МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА
Факультет математики, фізики та інформаційних технологій
Кафедра комп'ютерної алгебри та дискретної математики

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»

на тему «Рюкзачні криптосистеми»
«Knapsack cryptosystems»

Виконав: студент денної форми навчання
спеціальності 123 – Комп'ютерна інженерія
Богдан Георгій Васильович
Керівник Бєлозьоров Г.С._____
(науковий ступінь, вчене звання, прізвище та ініціали, підпис)

Рецензент Савастру О.В._____
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
р.
№___ від «___»_____ 2021 р.
Завідувач кафедри

_____ _____
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК №
протокол № __ від «___» ____ 2021

Оцінка_____
Голова ЕК

_____ _____
(підпис) (прізвище, ініціали)

Одеса - 2021

АНОТАЦІЯ

Мета даної дипломної роботи:

1. Створити огляд проблеми пакування рюкзака з точки зору її криптографічного використання.
2. Дати крипто аналітичний огляд існуючих криптосистем ,та описати можливі атаки .
3. Дати оцінку криптографічной стійкості існуючих систем.
4. Створити програмну реалізацію системи з криптографічно значущими параметрами.

АННОТАЦИЯ

Цель данной дипломной работы:

1. Создать обзор проблемы упаковки рюкзака с точки зрения ее криптографического использования.
2. Дать крипто аналитический обзор существующих криптосистем, и описать возможные атаки.
3. Дать оценку криптографической устойчивости существующих систем.
4. Создать программную реализацию системы с криптографической значимыми параметрами.

ANNOTATION

The purpose of this thesis:

1. Provide an overview of the knapsack packaging problem from the point of view of its cryptographic use.
2. Give a crypto-analytical overview of existing cryptosystems, and describe possible attacks.
3. To assess the cryptographic stability of existing systems.
4. Create a software implementation of a system with cryptographically significant parameters.

ЗМІСТ

ВСТУП ……..6
1.КЛАСИЧНИЙ ПІДХІД РІШЕННЯ ЗАДАЧІ ПРО ЗАПОВНЕННЯ РЮКЗАКУ……...…….....7
2.ОГЛЯД ІСНУЮЧИХ КРИПТОСИСТЕМ ……....11
3.ВИБІР МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ……19
4.РАНЦЕВІ КРИПТОСИСТЕМИ……...20
5.МОЖЛИВІ АТАКИ РАНЦЕВИХ КРИПТОСИСТЕМ……...23
6.АЛГОРИТМ МЕРКЛА-ХЕЛМАНА……...27
7.ПРОГРАМНА РЕАЛІЗАЦІЯ КРИПТОСИСТЕМЫ МЕРКЛА-ХЕЛМАНА…….32
ВИСНОВОК……...35
ПЕРЕЛІК ВИКОРИСТАННИХ ДЖЕРЕЛ…….....36
ДОДАТОК А……37

ВСТУП

Перший алгоритм для шифрування на основі завдання про рюкзаку був розроблений Меркле і Хелманом в 1978 році і отримав назву алгоритм Меркле-Хелмана .Алгорітм міг бути використаний тільки для шифрування, але ізраїльський криптоаналитик Аді Шамір адаптував його для використання в цифрових підписах. Після опублікування схеми Меркле запропонував винагороду в 100 доларів тому, хто вдало здійснить злом одностадійного алгоритму. У 1982 році Шамір здійснив успішну атаку і отримав обіцяну винагороду. Але навіть після його сплати Меркле був упевнений вкриптостойкости мультістадійной системи і запропонував 1000 доларів в разі її успішного злому. У 1984 році американський математик Ернест Брікелл зумів здійснити злом для сорока-стадійного варіанти трохи більше ніж за 1 годину на машині Cray-1.

Незалежно один від одного ще в 1980 році американський математик Рон Грем і Аді Шамір виявили спосіб підвищити криптостойкость схеми Меркле-Хелмана, але вже в 1983 році отримана схема Грема-Шаміра була зламана американським вченим Леонардом Адлеманом. Однак пошуки модифікацій, позбавлених недоліків схеми Меркле-Хеллмана, продовжилися. У 1985 році британський ад'юнкт-професор Родні Гудман і американський інженер Ентоні Маколі створили схему, засновану на модульних рюкзаках з секретної лазівкою не на основі сверхвозрастающіх векторів, а з використанням китайської теореми про залишки.

Згодом стало ясно, що схема вразлива для атак, заснованих на пошуку секретних лазівок. Однак в 1990 році Валттері Ніємі запропонував нову схему на основі тієї ж завдання модульного рюкзака. Вона була зламана вже в наступному році Чи Е меном, Антуаном Жу і Жаком Штерном незалежно один від одного, злегка різними методами. Крім використання модульних рюкзаків були спроби застосування інших видів рюкзака. Так, в 1986 році Харальд Нідеррайтер опублікував рюкзачной криптосистему на основі алгебраїчної теорії кодування, яка в подальшому була зламана, а в 1988 році Масакацу Морії і Масао Касахара розробили криптосистему з використанням мультиплікативного рюкзака. Ця ідея виявилася вдалою і поки система на мультиплікативний рюкзаків не зламана.

1.КЛАСИЧНИЙ ПІДХІД РІШЕННЯ ЗАДАЧІ ПРО ЗАПОВНЕННЯ РЮКЗАКУ

Задача про пакування рюкзака - це одна з NP-повних задач комбінаторної оптимізації. Суть цієї задачі в тому, щоб укласти максимально більше цінних речей у рюкзак, не перебільшуючи максимальну вагу рюкзака. Теоретична задача про пакування рюкзака залишається актуальною та часто фігурує у проблемах різноманітних спрямованостей: її модифікації можна зустріти в криптографії, економіці, логістиці, генетиці; для пошуку оптимальних маршрутів, оптимального завантаження трюмів, складів, тощо.

Серед поширених методів вирішення задачі про пакування рюкзака можна відзначити: повний перебір, жадібний алгоритм, метод гілок та кордонів та генетичний алгоритм. Усі методи поділені на дві групи: методи з точним рішенням (повний перебір, метод гілок та кордонів) та методи з приближеним рішенням (генетичний та жадібний алгоритми).

1.1.NP-повні задачі

Задача про пакування рюкзака, як і задача, яка вирішується, є NP-повною[3]. На відміну від поліноміальних алгоритмів, виконання яких у найгіршому випадку потребує часу, де n – кількість вхідних даних, а k – константа, існують і такі, які за реальний час виконати неможливо, незалежно від обчислювальних потужностей – наприклад, відома проблема зупинки Тюрінга. Є також задачі, що, теоретично, можуть бути вирішені, але не за час . Прийнято казати, що проблеми, що є поліноміальними – легкі, а ті, що потребують суперполіноміальний об’єм часу – складні.

Одна з груп, що відноситься до «складних» або до «легких» проблем – «NP-повні» проблеми, характеристика яких досі залишається невизначеною.

Ще не винайдений поліноміальний алгоритм, що міг би вирішувати NP-повні проблеми, проте, не доведено, що задачі такого типу повинні вирішуватися неполіноміальними алгоритмами. Деякі задачі, що відносяться до NP-повних, мають дуже схожу структуру до тих проблем, що вже вирішуються поліноміальними алгоритмами, що дає підстави припустити, що дані NP-повні задачі також можуть бути вирішені відносно легко. Наприклад, задача про пошук найкоротшого та найдовшого шляхів. Хоча різниця здається зовсім несуттєвою, пошук найкоротшого шляху у

орієнтованому графі $S = (A, B), O(AB)$ є «легкою» задачею, а пошук найдовшого – «складною».

NP-повні задачі відносяться до одного сімейства з P (поліноміальні) та NP (неполіноміальні) задачами. Проблеми NP-класу вирішити за поліноміальний час неможна, але можна перевірити вже існуючі рішення за час, що залежить від об'єму вхідних даних. Слід зазначити, що задачі класу P відносяться також до класу NP, тому що будь-яке рішення, що може бути знайдене за поліноміальний час, так само може бути за нього перевірене. NP-повною проблема є тоді, коли вона відноситься до класу NP та є так само складною, як і будь-яка задача з даного класу. Рішення будь-якої задачі з NP або NP-повного класу за поліноміальний час буде показником того, що і всі інші NP-повні задачі можуть бути вирішені за поліноміальний час. Так само і для задачі пакування рюкзака не знайдено поліноміального алгоритму вирішення.

1.2.Класична постановка задачі про пакування рюкзака:

Нехай доступний певний набір предметів, кожен з яких має дві характеристики – цінність та вага. Також доступний рюкзак з певною місткістю. Потрібно запакувати рюкзак так, щоб всредеині знаходились лише предмети з максимальною цінністю, не перевищуючи місткість рюкзака[2]. Математична постановка задачі: Кожному предмету[i] задана його вага $p_i > 0$ $c_i > 0$, $i = 1, 2, 3\ldots n$. Максимальне обмеження на вагу рюкзака – P. Кожен x_i може приймати лише одне з двох значень: 0 – коли предмет не запакований, та 1 – коли запакований. Треба вибрати із заданої множини предметів набір з максимальною цінністю: $\sum_{i=1}^{n} c_i x_i$ при тому, що зберігається обмеження на сумарну вагу предметів рюкзака.

1.3.Різновиди задач про пакування рюкзака:

Умови відповідають за відмінності , накладених на рюкзак, предмети або їх вибір[1]:

1. Рюкзак 0-1 або класична задача - кожен предмет можна брати тільки один раз;
2. Необмежений рюкзак - кожен предмет можна брати скільки завгодно раз;
3. Обмежений рюкзак - кожен предмет можна брати певну кількість разів;
4. Безперервний рюкзак - можна взяти будь-яку дробову частину від предмета;

5. Рюкзак з мультивибором - є кілька класів предметів, з яких можна брати одного представника, причому деякі речі мають більший пріоритет, ніж інші;
6. Мультиплікативний рюкзак – є декілька класів предметів, з яких можна вибрати одного представника, причому деякі речі мають більший пріорітет, ніж інші;
7. Багатомірний рюкзак – є більше, ніж одне обмеження на рюкзак.

1.4.Алгоритми вирішення задачі

Практична задача про розміщення вантажів у контейнері найближче всього до класичної задачі, коли кожен предмет можна брати лише один раз та кількість предметів обмежена, але при цьому вона суттєво не покриває всі вимоги поставленої задачі.Сюди можна додати методи рішення(Таблиця 1.1 – Алгоритми вирішення задачі), що можна застосувати для всіх вищезазначених варіацій :

1. Повний перебір;
2. Метод гілок та кордонів;
3. Жадібний алгоритм;
4. Генетичний алгоритм;
5. Динамічне програмування.

Існує проблема вибору алгоритму рішення задачі про пакування рюкзака доводиться вибирати між точними алгоритмами, які не застосовні для рюкзаків великої розмірності, і наближеними, які працюють швидко, але не забезпечують оптимального рішення задачі. Вибір використання того чи іншого методу є спірним питанням. Все залежить від постановки задачі, а також від того, які цілі поставлені. Якщо потрібно знайти точне рішення, то необхідно використовувати точні методи. У ситуації невеликого набору вхідних даних (до 10-20 предметів), краще застосовувати методи перебору або гілок і меж, в силу простоти реалізації. Якщо ж точність рішення не грає великої ролі, або вхідні дані такі, що жоден з точних методів не працездатний, то для вирішення завдання можна використовувати лише наближені алгоритми рішення задачі.

Таблиця 1.1 – Алгоритми вирішення задачі

Метод	**Тип алгоритму**	**Переваги**	**Недоліки**
Повний перебір	Точний	Проста реалізація, точний результат	Невеликі вхідні дані, займає багато часу
Метод гілок та кордонів	Точний	Швидший за повний перебір	Невеликі вхідні дані, займає багато часу
Жадібний алгоритм	Наближений	Швидкість, об’єм, проста реалізація	Неточне, неефективне рішення
Генетичний алгоритм	Наближений	Швидкість, великі об’єми, формат даних	Не гарантує точність результату
Динамічне програмування	Точний	Не залежить від вхідних даних,	Об’єм обчислювальної роботи

2. ОГЛЯД ІСНУЮЧИХ КРИПТОСИСТЕМ

Так само хотілося б відзначити що алгоритм Меркле-Хелмана, був першим алгоритмом ,який використовує асиметричну схему шифрування, тому варто провести історичний аналіз того ,чому це так важливо.

Довгий час криптографія була прерогативою держави, а криптографічні алгоритми вважалися військовими технологіями. Поширення інформаційних технологій вимусило до необхідності інтеграції в них все більш стійких механізмів безпеки, що неможливо без використання надійних криптографічних алгоритмів і це призводить до того, що криптографія втрачає військовий статус. Внаслідок такої ситуації, до недавнього часу, з причини браку інформації було важко визначити ступінь надійності криптографічних алгоритмів, та, навіть, знайти їх описи. Це призводило до використання різних примітивних методів криптографічного захисту чи до створення абсолютно ненадійних алгоритмів.

На сьогоднішній день існує величезна кількість криптографічних алгоритмів, що відрізняються як своїми загальними характеристиками, так і принципами, на яких базується їх робота. Не всі вони є однаково надійними - серед них є навіть такі, що оформлені як стандарти та при цьому не забезпечують скільки-небудь реального захисту. Насправді ж, створення надійного криптографічного алгоритму - дуже важка задача. Крім того, надійність є відносна річ - багато з раніше розроблених алгоритмів, які вважалися надійними, тепер або ненадійні, або ця надійність викликає великий сумнів. Тому при розробці криптографічного алгоритму необхідно враховувати тенденції розвитку комп'ютерної техніки а також інші фактори, що потенційно можуть знизити його стійкість в майбутньому.

Традиційний метод шифрування, що застосовується при організації захисту даних, які пересилаються інформаційно-обчислювальними мережами, базується на факті знання і використання відправником та адресатом повідомлення одного й того ж секретного ключа. Ідея цього, так званого методу криптографії з секретним ключем або симетричної криптографії, полягає в тому, що відправник використовує секретний ключ для того, щоб зашифрувати текст повідомлення, а адресат застосовує цей же ключ для його розшифрування. У випадку компрометації ключа стає

можливим несанкціонований доступ до даних, тому для надійного функціонування системи необхідна його періодична заміна. Особливо важливим питанням є спосіб конфіденційного узгодження обома сторонами ключів до використання. Якщо відправник та адресат розділені значними відстанями в просторі, постає питання надійності засобів зв'язку, що ними пересилаються секретні дані: будь-хто, випадково або цілеспрямовано перехопивши секретний ключ, зможе безконтрольно читати, змінювати чи фальсифікувати всі супроводжувані цим ключем повідомлення. Сукупність заходів по створенню (генерації), передачі та зберіганню ключів носить назву адміністрування ключів (АК) і особливо важко забезпечується у відкритих системах з великою кількістю користувачів.

Для вирішення проблем, пов'язаних із АК, в 1976році Уітфілдом Діффі та Мартіном Хелманом була запроваджена концепція системи шифрування з відкритим ключем, яка виключає необхідність відправнику та адресату ділитись секретною інформацією. Кожна зі сторін має свою персональну пару ключів, пов'язаних між собою певною математичною залежністю: відкритий, що публікується і використовується для пересилання повідомлень, і секретний що його використовують для розшифрування прийнятих даних і зберігають у таємниці. Для пересилання конфіденційного повідомлення відправник шифрує його текст за допомогою відкритого ключа адресата, після чого відсилає за місцем призначення, а адресат розшифровує отриману інформацію, застосовуючи свій секретний ключ. Таким чином, будь-хто може прийняти "чуже" повідомлення, але тільки безпосередній адресат може його прочитати.

Генерація ключів в обох системах, як правило, здійснюється за допомогою генераторів псевдовипадкових чисел (ПВЧ), що повинні забезпечувати вихідну послідовність із достатньо довгим циклом повторювання. Це необхідно для виключення можливості появи повторюваних блоків вихідної послідовності, а відтак, її прогнозованості.
Одною з переваг системи шифрування з відкритим ключем є відсутність необхідності покладання на безпеку засобів комунікацій, оскільки між абонентами передаються лише відкриті ключі. Єдина вимога до системи полягає в підборі пари ключів таким чином, щоб секретний ключ неможливо було вирахувати з відкритого. Слід також взяти до уваги забезпечення системою можливості електронного підпису документів, коли адресатові надається додаткова можливість перевірки аутентичності

та цілісності одержаного повідомлення, тобто факту, що документ передано в незмінній формі саме від означеного відправника. Недоліком в порівнянні з системою шифрування з секретним ключем є недостатня швидкість функціонування методу.

Слід відзначити, що, попри доцільність окремого застосування кожного із методів при забезпеченні певних типів захисту інформації, в інформаційно-обчислювальних мережах часто застосовується комбінація обох систем для сполучення таких їх позитивних якостей, як швидкість обробки секретних ключів та безпека передачі відкритих. Реалізація такого протоколу "цифрового конверта" передбачає шифрування відкритим ключем секретного, який, в свою чергу, використовується для шифрування тексту повідомлення. Таким чином, обидва підходи при застосуванні в розподілених інформаційних середовищах є взаємодоповнюючими в забезпеченні конфіденційного обміну інформацією.

Алгоритми шифрування можуть бути потоковими чи блочними з довжиною блока 64 біта. Достатня довжина блоку є необхідною умовою для забезпечення високої криптостійкості алгоритму; вважається, що довжина блоку в 64 біта є достатньою.Найбільш простим і очевидним режимом застосування алгоритму шифрування є шифрування блоків відкритого тексту кожного окремо. Такий режим називається ECB (Electronic CodeBook mode - "режим електронної шифрувальної книги"). Але цей режим має багато недоліків і ніколи не повинен використовуватись для надійного шифрування. Одним із цих недоліків є той, що однаковим блокам відкритого тексту відповідають однакові блоки зашифрованого тексту.

Найбільш розповсюджений режим застосування алгоритму шифрування має назву CBC (CipherBlock Chaining - "зчеплення зашифрованих блоків"). В цьому режимі до кожного з блоків відкритого тексту, перед шифруванням, за модулем два додається попередній зашифрований блок. До першого блока відкритого тексту за модулем два додається випадковий блок, вектор ініціалізації, який передається у відкритому вигляді разом із зашифрованим текстом. Третім режимом застосування алгоритму шифрування є CFB (Cipher FeedBack - "шифрування зі зворотним зв'язком"). Тут кожний блок зашифрованого тексту утворюється шляхом додавання до блока відкритого тексту за модулем два результату шифрування попередньогоутвореного блока. Вектор ініціалізації використовується в якості нульового зашифрованого блока. Схема

побудови більшості, якщо не всіх алгоритмів шифрування є такою, що основа процесу шифрування уявляє собою послідовність приблизно однакових перетворень, яка повторюється декілька ітерацій. Більша кількість ітерацій може підвищувати криптостійкість, але все в основному залежить від перетворень, які повторюються в ітераціях.

В залежності від кількості ітерацій, характеру перетворень, які виконуються в симетричному алгоритмі шифрування, алгоритм може мати ту чи іншу швидкість роботи. Зрозуміло, не обов'язково, що алгоритм, який має меншу швидкість роботи, є більш криптостійким. Симетричні алгоритми шифрування можуть бути різними за своєю побудовою - існують оригінальні алгоритми, що сильно відрізняються від інших, та існують алгоритми, що мають одну з класичних схем побудови.

Одним із розповсюджених класів алгоритмів шифрування є так звані шифри Фейстеля.

В шифрі Фейстеля відкритий текст, що підлягає шифруванню, розділяється на дві половини. В ітераціях застосовується функція, першим аргументом якої є одна з цих половин, а другим аргументом -один із підключів, які певним чином отримуються з ключа шифрування; генерація підключів із ключа шифрування, як правило, може бути виконана один раз перед власно процесом шифрування. Результат функції додається за модулем 2 до іншої половини, після чого ці половини міняються місцями. За таким шаблоном виконуються всі ітерації, крім останньої, в якій зміна місцями лівої та правої половин не відбувається (з метою спрощення процесу дешифрування).

Головною властивістю шифру Фейстеля є те, що процеси шифрування та дешифрування є структурно ідентичними. Різниця полягає тільки в тому, що при дешифруванні підключі застосовуються в послідовності, оберненій до тієї, що використовувалась у процесі шифрування.

В багатьох симетричних алгоритмах шифрування використовується поняття S-комірки (S-Box - Substitution Box - комірка підстановки). S-комірка розмірністю уявляє собою таблицю розмірністю, яка задає відображення вхідних бітів на вихідних бітах. S-комірка підставляє (заміщує) вхідне значення на вихідне значення таким чином, що будь-яка зміна у вхідному значенні призводить до хаотичної зміни у вихідному значенні.

Взагалі кажучи, "ідеальний" блочний шифр (із конкретним ключем) може бути представлений як S-комірка, кількість вхідних і вихідних бітів

якої дорівнює довжині блока. Але оскільки при прийнятній довжині блока безпосередня практична побудова такої S-комірки неможлива, в сучасних блочних алгоритмах шифрування фактично використовуються різноманітні прийоми для імітації цієї комірки. При цьому, іноді в алгоритмі шифрування можуть навіть взагалі не застосовуватись S-комірки.

Розмірність і зміст S-комірок, які використовуються в алгоритмі, мають дуже високий вплив на його криптостійкість, в тому числі на його стійкість до лінійних і диференціальних атак. Зрозуміло, що S-комірка заданої розмірності може бути представлена як набір із булевих функцій, кожна з яких має задану кількість аргументів.

Іншою характеристикою S-комірки, що також визначає її надійність, є так звана таблиця розподілу різниць. Перед тим, як дати означення таблиці розподілу різниць, визначимо декілька термінів. При додаванні двох двійкових значень за модулем 2, якщо біти цих значень, що знаходяться на однакових позиціях, є однаковими, то відповідний біт результату приймає значення 0, а якщо різними, то значення 1. Вхідною різницею S-комірки будемо називати результат додавання за модулем 2 двох вхідних значень цієї S-комірки. Відповідно, вихідною різницею S-комірки будемо називати результат додавання за модулем 2 двох вихідних значень цієї S-комірки.'
Нехай X - вхідна різниця, Y - вихідна різниця. Тоді якщо для деякої S-комірки існує пара вхідних значень із різницею X, і існує пара відповідних їм вихідних значень із різницею Y, то будемо казати, що для даної S-комірки X може викликати Y. Якщо ж для заданої S-комірки та значень X і Y такої пари не існує, то будемо казати, що для даної S-комірки X не може викликати Y.

Таблиця розподілу різниць S-комірки має розмірність. її рядки відповідають всім можливим вхідним різницям, а стовпчики - всім можливим вихідним різницям. Елементом таблиці, що відповідає вхідній різниці X і вихідній різниці Y, є кількість пар вхідних значень із різницею X, яким відповідають пари вихідних значень із різницею Y. Таким чином, якщо X викликає Y, то елементом таблиці є число більше 0, а якщо Хне викликає Y, то елементом таблиці є 0.

Будемо казати, що для певної S-комірки X може викликати Y із ймовірністю P, якщо P є відношення кількості пар вхідних значень із різницею X до кількості пар відповідних вихідних значень із різницею Y. Важливість таблиці розподілу різниць полягає в тому, що вона дозволяє

знайти ймовірні пари вхідних і вихідних значень по вхідним і вихідним різницям.

Отже, знаючи лише різницю між двома вхідними значеннями та ' різницю між двома вихідними значеннями S-комірки, можна з певною ймовірністю визначити, які ці вхідні та вихідні значення.

Таблиця розподілу різниць, яка відповідає надійній S-комірці, повинна містити елементи, що не перевищують 2. Число 2 відповідає деякій парі значень (A,B) і двоістій парі (B,A).

До недавнього часу найбільш широко використовуваним алгоритмом шифрування був DES (DataEncryption Standard - Стандарт Шифрування Даних). Алгоритм розроблений у 1977 році Агентством Національної Безпеки США на основі алгоритму, створеного фірмою IBM, який мав довжину ключа 128 бітів. Алгоритм DES належить до класу шифрів Фейстеля із кількістю ітерацій 16, довжиною ключа 56 біта і довжиною блока 64 біта.

Хеш-функція - це перетворення, яке для вхідного значення повертає результат фіксованої довжини. Хеш-функції тільки з цією властивістю мають багато застосувань при обчисленнях, але для криптографічного використання хеш-функції, як правило, повинні мати декілька додаткових властивостей.

Базовими вимогами до криптографічних хеш-функцій є такі:

1. результат повинен обчислюватись для аргументу будь-якої довжини;
2. алгоритм обчислення результату повинен бути відносно простим;
3. хеш-функція повинна бути одного напрямку;
4. хеш-функція не повинна мати колізій.

Під тим, що хеш-функція повинна мати один напрямок, розуміється, що її важко обернути, тобто для заданого результату хеш-функції неможливо (тут і далі мається на увазі неможливість практичного обчислення) знайти хоча б один відповідний аргумент.

Якщо для аргументу деякої хеш-функції неможливо знайти інше значення її аргументу таке, що результат обчислення хеш-функції буде той же самий, то кажуть, що ця хеш-функція не має колізій в слабкому сенсі. Якщо для деякої хеш-функції неможливо знайти два різних значення аргументу, яким відповідає однаковий результат, то кажуть, що ця хеш-функція не має колізій у сильному сенсі.

Головні галузі застосування криптографічних хеш-функцій: ^перевірка цілісності повідомлень, ^цифровий підпис, ^цифрове маркування часу

повідомлень. Будь-яка зміна в повідомленні під час його передачі з дуже великою ймовірністю призведе до зміни хеш-функції від цього повідомлення. Алгоритми генерації цифрових підписів, як правило, мають досить малу швидкість роботи, тому цифрові підписи ефективніше генерувати не від самого повідомлення, а від його хеш-функції. Крім того, значення криптографічної хеш-функції від повідомлення може розповсюджуватись, не знижуючи конфіденційності самого повідомлення. Це важливо для технології цифрового маркування часу повідомлень, коли необхідно забезпечити отримування часових відміток повідомлення, не відкриваючи його змісту.

Побудова більшості алгоритмів генерації криптографічних хеш-функцій, як правило, базується на визначенні хеш-функції в термінах так званої функції компресії. Функція компресії перетворює вхідне значення фіксованої довжини на вихідне значення меншої фіксованої довжини. Якщо маємо функцію компресії, то відповідна хеш-функція може бути визначена як повторення застосувань функції компресії, доки не буде оброблене все повідомлення. Процес обробки повідомлення починається з того, що вхідне повідомлення довільної довжини розбивається на блоки, довжина яких визначається функцією компресії; перед цим повідомлення спеціальним чином вирівнюється на границю блока (із міркувань безпеки). Потім здійснюється послідовна обробка блоків, при якій вхідними даними для обробки кожного блоку є цей поточний блок і результат хешування попереднього блоку, а вихідним даним обробки останнього блоку - значення результату хеш-функції від повідомлення.

Алгоритм шифрування RSA. Алгоритм шифрування RSA відноситься до криптографічних систем із відкритим ключем. Криптосистеми з відкритим ключем (асиметричні криптосистеми) були розроблені в другій половині семидесятих років. В асиметричних криптосистемах процедури прямого і зворотнього криптоперетво-рення виконуються на різних ключах і не мають між собою очевидних і легковідсліджуваних зв 'язків, що дозволяють за одним ключем визначити інший. В такій схемі знання тільки ключа зашифрування не дозволяє розшифрувати повідомлення, тому він не є секретним елементом шифру і зазвичай публікується учасником обміну для того, щоб будь-хто бажаючий міг послати йому зашифроване повідомлення.

Принцип функціонування асиметричної криптосистеми полягає в наступному:

1. Користувач А генерує два ключі - відкритий (незасекречений) і секретний - і передає відкритий ключ по незахищеному каналу користувачу Б;
2. Користувач Б шифрує повідомлення, використовуючи відкритий ключ шифрування користувача А;
3. Користувач Б посилає зашифроване повідомлення користувачу А по незахищеному каналу;
4. Користувач А отримує зашифроване повідомлення і дешифрує його, використовуючи свій секретний ключ.

Пари (відкритий ключ; секретний ключ) обчислюються за допомогою спеціальних алгоритмів, причому жоден ключ не може бути виведений з другого.

Криптографічна система RSA (Rivest-Shamir-Adleman). Авторами алгоритму RSA, запропонованого в 1977 р., є Р.Рівест (Rivest), А.Шамір (Shamir) і А.Адлеман (Adleman). Надійність алгоритму базується на складності факторизації (розкладення на множники) великих чисел і складності обчислення дискретних алгоритмів знахождення x при відомих a, b і п.

3.ВИБІР МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ

Для вибору потрібного алгоритму необхідно орієнтуватися на наступні критерії:

1. Наскільки алгоритм ефективний для класичної задачі;
2. Наскільки легко алгоритм адаптувати;
3. Наскільки алгоритм ефективно реалізуємий;
4. Як можна покращити алгоритм евристиками;
5. Наскільки якісне рішення можливо отримати в теорії.

Метод повного перебору є ресурсномістким. Він дає можливість отримати точні рішення, однак даний метод важко застосовувати при великих об’ємах вхідних даних. Проте, метод повного перебору є єдиним, що охоплює повну множину можливих рішень, що робить його ідеальним базовим методом та основою для проведення порівнялного аналізу з іншими методами.

Жадібний алгоритм добре розв'язує деякі задачі, а інші — ні. Більшість задач, для яких він спрацьовує добре, мають дві властивості: по-перше, до них можливо застосувати принцип жадібного вибору, по-друге, вони мають властивість оптимальної підструктури.

Динамічне програмування має такі самі вади, як і попередній метод: проблематична інтеграція додаткових обмежень, громіздкість реалізації, великі витрати ресурсних об’ємів.

Генетичні алгоритми теоретично мають високу швидкість роботи навіть при великих об’ємах вхідних даних, можуть швидко конфігуруватись, працюють з будь-яким форматом вхідних даних, якщо їх можна формалізувати до потрібного виду. Недоліків генетичних алгоритмів – локальних оптимумів – можна уникнути при правильній початковій конфігурації мутацій, а гнучкість функції пристосування дозволяє додавати стільки обмежень, скільки потрібно.

Метод гілок та границь хоч і дозволяє значно пришвидшити пошук рішення, зустрічається з проблемою специфічних обмежень. Його важко адаптувати під задачу розміщення та впровадити додаткові обмеження. Навіть якщо це вдасться, не можна гарантувати оптимальність рішення за даним методом, тому що є ризик зітнутися з локальним оптимумом або з неоптимальним рішенням загалом. Саме тому у якості базового алгоритму для заповнення рюкзака при дешифруванні повідомлення був обран жадібний алгорітм.Так як в реалізованої криптосистеме завдання ранця спрощено ,воно не має вартості

4. РАНЦЕВІ КРИПТОСИСТЕМИ

Одна з перших асиметричних криптосистем це криптосистема Меркла-Хелмана[5].В основі алгоритму Меркла-Хелмана лежить ідея шифрувати повідомлення на основі ключа – послідовності ваг задачі про рюкзак (відкритий ключ). Предмети з набору обираються за допомогою блоку відкритого тексту, рівного за довжиною кількості предметів у наборі (біти відкритого тексту відповідають значенням b i , i = 1, n), а шифротекст є отриманою сумою.

Виявилось, що ця схема є криптографічно нестабільною. Як суттєве покращення базового алгоритму Меркла-Хелмана було запропоновано створення закритого ключа, який є перетвореною послідовністю ваг задачі про рюкзак спеціального вигляду. Цей варіант використовувався у двох модифікаціях: одноетапній та багатоетапній. Але запропоновані вдосконалення не забезпечили криптостійскості алгоритму. Вперше про його небезпеку було повідомлено у роботі . Більш того, схема алгоритму дозволяє визначити вхідну послідовність без використання будь-якого закритого ключа . Тому зрозуміло, що практично одразу після створення розпочався пошук модифікацій запропонованого алгоритму, що забезпечують підвищений захист від зламу. Для подолання недоліків базової схеми Родні Гудман та Ентоні Маколі розробили процедуру, що базується на модульних рюкзаках. Надалі з’ясувалося, що ця схема також небезпечна. Окрім використання модульних рюкзаків, були запропоновані схеми використання інших видів рюкзака.

У 1986 р. Харальд Нідерайтер опубліковав рюкзачну криптосистему на основі алгебраїчної теорії кодування, яка також була зламана, а у 1988 р. Масакацу Морії та Масао Касахара розробили криптосистему з використанням мультиплікативного рюкзака. Ця ідея виявилася вдалою і поки що система на мультиплікативних рюкзаках не зламана. Також вдалою виявилась ідея Хусейна Алі Хусейна, Джафара Ваді Абдули Сада та М. Каліфа , які у 1991 р. запропонували багатоетапну рюкзачну криптосистему. У ній фіксується рюкзачний вектор на кожному етапі, а вихід (зашифроване повідомлення) після кожної стадії алгоритму використовується як вхідні дані (текст) на наступному етапі. Про вдалу атаку на цю схему на поточний момент не відомо. Криптосистема Чор-Рівеста є однією з небагатьох рюкзачних систем, які не були зламані, і є однією з найпривабливіших.

Продовжено покращення і класичного алгоритму Меркла-Хелмана.

Алгоритм криптосистеми полягає в наступному, відкритим ключем є послідовність з n натуральних чисел a1, a2, a3, ... an.Сторона A для передачі повідомлення в бітовому вигляді x1, x2, x3,. . . xn, xi ∈ {0,1} обчислює суму відповідних творів $sum = \sum_{i=1}^{n} a_i x_i$ (Можна вважати, виконується скалярний добуток векторів) і по-силает по відкритому каналу результат. Для розшифровки повідомлення сторона У повинна вирішити зворотну задачу, відому, як завдання про ранці: за сумою ваг деяких об'єктів, а також за вагами всіх об'єктів визначити об'єкти, які беруть участь в сумі. У загальному вигляді дана задача є NP-повною, а при деяких наборах ваг відповідей може бути навіть кілька. (Наприклад, під час написання ваг {2, 4, 7}). Ввипадку, коли для кожного 1 <i ⩽ n виконується суворе нерівність $\sum_{j=1}^{i-1} a'_j < a'_i$ (Послідовність a'1, a'2, a'3,... A'n в такому випадку називається суперзростаючої),то існує таке рішення, працююче за поліноміальний час. Спочатку инициализируется змінна sum ', рівна отриманої по каналу зв'язку сумі. В ході алгоритму з неї будуть відніматися ваги, відповідні одиницям в дешіфрованному повідомленні. Розглядаються ваги a'[i] в порядку убування, в тому випадку, коли поточна сума sum 'менше, ніж a'i, то, очевидно, i-й об'єкт не входить у відповідь. Якщо ж сума sum 'більше або дорівнює поточного ваги a'[i], то він зобов'язаний входити у відповідь, інакше $\sum_{j=1}^{i-1} a'_j < a'_i \leqslant sum'$, і навіть всі об'єкти, що залишилися не забезпечать потрібну суму. Для переходу до наступного вазі потрібно додати поточний об'єкт у відповідь і відняти від суми поточний вагу.

Для несанкціонованої боку З рішення даної задачі повинно бути сформульовано в загальному вигляді. Для цього випадовим чином беруться два числа: $M > \sum_{i=1}^{n} a'_i$ и $W : (W, M) = 1$,а також число W^{-1} ,є зворотним до W по модулю M. При заміні $a_i = a'_i W^{-1} \bmod M$ ваги a[i] виявляються рівномірно розподіленими по модулю M, а операції на стороні A практично не ускладнюються (просто змінюються використовувані ваги, причому не змінюється порядок максимальной ваги, якими оцінюється складність алгоритму дешифрування). На стороні B отримана сума sum множиться на W по модулю M, в результаті виходить

$sum = sumW \bmod M = \sum_{i=1}^{n} a_i' W^{-1} W \bmod M = \sum_{i=1}^{n} a_i'$ Таким чином за допомогою секретного ключа завдання велено до полиномиальной. При реалізації, крім домноження на W^{-1} використовується також перестановка ваг, при розшифровці застосовується зворотна.

Залишається вказати спосіб побудови чисел W і M, а також спосіб побудови супер зростаючої послідовності. Нехай необхідно побудувати ключі для деякої кількості об'єктів n. Вага a'1 вибіра- ється випадковим чином з рівномірного розподілу цілих чисел по відрізку $[1, 2^n]$. Наступний вага a'2 вибирається рівномірно з відрізка $[1 * 2^n + 1, 2 * 2^n]$, Наступний за ним a'3 вибирається з $[3 * 2^n + 1, 4 * 2^n]$ і так далі; a'i вибирається з відрізка $[(2^{i-1} - 1) * 2^n + 1, 2^{i-1} * 2^n]$.

При таких розподілах умова супер зростання, очевидно, буде така щоб нижня межа кожного відрізка в точності дорівнює збільшеною на одиницю сумі всіх правих меж попередніх відрізків. Число M при цьому вибирається з відрізка $[2^{2n+1} + 1, 2^{2n+2} - 1]$, А число W вибирається випадково з відрізка [2, M - 2] до тих пір, поки не буде взаємно просто з M.

5.МОЖЛИВІ АТАКИ РАНЦЕВИХ КРИПТОСИСТЕМ

У цьому розділі показано, як зламати основну криптосистему рюкзака Меркла-Хелмана .Ця атака стала першим серйозним криптоаналітичним нападом на рюкзаки, і це призвело до зламу численних інших систем рюкзаків. Це пов'язано з А. Шаміром [7].

Багато з нижченаведеного описано в Л. М. Адлеман [6].Перш ніж описувати атаку Шаміра, ми скажемо кілька слів про вибір параметрів (3.1). Якщо деякі з b_i а отже, також M, великі, тоді рюкзак буде неефективним, оскільки n бітів інформації буде закодовано приблизно $\log_2$ M біт.З іншого боку, небезпечно дозволяти будь-якому з b_j бути замалим.Якби ми мали $b_1 = 1$, скажімо, тоді $a_j = W$ для деякого j, і оскільки рюкзак можна показати неломливим, якщо виявлено W або M, можна спробувати кожен з a_j як можливий W і, таким чином, скомпрометувати система. Тому розумним компромісом між ефективністю та безпекою, здавалося б, був вибір параметрів у (3.1). Можна вибрати $b_i = c2^{i-1}$, $1 \le i \le n$, для деяких $c \approx 2^n$. Цей вибір задовольняв би (3.1), але дав би дуже невпевнену систему. Причина в тому, що для $1 \le j \le n - 1$ ми б мали

$$a'_{j+1} = 2a'_j \text{ або } 2a'_j - M,$$

кожен випадок зазвичай трапляється приблизно n/2 рази. Отже, прорахувавши

$$a_i - 2a_j, \ 1 \le i, j \le n,$$

отримали б приблизно n / 2 випадків - M серед n^2 відмінностей, щоб ми могли вивести, що таке M, і це зламало б систему. Також були запропоновані інші підходи. Наприклад, можна взяти $b_i = 2^{i-1}$, але приховати цю структуру подвійною ітерацією методу модульного множення. Однак ця конструкція, яка є більш безпечною, ніж те, що було представлено вище, автором було продемонстровано небезпечно (не опубліковано) використанням продовжених дробів. Мораль, яку слід взяти з наведених вище прикладів, полягає в тому, що легко побудувати ранцеві

криптосистеми, які здаються привабливими, але при цьому небезпечними. Це пояснює загалом високий рівень підозр щодо експертів щодо рюкзаків з моменту їх пропозиції.

$$b_1 \approx 2^n,\ b_j > \sum_{i=1}^{j-1} b_i,\ 2 \le j \le n,\ b_n \approx 2^{2n} \quad (3.1)$$

$$a'_j \equiv b_j W (mod\ M),\ 0 < a'_j < M \quad (3.2)$$

$$a_j = a'_{\pi(j)},\ 1 \le j \le n \quad (3.3)$$

Тепер ми перейдемо до опису нападу Шаміра на основну систему Меркле-Хеллмана. Будемо вважати, що секретні ваги рюкзака $b_1, \ldots, b_n$ задовольняють (3.1),і що вони трансформуються у загальнодоступні ваги $a_1, \ldots, a_n$ за допомогою (3.2) та (3.3). Нехай $U \equiv W^{-1} (mod\ M)$, 0 <U <M. За (3.2) маємо

$$a_j \equiv b_{\pi(j)} W\ (mod\ M) \quad (3.4)$$

або

$$b_{\pi(j)} \equiv a_j U\ (mod\ M) \quad (3.5)$$

а це означає, що для деякого цілого числа k_j,

$$a_j U - k_j M = b_{\pi(j)} \quad (3.6)$$

Отже

$$\frac{U}{M} - \frac{k_j}{a_j} = \frac{b_{\pi(j)}}{a_j M} \quad (3.7)$$

Це означає, що всі k_j / a_j близькі до U / M. Криптоаналітик не знає U, M, π, k_j або b_j, але знає a_j. Проблема зараз полягає у вилученні деякої інформації про всі ці невідомі з наведеного зауваження щодо (3.7).

З (3.1) ми бачимо, наприклад, що

$$b_1, b_2, \ldots, b_5 \le 2^n \quad (3.8)$$

Тому, якщо ми дозволимо $j_i = \pi^{-1}(i)$, то отримуємо

$$\left| \frac{U}{M} - \frac{k_{j_i}}{a_{j_i}} \right| \le \frac{2^n}{2^{4n}} = 2^{-3n},\ 1 \le i \le 5 \quad (3.9)$$

і, отже, віднімаючи i = 1 доданок від інших,

$$\left| \frac{k_{j_i}}{a_{j_i}} - \frac{k_{j_i}}{a_{j_i}} \right| \le 2^{-3n},\ 2 \le i \le 5 \quad (3.10)$$

Це, в свою чергу, означає це

$$\left| k_{j_i} a_{j_i} - k_{j_i} a_{j_i} \right| \le 2^n \ , \ 2 \le i \le 5 \ (3.11)$$

Нерівності (3.11) показують, наскільки незвичайними є a_{j_i} та k_{j_i}. Зрештою, кожен з них має порядок 2^{2n}, тому $a_{j_i} k_{j_i}$ знаходиться в порядку 2^{4n}. Щоб різниця двох таких термінів була на порядок 2^n вимагає якоїсь дуже особливої структури. Це не може траплятися часто, і (в більшості випадків) k_{j_i}, $1 \le i \le 5$, однозначно визначаються (3.11). (Пам'ятайте, що a_{j_i} є загальнодоступними.) Питання: Як ми визначаємо k_{j_i} на практиці? Головним внеском Шаміра було помітити, що це можна було зробити за поліноміальний час, використовуючи теорему Х. В. Ленстри про те, що цілочисельна задача програмування у фіксованій кількості змінних може бути вирішена за поліноміальний час . Це дає k_{j_i}, $1 \le i \le 5$. Як тільки k_{j_i} буде знайдено, отримують наближення до U / M з (3.9), і це дозволяє побудувати пару (U ′, M ′) з U ′ / M ′, близьку до U / M, таку, щоб ваги c_j, отримані у

$$c_j \equiv a_j U' \left(mod \ M' \right) \ , \ 0 < c_j < M' \ , \ 1 \le j \le n \ (3.12)$$

утворюють суперзростаючу послідовність (після того, як вони розташовані у зростаючому порядку). Зауважте, що ця атака не відновлює U та M, шо були сконструйовані оригінальним конструктором системи. Існує нескінченно багато пар U ′, M ′, які дають суперзростаючу послідовність c_j, і будь-яку з них можна використовувати для дешифрування повідомлень.

Одним із моментів, який було розкрито у вищезазначеному обговоренні, є пошук $j_{1,\ldots,}j_5$. Врешті-решт, перестановка π є таємною, тож як може криптоаналітик бути впевненим, що вона вибрала правильні індекси? Відповідь на це проста; криптоаналітик розглядає всі можливі варіанти $j_{1,\ldots,}j_5$, і оскільки їх існує лише на порядок n^5, це зберігає поліном часу роботи. (На практиці можна було б застосувати інший підхід, обираючи одночасно більше 5 ваг, але не вимагаючи, щоб вони походили з 5 найменших секретних ваг, а, скажімо, з половини найменших секретних ваг.)

Найважливішим інструментом, який досяг успіху вищезазначеної атаки, був результат Ленстри щодо цілочисельного програмування у фіксованій кількості змінних. Він був застосований для вирішення (3.9),

що є типовою проблемою в неоднорідному наближенні діофантину. У випадку базового рюкзака Меркле-Хеллмана з параметрами, заданими в (3.1), виявляється, що атака може бути здійснена також із використанням неперервних дробів, набагато простішого (і ефективнішого) інструменту, ніж алгоритм Ленстри . Продовжувані фракції, хоча і дуже корисні, мають, на жаль, лише обмежене застосування до рюкзаків. Навіть для основного рюкзака Меркле-Геллмана з b і більшим, ніж зазначено в (3.1) (скажімо, $b_1 \approx 2^{2n}, b_n \approx 2^{3n}$), метод продовження дробу не вдається. Результат Ленстра є потужним, але представляє переважно теоретичний інтерес, оскільки час його роботи задається поліномом високого ступеня, і тому він ніколи не був реалізований. Однак інший інструмент - зменшення основи ґратки Лови алгоритм, який був мотивований методом Ленстри, незабаром був застосований до рюкзаків і мав вирішальне значення для розвитку атак низької щільності, а також для злому інших систем.

6.АЛГОРИТМ МЕРКЛА-ХЕЛМАНА

4.1.**Надзростаючі рюкзаки.**

Означення: надзростаючою послідовністю називається числова послідовність, кожний член якої більший суми усіх попередніх членів. Наприклад , числова послідовність {1,3,6,13,27,52} є надзростаючою, а послідовність {1,3,4,9,15,25} – ні. В цьому контексті варто згадати і добре відому послідовність чисел Фібоначчі, яка, незважаючи на швидке зростання елементів, не є надзростаючою. Розглянемо просту проблему рюкзака. Якщо перелік ваг предметів у наборі є надзростаючою послідовністю, то отриману проблему рюкзака можна легко вирішити. Розв'язок надзростаючого рюкзака здійснюється наступним чином. Необхідно взяти повну вагу і порівняти її з найбільшим числом послідовності. Якщо повна вага менша цього числа, то його не кладуть у рюкзак. Якщо повна вага більша або дорівнює цьому числу, то воно кладеться у рюкзак. Зменшимо вагу рюкзака на це значення і перейдемо до наступного за величиною числа послідовності. Будемо повторювати ці дії, доки процес не завершиться. Якщо повна вага зменшується до нуля, то розв'язок знайдено. У протилежному випадку – ні.

Покладемо для прикладу , що повна вага рюкзака має бути 70, а надзростаюча послідовність ваг {2,3,6,13,27,52}. Найбільша вага – 52, яка менша 70, кладеться у рюкзак. Віднімаючи 52 від 70, отримуємо 18. Наступна вага – 27, більша 18, тому число 27 у рюкзак не кладуть. Наступну вагу 13, яка менше 18, кладуть у рюкзак. Віднімаємо 13 з 18 і отримуємо 5. Чергова вага – 6, більша за 5, не кладеться у рюкзак. Продовжуючи цей процес, отримаємо, що ваги 3 і 2 кладуть у рюкзак, і повна вага зменшується до 0, що свідчить про знайдений розв'язок. Якщо розглядати цю процедуру як блок шифрування методом рюкзака Меркла-Хелмана, відкритий текст, отриманий із значення шифротексту 70, був би рівний 110101. Пошук заповнення нормальних рюкзаків, які не є надзростаючими, являє собою складну проблему. Швидкого алгоритму для вирішення цієї задачі в реальному часі поки не знайдено. Єдиним відомим способом визначити, які предмети упаковано в рюкзаку, є методична перевірка можливих розв'язків до знаходження правильного. Найшвидший алгоритм, беручи до уваги різні евристики, має експоненційну залежність від кількості можливих предметів. Тому, якщо додати до послідовності ваг лише один елемент, знаходження розв'язку задачі стає вдвічі складнішим.

Це набагато важче надзростаючого рюкзака, в якому, якщо додати один предмет до послідовності, складність пошуку розв’язку зростає на одну операцію.Алгоритм Меркла-Хелмана базується на цій властивості. Закритий ключ є послідовністю ваг задачі надзростаючого рюкзака. Відкритий ключ – це послідовність ваг проблеми нормального рюкзака з тим самим розв’язком. Р. Меркл та М. Хелман , застосовуючи цілочислову арифметику, розробили спосіб перетворення проблеми надзростаючого рюкзака в проблему нормального рюкзака.

Підсумовуючи, відзначимо, що існує дві різні задачі, одна з яких вирішується за лінійний час,а інша, як вважається, – ні. Просту задачу можна перетворити у складну. Відкритий ключ являє собою складну (важку) проблему, яку дуже просто можна використати для шифрування, але неможливо – для розшифрування повідомлень. Закритий ключ є простою (легкою) проблемою, що надає простий спосіб розшифрування повідомлення. Якщо невідомий закритий ключ, потрібно розв’язувати складну задачу про рюкзак.

4.2.**Створення відкритого ключа.**

Розглянемо роботу алгоритму, не заглиблюючись у теорію чисел. Для отримання нормальної послідовності рюкзака візьмемо надзростаючу послідовність рюкзака, наприклад наведену раніше K ={2,3,6,13,27,52}, і домножимо всі значення на число T за модулем N . Значення модуля повинне бути більшим суми всіх чисел послідовності, тобто утворювати з заданою послідовністю надзростаючу послідовність. Оберемо, наприклад, N =105. Множник T повинен бути взаємно простим числом з модулем N , тобто НСД (N ,T) = 1. Покладемо, наприклад, T = 31. Нормальною послідовністю рюкзака у цьому випадку буде {62,93,81,88,102,37}, де 62 = 2*31 mod 105;

93 = 3*31 mod 105; 81 = 6*31 mod 105; 88 = 13*31 mod 105; 102 = 27*31 mod 105; 37 = 52*31 mod 105.

Надзростаюча послідовність рюкзака разом з числами N та T {2,3,6,13,27,52; 105, 31} є закритим ключем, а нормальна послідовність рюкзака {62,93,81,88,102,37} – відкритим.

4.3.**Шифрування.**

Для шифрування повідомлення розбивається на блоки, що за довжиною дорівнюють кількості елементів послідовності рюкзака. Далі, вважаючи, що одиниця відповідає наявності члена послідовності, а нуль – його відсутності, обчислюємо повні ваги рюкзаків – по одному для кожного

блоку повідомлень.Якщо припустити, що повідомлення у бінарному вигляді подається як 0110001101011101110, шифрування, яке використовує попередню послідовність рюкзака, буде здійснюватися таким чином: вхідне повідомлення у блочному вигляді = 011000 110101 101110, звідки 011000 відповідає числу 93 + 81 = 174; 110101 відповідає числу 62 + 93 + 88 + 37= = 280; 101110 відповідає числу 62+81 + 88 + 102= = 333. У результаті шифротекстом повідомлення 0110001101011101110 є числова послідовність 174,280,333.

4.4.**Розшифрування.**

Законний отримувач даного повідомлення знає закритий ключ: оригінальну надзростаючу послідовність, а також значення N та T , які використовувалися для перетворення її в нормальну послідовність рюкзака. Для розшифрування повідомлення отримувач визначає мультиплікативне обернене T -1 , таке що T *(T -1) (mod N) = 1. Кожне значення щифротексту домножується на T -1 mod N , а потім розгортається в суму за допомогою закритого ключа, щоб отримати значення відкритого тексту. Для наведеного вище прикладу надзростаючою послідовністю є {2,3,6,13,27,52}, N = 105 , T = 31. Шифротекстом буде послідовність 174,280,333. У цьому випадку T -1 дорівнює 61 (31*61mod105=1891mod105=1), тому значення шифротексту домножуються на величину 61mod105. Таким чином, маємо 174*61 mod 105 = 9 = 3 + 6, що відповідає 011000; 280*61 mod 105 = 70 = 2 + 3 + 13 + 52, що відповідає 110101; 333*61 mod 105 = 48 = 2 + 6 + 13 + 27, що відповідає 101110.Розшифрованим відкритим текстом є бінарні послідовності 011000 110101 101110, що повністю відповідає вхідному повідомленню.

4.5.**Демонстрація прикладу роботи алгоритму**

Спочатку ми генеруем випадкову надзростаючу послідовність вона і буде секретним ключем, потім взаємно прості числа Q і R, на їх основі створюється відкритий ключ(Рисунок 4.1 - алгоритм криптосистеми Мерлка-Хелмана.).Приклад генерації ключів(рисунок 4.2 - Створення секретного і відкритого ключа.)

Для шифрування ми розбиваємо двійкове повідомлення на блоки довжиною масиву відкритого ключа і вважаємо суму елементів ,де кожна 1 відповідає елементу відкритого ключа.Приклад шифрування (рисунок 4.3 - Шифрування і розшифрування повідомлення.)

Для розшифрування ми знаходимо x a після за допомогою жодного алгоритму вирішуємо завдання ранця використовуючи закритий ключ як набір предметів a x як його максимальна вага(рисунок 4.3 - Шифрування і розшифрування повідомлення).

1) Генерація ключів

$$A=\{a_i,\ldots a_n\} \quad a_i>\sum_{j=1}^{i-1} a_j \quad B=\{b_1,\ldots b_n\} \quad b_i=a_i\cdot R \bmod Q$$

НСП(Q,R)=1 $\quad Q>\sum_{i=1}^{n} a_i$

2) Шифрування $\quad m=m_i,\ldots m_n \quad m\in\{0,1\}$

$$c=\sum_{i=1}^{n} m_i b_i \bmod Q$$

3) Розшифровка $\quad x=c\cdot R^{-1} \bmod Q \quad x=\sum_{i=1}^{n} m_i a_i \rightarrow m_1\ldots m_n$

Рисунок 4.1 - алгоритм криптосистеми Мерлка-Хелмана.

1) Генерація ключів

A={3,5,10,25,50,90}; Q=189>183; R=10 ;(Q,R)=взаємно прості;

B={30,50,100,61,122,144}

$$b_1=3\cdot 10 \bmod 189 = 30$$
$$b_2=5\cdot 10 \bmod 189 = 50$$
$$b_3=10\cdot 10 \bmod 189 = 100$$
$$b_4=25\cdot 10 \bmod 189 = 61$$
$$b_5=50\cdot 10 \bmod 189 = 122$$
$$b_6=90\cdot 10 \bmod 189 = 144$$

(A,R,Q) - секретний ключ

(B) - відкритий ключ

Рисунок 4.2 - Створення секретного і відкритого ключа.

2) Шифрування

у нас є повідомлення m=100111 ;Q=189; $B = \{30_1, 50_0, 100_0, 61_1, 122_1, 144_1\}$

c=(30,61,122,144)mod189=357mod189=168

3)Розшифрування

$x = 168 \cdot 10^{-1} mod 189 = 168 \cdot 19 mod 189 = 3192 mod 189 = 168$

$A = \{3_1, 5_0, 10_0, 25_1, 50_1, 90_{1\leftarrow}\}$ $168 - 90 = 78 - 50 = 28 - 25 = 3 - 3 = 0$

$(10^{-1} mod 189)$ Використовуючи розширений Алгоритм Евкліда це 19

m=100111

Рисунок 4.3 - Шифрування і розшифрування повідомлення.

7.ПРОГРАМНА РЕАЛІЗАЦІЯ КРИПТОСИСТЕМЫ МЕРКЛА-ХЕЛМАНА

Для програмної реалізації була вибрана мова програмування C#.Код програми і структуру проекту можна подивитися в Додатку А.

5.1.**Опис інтерфейсу.**

В інтерфейсі програми (рисунок 5.1.інтерфейс програми) є поля секретного і публічного ключів, де ми можемо побачити ці ключі при генерації.Також є поля повідомлення та шіфтекста в яких ми можемо побачити шифрування і дешифрування повідомлення.Маніпілуровать інтерфейсом ми можемо за допомогою кнопки Generate key яка створює пару ключів, кнопок Encrypt и Decrypt которые шифруют и дешифруют сообщение и шифртекст відповідно.

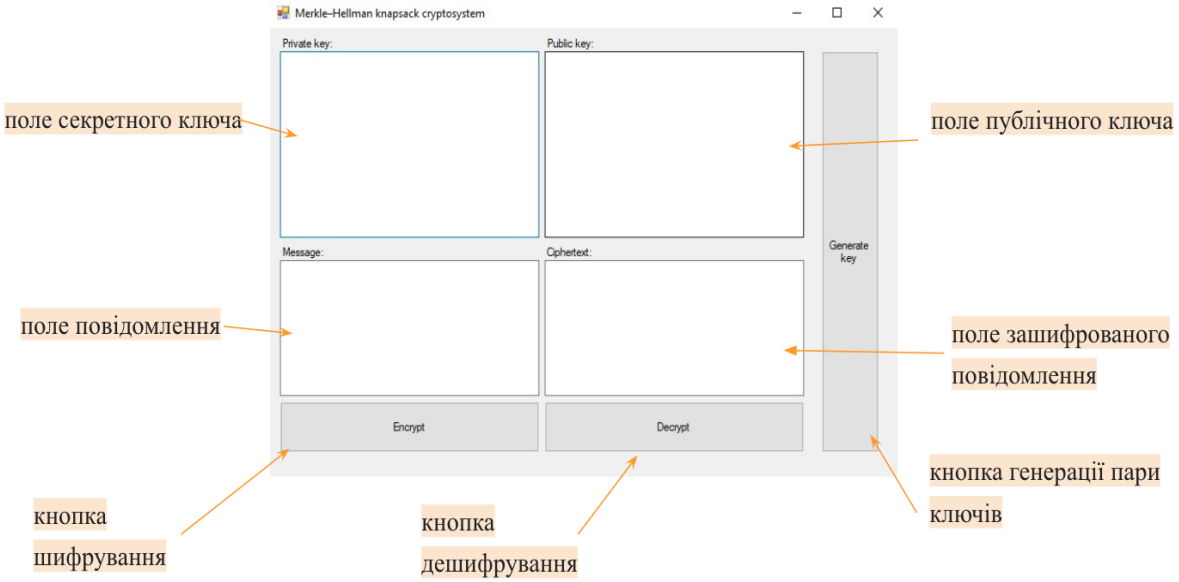

Рисунок 5.1.інтерфейс програми

5.2.**Експлуатація програми.**

Для генерації ключів потрібно натиснути кнопку Generate key і ми отримаємо секретний ключ, числа R і Q у полі Private key, відкритий ключ у полі Public key (рисунок 5.2.створення пари ключів).

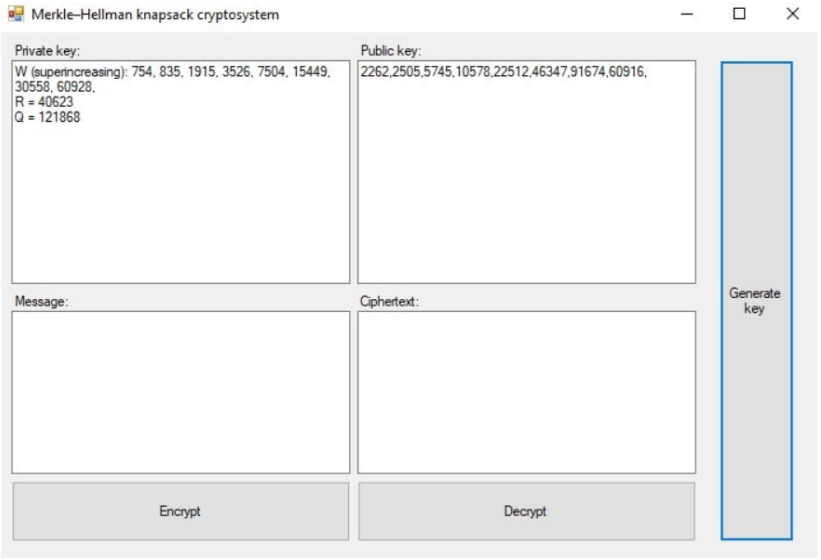

Рисунок 5.2.створення пари ключів

Після чого вводимо наше повідомлення для шифрування в поле Message і натискаємо кнопку Encrypt (рисунок 5.3.введення повідомлення). У полі Ciphertext ми отримуємо шифр текст нашого повідомлення (рисунок 5.4. шифрування повідомлення).

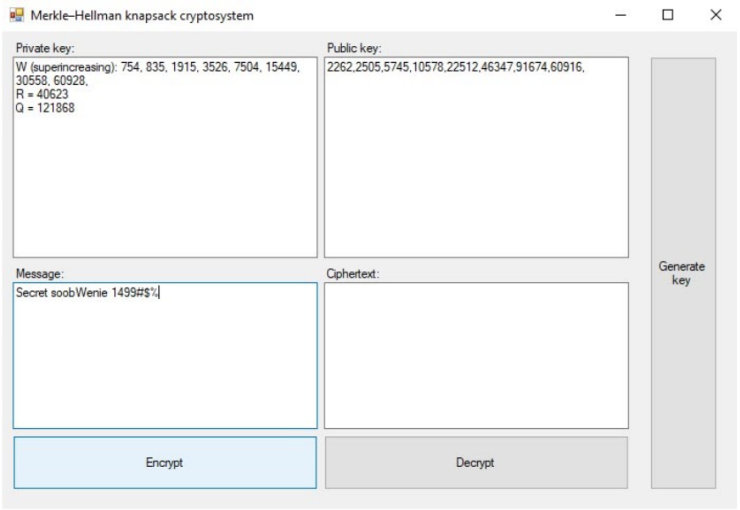

Рисунок 5.3.введення повідомлення

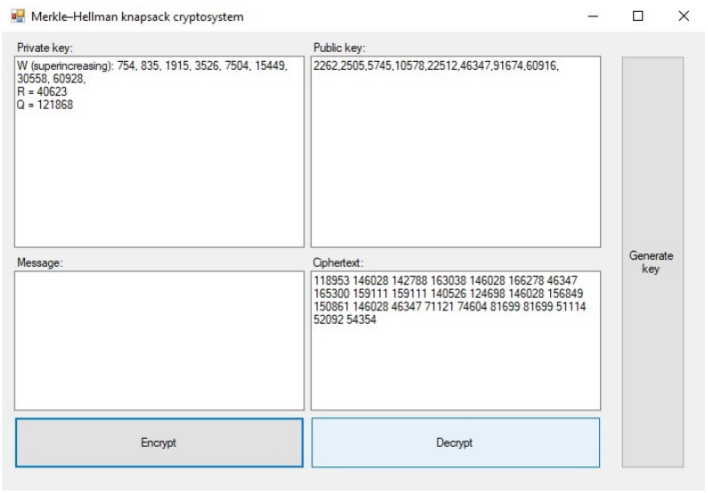

Рисунок 5.4. шифрування повідомлення

Для дешіфрування шифртекста потрібно натиснути кнопку Decrypt, після чого у вікні message ми отримаємо наше повідомлення(рисунок 5.5.дешифрування повідомлення)

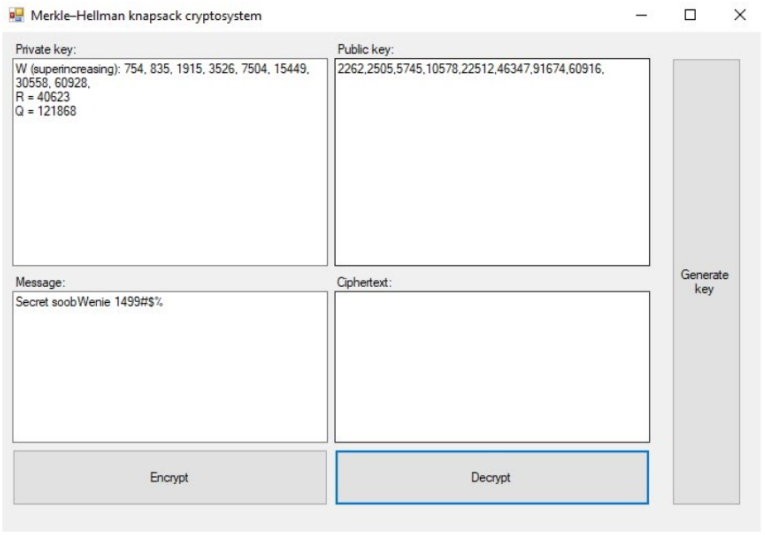

Рисунок 5.5.дешифрування повідомлення

ВИСНОВОК

У даній дипломній роботі був створен огляд проблеми пакування рюкзказу з точки зору її криптографічного використання.

Зроблено аналітичний огляд існуючих криптосистем ,та описані можливі атаки на ці криптосистеми, а зокрема атака Шаміра на криптосистему Меркла-Хелмана.Також програмно реалізована Ранцева криптосистема Меркле Хеллмана.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Берник В.И. Харин Ю.С. и др. – Математические и компьютерные основы криптологии.– Минск, ООО «Новое знание», 2003. 140–142с.
2. Коблиц Н. – Курс теории чисел и криптографии. – М. ТВП, 2001.78–81с.
3. Саломаа А. – Криптография с открытым ключом. – М. Мир, 1995. 101–112с.
4. Тилборг Х.К.А. Основы криптологии. – М. Мир, 2006. 302–304с.
5. R. C. Merkle and M. E. Hellman –"Hiding Information and Signatures in Trapdoor Knapsacks –" *IEEE Trans. Inform. Theory*, vol. 24, 1978,–525с.
6.L. M. Adleman –"On Breaking Generalized Knapsack Public Key Cryptosystems," *Proc. of the Fifteenth ACM Symposium on Theory of Computing, 1983*, 408–412с.
7. Adi Shamir. – A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem – CRYPTO-1982,284–288с.

ДОДАТОК А

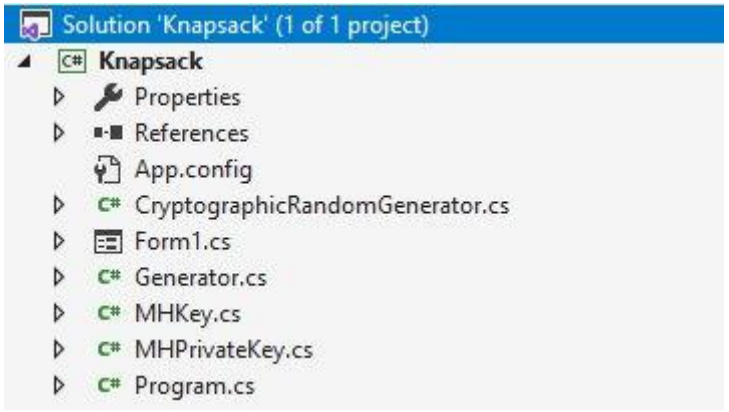

Рисунок А.1. Структура проекту.

CryptographicRandowGenerator.cs:

```
using System;
using System.Security.Cryptography;


namespace Knapsack
{
        public class CryptographicRandomGenerator :
IDisposable
    {
        private readonly RNGCryptoServiceProvider csp;
                public CryptographicRandomGenerator()
        {
            csp = new RNGCryptoServiceProvider();
        }
              public int Next(int minValue, int
maxExclusiveValue)
        {
            if (minValue == maxExclusiveValue) return
minValue;

            if (minValue > maxExclusiveValue)
```

```
            {
                throw new
ArgumentOutOfRangeException($"{nameof(minValue)} must be lower
than {nameof(maxExclusiveValue)}");
            }

            var diff = (long)maxExclusiveValue - minValue;
            var upperBound = uint.MaxValue / diff * diff;

            uint ui;
            do
            {
                ui = GetRandomUInt();
            } while (ui >= upperBound);
            return (int)(minValue + (ui % diff));
        }

        private uint GetRandomUInt()
        {
            var randomBytes =
GenerateRandomBytes(sizeof(uint));
            return BitConverter.ToUInt32(randomBytes, 0);
        }

        private byte[] GenerateRandomBytes(int bytesNumber)
        {
            var buffer = new byte[bytesNumber];
            csp.GetBytes(buffer);
            return buffer;
        }
        private bool _disposed;

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }
```

```
            {
            if (_disposed)
            {
                return;
            }

            if (disposing)
            {

                csp?.Dispose();
            }

            _disposed = true;
        }
    }

}
```

Generator.cs:

```
using System.Collections.Generic;
using System.Linq;

using Org.BouncyCastle.Math;


namespace Knapsack
{


    public class Generator
    {
                        public List<BigInteger>
W_Superincreasing { get; set; } = new List<BigInteger>();

                public BigInteger Q { get; set; }
```

```
        public BigInteger R { get; set; }

        private CryptographicRandomGenerator RandomGenerator {
get; set; } = new CryptographicRandomGenerator();

        public Generator(int length)
        {
            Generate(length);
        }

        private void Generate(int length)
        {

            int maxRandVal = 1000;
            BigInteger sum = BigInteger.ValueOf(0);
            for (int i = 0; i < length; i++)
            {
                BigInteger tmp =
sum.Add(BigInteger.ValueOf(RandomGenerator.Next(1,
maxRandVal)));
                W_Superincreasing.Add(tmp);
                sum = sum.Add(tmp);
            }

            Q =
sum.Add(BigInteger.ValueOf(RandomGenerator.Next(1,
maxRandVal)));


            FindCoprimeR();
        }

        private void FindCoprimeR()
        {
            R = BigInteger.Three;
```

```
            while (GetGreatestCommonDenominator(Q,
R).CompareTo(BigInteger.One) != 0)
            {
                R = R.Add(BigInteger.One);
            }
        }

        private BigInteger
GetGreatestCommonDenominator(BigInteger bd1, BigInteger bd2)
        {
            BigInteger bigger = (bd1.CompareTo(bd2) > 0) ? bd1
: bd2;
            BigInteger smaller = (bd1.CompareTo(bd2) < 0) ?
bd1 : bd2;
            BigInteger gcd = smaller;
            while (!BigInteger.Zero.Equals(smaller))
            {
                gcd = smaller;
                smaller = bigger.Mod(smaller);
                bigger = gcd;
            }
            return gcd;
        }


        public override string ToString()
        {
            string tmp = "";
            foreach (var item in W_Superincreasing)
            {
                tmp += $"{item}, ";
            }
            tmp += ";" + R;
            tmp += ";" + Q;

            return tmp;
        }
```

```
    }

}
```

MHKey.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;

using Org.BouncyCastle.Math;


namespace Knapsack
{
    public class MHKey
    {
        public List<BigInteger> B_PublicKey { get; set; } =
new List<BigInteger>();
        public MHPrivateKey PrivateKey { get; set; }
        public MHKey()
        {
            Generator gen = new Generator(8);

            BigInteger w = gen.R;

            foreach (var item in gen.W_Superincreasing)
            {
                B_PublicKey.Add(w.Multiply(item).Mod(gen.Q));
// w*a Mod n ..
            }

            BigInteger t = BigInteger.ValueOf(1);
            BigInteger one = BigInteger.ValueOf(2);
            while (one.IntValue != 1)
            {
                t = t.Add(BigInteger.One).Mod(gen.Q);
```

```
                one = w.Multiply(t).Mod(gen.Q);
            }

            BigInteger r = w.ModInverse(gen.Q);
            PrivateKey = new
MHPrivateKey(gen.W_Superincreasing, r, gen.Q);
        }



        public List<BigInteger> Encipher(string plain)
        {
            List<BigInteger> enciphered = new
List<BigInteger>();
            char[] charArray = plain.ToCharArray();
            for (int i = 0; i < charArray.Length; i++)
            {
                BigInteger tmp = BigInteger.ValueOf(0);
                string binary = CharToBinary(charArray[i]);
                for (int j = binary.Length - 1; j > -1; j--)
                {
                    if (binary[j] == '1')
                    {
                        tmp = tmp.Add(B_PublicKey[7 - j]);
                    }
                }
                enciphered.Add(tmp);
            }
            return enciphered;
        }

        public string Decipher(List<BigInteger> cipher,
MHPrivateKey key)
        {
            string decrypted = "";

            for (int i = 0; i < cipher.Count; i++)
```

```
            {
                BigInteger temp = cipher.ElementAt(i);
                int bitlen = temp.BitLength;
                int ff = 0;
                while (bitlen < (int)Math.Pow(2, ff))
                {
                    ff++;
                }
                if (ff > bitlen)
                {
                    bitlen = ff;
                }
                BigInteger bits = BigInteger.ValueOf(0);
                for (int j = 0; j < bitlen; j++)
                {
                    if
(temp.Mod(BigInteger.ValueOf(2)).CompareTo(BigInteger.ValueOf(
1)) == 0)
                    {
                        bits =
bits.Add(key.R.Multiply(BigInteger.ValueOf((long)Math.Pow(2,
j))));
                    }
                    temp = temp.ShiftRight(1);
                }
                bits = bits.Mod(key.Q);
                List<BigInteger> list = key.W_Superincreasing;
                BigInteger temper;
                int tmp = 0;
                int k = key.W_Superincreasing.Count - 1;
                while (k >= 0)
                {
                    temper = list.ElementAt(k);
                    if (bits.CompareTo(temper) > -1)
                    {
                        tmp += (int)Math.Pow(2, k);
                        bits = bits.Subtract(temper);
```

```
                    }
                    k--;
                }
                decrypted +=
(BinaryToChar(Convert.ToString(tmp, 2))).ToString();
            }
            return decrypted;
        }

        private


string CharToBinary(char ch)
        {
            string chBin = Convert.ToString((int)ch,
2).PadLeft(8, '0');
            return chBin;
        }

        private char BinaryToChar(string binStr)
        {
            char[] temp = binStr.ToCharArray();
            int sum = 0;
            for (int i = 0; i < temp.Length; i++)
            {
                sum += (int.Parse(char.ToString(temp[i])) <<
(temp.Length - i - 1));
            }

            return (char)sum;
        }



        public override string ToString()
        {
            string str = "";
            foreach (var item in B_PublicKey)
```

```
            {
                str += item + ",";
            }
            return str;
        }
    }
}
```

MHPrivateKey.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Org.BouncyCastle.Math;


namespace Knapsack
{
    public class MHPrivateKey
    {
        public List<BigInteger> W_Superincreasing { get; set;
}
                public BigInteger Q { get; set; }

                public BigInteger R { get; set; }


        public MHPrivateKey(List<BigInteger> w, BigInteger r,
BigInteger q)
        {
            W_Superincreasing = w;
            R = r;
            Q = q;
        }

        public override string ToString()
        {
```

```
            string tmp = "W (superincreasing): ";
            foreach (var item in W_Superincreasing)
            {
                tmp += $"{item}, ";
            }

            tmp += $"\r\nR = {R}\r\nQ = {Q}";

            return tmp;
        }
    }

}
```

Program.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Knapsack
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```