Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка

Кафедра математичної інформатики факультету комп’ютерних наук та кібернетики

**ЗАСТОСУВАННЯ RSA-СХЕМ ДЛЯ ЗАДАЧ ШВИДКОЇ АУТЕНТИФІКАЦІЇ**

**Текстова частина до курсової роботи**

**за спеціальністю “Інформатика” 6.040302**

Керівник курсової роботи

проф., д. ф-м.н.

Анісімов А. В.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 р.

Виконав студент 3 курсу

Хоменко Тарас Вадимович

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 р.

Київ 2017

Зміст

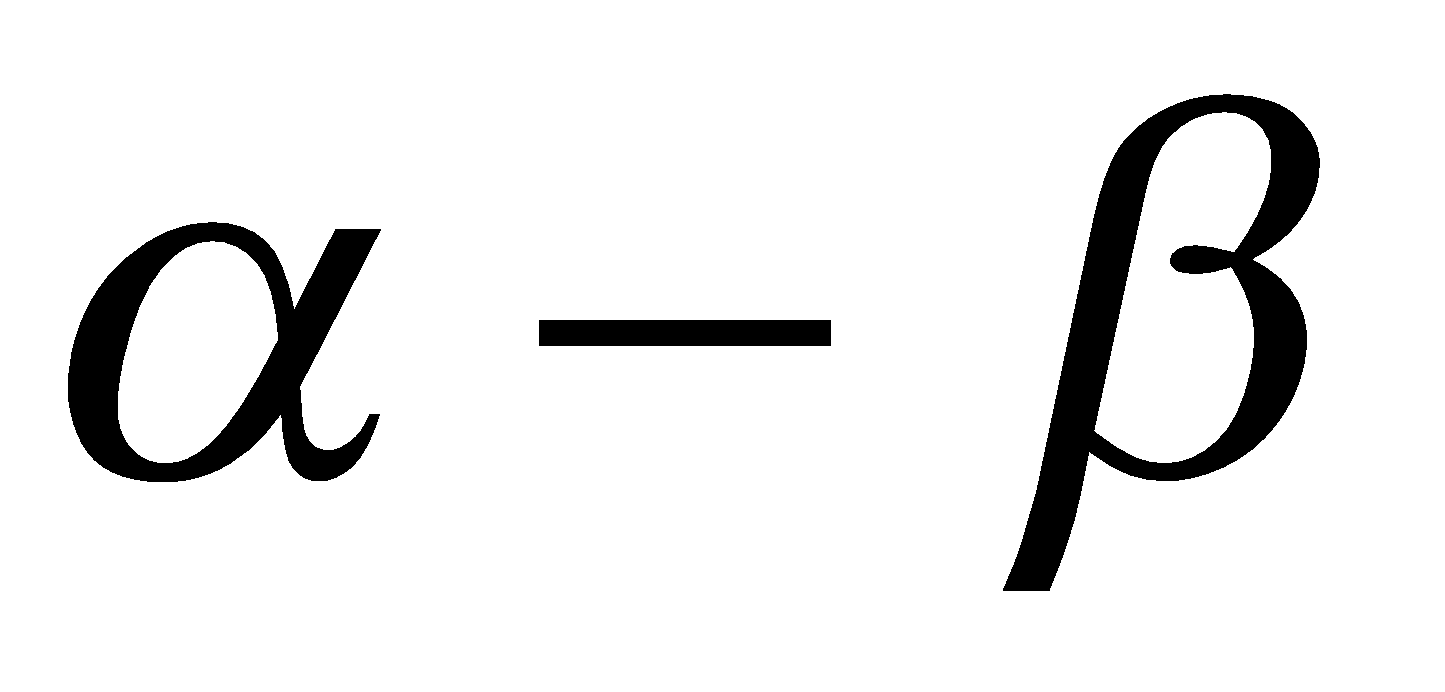
Анотація 3

Вступ 4

1 Теоретична частина 6

* 1. Аналіз існуючих підходів розв’язання задачі швидкої аутентифікації 7
  2. Аналіз запропонованого підходу розв’язання задачі

швидкої аутентифікації 12

* + 1. Опис другої  процедури 14
  1. Використані алгоритми 15
     1. Тест Міллера-Рабіна на простоту 15
     2. Розширений алгоритм Евкліда 16

2 Практична частина 17

2.1 Вибір інструментів для програмної реалізації 17

2.2 Структура програмної реалізації 17

2.2.1 Пакет «service» 17

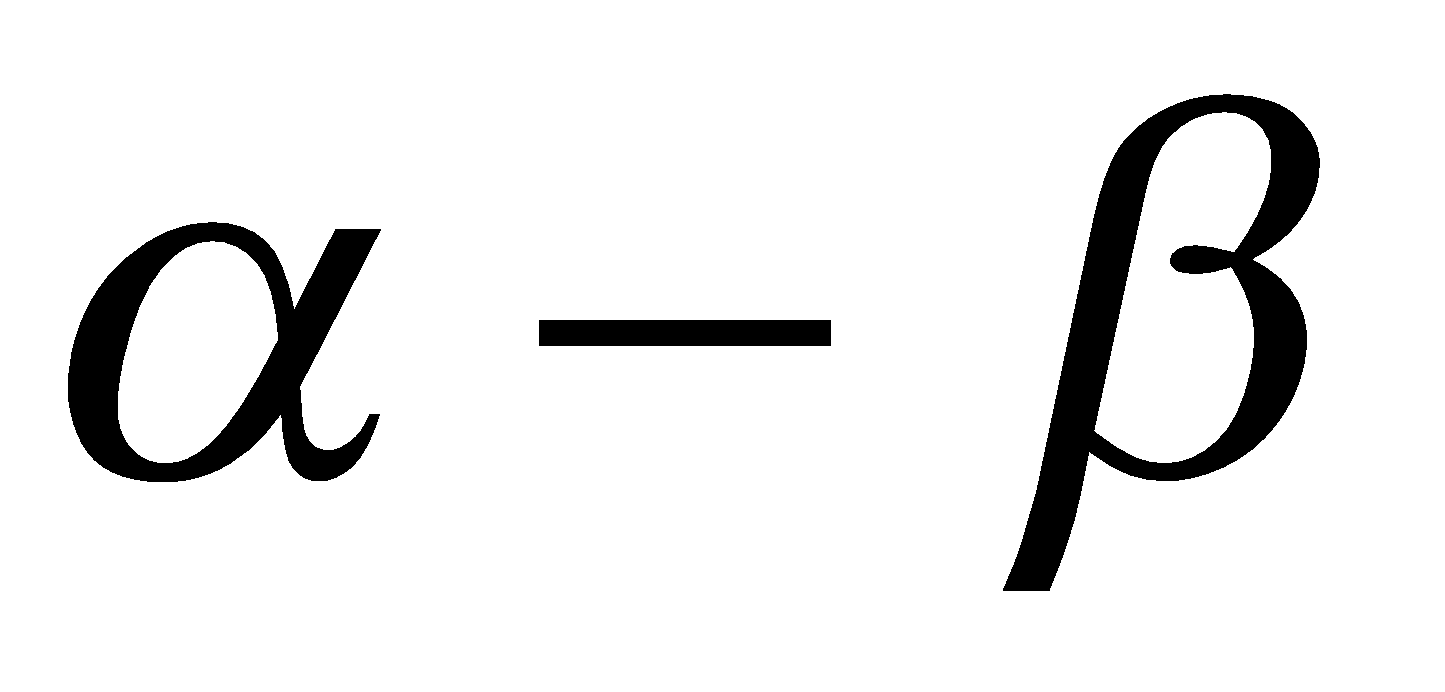
2.2.2 Пакет «main» 18

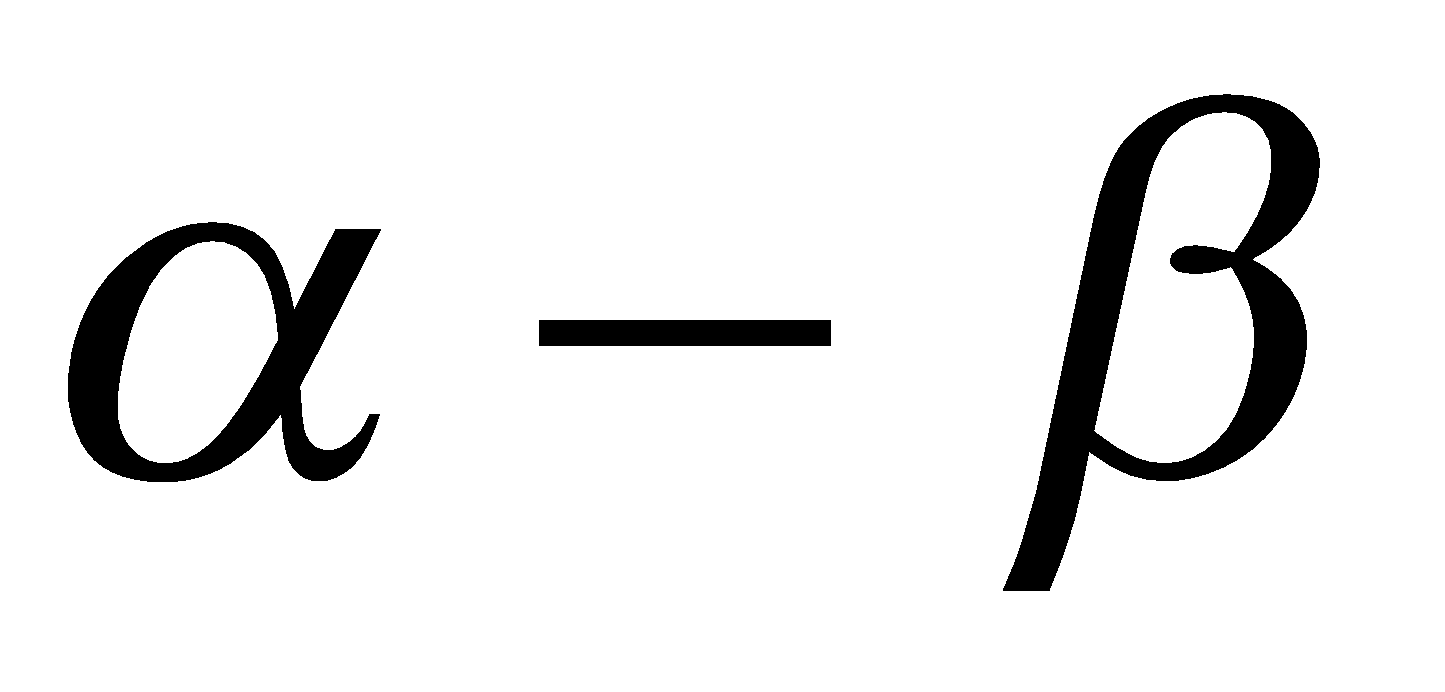
2.3Порівняння результатів виконання 21

Висновки 22

Список використаних джерел 23

**Анотація**

В даній роботі проведено аналіз існуючих підходів розв’язання задачі швидкої аутентифікації та запропоновано новий підхід. Він базується на основі використання  процедур запропонованих професором Анісімовим А.В. у статті «Коалиционные схемы с ключами общего доступа».

В процесі виконання роботи була написана програма, в якій було реалізовано другу  процедуру та приклад аутентифікації на основі даних процедур.

В результаті виконання роботи було встановлено, що кожен з цих підходів має свої переваги та недоліки. Застосування процедур є доволі ефективним підходом до створення таких схем на основі ключів відкритого доступу, які є одним з методів розв’язання задачі швидкої аутентифікації.

**Вступ**

Аутентифікація - процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора. У загальному випадку цей термін може відноситися до всіх аспектів інформаційної взаємодії: сеансу зв'язку, сторонам, переданим повідомленням і т.д.

Встановлення достовірності (тобто перевірка і підтвердження) всіх аспектів інформаційної взаємодії є важливою складовою частиною проблеми забезпечення достовірності одержуваної інформації. Особливо гостро ця проблема стоїть в разі, коли сторони не довіряють один одному, коли джерелом загроз може служити не тільки третя сторона (супротивник), але і сторона, з якої здійснюється взаємодія.

Проблеми швидкої аутентифікації є дуже актуальними в наш час, оскільки з розвитком інформаційних технологій проблеми захисту інформації при спілкуванні в глобальній мережі Інтернет, ідентифікації користувачів та несанкціонованого доступу постають все дедалі гостріше. А через те, що з кожним роком обчислювальні потужності комп’ютерів зростають, в схемах захисту інформації необхідно використовувати ключі зі все більшою та більшою довжиною. В даний час рекомендують використовувати ключі довжини 1024 або 2048 біт. З кожним роком дилема «безпека даних – швидкість передачі інформації» стає дедалі актуальнішою. В результаті хотілося б використовувати такі схеми захищеної передачі даних, які дають достатній рівень безпеки та високу швидкість передачі інформації.

Одним з напрямків, які можуть вирішити дану дилему є створення таких схем на основі ключів відкритого доступу, які дозволяють підтримувати високу надійність передачі інформації, але пришвидшують цю передачу за допомогою зменшення довжин ключів. Саме цьому напрямкові присвячена робота професора Анісімова під назвою «Коалиционные схемы с ключами общего доступа».[1]

Метою даної роботи є дослідження існуючих підходів розв’язання задач швидкої аутентифікації та аналіз нових підходів, які б могли реалізувати такі схеми, які б вирішували дану проблему. Під дослідженням мається на увазі: розгляд теоретичного обґрунтування коректності, ефективності та складності, оцінка складності реалізації, необхідних машинних ресурсів та часу виконання.

В наш час дослідження в області побудови швидких алгоритмів факторизації інтенсивно ведуться в усьому світі. Протягом останніх 5-10 років тема аутентифікації є однією з найбільш обговорюваних практично на всіх конференціях з інформаційної безпеки міжнародного та національного рівнів.

Навіть одні з найбільш поширених технологій для швидкої аутентифікації, такі як SecurID мають свої недоліки. Зловмисник може блокувати для користувача доступ і підключитися до сервера, поки не буде згенерований наступний токен-пароль. Або випадковий підбір пароля. Втрата або крадіжка токенів, що є найнебезпечнішою і майже непереборною вразливістю.

Таким чином, дослідження та розробка підходів розв’язання задач швидкої аутентифікації є дуже актуальним в наш час.

**1 Теоретична частина**

В даному розділі описано існуючі підходи до розв’язання задач швидкої аутентифікації та підходи запропоновані в даній роботі.

**1.1** **Аналіз існуючих підходів до розв’язання проблеми швидкої аутентифікації.**

Розв’язання даної проблеми базується на основі шифрування. Розрізняють симетричне та асиметричне шифрування.

При симетричному шифруванні, або шифруванні з секретним ключем, процес шифрування та розшифрування використовує деякий секретний ключ. Розрізняють поточне(побітове) та блочне(при шифруванні текст попередньо розбивається на блоки, як правило не менше 64 біт) шифрування.

Яскравим прикладом симетричного шифрування є стандарт шифрування DES(Data Encryption Standard). Він представляє собою блочний шифр, що поєднує дві базові техніки шифрування перемішування та підстановки, добутку по модулю 2, з довжиною блока 64 біта і довжиною ключа 56 біт. Зараз DES вважається ненадійним в основному через малу довжину ключа та розмір блоку. У 1999 році ключ DES було публічно дешифровано за 22 години 15 хвилин. Вважається, що алгоритм достатньо надійний для застосування у модифікації 3-DES, хоча існують розроблені теоретичні атаки. DES поступово витісняється алгоритмом AES(Advanced Encryption Standard), в якого розмір блока сягає 128 біт, а ключа - 128/192/256 біт. З 22 травня 2002 року AES прийнятий в якості американського стандарту шифрування урядом США.[2]

Переваги симетричного шифрування полягають у швидкості виконання перетворень. У свою чергу недоліки - секретний ключ відомий і отримувачу, і відправнику, що створює проблеми при розповсюдженні ключів.

При асиметричному шифруванні, або шифруванні з відкритим ключем, ключі шифрування та розшифрування завжди різні, але зв’язані між собою. Перший ключ є відкритим і може бути опублікованим для використання усіма користувачами системи, які шифрують дані. Розшифровування даних за допомогою відкритого ключа неможливе. Для розшифровування даних отримувач зашифрованої інформації використовує другий ключ, який є секретним. Зрозуміло, що ключ розшифровування не може бути визначеним з ключа зашифровування.

Прикладами симетричного шифрування є:

* алгоритм RSA;
* алгоритм Ель-Гамеля;
* алгоритм Мессі-Омури.

### Найбільш поширеним є алгоритм RSA. Його безпека побудована на принципі складності факторизації чисел. Генерація ключів проводиться таким чином:

1. вибираються два великі [прості числа](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) p\, і q\, приблизно 512/1024 біт завдовжки кожне;
2. обчислюється їх добуток n=pq \,;
3. обчислюється функція Ейлера

\varphi(n)=(p-1)(q-1)

Вона рівна кількості натуральних чисел, не більших n і взаємно простих з ним.

1. вибирається ціле e\, таке, що 1<e<\varphi(n) та e\, взаємно просте з \varphi(n);
2. за допомогою розширеного алгоритму Евкліда знаходиться число d\, таке, що

ed\equiv 1\pmod{\varphi(n)}.

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

Для того, щоб зашифрувати повідомлення m<n\, обчислюється

c=m^e\bmod\,n \,.

Число c\, і використовується в якості шифртексту. Для розшифрування потрібно обчислити

m=c^d\bmod\,n \,.

Неважко переконатися, що при розшифруванні ми відновимо вихідне повідомлення:

c^d\equiv (m^e)^d\equiv m^{ed}\pmod n\,

З умови ed\equiv 1\pmod{\varphi(n)} випливає, що

ed=k\varphi(n)+1 для деякого цілого k\,, отже

m^{ed}\equiv m^{k\varphi(n)+1}\pmod n

Згідно з теоремою Ейлера: m^{\varphi(n)}\equiv 1\pmod n, тому

m^{k\varphi(n)+1}\equiv m \pmod n

c^d\equiv m\pmod n\,

Система RSA використовується для захисту програмного забезпечення й у схемах цифрового підпису. Число *n* повинне мати розмір не менше 512 біт. На 2007 рік система шифрування на основі RSA вважалась надійною, починаючи з величини N в 1024 біта.

Алгоритм Ель-Гамаля заснований на принципі складності обчислення дискретних логарифмів в кінцевому полі. Криптосистема включає в себе алгоритм шифрування і алгоритм цифрового підпису. Алгоритм Ель-Гамаля лежить в основі стандартів електронного цифрового підпису в Росії (ГОСТ Р 34.10-94). Вона була запропонована Тахером Ель-Гамалем в 1984 році. [3] Він удосконалив систему Діффі-Хеллмана і отримав два алгоритми, які використовуються для шифрування і для забезпечення аутентифікації.

Алгоритм Мессі-Омури використовує просте число р, таке що р-1 має великий простий дільник в якості відкритого ключа, а секретний ключ визначається в процесі діалогу між користувачами.

Один з найбільш ефективних практичних протоколів аутентифікації – протокол Шнорра.[4] Нехай p і q – прості числа такі, що q ділить p-1 . Шнорр пропонує використовувати p довжини порядку 512 бітів і q – довжини близько 140 бітів. Нехай $ g\in Z_p$ таке, що

$ g^q=1 \pmod p$, $ g\neq 1$,$ x\in _RZ_q$ і $ y=g^x\pmod p$.

Задача обчислення значення x за заданим значенням y при відомих p, q і g називається завданням дискретного логарифмування. Для задачі дискретного логарифмування на даний момент не відомо ефективних алгоритмів. Тому в криптографії широко використовується гіпотеза про обчислювальну складність дискретного логарифма. Сформулюємо її більш суворо. Нехай n - зростаючий цілочисельний параметр, число p вибирається з множини всіх простих чисел довжини n таких, що p-1 має простий дільник довжиною не менше $ n^\varepsilon$ для деякої константи $ \varepsilon>0$, q – з множини всіх таких простих дільників числа p-1, g – з множини всіх чисел g таких, що

$ g^q=1 \pmod p$, $ x\in Z_q$.

Тоді функція $ f(x,p,q,g)= (g^x \nmod p, p, q,g)$ - одностороння. Рекомендації, дані Шнорром щодо довжин чисел p і q, можна трактувати наступним чином. На той момент (1989 р.) інвертування функції f вважалося практично нездійсненним вже для p і q довжини порядку 512 і 140 бітів відповідно. Тут, однак, слід враховувати, що прогрес в області обчислювальної техніки і в алгоритмічної теорії чисел може призвести до необхідності перегляду цих величин.

Нехай є два користувачі – користувач А та користувач В. В якості секретного ключа схеми аутентифікації користувач А вибирає випадкове число x з {1,…,q-1}. Далі А обчислює $ y=g^{-x}\nmod p$ і публікує відкритий ключ y. Відкриті ключі всіх учасників схеми повинні публікуватися таким чином, щоб виключалася можливість їх підміни (таке сховище ключів називається загальнодоступним сертифікованим довідником). Ця проблема, яка часто називається проблемою автентичності відкритих ключів, складає окремий предмет досліджень в криптографії і в даній главі не розглядається.

Схема аутентифікації Шнорра:

1. Користувач А вибирає довільне число $ k$ із множини{1,…,q-1}, обчислює

$ r=g^k\nmod p$

і відправляє $ r$ користувачу Б.

1. Користувач Б вибирає довільний запит $ e$ із множини $ \{ 0,\dots ,2^t-1\}$, де $ t$ - деякий параметр, і відправляє $ e$ користувачу А.
2. Користувач А обчислює

$ s=(k+xe) \nmod q$

і відправляє $ s$ користувачу Б.

1. Користувач Б перевіряє відношення

$ r=g^sy^e\nmod p$

і, якщо воно виконується, приймає доказ, в іншому випадку - відкидає.

В наш час досить поширеним засобом аутентифікації є SecurID (також RSA SecurID). Це технологія, розроблена компанією RSA, для надання двох-факторної аутентифікації між користувачем і мережевими пристроями. Двох-факторна аутентифікація RSA SecurID грунтується на тому, що користувач знає пароль (або PIN-код) і у нього є ключ, таким чином забезпечується набагато більш надійний рівень перевірки достовірності користувача в порівнянні з багаторазово використовуваними паролями. Дане рішення є єдиним рішенням, яке автоматично змінює пароль кожні 60 секунд. RSA пропонує підприємствам широкий спектр можливостей аутентифікації користувачів, які допомагають впевнено визначати користувачів перш, ніж вони зможуть взаємодіяти з критично важливими даними і додатками через різні мережі та ресурси.

Процес аутентифікації проходить наступним чином. При запиті користувача доступу до ресурсу, будь це веб-портал, робоча станція, мережеве сховище, VPN або віддаленого доступу, сервер замість стандартного запрошення на введення логіна і пароля запитується логін і кодова фраза (пароль). Кодова фраза представляється у вигляді особливої ​​ комбінацію пін-коду (4 цифри, які користувач пам'ятає) і токен-коду (6 цифр, які в даний момент висвічуються на токені). Користувач необхідно просто послідовно ввести ці 2 числа. Надану користувачем інформацію агент передає на сервер в зашифрованому вигляді. У сервері зберігається пін-коди користувачів і програмні копії всіх зареєстрованих токенів, відповідно він може перевірити надану користувачем інформацію. Залежно від результату перевірки агент або надає користувачеві доступ до ресурсу, або відмовляє йому в доступі.

Кожному токену відповідає 128-ми бітове випадкове число - початковий вектор генерації. Також в кожен токен вбудований годинник. Токен-код - результат роботи запатентованого компанією RSA алгоритму, який у якості параметрів бере поточний час, та початковий вектор генерації. За токен-кодом неможливо відновити початковий вектор генерації, так як алгоритм працює в одну сторону. [6]

Токен-код дійсний протягом однієї хвилини (змінюється раз на хвилину і тільки один раз). Так як на сервері зберігається відповідні токену початкові вектори генерації, то він у будь-який момент часу може по тому ж самому алгоритмом відновити поточний токен-код. Для випадку якщо годинник у сервера і токену розходяться, передбачена автоматична синхронізація. Тобто у разі якщо, наприклад, годинник у токена втік вперед, то сервер заносить до бази величину зсуву відповідну конкретному токену. Це досягається завдяки тому, що сервер обчислює пароль не тільки для поточної хвилини, але так само минулої і майбутньої хвилин. Таким чином, якщо PIN-код, введений користувачем вірний, а токен-код відповідає сусіднім хвилинах, то розсинхронізація враховується для подальших операцій. [7]

Недоліки технології SecurID:

* не захищена від атак виду людина-по-середині (man-in-the-middle). Зловмисник може блокувати для користувача доступ і підключитися до сервера, поки не буде згенерований наступний токен-пароль.
* випадковий підбір пароля. SecurID намагається вирішити цю проблему за допомогою обмеження кількості запитів на аутентифікацію протягом часового проміжку, на якому пароль не згенерований заново.
* є втрата або крадіжка токенів, що є найнебезпечнішою і майже непереборною вразливістю.

**1.2 Аналіз запропонованого підходу розв’язання задачі**

**швидкої аутентифікації**

Припустимо, ми маємо колектив дружніх учасників, які бажають встановити між собою безпечний зв’язок. Всю основну роботу по встановленню такого зв'язку вони доручають одному члену групи, який називається «лідером». Для забезпечення достатнього рівня безпеки комунікаційних повідомлень лідер повинен володіти необхідним набором засобів, включаючи організаційні заходи безпеки, а також потужний обчислювальний і алгоритмічний потенціал. Від решти учасників потрібно лише виконання мінімальних процедур, не пов'язаних зі складними процесами генерації надійних ключів.

Можливий такий варіант взаємодії лідера А та користувача В. Припустимо В необхідно організувати RSA-схему для безпечного отримання зовнішньої інформації. В звертається до А з відповідним запитом. Лідер А формує для В необхідну RSA-схему, тобто генерує два сильні простих числа p та q і вибирає експоненти кодування e і декодування d. Потім А від імені B розміщує для відкритого користування загальні ключі n = p\*q разом з експонентою кодування е. При цьому жодних передач від А до В не здійснюється.

Задача В, полягає в тому, щоб, прочитавши загальні параметри n і е, знайти d. Фактично В необхідно розкласти на множники число n, щоб визначити (n) і знайти d з рівняння

e ∙ d  1 mod (n).

Так як В не володіє загальним алгоритмом швидкого розкладання на множники, то А, при формуванні схеми для В, повинен врахувати деякий параметр – «лазівку», яку В знає апріорі. При цьому В не повинен виконувати ніяких складних дій для генерації «лазівки». В описаному нижче варіанті вирішення цього завдання від В потрібно тільки згенерувати «що-небудь» і надіслати цю інформацію лідерові А по захищеному напрямку від В до А.

Користувач В формує випадкове число i та передає PA(i).Лідер А, обчисливши i, повинен сформувати два простих числа p та q. Задача В, використовуючи i та число n, відновити p і q. Назвемо процедуру генерації p і q по i процедурою α, а алгоритм обчислення p і q по i та n - процедурою β. Ясно, що процедури α і β повинні бути пов'язані між собою. По суті справи, параметр i є "лазівкою" для обчислення звернення односторонньої функції, що задає добуток двох чисел. Найбільший інтерес представляють алгоритми процедур α і β.

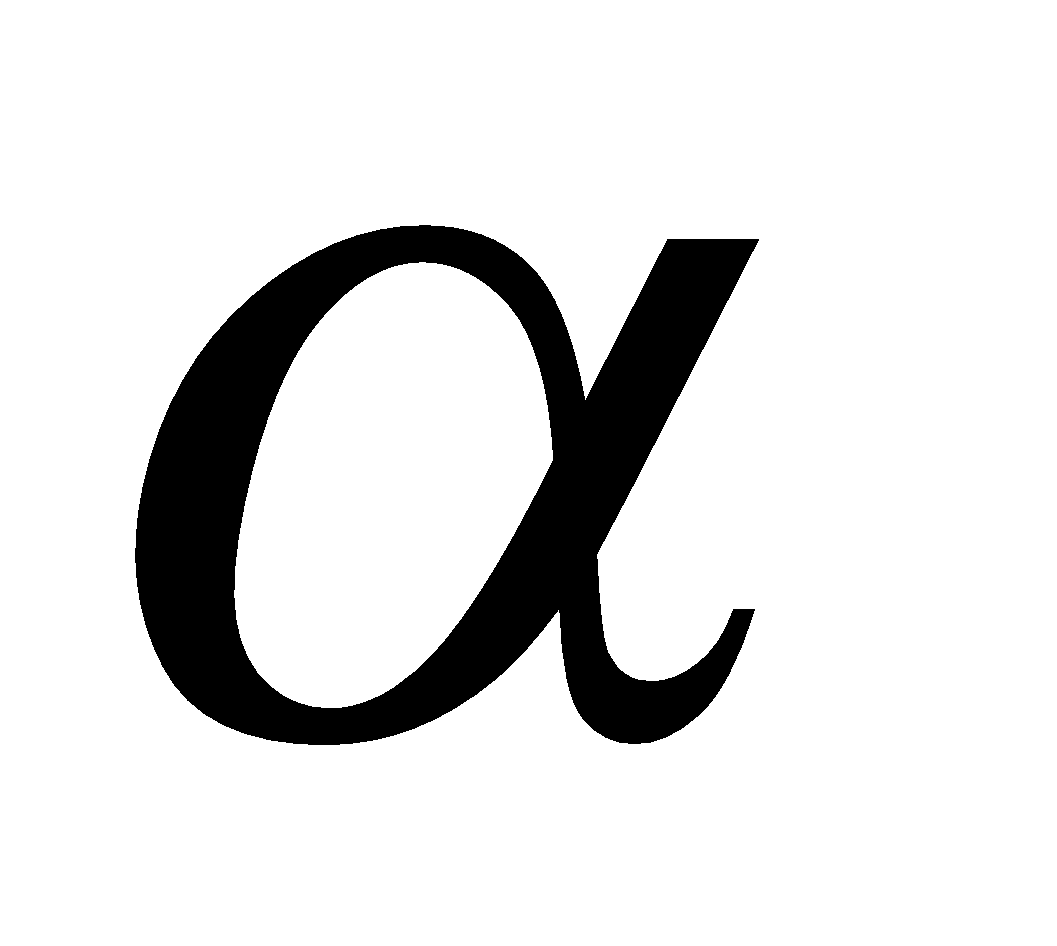
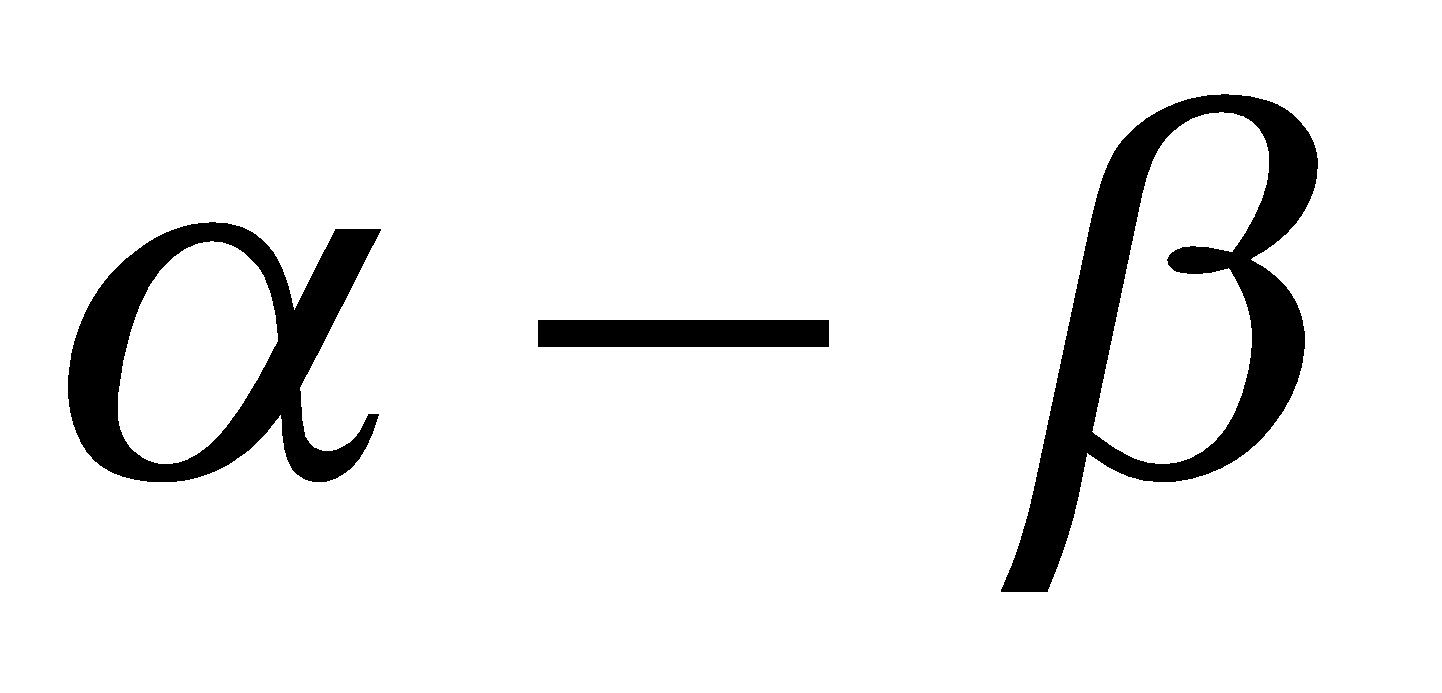
Наприклад, при аутентифікації учасник коаліції звертаючись до лідера А, аутентифікує себе включаючи в стартове, що посилається А, повідомлення стандартну ідентифікаційну інформацію (паролі, аргумент і значення хеш-функції на цьому аргументі, інформація для аутентифікації А, додаткова інформація). Лідер аутентифікує себе правильним підбором числа N = P ∙ Q, що посилається В і знанням аутентифікованої інформації. Для зовнішнього спостерігача Е імітація дій лідера коаліції А вкрай важко. Не володіючи стартовим посиланням i, Е може посилати В, тільки якесь довільне повідомлення заданого формату.

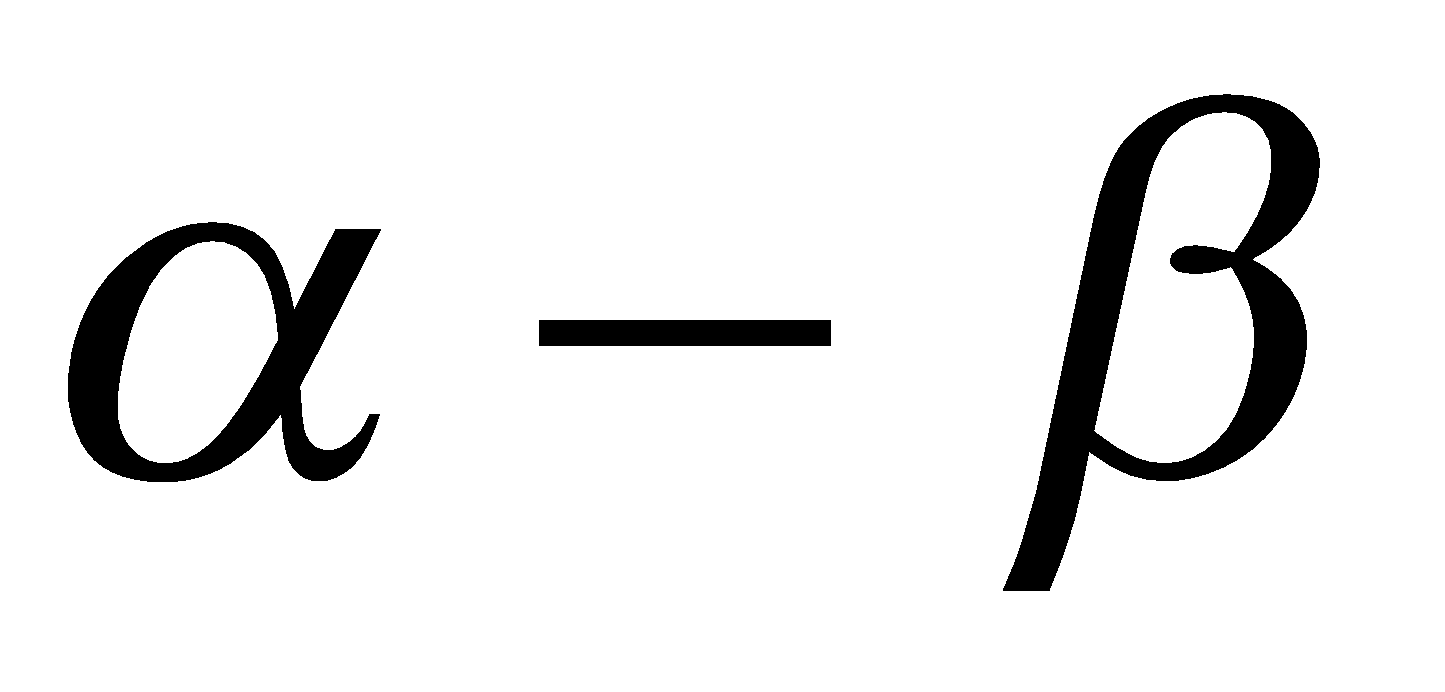
Якщо включити в алгоритм та процедуру β перевірку чисел на простоту, то три умови "S - просте", "р = HOD (n, λ1 + jv) - просте" та "q = - просте " вже роблять імітацію дій А вкрай малоймовірною. Крім того,

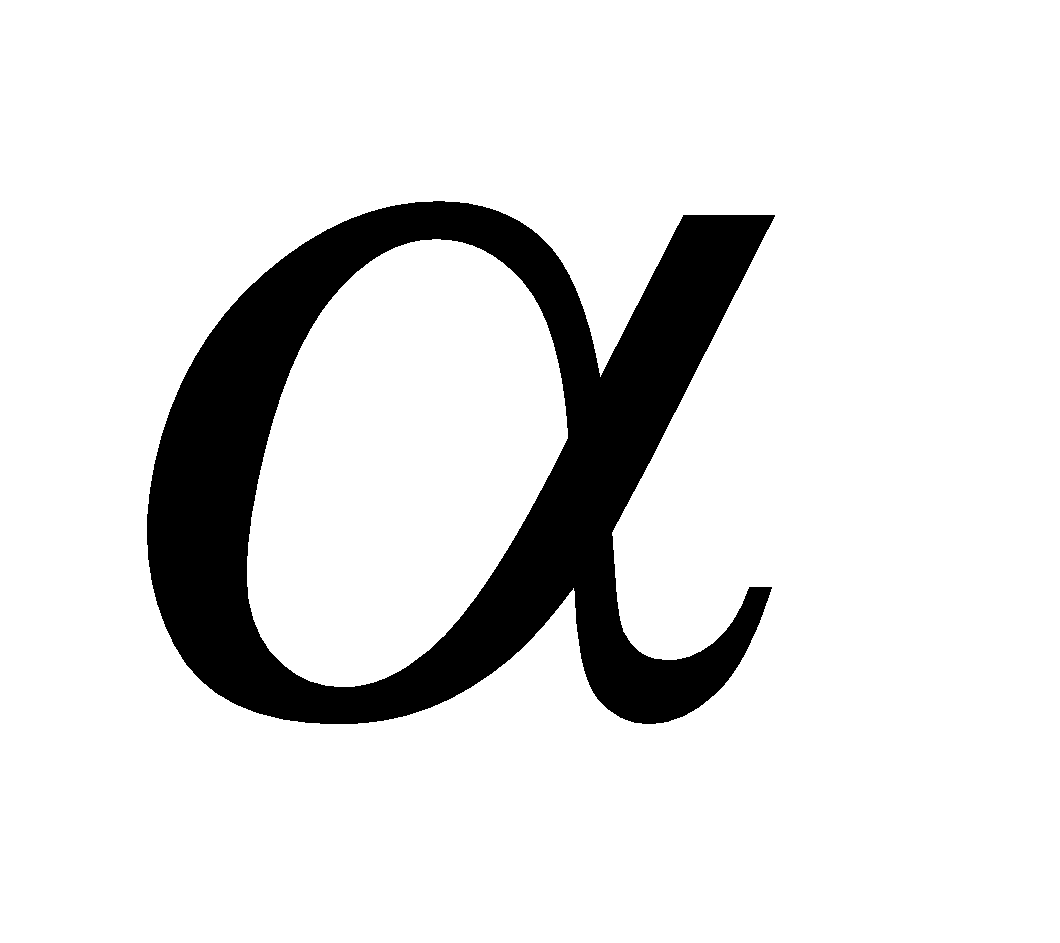
налаштувавши комунікаційний канал, А і В можуть обмінятися перевірочної

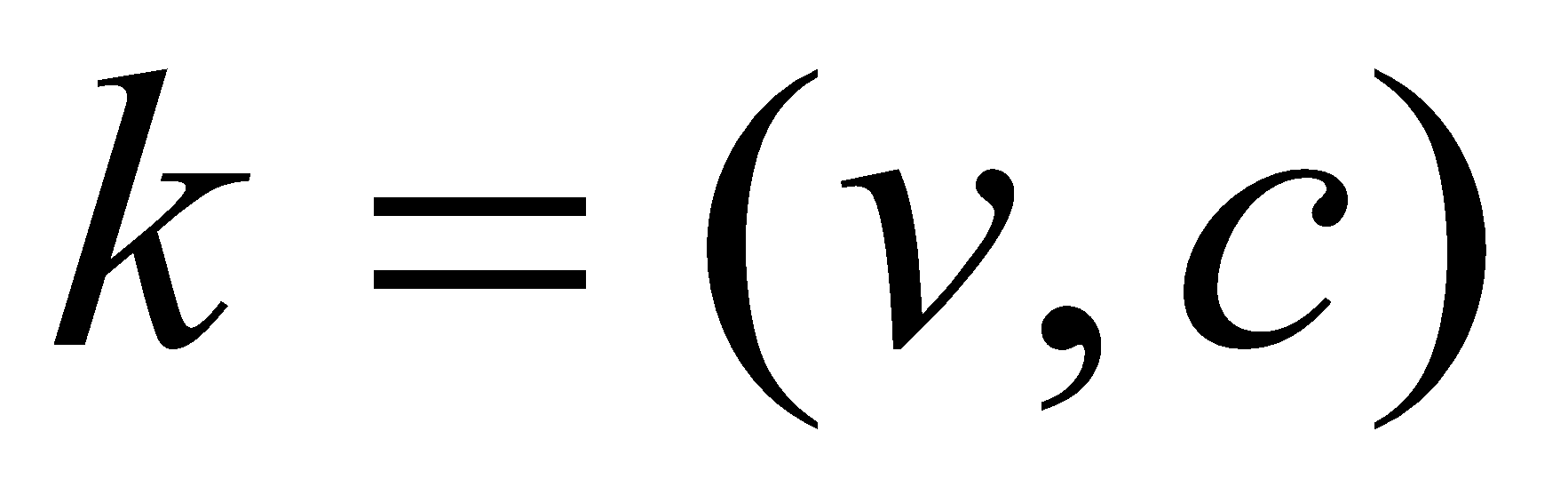
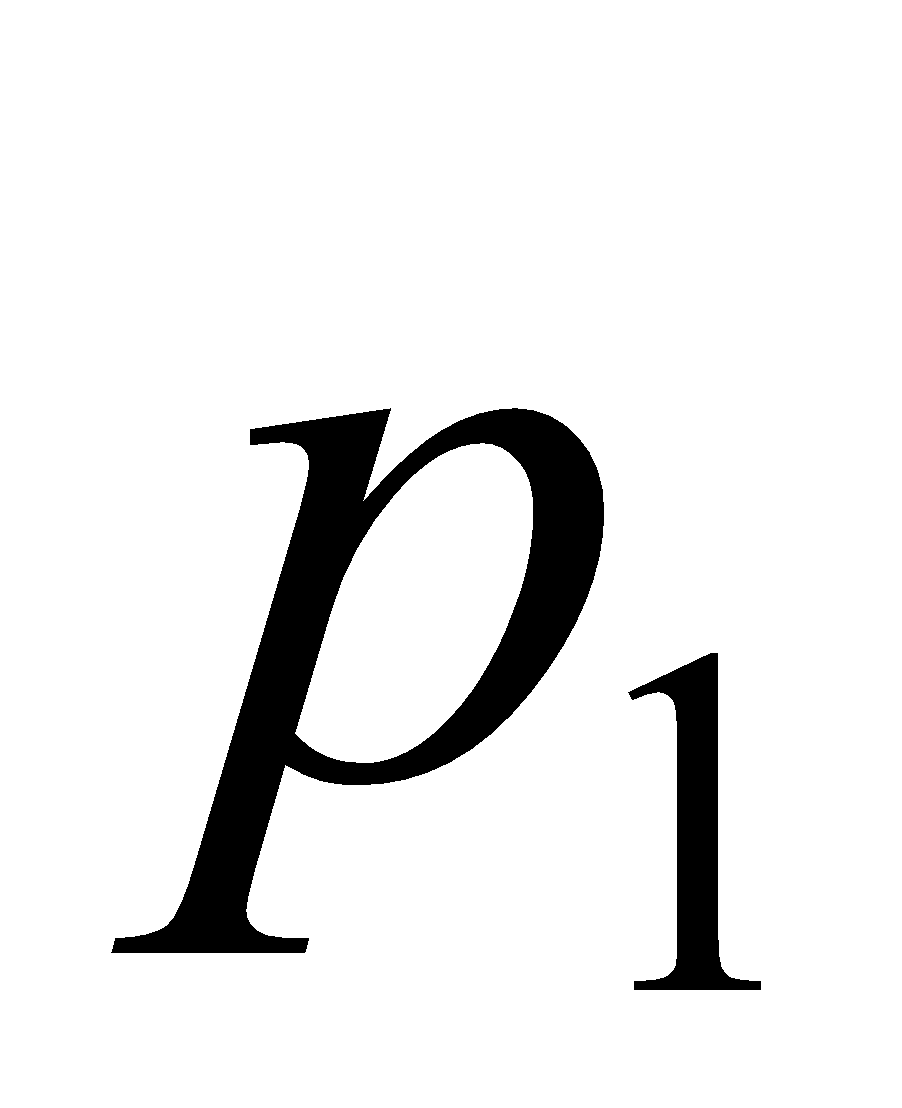
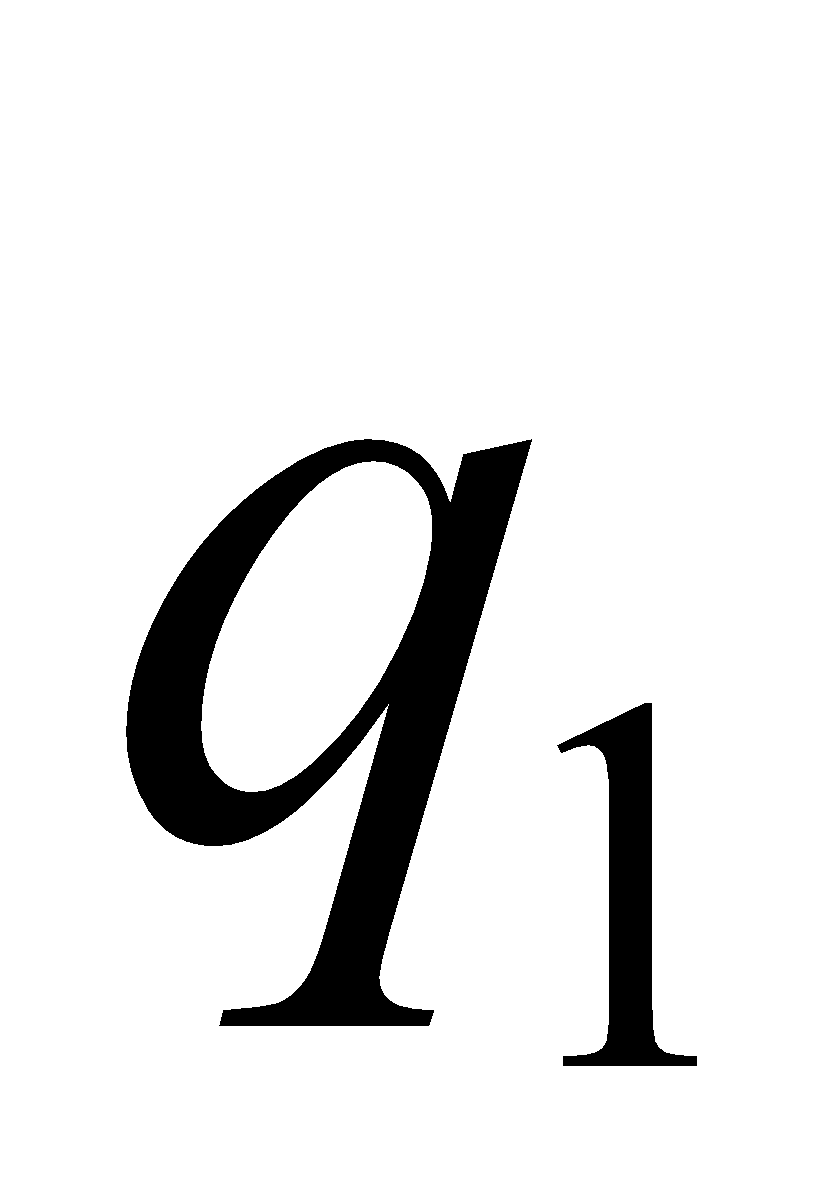
інформацією. У цьому випадку одержання однакових результатів при різних

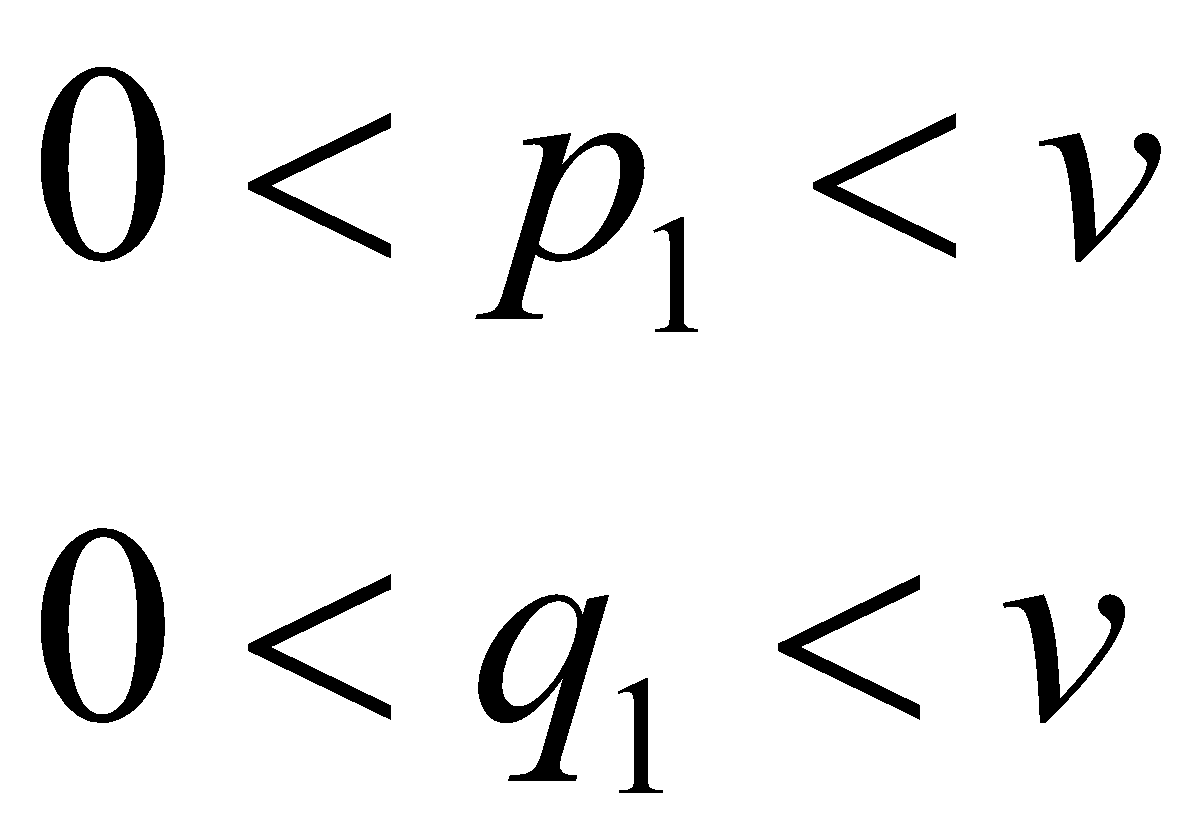
параметрах n = p1 · q1 та n'= p1' · q1' практично неможливо.

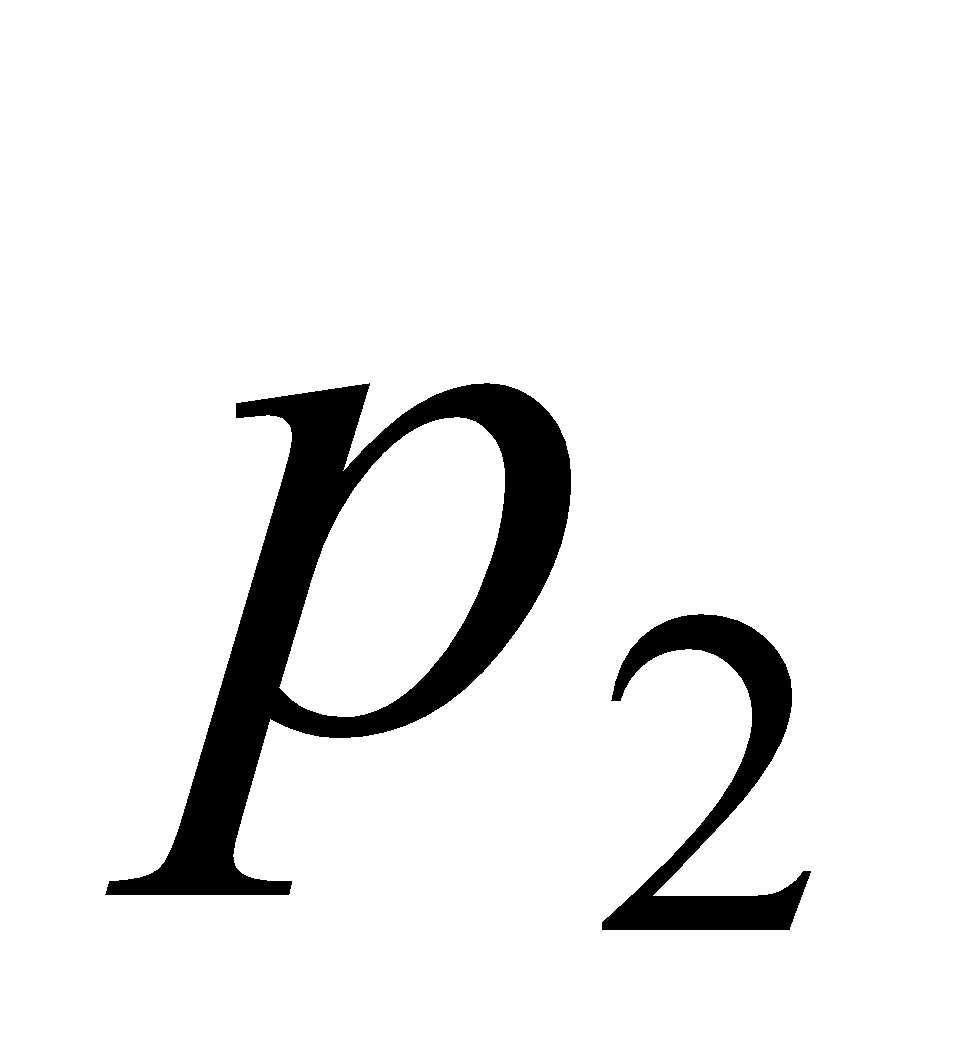
Процедура будує важко-вирішувану проблему P. При цьому використовується секретний ключ k. Знаючи k можливе швидке рішення проблеми P. Розглянемо це більш детально на прикладі другої процедури запропонованої професором Анісімовим Анатолієм Васильовичем в статті «Коалиционные схемы с ключами общего доступа».

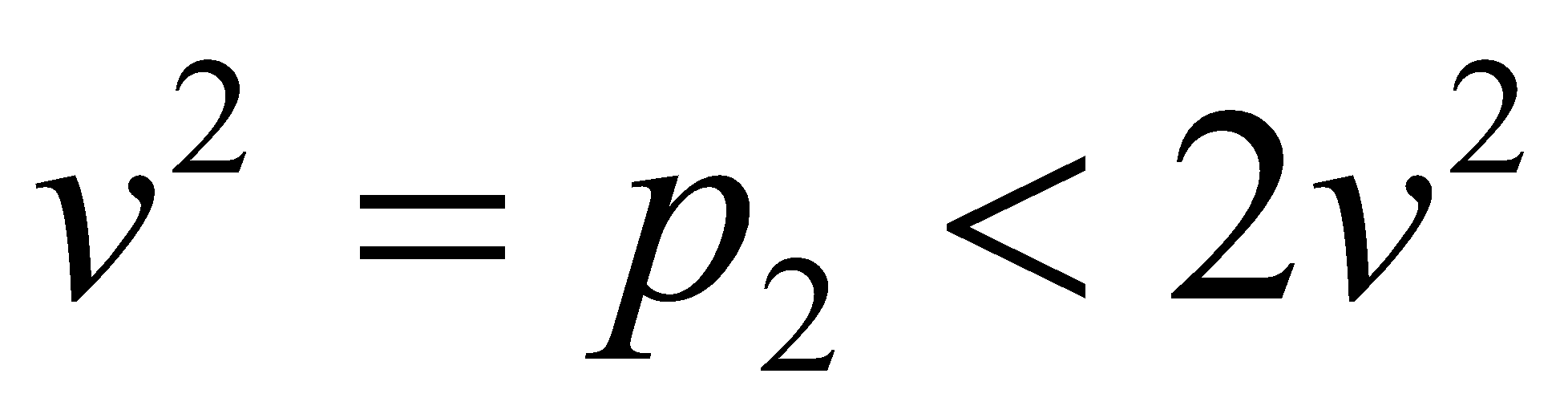
**1.2.1 Опис другої**  **процедури**

Процедура .

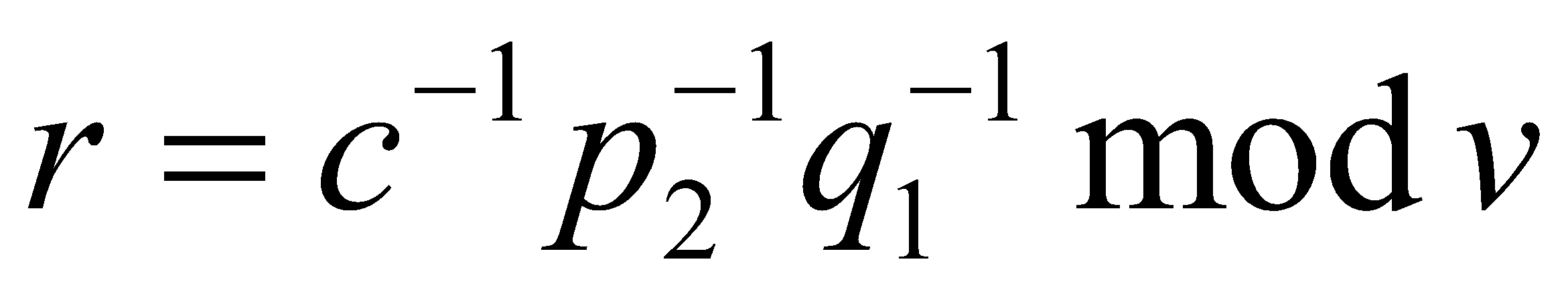
1. Згенерувати секретні параметри .
2. Згенерувати  и   - довільні прості числа, такі що



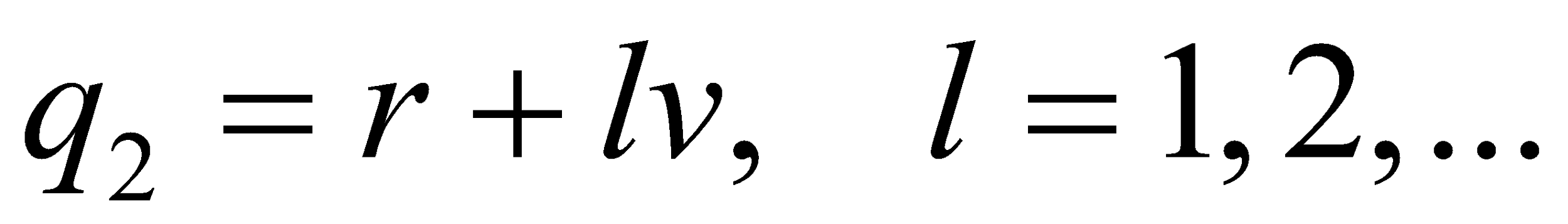
1. Згенерувати - довільне, таке що

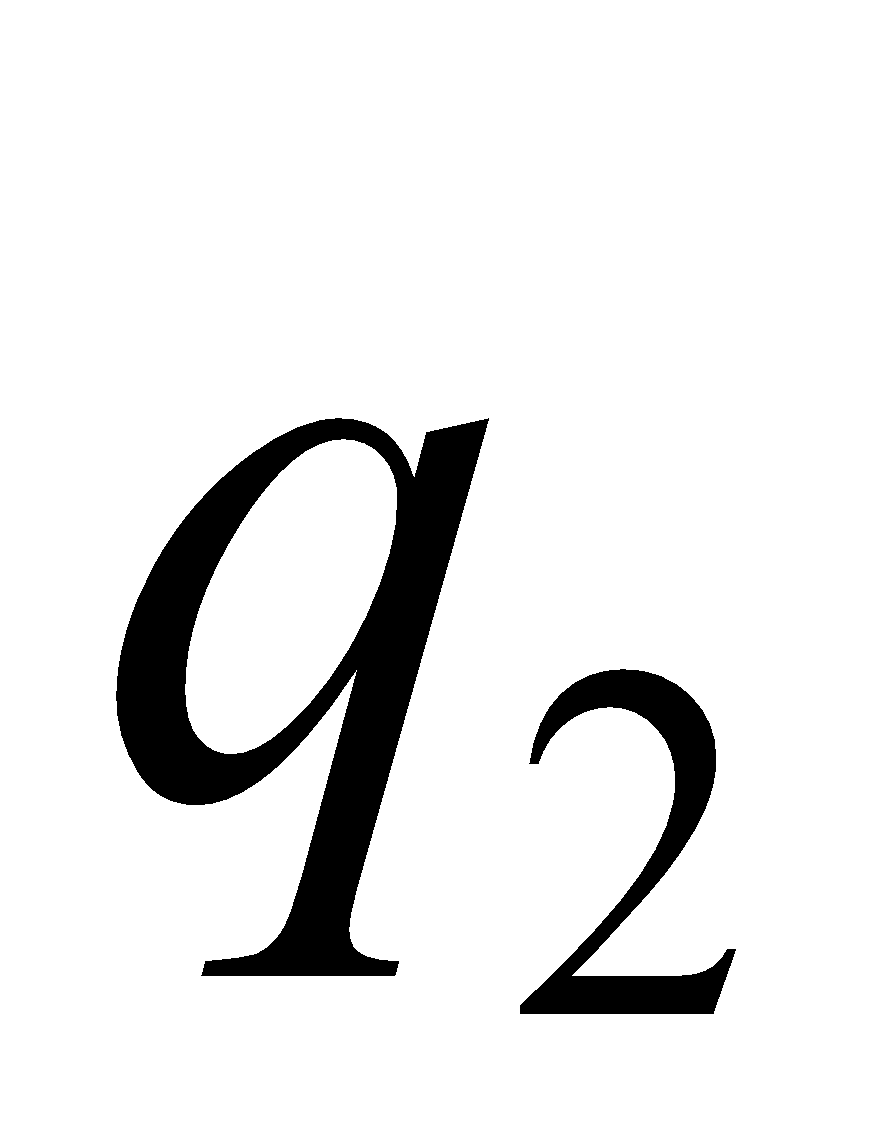


1. Обчислити

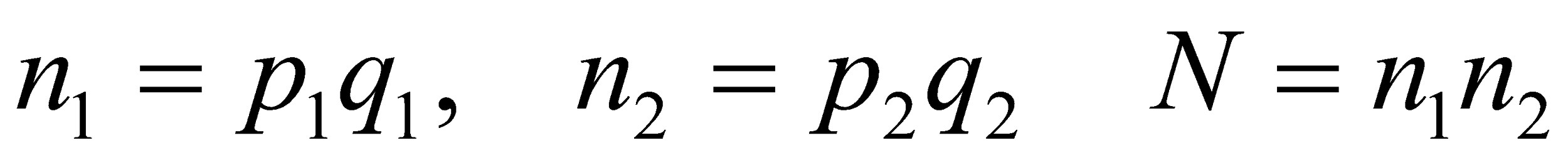


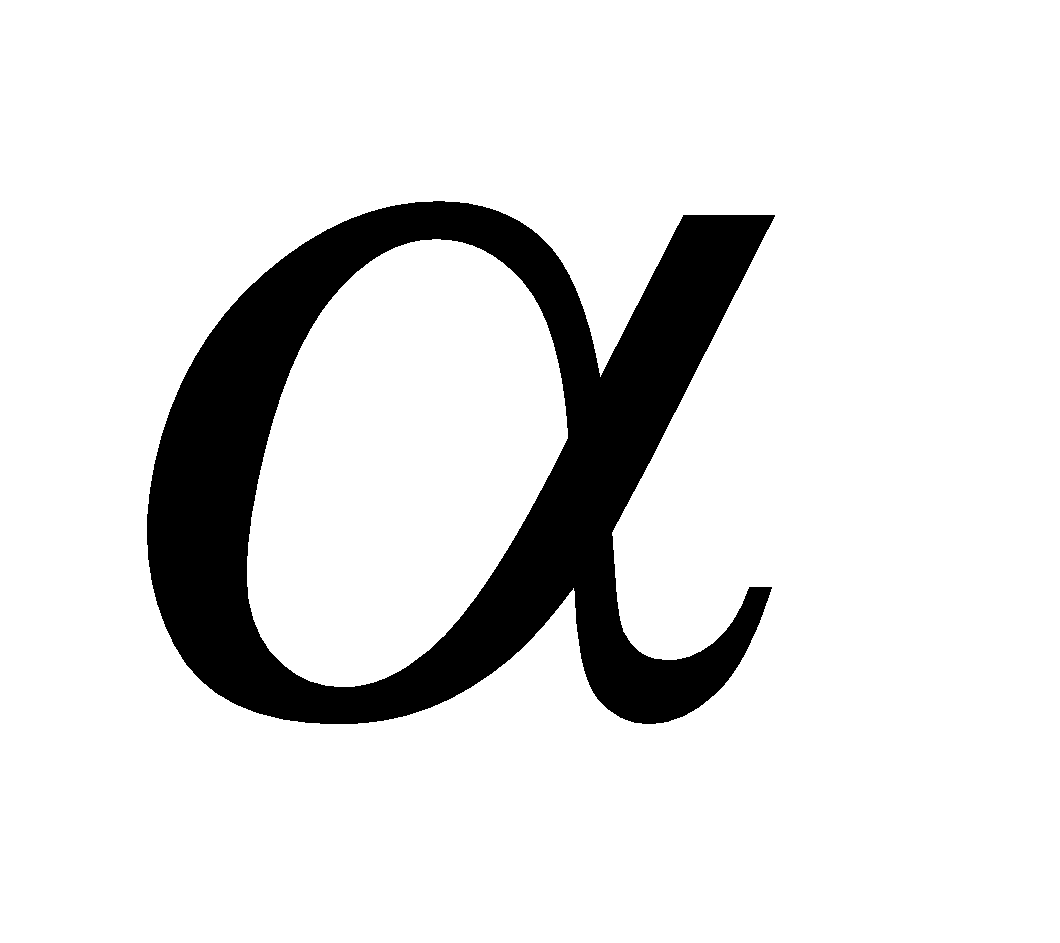
1. В арифметичній прогресії



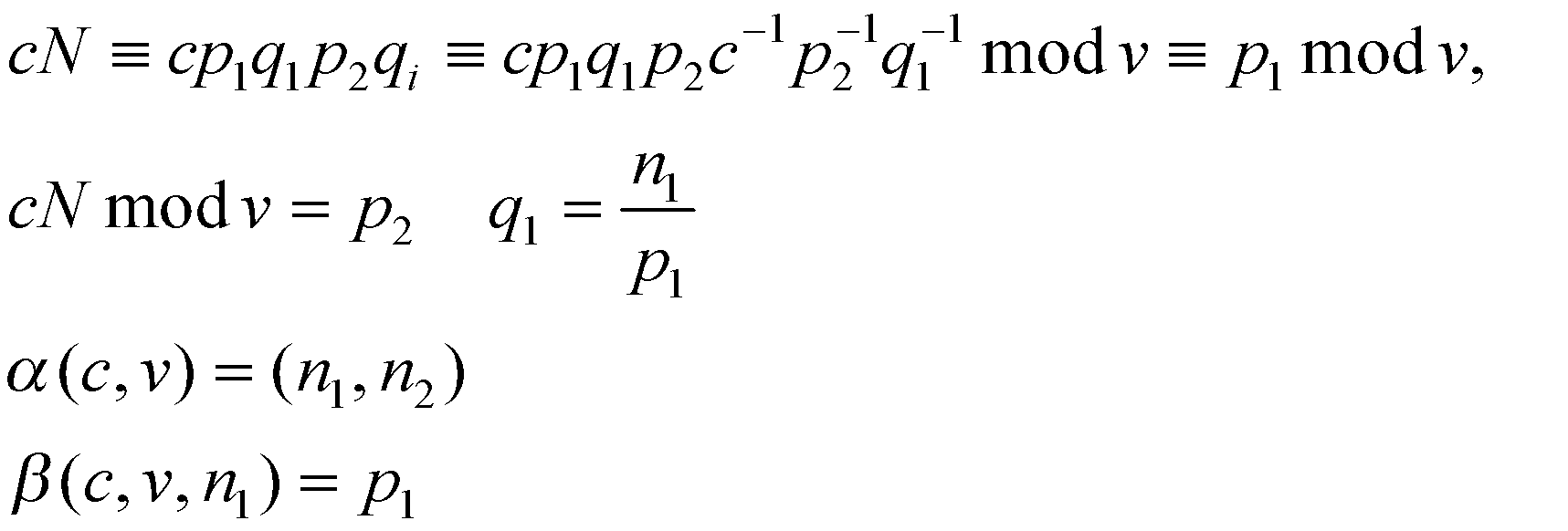
знайти перше просте .

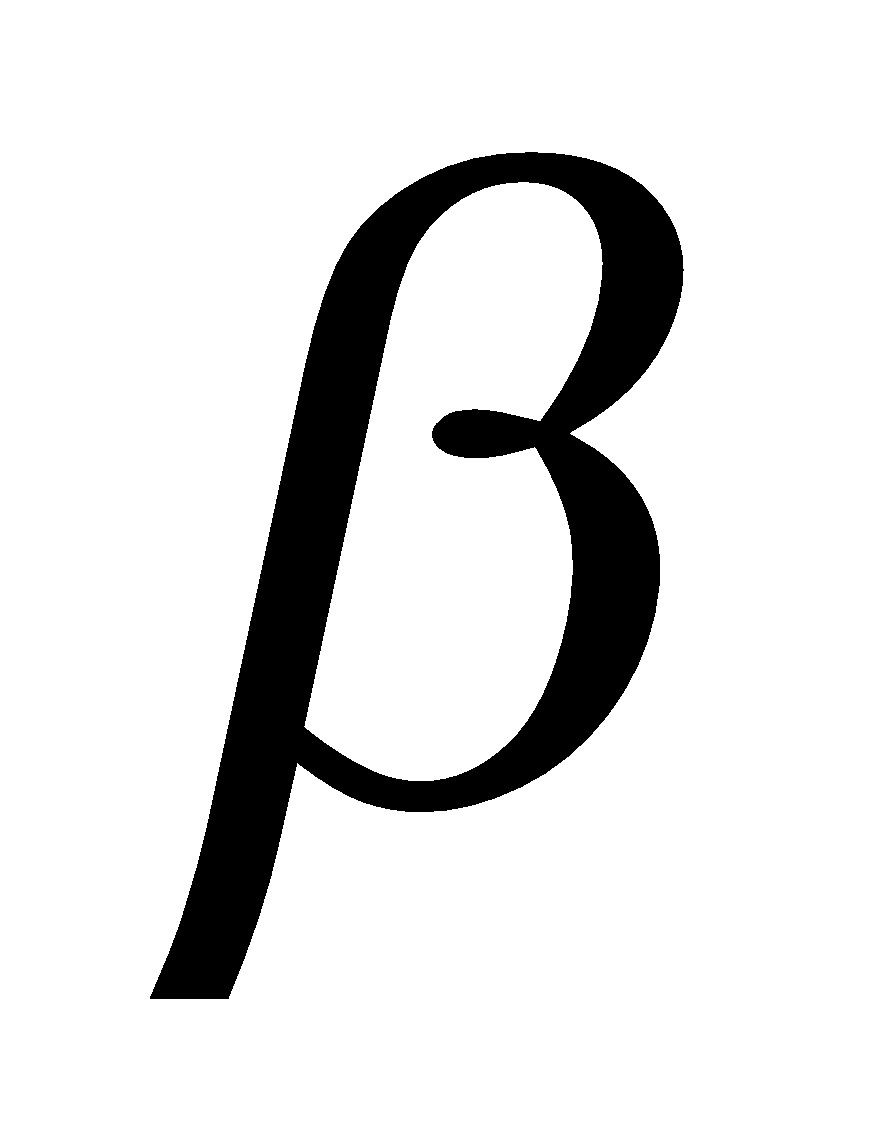
1. Обчислити

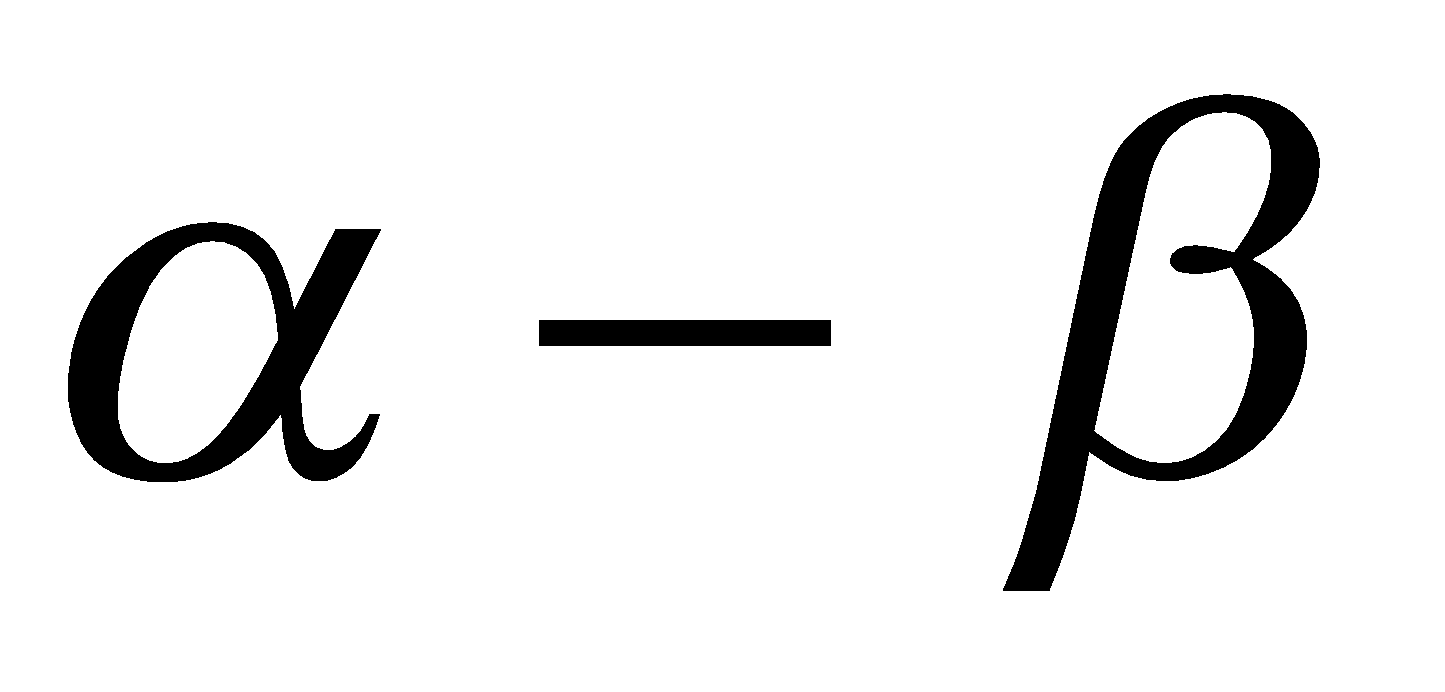


Кінець .

Процедура ß.



Кінець .

Доведення  процедур очевидне.

**1.3 Використані алгоритми**

Для реалізації процедур були використані алгоритми:

1. розширений алгоритм Евкліда – для представлення n у вигляді

n = λ1 u + λ2 v, де 0 <λ1 <v, де HOD (u, v) = 1;

1. тест Міллера-Рабіна на простоту – для генерації простих чисел та перевірки чи число є простим;

**1.3.1 Тест Міллера-Рабіна на простоту**

**Означення 1.** Число n називається псевдопростим за основою a, якщо виконується твердження малої теореми Ферма: an-1 = 1 (mod n).

Будь-яке просте число n буде псевдопростим за будь-якою основою a є Zn+.

Якщо вдалося знайти число а, за яким n не буде псевдопростим, то можна з впевненістю сказати, що n – складене.

**Означення 2.** Розв`язок рівняння x2 = 1 (mod n) не порівнюване по модулю n з +1, -1 називається нетривіальним коренем з одиниці в Zn.

**Твердження 1.** Якщо Zn містить нетривіальний корінь з одиниці, то число n – складене.

Ймовірнісний алгоритм Міллера-Рабіна побудований на наступних двох принципах:

1)Тест перевіряє число на псевдопростоту за декількома основами.

2)Тест по ходу обчислень перевіряє чи існує в Zn нетривіальний корінь з одиниці.

На початку алгоритму число n-1 представляється у вигляді n-1 = 2tu і задається кількість ітерацій перевірки s. Це робиться для того, щоб обчислювати an-1 (mod n) методом повторного піднесення до квадрату, що дозволяє перевіряти, чи не натрапили на нетривіальний корінь з одиниці. Якщо так, то число n – складене і алгоритм завершується. Якщо ні, то результат піднесення до степеня по модулю залишається у змінній xt. Якщо цей результат не дорівнює одиниці, то число n – не псевдопросте, а отже, і не просте.

**Твердження 2.** Нехай n – непарне складене число. Тоді серед елементів Z+n існує не менш ніж (n-1)/2 чисел, що є свідками того, що n – складене.

Таким чином для непарного n і для будь-якого цілого додатнього числа s ймовірність помилки алгоритму не більше ніж 2-s.

**1.3.2 Розширений алгоритм Евкліда**

Розширений алгоритм Евкліда  — це розширення алгоритму Евкліда. Окрім знаходження найбільшого спільного дільника для цілих *a* і *b*, як це робить алгоритм Евкліда, він також знаходить цілі *x* і *y* (одне з яких зазвичай від'ємне), які задовільняють рівнянню Безу.

ax + by = \gcd(a, b).

Розширений алгоритм Евкліда особливо корисний коли *a* і *b* взаємно прості числа, бо *x* — обернене щодо *a* за модулем *b*, а *y* — обернене щодо *b* за модулем *a*.

Опишемо розширений алгоритм Евкліда.

Якщо b=0, то НСД дорівнює a, і коефіцієнт біля a — одиниця, біля b — нуль. Інакше алгоритм рекурсивно викликає себе для b і a mod b, отримуючи в результаті d=bx'+(a mod b)y'. Лема Безу говорить про те, що d все одно можна представити у вигляді d = ax+by, і до такого виду рівняння приводиться нижче:

d=bx'+(a-(a [div](http://uk.wikipedia.org/w/index.php?title=Div&action=edit&redlink=1) b)\*b)y';

d=ay'+b(x'-(a [div](http://uk.wikipedia.org/w/index.php?title=Div&action=edit&redlink=1) b)y');

Звідси, коефіцієнт біля

x = y' (коефіцієнт біля a),

y = x' — (a [div](http://uk.wikipedia.org/w/index.php?title=Div&action=edit&redlink=1) b)\*y'.

1. **Практична частина**

**2.1 Вибір інструментів для програмної реалізації**

Програмна реалізація була реалізована на мові програмування Java. Проект написаний в програмному середовищі Intellij IDEA Community Edition 2016.3.5.

Java — об'єктно-орієнтована мова програмування, випущена в 1995 році компанією Sun Microsystems як основний компонент платформи Java. Зараз мовою займається компанія Oracle, яка придбала Sun Microsystems у 2009 році. Синтаксис мови багато в чому схожий на C та C++. У офіційній реалізації, Java програми компілюються у байткод, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Таким чином, Java має ряд переваг при реалізації даних алгоритмів:

* незалежність від архітектури, тобто те, що програма, написана на мові Java, працює на будь-якій підтримуваній апаратній чи системній платформі без змін у початковому коді та перекомпіляції;
* наявність ефективної реалізації довгої арифметики (клас BigInteger);

## 2.2 Структура програмної реалізації

Проект складається з наступних пакетів:

* + com.crypto.service;
  + com.crypto.main;

**2.2.1 Пакет** «service»

Даний пакет містить:

* Класс Triple, який реалізує структуру для реалізації розширеного алгоритму Евкліда:
* **private** BigInteger x,y,z;  
    
   **Triple**(BigInteger z,BigInteger x, BigInteger y)  
   {  
   **this**.x = x;  
   **this**.y = y;  
   **this**.z = z;  
   }  
   **public** BigInteger **getX**(){  
   **return** x;  
  }  
  **public** BigInteger **getY**(){  
   return y;  
  }  
  **public** BigInteger **getZ**(){  
   **return** z;  
  }
* клас Numbers:

**package** service;

**import** javafx.util.Pair;  
**import** java.math.BigInteger;

**public** **class** Numbers {

**public** BigInteger genPrime();

**public** boolean isprime(BigInteger prime);

**public** BigInteger find\_reverse(BigInteger p, BigInteger module)

private Triple Extended\_gcd(BigInteger a, BigInteger b)

}

* клас Procedures:

**package** service;

**import** java.math.BigInteger;

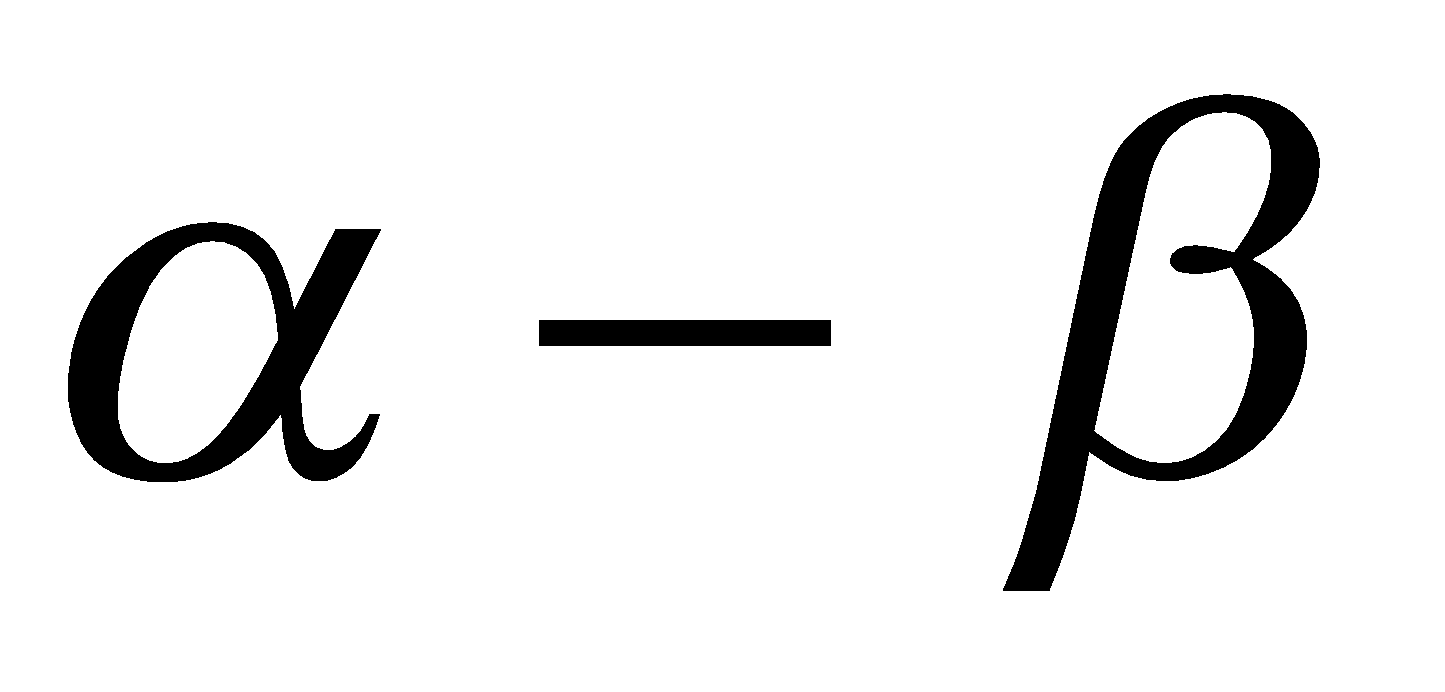
**public** **class** Procedures {

**public** Pair ProcedureAlpha(BigInteger v, BigInteger c);

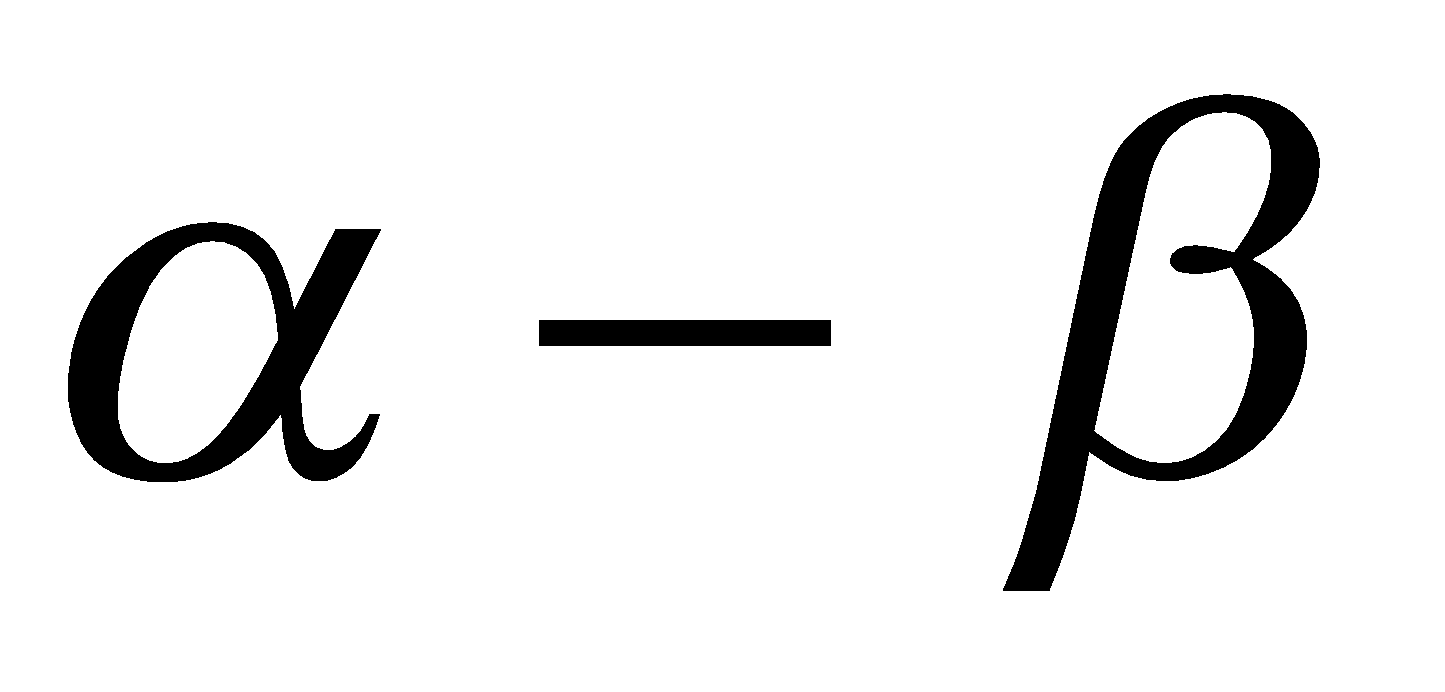
**public** BigInteger ProcedureBeta(BigInteger N, BigInteger v, BigInteger c);

}

**2.2.2 Пакет** «main»

Даний пакет містить класс Main, в якому відбувається запуск процедур та приклад аутентифікації. Результати роботи програми виводяться у консоль.

Продемонструємо виконання програми. Довжини згенерованих ключів 1024 біт.

Приклад другої процедури:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Example AlphaBeta Procedures #2:

----------------------------------------------

Generated secret key :

c = 132517196171540287802510647705462502334019536939675090496770617467722350405928350413249395260782678296399722333577717255668708823495935675589621908854833649486065683287171360099473865708890698160977504163202088106480759757812735006605230518657215469485154731483860537406738422479616806786450051820151959667861

v = 147265512396072212530372130468844502315337186341790395464007550853881532644596374563248122982067980815065170375135512697268118620016968106239369474948702148618652499684878370400869313690887654474714125605000351389374587643262993997975781860671656022889671378274458644268888000371955309654887677957536983782539

----------------------------------------------

AlphaProcedure :

p1 = 135162315756259508627755885095568490391750121928673603915362983823723706550203594649287039622626760755894137946303478119648743597588750773153413304926112006480502186869748799533867086812856076650239478296442440834727983913745248634072321285318398875214903255571453549364892838832171005532048000337262408815459

q1 = 125258263753291048244454951346755411841500286907536581114891920654054308794294327422187711110900645086188986375959487187060502803358795390591985605410295053324716873324209870185531920896040878323060461488549418699744083527597284339261577583058080122825246860551219636285259055607932237278831742987638636284867

p2 = 21687131141277698491389332226754641440896907408736795893243116768215052683310519862073346082007885359537888861134575248235021230763403058841966006250720192060736951032077290187582928612284799382861337097958664087286353295720067944257262055598490709601471994239009246309854015866504066159453270207574418227420965617835242189803594106324603011308301501473440884006433411265917420389923337589237281240383857855553020687291063347772631684194468633942534276688836800869967446469579614017645269107690608520420861392483871984780997093546133044787827262287413088639175400934460579523252574121309420511844374739061247271214031

q2 = 133415060612277644046176597771674970055939083880085384417992593239258318260697222453474641764882865038909214013163775055921282259183035998897058734094700909670801153916218644882986666574196282258415585627209651777046042654994462523502144872959605223762600023943477585071535391236055408636088763805226511099353293757060977554724222592812400771179044580284418685374226015580198888195363459711287339977117645523116825726006740002883955028531415777379447895099360976778798432631564780705354486148389126074490692913810928417817775643684479201479755596875756616918460025804918383304534109770511471837172795329994455439801191856347604774122590790417879066554782877476798171113810724181431423765250465209082490300415587386009411794961335617358695540919186565746033153853999793351339836701453268278334469994661394255571940357987468205265249988489114412324078164797424161224608538581303332746856951580762609781782213450663677928385429

AlphaProcedure time: 13821 mls

----------------------------------------------

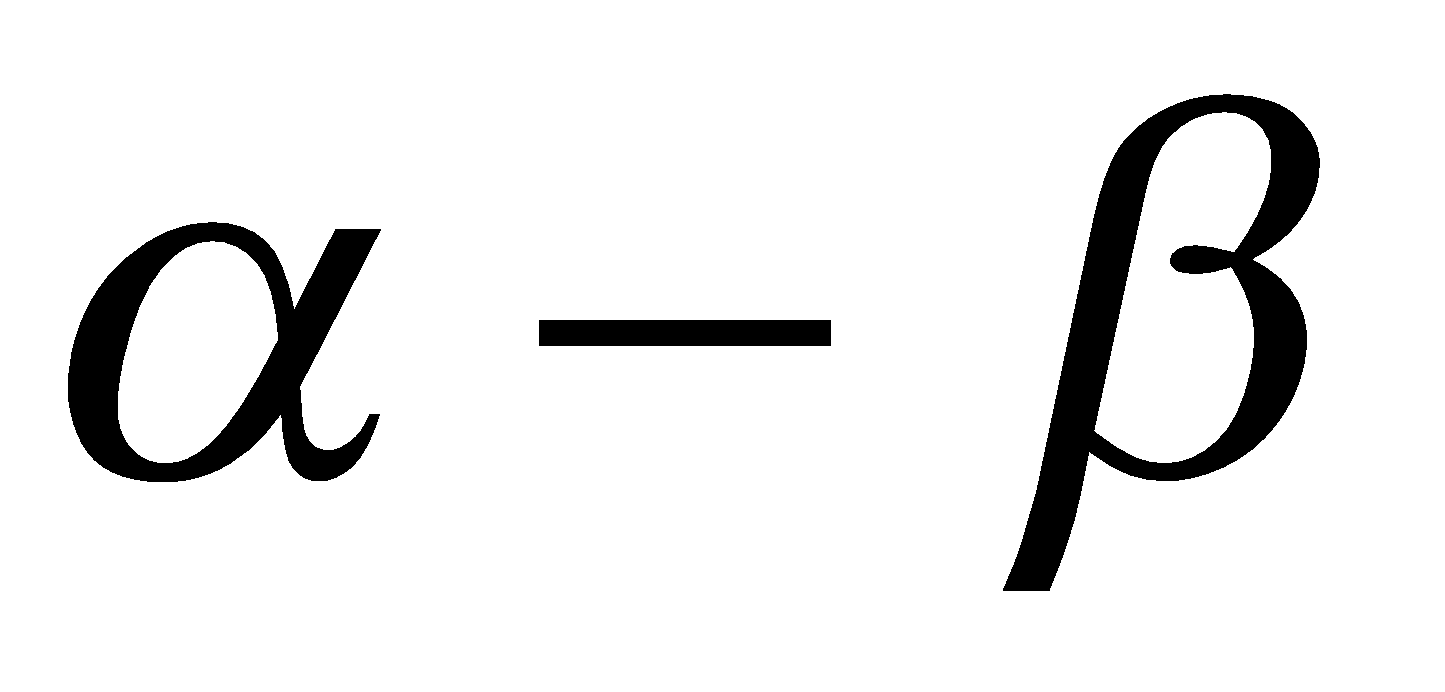
BetaProcedure :

p1 = 135162315756259508627755885095568490391750121928673603915362983823723706550203594649287039622626760755894137946303478119648743597588750773153413304926112006480502186869748799533867086812856076650239478296442440834727983913745248634072321285318398875214903255571453549364892838832171005532048000337262408815459

q1 = 125258263753291048244454951346755411841500286907536581114891920654054308794294327422187711110900645086188986375959487187060502803358795390591985605410295053324716873324209870185531920896040878323060461488549418699744083527597284339261577583058080122825246860551219636285259055607932237278831742987638636284867

BetaProcedure time: 16 mls

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Приклад аутентифікації з використанням другої  процедуи:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Example Authentication using AlphaBeta Procedures #2:

----------------------------------------------

Generated secret key :

p1 = 135162315756259508627755885095568490391750121928673603915362983823723706550203594649287039622626760755894137946303478119648743597588750773153413304926112006480502186869748799533867086812856076650239478296442440834727983913745248634072321285318398875214903255571453549364892838832171005532048000337262408815459

q1 = 125258263753291048244454951346755411841500286907536581114891920654054308794294327422187711110900645086188986375959487187060502803358795390591985605410295053324716873324209870185531920896040878323060461488549418699744083527597284339261577583058080122825246860551219636285259055607932237278831742987638636284867

p2 = 21687131141277698491389332226754641440896907408736795893243116768215052683310519862073346082007885359537888861134575248235021230763403058841966006250720192060736951032077290187582928612284799382861337097958664087286353295720067944257262055598490709601471994239009246309854015866504066159453270207574418227420965617835242189803594106324603011308301501473440884006433411265917420389923337589237281240383857855553020687291063347772631684194468633942534276688836800869967446469579614017645269107690608520420861392483871984780997093546133044787827262287413088639175400934460579523252574121309420511844374739061247271214031

q2 = 133415060612277644046176597771674970055939083880085384417992593239258318260697222453474641764882865038909214013163775055921282259183035998897058734094700909670801153916218644882986666574196282258415585627209651777046042654994462523502144872959605223762600023943477585071535391236055408636088763805226511099353293757060977554724222592812400771179044580284418685374226015580198888195363459711287339977117645523116825726006740002883955028531415777379447895099360976778798432631564780705354486148389126074490692913810928417817775643684479201479755596875756616918460025804918383304534109770511471837172795329994455439801191856347604774122590790417879066554782877476798171113810724181431423765250465209082490300415587386009411794961335617358695540919186565746033153853999793351339836701453268278334469994661394255571940357987468205265249988489114412324078164797424161224608538581303332746856951580762609781782213450663677928385429

----------------------------------------------

Leader sent N to B...

----------------------------------------------

BetaProcedure by user:

p1 = 36169550359003039213563039756209989114292479373671704593893421566873985365909246418932654141167925444032791142486374085866555866923807815788812773009390563228115587611614266043912669757560483117054350865314152175915874124594673917072166472031941362053625496911083588325427691648837783179119785330346866422309

BetaProcedure time: 16 mls

----------------------------------------------

Secret param p are equal

Authentication was passed successfully

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**2.3 Порівняння результатів виконання**

Таблиця 2.1 - Порівняння часу роботи процедур

|  |  |  |  |
| --- | --- | --- | --- |
| № | Довжини згенерованих ключів, бітів | Час роботи другої α процедури, млс | Час роботи другої ß процедури, млс |
| 1 | 32 | 9 | 0 |
| 2 | 64 | 26 | 0 |
| 3 | 128 | 119 | 0 |
| 4 | 256 | 738 | 0 |
| 5 | 512 | 3243 | 7 |
| 6 | 1024 | 11349 | 16 |

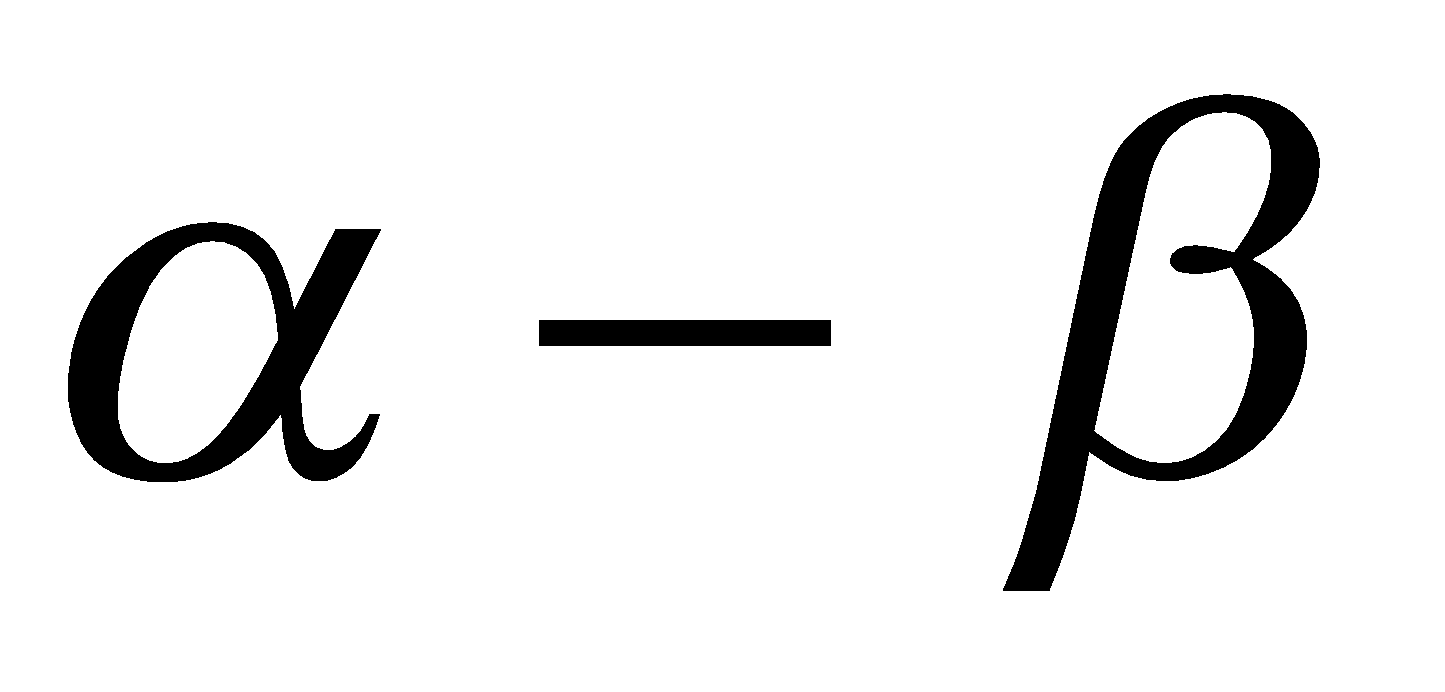


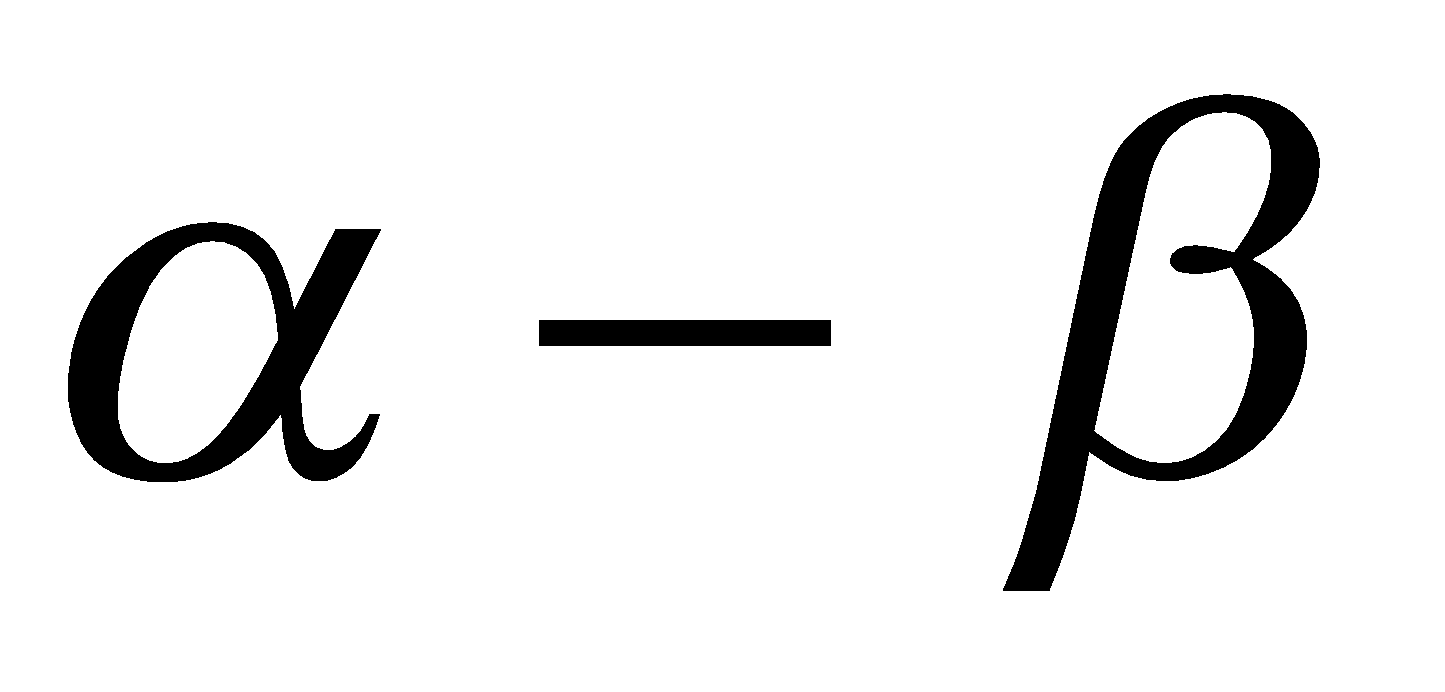
Рисунок 2.1 - Порівняння часу роботи процедур

.

Також результати програми показали, що час роботи процедури α зростає досить швидко зростає зі збільшенням довжини чисел, та залежить від значень згенерованих ключів. Коли час ß процедури не суттєво збільшується зі зростанням довжини чисел.

**Висновки**

Проблема розв’язання задач швидкої аутентифікації, тобто підтвердження автентичності ідентифікатора об'єкта, є дуже актуальною в наш час. Щоденно в глобальній мережі Інтернет проводяться безліч різноманітних важливих операцій. Відповідно, питання про необхідність достеменно знати, чи той це ресурс, за який він себе видає, і чи є користувач, який прагне отримати доступ до даного ресурсу, легальним загострюється все дедалі більше. Саме цю проблему було досліджено в даній роботі на основі вже існуючих підходів та реалізації нових на основі використання  процедури запропонованої професором Анісімовим А.В. у статті «Коалиционные схемы с ключами общего доступа».

В роботі було проаналізовано та реалізовано одну з двох  процедури та наведено приклад аутентифікації за домогою цієї процедури. В результаті виконання роботи було встановлено, що навіть при великій довжини ключів користувачі можуть швидко ідентифікувати себе. Отже, застосування α-ß процедур є доволі ефективним підходом до створення таких схем на основі ключів відкритого доступу, які є одним з методів розв’язання задачі швидкої аутентифікації.

**Список використаних джерел**

1. *УДК 621.391.7 : 336.71 (075.8)* «Коалиционные схемы с ключами общего доступа», Анісімов А. В., 26 с.
2. Data Encryption Standard [Електроний ресурс] : Вікіпедія. Вільна енциклопедія. – 2013. – Режим доступу до статті :

<http://en.wikipedia.org/wiki/Data_Encryption_Standard>

1. Data Encryption Standard [Електроний ресурс] : Вікіпедія. Вільна енциклопедія. – 2013. – Режим доступу до статті :

<http://en.wikipedia.org/wiki/ElGamal_encryption>

1. Pollard J. Theorems on factorization and primality testing // Proceedings of the Cambridge Cambridge Philosophical Society. — 1974. — Vol. 76. — Pp. 521–528.
2. Lehman S. Factoring large integers // Mathematics Of Computation. —1974. — Vol. 28, no. 126. — Pp. 637–646.
3. Miller G. Riemann’s hypothesis and tests for primality // Journal of Computer and System Sciences. — 1976. — Vol. 13, no. 3. — Pp. 300–317.
4. Montgomery P. Speeding the Pollard and elliptic curve methods of factorization // Mathematics Of Computation. — 1987. — Vol. 48. — Pp. 243–264.
5. Nivash G. Cycle detecting using a stack // Journal Information Processing Letters. — 2004. — Vol. 90.
6. *Василенко О. Н.* Теоретико-числовые алгоритмы в криптографии. — М.: МЦНМО, 2003. — 328 с. — ISBN 5-94057-103-4