують стійкість до аналітичних методів криптоаналізу. Ці показники є домінуючими тому, що будь-яке криптографічне перетворення інформації ґрунтується насамперед, на оцінці стійкості алгоритмів шифрування до відомих на сьогодні атак, а також атак спеціального виду.

Стійкість алгоритмів шифрування на ЕК залежить від обчислювальної складності дискретного логарифму. Основними складовими показників стійкості є наступні критерії: $R_{дл}$ - показник обчислювальної складності дискретного алгоритму на ЕК [104], що залежить від: - показник обчислювальної складності генерування базової точки на ЕК $R_{гбт}$, - показник обчислювальної складності генерування параметрів ЕК $R_{гп}$ [7,8]; $R_{засв}$ - показник стійкості до атак спеціального вигляду, що залежить від: - гомогенні операції $R_{го}$, - рандомізованість обчислень $R_{ро}$ [103]; $П_{пр}$ - програмно- реалізаційні показники характеризують ефективність програмної реалізації і вказують на практичну цінність схем криптографічного перетворення. Вони включають наступні показники: - обсяг використаної пам'яті для реалізації алгоритму шифрування на ЕК $R_{овп}$, - швидкодія шифрування інформації $R_{шші}$, - кількість використаних арифметичних операцій $R_{квао}$; $П_{ар}$ - апаратно –реалізаційні показники характеризують ефективність апаратної реалізації, які вказують на практичні цінності алгоритмів шифрування включають наступні складові: $R_{оз}$ - показник оцінки апаратних затрат (кількість примітивів): $R_{ле}$ - показник кількості логічних елементів, $R_{еп}$ - показник кількості елементів пам'яті, $R_{вв}$ - показник кількості входів/виходів; $R_{отч}$ - показник тактової частоти, $R_{оп}$ - показник продуктивності; $П_{кт}$ - конструктивно-технологічні показники характеризують: $R_{пк}$ - прозорість конструкції, $R_{мппа}$ – показник можливості проведення порівняльного аналізу, $R_{п}$ - показник перспективності схем, $R_{зс}$ - показник запасу стійкості.

α_i - вагові коефіцієнти, що визначають міру впливу кожного з показників на результуючу ефективність. Вони визначаються на основі експертних оцінок в залежності від пріоритетного напрямку застосування алгоритмів шифрування в реальних прикладних задачах захисту інформаційних ресурсів.

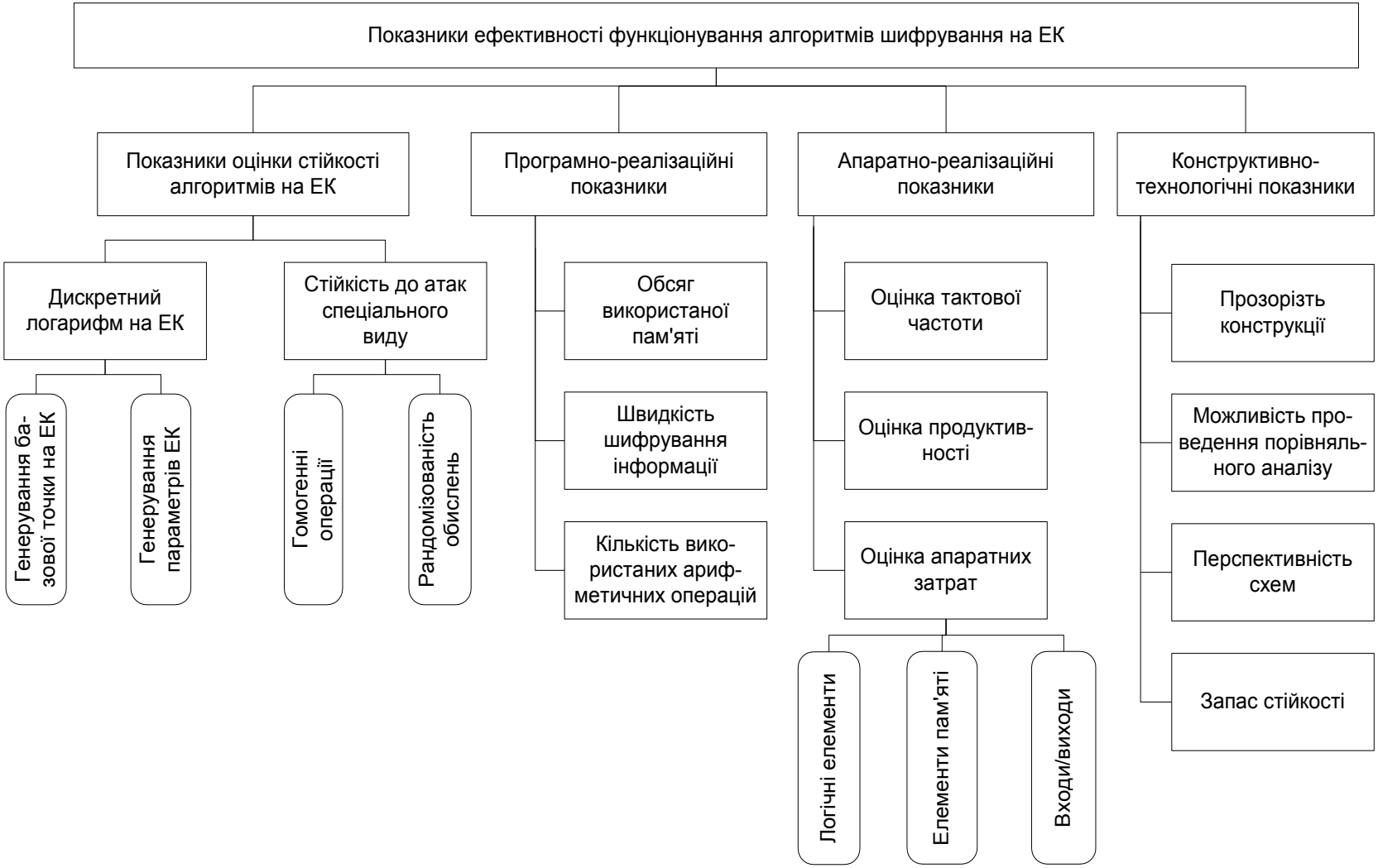


Рисунок 2.6 – Показники оцінки ефективності функціонування алгоритмів шифрування на ЕК.

На цьому етапі дослідження створюється математична модель, що визначає параметри комп'ютерної системи. Тому доцільно скористатися класифікацією атак на ЕК (рис. 2.7). Для забезпечення необхідного рівня стійкості сучасних систем захисту інформації доцільно використовувати нові методи та алгоритми опрацювання інформаційних потоків для реалізації алгоритму Шуфа, генерування параметрів ЕК. Щоб модель була адекватною, потрібно врахувати основні види впливу атак на роботу системи [23]:

1) програмні атаки – стосуються електронного-цифрового підпису;

2) математичні атаки – використовуються дискретного логарифму;

3) атаки на засоби реалізації криптоалгоритмів:

3.1) аналіз часу виконання окремих операцій;

3.2) аналіз енергоспоживання (звичайний – SPA та диференціальний – DPA);

3.3) диференційний аналіз помилок;

3.4) аналіз електромагнітного випромінювання;

3.5) аналіз радіоактивного випромінювання.

В рис. (2.7) включено критерії стійкості алгоритмів шифрування у КМ згідно до атак на ЕК. Спираючись на підхід [84], а також на результати аналізу сучасного стану застосування ЕК, слід скористатися наступними показниками оцінки стійкості:

— показники стійкості до програмних атак;

— показники стійкості до математичних атак;

— показники стійкості до апаратних атак.

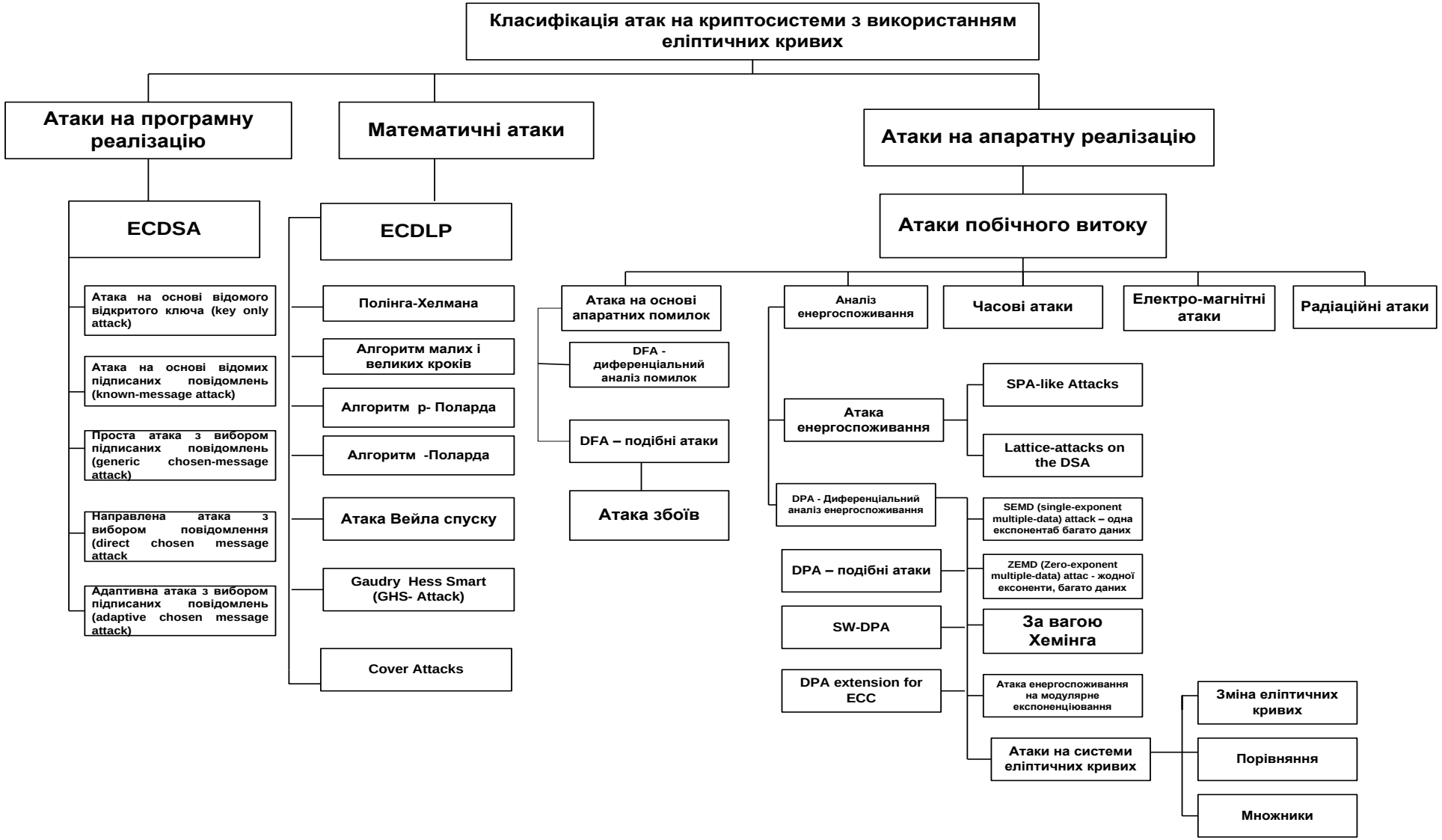


Рисунок 2.7 – Класифікація атак на ЕК.

Формалізовану модель оцінки стійкості до атак на ЕК M_{pr} створено на основі використання методу вагових коефіцієнтів з нормованими аргументами:

$$\begin{cases} M_{pr} = \alpha_1 \cdot P_{rpra} + \alpha_2 \cdot P_{rma} + \alpha_3 \cdot P_{rha} \\ \sum_{i=1}^{3} \alpha_i = 1 \\ P_{rpra} = \left\langle M_{dsa} \left\langle M_{koa,} M_{kma,} M_{gcma,} M_{dcma,} M_{acma} \right\rangle \right\rangle \\ P_{rma} = \left\langle M_{dlp} \left\langle M_{ph,} M_{bgs,} M_{\rho-P,} M_{\lambda-P,} M_{wa,} M_{ghsa,} M_{ca} \right\rangle \right\rangle \\ P_{rha} = \left\langle M_{sca} \left\langle M_{fa} \left\langle M_{dfa,} M_{dfa-like} \left\langle M_{ga} \right\rangle \right\rangle, M_{pa} \left\langle M_{spa,} M_{dpa} \right\rangle, M_{ta,} M_{ema,} M_{ra} \right\rangle \right\rangle \end{cases}, \quad (2.15)$$

де P_{rpra} —показник стійкості алгоритмів шифрування на ЕК до програмних атак.

Основними складовими показника стійкості є :

– M_{dsa} — показник стійкості, зумовлений атакою на ЕЦП;

– M_{dsa} M_{koa} — показник стійкості, зумовлений атакою на основі відкритого ключа;

– M_{dsa} M_{kma} — показник стійкості, зумовлений атакою на основі підписаних повідомлень;

– M_{dsa} M_{gcma} — показник стійкості, зумовлений простою атакою з вибором підписаних повідомлень;

– M_{dsa} M_{dcma} — показник стійкості, зумовлений направленою атакою з вибором повідомлень;

– M_{dsa} M_{acma} — показник стійкості, зумовлений адаптивною атакою з вибором підписаних повідомлень;

– P_{rma} — показник стійкості до математичних атак.

– Він включає наступні критерії:

– M_{dlp} — показник стійкості, зумовлений атакою на дискретний логарифм;

– M_{ph} — показник стійкості, зумовлений атакою з використанням алгоритму Полінга-Хелмана;

– M_{bgs} — показник стійкості, зумовлений атакою з використанням алгоритму малих і великих кроків;

– $M_{\rho-P}$ — показник стійкості, зумовлений атакою з використанням алгоритму ρ-Поларда;

– $M_{\rho-P}$ $M_{\lambda-P}$ — показник стійкості, зумовлений атакою з використанням алгоритму λ-Поларда;

– $M_{\rho-P}$ M_{wa} — показник стійкості, зумовлений атакою спуску Вейла; M_{ghsa} — показник стійкості, зумовлений атакою Gaudry Hess Smart;

– $M_{\rho-P}$ M_{ca} — показник стійкості, зумовлений cover атакою.

– P_{rha} — показник стійкості до апаратних атак.

– Включає наступні критерії:

– M_{sca} — показник стійкості, зумовлений атакою побічного витоку;

– M_{fa} — показник стійкості, зумовлений атакою на основі апаратних помилок;

– M_{dfa} — показник стійкості, зумовлений атакою на основі диференціального аналізу помилок, $M_{dfa-like}$ — DFA-подібні атаки, M_{ga} — показник стійкості, зумовлений атакою збоїв, M_{pa} — показник стійкості, зумовлений атакою енергоспоживання, M_{spa} — показник стійкості, зумовлений простою атакою енергоспоживання, M_{dpa} — показник стійкості, зумовлений атакою диференціального аналізу енергоспоживання, M_{ta} — часова атака, M_{ema} — показник стійкості, зумовлений електро-магнітною атакою, M_{ra} — показник стійкості, зумовлений радіаційною атакою. α_i — вагові коефіцієнти, що визначають міру впливу кожного з показників на результуюче значення стійкості алгоритму шифрування.

Запропонована класифікація сучасних методів криптоаналізу та формалізована модель дозволяють ефективно оцінювати рівень захисту ІП в КС [105, 106].

2.5.1 Визначення стійкості та продуктивності алгоритмів експоненціювання точок на еліптичних кривих

Найпоширенішою операцією у всіх мережних криптографічних алгоритмах на ЕК є k - кратне додавання точки P, яке позначається kP. Загалом цю операцію можна віднести до скалярного множення, або, звертаючись до термінології мультиплікативної групи, експоненціювання точки ЕК [81].

Для підвищення продуктивності обчислення точки kP багатьма авторами запропоновані різні методи [26]. Підхід до знаходження точки kP може відрізнятися залежно від того, чи є точка P заздалегідь відомою чи невідомою, а також її випадковість. У першому випадку завжди можна користуватися попереднім обчисленням точок, наприклад, $2^i P$, які зберігаються в пам'яті. Двійкове подання числа k дозволяє вибрати його з отриманого набору чисел, які в результаті підсумовування утворять точку kP. У другому, більш загальному випадку, всі обчислення доводиться проводити у реальному часі.

Нехай порядок Ord P=r, $[\log_2 r]=L$ і число k подане у двійковій системі:

$$k = \sum_{i=1}^{L-1} k_i 2^i \ . \qquad (2.16)$$

Розглянемо спочатку основні алгоритми експоненціювання для невідомої заздалегідь точки P.

Згідно з найпростішим методом, обчислення здійснюються за формулою [26] :

$$kP = \sum_{i=1}^{L-1} k_i 2^i P = \sum_{i=1}^{L-1} k_i P_i \ , \qquad P_i = 2^i P \ . \qquad (2.17)$$

Тоді можна застосувати розрахунок зліва направо за допомогою одного

з двох нижче наведених алгоритмів [26].

Алгоритм 1.

Вхід: $k=(k_{L-1}, k_{L-2},..., k_0)$, $P \in E$.

Вихід: kP.

1. Присвоюємо початкове значення R=0.

2. for i=L-1 to 0 do

2.1. P←2P.

2.2. if k_i=1 then R←R+P.

3. return R.

Реалізація методу вимагає (L-1) операцій D подвоєння точки й W(k)-1 додавань A, де W(k) - вага Хемінга двійкового вектора, k - кількість одиниць цього вектора. Беручи до уваги, що в середньому кількість одиниць випадкового вектора k дорівнює 1/2, то загальна кількість групових операцій оцінюється значенням 0,5LA+LD.

Цей алгоритм пропонується вдосконалити, якщо ввести додаткову операцію - віднімання точки. Нехай число k=31 у двійковій системі має вагу W(k)=5. Тоді його можна подати у вигляді 2^5-1 з вагою 2. При цьому використано підхід до зниження ваги Хемінга (і, відповідно, кількості групових операцій), що реалізується переходом від двійкового подання числа k до потрійного NAF(k) з коефіцієнтами $k \in \{0,1,-1\}$ (NAF - non-adjacent form). Одна з суттєвих переваг подання NAF(k) полягає у відсутності суміжних пар ненульових елементів, завдяки чому зростає питома вага нульових елементів k_i.

Для розрахунку NAF(k) використовується додатковий проміжний алгоритм.

Проміжний алгоритм.

Вхід: позитивне ціле число k.

Вихід: NAF(k).

1. i←0.

2. while $k \geq 1$ do

2.1. if k is odd then: $k_i \leftarrow 2-(k \bmod 4)$, $k \leftarrow k-k_i$;

2.2. else $k_i \leftarrow 0$;

2.3. $k_i \leftarrow k/2, i \leftarrow i+1$.

3. return $(k_0, k_1, \ldots, k_{L-1})$.

Після розрахунку NAF(k) обчислюється точка kP методом зліва на право за допомогою алгоритму 2.

Алгоритм 2.

Вхід: NAF(k), $P \in E$.

Вихід: kP.

1. $R \leftarrow O$.

2. for i=L-1 to 0 do

2.1. $P \leftarrow 2P$.

2.2. if $k_i=1$ then $R \leftarrow R+P$.

2.3. if $k_i=-1$ then $R \leftarrow R-P$.

3. return R.

NAF-подання числа k може виявитися на один біт більше двійкового. У той же час для випадкового k імовірність появи ненульових елементів 1 та -1 знижується від 1/2 до 1/3, тобто в середньому для L-розрядного числа їхня кількість оцінюється значенням L/3. Тоді загальну середню кількість групових операцій додавання A та подвоєння D в алгоритмі 2 можна оцінити сумою (L/3)A + LD.

До однієї з найважливіших характеристик алгоритму належить швидкодія. Як відомо, для виконання різних операцій процесор витрачає різний час. В роботах Черкаського М.В. запропоновано п’ять характеристик складності алгоритмів для аналізу, синтезу і оптимізації Software/Hardware(SH) моделей, а саме: апаратна, часова, ємнісна, програмна та структурна табл.2.8.

Таблиця 2.8 – Характеристики складності SH – моделей

№ п/п	Характеристика складностей SH – моделей	Аналітичне представлення складностей SH – моделей
1.	Апаратна складність	$A = \lvert X \rvert$
2.	Часова складність	$L = \lvert \max X_i \rvert$
3.	Програмна складність	$P = -F \log_2 \frac{F}{n \cdot m}$
4.	Структурна складність	$S = -E \log_2 \frac{E}{r(r-1)}$

де, X – множина елементів схеми,

$F = \sum_{L} f_l$;

n - кількість входів управління;

m - кількість дискретів часу (часової діаграми);

f_l - кількість сигналів управління l-того фрагменту часової діаграми для вибраного рівня ієрархії побудови апаратних засобів;

L - кількість фрагментів часової діаграми, конфігурації яких не повторюються;

E – кількість елементів матриці інцидентності системи;

R – розмір матриці.

Згідно ємнісного критерію, запропонованого Черкаським М.В., ємнісна складність визначається необхідним об’ємом пам’яті для реалізації алгоритму. Аналіз типових програмних продуктів, які призначені для опрацювання ІП даних, показує, що загальний об’єм пам’яті згідно експертних оцінок може бути виражений аналітично через адитивну функцію:

$$M_з = \sum_{i=1}^{k} M_i \,, \qquad (2.18)$$

де M_i - об'єм пам'яті, яка виділяється для реалізації компонентів та окремих модулів алгоритму.

Згідно систематизації: M_1 - об'єм вхідних даних, яке визначається числом вхідних даних та їхньою розрядністю:

$$M_1 = N_1 \cdot n_1, \tag{2.19}$$

M_2 - об'єм пам'яті необхідний для зберігання матриць проміжних результатів:

$$M_2 = N_2 \cdot n_2, \tag{2.20}$$

де N_2 - розмірність матриці,

n_2 - розрядність компонентів матриці.

Аналогічно можна оцінити ємнісну складність інших компонентів алгоритму, включаючи вихідні дані, які можна представити у вигляді діалогових та телекомунікаційних фреймів.

Таким чином, в загальному випадку оцінку ємності алгоритму можна представити у вигляді наступного аналітичного виразу:

$$M_з = \sum_{i=1}^{k} \prod_{l=1}^{m} M_{il} \cdot n_{il}. \tag{2.21}$$

Крім цього, час виконання будь-якого алгоритму можна подати через суму інтервалів часу, що витрачаються на виконання кожної операції розглянутих алгоритмів. Так, подвоєння-додавання в алгоритмі 1 та подвоєння-додавання-віднімання в алгоритмі 2 супроводжуються наведеними в табл. 2.9 витратами часу на виконання кожної з основних операцій [69].

Таблиця 2.9 – Витрати часу на виконання основних операцій алгоритмів експоненціювання точки кривої

Зміст операції	Витрачений час (такти)
присвоєння ($a = b$)	t_1
присвоєння за модулем ($a = b \bmod c$)	t_2
переведення числа в бінарну систему числення ($k = (k_{L-1}, k_{L-2}, ..., k_0)$)	t_3
додавання ($a = b + c$)	t_4
множення ($a = b \cdot c$)	t_5
ділення ($a = \frac{b}{c}$)	t_6

Загалом можна прийняти таке співвідношення між цими часами:

$$t_1 \le t_2 \le t_3 \le t_4 \le t_5 \le t_6. \qquad (2.22)$$

На підставі наведених у табл. 2.9 даних та співвідношення (2.22) можна побудувати математичні моделі загального часу виконання кожного алгоритму.

Для пошуку кратної точки на основі розрахунку зліва направо згідно з алгоритмом 1 використано бінарне зображення k. Тоді відбувається побітове зчитування послідовності $k = (k_{L-1}, k_{L-2}, ..., k_0)$, де L - довжина даної послідовності. Крім цього, в даному алгоритмі на кроці 2.1 виконується операція подвоєння точки. Необхідно зазначити, що при додаванні двох точок $P(x_1, y_1)$ та $Q(x_2, y_2)$ координати суми визначаються так:

$$x = \left(\frac{y_1 - y_2}{x_1 - x_2} \right) - x_1 - x_2, \qquad (2.23)$$

$$y = -y_1 + \left(\frac{y_1 - y_2}{x_1 - x_2} \right) \cdot \left(x_1 - x_2 \right).$$

Беручи до уваги, що операція віднімання вимагає стільки ж часу, що і додавання, то час, витрачений на виконання операції додавання двох точок, можна подати у вигляді [69]:

$$t_0 = t_4 + t_4 + t_6 + t_4 + t_4 + t_4 + t_4 + t_6 + t_4 + t_5 + t_4 = 8 \cdot t_4 + t_5 + 2 \cdot t_6 . \quad (2.24)$$

Звідси отримуємо, що для виконання алгоритму 1 потрібно витратити час [69]:

$$T_1(k) = t_3 + t_1 + \sum_{i=L-1}^{0} t_0 + \sum_{\substack{i=L-1 \\ k_i=1}}^{0} t_0 = t_3 + t_1 + L \cdot t_0 + W(k) \cdot t_0, \quad (2.25)$$

де $W(k)$ - вага Хемінга двійкового вектора k.

Оскільки L - довжина бінарної послідовності $k = (k_{L-1}, k_{L-2}, ..., k_0)$, то $L = \lceil \log_2 k \rceil$. Крім цього, в загальному випадку можна вважати, що вага Хемінга $W(k) = \frac{L}{2}$ для найсприятливішого випадку для проведення атаки. Звідси та зі співвідношень (2.22) і (2.23) випливає [69]:

$$\begin{aligned} T_1(k) &= t_3 + t_1 + \lceil \log_2 k \rceil \cdot t_0 + \frac{\lceil \log_2 k \rceil}{2} t_0 = t_3 + t_1 + \frac{3\lceil \log_2 k \rceil}{2} t_0 = \\ &= t_3 + t_1 + \frac{3}{2} \lceil \log_2 k \rceil \cdot (8t_4 + t_5 + 2t_6) \end{aligned} . \quad (2.26)$$

Для алгоритму 2 подвоєння-додавання-віднімання використовується потрійне подання числа k NAF(k) з коефіцієнтами $k \in \{0,1,-1\}$. Перехід від бінарного до NAF-подання здійснюється за проміжним алгоритмом. Додатково в проміжному алгоритмі на кроці 2.1 перевіряється парність числа k. Тому загальний час виконання даного алгоритму буде залежати від k. Витрачений на NAF-подання час можна обчислити так [69]:

$$T_2(k) = t_1 + \sum_{\substack{i=k \\ k_i \neq 2}}^{1}(t_2 + t_4 + t_4) + \sum_{\substack{i=k \\ k_i \div 2}}^{1} t_1 + t_6 + t_4 =$$

$$= t_1 + \frac{k}{2}(t_2 + 2t_4) + \frac{k}{2}t_1 + t_6 + t_4 = \quad . \qquad (2.27)$$

$$= (\frac{k}{2} + 1)t_1 + \frac{k}{2}t_2 + (k+1)t_4 + t_6$$

На підставі співвідношень (2.22) та (2.25), а також імовірності появи ненульових елементів у $NAF(k)$, час, витрачений на виконання алгоритму подвоєння-додавання-віднімання (алгоритм 2), такий [69]:

$$T_3(k) = T_2(k) + t_1 + \sum_{i=L-1}^{0} t_0 + \sum_{\substack{i=L_{NAF}-1 \\ k_i=1}}^{0} t_0 + \sum_{\substack{i=L_{NAF}-1 \\ k_i=-1}}^{0} t_0 =$$

$$= t_1 + \frac{k}{2}(t_2 + 2t_4) + \frac{k}{2}t_1 + t_4 + t_6 + t_1 + L_{NAF} \cdot t_0 + \frac{L_{NAF}}{3}t_0 + \frac{L_{NAF}}{3}t_0 = \quad , \qquad (2.28)$$

$$= \left(\frac{k}{2} + 2\right) \cdot t_1 + \frac{k}{2} \cdot t_2 + \left(k + 1 + \frac{40}{3}L_{NAF}\right) \cdot t_4 + \frac{5}{3}L_{NAF} \cdot t_5 +$$

$$+ \left(\frac{10}{3}L_{NAF} + 1\right) \cdot t_6$$

де L_{NAF} - довжина $NAF(k)$.

З аналізу співвідношень (2.24) та (2.26) випливає, що загальний час виконання алгоритмів 1 та 2 залежить лише від значення числа k. Тому можна оцінити продуктивні характеристики алгоритмів, що дозволяє визначити кращий з них для застосування з точки зору швидкодії.

На рис. 2.8 зображено залежність T_1 та T_3 від k для довжини бітової послідовності $L = 1..256$. Слід зазначити, що для даної оцінки використовуються значення часів $t_1 = 1$, $t_2 = 1{,}5$, $t_3 = 1{,}6$, $t_4 = 1{,}8$, $t_5 = 10$, $t_6 = 12$ (в тактах).

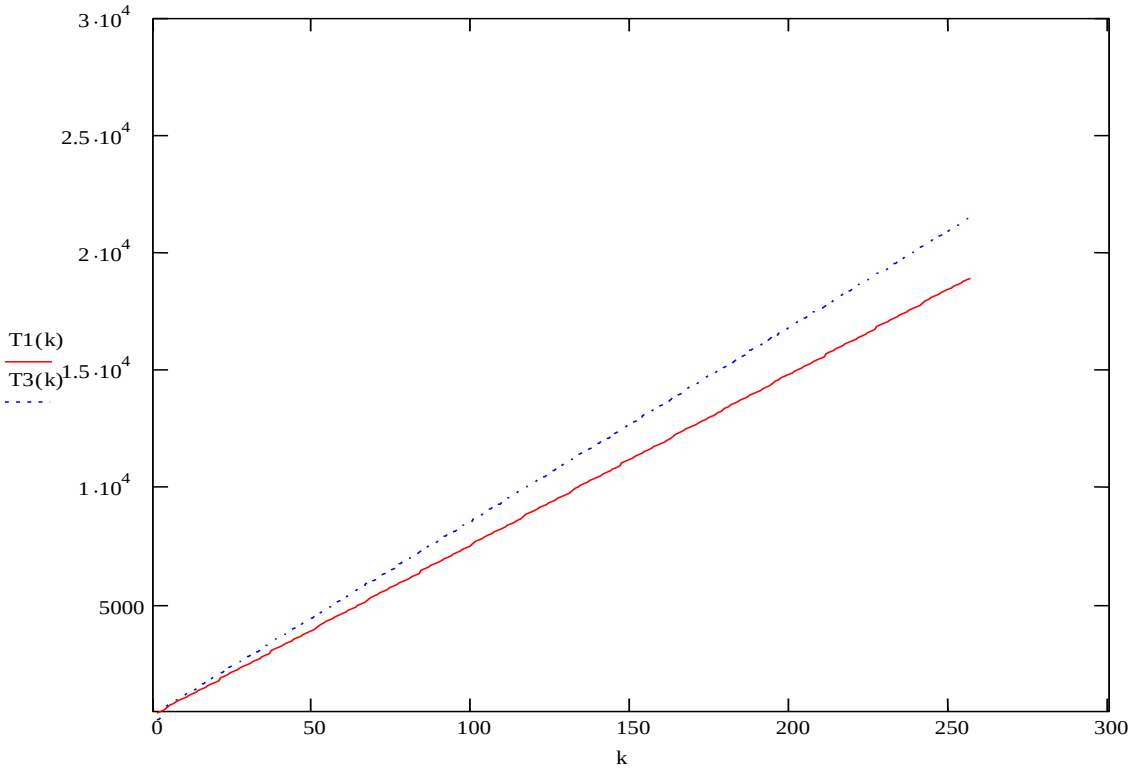


Рисунок 2.8 – Оцінка продуктивності досліджуваних алгоритмів 1 та 2

Аналіз наведених на рис. 2.8 залежностей показує, що вища продуктивність притаманна алгоритму 1. Проте і продуктивність алгоритму подвоєння-додавання-віднімання (алгоритм 2) цілком задовольняє вимогам сучасного користувача.

Час виконання криптографічних операцій залежить не лише від ефективності реалізації конкретного алгоритму, але й також, інколи суттєво, від вхідних даних. Особливо така кореляція сильно проявляється для додавання чи множення точок на ЕК. Загалом, ці криптографічні операції є обчислювально складними і для підвищення продуктивності виконання процедури шифрування повідомлення чи формування ЕЦП доцільно застосувати спеціальні алгоритми, які базуються на використанні інформації про кількість бітів у ключі шифрування. Такий підхід дозволяє пришвидшити виконання криптографічних операцій завдяки обходу виконання деяких операцій алгоритму для нульових бітів ключа. Звідси очевидно, що завжди можна виявити певну кореляцію між кількістю одиничних бітів ключа та часом виконання такого алгоритму. Володіння криптоаналітиком саме такою інформацією дозволяє висунути гіпотезу щодо кількості одиничних та

нульових бітів у приватному ключі, кількісним еквівалентом якого може бути вага Хемінга. В подальшому це дозволяє здійснити атаку повного перебору в певному піддіапазоні ключового простору, для чого вимагаються значно менші обчислювальні ресурси. Таким чином, оцінка кореляції часу виконання криптографічних операцій з вагою Хемінга щодо ключа дозволяє криптоаналітику зменшити обчислювальну складність атаки часового аналізу на систему захисту інформаційних ресурсів.

Отже, для дослідження стійкості алгоритмів 1 та 2 необхідно встановити залежність часу виконання відповідного алгоритму від ваги Хемінга. Тоді слід задати залежність згаданого часу від ваги Хемінга.

З рівності (2.23) та аналізу алгоритму 1 випливає, що:

$$T_1(k) = t_3 + t_1 + \left(\log_2 k + W(k)\right)\cdot\left(t_4 + t_5 + 2t_6\right). \qquad (2.29)$$

На основі результатів проведеного моделювання отримано залежність часу виконання алгоритму 1 від ваги Хемінга (рисунок 2.9) відповідно для довжин ключа $L = 64$ біт, $L = 128$ біт та $L = 256$ біт.

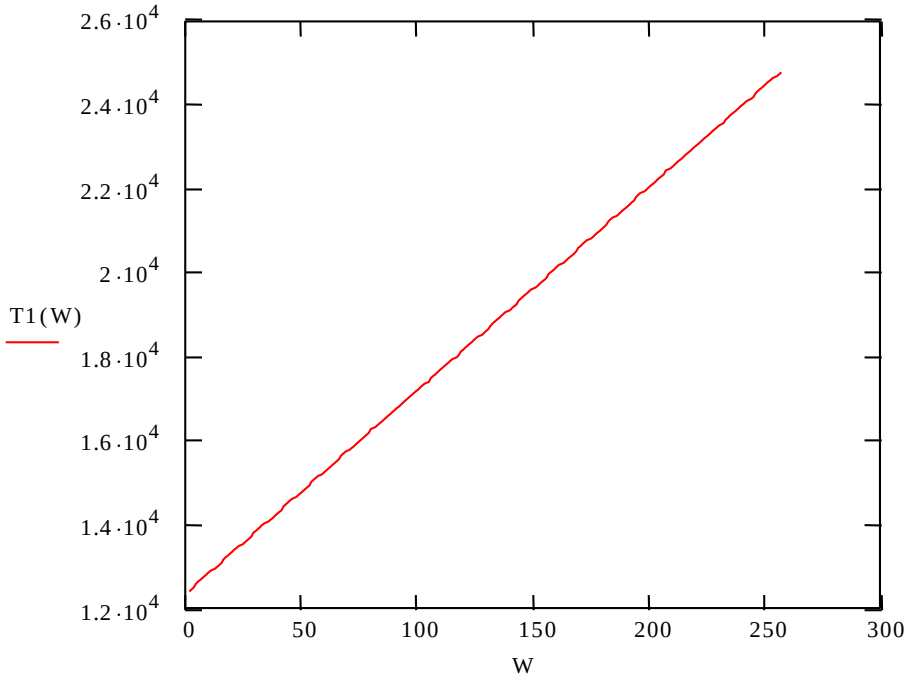


Рисунок 2.9 – Залежність часу виконання алгоритму подвоєння-додавання від ваги Хемінга для $L = 256$ біт

З аналізу зображеної на рис. 2.9 залежності випливає, що алгоритм подвоєння-додавання (алгоритм 1) характеризується лінійною залежністю часу від ваги Хемінга. Це означає, що вимірявши час здійснення множення точки ЕК, криптоаналітик може знайти вагу Хемінга – а цього є достатньо для знаходження значення k.

Залежність часу виконання алгоритму 2 від ваги Хемінга пропонується задати таким чином [66]:

$$T_3(k) = \left(\frac{k}{2} + 2\right) \cdot t_1 + \frac{k}{2} \cdot t_2 + \left(\frac{k}{2} + 1\right) \cdot t_4 + L_{NAF} \cdot t_0 + W_1(k) \cdot t_0 + W_{-1}(k) \cdot t_0, \quad (2.30)$$

де $W_1(k)$ - кількість одиниць у $NAF(k)$-поданні, а $W_{-1}(k)$ - кількість одиниць у даній рівності. Враховуючи імовірність появи нулів у $NAF(k)$, що дорівнює $\frac{1}{3} L_{NAF}$, можна в загальному вважати, що:

$$W_1(k) = \frac{2}{3} L_{NAF} - W_{-1}(k). \quad (2.31)$$

На підставі співвідношень (2.29) та (2.30) можна побудувати графік залежності часу виконання алгоритму подвоєння-додавання-віднімання від кількості одиниць у $NAF(k)$, відповідно для $L_{NAF} = 64$ біт, $L_{NAF} = 128$ біт та $L_{NAF} = 256$ біт (рис. 2.10).

Слід зазначити, що кількість одиниць у $NAF(k)$, аналогічно як і вага Хемінга в бінарному поданні числа k, дає достатньо інформації криптоаналітику щодо заповнення $NAF(k)$, тобто дозволяє правильно визначити значення k. Звідси та з аналізу залежностей, наведених на рис. 2.10, можна зробити висновок, що алгоритм подвоєння-додавання-віднімання (алгоритм 2) є абсолютно стійким до часового аналізу.

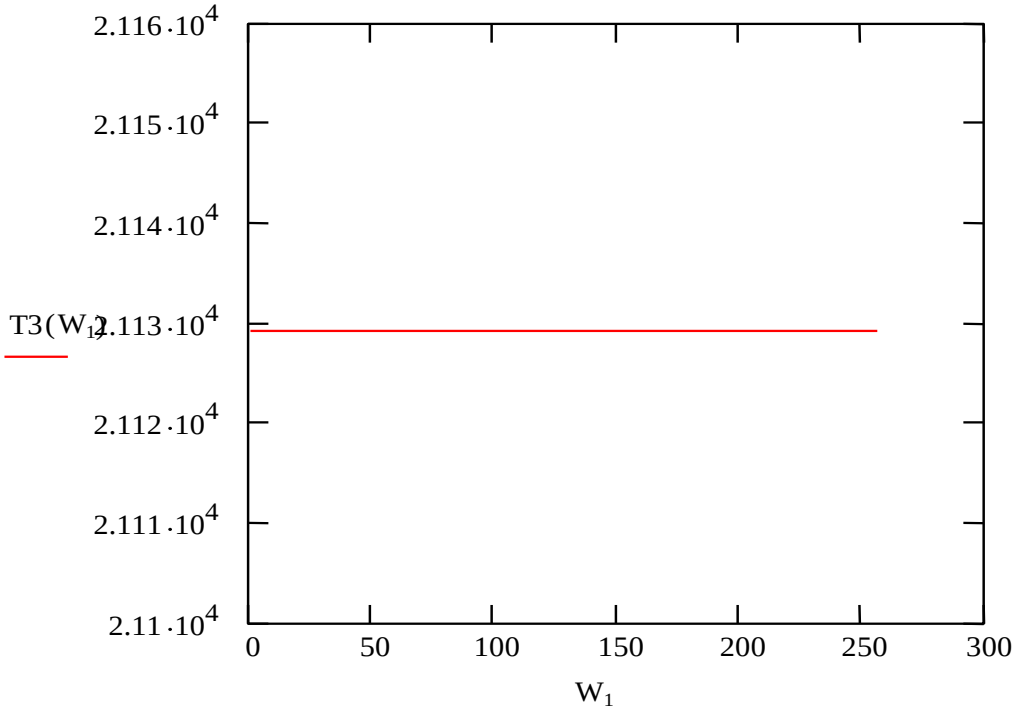


Рисунок 2.10 – Залежність часу виконання алгоритму подвоєння-додавання-віднімання від кількості одиниць при $L_{NAF} = 256$ біт

Отже, враховуючи продуктивність та стійкість алгоритму 2 до часового аналізу, рекомендується застосовувати його на практиці для скалярного множення базових точок ЕК.

ВИСНОВКИ РОЗДІЛУ 2

1. Розроблено метод модулярного множення та експоненціювання з використанням матрично-модульних перетворень ТЧБ Радемахера-Крестенсона, який, на відміну від відомих, дозволяє зменшити складність з поліноміально-експоненційної до логарифмічної або лінійно-логарифмічної, що на 1-2 порядки збільшило швидкодію рішення задач даного класу.

2. Розроблені алгоритми захисту ІП на основі RSA і Ель-Гамала застосування запропонованих алгоритмів з використанням ТЧБ Радемахера-Крестенсона, які, на відміну від існуючих, дали змогу зменшити складність більше ніж на 1 порядок, та збільшити рівень захисту ІП відповідним збільшенням розрядностей базових параметрів існуючих криптосистем.

3. На підставі алгоритмів експоненціювання точок на ЕК побудовано математичні моделі часу виконання кожного з методів розглянутих алгоритмів експоненціювання точок на ЕК, що дало змогу провести детальний аналіз їх продуктивності та стійкості до атак спеціального виду, зокрема часового аналізу.

4. На основі аналізу запропонованих моделей досліджено продуктивність кожного з методів експоненціювання точок на ЕК, що дало змогу виявити алгоритм з найбільшим рівнем продуктивності. Встановлено, що найвищу швидкодію має бінарний алгоритм експоненціювання точки на ЕК, яка на 7% перевищує швидкодію існуючих алгоритмів.

5. Проведено аналіз залежності часу виконання криптографічних операцій від значення розмірності ключа та досліджено залежність швидкодії кожного методу від ваги Хемінга, причому доведено, що для певних заданих умов найвищу стійкість системи забезпечує алгоритм подвоєння-додавання-віднімання, завдяки чому можна обгрунтувати вибір алгоритму та його параметрів для реалізації засобів криптографічного захисту інформації, стійких до атак спеціальних впливів.

6. Досліджено значення нормованої стійкості та швидкодії бінарного алгоритму і алгоритму подвоєння-віднімання-додавання для заданих

параметрів, що можуть бути використанні при побудові сучасних високопродуктивних засобів захисту інформації, стійких до атак спеціального виду, які базуються на асиметричних криптографічних алгоритмах.

РОЗДІЛ 3
ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ РЕАЛІЗАЦІЇ АЛГОРИТМУ ШУФА НА БАЗІ МЕРЕЖЕВИХ ЗАСОБІВ ТА МАТРИЧНО-МОДУЛЬНИХ СПЕЦПРОЦЕСОРІВ.

3.1. Ефективні алгоритми генерування параметрів та точок еліптичних кривих в комп'ютерних мережах.

3.1.1 Еволютивні алгоритми генерування параметрів ЕК.

Незважаючи на вагомі переваги застосування ЕК, поряд з цим існують певні завдання та труднощі. Зокрема, виділяють такі класи задач[8]:

1) генерування параметрів ЕК;
2) обчислення точок ЕК;
3) задача дискретного логарифма.

Задачею генерування параметрів ЕК вигляду (1.10) є знаходження таких параметрів:

a) просте число p – модуль перетворення груп точок ЕК. Просте число p повинно задовольняти нерівність $p > 2^{255}$. Верхня границя числа визначається конкретною реалізацією криптосистеми. Для створення цифрового підпису довжиною 512 біт, як це було для стандарту ГОСТ 34.310-95, число p повинно задовольняти нерівність $p < 2^{256}$. В подальшому вважаємо, що число p лежить в межах $2^{255} < p < 2^{256}$;

b) просте число q, яке визначає порядок еліптичної кривої E циклічної підгрупи групи точок [80].

Для генерування простого числа можна використовувати такі процедури: а)генерування числа q в стандарті ГОСТ 34.310-95; б)генерування випадкового простого числа довжиною 256 біт; в)генерування сильного простого числа; г) коефіцієнти $a, b \in F_p^2$, що задають ЕК E.

Для генерування простого числа можна скористатися розробленим алгоритмом, який дозволяє ефективно шукати прості числа [70].

В роботі [70] запропоновано пошук НСД за наступним алгоритмом:

Алгоритм 3.1 – пошук найбільшого простого числа.

1. Маємо останнє просте число $p_i - \max$.

2. Визначаємо двійкове представлення цього числа n - розрядів.

3. Через розмежовану систему числення залишкових класів та властивість періодичності рекурентним шляхом отримуємо залишок числа p_i по заданому модулю, наступним чином:

$$\begin{cases} c_{i+1} = c_i \cdot 2(\text{mod } p), c_1 = 1, a_1 = 1 \\ b_{i+1} = (c_{i+1} \cdot a_i + b_i) \bmod p, b_1 = 1, i = 1, a_i = 0,1 \end{cases}. \quad (3.1)$$

Програмним шляхом організовуємо генератор залишків по модулю p_j. В результаті знаходимо зображення цього числа в СЗК до числа простих чисел, які не перевищують $\sqrt{p_i}$ - половина розряду p_i.

4. Додаємо до залишків число 2 по всіх модулях, ця операція припиняється, коли по одному з модулів отримаємо 0. Тоді пропускаємо числа, кратні модулю, по якому залишок дорівнює 0. Операцію повторюємо до тих пір, поки не буде знайдено найбільше в світі просте число. Блок схема алгоритму подана у розділі 4.

Основною перевагою даного алгоритму є те, що операції виконуються тільки над залишками, а не з великими простими числами. Отже, для генерування р – модуля перетворення груп точок ЕК можна скористатися алгоритмом 3.1, який дозволяє спростити процедуру знаходження простого числа.

В стандарті X9.62-1998 пропонується 3 способи отримання параметрів ЕК [4]. Перший спосіб полягає у використанні кривих, заданих в цьому стандарті. Серед них тільки одна ЕК задовольняє потрібному простому числу. Другий спосіб зводиться до випадкового вибору еліптичної кривої та перевірки її параметрів. Третій спосіб ґрунтується на виборі потрібних параметрів і побудові кривої за цими параметрами.

Незважаючи на існуючі підходи щодо генерування параметрів ЕК, існують певні нерозв'язані задачі стосовно продуктивності генерування потрібної кількості кривих за певний період часу. Нижче пропонується один з ефективних методів вирішення цього завдання, що базується на еволютивному підході, запропонованого в роботі [8].

Якщо на деякій множині задана складна (цільова) функція від кількох змінних, то еволютивний алгоритм – це алгоритм, який дозволяє за оптимальний час знаходити максимально наближене значення [76]. Таким чином еволютивний підхід можна ефективно застосовувати для знаходження оптимальних (або квазі-оптимальних) розв'язків в задачах з обмеженням на часовий ресурс.

Для реалізації еволютивного підходу щодо розв'язання задачі генерування параметрів ЕК було взято за основу алгоритм A.12.4 стандарту IEEE P1363 [14]. Згідно нього розроблено алгоритм генерування параметрів ЕК з використанням еволютивного підходу (рис.3.1), дослідження якого проведено при аналізі часових характеристик.

Ключовим моментом при розробці еволютивних алгоритмів є визначення критерію (цільової функції), на основі якої здійснюється відбір нових об'єктів. Доцільно використати такий критерій на підставі наведеного в стандарті співвідношення оцінки гладкості кривої, яке визначається

$$f = c \cdot b^2 (\mathrm{mod}\, p) - a^3 (\mathrm{mod}\, p), \qquad (3.2)$$

де $a, b \in GF(p)$.

Суть розробленого алгоритму наближається до типових генетичних алгоритмів і полягає в наступному (рис. 3.1). Спочатку в Init Population генеруються випадковим чином об'єкти з параметрами еліптичної кривої a, b. В подальшому здійснюється сортування параметрів в порядку зростання значення f. З першого етапу вибираються 1024 найкращі, для яких абсолютне значення критерію f є найменше, і заносяться у масив Mating Pool. Далі здійснюється ітеративна процедура модифікації об'єктів P_i з масиву Mating

Pool шляхом застосування операції мутації. При цьому в операторі мутації враховано напрям зміни коефіцієнтів a, b. Далі обчислюється критерій f для отриманого шляхом мутації нового об'єкта. Якщо отримане абсолютне значення менше за попереднє, то новий об'єкт записується на місце попереднього (предка). Наприкінці алгоритму здійснюється перевірка на невиродженість отриманої еліптичної кривої.

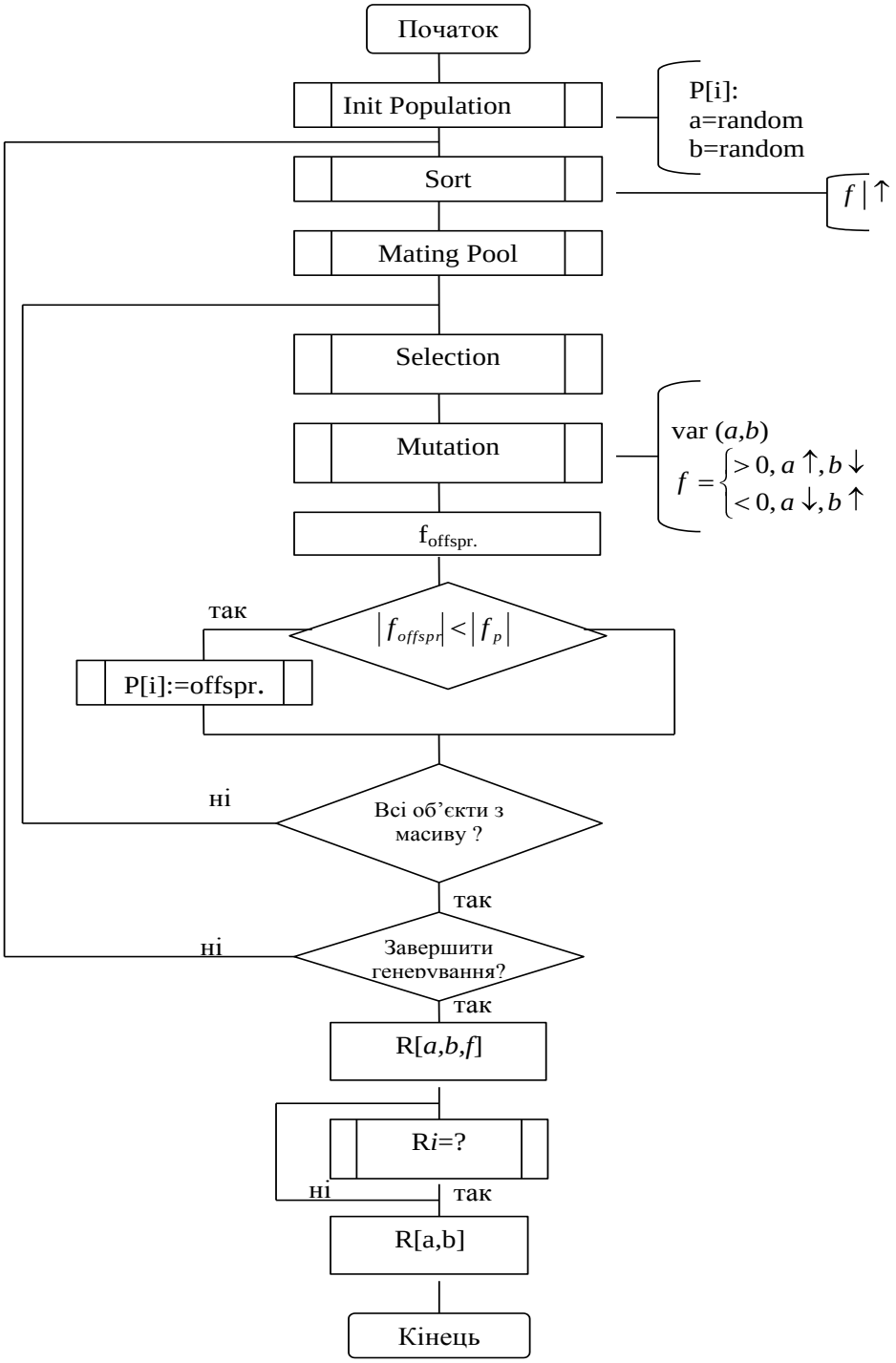


Рисунок 3.1 – Алгоритм генерування параметрів еліптичної кривої з використанням еволютивного підходу.

На основі запропонованого алгоритму розроблено програмний засіб, який дозволяє отримати пари коефіцієнтів a, b, що придатні для побудови стійких криптосистем на основі еліптичних кривих. Результати роботи програми наведено у (додатках А, Б) та на рис. 3.2, 3.3. Загалом було отримано більше 600 пар, проте після додаткової перевірки на невиродженість залишено 208. В таблиці 3.1 наведено коефіцієнти c_i , для яких згідно з алгоритмом обчислені параметри a, b.

У табл. 3.2 поміщено кількість згенерованих пар параметрів ЕК для певних коефіцієнтів c_i . Також в таблиці цього додатку показано кількість пар a, b однакової та різної розмірностей.

Таблиця 3.1–Коефіцієнти c_i.

Коефіцієнт c_i	Значення коефіцієнта
c_1	51799996682396758944004974568555459888197891979211637345686009034147319967190
c_2	14186545527804375329915594346183841971258756371427817362289709707294098520 26
c_3	34002950479911914546202967720669257702764443651718841281013484396596574419760
c_4	367171650888157766423778390433037732858905606854339925429282583561340337568 28
c_5	26359442492233452681396927792227654714651114587456290454109226350679969301 32
c_6	38450574121749404090627999052187786901395521010345406268485445215400329092998

Таблиця 3.2 – Кількості згенерованих пар (a,b).

Коефіцієнти c_i	Загальна кількість пар	Кількість пар приблизно рівної розмірності $\lg a \approx \lg b$	Кількість пар різної розмірності $\lg a \neq \lg b$
c_1	4	1	3
c_2	18	9	9
c_3	16	5	11
c_4	20	7	13
c_5	125	63	62
c_6	25	16	9
Всього	208	101	107

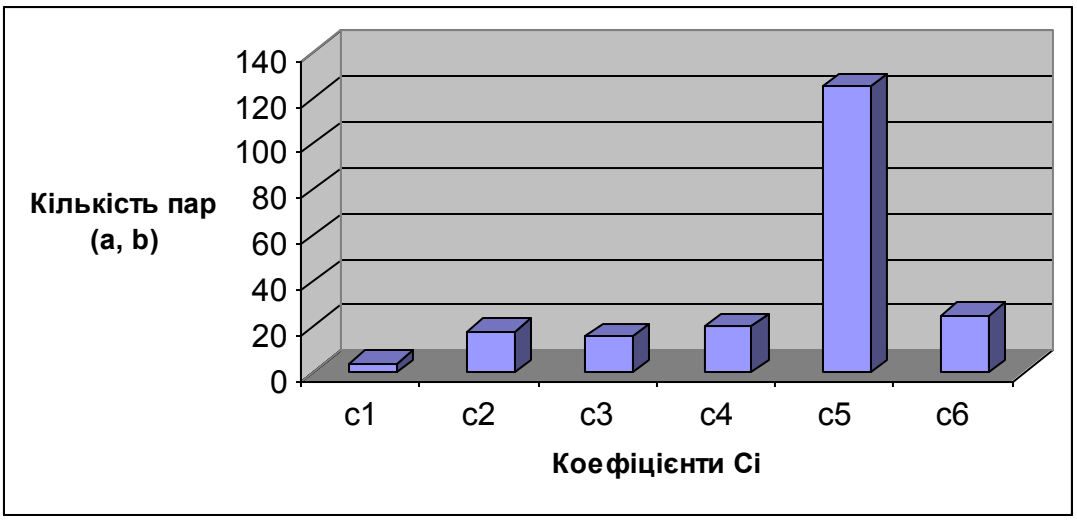


Рисунок 3.2 – Залежність кількості пар від коефіцієнтів Ci.

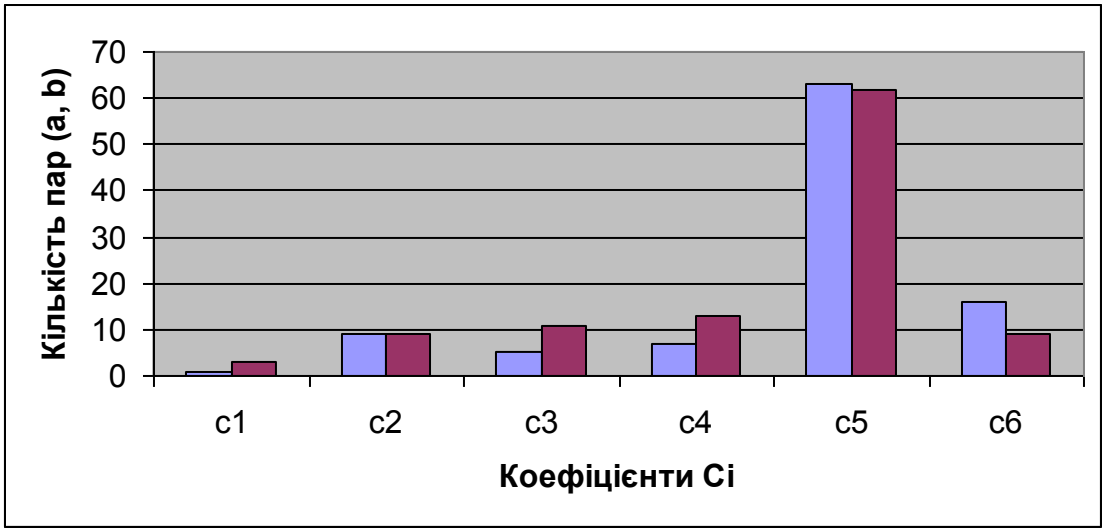


Рисунок 3.3 – Співвідношення кількості пар параметрів (a, b) приблизно однакової та різної розмірностей для коефіцієнтів Ci.

Аналіз отриманих діаграм дозволяє встановити, що при генеруванні параметрів ЕК для заданого коефіцієнта C_1 було згенеровано 8 пар з яких 5 одинакової розмірності та 3 різної. При коефіцієнті C_2 отримали 18 пар половина яких однакової розмірності, для C_3 згенеровано 15 пар, при чому 4 пари однакової розмірності та 11 різної, відповідно для C_1 отримали 19 пар – 12 одинакової розмірності, 7 різної. Максимальну кількість згенерованих пар (a, b) було отримано при коефіцієнтові C_5 - 123 пари з яких 62 пари однакової розмірності.

Дослідження показали, що при генеруванні параметрів ЕК співвідношення кількості пар (a, b) приблизно однакової та різної розмірності

для коефіцієнтів Сі приблизно 50%, а найбільшу кількість пар було згенеровано при C_5.

Незважаючи на вказані переваги, слід зазначити, що запропонований алгоритм має функціональні обмеження, а саме: простий перебір параметрів (a, b), хоча і з врахуванням операторів мутації та напрямків зміни коефіцієнтів a, b. Отже, для ефективного генерування параметрів ЕК потрібно використовувати інші підходи, які б дозволили пришвидшити роботу алгоритму.

3.1.2 Розробка та моделювання алгоритму генерування параметрів ЕК з використанням символів Якобі.

Еволютивний алгоритм генерування параметрів є імовірнісним. Тому для підвищення швидкодії розробленого програмного засобу використано новий підхід з використанням символів Якобі [7]. Дослідження показали, що це дає змогу зменшити часовий ресурс і збільшити загальну кількість пар параметрів *(a, b)*, що генеруються. Символи Якобі дають можливість визначити, чи існує таке h, яке задовільняє рівність, яка отримана шляхом перетворення виразу (3.2):

$$h = (c^{-1}(mod\ p)\cdot a^3)\ mod\ p. \qquad (3.3)$$

Завдяки цьому можна відсіювати значення h, для яких символ Якобі дорівнює –1. Якщо символ Якобі дорівнює 1, то

$$b^2 \equiv h(\mathrm{mod}\ p). \qquad (3.4)$$

Запропонований алгоритм збільшує швидкодію в порівнянні з еволютивним алгоритмом пошуку параметрів ЕК за рахунок відсіювання параметрів *(a, b)*, для яких символ Якобі недорівнює 1.

Блок-схема алгоритму подана на рис. 3.4. З врахуванням вище сказаного загальний алгоритм генерування параметрів ЕК запишеться наступним чином:

1) генерування числа *c*;
2) генерування випадково числа *a*;
3) обчислення значення *h*;
4) знаходження символу Якобі $j = J(h, p)$;
5) якщо $j = -1$, то перейти до кроку 2;
6) знаходження числа *b;*
7) генерування базової точки ЕК;
8) перехід на крок 1.

Слід зазначити, що при кожному генеруванні нового числа *c* для кожної пари параметрів ЕК значно зменшується швидкодія алгоритму.

З використанням наведеного алгоритму розроблено інтерфейсну програму генерування параметрів ЕК для заданого модуля *p* (рис. 3.3). Для визначення швидкодії запропонованого алгоритму було згенеровано 10 тисяч пар параметрів ЕК. Загальний час генерування становить 2 хв. 26,58 сек. Тобто приблизний час генерування однієї пари дорівнює 14 мс. Варто зазначити, що при визначенні швидкодії число *a* було 256-бітним, а алгоритм допускає зменшення його розрядності. Тоді швидкодія алгоритму буде ще вищою. Програма, розроблена для реалізації поданого алгоритму, була доповнена генеруванням базової точки на ЕК. Модуль кривої *p* було взято з стандарту X9-62: p = 115792089210356248762697446949407573530086143415290314195533631308867097853951.

На рис. 3.7 подано приклад згенерованих параметрів і координат базових точок ЕК.

Для генерування параметрів ЕК, було розроблено програмний продукт в середовищі візуального програмування C++ Builder 6.0, тому по своїй суті є подійнісно-орієнтованим.

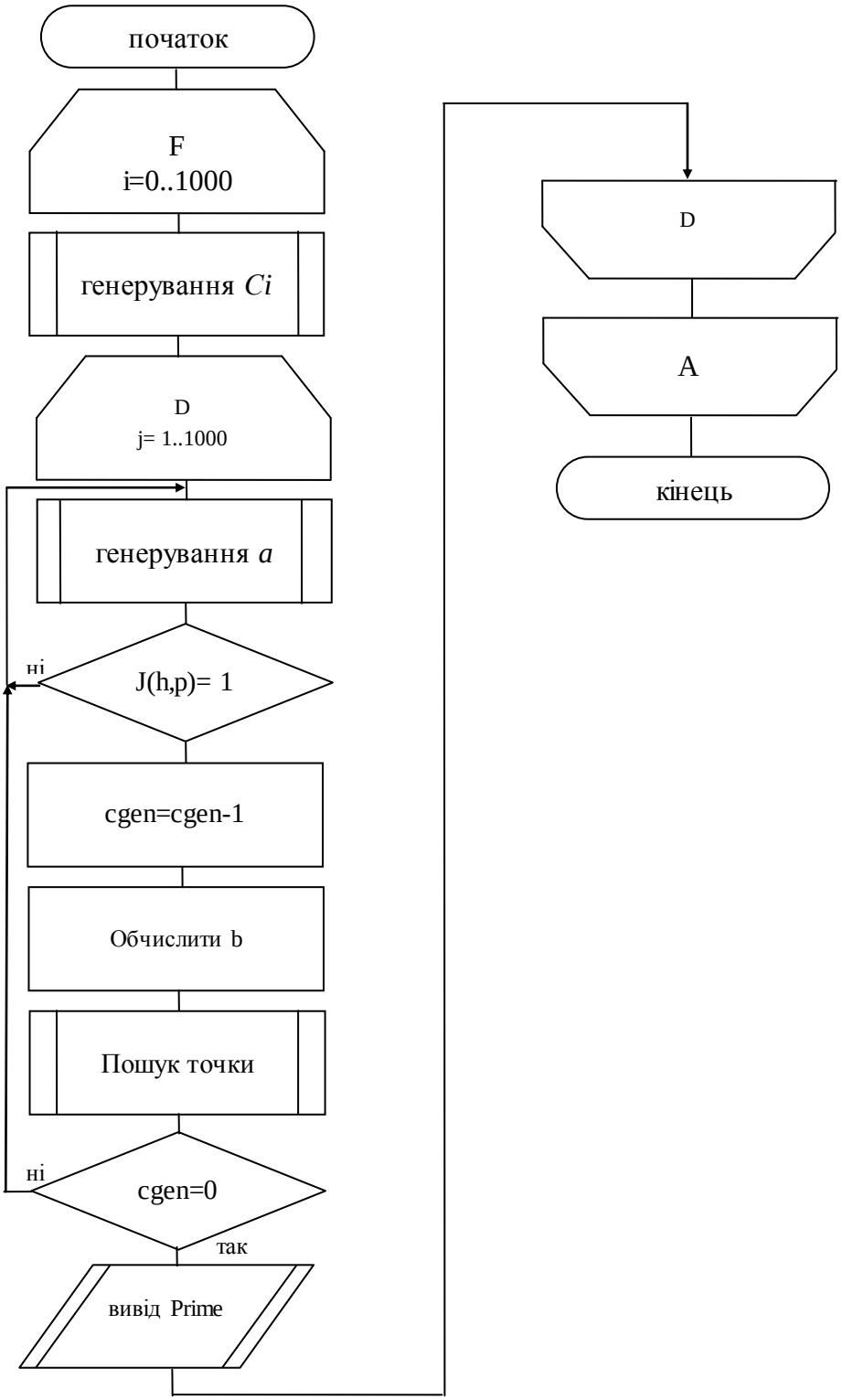


Рисунок 3.4 – Алгоритм генерування параметрів ЕК з використанням символів Якобі.

Робоче вікно програми подано на рис. 3.5. Текст основних модулів програми поданий в додатку А. Текст бібліотек *lip.c* та *BigInt.cpp* в додатку А не наведений через занадто великий обсяг коду.

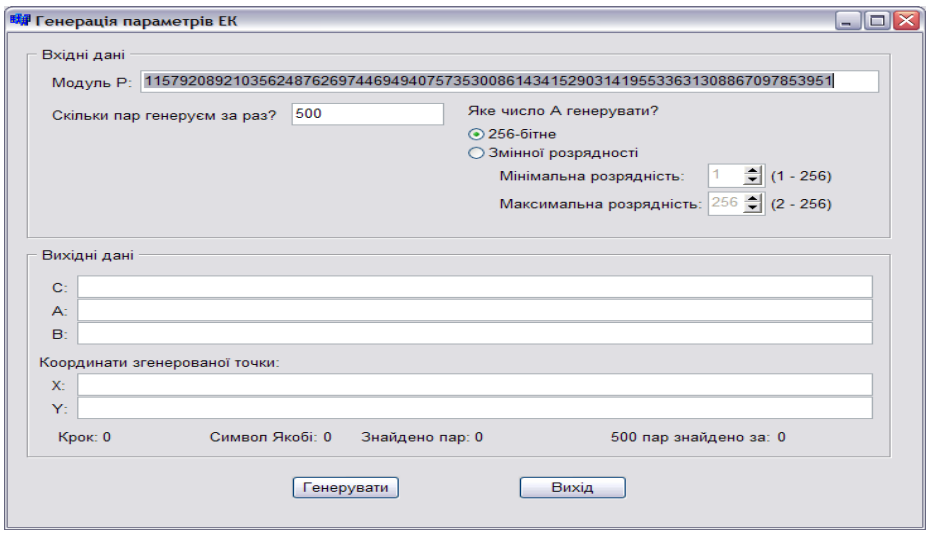


Рисунок 3.5 – Інтерфейс програми алгоритму генерування параметрів ЕК.

Кількість пар, які генеруються, визначає паузу між циклами генерування. Під час генерування параметрів доступу до елементів керування вікном немає, того після генерування кількості пар, яка задана в вищезазначеному полі, робиться пауза на 100 мс. При вказанні дуже малих значень швидкодія програми суттєво зменшиться, бо буде багато пауз, під час яких параметри генеруватися не будуть. При заданні великих значень швидкодія буде високою, зате майже не буде доступу до вікна, необхідно затратити значну кількість часу для генерування визначеної кількість пар параметрів. Оптимальний вибір – 500-1000 пар. Назва поля редагування, в якому встановлюється кількість генерованих пар: **EFreq**. По замовчуванні в цьому полі встановлюється число 500.

3.1.3 Обчислення розрядності параметрів ЕК.

Не завжди потрібно генерувати саме 256-бітне число a. Його розрядність можна змінити переключившись на змінну, розрядність і вказати межі бітності числа a. Якщо потрібно, щоб це число було строгої розрядності (наприклад, 200 біт) то потрібно встановити однакові значення для максимальної та мінімальної розрядності. Якщо розрядність встановити неправильно, то подається повідомлення, представлене на рис. 3.6.

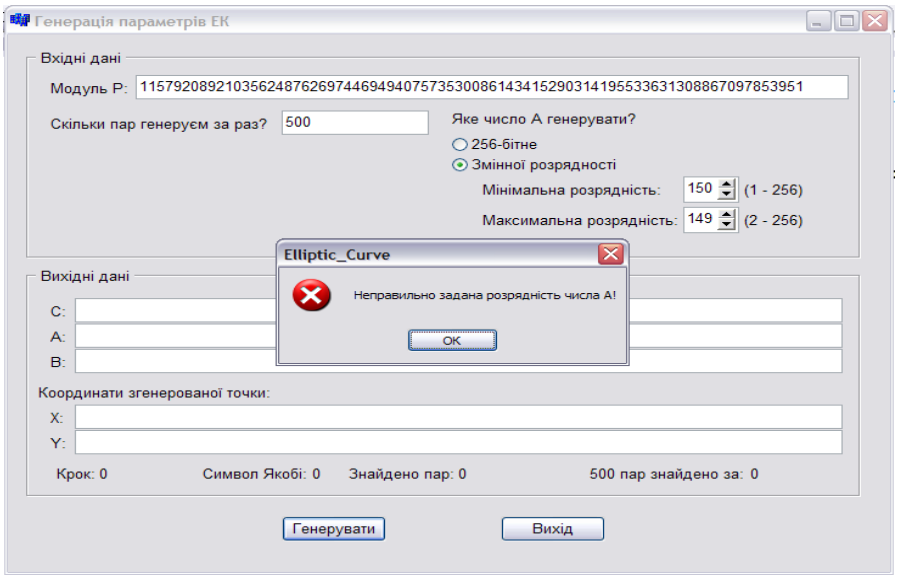


Рисунок 3.6 – Процес генерування параметрів ЕК.

До вихідних даних програми відносяться згенеровані параметри, число *c*, координати точки на ЕК із знайденими параметрами, кількість запусків процедури генерації параметрів, символ Якобі для конкретного числа *a,* кількість знайдених параметрів та час пошуку параметрів (500, 1000, 1500 і т.д.).

Числа a, b, c виводяться в поля редагування з іменами ANum, BNum, CNum відповідно. В цих полях властивість ReadOnly встановлена в true, тобто змінювати в них текст користувач не може, але може скопіювати його в буфер обміну ОС Windows. Всі інші вихідні дані виводяться за допомогою міток (тип TLabel).

Їх імена для значень кількості запусків процедури генерації символу Якобі, кількості знайдених пар *(a, b)*, мітки, яка містить кількість згенерованих пар (для часової характеристики значення кількості пар кратні 500) та мітки, яка містить загальний час генерації, мають значення відповідно CStep, Jac, Couples, LFound, Found.

Приклад роботи програми для генерації 1200 пар з бітовим коридором для *a* 120-200 біт подано на рис. 3.7.

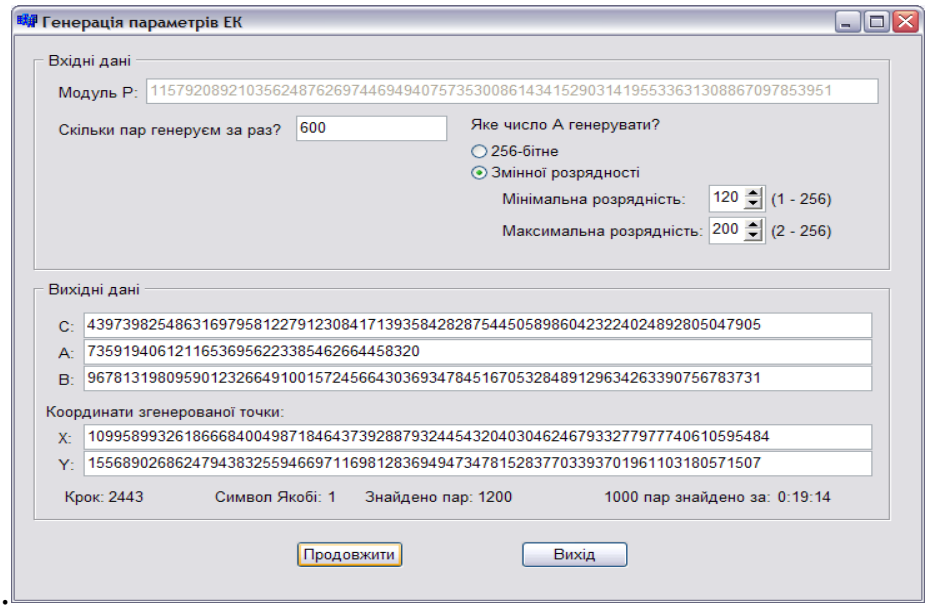


Рисунок 3.7 – Інтерфейс генерування параметрів та базової точки ЕК

Також вихідні дані виводяться в два файли – **Numbers.txt** та **Time.txt**. В перший файл виводяться згенеровані параметри та координати точки. Він програмою ніколи не стирається. В файл Time.txt виводиться часова характеристика. При кожному запуску програми цей файл очищується. Програма використовує зовнішні бібліотеки *lip.c* та *BigInt.cpp*. В першій бібліотеці містяться функції для реалізації алгоритмів, які використовує IEEE P1363 A.12.4 та IEEE P1363 A.11.1. Друга містить функції, які використано для реалізації процедури *int2str*

Після доповнення програми генеруванням точки швидкість виконання алгоритму незначно зменшилася. Результати моделювання алгоритму часових характеристик наведено в табл. 3.4.

Таблиця 3.4 –Часові характеристики роботи програми

К-сть пар пара-метрів (a, b) ЕК	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
Час, с Алгоритм на основі символів Якобі	10,89	21,19	31,77	42,69	53,25	63,66	74,25	84,80	95,52	106,05
Еволютивний алго-ритм, 10^{-4} с	108,001	209,146	315,0749	423,2771	528,098	631,339	736,391	840,961	947,157	1051,764

Аналіз результатів моделювання показав, що підхід з використанням символів Якобі дозволяє пришвидшити час роботи алгоритму в 10^3 разів та генерувати параметри ЕК, які придатні для побудови стійких систем захисту інформаційних потоків з використанням математичного апарату ЕК.

Проведений аналіз генерування параметрів, які використовуються в алгоритмі Шуфа, показує, що для побудови стійких систем захисту інформаційних потоків з використанням ЕК потрібно розв'язати задачу знаходження порядку групи точок на ЕК за допомогою наступних алгоритмів: «крок гіганта-крок малюка», Шуфа, Еткіна та Еліза. Крім цього потрібно, щоб порядок був максимальним та містив великий простий дільник. Слід зазначити, що для визначення стійкості ЕК потрібно знати точну кількість точок, які можна знайти з використанням алгоритму Шуфа.

3.2 Дослідження та моделювання системних характеристик алгоритму Шуфа в задачах застосування еліптичних кривих.

3.2.1 Підвищення ефективності перетворення Фробеніуса в задачах захисту інформаційних потоків в комп'ютерних мережах.

Обчислення порядку групи точок ЕК над кінцевим простим полем визначає основні технічні параметри стійкості паралельних алгоритмів шифрування інформаційних потоків і є фундаментальним параметром при розв'язку задачі захисту в комп'ютерних мережах. Для пошуку порядку ЕК доцільно скористатися алгоритмом Шуфа, складність якого базується на: перетворенні Фробеніуса; знаходженні значення $x^p, y^p, x^{p^2}, y^{p^2}$ по модулю $f_l(x)$; знаходженні найбільшого спільного дільника двох поліномів; китайській теоремі про лишки.

Покращення основних трудомістких операцій в алгоритмі Шуфа дозволить зменшити його часову складність, яка оцінюється $O(\log^8 p)$ табл. 1.6. та пришвидшити пошук порядку ЕК.

Нехай p - просте число, $p > 3$, еліптична крива $E = E_{a,b}$ над полем Z / pZ задана рівнянням 1.10.

Позначимо через $\overline{Z / pZ}$ алгебраїчне доповнення поля Z / pZ. Відображення Фробеніуса $\varphi : \overline{E(Z / pZ)} \to \overline{E(Z / pZ)}$ визначається співвідношенням [24]:

$$\varphi(x, y) = (x^p, y^p),\ \varphi(O) = O. \qquad (3.2)$$

Для виконання операції x^p, y^p скористаємося алгоритмом піднесення до степеня у розмежованій системі числення базису Крестенсона, описаного в пункті 2.2, та з використанням табл. 3.5 і 3.6. В стовбцях матриці записано величини x^{2^i}, y^{2^i} в базисі Радемахера, тобто y_{ij} та x_{ij}=0, 1. Тоді будь–який степінь *x,y* можна записати за степенями 2: $p = p_{n-1}2^{n-1} + \ldots + p_i 2^i + \ldots + p_1 2 + p_0$, що дозволяє зменшити обчислювальну складність і шуканий результат отримуємо, перемноживши відповідну кількість стовбців з використанням табл. 2.4.

Таблиця 3.5 – Матриця знаходження x^{2^i} в базисі Радемахера–Крестенсона

$x_{n-1\ n-1}$	...	$x_{i\ n-1}$		$x_{1\ n-1}$	$x_{0\ n-1}$
...	...	...	...	...	...
$x_{n-1\ j}$	...	$x_{i\ j}$	...	$x_{1\ j}$	$x_{0\ j}$
...	...	...	...	...	...
$x_{n-1\ 1}$	...	$x_{i\ 1}$	...	$x_{1\ 1}$	$x_{0\ 1}$
$x_{n-1\ 0}$	...	$x_{i\ 0}$	...	$x_{1\ 0}$	$x_{0\ 0}$
$x^{2^{n-1}}$	...	x^{2^i}	...	x^{2^1}	x^{2^0}

Таблиця 3.6 – Матриця знаходження y^{2^i} в базисі Радемахера–Крестенсона

$y_{n-1\ n-1}$	⋯	$y_{i\ n-1}$		$y_{1\ n-1}$	$y_{0\ n-1}$
⋯	⋯	⋯	⋯	⋯	⋯
$y_{n-1\ j}$	⋯	$y_{i\ j}$	⋯	$y_{1\ j}$	$y_{0\ j}$
⋯	⋯	⋯	⋯	⋯	⋯
$y_{n-1\ 1}$	⋯	$y_{i\ 1}$	⋯	$y_{1\ 1}$	$y_{0\ 1}$
$y_{n-1\ 0}$	⋯	$y_{i\ 0}$	⋯	$y_{1\ 0}$	$y_{0\ 0}$
$y^{2^{n-1}}$	...	y^{2^i}	...	y^{2^1}	y^{2^0}

Відображення φ є гомоморфізмом, тому алгебраїчне доповнення $E(\overline{Z/pZ})$ при дії φ переходить в $E(Z/pZ)$. Згідно теореми Хасе [24]:

$$\left|E(Z/pZ)\right| = p+1-t, \tag{3.3}$$

де $|t| < 2\sqrt{p}$. Ціле число t називається слідом відображення Фробеніуса. Тоді відображення φ задовольняє рівняння [24]:

$$\varphi^2 - t\varphi + p = 0. \tag{3.4}$$

Для кожного натурального числа n позначимо через $E[n]$ підгрупу $E(\overline{Z/pZ})$, що складається з точок, порядок яких ділить n:

$$E[n] = \left\{P \in E(\overline{Z/pZ}) \,\middle|\, nP = O\right\}. \tag{3.5}$$

Для перевірки коректності поліному $\psi_3(x,y)$ слід скористатися математичними викладками теореми 1[24].

Теорема1. Якщо $n>1$ і p не ділить n, то група $E[p]$ ізоморфна $Z/pZ \times Z/pZ$.

Тоді поліноми $\psi_n(x,y) \in Z/pZ[x,y]$, $n=-1,0,1,2,\ldots$, визначаються з наступних рекурентних співвідношень:

$\psi_{-1}(x,y)=-1$, $\psi_0(x,y)=0$, $\psi_1(x,y)=1$, $\psi_2(x,y)=2y$,

$\psi_3(x,y)=3x^4+6ax^2+12bx-a^2$,

$\psi_4(x,y)=4y(x^6+5ax^4+20bx^3-5a^2x^2-4abx-8b^2-a^3)$;

далі при $n \geq 3$

$$\psi_{2n}(x,y)=\psi_n(x,y)(\psi_{n+2}(x,y)\psi_{n-1}(x,y)^2-\psi_{n-2}(x,y)\psi_{n+1}(x,y)^2)/(2y), \quad (3.6)$$

і при $n \geq 2$

$$\psi_{2n+1}(x,y)=\psi_{n+2}(x,y)\psi_n(x,y)^3-\psi_{n+1}(x,y)^3\psi_{n-1}(x,y), \quad (3.7)$$

всюди y^2 потрібно заміняти на x^3+ax+b.

Для зменшення обчислювальної складності, всі операції множення та експоненціювання здійснюються згідно запропонованих алгоритмів з використанням матричних методів на базі ТЧБ Радемахера та Крестенсона.

Для перевірки коректності поліному $\psi_3(x,y)$ знайдемо x - координату для $2P$ і $-P$ з співвідношення (1.11):

$$x=\lambda^2-2x=\frac{(3x^2+A)^2}{4y^2}-2x, \quad (3.8)$$

і $(3x^2+A)(4y)=9x^4+6Ax^2+A^2$, якщо y^2 замінити на x^3+ax+b, то $12(x^4+Ax^2+Bx)=9x^4+6Ax^2+A^2$. Після елементарних перетворень отримаємо:

$$\psi_3(x,y)=3x^4+6ax^2+12bx-a^2. \quad (3.9)$$

Якщо $\psi_3(x,y)=0$ і $2P=\pm P$, тоді $P=O$ і P точка 3 порядку $P\in E$ [-].

Поліноми $\psi_n(x,y)$ є поліномами ділення, а $f_n(x)$ визначається рівністю [24]:

$$f_n(x)=\begin{cases}\psi_n(x,y), & \text{n непарне},\\ \psi_n(x,y)/y, & \text{n парне}.\end{cases} \tag{3.10}$$

$f_n(x)$ є поліноми від x, тобто $f_n(x)\in Z/pZ$ [-]. Крім цього, якщо n - непарне, p не ділить націло n, то $\deg f_n(x)=(n^2-1)/2$.

Для пошуку операції експоненціювання на ЕК доцільно скористатися аналітикою наступних теорем [24].

Теорема2. Нехай $P=(x,y)\in E(\overline{Z/pZ})\setminus E$ [-]. Нехай $n\geq 3$. Рівність $nP=O$ виконується тоді і тільки тоді, коли $f_n(x)=0$.

Теорема3. Нехай $P=(x,y)\in E(\overline{Z/pZ})\setminus E$ [-], $n\geq 2$, причому $nP\neq O$. Тоді

$$nP=\left(x-\frac{\psi_{n-1}(x,y)\psi_{n+1}(x,y)}{\psi_n(x,y)^2},\frac{\psi_{n+2}(x,y)\psi_{n-1}(x,y)^2-\psi_{n-2}(x,y)\psi_{n+1}(x,y)^2}{4y\psi_n(x,y)^3}\right). \tag{3.11}$$

При цьому операції множення та експоненціювання даного виразу виконуються на основі запропонованих алгоритмічних обчислень з використанням ТЧБ Радемахера-Крестенсона.

З оцінки чисельного експерименту складностей операції множення двох n розрядних чисел (рис.2.1), показано ефективність використання запропонованого в розділі 2.2 алгоритму (рис.2.2).

3.3 Зменшення складності знаходження найбільшого спільного дільника з застосуванням теоретико-числових базисів Радемахера–Крестенсона в алгоритмі Шуфа.

Однією з базових та трудомістких операцій алгоритму Шуфа є знаходження НСД та значення $t(\mathrm{mod}\prod l)$ з використанням китайської теореми про лишки. При цьому з теореми Хасе випливає, що шукане значення буде рівне: $|E(Z/pZ)| = p+1-t$.

Розглянемо спочатку випадок $l=2$. За теоремою Ферма $x^{p-1} \equiv 1(\mathrm{mod}\ p) \Leftrightarrow x^p - x \equiv 0(\mathrm{mod}\ p)$. Це означає, що $x^p - x \vdots p$ Тоді в $E(Z/pZ)$ знайдеться ненульова точка $P=(x,0)$ другого порядку, яка належить кривій $x^3 - ax + b \equiv 0(\mathrm{mod}\ p) \Rightarrow x^3 - ax + b \vdots p$ тоді і тільки тоді, коли виконується умова:

$$НСД\,(x^p - x, x^3 + ax + b) \neq 1. \qquad (3.12)$$

Ця умова рівносильна парності числа $|E(Z/pZ)|$ що, в свою чергу, рівносильно парності t, поскільки $p+1$ парне. Тобто, $t \equiv 0(\mathrm{mod}\,2)$, тоді і тільки тоді, коли:

$$НСД\,(x^p - x, x^3 + ax + b) \neq 1, \qquad (3.13)$$

в іншому випадку $t \equiv 1(\mathrm{mod}\,2)$.

Для знаходження найбільшого спільного дільник можна скористатися класичним алгоритмом Евкліда [83]

Алгоритм 3.1 (Евкліда) – знаходження найбільшого спільного дільника двох чисел X, Y.

Нехай X,Y цілі числа, не рівні одночасно нулю, і послідовність чисел $X,Y,r_1 > r_2 > \ldots > r_i > \ldots > r_n$ визначена так, що кожне r_i - залишок від ділення передпопереднього числа на попереднє, а передостаннє ділиться націло на останнє, тобто:

$$X = Y \cdot q_0 + r_1;$$
$$Y = r_1 \cdot q_1 + r_2;$$
$$r_1 = r_2 \cdot q_2 + r_3;$$
$$r_2 = r_3 \cdot q_3 + r_4;$$
$$\ldots$$
$$r_i = r_{i+1} \cdot q_{i+1} + r_{i+2};$$
$$\ldots$$
$$r_{n-1} = r_n \cdot q_n.$$

Тоді НСД(X,Y) рівний r_n, останньому ненульовому члену даної послідовності.

В [28] оцінено складність цього алгоритму і вона не перевищує 5κ операцій ділення з остачею, де κ - кількість цифр в десятковому записі більшого з чисел X,Y. Оскільки операція ділення двох чисел розрядності n має обчислювальну складність $O((n+1)^2)$, що значно збільшує час $O1(17,5n(n+1)^2)$ виконання операції пошуку найбільшого спільного дільника з використанням алгоритму Евкліда в базисі Радемахера (двійковій формі).

Алгоритм 3.2 – Удосконалення реалізації алгоритму Евкліда з використанням розмежованої системи числення Радемахера – Крестенсона.

Для знаходження залишків r_i з використанням розмежованої системи числення Радемахера – Крестенсона операція ділення замінюється операцією додавання залишків степеневих коефіцієнтів по заданому модулі наступним чином:

1. Представляємо X,Y в базисі Радемахера:

$$X = x_{n-1} \cdot 2^{n-1} + x_{n-2} \cdot 2^{n-2} + \ldots + x_i \cdot 2^i + \ldots + x_0 \cdot 2^0;$$
$$Y = y_{n-1} \cdot 2^{n-1} + y_{n-2} \cdot 2^{n-2} + \ldots + y_i \cdot 2^i + \ldots + y_0 \cdot 2^0,$$

де y_i, x_i =0, 1, n–розрядності X,Y.

Для виконання 1 кроку алгоритму Евкліда, потрібно знайти значення $r_1 = res(X \bmod(Y))$, $r_2 = res(Y \bmod(r_1))$, …, $r_{n-1} = res(r_{n-1} \bmod(r_n))$ згідно формул:

$$
\begin{aligned}
r_1 &= \left(\sum_{i=1}^{n} res(x_i \cdot 2^i) \bmod Y \right) \bmod Y; \\
r_2 &= \left(\sum_{i=1}^{n} res(x_i \cdot 2^i) \bmod r_1 \right) \bmod r_1; \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
r_{n-1} &= \left(\sum_{i=1}^{n} res(x_i \cdot 2^i) \bmod r_n \right) \bmod r_n .
\end{aligned}
\qquad (3.13)
$$

Таким чином, з врахуванням кількості кроків знаходимо всі залишки в алгоритмі Евкліда зі складністю $O\left(17{,}5n \cdot \left(\log_2 \frac{n}{2}\right)\right)$. На рис. 3.8 подано оцінки складностей алгоритму Евкліда та його удосконалення з використанням розмежованої системи числення Радемахера – Крестенсона.

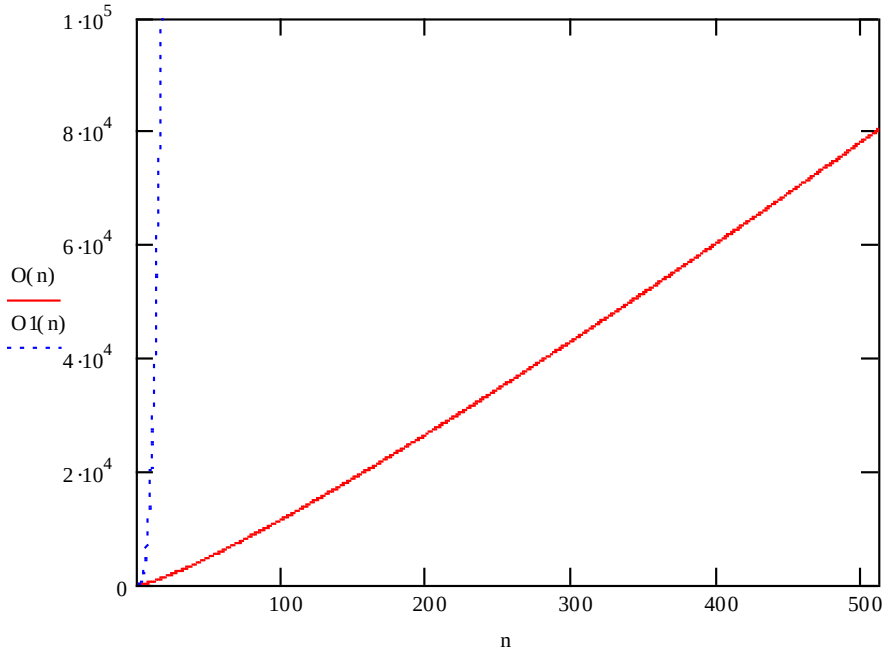


Рисунок 3.8 – Складності алгоритму Евкліда

Результати чисельного експерименту показали, що використання розмежованої системи числення дозволяє значно зменшити складність. Тому для зменшення складності паралельного алгоритму Шуфа доцільно

скористатися СЗК, яка може бути використана для реалізації високопродуктивних алгоритмів формування і опрацювання інформаційних потоків, а також застосувати ТЧБ Крестенсона та Радемахера, який породжує двійкову систему числення на основі відповідних критеріїв для знаходження НСД. Для цього доцільно використати запропонований алгоритм знаходження НСД. Незважаючи на переваги, слід зазначити, що основним недоліком цього алгоритму є його послідовність і неможливість розпаралелення.

З метою реалізації алгоритму на основі розпаралелення обчислювальних операцій, розроблений алгоритм на базі використання ТЧБ Радемахера-Крестенсона, який базується на виявленні та порівнянні нульових залишків в системі простих модулів і дозволяє вирішувати дві задачі, а саме:

1) факторизація чисел (розклад на прості множники);
2) знаходження найбільшого спільного дільника.

Алгоритм 3.3 – знаходження НСД з застосуванням ТЧБ Радемахера–Крестенсона.

Вхід: X, Y.

Вихід: $Z = НСД(X, Y)$.

1. Представляємо X, Y у базисті Радемахера в двійковій системі числення:

$$X = x_{n-1} \cdot 2^{n-1} + x_{n-2} \cdot 2^{n-2} + \ldots + x_i \cdot 2^i + \ldots + x_0 \cdot 2^0,$$
$$Y = y_{n-1} \cdot 2^{n-1} + y_{n-2} \cdot 2^{n-2} + \ldots + y_i \cdot 2^i + \ldots + y_0 \cdot 2^0. \qquad (3.14)$$

де n – розрядність процесора які представляють X, Y.

2. Представляємо двійкові коди чисел X, Y в розмежованій системі чисел залишкових класів у вигляді векторів залишків згідно виразів:

$$res(x_i \cdot 2^i) \bmod p_j = a_i,$$

$$res(y_i \cdot 2^i) \bmod p_j = b_i. \qquad (3.15)$$

В результаті коди чисел X, Y можна представити у вигляді векторів залишків у системі простих модулів p_j, які утворюють відповідні матриці (3.16), причому $E\lceil \log_2 p_k - 1 \rceil \le E\lceil \log_2 2^n \rceil = n$, тобто:

$$\left|\begin{array}{cccccc|c|cccccc|}
a_{n-1,1} & a_{n-2,1} & \cdots & a_{i,1} & \dots & a_{0,1} & \bmod p_1 & b_{n-1,1} & b_{n-2,1} & \cdots & b_{i,1} & \dots & b_{0,1} \\
a_{n-1,2} & a_{n-2,2} & \dots & a_{i,2} & \dots & a_{0,2} & \bmod p_2 & b_{n-1,2} & b_{n-2,2} & \dots & b_{i,2} & \dots & b_{0,2} \\
\dots & \dots & \dots & \dots & \dots & & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
a_{n-1,j} & a_{n-2,j} & \dots & a_{i,j} & \dots & a_{0,j} & \bmod p_j & b_{n-1,j} & b_{n-2,j} & \dots & b_{i,j} & \dots & b_{0,j} \\
\dots & \dots & \dots & \dots & \dots & & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
a_{n-1,k} & a_{n-2,k} & \dots & a_{i,k} & \dots & a_{0,k} & \bmod p_k & b_{n-1,k} & b_{n-2,k} & \dots & b_{i,k} & \dots & b_{0,k}
\end{array}\right| \qquad (3.16)$$

Приведене представлення X, Y у вигляді векторів залишків окремих бітів можна узагальнити наступним чином:

$X = \|a_{ij}\|$, $Y = \|b_{ij}\|$ і $i \in \overline{0,n}$, $j \in \overline{0,k}$ та $0 \le a_{ij}, b_{ij} \le p_j - 1$. Причому

$$1 \le Z \le \prod_{j=1}^{k} p_j . \qquad (3.17)$$

3. Для отримання значення (3.18) потрібно представити матриці X і Y у вигляді двох векторів згідно виразів:

$$a_j = res\left(\sum_{i=1}^{n-1} a_{ij} (\bmod p_j)\right);$$

$$b_j = res\left(\sum_{i=1}^{n-1} b_{ij} (\bmod p_j)\right), \qquad (3.18)$$

для всіх $j \in \overline{0,k}$.

З (3.18) отримуємо два вектори, які представляють числа X і Y у цілочисельній системі залишкових класів в базисі Крестенсона:

$$\begin{aligned} X &= (a_1 \quad a_2 \quad \dots \quad a_j \quad \dots \quad a_k); \\ Y &= (b_1 \quad b_2 \quad \dots \quad b_j \quad \dots \quad b_k); \\ p_j &= p_1 \quad p_2 \quad \dots \quad p_j \quad \dots \quad p_k. \end{aligned} \tag{3.19}$$

Для визначення компонентів (модулів) найбільшого мультиплікативного спільного дільника Z необхідно виконати порівняння $a_i = b_i = 0, \forall i \in \overline{1,k}$. Тоді найбільший мультиплікативний дільник знаходимо згідно виразу:

$$z = \prod_{j=1}^{k} p_j \,, \tag{3.20}$$

та умови

$$p_j = \begin{cases} p_j, a_j = b_j = 0 \\ 1, a_j \neq b_j \end{cases}. \tag{3.21}$$

4. Для пошуку НСД для компонентів p_j які відповідають умові (3.21) проводиться представлення вхідних чисел X і Y в розмежованій системі числення згідно (3.16) створюємо матрицю залишків по $p_j^{\,m}$, де m - показник степеня, при якому виконується умова (3.21), тобто отримуємо вектори:

$$\begin{bmatrix} a_j^{(2)} = res\left(\sum_{i=1}^{n-1} a_{ij} (\mathrm{mod}\ p_j^{\,2})\right) \\ a_j^{(3)} = res\left(\sum_{i=1}^{n-1} a_{ij} (\mathrm{mod}\ p_j^{\,3})\right) \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ a_j^{(n)} = res\left(\sum_{i=1}^{n-1} a_{ij} (\mathrm{mod}\ p_j^{\,m})\right) \end{bmatrix}; \quad \begin{bmatrix} b_j^{(2)} = res\left(\sum_{i=1}^{n-1} b_{ij} (\mathrm{mod}\ p_j^{\,2})\right) \\ b_j^{(3)} = res\left(\sum_{i=1}^{n-1} b_{ij} (\mathrm{mod}\ p_j^{\,3})\right) \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ b_j^{(n)} = res\left(\sum_{i=1}^{n-1} b_{ij} (\mathrm{mod}\ p_j^{\,m})\right) \end{bmatrix}. \tag{3.22}$$

для всіх $j \in \overline{1,k}$.

Перевірка умови (3.12) для степенів $p_j^{\,m}$ виконується згідно порівнянь

$$\begin{array}{lll} X^{(n)} = & (a_j^{(1)} \quad a_j^{(2)} \quad \dots \quad a_j^{(n)}); \\ Y^{(n)} = & (b_j^{(1)} \quad b_j^{(2)} \quad \dots \quad b_j^{(n)}); \\ P_j^{(k)} = & p_j^{1} \quad p_j^{2} \quad \dots \quad p_j^{k}. \end{array} \qquad (3.23)$$

Тоді значення Z обчислюється згідно наступної мультиплікативної функції $Z = \prod_{j=1}^{k} p_j^{m_j}$.

В порівнянні з відомим алгоритмом Евкліда запропонований алгоритм знаходження НСД характеризується наступними перевагами:

1.Обчислення матриць a_j^{m}, b_j^{m} двох векторів можна обчислити паралельно з використанням двох процесорів.

2.Виконується операція порівняння двох компонент $X^{(n)}$ і $Y^{(n)}$.

3.Отримання добутків $p_j^{m_j}$ які дорівнюють шуканому НСД.

Алгоритм 3.4 – Удосконалення алгоритму 2.3.

Запропонований алгоритм можна суттєво удосконалити шляхом скорочення кроків алгоритму вилучення 3 кроку згідно виразів (3.18) з виконанням додаткової умови:

$$\min j : a_j = b_j = 0. \qquad (3.24)$$

Для пошуку найбільшого спільного дільника для компонентів p_j які відповідають умові (3.24) проводиться представлення вхідних чисел X і Y в розмежованій системі числення згідно (3.16) створюємо матрицю залишків по p_j^{m}, де m - показник степеня, при якому виконується умова $a_j^{(m)} = b_j^{(m)} = 0$, тобто отримуємо вектори:

$$\begin{bmatrix} a_j^{(1)} = res\left(\sum_{i=1}^{n-1} a_{ij} (\mathrm{mod}\ p_j^{2})\right) & b_j^{(1)} = res\left(\sum_{i=1}^{n-1} b_{ij} (\mathrm{mod}\ p_j^{2})\right) \\ a_j^{(2)} = res\left(\sum_{i=1}^{n-1} a_{ij} (\mathrm{mod}\ p_j^{3})\right) & b_j^{(2)} = res\left(\sum_{i=1}^{n-1} b_{ij} (\mathrm{mod}\ p_j^{3})\right) \\ \dots\dots\dots\dots\dots\dots\dots & \dots\dots\dots\dots\dots\dots\dots \\ a_j^{(n)} = res\left(\sum_{i=1}^{n-1} a_{ij} (\mathrm{mod}\ p_j^{m})\right) & b_j^{(n)} = res\left(\sum_{i=1}^{n-1} b_{ij} (\mathrm{mod}\ p_j^{m})\right) \end{bmatrix}. \qquad (3.25)$$

Після цього знаходимо значення залишків по $p_{j,k}^{m_s} = p_j^{m} \cdot p_{j+k}^{s}$, де s - показник степеня, при якому виконується умова $a_{j,k} = b_{j,k} = 0$, k - наступний після j-го простого модуля, для якого виконується умова $a_{j,k} = b_{j,k} = 0$, тобто:

$$\begin{bmatrix} a_{j,k}^{(m_2)} = res\left(\sum_{i=1}^{n-1} a_{ij} (\mathrm{mod}\ p_{j,k}^{m_2})\right) \\ a_{j,k}^{(m_3)} = res\left(\sum_{i=1}^{n-1} a_{ij} (\mathrm{mod}\ p_{j,k}^{m_3})\right) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{j,k}^{(m_s)} = res\left(\sum_{i=1}^{n-1} a_{ij} (\mathrm{mod}\ p_{j,k}^{m_s})\right) \end{bmatrix} \begin{bmatrix} b_{j,k}^{(m_2)} = res\left(\sum_{i=1}^{n-1} b_{ij} (\mathrm{mod}\ p_{j,k}^{m_2})\right) \\ b_{j,k}^{(m_3)} = res\left(\sum_{i=1}^{n-1} b_{ij} (\mathrm{mod}\ p_{j,k}^{m_3})\right) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ b_{j,k}^{(m_s)} = res\left(\sum_{i=1}^{n-1} b_{ij} (\mathrm{mod}\ p_{j,k}^{m_s})\right) \end{bmatrix}. \quad (3.26)$$

Таким чином, добуток всіх модулів до $\sqrt{Y}$ степеня для яких виконується умова $a_{j,k} = b_{j,k} = 0$, буде НСД, тобто:

$$Z = \prod_{j=1}^{k} p_j^{m_j}. \quad (3.27)$$

Основною перевагою цього алгоритму є отримання добутків $p_j^{m_j}$, пропустивши 3 крок алгоритму 3.3, що суттєво пришвидшить роботу алгоритму.

Складність запропонованих алгоритмів визначається обчислювальною складністю наступних операцій: знаходженні залишків a_j, b_j чисел X, Y по простих модулях $p_j^{m_j}$ для яких виконується умова $a_j = b_j = 0$; обчислення добутку модулів $Z = НСД(X,Y) = \prod_{j=1}^{k} p_j^{m_j}$.

В таблиці 3.7 подано оцінки часової складності основних операцій алгоритму 3.3, що дозволяє зробити порівняльний аналіз з вищенаведеними алгоритмами пошуку найбільшого спільного дільника.

Таблиця 3.7 – Обчислювальна складність основних операцій алгоритму 3.3 та удосконаленого алгоритму 3.4.

№	Основні операції	Обчислювальна складність
1.	$p_j^{m_j}$	$O\left(\log_2 n \cdot \left(\log_2 n + \frac{n}{2}\right)\right)$
2.	$a_j^{(n)} = res\left(\sum_{i=1}^{n-1} a_{ij} (\mathrm{mod}\ p_j^m)\right)$ $b_j^{(n)} = res\left(\sum_{i=1}^{n-1} b_{ij} (\mathrm{mod}\ p_j^m)\right)$	$O(\log_2 n/2)$
3.	$Z = \prod_{j=1}^{k} p_j^{m_j}$	$O(k \cdot \log_2 n)$

де k - кількість модулів для яких виконується умова $a_j = b_j = 0$.

З врахуванням даних табл. 3.7, загальна обчислювальна складність запропонованого алгоритму 3.3, та удосконаленого алгоритму 3.4 буде визначатися сумою складностей основних операцій, а саме: $O\left(\log_2 n \cdot \left(\log_2 n + \frac{n}{2} + k \cdot \log_2 n\right) + n \cdot \log_2 n/2\right)$ і $O3\left(\log_2 n\left(\log_2 n + k \cdot \log_2 n + \frac{n}{2}\right) + \frac{n}{2} \cdot \log_2 \frac{n}{2}\right)$ відповідно. На рис. 3.9 показані графіки які характеризують складності існуючого та запропонованих алгоритмів в залежності від розрядності компонентів Z.

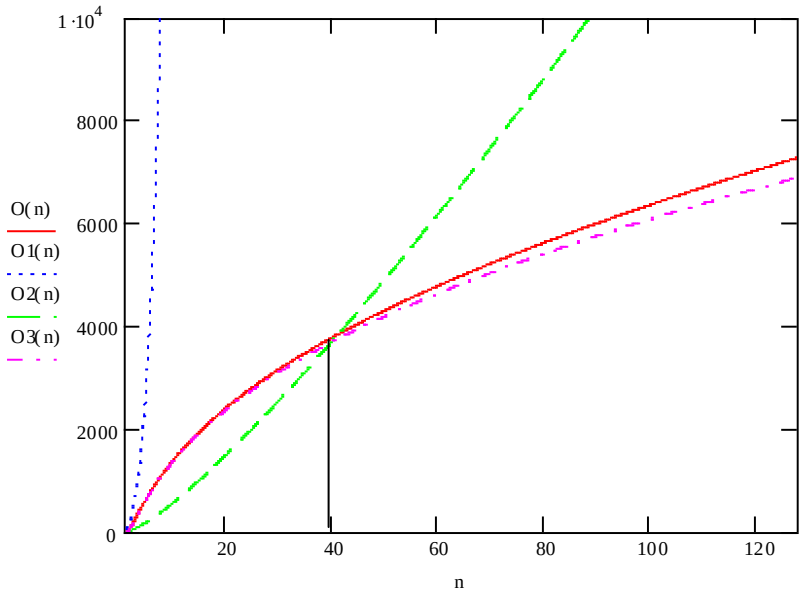


Рисунок 3.9 – Складності алгоритмів пошуку НСД(X, Y).

Результати досліджень показали, що для пошуку НСД двох чисел існуючий алгоритм Евкліда, який традиційно використовується для пошуку НСД для чисел великої розрядності, при сучасному рівні комп'ютерної техніки стає практично нездійсненний, оскільки потребує роботи процесорів на інтервалі близько 100 років. Чисельний експеримент оцінки складностей запропонованих алгоритмів пошуку НСД показує, що в діапазоні двійкових розрядів від 0 до 40 бітів, слід використовувати удосконалення реалізації алгоритму Евкліда 3.2, а при збільшенні розрядності чисел потрібно застосовувати алгоритм 3.3 та удосконалений алгоритм 3.4.

Слід зауважити, що застосування форм СЗК, які використовуються для реалізації високопродуктивних алгоритмів опрацювання і захисту інформаційних потоків, а саме: знаходження модулярного множення та піднесення до степеня, знаходження найбільшого спільного дільника, оберненого елемента по модулю і застосування для китайської теореми про лишки, дозволяє зменшувати алгоритмічну складність та проводити обчислення з використанням паралельної технології.

Китайська теорема про залишки (КТЗ).

Нехай $n_1, n_2, \ldots, n_k$ – взаємно прості числа. Тоді система порівнянь

$$x \equiv a_1 \pmod{n_1}$$

$$x \equiv a_2 \pmod{n_2}$$

. . . .

$$x \equiv a_k \pmod{n_k}$$

має єдиний розв'язок за модулем $n = n_1 * n_2 * \ \ldots * n_k$.

Алгоритм Гауса розв'язку системи лінійних порівнянь з китайської теореми про залишки.

Значення x обчислюється наступним чином:

$$x = \sum_{i=1}^{k} a_i N_i M_i \bmod n, \text{ де } N_i = n / n_i, M_i = N_i^{-1} \bmod n_i.$$

При знаходженні значення x, спочатку нам потрібно знайти $N_i = n / n_i$, $M_i = N_i^{-1} \bmod n_i$, тобто обернений елемент по модулях n_i.

Для цього скористаємось наступним алгоритмом пошуку оберненого значення за модулем.

Алгоритм 2.5 – Знаходження $M_i = N_i^{-1} \bmod n_i$ з використанням системи числення Крестенсона.

1. Знаходимо залишки по простих модулях p_i чисел n_i, N_i і зупиняємо пошук тоді коли одне з чисел n_i, N_i діляться націло, тобто залишок $r = 0$.

2. Шукаємо обернене значення $M_i = N_i^{-1} \bmod n_i$ згідно наступних співвідношень:

$$\left[\begin{array}{l|l} n_i (\bmod N_i) = r; & n_i; \\ (r+1)(\bmod N_i) = r_1; & n_i + 1; \\ r_1 (\bmod N_i) = r_2; & 2n_i + 1; \\ \ldots\ldots\ldots\ldots\ldots & \ldots\ldots \\ r_{i-1} (\bmod N_i) = r_i; & in_i + 1; \\ \ldots\ldots\ldots\ldots\ldots & \ldots\ldots \\ (r_{n-1} + r) (\bmod N_i) = r_n. & nn_i + 1. \end{array}\right. \quad (3.30)$$

3. Якщо $r_i = 0$, тоді шукаємо залишок від значення $in_i + 1$ по простих модулях p_i, тобто:

$$(n_i + 1) \bmod p_i = a_i, \quad (3.31)$$

4. Тоді шукане значення $M_i = \prod_{i=1}^{n} p_i$, для яких $a_i = 0$.

Наступним етапом в КТЗ є знаходження

$x = \sum_{i=1}^{k} a_i N_i M_i \bmod n$, тобто суми добутків. Для цього доцільно скористатися алгоритмом модулярного експоненціювання запропонованим в розділі 2.2, для зменшення алгоритмічної складності.

Приклад. Нехай потрібно знайти число N, яке при діленні на p_1=43 дає остачу r_1=10, а при діленні на p_2=209 – остачу r_2 =100 (P=209·43=8987).

p_3=209 *mod* 43

2^i	128	64	32	16	8	4	2	1
209	1	1	0	1	0	0	0	1
2^i*mod* 43	42	21	32	16	8	4	2	1

p_3=(42+21+16+1) *mod* 43=37, p_3^{-1} *mod* 43=37^{-1}*mod* 43. Для цього шукаємо 43 *mod* 37=6, додаємо 1 і послідовно додаємо 6, поки в результаті додавання за *mod* 37 не буде 0. Представимо це у вигляді таблиці:

i	0	1	2	3	4	5	6
p_{1i}	6	7	13	19	25	31	0

Звідси число K_1=6·43+1, яке націло ділиться на 37. Добуток 6·43, де 6=$(110)_2$; 43=$(101011)_2$:

	0	0	0	1	1	0
1	1024	512	256	128	64	32
0	512	256	128	64	32	16
1	256	128	64	32	16	8
0	128	64	32	16	8	4
1	64	32	16	8	4	2
1	32	16	8	4	2	1

K_1=6·43+1=(128+64+32+16+8+4+4+2)+1=259. Діленням можна знайти, що p_3^{-1} *mod* p_1=37^{-1}*mod* 43=259:37=7.

2^i	256	128	64	32	16	8	4	2	1	
259	1	0	0	0	0	0	0	1	1	
2^i*mod* 3·37	34	17	64	32	16	8	4	2	1	(34+2+1)*mod* 111=37
2^i*mod* 5·37	71	128	64	32	16	8	4	2	1	(71+2+1)*mod* 185=74
2^i*mod* 7·37	256	128	64	32	16	8	4	2	1	(256+2+1)*mod* 259=0

p_3^{-1} *mod* p_1=37^{-1}*mod* 43=7. Шукаємо p_1^{-1} *mod* p_2=43^{-1}*mod* 209.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p_{2i}	37	38	32	26	20	14	8	2	39	33	27	21	15	9	3	40	34	28	22
i	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	
p_{2i}	16	10	4	41	35	29	23	17	11	5	42	36	30	24	18	12	6	0	

K_2=36·209+1=7525 і p_1^{-1} *mod* p_2=43^{-1}*mod* 209=7525:43=175.

2^i	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	
7525	1	1	1	0	1	0	1	1	0	0	1	0	1	
$2^i mod$ 3·43	97	113	121	125	127	128	64	32	16	8	4	2	1	(97+113+121 +127+64+32 +4+1) *mod* 129=45
$2^i mod$ 5·43	11	113	164	82	41	128	64	32	16	8	4	2	1	(11+113+164 +41+64+32+ +4+1)*mod* 215=0
$2^i mod$ 5^2·43	871	973	1024	512	256	128	64	32	16	8	4	2	1	(871+973+10 24+256+64+ 32+4+1)*mod* 1075=0
$2^i mod$ 5^3·43	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	(4096+2048+1 024+256+64+ 32+4+1)*mod* 5375=2150
$2^i mod$ 5^2·7·43	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	(4096+2048+ 1024+256+64 +32+4+1) *mod* 7525=0

Отже, $p_1^{-1}\ mod\ p_2$=$43^{-1} mod$ 209=5^2·7=175. Шукане число:

N=(209·7·10+43·175·100) *mod* 8987=3235.

Основними операціями в КТЗ є: знаходження оберненого значення за модулем; знаходження залишків за модулем; модулярне експоненціювання.

Тому при визначенні складності запропонованого алгоритму, який дозволяє реалізувати КТЗ, потрібно враховувати складності вищезазначених операцій, складності яких подані в табл. 3.8.

Таблиця 3.8 – Часова складність базових операцій КТЗ

№	Основні операції	Часова складність запропонованого алгоритму	Часова складність з використанням класичного методу, а саме базису Радемахера за модулем 10
1.	$M_i = N_i^{-1} \bmod n_i$	$O\left(\frac{n^2 \cdot k}{2} \cdot\right)$	$O1(17{,}5k \cdot ((n+1)^2 + n^2 + n)$
2.	$r_{i-1} (\bmod N_i) = r_i$	$O(\log_2 n / 2)$	$O1((n+1)^2 + n)$
3.	$x = \sum_{i=1}^{k} a_i N_i M_i \bmod n$	$O(\log_2 k \cdot (2 \cdot \log_2{}^2 n + n)$	$O1(k \cdot (2n^2 + n))$

де k - кількість взаємно простих модулів $n = n_1 * n_2 * \ \ldots\ * n_k$.

Враховуючи табличні дані, часова складність КТЗ з використанням запропонованого методу буде $O\left(\log_2 k \cdot (2 \cdot \log_2{}^2 n + n) + \frac{n^2 \cdot k}{2} + \log_2 n/2\right)$, а в класичному випадку $O1(37k \cdot n^2 + 53{,}5k \cdot n + 17{,}5k + n^2 + 3n + 1))$.

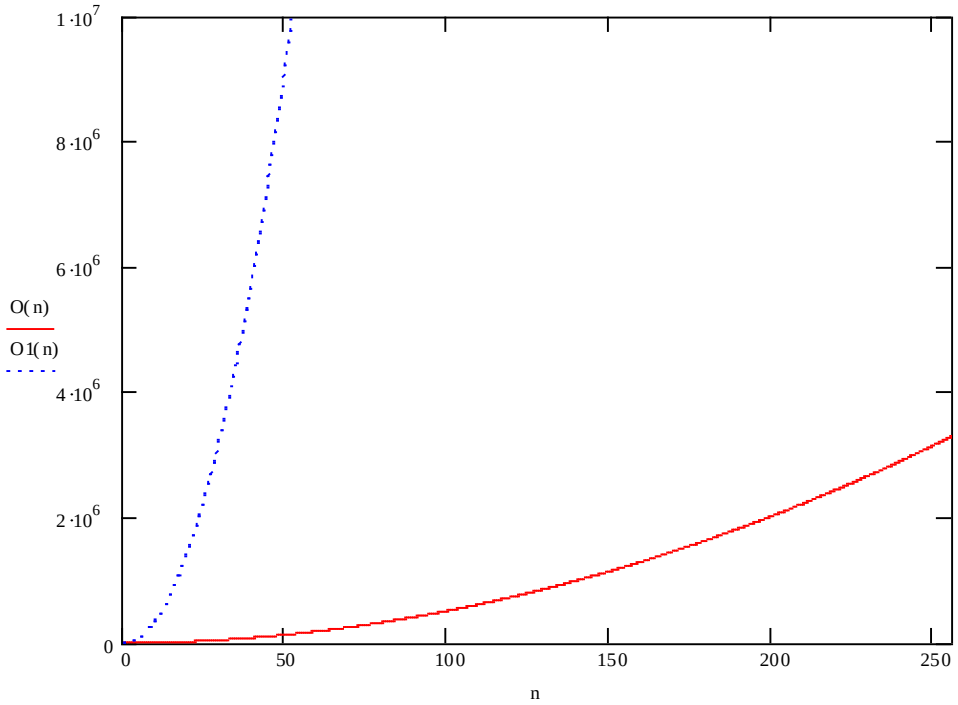


Рисунок 3.10 - Складності КТЗ, $O(n)$ - складність запропонованого методу реалізації основних компонентів КТЗ, $O1(n)$ - КТЗ з використанням класичного методу, а саме базису Радемахера за модулем 10.

Результати досліджень показали, що в основу часової складності КТЗ входять складності операцій пошуку оберненого елемента по модулю, залишків по модулю та модулярного множення. З рис. 3.10 видно, що використання запропонованого алгоритму, який грунтується на використанні ТЧБ Крестенсона-Радемахера, дозволяє на 10 % зменшити обчислювальну складність КТЗ відносно класичного методу в десятковій системі числення.

ВИСНОВКИ ДО РОЗДІЛУ 3

1. Викладені теоретичні основи ефективних алгоритмів генерування параметрів та точок ЕК в комп'ютерних мережах, що дозволяє понизити складність базових обчислювальних процесів в алгоритмі Шуфа з експоненційної до лінійної та лінійно-логарифмічної.

2. Показано, що запропонований алгоритм генерування параметрів дозволяє пришвидшити час роботи алгоритму в 10^3 разів, та генерувати параметри ЕК, які придатні для побудови стійких систем захисту інформаційних потоків з використанням математичного апарату ЕК.

3. Вперше побудовані нові швидкі перетворення базису Радемахера в базис Крестенсона, алгоритми пошуку НСД двох чисел, простого числа, Китайська теорема про залишки, які на відмінну від відомих забезпечують підвищення швидкодії та зниження часової складності на 1-2 порядки.

4. Розроблені теоретичні основи матрично-модульних перетворень з використанням ТЧБ Радемахера-Крестенсона та критерії їх ефективного застосування в алгоритмі Шуфа.

5. Досліджено операції множення та експоненціювання запропонованих алгоритмічних обчислень з використанням математичного апарату ТЧБ Радемахера-Крестенсона, зменшення складності удосконаленого алгоритму Евкліда, які підтвердили їх переваги по відношенню до відомих алгоритмів. Таким чином обгрунтовано доцільність використання даного класу запропонованих алгоритмів в якості програмно-апаратних засобів в комп'ютерних мережах та системах.

РОЗДІЛ 4
РЕАЛІЗАЦІЯ ПРОГРАМНО-АПАРАТНИХ ЗАСОБЫВ СИСТЕМИ ВИЗНАЧЕННЯ ПОРЯДКУ ЕЛІПТИЧНИХ КРИВИХ

4.1 Розробка та реалізація структурної організації алгоритмів захисту інформаційних потоків

Розробку структурної схеми організації захисту інформаційних потоків виконаємо згідно запропонованих в розділах 2.2, 3.1, 3.2 алгоритмів та структурної організації програмних засобів, поданих на рис. 4.1.

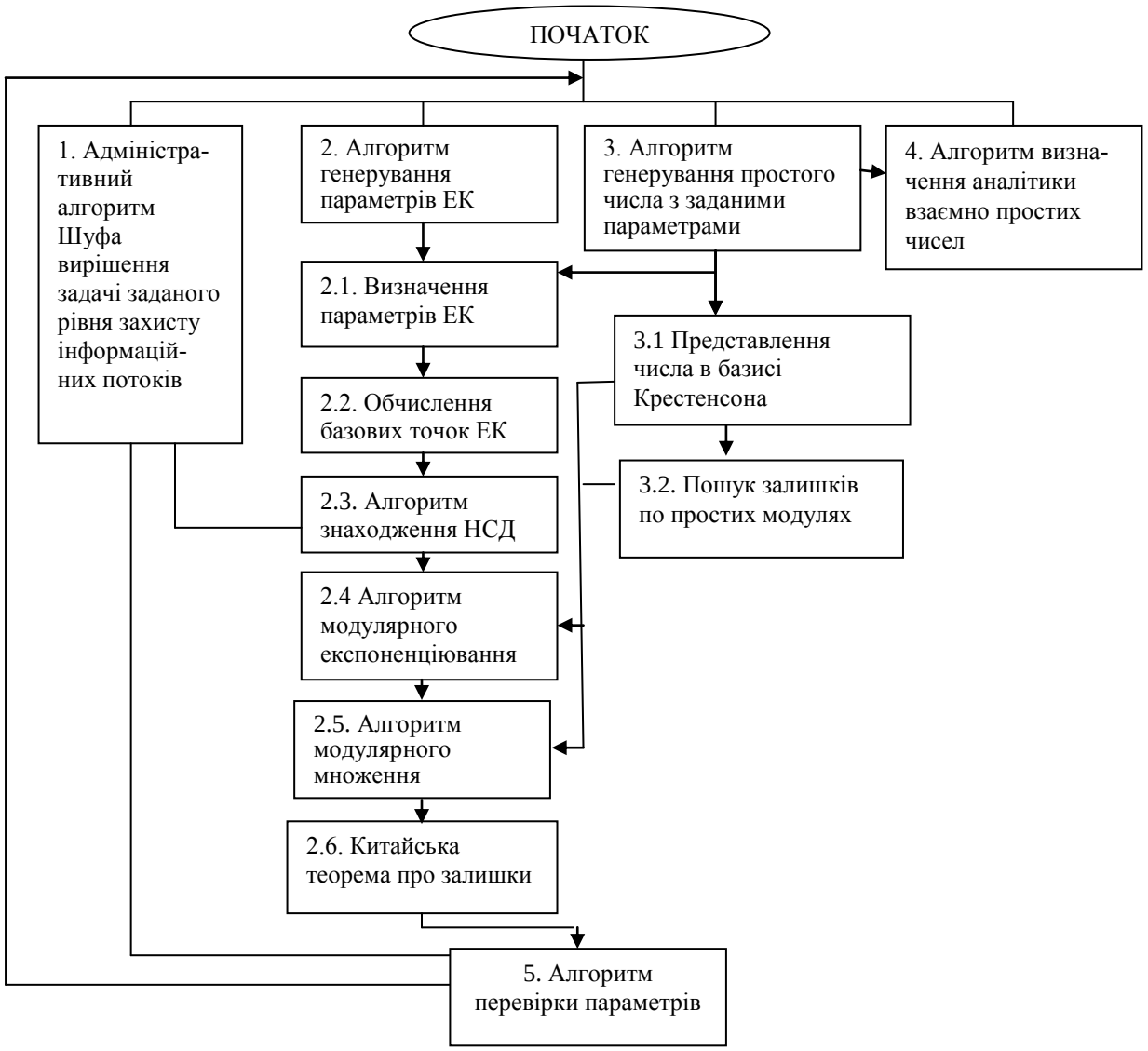


Рисунок 4.1 – Структурна організація захисту інформаційних потоків в КС на основі використання ЕК.

На рис. 4.1 програмний модуль (1) виконує адміністративні функції координації та управління виконання часових процесів компонентними програмними модулями алгоритму Шуфа. Алгоритми генерування параметрів ЕК (2) та простих чисел (3) та (4) виконуються паралельно в мультипрограмному режимі, або відповідним числом сопроцесорів. Програмні модулі (2.1-2.6) та (3.1-3.2) виконуються в конвеєрному режимі згідно їх часової координації на основі моделі мережевих та матричних моделей руху даних [17].

Для реалізації поставленого завдання було обрано мову програмування С++, а середовище програмування С++ Buider 6.0.

Перевагами мови програмування С++ є гнучкість використання, інкапсуляція, поліморфізм, структурованість. Великою перевагою є тонкість програмного коду, що дозволяє зменшити затрати процесорного часу. Велику роль при обранні мови програмування відіграла наявність зовнішніх модулів, що забезпечують виконання операцій на великими числами.

4.2 Реалізація компонентів адміністративного алгоритму Шуфа

4.3.1. Алгоритм ідентифікації простого числа з використанням базису Крестенсона

Структура програмного забезпечення даного алгоритму базується на теоретичних викладках розділу 3.2 і зображена на рис. 4.2. Основними перевагами даного алгоритму ідентифікації простого числа n є використання системи залишкових класів по всіх простих модулях до $\sqrt{n}+1$, якщо один з залишків рівний нулю, то до залишків додаємо 2 по всіх модулях, ця операція припиняється коли по одному з модулів отримаємо 0. Тоді пропускаємо числа кратні модулю, по якому залишок 0 і знаходимо послідовність простих чисел. Це дозволяє значно зменшити складність алгоритму пошуку простих чисел, за рахунок введення циклу D.

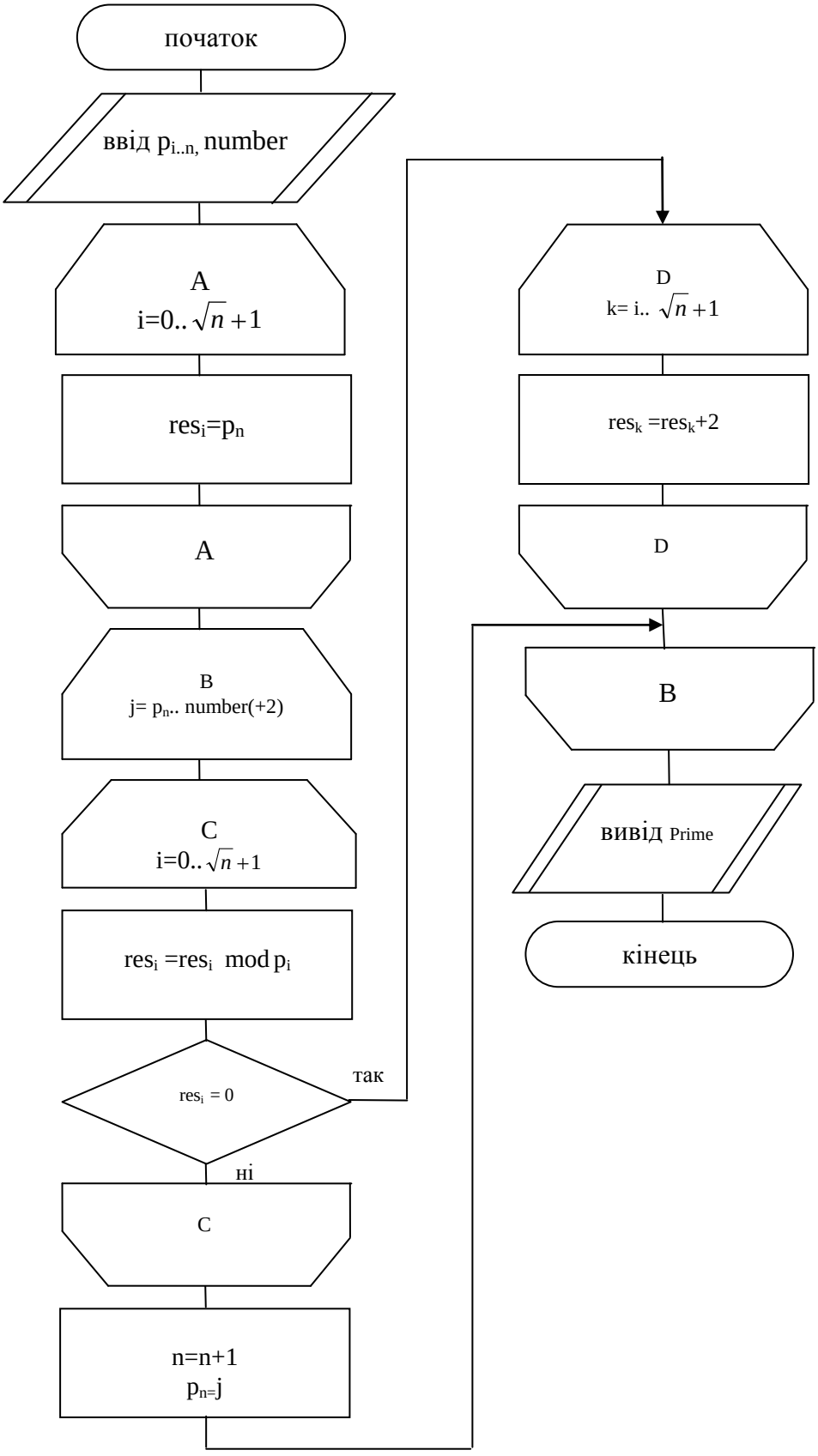


Рисунок 4.2 – Блок схема алгоритму генерування простого числа з використанням базису Крестенсона.

Особливості даної структури, яка по ефективності переважає відомі алгоритми завдяки введення блоку (3) швидкого перетворення з базису Радемахера в базис Крестенсона та модуля пошуку залишків в розмежованій системі числення.

Алгоритм дозволяє оперативно обчислювати прості числа в діапазоні до 2^{256} за 60 с, а в діапазоні 2^{512} за 180 с.

4.3.2. Алгоритм визначення простих та взаємнопростих чисел

Відомі алгоритми визначення простих та взаємно простих чисел базуються відповідно на використанні решета Еретосфена, ймовірносному тесті на простоту та алгоритму Евкліда. Функціональними обмеженнями даних алгоритмів є використання для обчислень багаторівневого базису Радемахера, які характеризуються часово складними операціями модульного ділення, множення та сумування з наскрізними переносами.

Запропонований алгоритм базується на рекурентному обчисленні залишків по заданому модулю шляхом обчислення:

$$b_{i+1} = 2 \cdot b_i (\mathrm{mod}\, p). \qquad (4.1)$$

При цьому, стартова позиція рекурентної перевірки подільності числа на прості множники визначається згідно виразу:

$$res\, 2^i (\mathrm{mod}\, p) + res \sum_{j=0}^{n} 2^j (\mathrm{mod}\, p) \equiv 0 (\mathrm{mod}\, p). \qquad (4.2)$$

Результати реалізації даного алгоритму представлено в таблиці 4.

Отримана аналітика простих чисел, які не задовольняють умову (4.2) ні по одному з простих модулів, які менші половини розрядності шуканого p, приведена в таблиці 4.2.

Таблиця 4.1 - Пошук простих чисел виду 2^n+3, розклад на прості множники.

131072		65536		32768		16384		8192		4096		2048		1024		512		256		128		64		32		16		8		4		2		1	
18		17		16		15		14		13		12		11		10		9		8		7		6		5		4		3		**2**		**1**	
5,7	1	107	1		1	7	1	5,11, 146	1		1	7	1	13, 79	1	5, 103	1	7, 37	1	131	1	67	1	5,7	1	19	1	11	1	7	1		**2**		**1**
7	1	13, 109	1	5,11	1	7, 131	1	83	1		1	5,7	1		1		1	7	1	5	1		1	7,11	1	13, 19	1	5	1	7, 59	1	29, 101	1		1
5,7, 11	1		1		1	7	1	79,5	1		1	7, 29	1	13	1	5	1	7, 37	1	11, 53	1		1	5,7	1	19, 97	1		1	7	1	5	1	61	1
7	1	13	1	5, 109	1	7	1		1		1	5,7	1		1	11	1	7, 137	1	5	1	103	1	7	1	13, 19	1	5, 131	1	7	1		1		1
5,7	1	79	1		1	7,	1	5	1		1	7, 11, 139	1	13, 131	1	5	1	7, 37	1	59	1		1	5,7,	1	19	1	29	1	7	1	5,11	1	67	1
7, 131	1	13	1	5	1	7, 109	1	11,29 53,97	1		1	5,7	1		1		1	7	1	5,83	1	61	1	7	1	13, 19	1	5,11	1	7	1		1		1
5,7	1	79	1	11	1	7	1	5	1		1	7, 101	1	13	1	5	1	7, 37	1		1		1	5,7, 11	1	19	1	103	1	7	1	5	1		1
7,11	1	13	1	5	1	7	1	109	1	67	1	5,7, 59	1		1		1	7	1	5, 11, 107	1	131	1	7,29	1	13, 19	1	5	1	7	1	79	1		1
5,7	1		1	29	1	7	1	5, 131	1	61	1	7	1	13	1	5,11	1	7, 37	1		1		1	5,7	1	19, 97	1		1	7	1	5	1		1
7	1	13	1	5	1	7	1		1	109	1	5,7, 11	1		1		1	7	1	5	1		1	7,53	1	13, 19, 79	1	5	1	7	1	11, 83	1	103	1
5,7, 139	1		1	59	1	7	1	5,11	1		1	7	1	13	1	5	1	7, 37	1	29	1		1	5,7	1	19	1	11	1	7, 131	1	5	1		1
7,29	1	13, 107	1	5,11, 103	1	7	1	97	1		1	5,7, 109, 137	1		1	131	1	7, 37	1	5, 79	1	67	1	7,11	1	13, 19	1	5	1	7	1		1		1
5,7, 11	1	131	1		1	7	1	5,83	1		1	7	1	13	1	5, 53	1	7	1	11	1		1	5,7	1	19	1	101	1	7	1	5	1	61	1
7	1	13	1	5	1	7	1		1		1	5,7	1	79, 109	1	11,29	1	7, 37	1	5	1		1	7	1	13, 19	1	5	1	7	1		1		1

Слід зазначити, що в результаті заповнення табличних даних комірки, які залишилися незаповненими, відповідають простим числам виду 2^n+3, а всі інші числа є складеними.

Слід зазначити, що на основі використання таблиці 4.1 було побудовано таблицю 4.2, яка дозволяє знаходити взаємно прості числа та вирішувати задачу факторизації чисел виду 2^n+3.

Таблиця 4.2

Аналітика простих та взаємно простих чисел.

Прості числа	Вирази виду 2^n+1, які діляться на прості числа	Вирази виду 2^n+3, які діляться на прості числа	Вирази виду 2^n+5, які діляться на прості числа	Вирази виду 2^n+11, які діляться на прості числа	Вирази виду 2^n+13, які діляться на прості числа
3	$2^{2n+1}+1$	-	-	-	-
5	$2^{4n+2}+1$	$2^{4n+1}+3$	-	-	-
7	-	$2^{3n+2}+3$	-	-	-
11	$2^{10n+5}+1$	$2^{10n+3}+3$	$2^{10n+9}+5$	-	-
13	$2^{12n+6}+1$	$2^{12n+10}+3$	$2^{12n+3}+5$	$2^{12n+2}+11$	-
17	$2^{8n+4}+1$	-	-	-	$2^{8n+3}+13$
19	$2^{18n+9}+1$	$2^{18n+4}+3$	$2^{18n+7}+5$	$2^{18n+4}+11$	$2^{18n+15}+13$
23	-	-	$2^{11n+6}+5$	$2^{11(n+1)}+11$	-
29	$2^{28n+14}+1$	$2^{28n+19}+3$	$2^{28n+8}+5$	$2^{28n+12}+11$	$2^{28n+5}+13$
31	-	-	-	-	-
37	$2^{36n+18}+1$	$2^{36n+8}+3$	$2^{36n+5}+5$	$2^{36n+13}+11$	$2^{36n+30}+13$
41	$2^{20n+10}+1$	-	$2^{20n+17}+5$	-	-
43	$2^{14n+7}+1$	-	-	$2^{14n+6}+11$	-
47	-	-	$2^{23n+9}+5$	$2^{46n+18}+11$	$2^{23n+8}+13$
53	$2^{52n+26}+1$	$2^{52n+43}+3$	$2^{52n+21}+5$	$2^{52n+32}+11$	$2^{52n+51}+13$
59	$2^{58n+29}+1$	$2^{58n+21}+3$	$2^{58n+35}+5$	$2^{58n+55}+11$	$2^{58n+17}+13$
61	$2^{60n+30}+1$	$2^{60n+36}+3$	$2^{60n+52}+5$	$2^{60n+46}+11$	$2^{60n+11}+13$
67	$2^{66n+33}+1$	$2^{66n+6}+3$	$2^{66n+48}+5$	$2^{66n+27}+11$	$2^{66n+53}+13$
71	-	-	-	$2^{35n+12}+11$	-
73	-	-	-	-	-
79	-	$2^{39n+10}+3$	-	-	-
83	-	$2^{82n+31}+3$	$2^{82n+51}+5$	$2^{82n+49}+11$-	-
89	-	-	-	$2^{11n+9}+11$	-
97	$2^{45n+24}+1$	$2^{45n+40}+3$	-	-	-
101	$2^{100n+50}+1$	$2^{100n+19}+3$	$2^{100n+74}+5$	$2^{100n+65}+11$	$2^{100n+17}+13$

4.3.3. Алгоритм модулярного множення

Алгоритм модулярного множення, блок-схема якого представлена на рис. 4.3, базується на виконанні операцій над векторами матриць (табл.2.2).

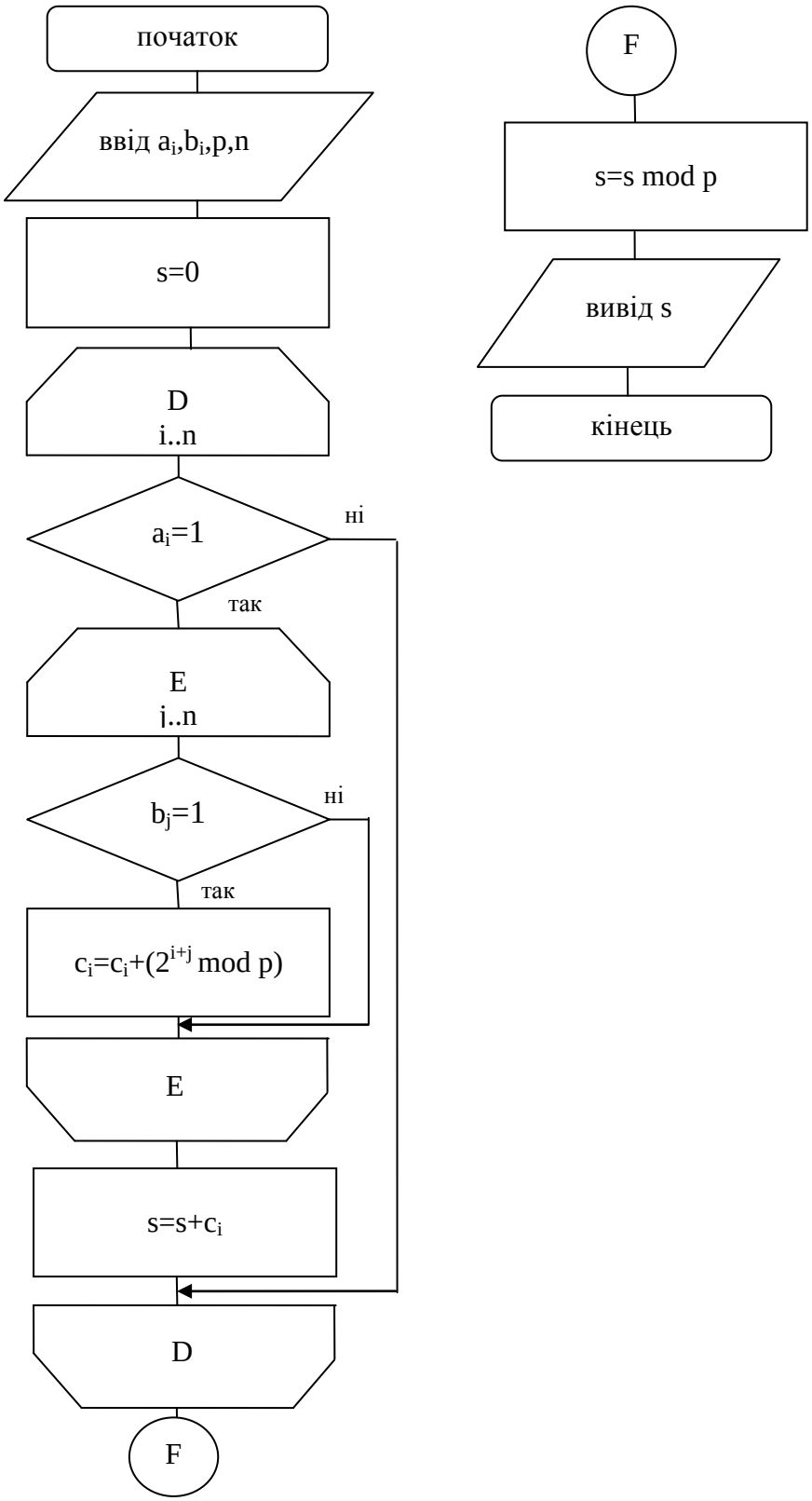


Рисунок 4.3 – Блок схема алгоритму модулярного множення з використанням розмежованої системи числення Крестенсона-Радемахера.

Основною перевагою розробленого алгоритму по відношенню до існуючих алгоритмів є заміна багататактної операції множення на однотактну операцію сумування залишків. Алгоритм дозволяє реалізувати матриці розрядності 1024 біти за 2-4 мс, що при тактовій частоті процесора 10^9 прискорює реалізацію алгоритму в 20-40 разів.

4.2.4 Алгоритм модулярного експоненціювання

Запропонований алгоритм базується на виконанні розроблених теоретичних засад модулярного експоненціювання розмежованої СЗК Радемахера-Крестенсона. Блок-схема алгоритму представлена на рис. 4.4.

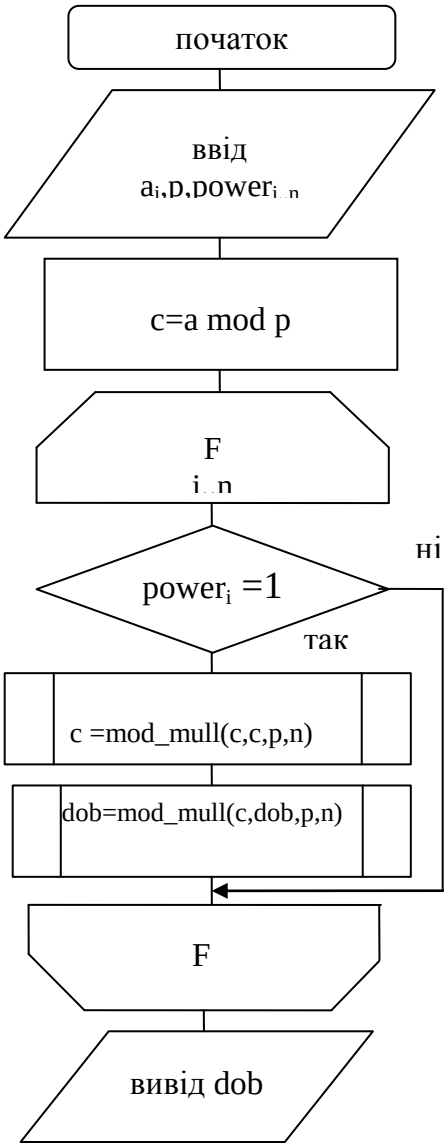


Рисунок 4.4. - Блок схема алгоритму модулярного експоненціювання з використанням розмежованої системи числення Крестенсона-Радемахера.

Відмінність та переваги запропонованого алгоритму полягають в тому, що операції піднесення до степеня виконуються на основі мультиплікативних функцій над залишками, що реалізується програмним модулем (5). Перевагою даного алгоритму є незмінність розрядності рекорентно отриманих результатів, яка не перевищує оцінку $\log_2(p-1)$. Даний алгоритм програмно реалізований для розрядності чисел 1024 біти, час виконання реалізації по відношенню до існуючих алгоритмів зменшується в 100 разів.

4.3. Реалізація програмно-апаратного забезпечення

4.3.1. Організація діалогової системи реалізації основних компонентів алгоритму Шуфа

Реалізація основних модулів, тобто пошуку простих чисел, знаходження залишку по модулю, модулярного множення, модулярного експоненціювання, найбільшого спільного дільника алгоритму Шуфа відбувається в програмному середовищі C++ Buider 6.0.

Слід зазначити, що при генеруванні простих чисел було використано високопродуктивний алгоритм 3.1. Його основна робота починається з завантаження користувачем головної форми за допомогою головного меню і надається можливість отримати доступ до основних функцій додатку. На цій формі знаходяться три вкладки:

- „Пошук простих чисел”
- „Діаграми”
- „Таблиця залишкових класів”

Кожна вкладка має специфічне функціональне навантаження. Основні можливості додатку розкриваються лише після того, як відбудеться пошук простих чисел. Для відбору простих чисел передбачено обмеження, пошук буде відбуватися до вказаного в параметрах числа.

На рисунку 4.5 показано процес пошуку простих чисел, менших від 1000000. Їх список буде відображений на екрані з вказаним часом початку пошуку та в кінці списку вказаним часом завершення пошуку.

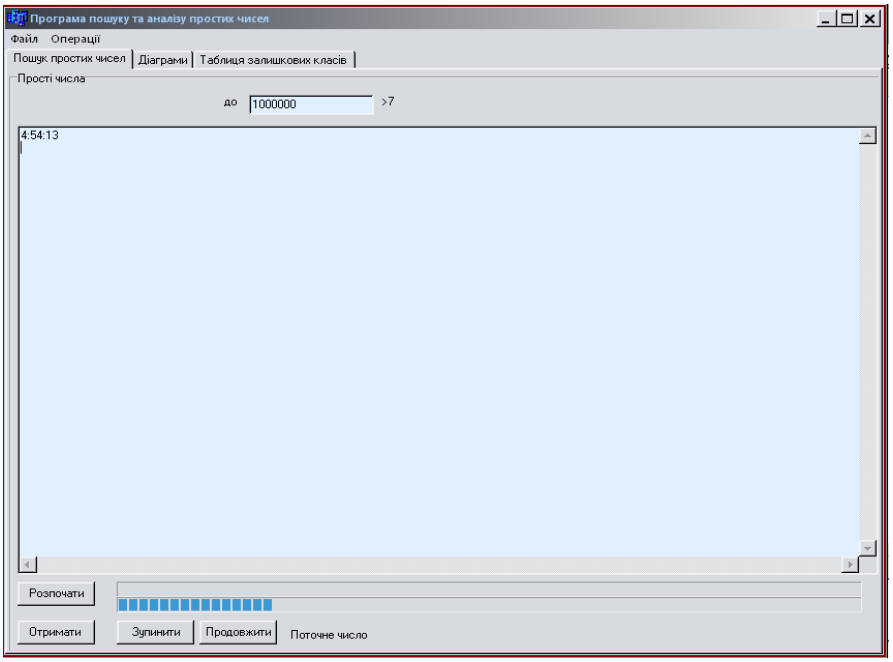


Рисунок 4.5 – Процес відбору простих чисел

Також в списку простих чисел будуть відмічені всі прості числа Мерсена та числа Ферма, які представляють особливий інтерес при побудові високопродуктивних спецпроцесорів. Після кожного десяткового представлення через пропуск відображено двійкове представлення чисел (рисунок 4.6).

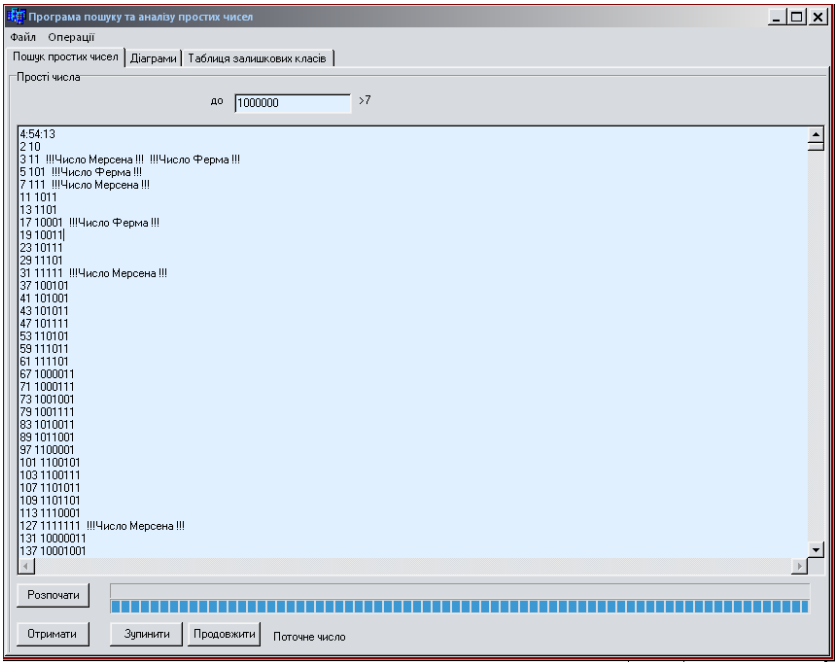


Рис. 4.6 – Результати пошуку простих чисел

Процес пошуку простих чисел відбувається з допомогою створення нового потоку, що дозволяє процесорові розподіляти час між частинами додатку, що в свою чергу забезпечує стабільну роботу, як додатку, так і операційній системі (ОС) в цілому під час обчислень.

Процес відбору простих чисел можна зупинити на певному етапі, та отримати результати пошуку, не очікуючи повного перебору заданого інтервалу. Про стан пошуку інформує також “Status bar”, який знаходиться в нижній частині, як показано на рисунку 4.6.

Вкладка „Діаграми” за функціональністю поділена на дві частини:

- „Налаштування параметрів”
- „Діаграми”

Перша вкладка дозволяє вводити критерії відбору простих чисел для побудови діаграми їх так званої швидкості зміни (рисунок 4.7).

Для кожного графіку є можливість встановлення певного кольору для графічної ідентифікації обмеженого інтервалу чисел.

Також серед простих чисел для побудови графічно відображеної залежності є можливість відбору за приналежністю до множини чисел Мерсена та Ферма. Серед специфічних функціональних можливостей є також вибір лише вказаних розрядів, що надає змогу проаналізувати їх зміну для певного виду чисел.

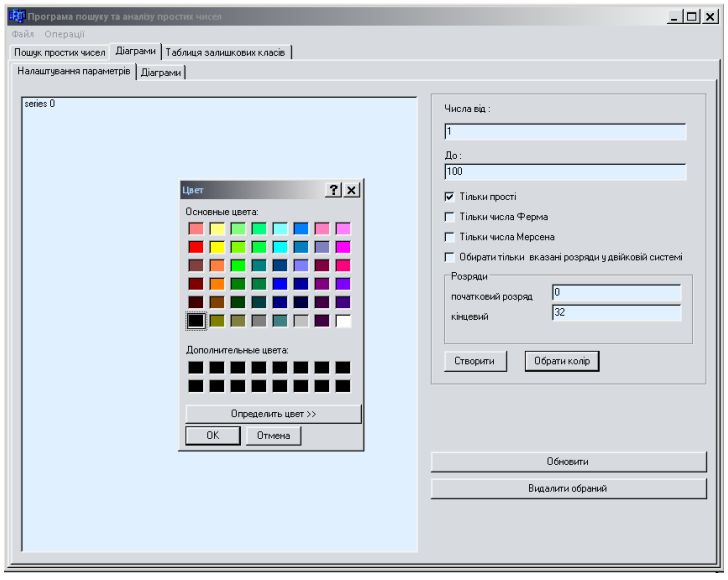


Рисунок 4.7 – Налаштування та створення діаграми швидкості зміни простих чисел та їх відбір згідно критеріїв

Також, при створенні діаграми інтервалу чисел можна виконувати операцію їхнього видалення.

На закладці діаграми – (рис. 4.8) відображаються раніше створені графіки залежностей.

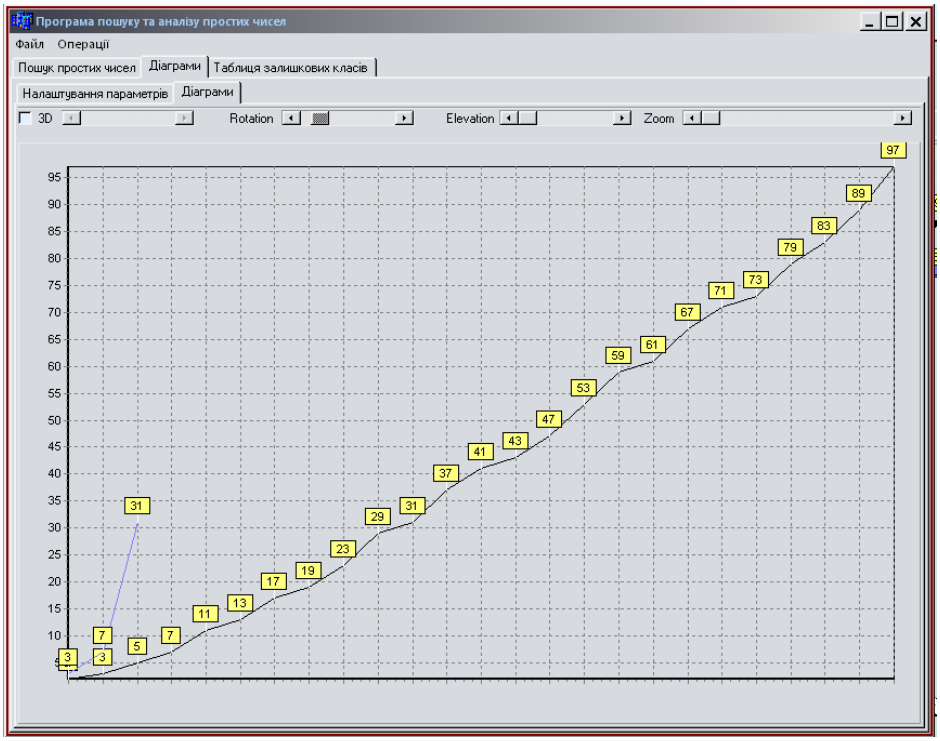
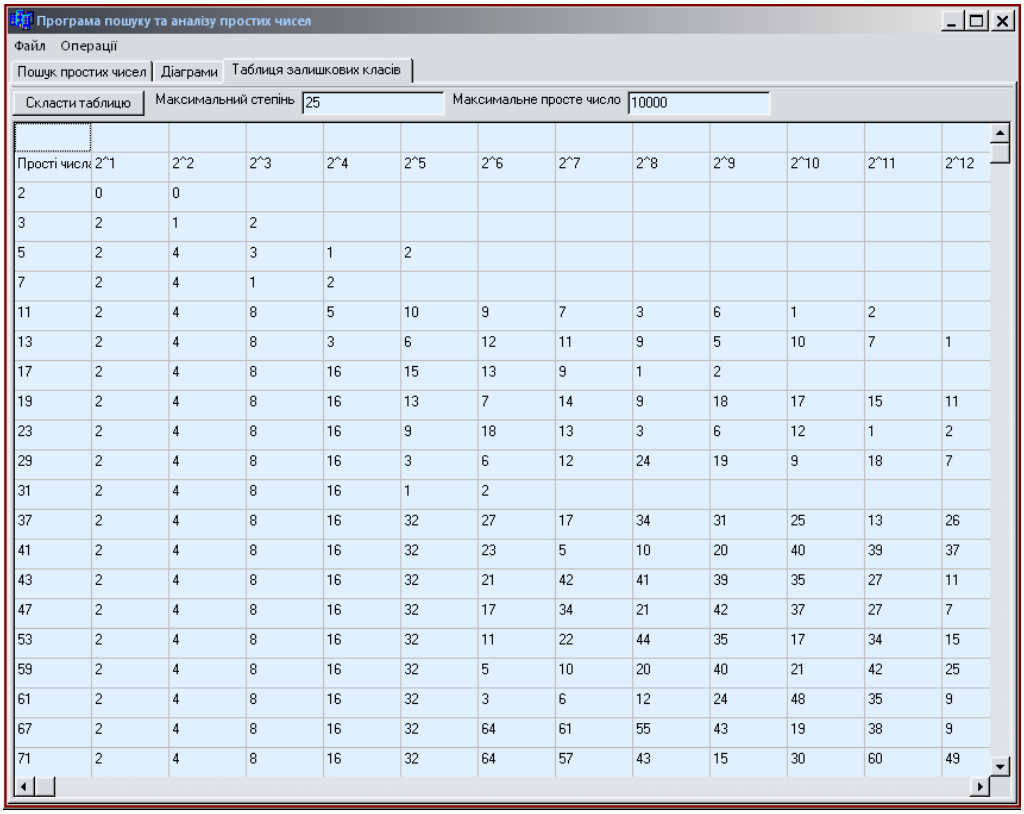


Рисунок 4.8 – Діаграма швидкості зміни простих чисел відносно чисел Мерсена

Для зручного перегляду та аналізу передбачено масштабування, 3-D перегляд, відображення під різними кутами як по вертикалі, так і по горизонталі.

На рис. 4.9 зображена форма, що містить таблицю залишкових класів, її можна побудувати лише після пошуку простих чисел оскільки вона прямо від них залежить. При побудові таблиці використовуються два обмеження: по максимальному показнику степеня і по максимальному простому числу, для якого будуть обчислюватись залишки.



Прості числа	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^10	2^11	2^12
2	0	0										
3	2	1	2									
5	2	4	3	1	2							
7	2	4	1	2								
11	2	4	8	5	10	9	7	3	6	1	2	
13	2	4	8	3	6	12	11	9	5	10	7	1
17	2	4	8	16	15	13	9	1	2			
19	2	4	8	16	13	7	14	9	18	17	15	11
23	2	4	8	16	9	18	13	3	6	12	1	2
29	2	4	8	16	3	6	12	24	19	9	18	7
31	2	4	8	16	1	2						
37	2	4	8	16	32	27	17	34	31	25	13	26
41	2	4	8	16	32	23	5	10	20	40	39	37
43	2	4	8	16	32	21	42	41	39	35	27	11
47	2	4	8	16	32	17	34	21	42	37	27	7
53	2	4	8	16	32	11	22	44	35	17	34	15
59	2	4	8	16	32	5	10	20	40	21	42	25
61	2	4	8	16	32	3	6	12	24	48	35	9
67	2	4	8	16	32	64	61	55	43	19	38	9
71	2	4	8	16	32	64	57	43	15	30	60	49

Рисунок 4.9 – Згенерована таблиця залишкових класів

Функція побудови таблиці припиняє обчислювати залишки для певного простого числа при виході за рамки вказаних умов або ж при виявленні закономірної циклічності повтору.

З головного меню програми надається можливість виконання операцій на основі модулярної арифметики з допомогою розроблених алгоритмів.

На рис. 4.10 показано приклад обчислення модуля числа з використання базису Радемахера-Крестенсона. Ця операція є досить важливою, оскільки використовується в більш складних функціях і від її часової складності та часових характеристик залежить ефективність обчислень.

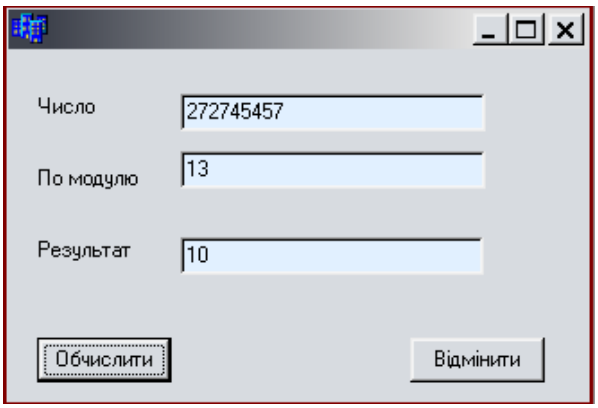


Рисунок 4.10 – Обчислення модуля числа

Приклад множення двох чисел по заданому модулю зображений на рис. 4.11. Для переведення чисел, над якими будуть виконані операції, в базис Радемахера–Крестенсона необхідно мінімум ресурсів, оскільки для цього достатньо побітно зчитати числа з пам’яті, використавши вказівники.

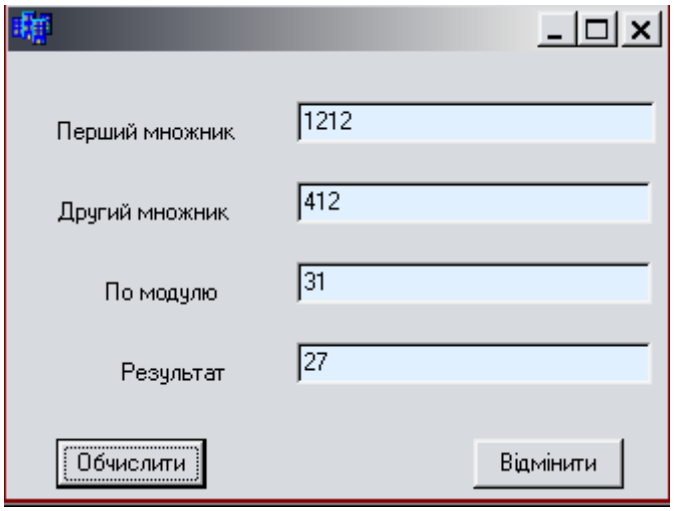


Рис. 4.11 – Обчислення залишку при модулярному множенні двох чисел

Для модулярного експоненціювання $a^x(modp)$ потрібно скористатися алгоритмом піднесення до степеня двійкового числа будь-якої розрядності за модулем p. При цьому використовується алгоритм 2.1 модулярного множення в базисі Радемахера-Крестенсона.

На рис. 4.12 показано приклад модулярного експоненціювання на основі застосування алгоритму 2.2 в розмежованій системі числення Радемахера-Крестенсона.

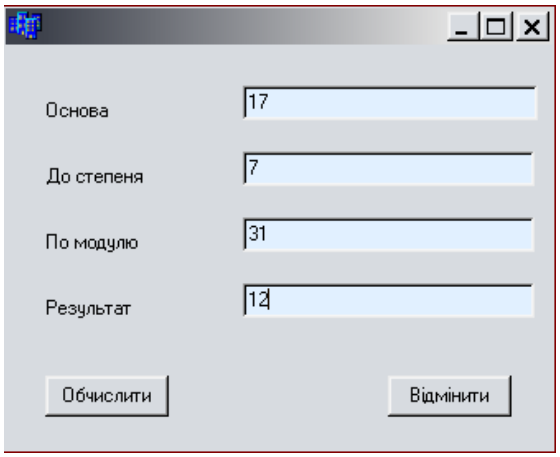


Рисунок 4.12 – Обчислення модуля числа

Рис. 4.13 наводить приклад пошуку НСД з використанням високопродуктивного алгоритму 3.3 на базі матричного методу.

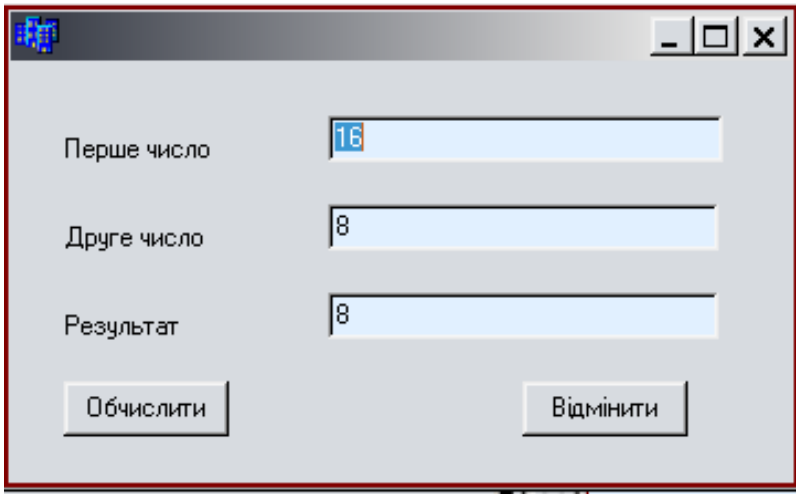


Рисунок 4.13 – Обчислення найбільшого спільного дільника.

Отже, в програмному середовищі C++ Buider 6.0 реалізовано основні компоненти, які відповідають теоретично розрахованим параметрам і підтверджують правильність та результативність запропонованого наукового підходу по вдосконаленню методів та алгоритмів опрацювання інформаційних потоків в комп'ютерних мережах за умови застосування ЕК.

4.3.2. Реалізація апаратних компонентів виконання модульних операцій в розмежованій системі.

Для отримання простих чисел, які формують таблиці подільності чисел, структурна схема процесора в розмежованій СЗК буде мати вигляд, зображений на рис. 4.5 [79].

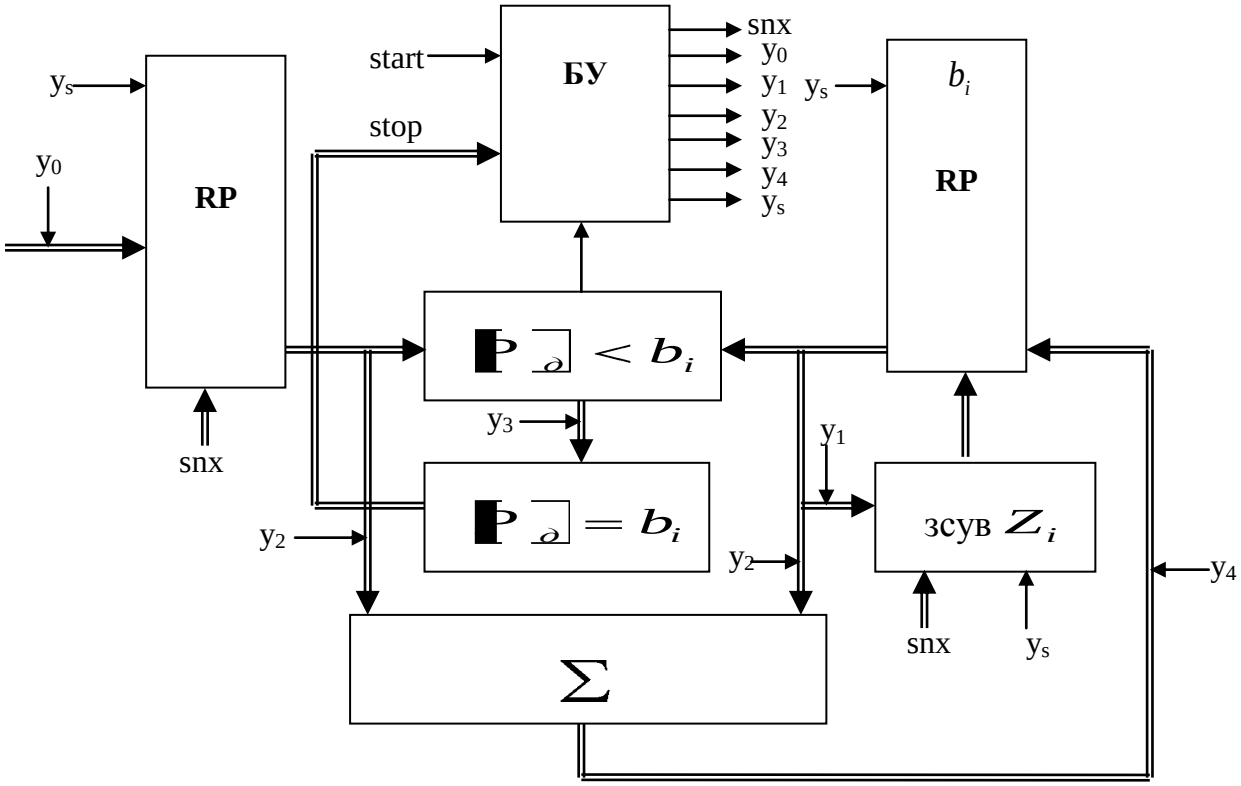


Рисунок 4.5 – Структура процесора.

Спецпроцесор складається з двох регістрів: RP, в якому заноситься модуль простого числа в доповнюючому коді $P_{(0)}$, та регістра RB, в якому формуються поточні значення залишків b_i по модулю p згідно виразу (4.2). В модульному блоці $P_{(0)} < b_i$ виконується порозрядне порівняння коду простого числа та поточного залишку. Суматор Σ виконує функції додавання доповнюючого коду $P_{(0)}$ і текучого залишку b_i. Блок управління формує синхронізуючі сигнали snx, які тактують зсуви зчитування інформації з регістрів RP і RB. Модуль $P_{(0)} = b_i$ виконує порівняння доповнюючого коду простого числа $P_{(0)} - 1 = \overline{P}$ з текучим значенням b_i. Блок управління формує управління окремими модулями спецпроцесора, які мають наступну інтерпретацію:

(start)

y_s) RP:=0, RB:=0, $Z_i = 0$;

y_0) RP:= $P_{(0)}$;

y1) $Z_i = Z_{i+1}$, $\boxed{P} > b_i$;

y2) $\Sigma := \boxed{P} + b_i$;

y3) stop:=1, $\overline{P} := b_i$.

y4) RB:= b_i +.

(stop).

Робота спецпроцесора починається сигналом start, який формується автономним тактовим генератором або сигналами системного процесора, який реалізується на базі мікроконтролера. По сигналу start згідно мікрокоманди y_s регістри RP і RB і лічильник зсувів Z_i скидаються в нульовий стан. Мікрокомандою y_0 виконується занесення коду $\boxed{P}$ в регістр RP. Під дією мікрокоманди y_1 формується число затримок зсувів в модулі Z_i до виконання умови $\boxed{P} < b_i$. Мікрокоманда y_2 дозволяє виконувати операції віднімання $P - b_i$ у вигляді сумування $b_{i+1} = \boxed{P} + b_i$. Мікрокоманда y_4 дозволяє записати отримані значення b_{i+1} в регістр RB.

У зв'язку з тим, що спецпроцесор призначений для виконання операцій над великорозрядними двійковими числами $n > 2^{10} - 2^{32}$, його реалізацію доцільно виконати на основі регістрів кристалів флеш-пам'яті, яка характеризується високою частотою порядку 1-2 ГГц та достатнім об'ємом пам'яті порядку 128-256 Гбайт.

Основними параметрами при дослідженні процесорів є апаратна складність та швидкодія. Порівняємо швидкодію запропонованого спецпроцесора з відомими аналогами, пристроями для обчислення залишку числа по модулю.

Перший пристрій складається з лічильника, лічильний вхід якого є входом в пристрій, паралельний вихід лічильника підключений до перших входів компаратора, на другі входи якого подається двійковий код модуля p, а вихід цифрового компаратора підключений до входу скидання в нуль лічильника, паралельний вихід якого є виходом пристрою. Швидкодію

такого пристрою позначимо через $V(n)$. Недоліком такого пристрою є низька швидкодія при обчисленні залишків великорозрядних чисел.

Другий пристрій містить регістр зсуву, послідовний вхід якого є входом в пристрій, виходи підключені до перших входів суматора, на другі входи якого подається доповнюючий код модуля p, виходи якого підключені до паралельних входів регістра. Функції пристрою реалізуються згідно мікрокоманд блоку управління. Недоліком пристрою є зростання розрядності суматора та регістрів при зростанні вхідного двійкового числа, а також низька швидкодія, обумовлена наскрізними переносами велико розрядного суматора. Швидкодію такого пристрою позначимо через $V1(n)$.

Відомий пристрій визначення залишку шляхом згортки по непарному модулю $P = 2^n - 1$, який мітить *n*-розрядний регістр і логічні схеми [107].

Недоліком такого пристрою є велика апаратна складність та функціональна обмеженість, що ускладнює його застосування при визначенні залишку багаторозрядних двійкових чисел з числом розрядів $n=2^{10}..2^{30}$, а також не забезпечує визначення залишку по довільному цілочисельному багаторозрядному модулю *P*.

Відомий також пристрій обчислення залишку шляхом згортки унітарного коду числа, який містить лічильник і логічні схеми [108].

Недоліком такого пристрою є низька швидкодія, що обмежує його функціональні можливості при визначенні залишку чисел великої розрядності.

Найбільш близьким за технічною суттю до запропонованого методу, є пристрій визначення залишку шляхом згортки по непарному модулю, який містить лічильник, *n*-розрядний регістр зсуву і *k*-розрядний суматор, охоплений зворотнім зв’язком, шину запису кодового представлення числа [107].

Недоліком пристрою є велика апаратна складність обумовлена наявністю багаторозрядного двійкового суматора з числом розрядів рівним

довжині коду k, багаторозрядного модуля P та обмежені функціональні можливості визначення залишку тільки по непарному модулю.

В основу запропонованого спецпроцесора поставлена задача зменшення апаратної складності та розширення функціональних можливостей пристрою обчислення залишку багаторозрядного двійкового по довільному цілочисельному багаторозрядному модулю p.

Поставлена задача вирішується завдяки тому, що згідно з корисною моделлю пристрій, який містить n-розрядний регістр зсуву, вхід якого підключений до шини запису кодового представлення числа, додатково введено шину запису кодового представлення модуля P, яка підключена до першого входу додатково введеного k+1-розрядного регістра зсуву, другий адресний вхід, якого підключений до першого виходу додатково введеного блоку управління, другий і третій виходи якого відповідно підключені до перших входів додатково введених третього і четвертого k+1-розрядних регістрів зсуву, виходи, яких відповідно підключені до першого і другого входів додатково введеного мультиплексора, третій вхід якого підключений до четвертого виходу блоку управління, а вихід підключений до першого входу додатково введеного накопичувального суматора, другий вхід якого підключений до виходу другого k+1-розрядного регістру зсуву, а вихід з'єднаний з другим входом блоку управління і додатково введеною вихідною шиною кодового представлення залишку.

Суть методу пояснюється тим, що в основу роботи пристрою покладено алгоритм обчислення залишку багаторозрядного двійкового числа Y по багаторозрядному цілочисельному модулю P згідно рекурсивного виразу:

$$b_i = [p]_{мд} + 2b_{i-1} + a_i, \ \ i = n, n-1, ..., 1, \qquad (4.3)$$

де n – розрядність числа Y, з якого визначається залишок b_i, a_i – біти двійкового числа Y, починаючи зі старшого розряду a_n, $[P]_{мд}$ – $k+1$

розрядна мантиса доповнюючого коду модуля P, b_i – текуче кодове значення залишку $(b_{i-1} = 0)$.

Спецпроцесор ілюструється кресленнями, де на рис. 4.6 показано структурну схему пристрою, де , 1 – перший *n*-розрядний регістр зсуву, 2 – шина запису кодового представлення числа *Y*, 3 – шина запису кодового представлення модуля *P*, 4 – блок управління, 5 – другий *k*+1-розрядний регістр зсуву, 6 – третій *k*+1-розрядний регістр зсуву, 7 – четвертий *k*+1-розрядний регістр зсуву, 8 – однорозрядний накопичувальний суматор, 9 – мультиплексор, 10 – вихідна шина кодового представлення залишку b_i.

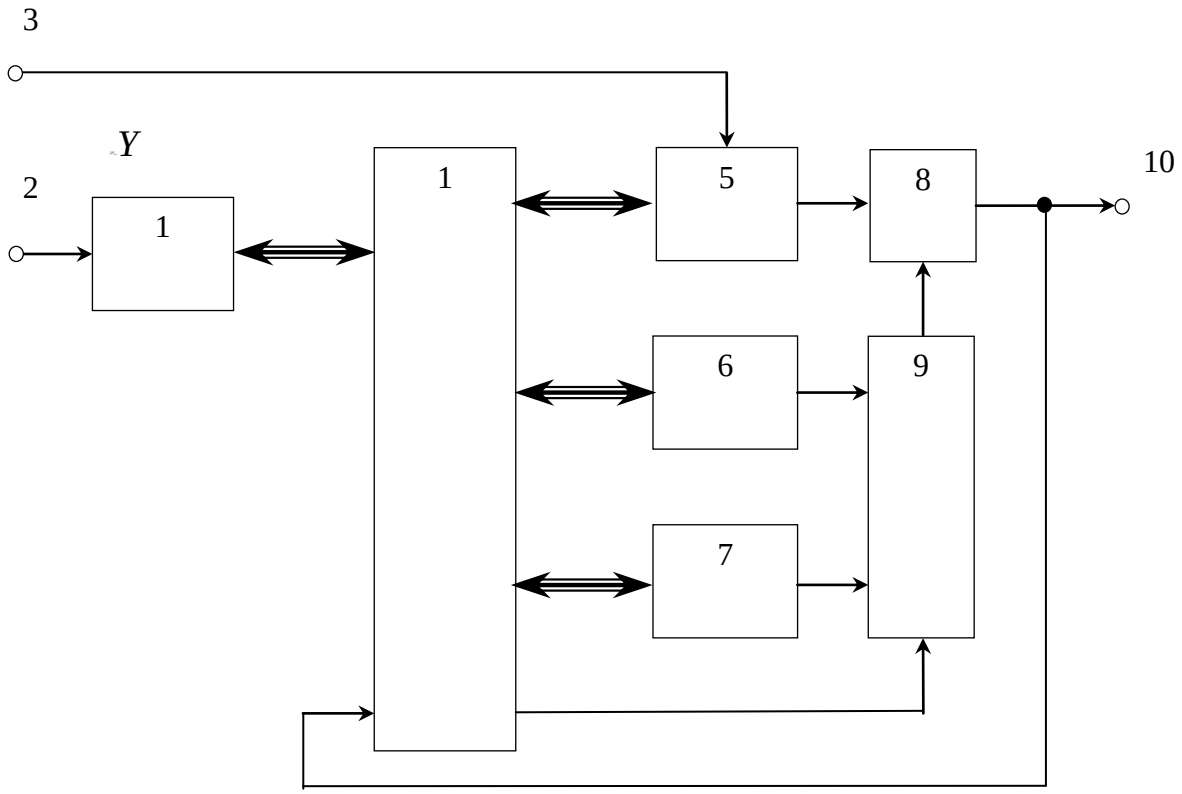


Рисунок 4.6. – Структурна схема пристрою пошуку залишків.

Пристрій працює наступним чином.

На початку циклів визначення залишку числа *Y* у перший *n*-розрядний регістр зсуву 1 і другий *k*+1-розрядний регістр зсуву 5, згідно адресних сигналів блоку управління 4 відповідно записуються двійковий код числа *Y* та мантиса доповнюючого коду модуля *P*, а в третій 6 і четвертій 7 *k*+1-розрядні регістри зсуву записуються нулі. На початку кожного наступного циклу роботи пристрою на четвертому виході блоку управління 4 формується

сигнал «1», що дозволяє зсув на один розряд в бік старших розрядів на коду залишку b_{i-1} у регістрі 6 та запис старшого біта числа Y a_i у молодший розряд третього регістра 6, що відповідає запису в цей регістр $2b_{i-1} + a_i$

В кожному текучому циклі роботи пристрою одночасно через мультиплексор 9 на входи однорозрядного накопичувального суматора 8, порозрядно зчитується код мантиси модуля $P([P]_{мд})$ розрядністю $k+1$ та код відповідного регістра 6 або 7 . При цьому одночасно відбувається запис нового залишку b_i у відповідний регістр 7 або 6 згідно адресних входів блоку управління 4.

У результаті на протязі $k+1$ такту у однорозрядному накопичувальному суматорі 8 порозрядно формується сума кодів згідно виразу (1), яка завершується формуванням на виході суматора 8 останнього біта «0» або «1», значення якого поступає на другий вхід блоку управління 4. При цьому, якщо вказаний біт є «1», то це означає, що текучий залишок b_i у регістрі 6 менший значення модуля p ($b_i < P$) і відбувається зсув інформації в регістрі 6 на один розряд в бік старших розрядів, а в молодший розряд записується наступний молодший біт числа Y. При формуванні на виході суматора сигналу «0» , це означає, що $b_i \geq P$ і на вхід мультиплексора 9 подається сигнал «0», який формується на четвертому виході блока управління 4 і починається новий цикл сумування у суматорі 8 кодів $[P]_{мд}$ другого регістра 5 та четвертого регістра 7, та запис інформації у регістр 6 до появи біта «0» в кінці k+1 такту на виході суматора 8. У результаті виконується попередній цикл.

Після зчитування останнього молодшого біта числа Y a_1 в одному з регістрів 6 або 7 формується код залишку b_1, який зчитується через мультиплексор 9 і суматор 8 на вихідну шину 10 пристрою, при цьому на виході другого регістра 5 формується сигнал «0».

Приклад, визначення залишку по непарному модулю:

Припустимо Y=$100_{(10)}$=$1100100_{(2)}$, P=$11_{(10)}$=1011.

Тоді $b_1 = res100(\mathrm{mod}\,11) = 1$.

Потрібно визначити b_1 над двійковими кодами *Y* та *P*:

1. У перший регістр 1 записуємо число $Y=1100100_{(2)}$

2. У другий регістр 5 записуємо $k+1$ розрядну мантису доповнюючого коду $[P]_{мд}=10101$, яку отримуємо наступним чином:

- код числа *P* записуємо з нульовим бітом у старшому розряді: $P=01011_{(2)}$
- інвертуємо цей код: $\overline{P}=10100$
- додаємо до цього коду «1»:

$$[P]_{мд}=\begin{array}{r}10100\\ +1\\ \hline 10101\end{array}.$$

3. У третій регістр 6 і четвертій 7 регістри записуємо нулі, тобто $b_i=00000\ \ b_{i-1}=00000\ ; i=n,n-1,...,1$.

4. Блок управління 4 формує біт «0»на керуючий вхід мультиплексора 9, що дозволяє відповідно записувати та зчитувати інформацію з другого регістра 5 і третього регістра 6(b_i)та четвертого регістра 7(b_{i-1}) та записувати коди порозрядно вихідні коди суматора .у четвертій регістр 7.

5. В кожному циклі роботи пристрою виконується сумування мантиси доповнюючого коду модуля *P* з значенням текучого залишку b_{i-m}. У нашому випадку має місце така операція

$$\begin{array}{ll}[p]_{мд} & 10101\\ 2b_i+a_i & {}^{+}00001 \qquad i=n-1\\ \hline [b_{i-m+1}]_{мд} & 11110\end{array}$$

6. Отримуємо мантису доповнюючого коду цієї операції додавання. Код цієї мантиси записуємо у регістр 6(b_i) або 7(b_{i-1}). Отримане значення «1» в $k+1$ такті сумування показує, що $b_{i-1}<P$. Тоді відбувається зсув інформації у відповідному регістрі 6 або 7 і запис у молодший розряд a_{i-1} біта числа *Y*, тобто

$$P > b_i \qquad \begin{array}{r} 10101 \\ +00011 \\ \hline 11100 \end{array} \qquad i = n-2$$

$$P > b_i \qquad \begin{array}{r} 10101 \\ +00110 \\ \hline 11011 \end{array} \qquad i = n-3$$

$$P \le b_i \qquad \begin{array}{r} 10101 \\ +01100 \\ \hline 00011 \end{array} \qquad i = n-4$$

7. Отриманий біт «0» поступає в блок управління, який запам'ятовує його і переключає мультиплексор 9, що приводить до подвоєння коду залишку в регістрі 7(b_{i-1}) та запису в молодший розряд текучого біта числа *Y*.

8. Після цього використовується операція сумування згідно розрахунків:

$$\begin{array}{rrr} 10101 & 10101 & 10101 \\ +00011 & +00110 & +01100 \\ \hline 10100 & 11011 & 00001 = b_1 \end{array}$$

Тобто $b_1 = 00001_{(2)} = 1_{(10)}$, що відповідає $b_1 = res100(\mathrm{mod}\,11) = 1$.

Аналогічні розрахунки можна виконати при визначенні залишку по парному модулю:

Припустимо $Y=25_{(10)}=11001_{(2)}$, $p=6_{(10)}=110$.

Запишемо $[P]_{мод} = 0110 \xrightarrow{\bar{p}} 1001 \xrightarrow{+1} 1010$.

Виконаємо наступні операції:

$$\begin{array}{rrr} 1010 & 1010 & 1010 \\ +0001 & +0011 & +0110 \\ \hline 1011 & 1101 & 0000 \end{array}$$

Оскільки в старшому розряді «0», то цей залишок зсувається на біт і додається новий біт числа:

$$\begin{array}{rr} 1010 & 1010 \\ +0000 & +0001 = b_1 \\ \hline 1010 & 1011 \end{array}$$

Оскільки використані всі біти числа Y і $b_i < P$, то $b_1 = 0001_{(2)}$, що відповідає $b_1 = res25(\text{mod}6) = 1$.

При реалізації пристрою доцільно в якості регістрів використати багаторозрядну флеш-пам'ять.

В основу запропонованого спецпроцесора входять два регістра на основі флеш-памяті, два цифрових компаратори, суматор, регістра зсуву та блоку управління (табл.4.3). Швидкодію запропонованого спец процесора позначимо через $V2(n)$.

Таблиця 4.3 – Характеристика спецпроцесора пошуку залишків

№	Компоненти	Кількість, шт.	Коефіцієнт апартної складності C_A, експлуат. один.	Кбіт
1	Флеш-память	2	10	1
2	Цифровий компаратор	2	1	0,01
3	Суматор	1	2	0,01
4	Блок управління	1	5	0,1
5	Регістр зсуву	1	3	0,03

Виходячи з вищенаведених характеристик пристроїв пошуку залишків по модулю, на (4.3)-(4.5) приведено аналітичні вирази швидкодії розглянутих пристроїв:

$$V(n) = \frac{1}{2 \cdot n}, \tag{4.3}$$

де $V(n)$ - кількість залишків з розрядністю n процесор шукає в унітарному коді за 1 с.

$$V1(n) = 2^{\log_2 p - n}, \tag{4.4}$$

де $V1(n)$ - кількість залишків з розрядністю n процесор шукає в двійковому коді за 1 секунду, p - модуль.

$$V2(n) = 2^{40-2n}, \tag{4.5}$$

де $V2(n)$ – кількість залишків з розрядністю n процесор шукає в базисі Крестенсона за 1 секунду, $i = 5\ldots9$.

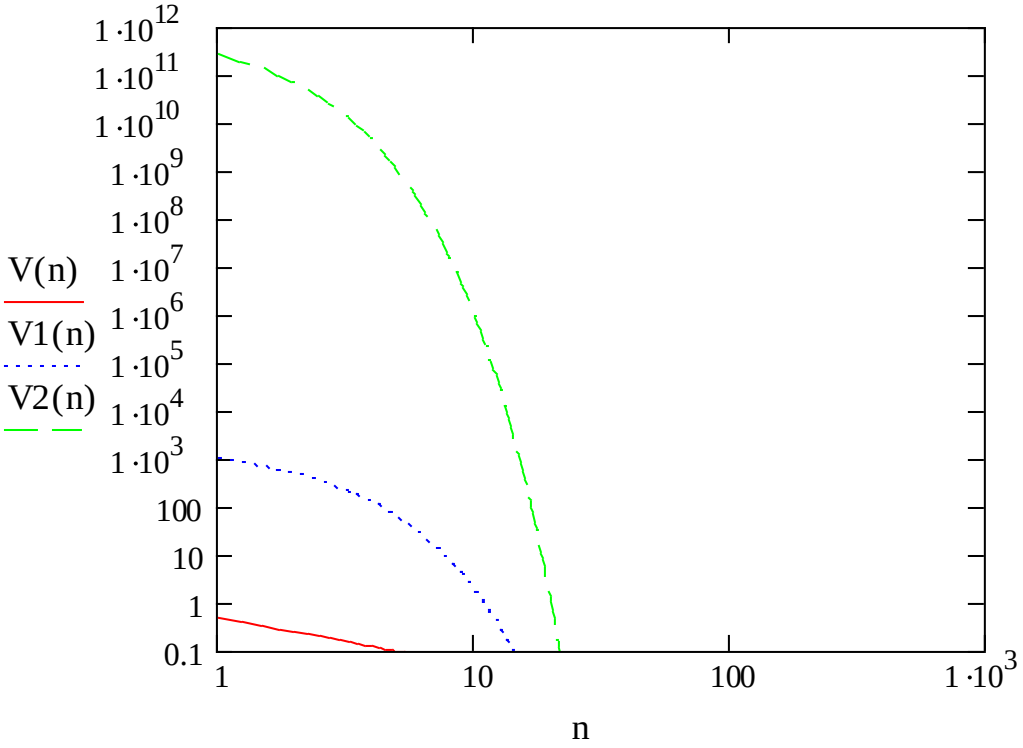


Рисунок 4.6 – Швидкодія розглянутих пристроїв пошуку залишків по модулю

З результатів дослідження видно, що запропонований спецпроцесор дозволяє знаходити найбільшу кількість залишків за 1 с в порівнянні з відомими аналогами.

При дослідженні апаратної складності запропонованого спецпроцесора слід врахувати дані табл. 4.3. Тоді апаратна складність буде обчислюватися згідно наступного співвідношення:

$$P = F(Ф + ЦК + \sum + РЗ + БУ), \tag{4.6}$$

де Ф - апаратна складність флеш-пам’яті,

ЦК - апаратна складність цифрового компаратора;

$\sum$ - апаратна складність суматора;

РЗ – апаратна складність регістру зсуву;

БУ – апаратна складність блоку управління.

Оскільки ЦК+$\sum$+РЗ+БУ= C_A=const, тоді загальна апаратна складність з врахуванням коефіцієнту апаратної складності флеш-памяті буде:

$$P(n) = F \cdot \log_2 n + C_A, \tag{4.7}$$

де n - розрядність числа.

На рис. 4.7 подано результати дослідження при використанні елементної бази різних виробників.

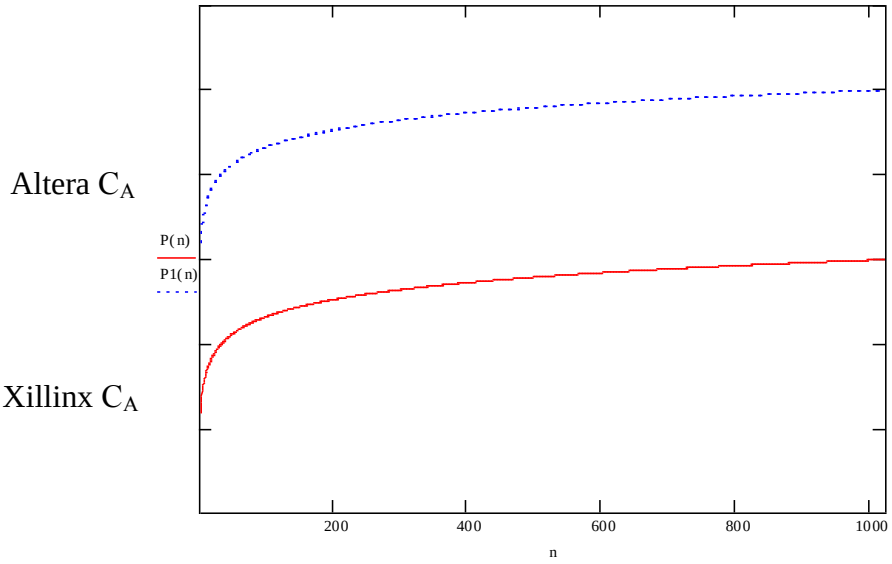


Рисунок 4.7 – Апаратна складність запропонованого методу при використанні елементної бази різних виробників.

Результати чисельного експерименту показали, що використання елементної бази фірми Xilinx при проектуванні спецпроцессора пошуку залишків по заданному модулю дозволяє зменшити апаратну складність на 20% по відношенню з елементною базою фірми Altera.

Реалізація спецпроцесора пошуку залишку великорозрядного числа по модулю виконана на базі стендової платформи IOG-PCIC3-033D-1 структурна схема представлена рис.4.8.

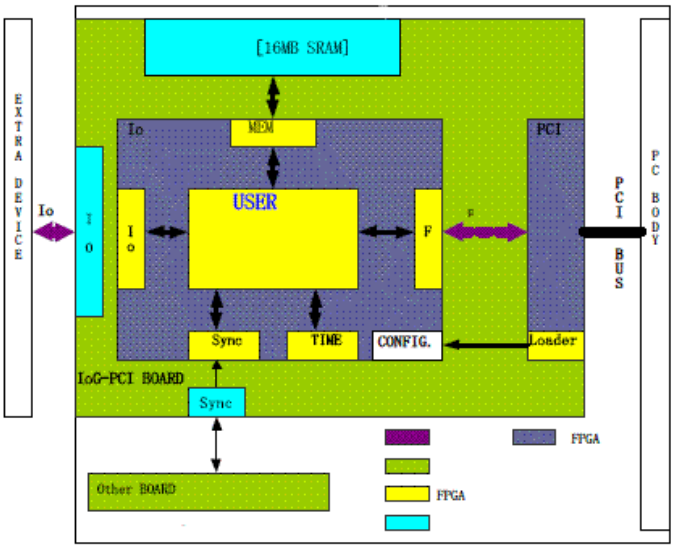


Рисунок 4.8 – Структурна схема платформи IOG-PCIC3-033D-1.

Платформа оснащена модулем інтерфейсу PCI, що дозволяє імплементовувати її в існуючі обчислювальні ситеми, як окремий елемент. Особливістю даної платформи є сумісне використання ПЛІС XilinxSpartanIII та AlteraCyclone, які мають доступ до спільної оперативної памяті SRAM розміром 16M, що дозволяє провети дослідження та аналіз системних характеристик спецпроцесора на різних кристал.

Зовнішній вигляд платформи IOG-PCIC3-033D-1 представлено на рис.4.9.

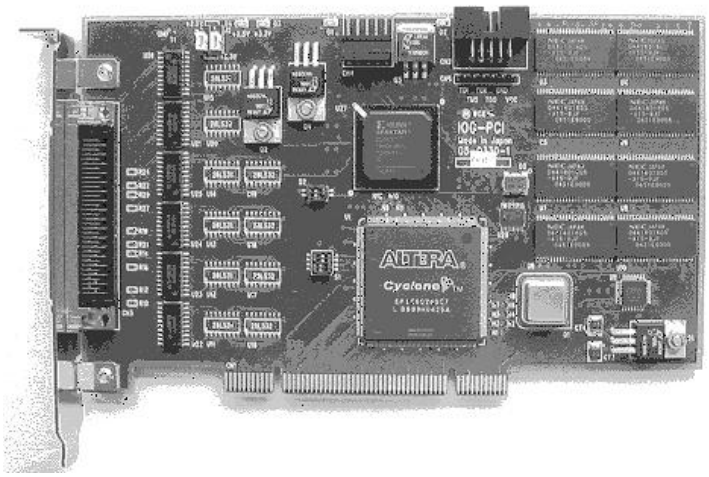


Рис. 4.9 – Зовнішній вигляд платформи IOG-PCIC3-033D-1

ВИСНОВКИ ДО РОЗДІЛУ 4

1. На основі запропонованих методів обчислень у базисі Крестенсона та алгоритмів захисту інформаційних потоків з використанням ЕК для КС розроблена базова структура взаємодії компонентів часових операцій алгоритму Шуфа з врахуванням розпаралелення та конвеєрного опрацювання інформаційних потоків, які характеризуються зменшеною часовою складністю, розширеними функціональними можливостями захисту інформаційних потоків по відношенню до відомих алгоритмів аналогічного класу.

2. Розроблено інструментальні програмні засоби пошуку параметрів ЕК, НСД двох чисел, простих та взаємнопростих чисел великої розрядності, алгоритмів модулярного множення та експоненціювання, а також дослідження впливу часової складності на характеристики запропонованих методів.

3. Реалізовано на базі C++ основні компоненти, які відповідають теоретично розрахованим параметрам і підтверджують правильність та результативність запропонованого наукового підходу по вдосконаленню методів та алгоритмів опрацювання інформаційних потоків в комп'ютерних мережах за умови застосування ЕК.

4. На базі розроблених теоретичних положень, результатів досліджень та моделювання розроблено пакет прикладних програм «YN-2010», який переданий для тестування на реалізації на низових рівнях спеціалізованих РКС ВАТ ТКБР «Стріла».

ЗАГАЛЬНІ ВИСНОВКИ

У дисертаційній роботі розв’язано наукову задачу розробки методів та алгоритмів опрацювання інформаційних потоків у КМ за умови використання ЕК. При цьому отримані наступні результати:

1. Проаналізовані і досліджені архітектури та трафіки КМ на основі оцінки їх емерджентності. Показано, що найбільш перспективною архітектурою мережевих технологій опрацювання інформаційних потоків є зірково-магістральна.

2. Виконано систематизацію системних об’єктів та аналітичних моделей комп’ютерних систем. Обґрунтовано ефективність застосування ТЧБ Радемахера та Крестенсона для зменшення часової складності формування ІП та захисту від несанкціонованого доступу.

3. Розроблено метод модулярного множення та експоненціювання з використанням матрично-модульних перетворень ТЧБ Радемахера-Крестенсона.

На відміну від відомих, які побудовані на швидкому перетворенні Фур’є та на основі десяткової системи числення, ефективність запропонованого методу при розрядності чисел до 64 бітів стрімко зростає на 1 порядок за рахунок заміни багататактної операції множення у базисі Радемахера матрично-модульною операцією сумування в базисі Крестенсона і на 20% при n>64біт.

4. Розроблено метод та теоретичні основи матрично-модульних перетворень з використанням системи числення залишкових класів Крестенсона, ТЧБ Крестенсона-Радемахера та критерії їх ефективного застосування в алгоритмі Шуфа.

5. Отримано аналітичні вирази характеристик складності формування та опрацювання ІП за умови застосування ЕК, які склали теоретичну основу спрощення часових компонентів алгоритму Шуфа та дозволили експоненційну складність привести до лінійної або квадратичної складності.

6. Побудовано та досліджено нові швидкі перетворення базису Радемахера в базис Крестенсона, на основі застосування системи числення

залишкових класів, ТЧБ Крестенсона-Радемахера та розмежованої системи числення Крестенсона-Радемахера розроблено алгоритми пошуку НСД двох чисел, простого числа, елементів Китайської теореми про залишки, які забезпечують підвищення швидкодії та зниження часової складності на 1-2 порядки.

7. Розроблено та реалізовано програмно-апаратні засоби опрацювання біторієнтованих ІП, які характеризуються великою розрядністю та лінійно-квадратичною складністю в порівнянні з експоненційною для існуючих алгоритмів.

8. На базі розроблених теоретичних положень, результатів досліджень та моделювання розроблено пакет прикладних програм «YN-2010», який переданий для тестування на реалізації на низових рівнях спеціалізованих РКС ВАТ ТКБР «Стріла».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Якименко І.З. Підвищення ефективності обчислення точок на еліптичних кривих над обмеженими полями.// І.З.Якименко, А.О.Ботюк, М.П.Карпінський/ Вісник Тернопільського державного технічного університету імені Ів.Пулюя, - Том 8, - №4 – 2003. – С: 67 – 73.

2. Карпінський М.П. Еліптична крива для асиметричної криптографічної системи. // М.П.Карпінський, І.З.Якименко, І. Дуда /Вісник Тернопільського державного технічного університету імені Ів.Пулюя, - Том 6, - №3 – 2001. – С: 91 – 95.

3. Алексеев В.Е. Основы информационных технологий. Графы и алгоритмы.// В.Е.Алексеев / Структуры данных. Модели вычислений.: М.: BHV, 2006. – 320 с.

4. X9.62-1998 [2]. Public Key Cryptography For The Financial Services Industry. Public Key Cryptography For The Financial Services Industry.

5. Столлингс В. Беспроводные линии связи и сети.: Пер. с англ./ В.Столлингс / – М.: Издательський дом «Вильямс», 2003. – 640 с.

6. Ахмед Н. Ортогональные преобразования при обработке цифровых сигналов./ Н.Ахмед, К.Рао / – М.: Связь, 1980. – 247с.

7. Карпінський М.П. Використання символів Якобі в криптографії ЕК // М.П.Карпінський, І.З.Якименко, А.А.Хомінчук / ДУІКТ/06/ Матеріали ІІІ Міжнародної науково-технічної конференції "Світ інформації та телекомунікацій-2006" Україна.– Київ. – 2006. – С.208.

8. М.Карпінський Метод генерування параметрів еліптичних кривих. // М.Карпінський, І.Васильцов, І.Якименко, Я.Кінах./ Правове, нормативне, та метрологічне забезпечення системи захисту інформації в Україні. К.:–2003– № 6.– С.74.

9. Блейхаут Р. Быстрые алгоритмы цифровой обработки сигналов. // Блейхаут Р./– М.: Мир. 1989. – 448 с.

10. Гайворонская Г.С. Модель синтеза оптимальной структуры телекоммуникационной сети // Г.С. Гайворонская / Захист інформації. – 2006. – №4. – С. 78-84.

11. Гайворонская Г.С. Определение оптимальной стратегии эволюции телекоммуникационной сети методом нелинейного программирования // Г.С. Гайворонская / Тр. VI междунар. науч.-практ. конф. «ССПОИ». – Одесса: ОНАС. – 2002. – С. 72-73.

12. Бессалов А.В. Криптосистемы на эллиптических кривых: Учеб. пособие.// А.В.Бессалов, А.Б.Телиженко / – К.: ІВЦ «Видавництво «Політехніка»».– 2004. – 224 С.

13. К. Айерлэнд Классическое введение в современную теорию чисел.// К. Айерлэнд, М. Роузен/ — М.: Мир.– 1987.

14. IEEE P1363 / D8(Draft Version 8). Standard Specifications for Public Key Cryptography.

15. Возна Н.Я. Дослідження ефективності розподілених інформаційних систем на основі епюр собівартості циклів руху даних Наукові вісті інституту менеджменту та економіки «Галицька академія».// Н.Я.Возна / – Івано-Франківськ. – 2006. – №2(10). – С. 74-78.

16. Возна Н.Я. Метод формування структуризованих даних та інформаційна технологія побудови сукупності моделей та епюр руху даних в багаторівневих комп'ютеризованих системах. // Н.Я.Возна, Я.М. Николайчук / – Поступ в науку. Збірник наукових праць Бучацького інституту менеджменту і аудиту. – Бучач. – 2008. – №4.Т.2. – С. 124-132.

17. Николайчук Я.М. Теорія джерел інформації./ Я.М. Николайчук / – Тернопіль: ТНЕУ, 2008. – 536 с.

18. Таненбаум Э. Компьютерные сети. 4-е изд./ Э. Таненбаум / – СПб.: Питер, 2003. – 992с.

19. Гриценко В.И. Распределенные информационные системы. Состояния. Перспективы развития // В.И.Гриценко, А.А. Урсатьев/ Управляющие системы и машины.– 2003.–№4.– с 11-21.

20. Литвин А.И. Организация векторных вычислений спектральных коэффициентов преобразования Хаара // А.И.Литвин, А.И. Май, Л.А.Писаренко / Тезисы докладов международной конференции по вычислительной математике (МКВМ-2002). – Новосибирск. – 2002. – С.46-58.

21. Дейтел Х.М. Операционные системы. Распределенные системы, сети, безопасность./ Х.М.Дейтел, П.Дж.Дейтел, Д.Р.Чофнес / - М.: БИНОМ.– 2006. – 704 с.

22. Левицький А.О. Метод формування повідомлень на основі інтегрально-імпульсних моделей.// А.О. Левицький/ BISTRO/96/ 0.52. Матеріали 2-ї Міжнародної науково-практичної конференції «Управління енерговикористанням». – Львів. – 1997. – С.36-39.

23. Дивак М.П. Властивості інтервальних моделей при інтервальній формі їх параметрів // М.П. Дивак/ Сб. науч. тр. международного науч.–учеб. центра информ. технологий и систем, науч. совет НАН Украины по пробл. „Кибернетика”. Моделирование и управление состоянием эколого–экономических систем региона.– К.–2001.–С.58–63.

24. А.А. Болотов Алгоритмические основы эллиптической криптографии./ А.А. Болотов, С.Б. Гашков, А.Б. Фролов, А.А. Часовских./ – Москва:МЭИ.– 2000. – 100 с.

25. Ахо А. Построение и анаализ вычислительных алгоритмов./ А.Ахо, Дж.Хопкрофт, Дж.Ульман / – М.: Мир.– 1979. - С. 37 - 40.

26. Дунець Р.Б. Аналіз та синтез топологій комп’ютерних видавничо–поліграфічних систем./ Р.Б.Дунець / – Львів: НВФ „Українські технології”.– 2003. – 192с.

27. Катренко А.В. Системний аналіз об’єктів та процесів комп’ютеризації: Навч. посібник. // А.В.Катренко /–Львів: «Новий світ – 2000».– 2001. – 424 с.

28. Варновский Н. П. Криптография и теория сложности // Математическое просвещение. / Н. П. Варновский / – М:1998. - №2. - С. 71 - 86

29. Николайчук Я.М. Разработка теории и комплексов технических средств формирования, передачи и обработки цифровых сообщений в низовых вычислительных сетях автоматизированных систем: Дис. доктор. техн. наук./ Я.М. Николайчук/ – К.: Академия наук УССР Ордена Ленина Институт кибернетики им. В.М.Глушкова.– 1991: –573 с.

30. Корнилов А.И. Принципы построения модулярных индексных умножителей // А.И. Корнилов, М.Ю.Семенов, О.В.Ласточкин / Известия ВУЗов. Электроника. – 2004. - №2. – С.48-55.

35. Левицький А.О. Метод формування повідомлень на основі інтегрально-імпульсних моделей. // А.О.Левицький / BISTRO/96/052 Матеріали 2-ї Міжнар. наук.-практ. конф. «Управління енерговикористанням». – Львів. – 1997. – С.36-39.

36. А.В.Потий Системапоказателей оценки эффективности функционирования схем поточного шифрования / А.В.Потий, Ю.А.Избенко // Радіотехніка:Всеукр. міжвід. наук.-техн. зб.– 2003. – №134.– с.49-61.

37. Головко, В.А. Проектирование интеллектуальных систем обнаружения аномалий // В.А. Головко, С.В.Безобразов // Международная научно-техническая конференция «Open Semantic Technologies for Intelligent Systems (OSTIS-2011) – Minsk, Belarus. 2011. – P. 70–74.

38. Локазюк В.М. Проблеми та методологія контролю і діагностування сучасних мікропроцесорних пристроїв та систем // В.М. Локазюк / Вимірювальна та часова техніка в технологічних процесах.– 2000.- № 2.–с.10–17.

39. Локазюк В.М. Контроль і діагностування часових пристроїв та систем : Навч. посібник для вузів.// В.М. Локазюк /– Хмельницький : ТУП.– 2001.–242 с.

40. Головко, В.А. Нейронная сеть как элемент системы обнаружения вторжений // В.А. Головко, Л.Ю. Войцехович // X Всероссийская научно-

техническая конференция «Нейроинформатика-2008»: сборник научных трудов в 2-х частях. – Москва: Московский инженерно-физический институт, 2008. – Ч.2.– С.86–93.

41. В.И. Гриценко Модель распределенной информационной системы широкого применения / В.И. Гриценко, Е.А. Котиков, А.А. Урсатьев и др.// УСиМ.-1999.-№5- С.32.-42.

42. Schoof R. Elliptic curves over finite fields and the computation of square roots modulo p.// R. Schoof / –Bordeaux: Math. Comput. –1985.–№ 44.– 483– 494p.

43. Schoof R. Counting points on elliptic curves over finite fields. J. The´or. Nombres. // R. Schoof / – Bordeaux 7.– 1995.– 219 –254p.

44. ATKIN A. O. L. 1986a. Schoof’s algorithm. Manuscript.

45. ELKIES N. D. Elliptic and modular curves over finite fields and related computational issues. In Computational Perspectives on Number Theory: Proceedings of a Conference in Honor of A. O. L. Atkins, D. A. Buell and J. T. Teitelbaum, eds. AMS/IP Studies in Advanced Mathematics, vol. 7. American Mathematics Society, Providence, R. I.– 1998.– pp. 21–76.

46. Рабинович З.Л. Типовые операции в вычислительных машинах.// З.Л. Рабинович, В.А. Раманаускас / – К.: Техніка.– 1980. –264 с.

47. Акушский И.Я. Машинная арифметика в остаточных классах.// И.Я. Акушский, Д.И. Юдицкий / – М.: Сов. Радио.– 1968. – 460 с.

48. Задірака В.К. Комп’ютерна арифметика багаторозрядних чисел: Наукове видання.// В.К. Задірака, О.С. Олексюк / – К.: 2003. – 264 с.

50. Столлингс В. Структурная организация и архитектура компьютеных систем., 5-е изд.: Пер. с англ.// В. Столлингс / – М.: Издательський дом "Вильямс", 2002. - 896с.

51. Бухштаб А.А. Теория чисел.// А.А. Бухштаб / – М.: Просвещение, 1966. – 384 с.

52. Бородін О.І. Теорія чисел.// О.І. Бородін / – К.: Вища школа, 1970. – 275 с.

53. Николайчук Я.М. Низові часові мережі: Учбовий посібник.// Я.М. Николайчук /–К:УМК ВО, 1990.– 64с.

54. Торгашев В.А. Система остаточных классов и надежность ЦВМ.// В.А.Торгашев /–М.:Советское радио, 1970. –118с

55. Николайчук Я.М. Теорія цифрових перетворень мультибазисного супершвидкодіючого процесора.// Я.М. Николайчук / Научно-теоретический журнал "Искусственный интеллект". ІПШІ МОН і НАН України "Наука і освіта". – 2008. - №4. – С.387-394.

56. Николайчук Я.М. Теоретичні основи побудови та структура спецпроцесорів в базисі Крестенсона.// Я.М. Николайчук, О.І. Волинський, С.В. Кулина / Вісник Хмельницького національного університету.- Хмельницький.- 2007.- №3.- Т1.- С.85-90.

57. Николайчук Я.М. Перспективи використання зірково - магістральної архітектури з пам'яттю колективного доступу в комп'ютерних мережах з глибоким розпаралелюванням // Я.М. Николайчук, Н.Д. Круцкевич / Вимірювальна та часова техніка в технологічних процесах: Збірник наукових праць- Хмельницький: ТУП, - 2002. -Т2. - №9. - С. 122 –126.

58. Николайчук Я.М. Матричні системи числення. Вісник Хмельницького національного університету. //Я.М. Николайчук, О.Д. Круцкевич/ - Хмельницький.- 2007.- №3.- Т1.- С.62-64.

59. Николайчук Я.Н., Божнев В.П. Метод уплотнения информации, вводимой в ЭВМ // Я.Н. Николайчук, В.П. Божнев / Управляющие системы и машины.-№1.-1977.

60. Николайчук Я. М. Основи побудови часових систем на базі вертикальної інформаційної технології // Я. М. Николайчук / Тези науково-практичної конференції професорсько-викладацького складу. Івано-Франківськ. – 1999. – С.90–92.

60. А.В. Палагин А.В. Микропроцессорные вычислительные системы обработки информации: проектирования и отладка / А.В. Палагин, Е.Л. Денисенко, Р.И. Белицкий, В.И. Вигалов. – К.: Нукова думка, 1993. – 352с.

61. Пітух І.Р. Системні характеристики формальних об’єктів моделей руху даних в комп’ютерних мережах // І.Р. Пітух / Тези доповідей ІІІ Міжнар. конф. „PHOTONICS–ODS 2005”. – Вінниця. – 2005. – С. 60–61.

62. Пітух І.Р. Теоретичні основи побудови моделей економічних епюр руху даних в комп’ютерних мережах з використанням різних теоретико – числових базисів // І.Р. Пітух / Збірник наукових праць. Інститут проблем моделювання в енергетиці НАН України. – 2006.– № 37. – С.42–46 .

63. М.М.Касянчук. Теорія та математичні закономірності досконалої форми системи залишкових класів.// М.М.Касянчук./ Праці Міжнародного симпозіуму „Питання оптимізації обчислень (ПОО–XXXV)”, Київ–Кацивелі (Україна), 2009, ст. 306–310.

64. M.Kasyanchuk. Conception of theoretical bases of the accomplished form of Krestenson’s transformation and its practical application.// M.Kasyanchuk./ Proceedings of the 4–th International Conference “Advanced Computer Systems and Networks: Design and Application (ACSN–2009)”, Lviv (Ukraine), 2009, p. 299–301.

65. Николайчук Я.М., Пітух І.Р., Возна Н.Я. Теорія моделей руху даних розподілених комп’ютерних систем. // Я.М. Николайчук, І.Р. Пітух, Н.Я. Возна / – Тернопіль: ТзОВ „Терн граф”, 2008. – 216 с.

66. Якименко І.З. Порівнення часових характеристик виконання базових криптографічних операцій на еліптичній та гіпереліптичній кривих.// І.З. Якименко / – Луганськ: Праці Луганського відділення Міжнародної Академії інформатизації, №1(16), 2008р. – С:127 - 134.

67. Осипов Д. Delphi Профессиональное программирование.// Д. Осипов / – М.: Россия, 2004. – 1056 с.

68. P. L. Montgomery “Modular multiplication without trial division.” // P. L. Montgomery / Math. Computation, vol. 44, pp.519 - 521, 1985.

69. Карпінський М.П. Оцінка продуктивності та стійкості до часового аналізу алгоритмів експоненціюва-ння точки еліптичної кривої.// М.П. Карпінський, І.З. Якименко, М. Гіжицкі / Вісник Хмельницького національного університету. – 2006. – № 5. – С. 23-30.

70. М.М. Касянчук Теоретичні основи аналітики та алгоритми оптимізації обчислень простих чисел.// М.М. Касянчук, І.З. Якименко, О.І. Волинський, С.В. Івасьєв./ Проблемно-наукова міжгалузева конференція «Інформаційні проблеми комп'ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління ПНМК-2010, Україна, Бучач, Східниця, Карпати 01-04 червня 2010 року.

71. Kinakh Y.I, Iakymenko I.Z. Reliability of Schoof algorithm and its computational complexity. // Y.I. Kinakh, I.Z. Iakymenko / Proceedings of the Xth International Conference "The Experience of Designing and Application of CAD Systems in Microelectronics". – Lviv-Polyana. – 2009. – С. 107.

72. Кінах Я.І Удосконалення мережевих криптоаналітичних алгоритмів на еліптичних кривих. // Я.І Кінах, І.З. Якименко/ Вісник Східноукраїнського національного університету імені Володимира Даля.- №6(136), Т.1.- 2009.- С.48-51.

73. Карпінський М.П. Використання технології паралельних обчислень для рахування кількості точок еліптичної кривої. // М.П. Карпінський, І.З. Якименко, А.А. Хомінчук / Вісник Східноукраїнського національного університету імені Володимира Даля.– 2008. – № 8 (126), Частина 1. – С. 207-210.

74. Якименко І.З. Прискорення алгоритму Шуфа методом паралельних обчислень // І.З. Якименко, А.А. Хомінчук / Матеріали дванадцятої наукової конференції Тернопільського державного технічного університету імені Івана Пулюя, Тернопіль, ТДТУ, 14-15 травня 2008 р. – С:116.

75. Пітух І. Інформаційна технологія побудови миттєвих та інтегральних економічних епюр руху даних на основі циклів матричних моделей комп'ютерних систем. // І. Пітух /Вісник Технологічного університету Поділля. Технічні науки. – Хмельницький. – 2007. – Т.1, №3. – С.130-134.

76. Исаев Сергей. Генетические алгоритмы – эволюционные методы поиска. // С. Исаев / http://ai-online.fromru.com/documents-genetic_algorithms.html.

77. Пітух І. Критерії ефективності використання ресурсів архітектури інформаційних систем, які реалізують моделі руху даних //І. Пітух / Вісник Технологічного університету Поділля. Технічні науки. – Хмельницький. – 2006. – №5.– С.106-109.

78. Пітух І. Проектування характеристик системних об’єктів комп’ютерних мереж з глибоким розпаралеленням інформаційних потоків // І. Пітух /Вісник Технологічного університету Поділля. Технічні науки. – Хмельницький. – 2005. – Т.2, Ч.1, №4. – С.133-136.

79. О.І.Волинський Розмежована система числення залишкових класів та спец процесори на її основі. // О.І.Волинський, Якименко І.З./ Проблемно-наукова міжгалузева конференція «Інформаційні проблеми комп’ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління ПНМК-2010, Україна, Бучач, Східниця, Карпати 01-04 червня 2010 року.

80. ГОСТ 34.310-95. Информационная технология. Криптографическая защита информации. Процедуры выработки и проверки электронной цифровой подписи на базе ассиметричного криптографического алгоритма.

81. Якименко І.З. Дослідження часових характеристик алгоритму множення точок на еліптичних кривих (ЕК). // І.З.Якименко, А.А. Хомінчук, Н.Р.Драбик / Сучасні інформаційні технології. 4. Інформаційна безпека, травень 2007. – С:165.

82. Пітух І.Р. Теоретичні основи побудови моделей економічних епюр руху даних в комп’ютерних мережах з використанням різних теоретико – числових базисів // І. Пітух / Збірник наукових праць. Інститут проблем моделювання в енергетиці НАН України. – 2006.– № 37. – С.42–46 .

83. Menezes, T. Okamoto, and SA Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field, IEEE Transactions on Information Theory, Volume 39, 1993.

84. Карпінський М.П., Васильцов І.В., Якименко І.З. Показники оцінки ефективності алгоритмів шифрування на еліптичних кривих. // М.П. Карпінський, І.В. Васильцов, І.З. Якименко / Правове, нормативне, та метрологічне забезпечення системи захисту інформації в Україні, Київ, випуск 8, 2004. – С: 121 – 124.

85. В.К. Задірака Теоретичні основи та високопродуктивний алгоритм обчислення мультистепеневої функції в базисі Крестенсона.// В.К. Задірака, М.М. Касянчук, Я.М. Николайчук /Проблемно-наукова міжгалузева конференція «Інформаційні проблеми комп'ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління ПНМК-2010, Україна, Бучач, Східниця, Карпати 01-04 червня 2010 року.

86. В.К. Задірака. Оцінка оцінки оцінки наближеного розв'язку задачі.// В.К. Задірака./ Проблемно-наукова міжгалузева конференція «Інформаційні проблеми комп'ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління ПНМК-2010, Україна, Бучач, Східниця, Карпати 01-04 червня 2010 року.

87. Н.В. Кошкіна Побудова стеганосистем з ЦВЗ для аудіосигналів.// Н.В. Кошкіна/ Проблемно-наукова міжгалузева конференція «Інформаційні проблеми комп'ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління ПНМК-2010, Україна, Бучач, Східниця, Карпати 01-04 червня 2010 року.

88. Л.Л. Нікітенко Метод вкраплення цифрового водяного знаку на основі перетворення фазових складових сигналу.// Л.Л. Нікітенко, О.Ю. Нікітіна/ Проблемно-наукова міжгалузева конференція «Інформаційні проблеми комп'ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління ПНМК-2010, Україна, Бучач, Східниця, Карпати 01-04 червня 2010 року.1

89. Залмазон Л.А. Преобразование Фурье, Уолша, Хаара и их применение в уравнении, связи, и других областях. // Л.А. Залмазон / – М.: Наука, 1989. – 496с.

90. Николайчук Я.М., Ищеряков С.Н. А.С.№.1115062. – Бюллетень №35.1984. Многоканальное устройство для вычисления модульной функции.

91. Стеклов В.К.. Проектування телекомунікаційних мереж. // В.К. Стеклов, Л.Н. Беркман /–К.: Техніка, 2002.–792с.

92. Круцкевич Н.Д. Принципи побудови дешифраторів кодів Галуа пам’яті колективного доступу // Н.Д. Круцкевич / Вісник технологічного університету Поділля. – 2004. – №2. – Ч.1. – Т2 – С.113-116.

93. Петришин Л.Б. Теоретичні основи перетворення форми та цифрової обробки інформації в базисі Галуа: Навч. посібник.// Л.Б. Петришин / – Київ.:ІЗіМН МОУ, 1997. – 237с.

94. H. Koblitz, Elliptic curve cryptosystems , in Mathematics of Computation 48, 1987, pp. 203–209.

95. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы.// В.Г. Олифер, Н.А. Олифер / – СПб.: Питер, 2000. – 672 с

96. Буров Є. Комп’ютерні мережі.// Є. Буров / – Львів: БаК, 1999. – 468 с.

97. Мартин Дж. Планирование развития автоматизированных систем.// Дж. Мартин /- М.: Финансы и статистика, 1984. – 196с.

98. Мартин Дж. Организация баз данных в вычислительных системах.// Дж. Мартин / – М.: Мир, 1980. – 662с.

99. АС №964693 СССР, МКИ G08 C19/28. “Устройство для передачи сигналов” Я.Н. Николайчук и Г.Я. Ширмовский. – Опубл. 07.10.82, Бюл. №37.

100. Круцкевич Н.Д. Перспективи розвитку комп’ютерних мереж з реконфігурацією архітектури на базі пам’яті колективного доступу // Н.Д. Круцкевич / Вісник «Радіоелектроніка та телекомунікації» Національного університету ”Львівська політехніка”, Львів, 2004, №508, С. 240 – 245.

101. Y. Nykolaychuk Architecture and system characteristic of distributed computer network with autonomus sensor equipment// Y. Nykolaychuk, N. Krutskevych, O. Zastavnyy /Proc. of the International Conf.”Modern problems of radio engineering, telecommunications and computer science” TCSET 2006. – Lviv-Slavsko (Ukraine). – P. 394 – 398c.

102. O Y. Nykolaychuk Perspective Architecture and Components of Computer Networks // O Y. Nykolaychuk, N. Krutskevych O. Zastavniy, T. Grinchyshyn/ Proc. Of the Second IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS, Lviv, Ukraine, 2003

103. Okeya K. Power Analysis Breaks Elliptic Curve Cryptosystems even Secure against the Timing Attack, Progress in Cryptology.// K.Okeya, K.Sakurai /– INDOCRYPT 2000, LNCS1977, (2000), 178-190.

104. Smart,N.P., The Discrete Logarithm Problem on Elliptic Curves of Trace One, Journal of Cryptology, Vol.12, No.2, (1999), 141-151.

105. Карпінський М.П. Класифікація атак на апаратно-програмні засоби криптосистем.// М.П.Карпінський, І.З.Якименко, Ю.М.Чайківська / Матеріали II міжнар. наук.-техн. конф. «Світлотехніка й електротехніка: історія, проблеми й перспективи» (24-27 травня 2005 р., Тернопіль, Україна). – Тернопіль: Тернопільський державний технічний університет імені Івана Пулюя. – 2005.

106. Karpinsky M.P. Formalization Assessment Criterion Attacks on Cryptosystems Using Elliptic Curves.// M.P.Karpinsky, I.Z.Yakymenko, J.M Chajkivska / Proceedings of the Third IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications “IDAACS’2005” (September 5-7, 2005, Sofia, Bulgaria). – Sofia. – 2005.

107. Хетагуров Я.А., Руднев Ю.П. Повышение надежности цифровых устройств методами избыточного кодирования. - М.: Энергия, 1974. – 272с.

108. Новиков Л.Г., Шурыгин И.Т. Счётчики импульсов с коэффициентами счёта, управляемыми с помощью двоичного кода. Журнал «Приборы и системы управления» № 6, 1972. –С.30-31.

109. D. Hankerson, A. Menezes, and S.A. Vanstone, Guide to Elliptic Curve Cryptography , Springer-Verlag, 2004.

110. I. Blake, G. Seroussi, and N. Smart, editors, Advances in Elliptic Curve Cryptography , London Mathematical Society 317, Cambridge University Press, 2005.

111. K. Malhotra, S. Gardner, and R. Patz, Implementation of Elliptic-Curve Cryptography on Mobile Healthcare Devices, Networking, Sensing and Control, 2007 IEEE International Conference on, London, 15–17 April 2007 Page(s):239–244.

112. Saikat Basu, A New Parallel Window-Based Implementation of the Elliptic Curve Point Multiplication in Multi-Core Architectures , International Journal of Network Security, Vol. 13, No. 3, 2011, Page(s):234-241.

113. Standards for Efficient Cryptography Group (SECG) , SEC 1: Elliptic Curve Cryptography , Version 1.0, September 20, 2000.

114. L. C. Washington: Elliptic Curves: Number Theory and Cryptography. Chapman & Hall/CRC, New York, 2003.

115. N. Harris. Math 168 Project. University of California, San Diego, Fall 2005.

116. H. Stark. Lecture notes for Math 204. University of California, San Diego, Fall 2005.

117. Lercier, R. and Morain, F., Counting the number of points on elliptic curves over finite fields: strategies and performances, Advances in Cryptology, Proc. Eurocrypt'95, LNCS 921, L.C. Guillou and J.J. Quisquater, Eds., Springer-Verlag, 1995, pp. 79-94.

118. Кулаков В.Г. Защита информации в телекоммуникационных системах./ Кулаков В.Г., Андреев А.Б., Заряев А.В. и др.// – Воронеж: Воронежский институт МВД России, 2002. – 300 с.

119. Schoof's Algorithm for Counting Points on E(Fp),

www.math.umn.edu/~musiker/ schoof .pdf

120. Schoof-Elkies-Atkin Algorithm - from Wolfram MathWorld, mathworld.wolfram.com

121. An exposition of schoof's algorithm, mathpost.asu.edu/~sjgm/issues/2005_spring/SJGM_alvarado.pdf

122. Schoof's algorithm | Facebook, www.facebook.com/pages/ Schoofs - algorithm /132871906750922.

123. Efficient Implementation of Schoof's Algorithm, www.springerlink.com/index/fxkqptxrkheea7tp.pdf

124. Schoof's algorithm - Java Forums, www.java-forums.org.

Додаток А

Лістинг додатку «Генерування параметрів та випадкової точки на еліптичній кривій»

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#define _len 130

#include "Main.h"
#include "lip.c"
#include "HashFunct.cpp"
#include "BigInt.cpp"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"

# define Path "Numbers.txt"

TFMain *FMain;
int Freq, v, s, w, coup = 0, cgen = 0, step, st2;
verylong p, a, b, c, t, h, t2;
bool gen;
char stp[256], hf[32], X[32];
FILE *nums;

//---------------------------------------------------------------------------
__fastcall TFMain::TFMain(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void int2str(char s[200], char str[255])
{
    BigInt a;
    a.zero();
    int i;
    for (i = 0; i < strlen(s); i++)
    {
            a.Coef[i] = s[i];
            a.Coef[i] &= 255;
    }
    a.Size = strlen(s);
    BigInt tm;
    ushort r;
    tm = a;
    for (i = 0; i < 200; i++)
            str[i] = 0;
    for (i = 0; i < 200; i++)
    {
            if (ZeroInt(tm)) break;
```

```
//            SDiv(tm, 10, q, r);
              r = tm % 10;
              tm = tm / 10;
              str[i] = r+'0';
    }
    char t;
    for (i = 199; i != 0; i--)
              if (str[i] != 0) break;
    int m = i;
    for (i = 0; i < m/2+1; i++)
    {
              t = str[i];
              str[i] = str[m-i];
              str[m-i] = t;
    }
}

void Generate()
{
    st2++;
    FMain->CStep->Caption = IntToStr(st2);
    FMain->Repaint();
    if (cgen) goto notc;

    for(int i = 0; i < 32; i++)
              X[i] = random(255)+1;
    X[32] = 0;

    Hash(X, hf);

    int2str(hf, stp);
    zsread(stp, &c);
    step++;
    if (ziszero(c))
              return;
    zsmul(c, 4, &t);
    zsadd(t, 27, &a);
    zmod(c, p, &t);
    if (ziszero(t))
              return;
    cgen = 1000;
    zswrite(stp, c);
    FMain->CNum->Text = stp;
notc:;
    cgen--;
    FMain->CNum->Text.sprintf("%s", stp);
    zsread(stp, &c);
    for(int i = 0; i < 32; i++)
              stp[i] = random(255)+1;
    stp[32] = 0;
    int2str(stp, stp);
    zsread(stp, &a);
    FMain->ANum->Text = stp;
    zswrite(stp, p);
```

```
    zswrite(stp, c);
    zinvmod(c, p, &t);
    zsexpmod(a, 3, p, &t2);
    zmul(t, t2, &b);
    zmod(b, p, &h);
    int j = zjacobi(h, p);
    FMain->Jac->Caption = IntToStr(j);
    if (zjacobi(h, p) == -1) return;
    zsqrtmod(h, p, &b);
    if (!b) return;
    zsexpmod(b, 2, p, &t);
    zmulmod(c, t, p, &t2);
    zsexpmod(a, 3, p, &t);
    if (zcompare(t, t2)) return;
    zswrite(stp, b);
    FMain->BNum->Text = stp;
    coup++;
    FMain->Couples->Caption = IntToStr(coup);
    zswrite(stp, a);
    fprintf(nums, "%s\n", stp);
    zswrite(stp, b);
    fprintf(nums, "%s\n", stp);
    zswrite(stp, c);
    fprintf(nums, "%s\n", stp);
//  FMain->Repaint();
}

void __fastcall TFMain::Timer1Timer(TObject *Sender)
{
    if (Timer1->Interval == 1) Timer1->Interval = 1000;
    Timer1->Enabled = false;
    for(int i = 0; i < Freq; i++)
            Generate();
    Timer1->Enabled = true;
}
//---------------------------------------------------------------------------

void __fastcall TFMain::GenerClick(TObject *Sender)
{
    EFreq->Enabled = false;
    EMod->Enabled = false;
    LoadB->Enabled = false;
    if (!p)
    {
            EMod->Text.sprintf("%s", stp);
            zsread(stp, &p);
    }
    if (!gen)
    {
            Freq = StrToInt(EFreq->Text);
/*          EMod->Text.sprintf("%s", stp);
            zsread(stp, &p);*/
            Gener->Caption = "Зупинити";
            Timer1->Enabled = true;
```

```
            Measure->Enabled = true;
            gen = !gen;
            return;
    }
    else
    {
            EFreq->Enabled = true;
            Gener->Caption = "Продовжити";
            Timer1->Enabled = false;
            Measure->Enabled = false;
            gen = !gen;
//          step = 0;
            return;
    }

    /*****************************************************\
    |*******        Логарифм з р при основі 2 завжди 256     *****|
    \*****************************************************/

    v = 256;
    s = 0;
    w = 255;
}


//---------------------------------------------------------------------------

void __fastcall TFMain::ExitClick(TObject *Sender)
{
    Close();
}
//---------------------------------------------------------------------------

void __fastcall TFMain::FormShow(TObject *Sender)
{
    randomize();
    FILE *f;
    if (f = fopen("ModulusP.txt", "r"))
    {
            fscanf(f, "%s", stp);
            EMod->Text = stp;
            fclose(f);
    }
    nums = fopen(Path, "a");
}
//---------------------------------------------------------------------------

void __fastcall TFMain::CNumEnter(TObject *Sender)
{
    CNum->SelectAll();
}
//---------------------------------------------------------------------------

void __fastcall TFMain::ANumEnter(TObject *Sender)
```

```
{
   ANum->SelectAll();
}
//---------------------------------------------------------------------------

void __fastcall TFMain::BNumEnter(TObject *Sender)
{
   BNum->SelectAll();
}
//---------------------------------------------------------------------------

void __fastcall TFMain::EModKeyPress(TObject *Sender, char &Key)
{
   if ((Key == 13) && (Gener->Enabled)) Gener->Click();
}
//---------------------------------------------------------------------------

void __fastcall TFMain::FormClose(TObject *Sender, TCloseAction &Action)
{
   fclose(nums);
}
//---------------------------------------------------------------------------

void __fastcall TFMain::LoadBClick(TObject *Sender)
{
   AnsiString s;
   s = GetCurrentDir();
   if (Load->Execute())
   {
            FILE *f;
            char s1[256];
            Load->FileName.sprintf("%s", s1);
            f = fopen(stp, "r");
            fscanf(f, "%s", stp);
            EMod->Text = s1;
            fclose(f);
            SetCurrentDir(s);
   }
   EMod->SetFocus();
}
//---------------------------------------------------------------------------

void __fastcall TFMain::CNumChange(TObject *Sender)
{
   Repaint();
}
//---------------------------------------------------------------------------

void __fastcall TFMain::ANumChange(TObject *Sender)
{
   Repaint();
}
//---------------------------------------------------------------------------
```

```
void __fastcall TFMain::BNumChange(TObject *Sender)
{
   Repaint();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("Main.cpp", FMain);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
   try
   {
            Application->Initialize();
            Application->CreateForm(__classid(TFMain), &FMain);
            Application->Run();
   }
   catch (Exception &exception)
   {
            Application->ShowException(&exception);
   }
   catch (...)
   {
            try
            {
                     throw Exception("");
            }
            catch (Exception &exception)
            {
                     Application->ShowException(&exception);
            }
   }
   return 0;
}
//---------------------------------------------------------------------------
```

Додаток Б

Лістинг додатку «Ідентифікації простих чисел»

```
//---------------------------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
#include "Unit6.h"
#include "Unit7.h"
#include "Unit8.h"
#include "Unit9.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
big_prime *prime;
TChrtBuilder *ChartBuilder1;
remaining *remainingtablebuilder1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{

}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button2Click(TObject *Sender)
{
prime->Suspend();
Label2->Caption=IntToStr(prime->inumber);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{

 prime= new big_prime(false);

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Close2Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Close1Click(TObject *Sender)
{
```

```
 if (SaveDialog1->Execute()){TestingMemo->Lines->SaveToFile(SaveDialog1->FileName);}
}
//---------------------------------------------------------------------------




void __fastcall TForm1::Button4Click(TObject *Sender)
{
prime->Resume();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->TestingMemo->Clear();
 TDateTime DateTime = Time();  // store the current date and time
  AnsiString str = TimeToStr(DateTime); // convert the time to a string
  Form1->TestingMemo->Lines->Add(str);
//testing_number(StrToInt(Form1->Edit3->Text),prime_list,Form1->TestingMemo);
if (prime)prime->from_vectro_to_memo(Form1->TestingMemo,prime->prime_list);
DateTime = Time();
str = TimeToStr(DateTime); // convert the time to a string
Form1->TestingMemo->Lines->Add(str);
//if (prime)prime_list=prime->prime_list;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{
Memo1->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button6Click(TObject *Sender)
{
bool isprime=true;
bool limitout=true;
long i=1;
long num=StrToInt(Edit1->Text);
  //Memo1->Lines->Add(IntToStr(prime->prime_list.size()));
long temp;//= new long;
while (limitout) {
if (i<prime->prime_list.size()){
//Memo1->Lines->Add(IntToStr(prime->prime_list[i]));
       if (prime->prime_list[i]<num){
       temp=(num-prime->prime_list[i]);
       Memo1->Lines->Add(IntToStr(temp));

              // for (int j=0;j==0; *temp>>=1, j++){
               if (temp&(1<<0)?1:0){
               isprime=false;
```

```
                Memo1->Lines->Add(IntToStr(0000000000000));
                };
               // };
        i++;

        }else{limitout=false;};

}else{limitout=false;};

};
if (isprime){Label4->Caption="Число просте ...";}else{Label4->Caption=" ";};
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button8Click(TObject *Sender)
{
//thre = new remaining(false);
}
//---------------------------------------------------------------------------


void __fastcall TForm1::ScrollBar1Change(TObject *Sender)
{
Chart1->Chart3DPercent=ScrollBar1->Position;

}
//---------------------------------------------------------------------------

void __fastcall TForm1::ScrollBar2Change(TObject *Sender)
{
if (ScrollBar1->Enabled ){
Chart1->View3DOptions->Orthogonal=false;
Chart1->View3DOptions->Rotation=ScrollBar2->Position;
};
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ScrollBar3Change(TObject *Sender)
{

if (ScrollBar1->Enabled) {
    Chart1->View3DOptions->Orthogonal=false;
    Chart1->View3DOptions->Elevation=ScrollBar3->Position;
  }

}
//---------------------------------------------------------------------------

void __fastcall TForm1::CheckBox5Click(TObject *Sender)
{
Chart1->View3D=CheckBox5->Checked;
ScrollBar1->Enabled=Form1->Chart1->View3D;
```

```
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button11Click(TObject *Sender)
{
ListBox1->Clear();
for (int i=0;i<Chart1->SeriesCount();i++){
ListBox1->Items->Add("series "+IntToStr(i));

};


}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button10Click(TObject *Sender)
{

Chart1->RemoveSeries(Chart1->Series[ListBox1->ItemIndex]);
// Chart1->Series[ListBox1->ItemIndex]->DestroyComponents();
ListBox1->Clear();
for (int i=0;i<Chart1->SeriesCount();i++){
ListBox1->Items->Add("series "+IntToStr(i));

};
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button9Click(TObject *Sender)
{
  ChartBuilder1= new TChrtBuilder(false);



}
//---------------------------------------------------------------------------



void __fastcall TForm1::Button12Click(TObject *Sender)
{
///Edit13->Text=IntToStr(GetChargePartNumber(StrToInt(Edit10->Text),StrToInt(Edit11->Text),StrToInt(Edit12->Text)));

}
//---------------------------------------------------------------------------


void __fastcall TForm1::Chart1Zoom(TObject *Sender)
{
ScrollBar4->Position=Chart1->View3DOptions->Zoom;
```

```
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ScrollBar4Change(TObject *Sender)
{
Chart1->View3DOptions->Zoom=ScrollBar4->Position;
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button7Click(TObject *Sender)
{

//void from_vector_to_table_string(vector<long>& mas1,TStringGrid *StringGrid1){
remainingtablebuilder1=new remaining(false);
}
//---------------------------------------------------------------------------




void __fastcall TForm1::Button13Click(TObject *Sender)
{
if( ColorDialog1->Execute() )
        Color1 = ColorDialog1->Color;

}
//---------------------------------------------------------------------------


void __fastcall TForm1::N2Click(TObject *Sender)
{
Form6->Show();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::N3Click(TObject *Sender)
{
Form7->Show();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::N4Click(TObject *Sender)
{
Form8->Show();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::N5Click(TObject *Sender)
{
Form9->Show();
}
//---------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit2.h"
#include "Unit1.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall big_prime::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------
//*****************************************************************************
void big_prime::from_memo_to_vector(TMemo *temp1,vector<long>& temp2){
temp2.resize(temp1->Lines->Count);
for (int i=0;i<temp2.size();i++){
temp2[i]=StrToInt(temp1->Lines->Strings[i]);
};
};
//*****************************************************************************
String big_prime::from_vector_to_string(vector<bool>& temp2){
String s="";
for (int i=0;i<temp2.size();i++){
if (temp2[i]==0) {s=s+'0';} else {s=s+'1';};
}; return s;};

//*****************************************************************************
bool big_prime::from_long_to_bit_vector(long num,vector<bool>& temp1)
{
  int i;
  for (i=0; num; num>>=1, i++){
  temp1.insert(temp1.begin(),bool((num&1)?(1):(0)));
  };
};

//*****************************************************************************
bool big_prime::from_long_to_bit_vector_reverse(long num,vector<bool>& temp1)
{
  int i;
  for (i=0; num; num>>=1, i++){
  temp1.push_back(bool((num&1)?(1):(0))) ;
  };
};
//*****************************************************************************
```

```
long big_prime::mod(long a,long p){
//if (a<1000){return a%p;}else{
long sum=0;
for (int i=0; a; a>>=1, i++){
if (bool((a&1)?(1):(0))==1){
sum=sum+(long(pow(2,i)) % (p));
};
};
return sum%p;
};
//};

//***************************************************************************

void big_prime::from_vectro_to_memo(TMemo *temp1,vector<long>& temp2){
AnsiString s;
char string[25];

for (int i=0;i<temp2.size();i++){
//temp2[i]
itoa(temp2[i], string, 2);
s=string;
if (ismersen(temp2[i])) s=s+"  !!!Число Мерсена !!!";
if (isferma(temp2[i])) s=s+"  !!!Число Ферма !!!";
temp1->Lines->Add(IntToStr(temp2[i])+" "+s);
};

};
//***************************************************************************
bool big_prime::mod_test(vector<long>& mas1,vector<long>& mas2,vector<long>& mas3,
long squarelimit){
bool have_null;
bool limit_out=false;
long temp;
mas3.resize(mas1.size());
for (int i=0;(i<mas1.size())&&(!limit_out);i++){
limit_out=(mas2[i]>squarelimit+1);
temp=(mas1[i]) % (mas2[i]);
if (temp==0)have_null=true;

if (temp==0){
mas3[i]=mas1[i]+2;
}else {
mas3[i]=temp;
};
};

return have_null;
};
//***************************************************************************
bool big_prime::mod_test(vector<long>& mas1,vector<long>& mas2,vector<long>&
mas3){
bool have_null;
```

```
long temp;
mas3.resize(mas1.size());
for (int i=0;(i<mas1.size());i++){
temp=(mas1[i]) % (mas2[i]);
if (temp==0)have_null=true;

if (temp==0){
mas3[i]=mas1[i]+2;
}else {
mas3[i]=temp;
};
};

return have_null;
};


//**************************************************************************
bool big_prime::mod_test(long num1,vector<long>& mas1,vector<long>& mas2){
long num2=sqrt(num1);
vector<long> mas3;
mas3.resize(mas1.size());
for(int i=0;i<mas1.size();i++){
mas3[i]=num1;
};
return mod_test(mas3,mas1,mas2,num2);
};
//**************************************************************************
long big_prime::modexp(long a,long b,long p){
long temp1,dob=mod(a,p);
for (int i=0; b; b>>=1, i++){
if (bool((b&1)?(1):(0))==1){
temp1=dob;
dob=modmul(a,temp1,p);
};
};
return dob%p;
};
//**************************************************************************
long nsd(long a,long b,vector<long>& mas1){
int i=0;
while ((mas1[i]<=a)&&(mas1[i]<=b)&&i<=mas1.size()){

i++;
};


};
//**************************************************************************
long big_prime::modmul(long a,long b,long p){
long sum=0;
long b1=b;
for (int i=0; a; a>>=1, i++){
 if (bool((a&1)?(1):(0))==1){
```

```
// Form1->TestingMemo->Lines->Add(IntToStr(i)+" біт 1 числа "+a);
    b=b1;
    for (int i1=0; b; b>>=1, i1++){

        if (bool((b&1)?(1):(0))==1){
  //        Form1->TestingMemo->Lines->Add(IntToStr(i1)+" біт 1 числа "+b);
        sum=sum+(mod(pow(2,i+i1),p));
    //    Form1->TestingMemo->Lines->Add(IntToStr(sum));
        };
    };
 };
};
return mod(sum,p);
};
void big_prime::fill_vector_from_interval(long num1,long num2,vector<long>& mas1){
for(long i=0;i<num2-num1;i++){
mas1[i]=num1+i;
};
};
bool big_prime::rulepermit(long inumber){
bool is=true;
for (int i;i<rules.size();i++){
if ((inumber-rules[i].number)==rules[i].modul){
rules[i].number=inumber;
is=false;
};
};

return is;
};
void big_prime::set_prime_numbers(vector<long>& prime_numbers){
prime_numbers.resize(3);
prime_numbers[0]=2;
prime_numbers[1]=3;
prime_numbers[2]=5;
//prime_numbers[3]=7;
};
void big_prime::fill_vector_by_number(long num,vector<long>& mas1,long size){
mas1.resize(size);
for(int i=0;i<mas1.size();i++){mas1[i]=num;};
};

bool big_prime::ismersen(long a){
bool is=true;
for (int i=0; a; a>>=1, i++){
if (bool((a&1)?(1):(0))!=1){
is=false;
};
};
return is;
};

bool big_prime::isferma(long a){
int countone=0;
```

```
        bool is=false;
        bool isbegin=false;
        if ((bool((a&1)?(1):(0))==1))isbegin=true;
        for (int i=0; a; a>>=1, i++){
        if (bool((a&1)?(1):(0))==1){
        countone++;
        };
        };
        if (isbegin&&countone==2)is=true;
        return is;
        };

        //---------------------------------------------------------------------------
        void big_prime::testing_number(long number,vector<long>& prime_numbers,TMemo
*temp1){
          set_prime_numbers(prime_numbers);

          //testing information++++++++++
         // temp1->Lines->Add("результати заповнення простими числами");
         // from_vectro_to_memo(temp1,prime_numbers);
          //testing information++++++++++

          vector<long> ostacha;
          vector<long> next_ostacha;
          fill_vector_by_number(7,ostacha,1);

          //testing information++++++++++
        //  temp1->Lines->Add("результати заповнення вектора тестовим стартовим числом");
        //  from_vectro_to_memo(temp1,ostacha);
          //testing information++++++++++

          bool have_null;
          bool limit_out=false;
          long temp,squarelimit;
          //testing information++++++++++
        //  temp1->Lines->Add("число з якого починається пошук - inumber"+IntToStr(inumber)
+ " пошук буде відбуватися до - number "+ IntToStr(number));
          //testing information++++++++++
           Form1->ProgressBar1->Max=number;
        for (inumber=7;inumber<=number;inumber=inumber+2){
           Form1->ProgressBar1->Position=inumber;
          //testing information++++++++++
        // temp1->Lines->Add("");
        //  temp1->Lines->Add("число яке перевіряється - inumber "+IntToStr(inumber));
          //testing information++++++++++

        if (rulepermit(inumber)){
        //temp1->Lines->Add();
        //*****************************


              have_null=false;
              limit_out=false;
```

```
            squarelimit=int(sqrt(inumber))+1;
            int i=0;
          //testing information++++++++++
         // temp1->Lines->Add("корінь з числа +1 - squarelimit "+IntToStr(squarelimit));
          //testing information++++++++++

         //       next_ostacha.resize(ostacha.size());

           //testing information++++++++++
         //temp1->Lines->Add("перевірка на остачу нуль в циклі до ostacha.size() або ж до
перевищення ліміту "+IntToStr(ostacha.size()));
         //testing information++++++++++
          //Form1->ProgressBar2->Max=squarelimit;
               while (!limit_out)
                                                  // for (int i=0;(i<ostacha.size())or(!limit_out);i++)
               {
          //       Form1->ProgressBar2->Position=prime_numbers[i];
               limit_out=(prime_numbers[i+1]>squarelimit);
                                                  //testing information++++++++++
                                                  // temp1->Lines->Add("статус  limit_out -
"+BoolToStr(limit_out));
                                                 //testing information++++++++++
               if (ostacha.size()-1>=i){                                          //  if
(ostacha[i]>=prime_numbers[i]){
               temp=mod((ostacha[i]) , (prime_numbers[i]));
               }else{
               ostacha.resize(ostacha.size()+1);
               ostacha[ostacha.size()-1]=inumber;
               temp=mod((ostacha[i]) , (prime_numbers[i]));


               };
                                                  //;}else{temp=0;};
                                                  //testing information++++++++++
              //  temp1->Lines->Add(IntToStr(ostacha[i])+" mod "+IntToStr(prime_numbers[i])+" =
"+IntToStr(temp));
                                                 //testing information++++++++++
               if (temp==0){
               ostacha[i]=2;
               have_null=true;                                  //testing information++++++++++
            //    temp1->Lines->Add("temp==0 залишок для наступного числа
"+IntToStr(ostacha[i]));
               }else {
               ostacha[i]=temp+2;
          //       temp1->Lines->Add("temp!=0 залишок для наступного числа
"+IntToStr(ostacha[i]));
               };
               i++;

               };

              if (!have_null){
              prime_numbers.resize(prime_numbers.size()+1);
              prime_numbers[prime_numbers.size()-1]=inumber;
```

```
        have_null=false;
  //testing information++++++++++
 // temp1->Lines->Add("
 // жодна остача не дорівнює нулю нове просте число додане до списку - "+
 // temp1->Lines->Add(IntToStr(inumber));
  //+" ************************************************************");
  //testing information++++++++++

       };


}else {for (int j=0;j<ostacha.size();j++)ostacha[j]=ostacha[j]+2;};
};
};




__fastcall big_prime::big_prime(bool CreateSuspended)
       : TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
void __fastcall big_prime::Execute()
{
Form1->TestingMemo->Clear();
TDateTime DateTime = Time();  // store the current date and time
AnsiString str = TimeToStr(DateTime); // convert the time to a string
Form1->TestingMemo->Lines->Add(str);
testing_number(StrToInt(Form1->Edit3->Text),prime_list,Form1->TestingMemo);
from_vectro_to_memo(Form1->TestingMemo,prime_list);
DateTime = Time();
str = TimeToStr(DateTime); // convert the time to a string
Form1->TestingMemo->Lines->Add(str);
Form1->prime_list=prime_list;
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit3.h"
#include "Unit1.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
```

```
//      void __fastcall remaining::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------
long remaining::mod(long a,long b){
long temp=a%b;
return temp;
};
//---------------------------------------------------------------------------

void remaining::from_vector_to_table_string(vector<long>& mas1,TStringGrid
*StringGrid1){
for(int i=0;i<mas1.size();i++){
if (i>=StringGrid1->ColCount)StringGrid1->ColCount++;
StringGrid1->Cells[i][StringGrid1->RowCount]=IntToStr(mas1[i]);};
StringGrid1->RowCount++;
};

void remaining::get_remainders(long n,long prime,vector<long>& remainders){
//remainders.resize(n+1);
bool haveretry=false;
int i=1;
remainders.resize(2);
remainders[0]=prime;
remainders[i]=mod(pow(2,i),prime);


while ((i<=n)&&(haveretry==false)){
i++;
remainders.resize(remainders.size()+1);
remainders[i]=mod(pow(2,i),prime);
haveretry=remainders[i]==remainders[1];

};
};

void remaining::build_table_remainder(long maxn,long maxprime,vector<long>&
primes,TStringGrid *StringGrid1){

StringGrid1->Cells[0][StringGrid1->RowCount]="Прості числа";
for(int i=0;i<=maxn;i++){
if (i>=StringGrid1->ColCount)StringGrid1->ColCount++;
if (i>0)StringGrid1->Cells[i][StringGrid1->RowCount]="2^"+IntToStr(i);
};
StringGrid1->RowCount++;



  int i=0;
  vector<long> remainders;
  while (i<primes.size()&&primes[i]<=maxprime){
 //  Form1->Memo3->Lines->Add("Get remainders for "+IntToStr(primes[i]));
  get_remainders(maxn,primes[i],remainders);
```

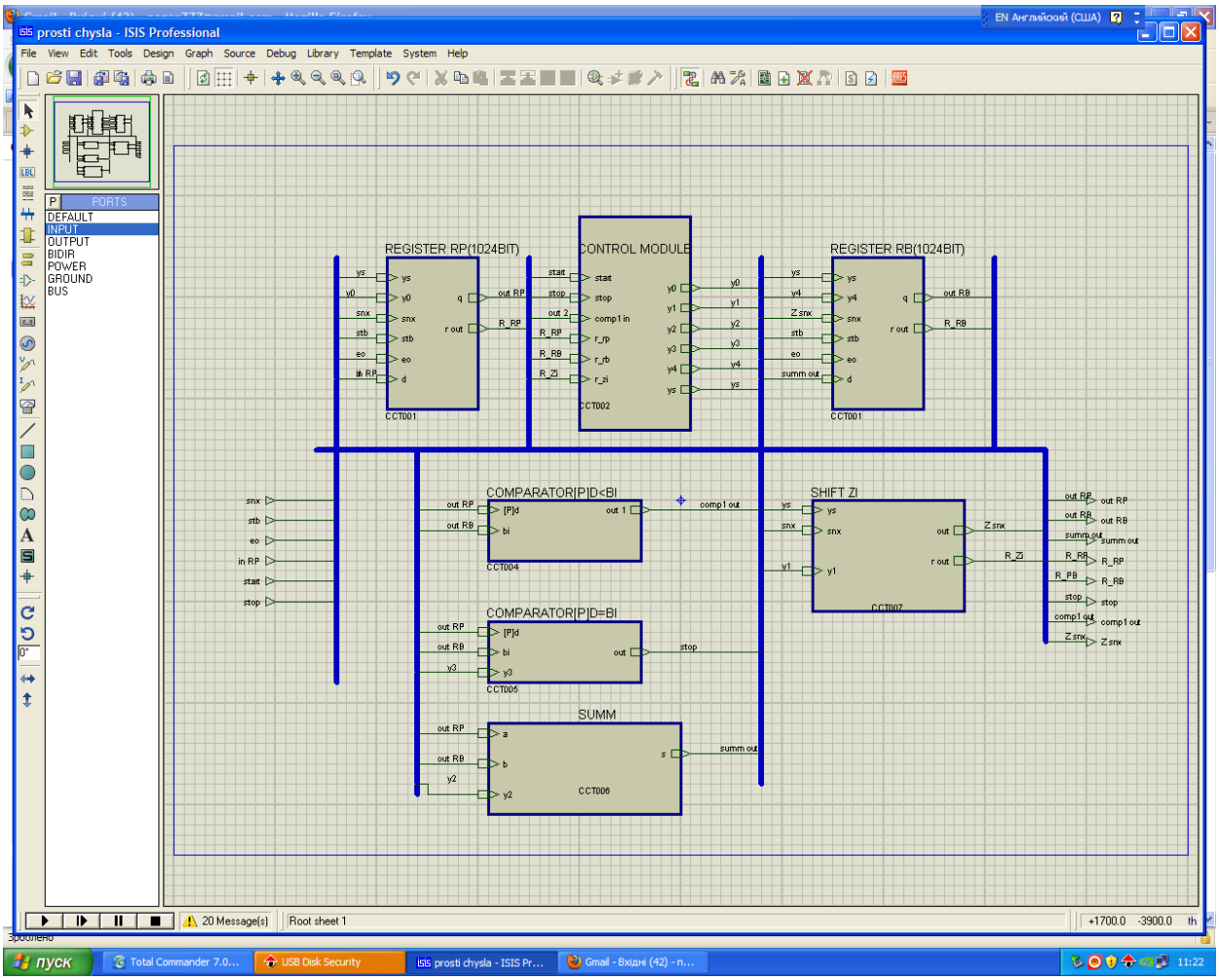
```
        //  Form1->Memo3->Lines->Add("Size of remainder "+IntToStr(remainders.size()));
          from_vector_to_table_string(remainders,StringGrid1);
          i++;
         // remainders.
          };
    };


    __fastcall remaining::remaining(bool CreateSuspended)
          : TThread(CreateSuspended)
    {
    }
    //---------------------------------------------------------------------------
    void __fastcall remaining::Execute()
    {
    Form1->StringGrid1->ColCount=1;
    Form1->StringGrid1->RowCount=1;
    //get_remainders(StrToInt(Form1->Edit4->Text),3,remainders_table[0].remainders);
     build_table_remainder(StrToInt(Form1->Edit2->Text),StrToInt(Form1->Edit4-
>Text),Form1->prime_list,Form1->StringGrid1);
    }
    //---------------------------------------------------------------------------
    //---------------------------------------------------------------------------

    #include <vcl.h>
    #pragma hdrstop

    #include "Unit4.h"
    #include "Unit1.h"
    #include "Unit2.h"
    #pragma package(smart_init)
    //---------------------------------------------------------------------------

    //   Important: Methods and properties of objects in VCL can only be
    //   used in a method called using Synchronize, for example:
    //
    //      Synchronize(UpdateCaption);
    //
    //   where UpdateCaption could look like:
    //
    //      void __fastcall TChrtBuilder::UpdateCaption()
    //      {
    //        Form1->Caption = "Updated in a thread";
    //      }


    long TChrtBuilder::GetChargePartNumber(long number,int charge_from,int chrge_to){
    long num=0;
    if (charge_from<chrge_to){

    char string1[(sizeof(long)*8)+2];
    AnsiString s;

    itoa(number,string1, 2);
```

```
        s=string1;
        if (chrge_to>s.Length())chrge_to=s.Length()-1;
        if (charge_from<0)charge_from=0;
        if (charge_from>s.Length())charge_from=s.Length();
        //if (charge_from=chrge_toLength())charge_from=s.Length();
        //delete string1;
        s=s.SubString(s.Length()-chrge_to,chrge_to-charge_from+1);

        //Form1->Memo2->Lines->Add(string1);
        //Form1->Memo2->Lines->Add(s1);
        //Form1->Memo2->Lines->Add("length = "+IntToStr(s1.Length()));
        for (int i =1;i<=s.Length();i++){
        if (s.SubString(i,1)=='1'){
              num=num+pow(2,s.Length()-i);
             // Form1->Memo2->Lines->Add("s1[i] = "+s1.SubString(i,1));
             // Form1->Memo2->Lines->Add("power = "+IntToStr(s1.Length()-i));
        };
        };};
        return num;


        };

        //---------------------------------------------------------------------------
        void TChrtBuilder::addseries(TChart *Chart1,long from,long to,bool
prime_only,vector<long>& primes,bool discharge_limit,int charge_from,int chrge_to,bool
isfermanumber,bool ismersennumber,TColor Color1){
        TLineSeries *series1;
        big_prime *prime1;
        int point1=0;
        series1 = new TLineSeries(Chart1);
        //series1->Name=name;
        int i=0;
        if (prime_only){

              while (i<=primes.size()&&primes[i]<=to){
                    if (primes[i]>from&&primes[i]<to){
                          if (discharge_limit==false){
                          if (!isfermanumber&&!ismersennumber)series1-
>Add(primes[i],primes[i],Color1);
                          if (isfermanumber)  if (prime1->isferma(primes[i])) series1-
>Add(primes[i],primes[i],Color1);
                          if (ismersennumber) if (prime1->ismersen(primes[i])) series1-
>Add(primes[i],primes[i],Color1);
                          }else{
                          if (!isfermanumber&&!ismersennumber)series1-
>Add(primes[i],GetChargePartNumber(primes[i],charge_from,chrge_to),Color1);
                          if (isfermanumber)  if (prime1->isferma(primes[i]))series1-
>Add(primes[i],GetChargePartNumber(primes[i],charge_from,chrge_to),Color1);
                          if (ismersennumber) if (prime1->ismersen(primes[i]))series1-
>Add(primes[i],GetChargePartNumber(primes[i],charge_from,chrge_to),Color1);

                          };
                    };
```

```
        i++;
        };

    }else{
        for(i=from;i<=to;i++){
        if (isfermanumber)  if (prime1->isferma(i))series1->Add(i,i,Color1);

        if (ismersennumber) if (prime1->ismersen(i))series1->Add(i,i,Color1);

        if (!isfermanumber&&!ismersennumber)  series1->Add(i,i,Color1);

        };

    };


    //series1->Marks->Style;
    series1->Marks->Visible=true;
    Chart1->AddSeries(series1);



    };


    __fastcall TChrtBuilder::TChrtBuilder(bool CreateSuspended)
        : TThread(CreateSuspended)
    {
    }
    //---------------------------------------------------------------------------
    void __fastcall TChrtBuilder::Execute()
    {
    addseries(Form1->Chart1,StrToInt(Form1->Edit5->Text),StrToInt(Form1->Edit6-
>Text),Form1->CheckBox3->Checked,Form1->prime_list,Form1->CheckBox4-
>Checked,StrToInt(Form1->Edit7->Text),StrToInt(Form1->Edit8->Text),Form1->CheckBox1-
>Checked,Form1->CheckBox2->Checked,Form1->Color1);
    Form1->ListBox1->Clear();
    for (int i=0;i<Form1->Chart1->SeriesCount();i++){
    Form1->ListBox1->Items->Add("series "+IntToStr(i));
    };
    }
    //---------------------------------------------------------------------------
```

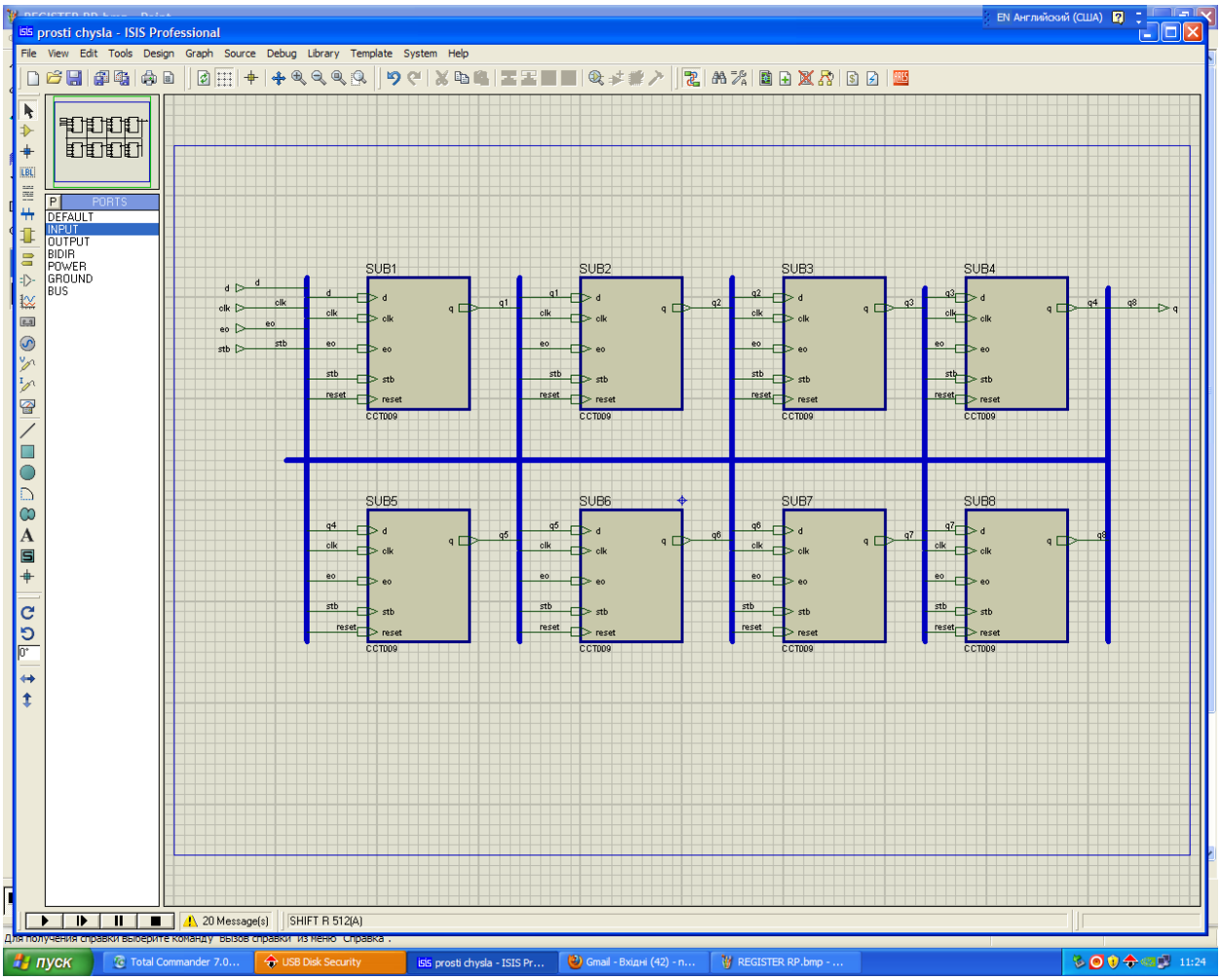
Додаток В

Схема високорозрядного спецпроцесора пошуку залишків по заданому модулю



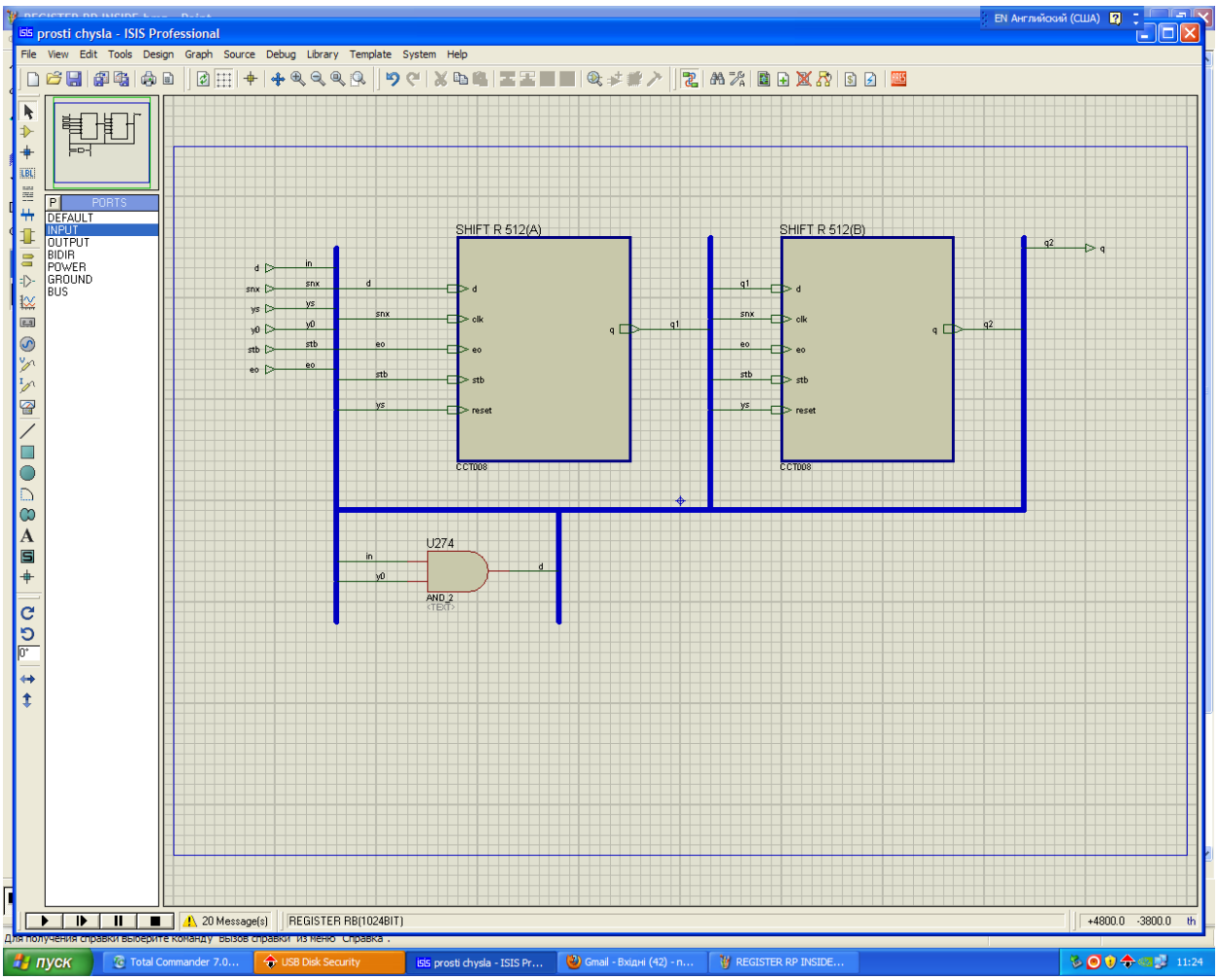
Додаток Г

Схема блоку управління



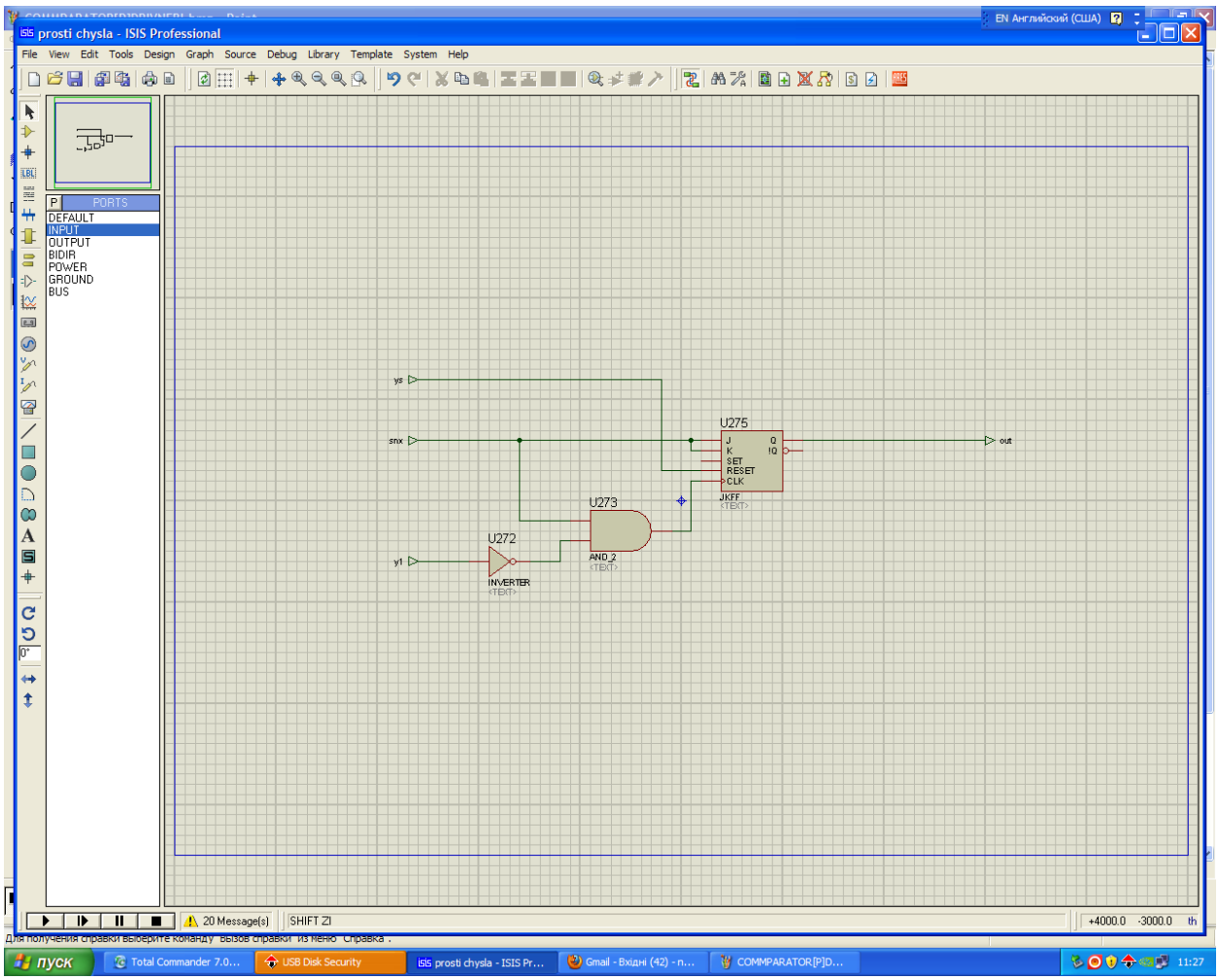
Додаток Д

Схема блоку пам’яті RB



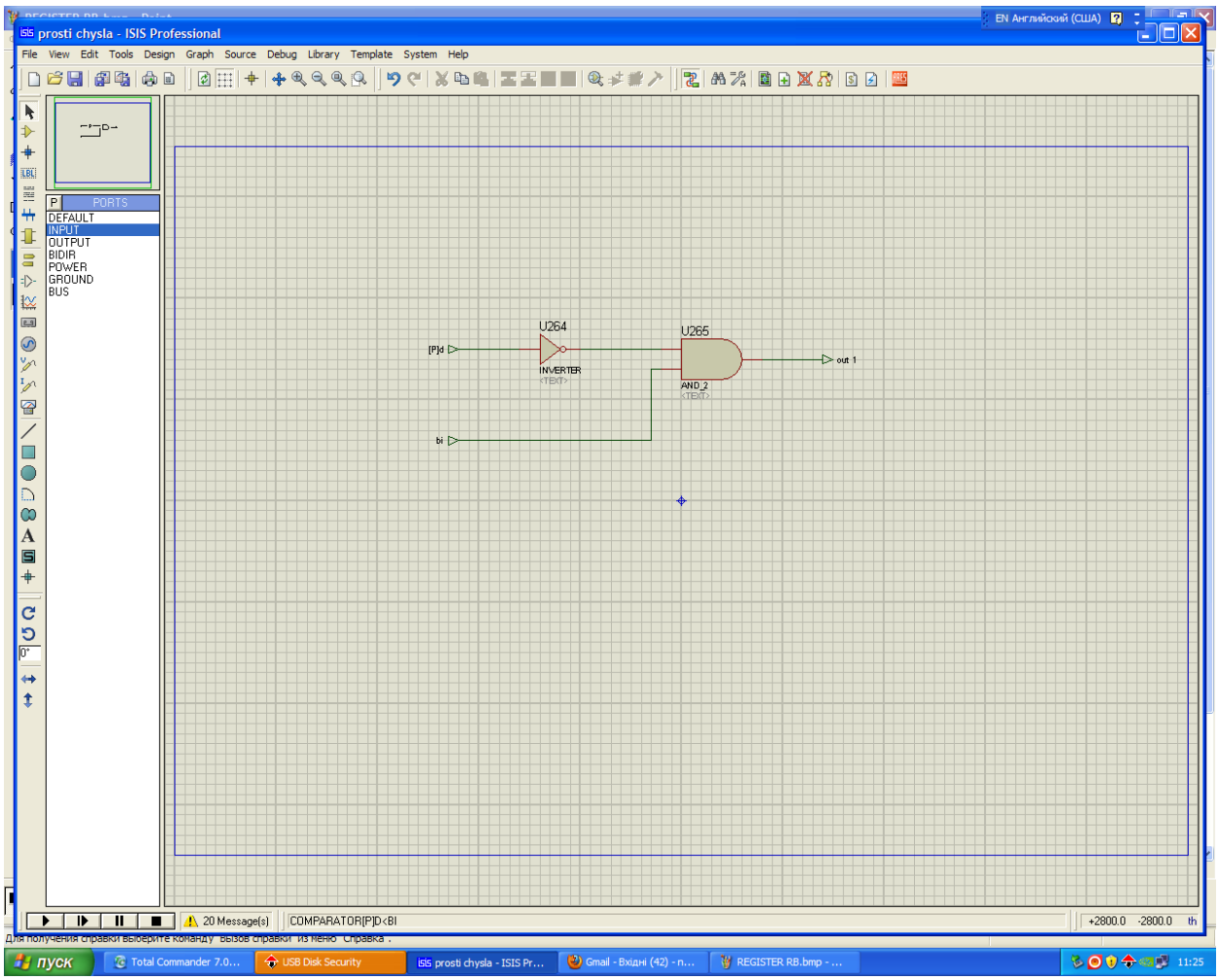
Додаток Е

Схема регістру зсуву



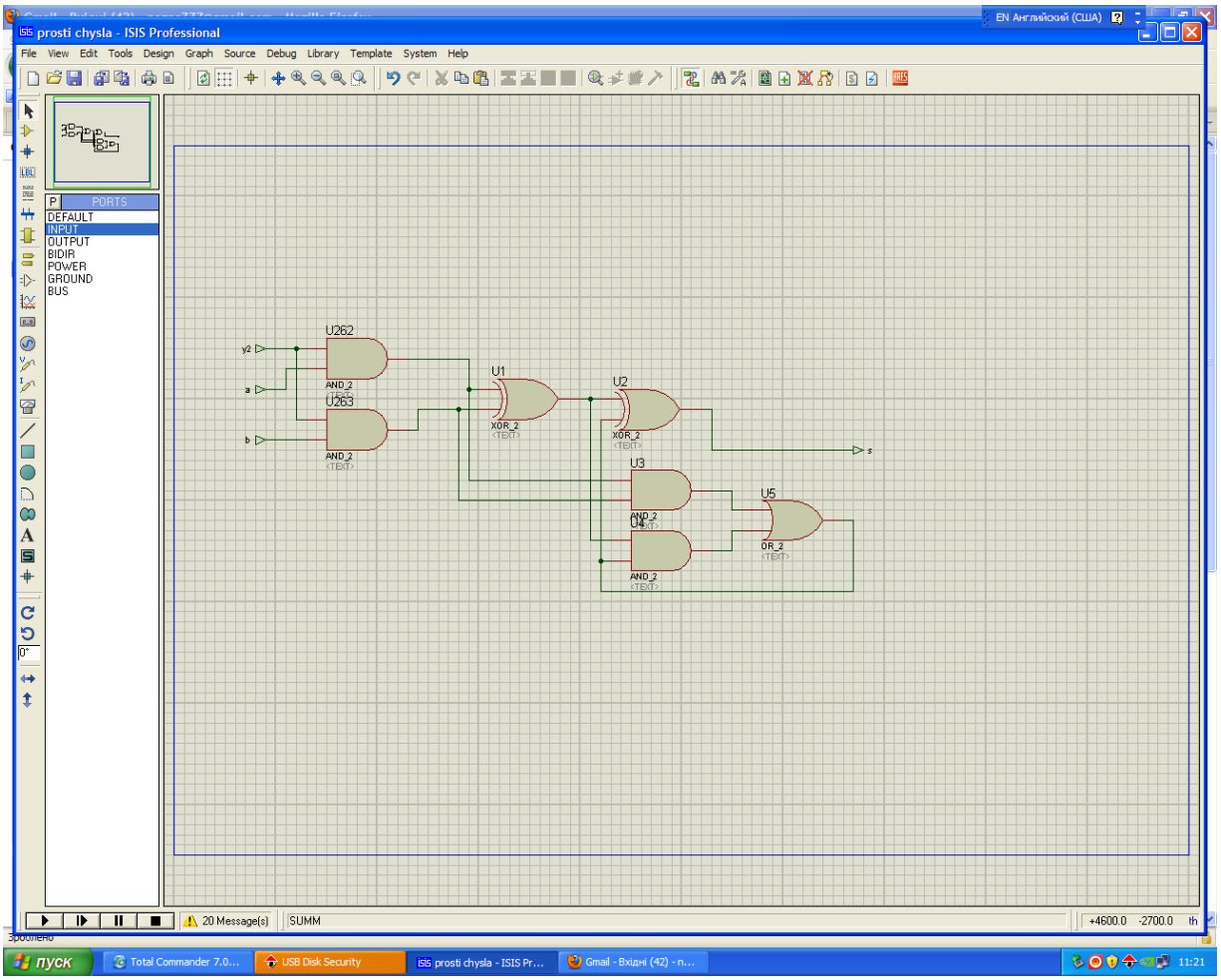
Додаток Ж

Схема цифрового компаратора



Додаток І

Схема суматора



Додаток Ж

Текст головно модуля, який реалізує побудову діаграм та зв’язок з іншими модулями

```
//---------------------------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
#include "Unit5.h"
#include <stdio.h>
#include <fstream>
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
big_prime *prime;
TChrtBuilder *ChartBuilder1;
remaining *remainingtablebuilder1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{

}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button2Click(TObject *Sender)
{
prime->Suspend();
Label2->Caption=IntToStr(prime->inumber);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{

 prime= new big_prime(false);

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Close2Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Close1Click(TObject *Sender)
{
 if (SaveDialog1->Execute()){TestingMemo->Lines->SaveToFile(SaveDialog1->FileName);}
}
//---------------------------------------------------------------------------




void __fastcall TForm1::Button4Click(TObject *Sender)
{
prime->Resume();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->TestingMemo->Clear();
 TDateTime DateTime = Time();  // store the current date and time
  AnsiString str = TimeToStr(DateTime); // convert the time to a string
  Form1->TestingMemo->Lines->Add(str);
//testing_number(StrToInt(Form1->Edit3->Text),prime_list,Form1->TestingMemo);
if (prime)prime->from_vectro_to_memo(Form1->TestingMemo,prime->prime_list);
DateTime = Time();
str = TimeToStr(DateTime); // convert the time to a string
Form1->TestingMemo->Lines->Add(str);
//if (prime)prime_list=prime->prime_list;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{
Memo1->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button6Click(TObject *Sender)
{
bool isprime=true;
bool limitout=true;
long i=1;
long num=StrToInt(Edit1->Text);
  //Memo1->Lines->Add(IntToStr(prime->prime_list.size()));
long temp;//= new long;
while (limitout) {
if (i<prime->prime_list.size()){
//Memo1->Lines->Add(IntToStr(prime->prime_list[i]));
        if (prime->prime_list[i]<num){
        temp=(num-prime->prime_list[i]);
        Memo1->Lines->Add(IntToStr(temp));

                // for (int j=0;j==0; *temp>>=1, j++){
                 if (temp&(1<<0)?1:0){
```

```
                isprime=false;
                Memo1->Lines->Add(IntToStr(0000000000000));
                };
               // };
        i++;

        }else{limitout=false;};

}else{limitout=false;};

};
if (isprime){Label4->Caption="Число просте ...";}else{Label4->Caption=" ";};
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button8Click(TObject *Sender)
{
//thre = new remaining(false);
}
//---------------------------------------------------------------------------


void __fastcall TForm1::ScrollBar1Change(TObject *Sender)
{
Chart1->Chart3DPercent=ScrollBar1->Position;

}
//---------------------------------------------------------------------------

void __fastcall TForm1::ScrollBar2Change(TObject *Sender)
{
if (ScrollBar1->Enabled ){
Chart1->View3DOptions->Orthogonal=false;
Chart1->View3DOptions->Rotation=ScrollBar2->Position;
};
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ScrollBar3Change(TObject *Sender)
{

if (ScrollBar1->Enabled) {
   Chart1->View3DOptions->Orthogonal=false;
   Chart1->View3DOptions->Elevation=ScrollBar3->Position;
  }

}
//---------------------------------------------------------------------------

void __fastcall TForm1::CheckBox5Click(TObject *Sender)
{
Chart1->View3D=CheckBox5->Checked;
ScrollBar1->Enabled=Form1->Chart1->View3D;
```

```
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button11Click(TObject *Sender)
{
ListBox1->Clear();
for (int i=0;i<Chart1->SeriesCount();i++){
ListBox1->Items->Add("series "+IntToStr(i));

};


}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button10Click(TObject *Sender)
{

Chart1->RemoveSeries(Chart1->Series[ListBox1->ItemIndex]);
// Chart1->Series[ListBox1->ItemIndex]->DestroyComponents();
ListBox1->Clear();
for (int i=0;i<Chart1->SeriesCount();i++){
ListBox1->Items->Add("series "+IntToStr(i));

};
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button9Click(TObject *Sender)
{
  ChartBuilder1= new TChrtBuilder(false);



}
//---------------------------------------------------------------------------



void __fastcall TForm1::Button12Click(TObject *Sender)
{
///Edit13->Text=IntToStr(GetChargePartNumber(StrToInt(Edit10->Text),StrToInt(Edit11-
>Text),StrToInt(Edit12->Text)));

}
//---------------------------------------------------------------------------


void __fastcall TForm1::Chart1Zoom(TObject *Sender)
{
```

```
ScrollBar4->Position=Chart1->View3DOptions->Zoom;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ScrollBar4Change(TObject *Sender)
{
Chart1->View3DOptions->Zoom=ScrollBar4->Position;
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button7Click(TObject *Sender)
{

//void from_vector_to_table_string(vector<long>& mas1,TStringGrid *StringGrid1){
remainingtablebuilder1=new remaining(false);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Mainfunctions1Click(TObject *Sender)
{
Form5->Show();
}
//---------------------------------------------------------------------------
void addtofile(vector<long>& temp){
 Form1->ProgressBar1->Max=temp.size();
 ofstream f("prime.dat",ios::binary);
 for (int i=0;i<temp.size();i++){
 f.write (reinterpret_cast<const char *>(&temp[i]),sizeof(long));
 Form1->ProgressBar1->Position=i;
 };
 //fclose (f)
 f.close();
};


void __fastcall TForm1::Button13Click(TObject *Sender)
{
addtofile(prime_list);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button14Click(TObject *Sender)
{
FILE *f;
// Form1->ProgressBar1->Max=temp.size();
 //ifstream f("curent_primes.dat",ios::binary);
 long temp;
 f = fopen("curent_primes.dat", "rb");
 while(!feof(f)){


//f.read(reinterpret_cast< char *>(&temp),sizeof(long));
fread(&temp,sizeof(long),1,f);
```

```
 Form1->TestingMemo->Lines->Add(IntToStr(temp));
 };
 fclose(f);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
// prime= new big_prime(false);
}
//---------------------------------------------------------------------------
```

Текст модуля, що забезпечує створення потоку для перевірки числа на простоту, збереження результату в файл, медулярне множення

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit2.h"
#include "Unit1.h"

#pragma package(smart_init)
//---------------------------------------------------------------------------

//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall big_prime::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------
//***************************************************************************

//***************************************************************************
void big_prime::from_memo_to_vector(TMemo *temp1,vector<long>& temp2){
temp2.resize(temp1->Lines->Count);
for (int i=0;i<temp2.size();i++){
temp2[i]=StrToInt(temp1->Lines->Strings[i]);
};
};
//***************************************************************************
String big_prime::from_vector_to_string(vector<bool>& temp2){
String s="";
for (int i=0;i<temp2.size();i++){
if (temp2[i]==0) {s=s+'0';} else {s=s+'1';};
}; return s;};

//***************************************************************************
bool big_prime::from_long_to_bit_vector(long num,vector<bool>& temp1)
{
  int i;
  for (i=0; num; num>>=1, i++){
  temp1.insert(temp1.begin(),bool((num&1)?(1):(0)));
  };
};

//***************************************************************************
bool big_prime::from_long_to_bit_vector_reverse(long num,vector<bool>& temp1)
{
  int i;
```

```
  for (i=0; num; num>>=1, i++){
  temp1.push_back(bool((num&1)?(1):(0))) ;
  };
};
//***************************************************************************

long  big_prime::mod(long a,long p){
//if (a<1000){return a%p;}else{
/*long sum=0;
for (int i=0; a; a>>=1, i++){
if (bool((a&1)?(1):(0))==1){
sum=sum+(long(pow(2,i)) % (p));
};
};

//return sum%p;*/
return a%p;
};
//};

//***************************************************************************

void big_prime::from_vectro_to_memo(TMemo *temp1,vector<long>& temp2){
AnsiString s;
char string[25];

for (int i=0;i<temp2.size();i++){
//temp2[i]
itoa(temp2[i], string, 2);
s=string;
temp1->Lines->Add(IntToStr(temp2[i])+" "+s);
};

};
//***************************************************************************
bool big_prime::mod_test(vector<long>& mas1,vector<long>& mas2,vector<long>& mas3,
long squarelimit){
bool have_null;
bool limit_out=false;
long temp;
mas3.resize(mas1.size());
for (int i=0;(i<mas1.size())&&(!limit_out);i++){
limit_out=(mas2[i]>squarelimit+1);
temp=(mas1[i]) % (mas2[i]);
if (temp==0)have_null=true;

if (temp==0){
mas3[i]=mas1[i]+2;
}else {
mas3[i]=temp;
};
};

return have_null;
```

```
};
//*************************************************************************
bool big_prime::mod_test(vector<long>& mas1,vector<long>& mas2,vector<long>& mas3){
bool have_null;
long temp;
mas3.resize(mas1.size());
for (int i=0;(i<mas1.size());i++){
temp=(mas1[i]) % (mas2[i]);
if (temp==0)have_null=true;

if (temp==0){
mas3[i]=mas1[i]+2;
}else {
mas3[i]=temp;
};
};

return have_null;
};
//*************************************************************************
long big_prime::modmul(long a,long b,long p){
long sum=0;
for (int i=1; a; a>>=1, i++){
if (bool((a&1)?(1):(0))==1){
   for (int i1=1; b; b>>=1, i1++){
      if (bool((b&1)?(1):(0))==1){
      sum=sum+(long(pow(2,i+i1)) % (p));
      };
   };
//sum=sum+(long(pow(2,i)) % (p));
};
};
return sum%p;
};
//*************************************************************************
bool big_prime::mod_test(long num1,vector<long>& mas1,vector<long>& mas2){
long num2=sqrt(num1);
vector<long> mas3;
mas3.resize(mas1.size());
for(int i=0;i<mas1.size();i++){
mas3[i]=num1;
};
return mod_test(mas3,mas1,mas2,num2);
};
//*************************************************************************
void big_prime::fill_vector_from_interval(long num1,long num2,vector<long>& mas1){
for(long i=0;i<num2-num1;i++){
mas1[i]=num1+i;
};
};

bool big_prime::rulepermit(long inumber){
bool is=true;
for (int i;i<rules.size();i++){
```

```
if ((inumber-rules[i].number)==rules[i].modul){
rules[i].number=inumber;
is=false;
};

};

return is;
};
void big_prime::set_prime_numbers(vector<long>& prime_numbers){
prime_numbers.resize(3);
prime_numbers[0]=2;
prime_numbers[1]=3;
prime_numbers[2]=5;

};


void big_prime::fill_vector_by_number(long num,vector<long>& mas1,long size){
mas1.resize(size);
for(int i=0;i<mas1.size();i++){mas1[i]=num;};
};


//---------------------------------------------------------------------------
void big_prime::testing_number(long number,vector<long>& prime_numbers,TMemo *temp1){
  FILE *fi;
  bool have_null;
  bool limit_out=false;
  long temp,squarelimit,y;
  long x;

  if (FileExists("curent_primes.dat")){
  prime_numbers.clear();
  set_prime_numbers(prime_numbers);
  Form1->TestingMemo->Lines->Add("start reading from file ");

   fi = fopen ("curent_primes.dat" , "rb");

  //ifstream fi("curent_primes.dat",ios::in);
  while(! feof (fi) ){
//  if (fi.read(reinterpret_cast< char *>(&x),sizeof(long))) prime_numbers.push_back(x);
  if (fread(&x,sizeof(long),1,fi)) prime_numbers.push_back(x);

  //Form1->TestingMemo->Lines->Add("reading from file "+ IntToStr(x));
  };
  fclose(fi);
  Form1->TestingMemo->Lines->Add("end element from file "+ IntToStr(x));
  }else{
  set_prime_numbers(prime_numbers);
  };

  Form1->ProgressBar1->Max=number;
 // ofstream fo("curent_primes.dat",ios::app);
```

```
  fi = fopen ("curent_primes.dat" , "ab");

Form1->TestingMemo->Lines->Add("Start find and save to file from  "+
IntToStr(prime_numbers[prime_numbers.size()-1]));

for (inumber=prime_numbers[prime_numbers.size()-
1]+2;inumber<=number;inumber=inumber+2){
Form1->ProgressBar1->Position=inumber;

        have_null=false;
        limit_out=false;
        squarelimit=int(sqrt(inumber))+1;
        int i=0;
          while (!limit_out)
          {
          limit_out=(prime_numbers[i+1]>squarelimit);
          if (mod(inumber,(prime_numbers[i]))==0) {
          have_null=true;
          limit_out=true;
          };
          i++;
          };

          if (!have_null){
          prime_numbers.push_back(inumber);

          //fo.seekp(1,ios_base::end);
//          fo.write(reinterpret_cast<const char *>(&inumber),sizeof(long));
          fwrite(&inumber,sizeof(long),1,fi);
  //        fo.write(&inumber,sizeof(long));

          };


};
//fo.close();
fclose(fi);
};




__fastcall big_prime::big_prime(bool CreateSuspended)
        : TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
void __fastcall big_prime::Execute()
{
    Form1->TestingMemo->Clear();
 TDateTime DateTime = Time();  // store the current date and time
  AnsiString str = TimeToStr(DateTime); // convert the time to a string
  Form1->TestingMemo->Lines->Add(str);
testing_number(StrToInt(Form1->Edit3->Text),prime_list,Form1->TestingMemo);
```

```
//from_vectro_to_memo(Form1->TestingMemo,prime_list);
DateTime = Time();
str = TimeToStr(DateTime); // convert the time to a string
Form1->TestingMemo->Lines->Add(str);
Form1->prime_list=prime_list;
}
long big_prime::modexp(long a,long b,long p){

};
//---------------------------------------------------------------------------
```

Модуль для створення таблиці залишкових класів

```
#include <vcl.h>
#pragma hdrstop

#include "Unit3.h"
#include "Unit1.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall remaining::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------
long remaining::mod(long a,long b){
long temp=a%b;
return temp;
};
//---------------------------------------------------------------------------

void remaining::from_vector_to_table_string(vector<long>& mas1,TStringGrid *StringGrid1){
for(int i=0;i<mas1.size();i++){
if (i>=StringGrid1->ColCount)StringGrid1->ColCount++;
StringGrid1->Cells[i][StringGrid1->RowCount]=IntToStr(mas1[i]);};
StringGrid1->RowCount++;
};

void remaining::get_remainders(long n,long prime,vector<long>& remainders){
//remainders.resize(n+1);
bool haveretry=false;
int i=1;
remainders.resize(2);
remainders[0]=prime;
remainders[i]=mod(pow(2,i),prime);


while ((i<=n)&&(haveretry==false)){
i++;
remainders.resize(remainders.size()+1);
remainders[i]=mod(pow(2,i),prime);
haveretry=remainders[i]==remainders[1];

};
};

void remaining::build_table_remainder(long maxn,long maxprime,vector<long>&
primes,TStringGrid *StringGrid1){
```

```
StringGrid1->Cells[0][StringGrid1->RowCount]="Прості числа";
for(int i=0;i<=maxn;i++){
if (i>=StringGrid1->ColCount)StringGrid1->ColCount++;
if (i>0)StringGrid1->Cells[i][StringGrid1->RowCount]="2^"+IntToStr(i);
};
StringGrid1->RowCount++;



       int i=0;
       vector<long> remainders;
       while (i<primes.size()&&primes[i]<=maxprime){
     //  Form1->Memo3->Lines->Add("Get remainders for "+IntToStr(primes[i]));
       get_remainders(maxn,primes[i],remainders);
     //  Form1->Memo3->Lines->Add("Size of remainder "+IntToStr(remainders.size()));
       from_vector_to_table_string(remainders,StringGrid1);
       i++;
      // remainders.
       };
};


__fastcall remaining::remaining(bool CreateSuspended)
       : TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
void __fastcall remaining::Execute()
{
Form1->StringGrid1->ColCount=1;
Form1->StringGrid1->RowCount=1;
//get_remainders(StrToInt(Form1->Edit4->Text),3,remainders_table[0].remainders);
 build_table_remainder(StrToInt(Form1->Edit2->Text),StrToInt(Form1->Edit4->Text),Form1-
>prime_list,Form1->StringGrid1);
}
```

Модуль для побудови діаграм

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit4.h"
#include "Unit1.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall TChrtBuilder::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }


long TChrtBuilder::GetChargePartNumber(long number,int charge_from,int chrge_to){
long num=0;
if (charge_from<chrge_to){

char string1[(sizeof(long)*8)+2];
AnsiString s;

itoa(number,string1, 2);
s=string1;
if (chrge_to>s.Length())chrge_to=s.Length()-1;
if (charge_from<0)charge_from=0;
if (charge_from>s.Length())charge_from=s.Length();
//if (charge_from=chrge_toLength())charge_from=s.Length();
//delete string1;
s=s.SubString(s.Length()-chrge_to,chrge_to-charge_from+1);

//Form1->Memo2->Lines->Add(string1);
//Form1->Memo2->Lines->Add(s1);
//Form1->Memo2->Lines->Add("length = "+IntToStr(s1.Length()));
for (int i =1;i<=s.Length();i++){
if (s.SubString(i,1)=='1'){
       num=num+pow(2,s.Length()-i);
      // Form1->Memo2->Lines->Add("s1[i] = "+s1.SubString(i,1));
      // Form1->Memo2->Lines->Add("power = "+IntToStr(s1.Length()-i));
};
};};
return num;


};
```

```
//---------------------------------------------------------------------------
void TChrtBuilder::addseries(TChart *Chart1,long from,long to,bool prime_only,vector<long>&
primes,bool discharge_limit,int charge_from,int chrge_to){
TLineSeries *series1;
series1 = new TLineSeries(Chart1);
//series1->Name=name;
int i=0;
if (prime_only){

      while (i<=primes.size()&&primes[i]<=to){
            if (primes[i]>from&&primes[i]<to){
                  if (discharge_limit==false){
                  series1->Add(primes[i],primes[i],clRed);}else{
                  series1-
>Add(GetChargePartNumber(primes[i],charge_from,chrge_to),GetChargePartNumber(primes[i]
,charge_from,chrge_to),clGreen);
                  };
            };
      i++;
      };

}else{
      for(i=from;i<=to;i++){
            series1->Add(i,clBlue);
      };

};


//series1->Marks->Style;
series1->Marks->Visible=true;
Chart1->AddSeries(series1);



};

__fastcall TChrtBuilder::TChrtBuilder(bool CreateSuspended)
      : TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
void __fastcall TChrtBuilder::Execute()
{
addseries(Form1->Chart1,StrToInt(Form1->Edit5->Text),StrToInt(Form1->Edit6-
>Text),Form1->CheckBox3->Checked,Form1->prime_list,Form1->CheckBox4-
>Checked,StrToInt(Form1->Edit7->Text),StrToInt(Form1->Edit8->Text));
Form1->ListBox1->Clear();
for (int i=0;i<Form1->Chart1->SeriesCount();i++){
Form1->ListBox1->Items->Add("series "+IntToStr(i));
};
}
//---------------------------------------------------------------------------
```