НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Кафедра математичних методів захисту інформації

«До захисту допущено»
В.о. завідувача кафедри
_____ Михайло САВЧУК
«___» _____ 2021р.

Дипломна робота
на здобуття ступеня бакалавра

зі спеціальності: 113 Прикладна математика

на тему:«Дослідження системи шифрування Бенало та її застосування для контролю цілісності делегованих обчислень»

Виконав: студент 4 курсу, групи ФІ-73
Морозюк Анастасія Олексіївна

Керівник: д.ф.-м.н., доц., в.о. зав. каф. ММЗІ Савчук М.М._____

Консультант: _ _____

Рецензент: звання, степінь, посада Прізвище І.П. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ — 2021

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Кафедра математичних методів захисту інформації

Рівень вищої освіти — перший (бакалаврський)

Спеціальність (освітня програма) — 113 Прикладна математика,

ОПП «Математичні методи криптографічного захисту інформації»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Михайло САВЧУК

«___» _____ 2021р.

ЗАВДАННЯ
на дипломну роботу

Студент Морозюк Анастасія Олексіївна

1. Тема роботи: «Дослідження системи шифрування Бенало та її застосування для контролю цілісності делегованих обчислень»,
Керівник: д.ф.-м.н., доц., в.о. зав. каф. ММЗІ Савчук М.М.,

затверджені наказом по університету №___ від «___» _____ 2021р.

2. Термін подання студентом роботи: 4 червня 2021р.

3. Вихідні дані до роботи: гомоморфне шифрування, криптосистеми RSA, Ель-Гамаля, Бенало, алгоритми контролю цілісності делегованих обчислень.

4. Зміст роботи: побудова програмної реалізації криптосистеми Бенало та алгоритмів контролю цілісності делегованих обчислень, дослідження робудованих систем на предмет наявності вразливостей, оцінки розшифрування в схемі Бенало.

5. Перелік ілюстративного матеріалу: Презентація доповіді.

6. Дата видачі завдання: 10 вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2020 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-жовтень 2020 р.	Виконано
3	Побудова програмної реалізації криптосистеми Бенало	Листопад-грудень 2020 р.	Виконано
4	Побудова програмної реалізації схеми контролю цілісності делегованих обчислень	Листопад-грудень 2020 р.	Виконано
5	Проведення пошуку недоліків криптосистеми Бенало	Січень-березень 2021 р.	Виконано
6	Побудова фактичних оцінок швидкості розшифрування повідомлень різного розміру	Січень-березень 2021 р.	Виконано
7	Проведення пошуку можливих вразливостей схеми контролю цілісності делегованих обчислень	Січень-березень 2021 р.	Виконано
8	Аналіз отриманих результатів та їх оформлення	Квітень-травень 2021 р.	Виконано

Студент _____ Морозюк А.О.

Керівник _____ Савчук М.М.

РЕФЕРАТ

Кваліфікаційна робота містить: 49 стор., 20 рисунків, 16 джерел.

Дана робота присвячена дослідженню криптосистеми Бенало та її використанню для контролю цілісності делегованих обчислень.

Об'єктом дослідження є інформаційні процеси в системах криптографічного захисту інформації.

Предметом дослідження є математичні моделі та алгоритми криптографічного захисту інформації з використанням криптографічної системи Бенало.

В роботі проведено огляд та аналіз гомоморфних криптографічних систем, які можна використовувати для спеціальних цілей, зокрема для перевірки цілісності делегованих обчислень. Особливо детально розглянута система шифрування Бенало. Її реалізовано програмно на великих числах та проведено експерементальні дослідження направлені на пошук вразливостей системи та уточнення часової складності розшифрування. В роботі також реалізовано систему перевірки цілісності делегованих обчислень на великих числах, яка використовує систему Бенало, та проведено експеремнтальні дослідження стосовно виявлення можливих слабких місць цієї системи.

ГОМОМОРФНІ КРИПТОСИСТЕМИ, СИСТЕМА БЕНАЛО, АЛГОРИТМИ ПЕРЕВІРКИ ДЕЛЕГОВАНИХ ОБЧИСЛЕНЬ

ABSTRACT

The thesis contains: 49 p., 20 figures, 16 references. The aim of the thesis is to do research of the Benalo cryptosystem and its use to control the integrity of delegated computations.

The object of research is information processes in systems of cryptographic protection of information.

The subject of research is mathematical models and algorithms of cryptographic protection of information using the Benalo cryptographic system.

The thesis reviews and analyzes homomorphic cryptographic systems that can be used for special purposes, in particular to verify the integrity of delegated computations. The Benalo encryption system is considered in particular detail. It has been implemented programmatically on large numbers and experimental studies have been conducted aimed at finding system vulnerabilities and clarifying the time complexity of decoding. The paper also implements a system for checking the integrity of delegated computations on large numbers, which uses the Benalo system, and conducted experimental studies to identify possible weaknesses of this system.

HOMOMORPHIC CRYPTOSYSTEMS, BENALOH SYSTEM, DELEGATED COMPUTATIONS INTEGRITY CONTROL ALGORITHMS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

НСД(a, b) - найбільший спільний дільник чисел a та b

$\mathbb{Z}$ - множина цілих чисел

$\mathbb{Z}_n$ - кільце лишків за модулем числа n

$\mathbb{Z}_n^*$ - мультиплікативна група кільця лишків за модулем n

ВСТУП

Актуальність дослідження.

У сучасному світі все більше і більше використовуються хмарні сервіси, коли користувач, не маючи в своєму розпорядженні значних обчислювальних потужностей відправляє свої дані на обробку на віддалені обчислювальні системи або ресурси. І хоча ідея хмарних обчислень висловлювалася ще в 60 роках 20 століття, фактичне їх використання почалося лише в кінці нульових років 21 століття, з моменту, коли швидкість каналів зв'язку дозволила передавати оперативно великі обсяги даних. Хмарні обчислення - один із стабільно зростаючих сегментів в IT галузі, який продовжує розширюватися. Наразі хмарні сервіси використовують як великі компанії, підприємства, так і приватні особи. Це обумовлено тим, що користувачеві потрібно здійснювати отплату лише за використанні ресурси та легко змінювати їх об'єм в залежності від потреб. Насправді необхідність віддалених обчислень зростає не тільки через недостатність ресурсів у локальній ЕОМ. З'явилося безліч компактних малопотужних мікро-ЕОМ і пристроїв здатних збирати дані, але апаратно нездатних їх обробляти. Тому, за сукупністю зазначених причин ринок хмарних обчислень стабільно зростає. І саме в випадку хмарних обчислень стає особливо актуальним захист інформації. Адже по суті ми делегуємо обробку даних третім особам (хмарі або серверу), які не мають довіри. Адже потік даних може бути скомпрометований зловмисником як під час передачі на хмару, отриманні результатів з хмари так і на етапі проведення обчислень. І якщо на етапі передачі і прийому даних можна скористатися, наприклад, одним з алгоритмів шифрування з відкритими ключами, то при розшифровці даних користувача всередині хмари, для подальшої обробки, дані ставляться легко доступними для атаки. Одним із рішенн для запобігання порушення цілісності даних є застосування концепції гомоморфного шифрування, суть якого полягає в можливості виконувати будь-які обчислення на зашифрованих даних, обробляти і аналізувати ці дані, при цьому не розшифровуючи їх.

Також постійно зв'являються нові системи шифрування, які потребують додаткових досліджень. Однією з таких систем є аддитивно гомоморфна криптосистема Бенало. Саме система Бенало ще мало досліджена, особливо з точку зору її недоліків та швидкості розшифрування, хоча вже і використовується в електронному голосуванні.

Метою дослідження є дослідження криптосистеми Бенало та її використання для контролю цілісності делегованих обчислень. Для досягнення мети необхідно розв'язати **задачу дослідження**, яка полягає побудові даних систем та проведенні на них експерементальних досліджень. Для розв'язання задачі необхідно вирішити такі завдання:

1) провести огляд опублікованих джерел за тематикою дослідження;

2) побудувати програмну реалізацію криптосистеми Бенало;

3) провести пошук недоліків криптосистеми Бенало;

4) отримати фактичні оцінки швидкості розшифрування повідомлень різного розміру;

5) побудувати програмну реалізацію схему контролю цілісності делегованих обчислень;

6) провести пошук можливих вразливостей схеми контролю цілісності делегованих обчислень.

Об'єктом дослідження є інформаційні процеси в системах криптографічного захисту інформації.

Предметом дослідження є математичні моделі та алгоритми криптографічного захисту інформації з використанням криптографічної системи Бенало.

При розв'язанні поставлених завдань використовувались такі *методи дослідження*: математичні алгоритми, криптографічні алгоритми, машинні експерименти і методи теорії складності алгоритмів.

Наукова новизна отриманих результатів полягає в практичному підтвердженні теоретичних оцінок ймовірностей появи недоліків в системі Бенало та в підході до вибору оптимальних параметрів для швидкого розшифрування. Бенало та ймовірності отримати неоднозначність розшифруваннння.

Практичне значення результатів полягає в отриманні

практичного підтвердження оцінок складності розшифрування системи та ймовірностей появи вразливостей системи. Було також виявлено розмір блоку повідомлення, який розшифровується за прийнятний час. На підставі проведено дослідження можна сформувати нові задачі, аби отримати більш точні оцінки ймовірностей неоднозначності розшифрування в криптосистемі Бенало та ймовірності підробити перевірку правильності делегованих обчислень.

1 ТЕОРИТИЧНІ ВІДОМОСТІ

В данному розділі пропонується розглянути основні теоретичні положення, які будуть використовуватись в даній роботі, а також результати досліджень, які проводились на дану тематику.

1.1 Проблематика делегування обчислень віддаленим пристроям

На сучасному етапі розвитку обчислювально техніки і засобів зв'язку виникла ситуація, коли необов'язково мати високі обчислювальні потужності локально. Досить легко передати складні обчислення на віддалений високопродуктивний сервіс і провести обчислення на ньому. Та незважаючи на те що, передача даних на зовнішній ресурс, за замовчуванням, передбачає високу ступінь довіри до такого ресурсу, виникає проблема. Проблематика делегування обчислень віддаленим пристроям полягає в наступному : По перше користувач (клієнт) делегує віддаленому пристрою(серверу) певні обчислення та вхідні дані, але найважливішим моментом є необхідність перевірити цілісніть результату обчислень без повторення самого процесу обчислень. По друге вважається, що клієнт не може довіряти серверу і повинен вжити певні заходи аби забеспечити цілісність даних перед їх передачею. Зокрема, можуть виникати наступні типи загроз:

– зловмисник може спотворити вхідні дані(додати нові дані, змінити значення інуючих даних, знищити дані частково або повністю);

– зловмисник може спортворити процес обчислення, алгоритми;

– зловмисник може спотворити дані, які передаються клієнту після виконання обчислень.

Очевидно, що передавати дані в початковому вигляді на сервер небезпечно, треба вжити певних заходів для підтвердження їх цілісності. В даній роботі розлядається попередня обробка даних за допомогою гомоморфного шифрування

1.2 Гомоморфне шифрування

Означення 1.1. *Гомоморфною криптографією* називають розділ криптографії, присвячений схемам, які допускають виконання обчислень на шифротекстах, а відповідні їм схеми шифрування - *гомоморфними* [15].

При цьому розрізняють частково гомоморфні схеми та повністю гомоморфні схеми.

Означення 1.2. Криптографічні схеми, в яких операції додавання та множення шифротекстів є гомоморфними, тобто виконуються наступні рівності:
$D(E(m_1) * E(m_2)) = m_1 * m_2$,
$D(E(m_1) + E(m_2)) = m_1 + m_2$,
де $E(\cdot)$ — функція зашифрування, а $D(\cdot)$ — функція розшифрування, називаються *повністю гомоморфними.*

Означення 1.3. Криптографічні системи, в яких хоча б одна із операцій множення або додавання є гомоморфною називаються *частково гомоморфними.*

Прикладами частково гомоморфних систем можуть слугувати схеми RSA, Ель-Гамаля, які є гомоморфними відносно опрації множення , та схема Бенало, яка є гомоморфною відносно операції додавання.

У 1978 році автори RSA Рональд Ріверст і Леонард Адлеман разом із Майклом Дертузосом вперше довели, що методом, що дозволяє успішно проводити операції над зашифрованими даними, не спотворюючи і не розшифровуючи їх, є гомоморфне шифрування [13]. У своїй роботі вони вперше описали концепцію повністю гомоморфного шифрування, а також задалися питаннями, чи можливо таке шифрування взагалі.

Перша повністю гомоморфна система була запропонована Крейгом Джентрі в 2009 році [5]. Головним мінусом запропонованої криптосистеми є великий час обробки даних. Швидкість обчислень настільки низька, що говорити про будь-яке практичне застосування було неможливо. Наразі існує вже 3 покоління повністю гомоморфного шифрування і робота над

ними триває. У 2020 році компанія IBM випустила набір публічних інструментів, які спрощують гомоморфність шифрування для iOS і macOS. Швидкість обробки зростає і є надія що алгоритми повністю гомоморфного шифрування будуть впроваджені в наше життя.

1.3 Криптосистема RSA

Криптосистема RSA була запропонована Р. Рівестом, А. Шаміром і Л. Адлеманом [12] в 1978 році та є одним із перших прикладів гомоморфного шифрування. Складність цієї схеми базується на задачі факторизації числа, яке є добутком двох великих простих чисел.

Генерування ключів

Наведемо етапи генерації ключів в криптосистемі RSA [12].

1) Обираються два великі прості числа p та q;

2) Обчислити $n = p * q$.

3) Обчислюється $\phi(n) = (p-1)(q-1)$

4) Обирається ціле число e, таке що $1 < e < \phi(n)$, причому $gcd(e, \phi(n) = 1$

5) Обчислюється $d = e^{-1} \, mod \, \phi(n)$

Тоді (n, e) становлять відкритий ключ, а (d,p,q) - секретний ключ.

Зашифрування

Наведемо схему зашифрування повідомлень в криптосистемі RSA [12].

Нехай $1 < M < n$ - повідомлення, тоді зашифрування відбувається за допомогою відкритого ключа:
$M^e \, mod \, n = C$, де C - шифротекст.

Розшифрування

Наведемо схему розшифрування в криптосистемі RSA [12].

Нехай C - шифротекст. Розшифрування відбувається за допомогою секретного ключа d:

$C^d \, mod \, n = M.$

Гомоморфність в схемі RSA [11]:
$E(M_1) * E(M_2) = (M_1^e \ mod \ n) * (M_2^e \ mod \ n) =$
$= (M_1 * M_2)^e \, mod \, n = E(M_1 * M_2)$

Як показано вище, добуток двох шифротекстів відповідає шифруванню добутків відповідних їм відкритих текстів, отже схема RSA є гомоморфною за множенням.

1.4 Криптосистема Ель-Гамаля

Нова схема шифрування з відкритим ключем була запропонована Т. Ель-Гамалем [8] в 1985 році. Схема Ель-Гамаля є модифікацією схеми Діффі-Хелмана [9] та її складність також базується на задачі дискретного логарифрування [10].

Генерування ключів

Наведемо етапи генерації ключів в схемі Ель-Гамаля [8].

1) Обирається велике просте число p та $\alpha \in F_p$.

2) Обирається випадкове число $1 < k < p$.

3) Обчислюється $y = \alpha^k \, mod \, p$

Тоді (p,α,y) складають відкритий ключ, а k - секретний ключ.

Зашифрування

Наведемо етапи зашифрування повідомлень в схемі Ель-Гамаля [8].

Нехай M - повідомлення, для зашифрування повідомлення виконуються наступні дії:

1) Обирається випадкове число $1 < x_M < p - 1$

2) Обчислити $c_1 = \alpha^{x_M} \, mod \, p$ та $c_2 = y^{x_M} M \, mod \, p$.

Сформована пара (c_1, c_2) складає шифротекст.

Розшифрування

Наведемо етапи розшифрування повідомлень в схемі Ель-Гамаля [8].

Нехай $(c_1,\ c_2)$- шифротекст. Розшифрування відбувається за допомогою секретного ключа k:
$c_1^k c_2\ mod\, p = \alpha^{-x_M k}\alpha^{k x_M}\ mod\, p = M.$

Гомоморфність в схемі Ель-Гамаля [11]:
$E(M_1)*E(M_2) = (\alpha^{x_{M_1}}, y^{x_{M_1}}M_1)*(\alpha^{x_{M_2}}, y^{x_{M_2}}M_2) = (\alpha^{x_{M_1}+x_{M_2}}, (M_1M_2)y^{x_{M_1}+x_{M_2}}) = E(M_1 * M_2)$

Як показано вище, добуток двох шифротекстів відповідає шифруванню добутків відповідних їм відкритих текстів, отже схема Ель-Гамаля є гомоморфною за множенням.

1.5 Криптосистема Бенало

Криптосистема Бенало [1] є прикладом імовірносного шифрування [6], тобто шифрування фіксованого значення за допомогою одного и того ж самого відкритого ключа може давати різні шифротексти, які при розшифруванні дають один і той самий відкритий текст. Вшерше схема імовірносного шифрування була запропонована Ш. Гольдвассером и С. Мікалі [6]. А схема Бенало є модифікацією схеми Гольдвассера-Мікалі, та дозволяє виконувати зашифрування не побітово, а цілими блоками.

Наведемо основні етапи оригінальної схеми Бенало

Генерування ключів

Наведемо схему генерації ключів в криптосистемы Бенало [1].

1) Обираються блок розміру r та два випадкові прості числа p та q, які обираються випадково, такі, що

– $(p-1)/r$ - ціле,

– $(p-1)/r$ та r - взаємнопрості,

– $(q-1)$ та r - також взаємнопрості,

– Обчислюємо $n = p * q$.

2) Обирається $y \in \mathbb{Z}_n^* = \{x \in \mathbb{Z}_n : gcd(x,n) = 1\}$ такий, що

$y^{(p-1)(q-1)/r} mod n \neq 1$.

Тоді (y,r,n) становлять відкритий ключ, а (p,q) - секретний ключ.

Зашифрування

Наведемо схему зашифрування повідомлень в криптосистемы Бенало [1].]par Нехай повідомлення $M \in \mathbb{Z}_r$, u елемент $\mathbb{Z}_n^*$ обраний випадково,тоді можемо обчислити рандомізоване зашифрування повідомлення M, використовуючи наступну формулу:

$E_r(M) = \{y^M u^r mod n : u \in \mathbb{Z}_n^*\}$.

Функція $E_r(M)$ має наступні властивості:

– знаючи повідомлення $M \in \mathbb{Z}_r$, легко обчислити $z \in E_r(M)$;

– для будь яких $M_1, M_2 \in \mathbb{Z}_r$, таких, що $M_1 \neq M_2$, $E_r(M_1) \neq E_r(M_2)$;

– припускається, що $E_r(M)$ - одностороння функція;

Розшифрування

Наведемо схему розшифрування повідомлень в криптосистемы Бенало [1].

Складність розшифрування базується на задачі знаходження лишків старших порядків [7], яка полягає в наступному: нехай дано z, r та $n = pq$, де p та q - невідомі, тоді не існує алгоритму, який визначає чи існує такий x, що $x^r = z \ mod \ n$ за поліноміальний час. Зауважимо, що n має бути таким, щоб його було важко факторизувати.

Нехай $z \in E_r(M)$

Спочатку зазначимо, що для будь-яких M, u

$z^{(p-1)(q-1)/r} \equiv (y^M u^r)^{(p-1)(q-1)/r} \equiv y^{M(p-1)(q-1)/r} u^{(p-1)(q-1)} \equiv$

$\equiv y^{M(p-1)(q-1)/r} \ mod \, n$.

Оскільки $M \in \mathbb{Z}_r$ та $y^{(p-1)(q-1)/r} \not\equiv 1 \ mod \ n$, Бенало вважає що $M = 0 \iff (y^M u^r)^{(p-1)(q-1)/r} \equiv 1 \ mod \ n$. Тоді при відомих q та p , ми можемо визначити чи $M = 0$. Якщо r невелике, ми можемо

розшифрувати z шляхом знаходження найменшого цілого невід'ємного M, такого, що $(y^{-M} z \, mod \, n) \in E_r(0)$. За допомогою попередніх обчислень, наприклад, за допомогою алгоритму алгоритма Гельфонда — Шенкса (baby-step giant-step) можна виконати розшифровку за час $O(\sqrt{r})$. Якщо ж r гладке, ми можемо використати алгоритм index-calculus.

Гомоморфність в схемі Бенало[11]:
$E_r(M_1 + M_2) = y^{M_1+M_2}(u_1 * u_2)^r \; mod \; n = = (y_1^M u_1^r \, mod \, n) * (y_2^M u_2^r \, mod \, n) = E_r(M1) * E_r(M_2)$.

Як показано вище, операція множення над будь-якими шифротекстами відповідає шифрованню суми відповідних відкритих текстів, отже схема Бенало гомоморфна за додаванням.

Недоліки в схемі Бенало

Розглянемо контрприклад, наведений в роботі Л. Фуса, П. Лафуркада та М. Алнуаймі [4]. Нехай дана криптосистема Бенало з параметрами $p = 241$, $q = 179$, $n = 241 * 179 = 43139$, $r = 15$ та $y = 27$. Можна легко превірити, що всі параметри задоволняють всім умовам генерування ключів.

Відповідно до схеми Бенало $z_1 = y^1 12^r = 24187 \; mod \; 43139$ є коректним шифротекстом повідомлення $m_1 = 1$ та $z_2 = y^6 4^r = 24187 \; mod \; 43139$ є коректним шифротекстом повідомлення $m_2 = 6$.
Таким чином в схемі Бенало має місце неоднозначність розшифрування для певних значеннь y.

За оцінками авторів [4], імовірність обрати такий y дорівнює $\rho = 1 - \frac{\phi(r)}{r-1}$. Ця імовірність залежить лише від r та нульова, якщо r просте. Проте в оригінальній схемі Бенало пропунюється обирати r, яке є добутком степенів невеликих простих чисел. Наприклад, для r виду $r = 3^k$, де k - натуральне число, алгоритм розшифрування працює досить

ефективно, але імовірність обрати "поганий" y зростає. Проте існують модифікації схеми Бенало в яких однозначність розшифрування завжди виконується.
Наразі криптосистема Бенало застосовується в електоронному голосуванні.

1.6 Обчислення на додавальній машині за допомогою схеми Бенало

В розглянутих мною роботах К. Новокшонова та А. Анісімова [3, 4, 16], в якості віддаленого обчислювального пристрою, розглядалась додавальна машина (addition machine) [14], яка була введена Д. Кнутом та Р. Флойдом. Вона являє собою обчислювальний пристрій з обмеженною кількість регістрів, над якими можна здійснювати наступні операції:

– введення даних в регістр x: $read\, x$
– присвоєння регістра y регістру x: $z \leftarrow y$
– додавання x та y: $x \leftarrow x + y$
– віднімання y від x: $x \leftarrow x - y$
– порівняння x та y: if $x \geqslant y$
– виведення даних із регістра x: write x

Хоча додавальна машина виконує лише операції додавання та віднімання, через її операції можна доволі ефективно виразити більш складні операції множення, знаходження найбільшого спільного дільника, цільчисельного ділення, піднесення до степню за модулем та знаходження лишків за модулем.

Одним із запропонованих методів вирішення проблеми контролю цілісності за допомогою гомоморфного, який дозволяє не зашифровувати повністю вхідні дані, а виконувати простішу попередню розмітку, є схема контролю цілісності делегованих обчислень, запропонована К. Новокшоновим та А. Анісімовим [3, 4, 16].

Наведемо її детальний опис. Основна ідея цієї схеми є вставка в

спеціальні місця програми зашифрованих констант - контрольних змінних. Ці константи беруть участь в формуванні цифрових підписів. Ці контрольні змінні(та їх перетворення в процесі обчислень) ніяк не впливають на коректність обчислень. Далі клієнт отримує результати обчислень від сервера та шляхом порівняння розшифрованих значень контрольних змінних та відкритих значень робить висновки про цілісніть даних(коректність виконаних обчислень).

Використовується динамічний цифровий підпис, який здійснюється за допомогою гомоморфної за додаванням криптографічної функції. Дії над цифровими підписами виконуються в додавальній машині. Цифровий підпис зміннної повинен враховувати наступні показники: поточне значення змінної, ім'я змінної та її місце в вхідних даних. Нехай f - відображення $\mathbb{Z} \rightarrow \mathbb{Z}_n^*$, будується на основі односторонньої функції.Вважаємо, що f - гомоморфне за додаванням, тобто

$f(x+y) = f(x) * f(y)$,

$f(x-y) = f(x) * f^{-1}(y)$.

Також визначена верифікаційна функція $V : \mathbb{Z}_n^* \rightarrow \mathbb{Z}_n^*$. Пару (x,y) $x \in \mathbb{Z}_n^*, y \in \mathbb{Z}_n^*$ вважаємо коректною, якщо $y = V(f(x))$.

Була обрана наступна схема побудови f на основі схеми Бенало [1]. Нехай p, q, g, r, c, a - секретні параметри, причому $(x, (p-1)(q-1)) = 1, c = (p-1)(q-1)/r$, а $n = p * q$ - відкритий ключ. Для $x \in \mathbb{Z}$ вважаємо, що $f(x) = g^{ax} b^r mod n$, де b - випадково обране число із $\mathbb{Z}_n^*$. Кожній змінній x клієнт ставить у відповідність випадкове обране фіксоване число $r_x, 0 < r_x < (p-1)(q-1)$. Всі числа r_x - секретні. Вважаємо, що знаючи значення зміної x та значення $y = f(x)$, зловмисник не може підмінити пару цих значень, оскільки в нього немає доступу до закритих параметрів f. Для захисту від спотворення імен зміних додається ідентифікуючий цифровий підпис $Id(x) = f(r_x) = g^{r_x} u^r \ mod \ n$ ($u \in \mathbb{Z}_n^*$ обирається випадково). Цифровий підпис змінної x задається формулою:

$E(x) = Id(x) f(x)$

$E(-x) = E^{-1}(x) = f^{-1}(r_x) f^{-1}(x)$

Зміна цифрового підпису після виконання присвоювання додавальної

машини задається наступним чином:

1.команда: $z \leftarrow x + y$,
підпис: $E(z) = Id(z)Id^{-1}(x)Id^{-1}(y)E(x)E(y)$

2.команда: $z \leftarrow x - y$,
підпис: $E(z) = Id(z)Id^{-1}(x)Id(y)E(x)E^{-1}(y)$

3.команда: $z \leftarrow x$,
підпис: $E(z) = Id(z)Id^{-1}(x)E(x)$

Виконання кожної команди додавальної машини [14] супроводжується попереднім цифровим підписом, відповідним до команди(присвоєння додавання, присвоєння віднімання, присвоєння). При цьому результуюча змінна z фіксується своїм ідентифікатором і цифровим підписом отриманого числового значення. Початкове значення змінної x підписується $E(x)$.

Константи, які застосовуються в програмі додавальної машини, розглядаються як змінні, приймаючи фіксоване значення. Ці значення підписуються як відповідні змінні. Введений цифровий підпис не дозволяє зловмиснику змінювати імена змінних в командах або вводити нові змінні. Для контроля цілісності команд та їх порядку при виконанні програми відбувається попередня числова розмітка всіх команд програми: кожній команді ставиться у відповідність унікальне ціле число. Вибір нумеруючих чисел вважаємо випадковим. Команді з номером i ставиться у відповідність константний ідентифікатор $Id(i) = f(i)$, а змінній z - цифрове значення $Id^{-1}(i)E(z)$.Тоді i-тій команді $i : z \leftarrow x + y$ відповідає цифрове значення:

$Id^{-1}(i)Id(z)Id^{-1}(y)E(x)E(y)$

Для перевірки правильності переходів між командами вводиться окрема змінна H, яка описує історію команд. Почакове значення $H = f(i_0)f(i_1)$, i_1 та i_0 - випадкові секретні числа, які відповідають першій команді та стартовому чилі на початку програми.

При переході від команди j до команди i, виконується присвоєння

$H \leftarrow Id^{-1}(j)Id(i)H$. Ця команда присвоєння вставляється одразу після команди j, тоді поточне значення H в момент виконання команди i : $H = f(i_0)Id(i) = f(i_0)f(i)$.
Таким чином модифікований блок, який відповідає вихідній команді $z \leftarrow x + y$, має вигляд:

$$H \leftarrow Id^{-1}(j)Id(i)H$$
$$E(z) \leftarrow Hf^{-1}(i_0)Id^{-1}(i)Id(z)Id^{-1}(x)Id^{-1}(y)E(x)E(y)$$
$$z \leftarrow x + y$$

де j - номер команди перед командою i. Вираз у правій частині оператора присвоєння $E(z)$ дорівнює $Id(z)f(x)f(y) = E(z)$

Значення констант $c_{ji} = Id^{-1}(j)Id(i)$ та $c_i = f^{-1}(i_0)Id^{-1}(i)Id(z)Id^{-1}(x)Id^{-1}(y)$ обчислюються заздалегідь, після виконання розмітки, c_i не залежить від попереднього стану.

Для контролю правильності виконання умовних операторів вводяться додаткові змінні. В них зберігаються значення останніх входів/виходів до умови.

Після виконання всіх вставок контрольних змінних і відповідних команд присвоєння введена розмітка знищується.
В якості результату виконання програми користувач отримує кортежі вигляду <ім'я змінної, значення змінної, значення цифрового підпису змінної>. Ці кортежі клієнт клієнт подаємо на систему, яка перевіряє верифікаційні умови.

Умова верифікації для змінної x, яка перевіряється клієнтом після отримання результатів програми, має вигляд:
$V(E(x)) : E^c(x) \, mod \, n = (g^{acx+cr_x}) \, mod \, n = (g^{ac} mod \, n)^x * g^{cr_x} mod \, n$

1.7 Схема Джентрі

Інший спосіб використання гоморфного шифрування в забезпеченні цілісності обчислень на віддаленому пристрої полягає в безпосередньому

зашифруванні даних перед відправкою їх на сервер гомоморфними схемами шифрування. Однією з можливих таких схем є схема Джентрі(Gentry) [5]. Ця схема є першою преставленою схемою повністю гомоморфного шифрування.

Наведемо основні етапи схеми Джентрі

Генерування ключів

Генерація ключів в схемі Джентрі [5] відбувається наступним чином: випадково обирається довільне число $p \in \mathbb{Z}$ виду $p = 2 \cdot k + 1$. Число p становить секретний ключ системи.

Зашифрування

Розглянемо зашифрування в схемі Джентрі [5] на прикладі обчислень в $\mathbb{Z}_2$. Отже, відкритий текст $m \in \{0,1\}$. Для його зашифрування спочатку генеруємо число $z = 2r + m$, де r - довільне ціле число. Тобто $z \equiv m \ (mod\ 2)$. Шифрування відбувається шляхом співставлення числа $c = pq + z$ повідомленню m, де $q \in \mathbb{Z}$ - довільне. Відповідно,
$E(m) = c = 2r + m + (2k + 1)q = 2(r + kq) + m + q.$

Тоді $c = E(m)$ - шифротекст.

Розшифрування

Розглянемо розшифрування в схемі Джентрі [5]. Нехай дано шифротекст c та числа p,q. Тоді розшифрування за допомогою секретного ключа p відбувається наступним чином:
$c\ (mod\ p) = (z + pq)(mod\ p) = z(mod\ p) = 2(r(mod\ p)) + m(mod\ p)$
Потім обчислюємо результат за модулем 2:
$(c(mod\ p))(mod\ 2) = (2(r(mod\ p)))(mod\ 2) = m(mod\ 2) = m.$

На жаль, наразі схема Джентрі не є дуже практичною. В ній

з'являється випадковий так званий «шумовий» компонент $(r = c \ mod \ p)$[5], розмір якого збільшується з кожжним проведенням гомоморфних операцій на шифротексті. Як тільки розмір шуму перевищить певний поріг, шифротекст не може бути коректно розшифрований. Це обмежує кількість гомоморфних операції, які можна буде виконати. Щоб обійти це обмеження, Джентрі вводить операцію *bootstrapping*, яка дозволяє «оновити» зашифрований текст і зменшити значення шуму. Саме ця процедура і призводить до зростання обчислювальної складності схеми.

І хоча за останні кілька років були проведені численні покращення даної моделі, вона все ще залишається теоретичною моделю, складно застовованною на практиці.

Слід зазначити, що крім схеми Джентрі і її модифікацій йде активна робота над створенням альтернативних симетричних гомоморфних криптосистем на основі поліномів від однієї(або декількох) змінних і матричних поліномів [15]. Перевага цієї моделі в тому, що є можливість розпаралелити обчислення в силу того, що всі операції над шифротекстами зводяться до добутків та додавання матриць. Відповідно, підвищується практична значимість цієї системи.

Висновки до розділу 1

В даному розділі розглянуті опубліковані джерела за тематикою моєї роботи, а саме теоретичні відомості щодо гомоморфного шифрування, існуючі алгоритми часткового та повного гомоморфного шифрування, а саме криптосистеми RSA, Ель-Гамаля, Бенало, повність гомоморфну криптосистему Джентрі, та можливість їх використання для забезпечення цілісності віддалених обчислень. В ході проведення досліджень з'ясовано, що наразі застосування повністю гомоморфним алгоритмів є не дуже практичним, оскільки для реалізації гомоморфого шифрування потрібні дуже великі ключі, відповідно складність зашифрування та інших операцій дуже велика. Проте цей напрям дуже перспективний, оскільки гомоморфне шифрування здатне забезпечувати конфіденційность інформації при її передачі та зберіганні.

2 КРИПТОСИСТЕМА БЕНАЛО

В цьому розділі наведено опис розробленої мною програмної реалізації класичної схеми Бенало на великих числах та набору додаткових функцій для дослідження цієї криптосистеми. Програма була написано мовою C#, оскільки ця мова включає в себе багато готових рішень, наприклад клас *BigInteger*, який дозволяє працювати з великими числами. Програмна реалізація складається з наступних модулів: Benaloh.cs - модуль, в якому реалізована основна логіка програми; Helpers.cs - модуль, в якому містяться допоміжні функції; Program.cs - місце виклику фунцій з інших модулів. Код програми можна знайти в файлах з відповідними назвами за посиланням код програмної реалізації або за url: https://github.com/AnastasiiaMoroziuk/Diploma_thesis.

2.1 Класична криптосистема Бенало

Класична криптосистема Бенало [1, 4] була програмно реалізована в модулі Benaloh.cs та являє собою клас Benaloh, який складається з наступних полів:

– публічні поля n, y та r, які становлять публічний ключ системи;

– приватні поля p та q, які становлять секретний ключ системи;

– приватне поле $safetyParam$ - так званий "параметр безпеки"системи, який дорівнює байтовій довжині p або q;

– приватне поле $primes$ - масив простих чисел, які використовуються для дослідження недоліків системи Бенало;

Також в класі Benaloh реалізовано наступні методи:

– метод *SetSafetyParam(int byteLength)* приймає на вхід ціле число та встановлює його параметром безпеки системи $safetyParam$;

– метод *SetPrivateKey()* генерує секретний ключ системи (параметри p та q), відповідно до процедури генерації ключів в схемі Бенало, довжиною

$safetyParam$ байт;

– метод *SetPrivateKey()* генерує секретний ключ системи (параметри n, y та r), відповідно до процедури генерації ключів в схемі Бенало;

– метод *SetEntireSystem(BigInteger p, BigInteger q, BigInteger r, BigInteger y, BigInteger n)* дозволяє одразу задати всі параметри системи p, q, r, y, n;

– метод *CheckKeys()* перевіряє чи задовольняють всі параметри утвореної системи умовам криптосистеми Бенало;

– метод *Encrypt(BigInteger message, BigInteger u)* зашифровує повідомлення *message*;

– метод *Decrypt(BigInteger ciphertext)* розшифровує шифротект повідомлення *ciphertext*.

2.2 Демонстрація роботи програми

В цьому підрозділі представлені результати роботи реалізованих функцій.

На рисунку 2.1 демонструється робота функцій *SetSafetyParam(int byteLength)*, *SetPrivateKey()*, та *SetPrivateKey()*. В результаті була утворена система Бенало, де байтова довжина параметрів p та q дорівнює 32 байти.

```
Параметр безпеки:  32 байти
Згенерована система:

   Публічний ключ:
   y = 588979555340741637625909
   r = 17974983264766340981991022108540782612734083245718531140907310249779908081 3
   n = 16422800093596408846051970821810671711191518915092616431245299362980266096161697640235056757803309725465579302768629837239423800838182728549547560616671 9

   Секретний ключ:
   p = 57519946447252291142371270747330504360749066386299299650903392799295700860 17
   q = 28551487106575567033171450836930144187183057696396236386254658236017575603807
```

Рисунок 2.1 – Ключі утворенної системи Бенало.

На рисунку 2.2 демонструється робота функції *CheckKeys()*, яка перевіряє чи задовольняє утворена система всім умовам схеми Бенало, наведеним в розділі 1. Як можна побачити з рисунку, всі необхідні умови виконуються.

```
Перевірка чи задовільняє система умовам:
    • r ділить p-1                    :   True;
    • НСД(r, p-1/r) = 1               :   True;
    • НСД(r, q-1) = 1                 :   True;
    • y^(p-1)(q-1)/r mod n != 1       :   True;
    • НСД(y^(p-1)(q-1)/r, n) = 1      :   True;
```

Рисунок 2.2 – Перевірка утвореної криптосистеми.

На рисунку 2.3 демонструється робота фунції зашифрування *Encrypt(BigInteger message, BigInteger u)* повідомлення *message*, параметр шифрування u було згенеровано допоміжною функцією *GenerateFromMultipicativeGroup(BigInteger n)* з модуля Helpers.cs, яка випадкового генерує елемент із $\mathbb{Z}_n^*$ та повертає його.

```
Повідомлення message = 6103328


Зашифрування повідомлення message:
   Шифротекст ciphetext = 14305633743725385552701750789140298497725916509903320305202651438288232627419095880204192986132121077904447590503974723170816553351016621341544625367086 1
```

Рисунок 2.3 – Зашифрування повідомлення.

На рисунку 2.4 демонструється робота функції розшифрування *Decrypt(BigInteger ciphertext)*. Як видно з рисунку, повідомлення *message* було розшифровано коректно.

На рисунку 2.5 перевіряється гомоморфність системи. Для повідомлень m_1 та m_2 було обчислено за допомогою функції

```
Розшифрування шифротекста ciphetext:
   Повідомлення message = 6103328
```

Рисунок 2.4 – Розшифрування отриманого шифротексту.

Encrypt(BigInteger message, BigInteger u) відповідні шифротексти $E(m_1)$ та $E(m_2)$. Також було обчислено шифротекст $E(m_1 + m_2)$ для суми повідомлень m_1 та m_2. Як видно з результатів, $E(m_1 + m_2) = E(m_1) * E(m_2)$, отже гомоморфність додавання виконується.

```
Гомоморфність в утворенній криптосистемі :

m₁ = 1092
m₂ = 1379

E(m₁ + m₂) = 714460731167632402424578692299642278072727148592254830149571639666373937310534033041442281088557809323284426689674915109010819971680194207676884138676271053403304144228108855780932328442668967491510901081997168019420767688413867627
E(m₁) * E(m₂) = 714460731167632402424578692299642278072727148592254830149571639666373937310534033041442281088557809323284426689674915109010819971680194207676884138676271053403304144228108855780932328442668967491510901081997168019420767688413867627
E(m₁ + m₂) = E(m₁) * E(m₂)
```

Рисунок 2.5 – Гомоморфність за додаванням в утворенній криптосистемі.

2.3 Дослідження недоліків криптосистеми Бенало

В цьому підрозділі описано проведенний експеремент, ціль якого полягала в знаходженні неоднозначностей розшифрування в криптосистемі Бенало для фіксованої множини параметрів. Для його проведення Benaloh в класі в модулі Benaloh.cs було реалізовано наступні методи:

– метод *GenAllSystemParams()* генерує кортежі всіх можливих параметрів системи виду $< p, q, r, y >$. Параметри p та q обираються з приватного поля класу - масиву *primes*;

– метод *GenAllCiphertexts(List <Tuple < BigInteger, BigInteger,*

List <Tuple <BigInteger, List <BigInteger> > > > > system) приймає на вхід список параметрів системи *system*(кортежі виду $< p, q, r, y >$) та генерує кортежі, які складаються з параметрів системи та всіх можливих повідомлень, які можна зашифрувати на цих параметрах та відповідні шифротексти;

– метод *GroupByCiphertext(List<Tuple<string, BigInteger, BigInteger» tuples)* приймає на вхід список кортержів виду $< p, q, r, y, m, E(m) >$ та групує їх за параметрами системи тп шифротексом, таким чином утворюючи кортежі вигляду $< p, q, r, y, E(m), M >$;

– метод *GroupBySystemParams(Dictionary<string, List<BigInteger» groups)* приймає на вхід список кортежів виду $< p, q, r, y, E(m), M >$ та групує їх за параметрами системи, таким ином утворюючи кортежі виду $< p, q, r, y, |C| >$.

Хід експерименту

Експремент проводився наступним чином: з множини простих чисел $P = \{5, 7, 11, 13, 17, 19, 23, 23, 29, 31\}$ було побудовано всі можливі пари (p,q), де $p,q \in P$ та $p \neq q$, причому пари вигляду (13,17) та (17,13) вважаємо однаковими. На кожну пару (p,q) було знегеровано множину чисел r згідно з правилами побудови параметрів в криптосистемі Бенало. Аналогічно на кожен кортеж $< p, q, r >$ було обрано множину параметрів $y \in \mathbb{Z}_n^*$.

Далі на кожен кортеж $< p, q, r, y >$ зашифровуємо всі можливі повідомлення $m \in Z_r$ генеруючи всі можливі параметри зашифрування $u \in \mathbb{Z}_n^*$. Таким чином було згенеровано всі можливі кортежі виду $< p, q, r, y, m, u, E(m) >$. Для обранної множини секретних параметрів P, таких кортежів вишло - 15 515 648.

Оскільки можлива ситуація, коли на однакових параметрах системи $< p, q, r, y >$ однакові відкриті тексти m переходять в однакові шифротести $E(m)$ при різних u, було вирішенно не враховувати надалі параметр u, оскільки він використовується лише для зашифрування і ніяк не впливає на розшифрування повідомлень, хоча подальша фільтрація

отриманих даних значно полегшується. Таким чином кількість всіх кортежей вигляду $< p, q, r, y, m, E(m) >$ становить - 1 950 208.

Далі їх було відфільтровано так, щоб для однакового набору $< p, q, r, y >$, шифротексти $E(m_1)$ та $E(m_2)$ не були рівні, якщо $m_1 \neq m_2$, де $m_1, m_2 \in Z_r$. Утворена множина кортежів виду $< p, q, r, y, E(m), M >$ має потужність 30 432, де M - множина відкритих тестів, які при зашифруванні дорівнюють $E(m)$. Кожен шифротекст $E(m)$ має приблизно по 2-3 прообрази з множини відкритих текстів, при чому всі вони правильні.

В якості наступного кроку експеременту, було згруповано отриманні результати за параметрами системи, тобто тепер ми маємо кортежі вигляду $< p, q, r, y, |C| >$, де $|C|$ - потужність множини всіх шифротекстів, які мають декілька прообразів з множини відкритих текстів при заданих параметрах системи. Загальна кількість таких кортежів склала 200.

На рисунку 2.6 представлена кількість всіх проміжних кортежей, які було отримано в результаті експерименту.

```
Всього кортежів <p, q, r, y, m, u, E(m)>:  15515648

Всього кортежів <p, q, r, y, m, E(m)>:  1950208

Кількість кортежів <p,q,r,y,E(m), M> :  30432,
  де M - множина відкритих текстів, які при зашифруванні дають однаковий шифротекст

Кількість кортежів <p,q,r,y, |C|> :  200
```

Рисунок 2.6 – Результати дослідження: кількість утворених кортежів.

На рисунку 2.7 представлена частина кортежей, які були отримані в результаті експерименту. Всі знайдені кортежі можна переглянути за посиланням benalohOutput.txt або за url: https://github.com/AnastasiiaMoroziuk/Diploma_thesis/blob/main/benalohOutput.txt. Як видно з результатів, кількість шифротекстів, які можуть бути неоднозначно розшифровані співпадає для систем з однаковими параметрами $< p, q, r >$, причому кожен шифротекст має приблизно 2-3 прообраза з множини відкритих текстів.

```
<19, 23, 9, 7   > CipherTexts :  132
<19, 23, 9, 8   > CipherTexts :  132
<19, 23, 9, 11  > CipherTexts :  132
<19, 23, 9, 12  > CipherTexts :  132
<19, 23, 9, 26  > CipherTexts :  132
<19, 23, 9, 27  > CipherTexts :  132
<19, 23, 9, 30  > CipherTexts :  132
<19, 23, 9, 31  > CipherTexts :  132
<19, 23, 9, 45  > CipherTexts :  132
<19, 23, 9, 49  > CipherTexts :  132
<19, 23, 9, 50  > CipherTexts :  132
<19, 23, 9, 64  > CipherTexts :  132
<19, 23, 9, 65  > CipherTexts :  132
<19, 23, 9, 68  > CipherTexts :  132
<19, 23, 9, 83  > CipherTexts :  132
<19, 23, 9, 84  > CipherTexts :  132
<19, 23, 9, 87  > CipherTexts :  132
<19, 23, 9, 88  > CipherTexts :  132
<19, 23, 9, 102 > CipherTexts :  132
<19, 23, 9, 103 > CipherTexts :  132
<19, 23, 9, 106 > CipherTexts :  132
<19, 23, 9, 107 > CipherTexts :  132
<19, 29, 9, 27  > CipherTexts :  168
<19, 29, 9, 30  > CipherTexts :  168
<19, 29, 9, 31  > CipherTexts :  168
<19, 29, 9, 45  > CipherTexts :  168
<19, 29, 9, 46  > CipherTexts :  168
<19, 29, 9, 49  > CipherTexts :  168
<19, 29, 9, 50  > CipherTexts :  168
<19, 29, 9, 64  > CipherTexts :  168
<19, 29, 9, 65  > CipherTexts :  168
<19, 29, 9, 68  > CipherTexts :  168
<19, 29, 9, 69  > CipherTexts :  168
<19, 29, 9, 83  > CipherTexts :  168
<19, 29, 9, 84  > CipherTexts :  168
<19, 29, 9, 88  > CipherTexts :  168
<19, 29, 9, 102 > CipherTexts :  168
<19, 29, 9, 103 > CipherTexts :  168
<19, 29, 9, 106 > CipherTexts :  168
<19, 29, 9, 107 > CipherTexts :  168
<19, 29, 9, 121 > CipherTexts :  168
<19, 29, 9, 122 > CipherTexts :  168
<19, 29, 9, 125 > CipherTexts :  168
<19, 29, 9, 126 > CipherTexts :  168
```

Рисунок 2.7 – Фрагмент усіх результуючих кортежів експерименту.

В оригінальній статті Бенало[1] зазначено, що найефективніше буде працювати розшифрування для систем з $r = 3^k$, де k - ціле невід'ємне число. Проте в статті [4] сказано, що ймовірність отримати такі y, щоб в системі була неоднозначність розшифрування дорівнює $\rho = 1 - \frac{\phi(r)}{r-1} = 1 - \frac{6}{8} = 1 - \frac{3}{4} = 0{,}25$. Тобто чверть всіх можливих кортежів з параметрами $< 19, 23, 9 >$ повина давати неонозначність розшифрування.

На рисунку 2.8 показано кількість всіх можливих y, які можуть бути четвертим параметром системи $< 19, 23, 9 >$ та фактична кількість всіх y, які були отримані в експерименті на яких була неоднозначність розшифрування. Отже фактична ймовірність натрапити на «поганий» y дорівнює $\frac{89}{392} = 0{,}227$, що є дуже близьким до теоретичної оцінки цієї ймовірності.

```
Загальна кількість можливих y : 392
Кількість отриманих "поганих" y : 89
```

Рисунок 2.8 – Відношення кількості можливих y до поганих y.

2.4 Швидкість розшифрування в криптосистемі Бенало

В цьому підрозділі йдеться про проведенний обчислювальній експеримент, ціль якого полягала в знаходженні реальної швидкості алгоритму розшифрування в схемі Бенало, так як алгоритм розшифрування по суті являє собою послідовний перебір таких $m \in \mathbb{Z}_r$, що $^{(p-1)(q-1)/r \neq 1 \, mod \, n}$, де c - шифротекст. Характеристики ноутбука, на якому було проведено тестування такі: процесор Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz, оперативна пам'ять 8 ГБ та операційна система Windows 10 Pro x 64 версії 20H2. Среда розробки - Visual Studio Community 2019 версії 16.9.6.

Перший крок експерименту полягав в генеруванні випадкової системи з параметром безпеки 48 бітів(6 байтів) за допомогою методів *SetPrivateKey()* та *SetPrivateKey()* з модулю Benaloh.cs та по 5 випадкових повідомлень довжиною 8, 16, 24, 32,40 бітів (1, 2, 3, 4, 5 байтів відповідно) за допомогою методу *Generate()* з модуля Helpers.cs.

На рисунку 2.9 зображено публічні та секретні параметри згенерованої системи, на якій буде проводитись подальше дослідженя.

```
Параметр безпеки:  48 біт
Згенерована система:

   Публічний ключ:
   y = 3382385
   r = 42988815050183
   n = 406053342100356386291357808 7

   Секретний ключ:
   p = 85977630100367
   q = 47227789557161
```

Рисунок 2.9 – Параметри згенерованої системи.

Наступним кроком було зашифрування кожного повідомлення за допомогою функції *Encrypt(BigInteger message, BigInteger u)* та їх розшифрування за допомогою функції *Decrypt(BigInteger ciphertext)*.

Параметр u генерувався за допомогою функції генерування елемента мультиплікативної групи $\mathbb{Z}_n^*$ *GenerateFromMultipicativeGroup(BigInteger n)* з модуля Helpers.cs. Час розшифрування фіксувався програмою для кожного повідомлення за допомогою вбудованної структури *DateTime*.

На рисунку 2.10 наведено приклад результатів генерації повідомлень, функції зашифрування та розшифрування для повідомлень розміром 24 біти.

```
Розмір повідомлення: 24 біт

Повідомлення m = 6948073,
Шифротекст = 67456510396572395707137641 1
Результат розшифрування: m = 6948073
Час виконання розшифрування : 00:00:39.8244170

Повідомлення m = 6080001,
Шифротекст = 18768498857593468201558145 86
Результат розшифрування: m = 6080001
Час виконання розшифрування : 00:00:32.9347168

Повідомлення m = 1806395,
Шифротекст = 17957579048996548607153124 20
Результат розшифрування: m = 1806395
Час виконання розшифрування : 00:00:08.9373410

Повідомлення m = 5465131,
Шифротекст = 26904054471075141423625777 14
Результат розшифрування: m = 5465131
Час виконання розшифрування : 00:00:29.2316829

Повідомлення m = 1497498,
Шифротекст = 12109670272409586729794738 75
Результат розшифрування: m = 1497498
Час виконання розшифрування : 00:00:07.3434947
```

Рисунок 2.10 – Результатів для повідомлень розміром 24 біт.

Потім для всіх повідомлень одного розміру було пораховано середній час розшифрування, результати виглядають наступним чином :

– розмір повідомлення - 8 бітів, середній час розшифрування - 0,00002952966 хвилин;

– розмір повідомлення - 16 бітів, середній час розшифрування - 0,0014010518 хвилин;

– розмір повідомлення - 24 бітів, середній час розшифрування - 0,39423884133334 хвилин;

– розмір повідомлення - 32 бітів, середній час розшифрування - 80,36004251 хвилин;

– розмір повідомлення - 40 бітів, середній час розшифрування - 408,6705 хвилин;

Далі було побудовано графік залежності часу(хвилини) від розміру повідомлення (біти), предствлений на рисунку 2.11, з метою побачити чи буде ця залежність схожа на графік квадратного кореня, так як найкраща можлива часова скаладність розшифрування в схемі Бенало дорівнює $O(\sqrt{r})$.

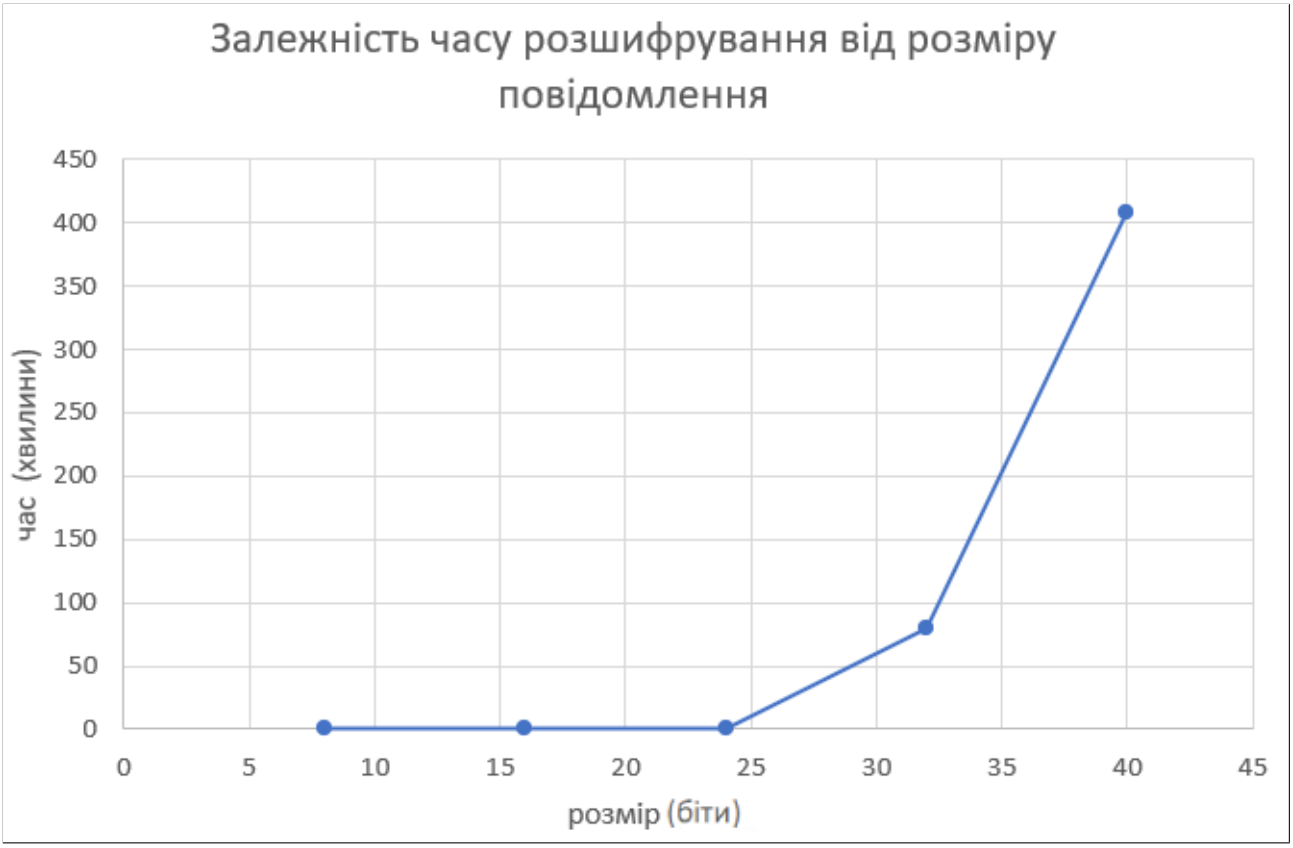

Рисунок 2.11 – графік залежності часу від розміру повідомлення.

Як видно з графіку, залежність часу від розміру дуже відрузняється від графіку квадратного кореня, відповідно часова складність цього алгоритму доволі далека від найкращої можливої.

В якості покращення швидкості роботи можна запропонувати розбиття повідомлень на блоки розміром до 24 біт. В цьому випадку розшифрування кожного блоку займатиме до 1 хвидини. Відповідно можно обмежити розмір парметра r, оскільки всі повідомлення $m \in \mathbb{Z}_r$ та генерувати r виду 3^k, де k - ціле невід'ємне число. Проте в цьоиу випадку, як було експерементально підтверджено, росте ймовірність неоднозначності розшифрування.

Висновки до розділу 2

Таким чином, в цьому розділі описано програмну реалізацію криптосистеми Бенало на великих числах, наведено приклади роботи даної програми та проведено дослідження недоліків системи Бенало.

В результаті проведенного експеременту з'ясовано, що дійсно система Бенало в класичному вигляді має серйозну вразливість, а саме при певному наборі параметрів існують шифротексти, які мають декілька прообразів з множини відкритих текстів і всі вони правильні, тобто має місце неоднозначніть розшифрування. Отримані результати практично підтверджують теоретичні оцінки в розглянутих роботах [4].

Наявність цієї вразливості свідчить про те, що криптосистема Бенало потребує ще додаткових досліджень. Також є сенс використовувати модифікації системи Бенало, в яких немає неоднозначності розшифрування, але їх так само необхідно більше дослідити в силу їх новизни.

Окрім того було проведено певні оцінки часової складості розшифрування. На відміну від інших схем, як наприклад RSA, криптосистема Бенало значно програє у швидкості розшифрування, у випадку використання реалізованого алгоритму розшифрування. Враховуючи цей факт використання цієї схеми для повідомлень більших за 24 біти не є доцільним, або ж необхідно реалізовати ефективніший алгоритм. Враховуючи отримані результати в якості подальших досліджень було б цікаво знайти компроміс між виглядом параметру r та ймовірністю неоднозначності розшифрування.

3 СХЕМА КОНТРОЛЮ ЦІЛІСНОСТІ ДЕЛЕГОВАНИХ ОБЧИСЛЕНЬ

В цьому розділі наведено розроблену мною програмну реалізацію схеми контролю цілісності делегованих обчислень, описану в розділі 1, на великих числах мовою C#. Також в цьому розділі представлені хід та результати дослідження можливості підробки результатів превірки контролю цілісності делегованих обчислень. Програмна реалізація складається з наступних модулів: Schema.cs - модуль, в якому реалізована основна логіка програми; Helpers.cs - модуль, в якому містяться допоміжні функції; Program.cs - місце виклику фунцій з інших модулів. Код програми можна знайти в файлах з відповідними назвами за посиланням код програмної реалізації або за url: https://github.com/AnastasiiaMoroziuk/Diploma_thesis.

3.1 Опис програмної реалізації

Схема котролю цілісності[3, 4, 16] була програмно реалізована в модулі Schema.cs та являє собою клас Schema, який складається з наступних полів:

– публічне поле $publicKey$, яке містить публічний ключ n;

– приватне поле $privateKey$, яке містить секретні параметри системи r, c, g_1, g_2;

– приватне поле $safetyParam$ - так званий "параметр безпеки"системи, який дорівнює байтовій довжині r;

– приватне поле phi, яке дорівнює $(p - 1) * (q - 1)$;

– приватне поле $usedStates$ - масив використаних станів згенерованої системи;

– приватні поля p та q - які відповідають параметрам p та q та беруть участь в формуванні секретного та публічного ключів.

Також в класі Schema реалізовано наступні методи:

– метод *SetSafetyParam(int byteLength)* приймає на вхід ціле число та встановлює його параметром безпеки системи $safetyParam$;

– метод *GenerateKey()* генерує набор параметрів систем, в залежності від параметра безпеки (байтової довжини r);

– метод *CheckSystem(bool logResults = false)* перевіряє чи задовольняють всі параметри згенерованої системи заданим умовам та може вивести результат перевірки кожної умови.

– метод *SetEntireSystem(BigInteger p, BigInteger q, BigInteger r, BigInteger g1, BigInteger g2)* дозволяє одразу ввести всі параметри системи $p, q, r, g_1, , g_2$ з клавіатури.

– метод *GenerateStates()* генерує початкові секретні стани.

– метод *GenerateImitInsertion(BigInteger b, BigInteger u, BigInteger x, BigInteger a_x)* обчислює імітовставку для повідомлення x з початковим секретним станом a_x.

– метод *Verify(BigInteger x, BigInteger e_x, BigInteger a_x)* перевіряє відповідність імітовставки e_x повідомленню x з секретним станом a_x;

Також було реалізовано допоміжні методи *F(BigInteger b, BigInteger x)* та *Id(BigInteger a_x, BigInteger u)*, які використовуються в методі *GenerateImitInsertion(BigInteger b, BigInteger u, BigInteger x, BigInteger a_x)* та по суті є варіаціями функції зашифрування Бенало. Окрім того, програмна реалізація містить додаткові методи для виводу секретних параметрів в консоль.

3.2 Демонстрація роботи програми

В цьому підрозділі представлени результати роботи реалізованих методів.

На рисунку 3.1 продемострована робота функцій *SetSafetyParam(int byteLength)*, *GenerateKey()*. Спочатку в системі генерується параметр r, далі - пармери p та q, потім обчислюються n та c, а останнім кроком - g_1 та

g_2, насправді ж замість осанніх двох параметрів можна використовувати єдиний g, але так буде більша стійкість. В результаті була утворена схема перевірки цілісності повідомлень з параметром безпеки 32 байти.

```
Параметр безпеки:  32 байти
Згенерована система:

Публічний ключ:
    n = 42275457575372587397127025395596250207039861561076419500436174419178692531037066287442645070410349752257739426527664267944563227407593441738785137781326192 3

Секретний ключ:
    r = 668198330849672253219297070393940146269800306647151710592794961263459914358 1,
    c = 6326782876218153489918418375136678627857352293854168766434848110686845230129380 0,
    g₁ = 263921028278447724898407553604184764419759508 2,
    g₂ = 1401334487799778938491969399196258695408677234972286020857314280396339015363564625418591217405
```

Рисунок 3.1 – Ключі утвореної системи перевірки цілісності делегованих обчислень.

На рисунку 3.2 демонструється робота функції *CheckSystem(bool logResults = false)*, яка перевіряє чи задовольняє утворена система всім умовам схеми контролю цілісності делегованих обчислень, наведеним в розділі 1. Як видно з рисунку, всі умови виконуються.

```
Перевірка :
    • r ділить p-1                  :   True
    • НСД(r, p-1/r)=1               :   True
    • НСД(r, q-1)=1                 :   True
    • p != q                        :   True
    • g1 != g2                      :   True
    • НСД(g1,n)=1                   :   True
    • g1^c mod n !=1                :   True
    • НСД(g2,n)=1                   :   True
    • g2^c mod n !=1                :   True
```

Рисунок 3.2 – Перевірка утвореної системи

На рисунку 3.3 демонструється робота функцій *GenerateStates()* та *GenerateImitInsertion(BigInteger b, BigInteger u, BigInteger x, BigInteger a_x)*. Було згенеровано повідомлення x_1 та x_2 за допомогою функції-генератора великих чисел заданої довжини з модуля Helpers.cs. Оскільки в цій схемі не накладається обмежень на повідомлення, було обрано довжину 32 байти, так як це значення параметру безпеки. Параметри u та b було згенеровано за допомогою функції-генератора елементів з мультиплікативної групи згенеровано з модуля Helpers.cs. Для повідомлень було згенеровано секретні стани a_{x_1} і a_{x_2} та обчислено імітовставки E_{x_1} і E_{x_2} відповідно.

```
 Повідомлення x₁ = 472939714871879990888539872060640766382185438772117618259515181935731264709 58
 Імітовставка Eₓ₁: 345489623965412834801378352718943627010093329517251624490780530284076821460138902495440235641601385105746570930773234185612303602413583259999947424115624867

 Повідомлення x₂ = 750017458423463475227476827083621876538269977211560495950051819099367580937 7
 Імітовставка Eₓ₂: 227865837661351752883245028602483484603463468779685980144516066929902124260693386365780083660708935564028760357083807377686307193017476906282932636468617545
```

Рисунок 3.3 – Повідомлення та обчислені імітовставки.

На рисунку 3.4 представлена перевірка цілісності повідомлень x_1 та x_2 за допомогою функції *Verify(BigInteger x, BigInteger e_x, BigInteger a_x)* , використовуючи відповідні їм імітовставки та секретні стани. Як видно з рисунку, перевірка працює коректно.

```
Результат перевірки цілісності повідомлення x₁: True
Результат перевірки цілісності повідомлення x₂: True
```

Рисунок 3.4 – Перевірка цілісності повідомлень.

На рисунку 3.5 представлен результат перевірки цілісності

повідомлення x_1 при підміні його на повідомлення x_2. Тобто представлен результат функції *Verify(BigInteger x, BigInteger e_x, BigInteger a_x)*, передаючи в якості параметрів повідомлення x_2 та, відповідні до повідомлення x_1, секретний стан a_{x_1} та імітовставку E_{x_1}. Як і очікувалось, результат такої перевірки дорівнює *false*.

```
Результат перевірки цілісності повідомлення x₁ при заміні
на повідомлення x₂: False
```

Рисунок 3.5 – Перевірка цілісності повідомлень при підміні.

На рисунку 3.6 представена сума повідомлень x_1 та x_2. Для них було обчислено відповідну імітовставку $E_{x_1+x_2}$. Для отриманих даних було провередено проведено перевірку цілісності за допомогою функції *Verify(BigInteger x, BigInteger e_x, BigInteger a_x)* , передаючи в якості параметрів суму повідомлень x_1 та x_2, імітовставку $E_{x_1+x_2}$ та суму секретних станів a_{x_1} та a_{x_2}. Як і очікувалось, перевірка підтвердила цілісність.

```
 Сума повідомлень x₁ + x₂: 54794146071422633841128755476900295403601243649327366785452036384566802280335
 Імітовставка суми Eₓ₁₊ₓ₂:  37664209127867547919003773914369835704145062467554369966034694027746276097543326954694481177006642445917118828092895101110035297959749351158455869888691913 5

 Результат перевірки цілісності суми повідомлень x₁ та x₂:  True
```

Рисунок 3.6 – Перевірка цілісності суми повідомлень.

3.3 Дослідження можливості підробки результатів перевірки цілісності делегованих обчислень

На основі результатів отриманих в дослідженні неоднозначностей розшифрування в класичній схемі Бенало, було проведено аналогічний експеремент для схеми перевірки цілісності делегованих обчислень. Для проведення експеременту було реалізовано декілька додаткових методів в модулі Schema.cs :

– метод *GenerateAllSystemParams()* генерує всі можливі параметри системи при фіксованих p, q, r.

– метод *FindAllTuples(List<Tuple<BigInteger, BigInteger, BigInteger, BigInteger, BigInteger, BigInteger» systemParams)* приймає на вхід параметри системи та генерує всі можливі кортежі з повідомленнями і імітовставками, та групує їх за значеннями імітовставок.

Оскільки в данній схемі більше секретних параметрів ніж в класичній схемі Бенало, було прийнято проводити перебір при фіксованих $< p, q, r >$, а саме на тих, які за результатами експеременту на схемі Бенало давали неоднозначність розшифрування. Таким чином для експеременту було обрано кортеж $< 19, 23, 9 >$.

Наступним кроком була побудова всіх можливих кортежів $< p, q, r, c, g_1, g_2 >$ при фіксованих p, q та r. Їх кількість становить - 123 552.

Оскільки ціль цього експременту полягає в тому, щоб показати, що існують такі пари повідомлень (x_1, x_2), що $E_{x_1} \neq E_{x_2}$, було вирішено продовжувати експеремент не для всіх отриманих 123 552 кортежів, а лише для одного. Цей кортеж - $< 19, 23, 9, 44, 2, 3 >$.

Далі задача полягала побудові всіх можливих кортежів $< p, q, r, c, g_1, g_2, x, E_x, a_x, b, u >$. З них $g_1, g_2 \in \mathbb{Z}_n^*$ такі, що $g_i^c \, mod \, n \neq 1$, $i \in \{1, 2\}$ та $g_1 \neq g_2$, $u, b \in \mathbb{Z}_n^*$, $a_x \in \mathbb{Z}_r \backslash \{0,1\}$. Так як в даній схемі не накладається обмежень на повідомлення ($x \in \mathbb{Z}$), було вирішено робити перебір для повідомлень $x \in [1, 50]$, щоб полегшити перебір. Таким чином, всього було отримано 53 787 888 таких кортежів.

Також для полегшення перебору було вирішено надалі не враховувати u та b, оскільки можлива ситуація, коли імітовставка E_{x_1} отримується за допомогою однакових повідомлень x_1 однакових секретних станів a_1 та різних (u_1, b_1) та (u_2, b_2).

Таким чином, всього було отримано 15 092 кортежів виду $< p, q, r, c, g_1, g_2, x, E_x, a_x >$.

Згрупувавши отримані результати за значенням імітовставки E_x було отримано 396 груп кортежів $< p, q, r, c, , g_1, g_2, x, E_x, a_x >$, при чому для деяких з них значення секретних станів також співпадають, тобто можлива ситуція, коли повідомлення x_1 та x_2 такі, що $x_1 \neq x2$ та $E_{x_1} = E_{x_2} = E_x$ і $a_{x_1} = a_{x_2} = a_x$.

На рисунку 3.7 представлені всі проміжні результати стосовно кількості кортежей, які були створені в ході експерименту.

```
Кількість всіх можливих кортежів виду < p, q, r, c, g₁, g₂> при фіксованих <p, q, r>
 : 123552

Кількість кортежів виду < p, q, r, c, g₁, g₂,  x, Eₓ, aₓ, b, u >  : 53787888

Кількість кортежів виду < p, q, r, c, g₁, g₂, x, Eₓ, aₓ >  : 15092

Кількість імітовставок Eₓ, які можуть бути отриманні за допомогою різних повідомлень
x,
для параметрів < p, q, r, c, g₁, g₂ >  : 396
```

Рисунок 3.7 – Результати дослідження: кількість утворених кортежів.

На рисунку 3.8 та 3.9 представлен результат експерименту. А саме групи кортежів, для яких імітовставка $E_x = 132$, при фіксованих параметрах p, q, r, c, g_1, g_2 , повідомлення $x \in [1, 50]$ та не рівні в межах однієї з утворених груп кортежей, секретний стан $a_x \in \mathbb{Z}_r \backslash \{0, 1\}$. Середня кількість кортежей в одній групі дорівнює 38. Всі повідомленн всередені однієї групи не повторюються. Причому повідомлення, для яких імітоставки однакові, існують для кожного секретного стану a_x. Повний перелік всіх отриманих кортежей можна переглянути за посиланням integrityOutput.txt або за url: https://github.com/AnastasiiaMoroziuk/Diploma_thesis/blob/main/integrityOutput.txt

Імітовставка : 132

p	q	r	c	g_1	g_2	x	E_x	a_x
19	23	9	44	2	3	1	132	2
19	23	9	44	2	3	10	132	2
19	23	9	44	2	3	19	132	2
19	23	9	44	2	3	28	132	2
19	23	9	44	2	3	37	132	2
19	23	9	44	2	3	46	132	2
19	23	9	44	2	3	6	132	3
19	23	9	44	2	3	15	132	3
19	23	9	44	2	3	24	132	3
19	23	9	44	2	3	33	132	3
19	23	9	44	2	3	42	132	3
19	23	9	44	2	3	2	132	4
19	23	9	44	2	3	11	132	4
19	23	9	44	2	3	20	132	4
19	23	9	44	2	3	29	132	4
19	23	9	44	2	3	38	132	4
19	23	9	44	2	3	47	132	4
19	23	9	44	2	3	7	132	5

Рисунок 3.8 – Результати експерименту.

19	23	9	44	2	3	7	132	5
19	23	9	44	2	3	16	132	5
19	23	9	44	2	3	25	132	5
19	23	9	44	2	3	34	132	5
19	23	9	44	2	3	43	132	5
19	23	9	44	2	3	3	132	6
19	23	9	44	2	3	12	132	6
19	23	9	44	2	3	21	132	6
19	23	9	44	2	3	30	132	6
19	23	9	44	2	3	39	132	6
19	23	9	44	2	3	48	132	6
19	23	9	44	2	3	8	132	7
19	23	9	44	2	3	17	132	7
19	23	9	44	2	3	26	132	7
19	23	9	44	2	3	35	132	7
19	23	9	44	2	3	44	132	7
19	23	9	44	2	3	4	132	8
19	23	9	44	2	3	13	132	8
19	23	9	44	2	3	22	132	8
19	23	9	44	2	3	31	132	8
19	23	9	44	2	3	40	132	8
19	23	9	44	2	3	49	132	8

Рисунок 3.9 – Результати експерименту.

Як видно з результатів експерименту, для всіх повідомлень з однаковим секретним станом a_x проглядається закономірнсть, а саме кожне наступне повідомлення більше попереднього на r. Саме тому важливо щоб не було різних повідомлень з однаковими секретними станами. Якщо ж допускати таку можливість, то для двох повідомлень з однаковим секретним станом (навіть не знаючи цей стан), різниця дорівнюватиме $k * r$, де $k \in \mathbb{Z}\backslash\{0\}$, розкладаючи це число на множники

можна відновити r. Знаючи r та алгоритм генерації ключів схеми (а за правилом Кергоффса ми вважаємо що всі алгоритми шифрування відомі) можна буде однозначно відновити множину можливих секретних станів та можна буде спробувати знайти всі можливі комбінації p, q, c, g_1, g_2.

Якщо ж стане відомим якийсь секретний стан повідомлення a_x (самі повідомлення x та імітовставки E_x ми вважаємо відомими), можна буде знайти легко знайти такі повідомлення x', для яких обчислені імітовставки співпадуть при однакових секретних станах і можна буде легко підробити цілісність результатів обчислень.

Проте, беручи до уваги, що даний пошук пар з однаковими імітовставками проводився знаючи всі секретні параметри системи на невеликих числах, реальна ймовірність знайти такі пари буде дуже мала.

Висновки до розділу 3

Отже, в цьому розділі описано реалізацію схеми контролю цілісності делегованих обчислень на великих числах та наведено приклади роботи даної програми.

Окрім того, врахуваючи результати дослідження криптосистеми Бенало, проведено дослідження схеми контролю цілісності делегованих обчислень на предмет існування таких пар <повідомлення, початковий секретний стан повідомлення>, що їх імітовставки співпадають. В результаті було встановлено, що дійсно такі пари існують, причому для кожного секретного стану системи на обраних параметрах. Відповідно, підробка результатів контролю цілісності можлива.

Враховуючи,що експеримент проводився знаючи секретні параметри системи, підробити таку перевірку на великих числах майже неможливо. Але мають місце атаки при наявності деяких секретних параметрів.

Також варто зазначити один із плюсів реалізованої системи - швидкість роботи, оскільки в данній схемі використовується лише функція зашифрування Бенало, яка досить швидка, та для перевірки цілісності не потрібно використовувати функцію розшифрування Бенало.

ВИСНОВКИ

У ході даної роботи був проведений аналіз опублікованих джерел за тематикою гомоморфного шифрування та його використання для контролю цілісності делегованих обчислень. Були розглянуті такі криптосистеми як RSA, Ель-Гамаля, Бенало та схема перевірки правильності делегованих обчислень. Огляд цих джерел показав, що наразі застосування лише частково гомоморфних систем є практичним. Однією з таких систем є відносно нова криптосистема Бенало, яка ще мало досліджена стосовно виявлення недоліків системи та оцінок швидкості розшифрування. У роботі було отримано такі результати:

1) Практично реалізовано криптосистему Бенало на великих числах.

2) Проведено пошук вразливостей в побудованій криптосистемі, за результатами якого були встановлені набори параметрів на який відбувається неоднозначність розшифрування. Було експрементально підтверджено ймовірність вибору такого параметра y, що в системі буде неоднозначність розшифрування.

3) Підраховано оцінки швидкості розшифрування на запропонованому Бенало алгоритмі розшифрування. Виявлено, що оптимальна довжина повідомлення для швидкого розшифрування повинна не перевищувати 24 біти.

4) Практично реалізовано схему перевірки правильності виконання обчислень делегованих віддаленому пристрою чи серверу на великих числах.

5) Знайдено такі повідомлення, що підробка результатів перевірки цілісності делегованих обчислень стає можливою та розглянута можливість виконання атак при умові наявності деякого секретного параметра.

В подальших дослідженнях необхідно вирішити задачу пошуку компромісу між виглядом і розміром параметру r та ймовірністю неодноначності розшифрування в криптосистемі Бенало. Адже як було показано в роботі швидкість розшифрування повідомлень в системи дуже повільна в порівнянні з ішими системами, таким як RSA наприклад, але зі зменшенням або преведенням до певного вигляду встрачається

стійкість системи. Або знайти ймовірність того, що випадково обране повідомлення буде розшифровано неправльно, тобто матиме місце неоднозначність розшифрування, на великих числах. Іншим можливим напрямком подальших досліджень може стати розробка модифіковано алгоритму розшифрування повідомлень, який дозволятиме виконувати розшифрування швидше не затрачаючи набагато більше пам'яті.

ПЕРЕЛІК ПОСИЛАНЬ

1. Benaloh J. *Dense probabilistic encryption.* Proc. of the workshop on selected areas of cryptography. Kingston, 1994. P. 120–128

2. Анісімов А.В., Новокшонов А.К. *Довірчі обчисления с використанням додавальної машини. I.* Кібернетика та системний аналіз. – 2017. – том 53, № 5. – С. 3-13.

3. Анісімов А.В., Новокшонов А.К. *Довірчі обчисления с використанням додавальної машини. II.* Кібернетика та системний аналіз. – 2018. – том 54, № 1. – С. 3-12.

4. Fousse, Laurent; Lafourcade, Pascal; Alnuaimi, Mohamed *Benaloh's Dense Probabilistic Encryption Revisited* - 2011.

5. Gentry C. *Fully homomorphic encryption using ideal lattices.* Symposium on Theory of Computing.2009. Vol. 9. P. 169–178.

6. Shafi Goldwasser and Silvio Micali, Probabilistic Encryption, Special issue of Journal of Computer and Systems Sciences, Vol. 28, No. 2, pages 270-299, April 1984

7. Zhang, Yuliang; Tsutomu Matsumoto; Hideki Imai (1988). "Cryptographic Applications of th-Residuosity Problem with an Odd Integer"

8. Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. In Advances in cryptology. Springer, 10–18. Junfeng Fan and Frederik Vercauteren. 2012

9. Whitfield Diffie and Martin E Hellman. 1976. New directions in cryptography. Information Theory, IEEE Transactions on 22, 6 (1976), 644–654

10. S McCURLEY Kevin. 1990. The discrete logarithm problem. Cryptology and computational number theory 42 (1990), 49.

11. ABBAS ACAR, HIDAYET AKSU, and A. SELCUK ULUAGAC, Florida International University MAURO CONTI, University of Padua *A Survey on Homomorphic Encryption Schemes: Theory and Implementation*

12. Ronald L Rivest, Adi Shamir, and Len Adleman. 1978b. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21, 2 (1978), 120–126.

13. R. Rivest, L. Adleman, M. Dertouzos. *Data banks and privacy homomorphisms*. Foundations of Secure Computation. – 1978. – C. 159–180.

14. Floyd R.W., Knuth D.E. Addition machines. SIAM Journal on Computing. 1990. Vol. 19, Iss. 2. P. 329–340

15. G. G. Arakelov, A. V. Gribov, and A. V. Mikhalev, *Applied homomorphic cryptography: examples*, Fundamentalnaya i prikladnaya matematika, vol. 21 (2016), no. 3, pp. 25—38.

16. Анатолій Анісімов, Андрій Новокшонов *Перевірка делегованих обчислень за допомогою додавальної машини*