Пояснювальна записка
до дипломного проекту

на тему: Система захисту програми від користувацького втручання

Київ - 2019

ЗМІСТ

ВСТУП

Наразі дуже актуальною стала потреба захисту даних від дій зловмисників, які можуть викрадати, зчитувати або змінювати дані програм, які розроблюються певними компаніями для задоволення потреб користувачів. Зараз в епоху, коли створюються візуальні коди програмування, полегшується середа розробки для розробника-новачка і, як наслідок, зростає кількість неповноцінних розробників – виникає проблема погіршення якості коду, що спричиняє проблеми не лише з оптимізацією написаного коду. Сучасна середа розробки для починаючих програмістів або візуальних програмістів також не передбачає захист графічних зображень, музичних файлів, текстових файлів тощо.

Об’єктом дослідження є усі файли проекту деякого розробника-новачка, які можна відкривати та редагувати без жодних проблем у спеціальних програмах для редагування тексту, зображень, відеофайлів тощо.

Предметом дослідження є розробка системи захисту файлів проекту для розробника-новачка за допомогою цілого комплексу програмних і технічних рішень, надання розробнику-новачку усіх елементів, що необхідні йому для захисту його власної програми від втручання зловмисника.

Метою є захистити програму від різних типів користувацького втручання, забезпечивши при тому конфіденційність, цілісність та доступність програми звичайному користувачу, з урахуванням неможливості зміни існуючих файлів.

Для досягнення поставленої мети були сформульовані та вирішенні наступні завдання:

- огляд існуючого положення справ у сфері захисту програмного забезпечення;
- розроблення структурної схеми системи;
- розроблення алгоритму роботи програми;
- вибір та обґрунтування окремих елементів;
- моделювання роботи системи захисту програми;

- реалізація синтезованої системи у вигляді алгоритму програми;
- розрахунок стиснення інформації.

Практичне значення результатів дозволяє бути впевненим, що наша система захистить розробника-початківця від різного роду втручань та змін файлів або суттєво збільшить час, необхідний для зламу системи.

У роботі запропоновано використати комплекс із різних програмних і технічних рішень для захисту програми розробника-початківця, а саме:

- використання коду Хаффмана для кодування файлів;
- взяття хеш суми SHA-256 для перевірки закодованих файлів на зміни;
- використання строго налаштованого відліку системного часу для унеможливлення несанкціонованого вивчення програми через переривання;
- отримання хешу через мережу від серверу, з яким можна з'єднатися через спеціально налаштований 3G модем за протоколом TCP/IP.

Дипломний проект складається з наступних розділів:

- вступу;
- основних розділів;
- висновків;
- списку використаних джерел із 18 найменувань;
- 2 додатків;
- графічної частина, яка включає 4 кресленика формату А3;
- загальний обсяг 74 сторінки.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ І АНАЛІЗ МЕТОДІВ ЗАХИСТУ ПРОГРАМИ ВІД КОРИСТУВАЦЬКОГО ВТРУЧАННЯ

Існує багато методів захисту програм, систем тощо. І як правило, жодний з методів не дає стовідсоткову гарантію того, що захист неможливо зламати, але усі ці методи можна комбінувати один з одним. Розглянуто нижче різні методи захисту програм.

1.1 Локальний програмний захист

Необхідність уведення серійного номера (ключа) при запуску або при встановленні виникла тоді, коли програми поширювалися лише на фізичних носіях (наприклад, компакт-дисках). Серійний номер позначався на коробці з диском і підходив тільки до певної копії програми.

Проблема поширення образів дисків та серійних номерів мережею постала після того, як з'явилися мережі. Тому зараз застосовується метод захисту лише у сукупності з іншими методами (наприклад, організаційними).

1.2 Мережевий програмний захист

1.2.1 Локальний

Сканування мережі, де є однакові реєстраційні ключі на двох комп'ютерах, які знаходяться в одній локальній мережі, не дозволяє одночасну роботу двох програм з однаковим реєстраційним ключем.

Недоліком є те, що з моменту, коли брандмауер налаштовують таким чином, щоб він забороняє пропускання пакетів, які належать програмі, що захищена, але потрібні для цього деякі користувацькі навички, щоб коректно налаштувати його. До того ж, програми можуть взаємодіяти через локальну мережу (наприклад, для

роботи мережевої гри). Після цього брандмауер без проблем пропускає подібний трафік.

1.2.2 Глобальний

Якщо програма має працювати із будь-яким централізованим сервером і без цього не має жодного сенсу на існування (наприклад, сервери оновлень антивірусів, сервери онлайн-ігор). Програма може надавати серверу свій серійний номер, і у випадку, якщо номер є неправильним, то сервер відмовляє в роботі програми. Проблема в тому, що завжди є можливість створення серверу, який не перевіряє серійний номеру. Наприклад, альтернативою до Battle.net (від компанії Blizzard Entertainment) працював сервер battle.da, який був функціонально аналогічний, але цей сервер, на жаль, не зупиняв неавторизованих користувачів, які мали однакові серійні ключі. Наразі даний сервер не працює, але існують численні PvPGN-сервери, які також не перевіряють наявність серійного номеру.

1.3 Захист за допомогою компакт-дисків

Суть захисту полягає в тому, що програма може не запуститися, якщо не вставити оригінальний компакт-диск. Такий спосіб застосовувався здебільше в іграх. Стійкість такого захисту мала через великий набір інструментів для створення та емуляції образів компакт-дисків.

Зазвичай захист, використаний на цьому компакт-диску застосовується для захисту програм, що одночасно є ключом, а саме:

- запис інформації на спеціальних секторах;
- перевірка «збійних секторів» на їх наявність та вміст;
- перевірка швидкості зчитування окремих секторів.

Перші два методи, які використовують відповідне програмне забезпечення, є даремними через можливість зняття повного образу диску та емуляцію. Третій

метод є найнадійнішим, але це не допомагає ніяким чином захистити диски від емуляції, враховуючи геометрію розташування даних, фактично оминаючи даний захист. В дисках від StarForce, серед інших перевірок, також наявна окрема перевірка на можливість запису диску. Якщо диск можна перезаписати, то він вважається неліцензійним. Така перевірку можна пройти, якщо записати образ диску CD-R. Але є можливість приховати CD-R та CD-RW диски таким чином, щоб це виглядало як звичайний CD-ROM, але в драйвері захисту може бути перевірка на наявність емуляції, що є невеликою перевагою.

Наразі найчастіше використовуються програми для захисту від копіювання, такі як Tages, StarForce, SafeDisc, CD-RX і SecuROM.

Через різноманітність систем продажу більшості програм, ця форма захисту недоступна (наприклад, shareware-програми).

1.4 Захист за допомогою електронних ключів

Електронний ключ (донгл), що містить дані ключів (що є ліцензією), вставлений в один з портів комп'ютера (з інтерфейсом LPT, USB або COM) і записаний розробником, а саме:

- інформація для читання та запису, що, на жаль, не застосовується нині, бо земулювати ключ для зчитування;
- ключі апаратних криптографічних алгоритмів, які зараз дуже розповсюджені;
- власні алгоритми, які створені розробником програми (останній спосіб полягає в створенні електронного ключа з мікропроцесором, що з’явився не так давно і здатний виконувати унікальний код, на даний момент він часто використовується).

Переваги захисту із використанням електронних ключів:

- ключ можна вставляти в будь-який комп'ютер, де треба запустити програму;
- він не займає та не потребує наявності дисковода;

- електронний ключ може виконувати різні криптографічні перетворення;
- сучасні ключі можуть виконувати будь-який код, що створюється розробниками захисту (приклад: Senselock, Guardant Code).

Безпека даних залежить від того, що захист інформації за допомогою ключів (криптографічні ключі або завантажуваний код) не залишає самого ключа під час роботи з ним.

Основні недоліки:

- ціна варіюється від 15 до 30 доларів за штуку, що не дуже приємно;
- необхідність надати ключ кінцевому користувачеві.

Раніше недоліком можна було вважати невисоку швидкодію ключа, порівнюючи з CPU комп'ютером, але найсучасніші ключі досягають продуктивності в 1.25 DMIPS (приклад, Guardant чи HASP), а техніка захисту не передбачає безперервний обмін з ключем.

Проблеми із встановленням ключа на певні платформи, які були здатні працювати з однією або численними копіями захищеної програми та просто перебували з ними в одній локальній мережі, на даний момент вирішені за допомогою програмних або апаратних засобів «перекидання» USB-пристроїв по мережі, а також за допомогою мережевих ключів.

1.5 Прив’язка до параметрів комп’ютера й активація

Прив'язка до інформації про користувача або серійних номерів складових його комп'ютера з подальшою активацією програмного забезпечення сьогодні застосовується досить часто (наприклад: ОС Windows).

У процесі встановлення, відповідно встановленим запчастинам комп'ютера та параметрам встановленої ОС, програма вимірює код активації, тобто її контрольне значення. Це значення передається розробнику програми. Залежно від цього розробник надає ключ для встановлення програми тільки для певної збірки

комп’ютеру. Інакше встановлення файлів або їх копіювання на інший пристрій призведе до того, що програма відмовиться працювати.

Перевагою є те, що не потрібне жодне спеціальне програмне забезпечення, а програма може розповсюджуватися через засоби цифрової дистрибуції (через онлайн магазини в мережі Інтернет).

Основний недолік: якщо користувач здійснює заміну деталей у випадку прив'язки до деталей персонального комп’ютеру або ноутбуку, то захист відмовляється працювати. Автори програм готові самі надати новий реєстраційний код у багатьох подібних випадках. Наприклад, компанія Microsoft у системі Windows XP дозволяла раз у 120 днів генерувати новий реєстраційний код, але іноді, це було можливим тільки тоді, коли користувач телефонував в службу активації та міг отримати новий код після спливання терміну.

Для отримання значення, зазвичай використовуються серійний номер вінчестера або серійний номер BIOS материнської плати. Щоб користувач не бачив ці дані, їх можна розмістити в нерозміченій області жорсткого диска.

До недавнього часу тільки розробники програмного продукту створювали подібний захист. Але наразі є набори розробок та технічної документації для роботи з програмними ключами, наприклад HASP SL від компанії Аладдін Р. Д. До того ж, все більше з’являються сервіси, що пропонують функції захисту разом із серверами активації й ліцензування одночасно (приклад: Protect online, Guardant Online).

1.6 Захист програм від копіювання шляхом їх перенесення в онлайн

Використання підходу SaaS являється іншим напрямом захисту програм, тобто перенесенням (повністю або частково) цих програм саме на сервіс. Це означає, що код програми перебуває та виконується на сервері, що доступний у глобальній мережі. Доступ до програми здійснюється за архітектурою тонкого

клієнта. Це один з кількох випадків, коли працює майже повний захист від копіювання.

Код виконується у «довіреному» місці, звідки код не може бути так просто скопійований.

Проте, виникають багато нових проблем з безпекою:

- такий захист залежить, серед іншого, від безпеки сервера, який використовує Інтернет;
- важливе надання конфіденційності запитів та автентифікації користувачів, можливість частого резервування та доступності рішення в цілому.

З'являються проблеми щодо довіри до сервісу, в тому числі в сфері права, бо він обробляє персональні дані користувачів, а також йому невідомо з яким захистом передається програмне забезпечення.

1.7 Уразливості сучасних методів захисту

Перевірку оригінального носія можна обійти спеціальною програмою, яка повністю копіює диск та створює драйвер віртуального дисковода, в якому розміщується образ, який програма приймає за ліцензійний диск. Це називається емуляцією диску. У багатьох іграх є «Mini Image», що по суті є використанням поданого варіанту, коли вставлений диск містить лише ліцензійну інформацію та має невеликий розмір у декілька мегабайт., тоді програма визнає його ліцензійним.

Основною вразливістю введення серійного номера є можливість розповсюдження дистрибутиву разом із серійним номером шляхом безперешкодного копіювання. Тому даний вид захисту зараз майже не використовується або використовується разом з іншими методами.

Активація програмного забезпечення, на відміну від попереднього методу, генерується з використанням унікальної інформації, а саме інформації про користувача та серійного номеру обладнання. В цьому випадку, під час генерації коду активації у момент встановлення програми є вірогідність того, що зловмисник

зробить емуляцію «універсального» апаратного оточення (наприклад, зчитування відповідної інформації при перехоплення звернень). Також, якщо спеціально не заплутати код захищеної програми чи використати слабкі методи, зловмисник зможе знайти код генерації та винести в окрему утиліту код активації, у так званий генератор ключів або keygen, або просто виокремити за межі програми всю процедуру активації, що буде складніше, оскільки дана процедура може викликатися в різних частинах програми.

Часто зустрічається судження про використання звернення до електронного ключа, де створюється можливість емуляції бібліотек інтерфейсу API або електронного ключа. Це дійсно можна зробити при поганій реалізації захисту електронного ключа, де програма читає та пише що-небудь під час перевірки наявності ключа. Проте засновані на виклику симетричних та асиметричних алгоритмів шифрування, програми власної розробки істотно виключають можливість його емуляції, оскільки зібрати достатню кількість інформації для створення статистики викликів неможливо, бо звернення до ключа відбуваються щоразу по різному. Таким чином, стійкість захисту загалом залежить від навичок програміста, що його писав, в окремому випадку від наявності унікальних захисних механізмів, зроблених розробником системи захисту. Однак, потенційна стійкість даного захисту майже завжди висока.

«Відключення» захисту шляхом зміни програмного коду може бути виконане при невикористанні або використанні слабких спеціальних інструментів заплутування коду. Після цього програма дизасемблюється або декомпілюється і потім код, що досліджується на наявність захисту, видаляється.

Багато захисних засобів надають інструменти протидії, а саме:

- шифрування коду, що унеможливлює вивчення коду за допомогою дизасемблера;
- заплутування коду «помилковими вітками», які збивають хакера з пантелику;
- перевірка цілісності файлу, що не дає переписувати код;

- віртуалізація коду з власною системою команд.

Всі ці рішення роблять неможливим вивчення та аналізування логіки захисту, а й заодно підвищують її стійкість.

1.8 Хешування даних

«Хешування (англ. hashing) — це перетворення вхідного масиву даних довільної довжини у вихідний бітовий рядок фіксованої довжини. Такі перетворення також називаються хеш-функціями, або функціями згортання, а їхні результати називають хешем, хеш-кодом, хеш-сумою, або дайджестом повідомлення (англ. message digest)» [1]

Хеш-функція використовується у хеш-таблицях, що по суті є структурою даних, широко використаних у програмному забезпеченні для швидкого пошуку даних. Хеш-функції застосовуються для оптимізації таблиць та баз даних завдяки тому, що в однакових записах однакові значення хеш-функції. У файлах великого розміру цей підхід пошуку дублікатів є ефективним. Прикладом цього буде пошук подібних ділянок у послідовностях ДНК. Криптографічна хеш-функція робить можливим перевіряти які саме вхідні дані зіставляються із значенням, заданим хешем, якщо вхідні дані є невідомими, то знаючи збережене значення хеш-функції буде складно відновити вхідне значення (або еквівалентну альтернативу). Це є будівельним блоком для HMACs, що застосовується для забезпечення цілісності переданих даних, які надають автентифікацію повідомлень.

Хешування використовують для пошуку дублікатів в серіях наборів даних, для побудови унікальних ідентифікаторів для наборів даних, застосовується для побудови асоціативних масивів, для зберігання паролів в системах захисту (у цьому випадку доступ до області пам'яті, де знаходяться паролі, не дозволяє відновити сам пароль), для контрольного підсумовування з метою виявлення випадкових або навмисних помилок при зберіганні або передачі, при створенні електронного підпису (на практиці часто підписується не саме повідомлення, а його хеш-образ).

Якщо кількість значень хеш-функцій менша, ніж число варіантів значень вхідного масиву, то однозначної відповідності між хеш-кодом та вихідними даними немає. Є велика чисельність масивів з різним вмістом, де на виході ми отримуємо однакові хеш-коди, так звані колізії. Важливу роль в оцінці якості хеш-функцій відіграє саме ймовірність виникнення колізій.

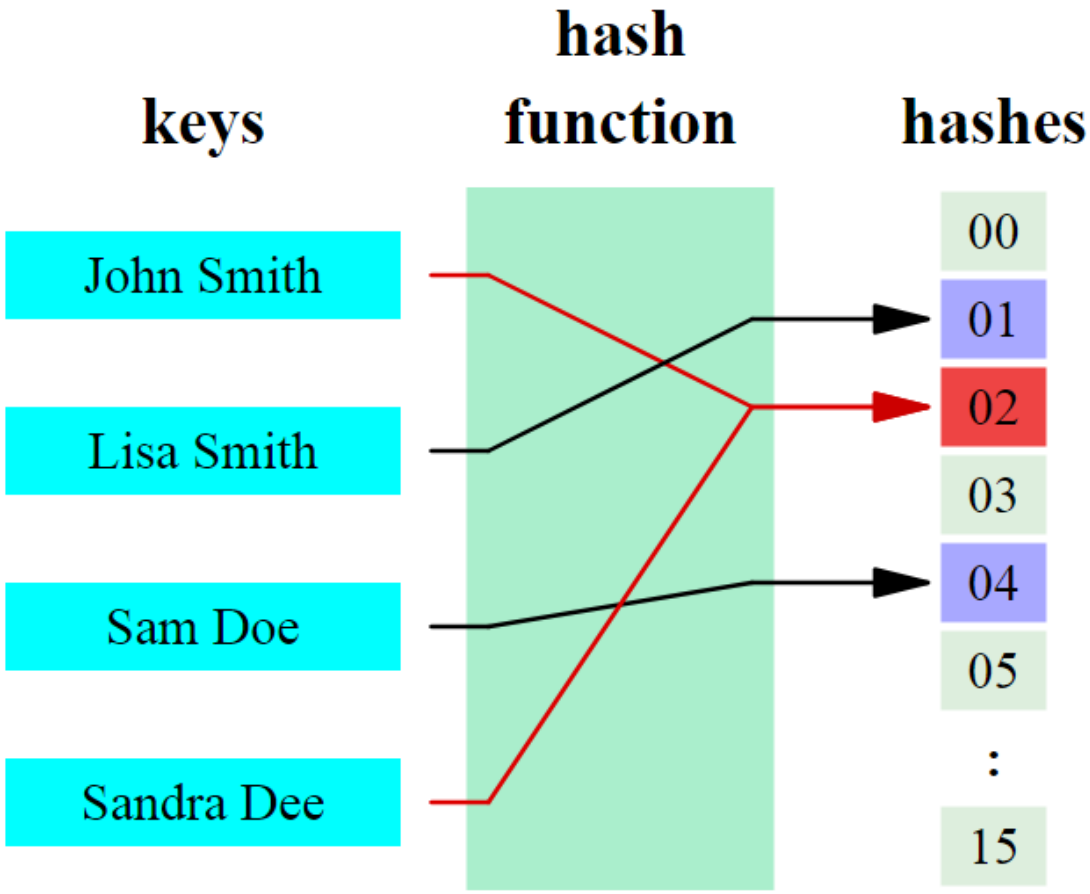


Рисунок 1.1 – Приклад колізії між «John Smith» та «Sandra Dee», яким відповідає одне значення

Наразі у світі розроблено багато алгоритмів хешування, які мають різну розрядність, криптостійкість та обчислювальну складність, тощо. Специфіка розв'язуваної задачі впливає на вибір тієї чи іншої хеш-функції. Контрольна сума або CRC як раз можуть бути найпростішими прикладами хеш-функцій.

Хеш-функції поєднані з рандомізацією функцій, контрольною сумою, відбитками пальців, контрольними цифрами, з шифрами і кодами, що виправляють помилки, але їх часто плутають. Кожні з них має свою власну область застосування,

оптимізованими та розробленими по-різному, наявні різні вимоги, хоча ці поняття певною мірою подібні один до одного.

1.9 Види хеш-функцій

Гарна хеш-функція повинна відповідати двом властивостям:

- швидко обчислюватися;
- мінімізувати кількість колізій.

Функція з M є прикладом «поганої» хеш-функції, яка десятизначному натуральному числу k співставляє три цифри, що були обрані з середини двадцятизначного квадрата числа k.

$$\forall k_0 \leq h(k) < M, \tag{1.1}$$

де k_0 = 0, M = 1000;

$h(k)$ має якесь значення між нулем та не більше ніж 1000.

Якщо ключі не мають великої кількості нулів справа чи зліва, то в такому випадку такий метод підходить для реальних даних, коли значення хеш-кодів мають рівномірно розподілитися між «000» і «999».

Проте є інші прості та надійні методи, на яких базується багато хеш-функцій.

1.9.1 Хеш-функції на основі ділення

Перший метод полягає у тому, що як хеш застосовується залишок від ділення на M, де M — це кількість всіх можливих хешів:

$$h(k) = K \; mod \; M, \tag{1.2}$$

де $K = k_{n-1}k_{n-2} \ldots k_0$ – це бінарні ключі.

Тому зрозуміло, що при парному M значення функції буде парним, при парному K. Спричинення істотного переміщення даних у файлах буде наявне при непарному M та непарному K. Не слід застосовувати системи числення комп'ютера як M базу, оскільки від декількох цифр числа K залежатиме лише хеш-код, ці цифри розташовуються праворуч і спричинить велику кількість колізій. Взагалі на практиці застосовують просте M, такий вибір повністю прийнятний у більшості випадків.

Варто згадати про метод хешування, основою якого є ділення на поліном по модулю два. В цьому методі M також має бути степенем двійки, а поліноми виглядають як бінарні ключі K. В даному випадку в якості хеш-коду, можуть взяти значення коефіцієнтів полінома, який отримують як залишок від ділення K на заздалегідь обраний поліном P степіня m:

$$h(x) = K(x)\ mod\ P(x), \tag{1.3}$$

де $h(x) = h_{m-1} \dots h_1 h_0$.

Якщо обрати правильно $P(x)$, то цей спосіб гарантує відсутність колізій між майже однаковими ключами.

1.9.2 Мультиплікативна схема хешування

Обрання деякої цілої константи A, взаємно простої з ω , тобто кількості можливих варіантів значень у вигляді машинного слова (в комп'ютерах IBM PC 2^{32}) є другим методом. Тоді виникає можливість взяти хеш-функцію виду:

$$h(K) = \left[M \left[\frac{A}{\omega} * K\right]\right], \tag{1.4}$$

У цьому випадку M являє собою степінь двійки, а h(K) укладатиметься зі старших бітів правої половини добутку A*K на комп'ютері з двійковою системою

числення. Перевагами даних двох методів варто виділити те, що вони вигідно використовують те, що реальні ключі невипадкові. Наприклад, за умови, якщо ключі являють собою арифметичну прогресію (припустимо послідовність назв «ім'я1», «ім'я2», «ім'я3»). Мультиплікативний метод відобразить арифметичну прогресію у наближену арифметичну прогресію різних хеш-значень і в порівнянні з випадковою ситуацією це зменшить кількість колізій.

Одним з видів даного методу є хешування Фібоначчі, яке базується на властивостях золотого перетину. В цьому випадку A обирається найближче до $\varphi^{-1} * \omega$ ціле число, взаємно просте з ω.

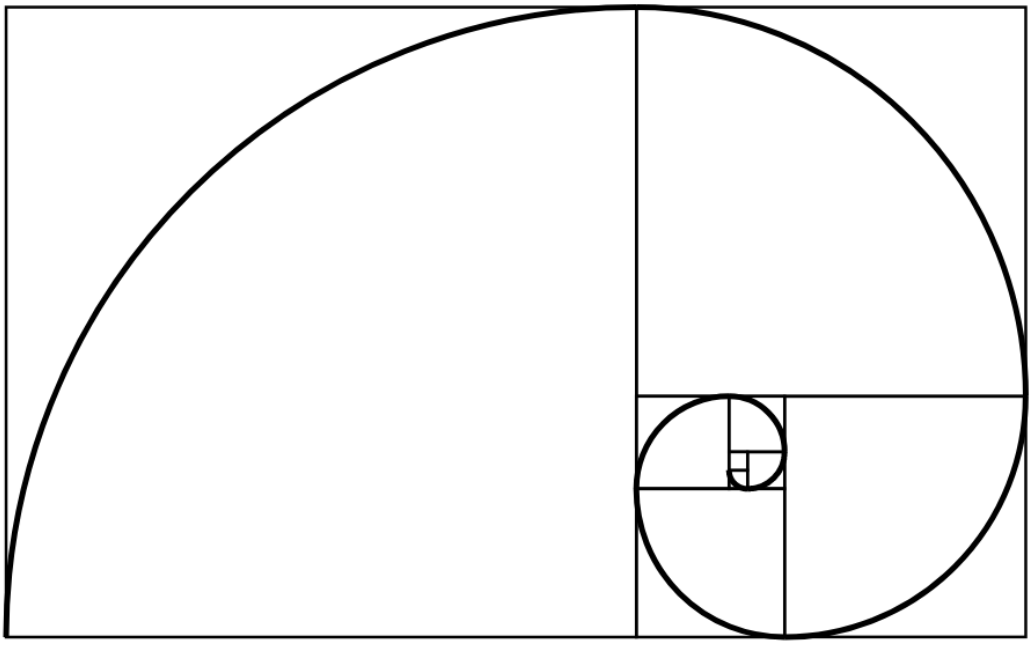


Рисунок 1.2 – Спіраль Фібоначчі, апроксимація золотої спіралі, що утворена дугами та проведені через протилежні кути квадратів Фібоначчі

1.9.3 Хешування рядків змінної довжини

Вище викладені методи застосовуються за умови, коли нам необхідно ретельно розглядати ключі, побудовані з декількох слів або ключі зі змінною довжиною. До речі, за допомогою додавання за модулем ω або операції «додавання по модулю 2» можна скомбінувати слова в одне. Хеш-функція Пірсона є одним з алгоритмів, що працюють за таким принципом.

Хешування Пірсона — алгоритм, запропонований Пітером Пірсоном (англ. Peter Pearson) для процесорів з 8-бітними регістрами, завданням якого є швидке обчислення хеш-коду для рядка довільної довжини. На вхід функція отримує слово W, що складається з n символів, кожен розміром 1 байт, і повертає значення в діапазоні від 0 до 255. При цьому значення хеш-коду залежить від кожного символу вхідного слова. [2]

Алгоритм варто описати таким псевдокодом, який використовує таблицю перестановок T та отримує на вхід рядок W:

```
h: = 0
For each  c  in  W  loop
    index := h xor c
    h := T [index]
End loop'
Return  h
```

Рисунок 1.3 – Псевдокод для хешування рядків змінної довжини

Перевагою алгоритму слід вважати:

- простоту обчислення;
- той факт, що не має таких вхідних даних, для яких імовірність колізії найбільша;

- можливість модифікації в ідеальну хеш-функцію.

Запропонуємо обчислення як альтернативний спосіб хешування K ключів, котрі складаються з l символів:

$$h(K) = \left(h_1(x_1) + h_2(x_2) + \cdots + h_1(x_1)\right) mod\ M, \quad (1.5)$$

де $K = x_1 x_2 \ldots x_1$.

Висновки до розділу

У першому розділі розглянуто велику кількість методів захисту інформації, зазначено їх особливості, переваги та недоліки.

Для розробки системи захисту програми від користувацького втручання виберемо використання локального програмного захисту лише для кодування та декодування файлів, а не для звичайного запиту та зберігання ключа.

Також вирішено обрати захист від копіювання програм, які перенесено в онлайн. Даний метод зараз вважається дуже надійним у випадку, якщо зв'язок між користувачем та сервером ліцензування захищено.

Цей метод захисту було обрано про причині розрахунку хеш суми за те, що яка б не відбулась зміна – це обов'язково призведе до зміни хеш суми.

Також показано, що жодний із методів не гарантує стовідсотковий захист. У наступному розділі будуть обрані конкретні рішення та буде обґрунтовано, чому саме обрано ті чи інші методи, які допоможуть розробити систему захисту програми.

2 АНАЛІЗ ТА ВИБІР ОПТИМАЛЬНОСТІ ТЕХНІЧНИХ РІШЕННЬ

2.1 Структура схема системи шифрування

Структура роботи приграми-шифратора представлена на рисунку 3.1

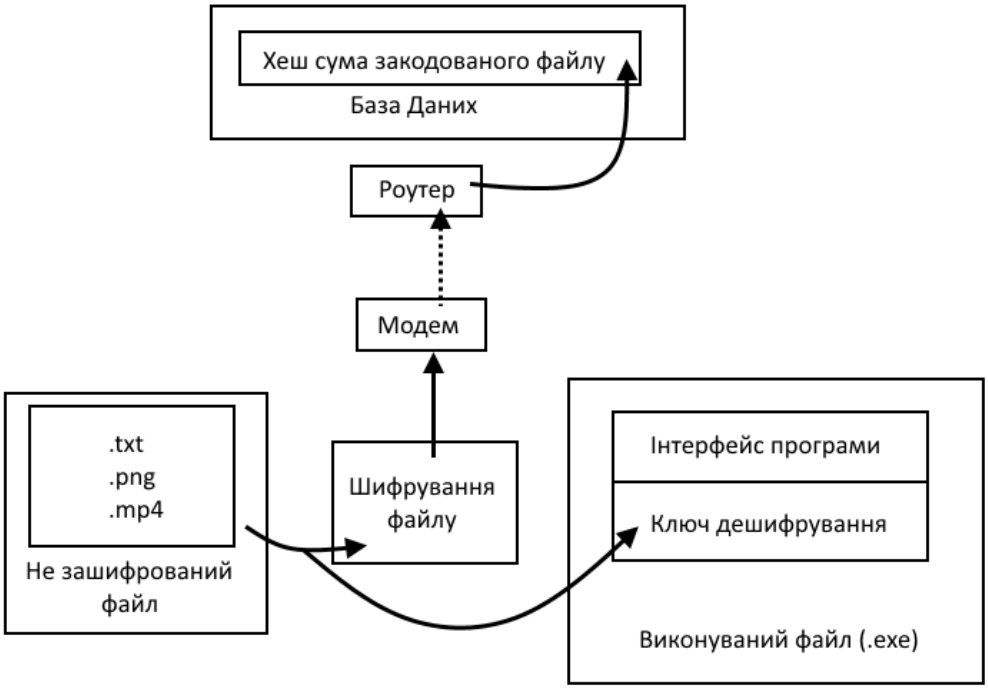


Рисунок 2.1 – Структурна схема системи шифрування

Наш не зашифрований файл, будь це текстовий файл, зображення або відеофайл існують в одній дирекрорії, що і виконуваний файл. Ми за допомогою кодеру на етапі розробки програми програмістом-початківцем починаємо шифрувати файл. Під час шифрування ми створюємо спеціальний файл, куди записуємо ключ, що необхідно буде програмісту-початківцю на етапі розробки додати у виконуваний файл своєї програми. Як зберігати він буде ключ – це вже не наша справа, головне, щоб під час запуску та дешифрування ключ міг бути використаний для дешифрування. Зашифрований файл на цьому етапі також запишеться в окремий файл.

Далі кодер викликає операцію взяття хешу від зашифрованого файлу та відправляє отриманий хеш на наш сервер ліцензування через спеціальний 3g для

того, щоб під час запуску програми ми з’єдналися через спеціально налаштований 3g модем до нашої бази даних, де буде зберігатися інформація про наш хеш, її назву та дату створення. Після цього кодер в окремому файлі згенерує код разом з ключем та запропонує додати це програмісту початківцю у свій код виконуваного файлу.

2.2 Структура роботи програми-дешифратора

Структура роботи програми-дешифратора представлена на рисунку.

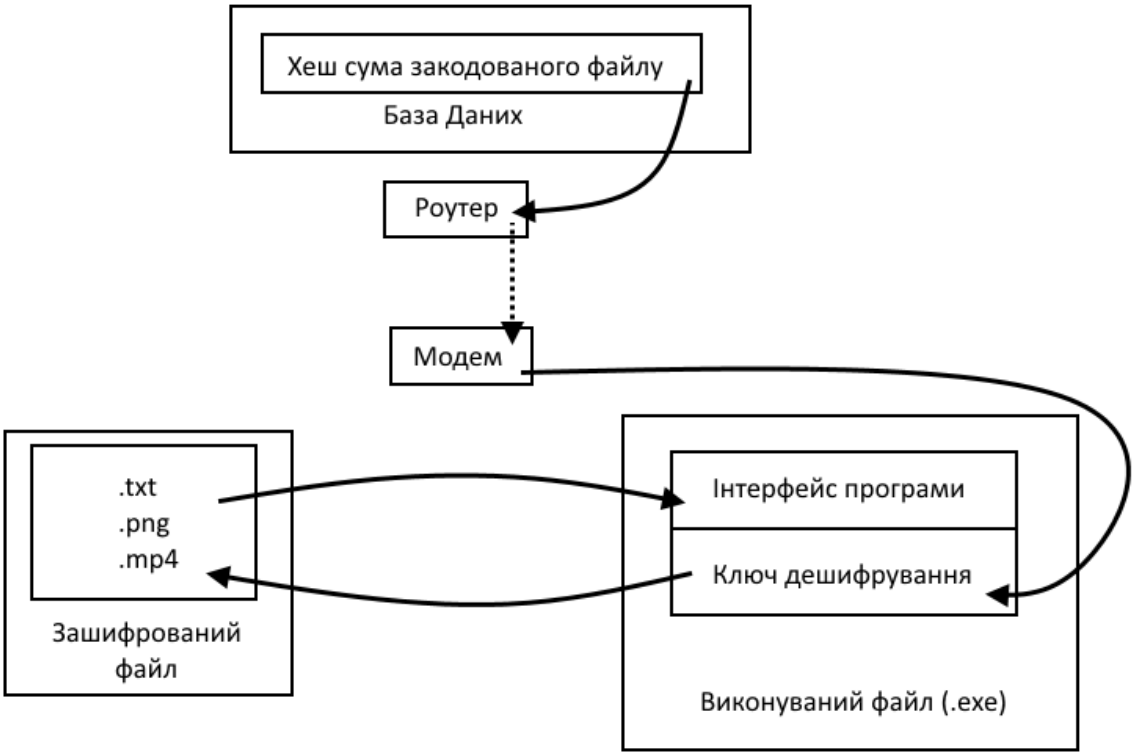


Рисунок 2.2 – Структура програми-дешифратора

Користувач запускає виконуваний файл, розроблений розробником-початківцем, та перше що відбувається – це з’єднання через наш спеціально налаштований 3g модем із сервером ліцензування, після чого ми звідти отримуємо хеш закодованого файлу. Наш зашифрований файл існує неподалік від виконуваного файлу, що у свою чергу звертається до зашифрованого файлу та виконує операцію взяття хешу. Після цього отриманий хеш порівнюється з тим, що прийшов через TCP протокол із серверу ліцензування, і якщо вони рівні – ми

продовжуємо роботу програми, якщо не рівні – програма завершується з помилкою «Деякі файли було змінено. Робота програми неможлива».

Після порівняння ми починаємо операцію дешифрування файлу, використовуючи ключ, який ми зберігаємо у виконуваному файлі і як наслідок – ми відкриваємо дешифровані файли в середині виконуваного файлу та бачимо те, що було зашифровано.

2.3 Модеми та їх призначення

Пристрій, призначений для обробки сигналів мережі Інтернет та їх декодування, є модемом. Він перетворює аналоговий сигнал у цифровий, наприклад, сигнал, який йде телефонними лініями декодується у сигнал, що зрозумілий комп'ютеру, і навпаки. Для зв'язку через комп'ютер, у модемі є цифровий інтерфейс, а також аналоговий для зв'язку через телефонну лінію. Перетворюється сигнал просто, а саме характеристики сигналу вимірюються з певною частотою, а потім за допомогою спеціального алгоритму перетворення вже записуються в цифровому вигляді.

За конструктивними особливостями розрізняють наступні їх різновиди:

- зовнішні;
- внутрішні.

Внутрішні модеми містяться в самому комп'ютері і зроблені вони у вигляді плати розширення, яка на материнській платі зазвичай підключається в слот PCI.

Зовнішні модеми виготовлені у вигляді окремого пристрою, який за допомогою роз'єму на мережевій карті підключається до комп'ютера. Є ще такі, що зустрічаються дуже рідко, але вони інтегровані в материнську плату.

Перевагою зовнішнього модему є те, що він працює від електромережі, і не створює для блоку живлення додаткове навантаження, завдяки присутності індикації за допомогою лампочок можна орієнтуватися в якому стані знаходиться

підключення до мережі. Знаходження внутрішнього модему всередині системного блоку робить його непомітним, що є головною перевагою.

Один з типів зовнішніх модемів - ADSL. Він має для підключення до телефонної лінії додатковий ADSL-роз'єм.

Бездротові модеми, гідно оцінені власники ноутбуків, яким потрібно в будь-якій точці земної кулі підключення до інтернету. Ці модеми використовують різні стандарти бездротового зв'язку для передачі даних. Загалом, сучасні бездротові модеми підключаються за допомогою портів USB.

Наразі модем з Wi-Fi роутером користується великим попитом. Користувач отримує один пристрій замість двох за рахунок його універсальності, такий пристрій забезпечує безперебійний доступ до інтернет-даних та виконує всі необхідні функції.

Модем має сигнальний, універсальний та модемний процесори, які виконують власне саму модуляцію, демодуляцію сигналу і керують режимами роботи інших частин модему. А також в модемі є аналогова частина, яка здійснює сполучення з мережею та пам'ять (ПЗУ, ППЗУ, ОЗУ), а всім цим керує спеціальний контролер.

Раніше цифро - аналогові модеми працювали на швидкостях до 56 Кбіт/С, то в даний момент популярні ADSL-модеми, які працюють за допомогою технології, що дозволяє передавати дані зі швидкістю до 100 Мбіт/с, а телефонна лінія, на доданок до цього, залишається вільною. Через обмеження в швидкості у телефонних лініях на практиці такі модеми передають дані на швидкості 1 - 24 Мбіт/с, що також є непоганим показником.

З кожним роком інтернет - провайдери повільно, але впевнено збільшують швидкість передачі даних по своїм лініям. В недалекому майбутньому модеми можливо передаватимуть дані на швидкостях рівних швидкостям сучасних локальних мереж.

2.4 Стек протоколів TCP/IP

За ініціативою Міністерства оборони США понад 20 років тому як набір загальних протоколів для різнорідного мережного середовища для зв'язку експериментальної мережі ARPAnet з іншими мережами був розроблений стек TCP/IP. Значний внесок у розвиток такого стеку, який отримав свою назву від популярних протоколів IP і TCP, внесли фахівці з університету Берклі, які реалізували протоколи стека у версії операційної системи UNIX. До поширення протоколів TCP, IP та інших протоколів стека призвела популярність цієї операційної системи. Наразі цей стек використовується у величезній кількості корпоративних мереж, а також для зв'язку вузлів всесвітньої інформаційної мережі Інтернет.

На нижньому рівні стек TCP/IP підтримує всі популярні стандарти фізичного й канального рівнів: для глобальних мереж — протоколи роботи на аналогових комутованих і виділених лініях PPP, SLIP, протоколи територіальних мереж ISDN та X.25, а для локальних мереж — це Token Ring, FDDI, Ethernet. Протоколи IP і TCP є основними протоколами стека і вони дали йому назву. В термінології моделі OSI, ці протоколи належать до мережного і транспортного рівнів, відповідно. За просування пакета по складеній мережі відповідає IP, а надійність його доставки гарантує TCP.

Використовуючись в мережах різних країн і організацій упродовж тривалого часу, стек TCP/IP увібрав у себе велику кількість протоколів прикладного рівня. До них належать такі популярні протоколи, як поштовий протокол SMTP, що використовується в електронній пошті мережі Інтернет, протокол емуляції терміналу telnet та гіпертекстові сервіси служби WWW, протокол пересилання файлів FTP і багато інших.

Популярність Інтернету призвела до змін у розміщенні сил у світі комунікаційних протоколів — протоколи TCP/IP, на яких побудований Інтернет, швидко почали тіснити безперечного лідера минулих років — стек IPX/SPX

компанії Novell. Триває процес просування стека TCP/IP на лідируючі позиції в будь-яких типах мереж і зараз обов'язково є програмна реалізація цього стека у комплекті постачання будь-якої промислової операційної системи. Є велика кількість локальних, територіальних і корпоративних мереж, що безпосередньо не є частинами Інтернету, у яких також використовуються протоколи TCP/IP. Їх називають мережами TCP/IP або просто IP-мережами, щоб відрізняти ці мережі від Інтернету.

Будучи спочатку створеним для глобальної мережі Інтернет, стек TCP/IP має багато особливостей, які надають йому перевагу перед іншими протоколами, коли йдеться про побудову мереж, що включають глобальні зв'язки. Зокрема, його здатність фрагментувати пакети є дуже корисною властивістю, завдяки якій протокол може застосовуватися у великих мережах. Дійсно побудована за зовсім різними принципами, складна складена мережа часто складається з мереж. Власна величина максимальної довжини одиниці переданих даних (кадру) може бути встановлена у кожній із цих мереж. Необхідність поділу переданого кадру на кілька частин може виникнути при переході з однієї мережі, що має більшу максимальну довжину кадру, в іншу, з його меншою максимальною довжиною. Ефективно розв'язує це завдання протокол IP стека TCP/IP.

Гнучка система адресації, що дозволяє в порівнянні з іншими протоколами аналогічного призначення більш просто включати до інтермережі (об'єднаної мережі), мережі інших технологій є іншою особливістю технології TCP/IP. Для побудови великих гетерогенних мереж ця властивість сприяє застосуванню стека TCP/IP.

Структура протоколу TCP/IP, як видно на рисунку 2.3, подібна до нижніх рівнів еталонної моделі взаємодії відкритих систем (моделі OSI). Протокол TCP/IP підтримує всі стандартні протоколи канального та фізичного рівнів.

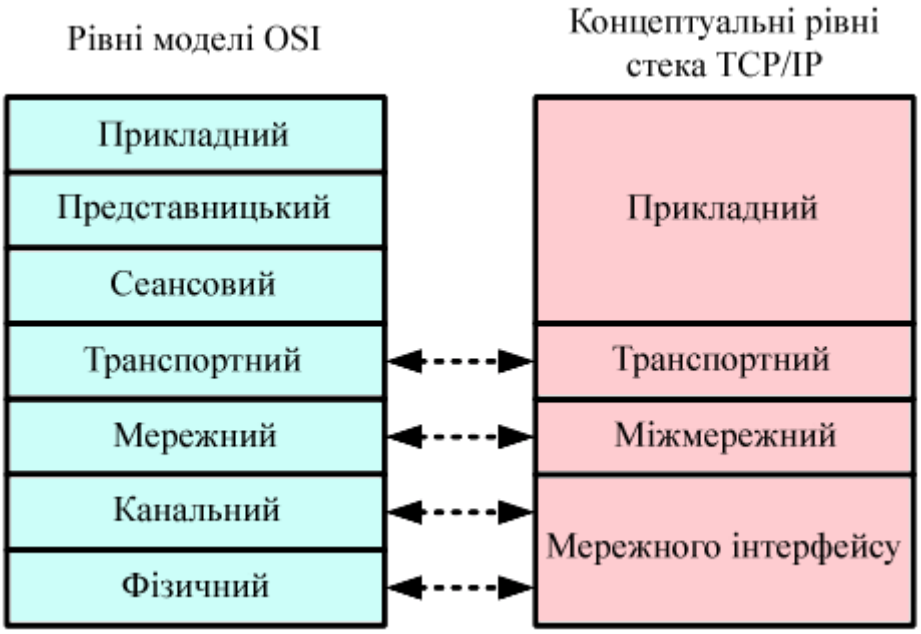


Рисунок 2.3 - Чотирьохрівнева модель стека TCP/IP і її взаємозв'язок з моделлю OSI

Інформація в протоколі TCP/IP передається у вигляді послідовності дейтаграм. Повідомлення може пересилатися у вигляді ряду дейтаграм, які збираються в повідомлення в місці прийому.

На рівні аплікацій (прикладний рівень) TCP/IP широко використані сервіси прикладного рівня, що зображено на рисунку 2.4. До них належать: протокол емуляції віддаленого терміналу (telnet), протокол вирішення імен (DNS), протокол передачі файлів між віддаленими системами (FTP), поштові протоколи тощо. Кожна прикладна програма вибирає такі типи транспортування, як послідовність окремих повідомлень або безперервний потік повідомлень. Дані передаються транспортному рівню в необхідній формі прикладною програмою.

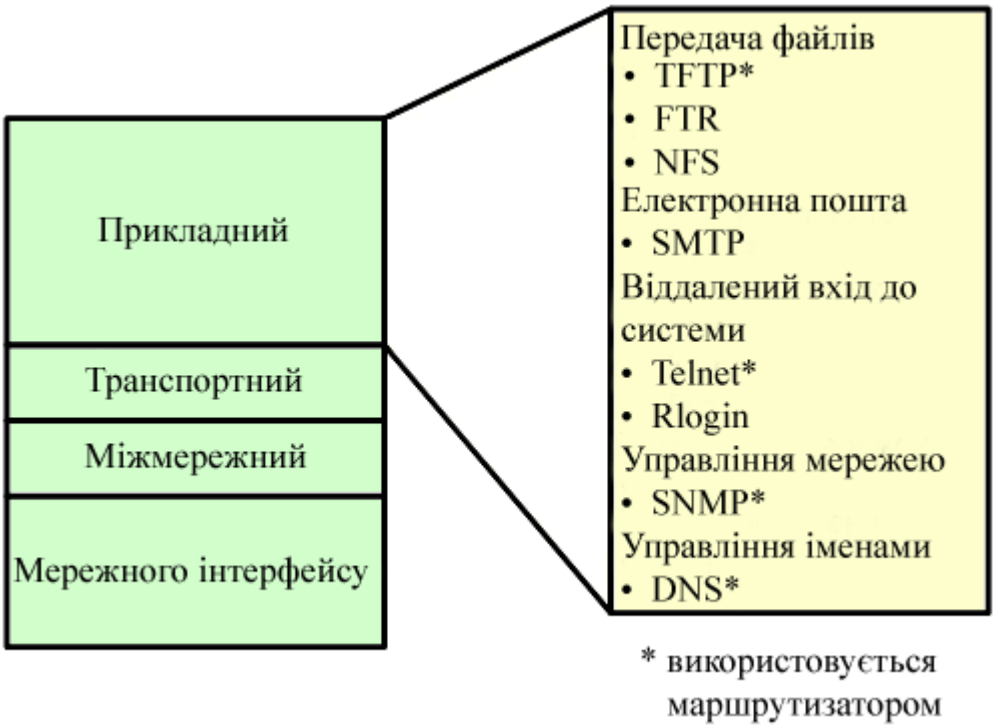


Рисунок 2.4 - Рівень аплікацій TCP/IP

Транспортний рівень стека TCP/IP. Взаємодія між прикладними програмами є основним завданням транспортного рівня стека TCP/IP. Транспортний рівень виконує дві функції:

- керує потоком, що забезпечується механізмом ковзних вікон;
- завдяки наявності порядкових номерів сегментів і підтверджень, гарантує надійність передачі.

Механізм підтвердження правильного прийому з дублюванням передачі загублених пакетів або пакетів, що надійшли з помилками, використаний для цього в TCP/IP. Від декількох прикладних програм транспортний рівень приймає дані та надсилає їх нижньому рівню. Він долучає додаткову інформацію до кожного пакета, в тому числі контрольну суму.

На транспортному рівні, як показано на рисунку нижче, використовуються два протоколи:

- TCP — протокол зі встановленням квітування та з'єднання. Він відповідає за розбиття повідомлень на сегменти, їх збирання у вузлу призначення,

повторне відсилання всього, що виявилося не отриманим, і збирання повідомлень із сегментів. Гарантовану доставку даних протокол TCP забезпечує за рахунок утворення логічних (віртуальних) з'єднань між віддаленими прикладними процесами;

- протокол дейтаграм користувача (User Datagram Protocol, UDP), не орієнтований на встановлення з'єднання. На цьому рівні квітування протокол UDP не застосовується, хоча і відповідає за передачу повідомлень, оскільки для перевірки доставки сегментів відсутнє програмне забезпечення.

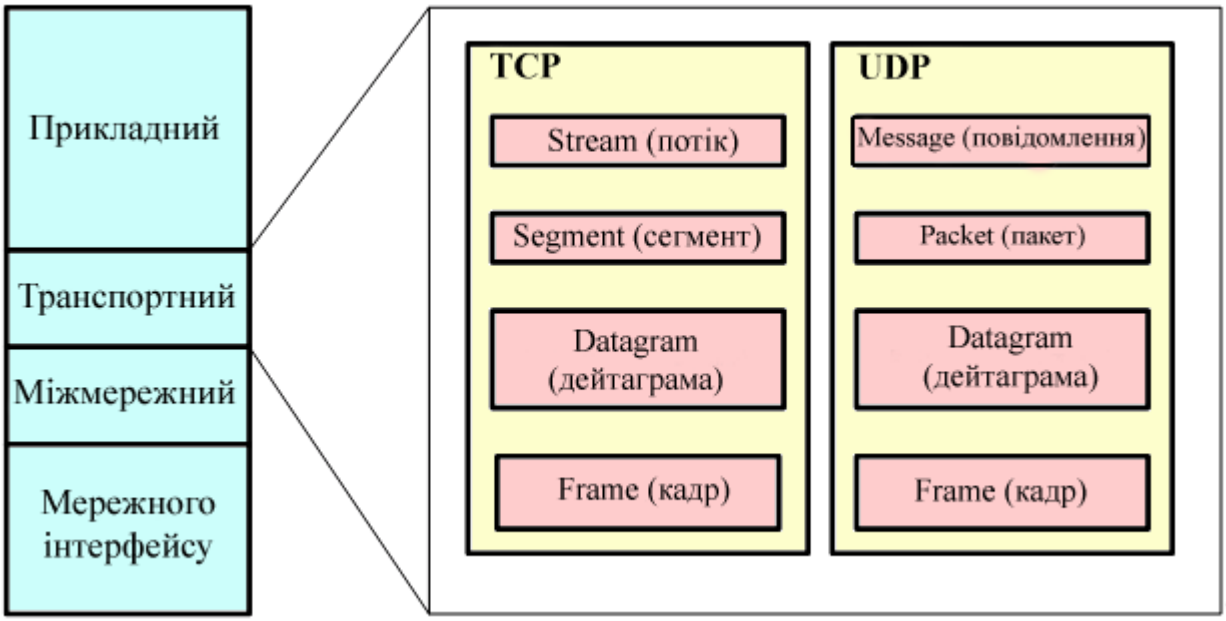


Рисунок 2.3 - Транспортний рівень з установленням з'єднання (TCP) або без установлення з'єднання (UDP)

Протоколи, які використовуються для передачі не дуже важливих повідомлень без установлення з'єднання, а також тоді, коли повідомлення є в наявності досить коротке і аплікація користувача може надіслати повторний запит при його втраті. Наприклад, для передачі широкомовних повідомлень усім вузлам підмережі використовується UDP. Протоколи без встановлення попереднього

з’єднання, незважаючи на низьку надійність, все-таки мають деякі переваги: їх простота і швидкість передачі обумовлюють нижчу вартість комунікації.

Рівнем міжмережевої взаємодії є мережний рівень стека TCP/IP. Рівень керує взаємодією між користувачами в мережі та приймає запит на посилку пакета від транспортного рівня разом із зазначенням адреси одержувача. Рівень також інкапсулює пакет у дейтаграму, заповнює її заголовок і використовує алгоритм маршрутизації за необхідності. Рівень перевіряє правильність інформації та обробляє дейтаграми, що надходять. На стороні одержувача програмне забезпечення мережного рівня знищує заголовок дейтаграми і визначає, який із транспортних протоколів оброблятиме пакет.

В якості основного протоколу мережного рівня, протокол IP використовується в стеку протоколів TCP/IP. Він створювався саме з метою передачі інформації в розподілених мережах. Перевагою протоколу IP є змога ефективно працювати в мережах зі складною топологією, при цьому раціонально використовуючи пропускну здатність низько швидкісних ліній зв’язку. В основі протоколу IP закладений дейтаграмний метод, що не гарантує доставку пакета, але лише «прагне» цього.

Рівень мережного інтерфейсу (рівень доступу) стека TCP/IP. Фізичному і канальному рівням моделі OSI відповідає найнижчий рівень стека TCP/IP. Цей рівень у стеку протоколів TCP/IP не регламентований. За прийом дейтаграм відповідає рівень мережного інтерфейсу, а також за їх передачу конкретною мережею. Драйвер пристрою або система (комутатор, маршрутизатор), що використовує свій протокол канального рівня, можуть реалізувати інтерфейс із мережею. Він підтримує стандарти фізичного і канального рівнів популярних локальних мереж: Token Ring, Ethernet, FDDI тощо. Протоколи з’єднань PPP і SLIP підтримуються для територіально-розподілених мереж, а для глобальних мереж — протокол X.25. Підтримка технології комутації чарунок — ATM теж підтримується. Включення до стеку протоколів TCP/IP нових технологій

локальних або розподілених мереж і регламентація їх новими документами RFC стало звичайною практикою.

2.5 3G модеми

Існують три критерії згідно з якими можна розділити всі 3G модеми на групи:

- дизайн і інтерфейс 3G модема;
- ціна 3G модема;
- технічні характеристики 3G модема.

2.5.1 Інтерфейс і дизайн 3G модема.

Модем підключається до комп'ютера, ноутбука та інших пристроїв за допомогою такого типу роз'єму, як інтерфейс. Існувало 3 різних інтерфейсу - USB, PCMCIA та Express card. 3G модем с USB - наразі, є самим поширеним і практичним. Такі модеми, як Express card і PCMCIA не випускаються з 2008 року. Перевага USB модемів полягає в тому, що USB (англ. Universal Serial Bus - «універсальна послідовна шина) – в обчислювальній техніці є послідовним інтерфейсом передачі даних для середньо швидкісних і низько швидкісних периферійних пристроїв.

По-перше, інтерфейс USB мають практично всі обчислювальні пристрої з операційними системами. Тому, придбаний 3G модем с USB інтерфейсом можна буде підключити як до ПК, так і до ноутбука, нетбуку або до Wi-Fi роутера з підтримкою USB модемів.

По-друге, саме роз'єм USB та компактні розміри подібних пристроїв дозволяють додавати в них всілякі додаткові функції, такі як слот для карт пам'яті формату MicroSD. Це дозволяє використовувати 3G бездротовий USB модем в якості флешки, або ж приймача GPS (система глобального позиціонування) для

орієнтування на місцевості. Наявність роз'єму для підключення зовнішньої антени є додатковою перевагою модемів USB.

По-третє, 3G модеми з інтерфейсом USB представлені на ринку інформаційних технологій в асортименті на будь-який смак і ціновий діапазон, тому як є найпоширенішими. Список найбільш популярних 3G USB модемів (Huawei E3131, Novatel U760, Huawei K4606, Huawei E173, Huawei EC306).

2.5.2 Ціна 3G модема.

Ціни варіюються в великих межах і не завжди висока ціна 3G модема є показником високої якості. Слід звернути увагу на USB роз'єм і матеріал корпусу, коли купуєте модем. Буде краще, якщо роз'єм USB буде виконаний на шарнірі, а корпус виконаний з міцного або прогумованого пластика. Останнім часом найпопулярнішими стали портативні Wi-Fi роутери з вбудованим 3G модемом.

Ціна нам важлива для того, щоб здешевити грошову ціну на нашу систему захисту, що зробить даний продукт більш розповсюдженим.

2.5.3 Технічні характеристики 3G модема.

Основні технічні характеристики, на які при виборі модему варто звернути увагу:

- швидкість прийому-передачі даних. Для CDMA 3G модемів в стандарті Rev A - вона становить до 3,1 Мб / сек, в стандарті Rev B - до 14,7 Мб / сек. Для GSM 3G модемів - швидкість досягає до 63,3 Мб / сек;
- функція відправки USSD запитів, яка є актуальною тільки для HDSPA модемів;
- можливість відправки SMS;
- наявність GPS приймача. Деякі модеми мають вбудований GPS приймач, однак слід розуміти, що це неповноцінний навігатор. За допомогою

приймача матимете змогу переглянути лише координати свого місця розташування, причому залишає бажати кращого швидкодія приймача GPS USB модему для ноутбука;

- наявність слота для карти пам'яті MicroSD дозволяє перетворити 3G модем ще й у флешку за допомогою карти MicroSD.

2.6 Налаштування модему (роутеру)

Саме мережевий пристрій (модем, роутер) працює на своїй маленькій операційний системі. Тому як вони є самостійними пристроями, тому драйверів для роутеру не існує. Потрібно лише зайти в панель управління роутеру, щоб його налаштувати. Панель управління називають по-різному: особистий кабінет, веб-інтерфейс, сайт з настройками маршрутизатора, сторінка роутеру, тощо.

В залежності від виробника й моделі, мережевий пристрій має заводські налаштування, які встановлені за замовчуванням. Серед них - адреса для входу в панель управління, заводський логін і пароль.

Якщо ви знаєте дані від панелі управління, тоді можна там знайти вкладку «Захист» і обрати «Мережевий екран» та увімкнути його, якщо це не зроблено. Після цього заходимо у вкладку «Фільтр IP адрес» і додаємо правила. У нашому випадку для захисту варто дозволити лише одну адресу, це адреса нашого серверу ліцензування, де збережені хеш значення.

Також слід влаштувати VPN на стороні користувача, який будемо надавати для захисту передачі даних. На нашому сервері, застосовуючи мережевий екран надаємо доступ лише до бази даних, куди будемо звертатися, щоб отримати хеш. Тому ми налаштовуємо модем VNP, увівши у вкладці «Мережа» і у графі «Тип підключення WAN», такі дані, як:

- тип підключення WAN: L2TP/PPTP;
- ім’я користувача;
- пароль;

- IP адресу сервера або доменне ім’я;

- режим підключення: Автоматичний.

Далі переходимо у вкладку «DHCP» та обираємо «Налаштування DHCP» та вводимо первинний та вторинний DNS в налаштуванні. Наприклад 8.8.8.8, що є DNS від компанії Google.

2.7 Оптимальне кодування

Знайти код, який був би оптимальним з будь-якої точки зору практично неможливо. Тому код можна вважати оптимальним лише за певних умов (з точок зору швидкості передачі інформації, тобто його здатності виправляти помилки тощо).

Існує декілька методик створення безнадмірних кодів, що підходять з точки зору швидкості передачі інформації.

До таких кодів (з точки зору їхньої довжини, тобто швидкості передачі інформації) належать нерівномірні коди, які завдяки комбінаціям мінімальної середньої довжини можуть передавати повідомлення. Можна вважати, що вони не є повністю безнадмірними, тому як такими можна вважати коди, які задовольняють умову кількості інформації та рівності обсягу. Через використання заборонених кодових комбінацій ці коди загалом мають потенціальну надмірність, до них належать комбінації, що доповнюють вершини неповного кодового дерева, яке має відповідати оптимальному нерівномірному коду (ОНК), до повного утворення такого рівномірного коду.

Оптимальним кодуванням називається процедура перетворення символів первинного алфавіту q_1 на кодові комбінації вторинного алфавіту q_2, при якій середня довжина повідомлення у вторинному алфавіті мінімальна. [3]

Отже, досягнення рівності між кількістю інформації I, що здійснюється джерелом повідомлень, та обсягом інформації Q на вході приймача повідомлень вважається головною метою цього кодування. Якщо $I = QI_{сер} = H$, то збільшення

швидкості передачі інформації завдяки поліпшенню процедури кодування стає неможливим.

Спираючись на теореми про статичну надмірність, можна надати кілька методик побудови ОНК для дискретних ансамблів повідомлень $\{X, p(x)\}$ із середньою довжиною кодових комбінацій.

$$\bar{n}(X) = H(X)/\log q \tag{2.1}$$

Перша більш універсальна методика побудови ОНК базується на методиці Шеннона-Фано. Вона має на увазі побудову в кодовому алфавіті з кількістю якісних значень q. Завдяки цій методиці можна здійснити такі процедури:

- розмістити множину з N повідомлень, що кодуються, у порядку спадання ймовірностей;
- впорядковані за ймовірностями повідомлення поділити, за можливості, на q рівноймовірні групи;
- кожна група в одній і тій самій послідовності матиме символи алфавіту q (тобто, всі повідомлення першої групи — дають першу якісну ознаку цього алфавіту, всім повідомленням другої групи — другу якісну його ознаку тощо);
- створені групи поділяють на рівноймовірні підгрупи, їх кількість дорівнює або менша ніж q (якщо після ділення в групі може залишитись одне повідомлення, то подальше розбиття стає неможливим);
- кожній з утворених підгруп надають якісні ознаки з алфавіту q;
- розбивання та надання ознак алфавіту q можна повторювати до тих пір, поки після чергового розбиття в новоутворених підгрупах залишиться не більш як одне повідомлення.

Слід зважити на відхилення від рівноймовірних значень, що з'являються при розбитті на підгрупи, для побудови ОНК за представленою методикою. Їх треба

враховувати відповідно до правил зарахування остач ділення та середнього відхилення:

- щоб повідомлення первинного джерела могли б поділити при наявності можливості на більш рівноймовірні підгрупи при утворенні ОНК з алфавітом *q*, остача попереднього ділення додаватиметься за абсолютним значенням сумарної ймовірності чергового ділення з остачею. І це називається різницею між квантом ділення та дійсним значенням сумарної ймовірності в групі (підгрупі), де квант ділення дорівнює *1/*q,
- середнє відхилення повинно бути меншим або має бути рівним значенню ймовірності першого символу чергового ділення. Якщо середнє відхилення не буде дорівнювати нулю, тоді середнє значення сумарної ймовірності в групі (підгрупі) при черговому діленні підраховується з доданням значення середнього відхилення (середнім відхиленням називається абсолютне значення суми остач від ділення на проміжних етапах побудови коду).

Розглянемо яким же чином для передачі 16 повідомлень за допомогою четверикового коду з алфавітом $q = 4$ будується ОНК, отож, якщо повідомлення на виході джерела буде з ймовірностями $p(x_i)$, як викладено на рисунку, то послідовність такої побудови буде такою:

- визначаємо квант поділу $1/q = 1/4 = 0{,}25$.
- поділяємо всі повідомлення, по можливості, на чотири рівномірні групи:

$\sum_{i=1}^{1} p(x_i) = 0{,}22$ остача від ділення дорівнює 0,03, оскільки (0,25 - 0,22) < (0,32 - 0,25);

$\sum_{i=2}^{4} p(x_i) = 0{,}28$, остача дорівнює 0,03, середнє відхилення по двох діагоналях нульове;

$\sum_{i=5}^{8} p(x_i) = 0{,}26$, остача дорівнює 0,01, середнє відхилення по двох діагоналях становить 0,01;

$\sum_{i=9}^{16} p(x_i) = 0{,}24$, остача дорівнює 0,01, середнє відхилення по двох діагоналях нульове.

Номер повідомлення	Імовірність повідомлення $p(x_i)$	Поділ на групи (підгрупи)			Кодова комбінація ОНК
		Перша	Друга	Третя	
1	0,22				0
2	0,1				10
3	0,1				11
4	0,08				12
5	0,07				20
6	0,07				21
7	0,06				22
8	0,06				23
9	0,05				30
10	0,05				31
11	0,04				320
12	0,03				321
13	0,02				330
14	0,02				331
15	0,02				332
16	0,01				333

Рисунок 2.4 – Таблиця ймовірностей на виході джерел

- За результатами першого поділу груп як перший символ кодових комбінацій надаємо послідовно якісні ознаки алфавіту $q = 4$ відповідно до третьої процедури першої універсальної методики побудови ОНК.
- Утворені групи, окрім першої, поділяємо на підгрупи. Друга та третя групи мають до чотирьох повідомлень, тому як другий символ кодових комбінацій та їм надаємо відповідно три та чотири якісні ознаки алфавіту q.
- Четверта група має вісім повідомлень, тому ділимо її на рівноймовірні підгрупи. Квант поділу (0,24 + 0,01) : 4 = 0,0625 (0,01 додається як остача від попереднього поділу). Поділ надання ознак алфавіту q виконуємо до тих пір, поки після чергового поділу в новоутворених підгрупах не зостанеться більш ніж одне повідомлення.

Слід визначити середню довжину $n_{сер}$ кодової комбінації ОНК для перевірки оптимальності коду відносно довжини кодових комбінацій. У випадку оптимальності така довжина не має перевищувати довжину рівномірного

четверикового коду, яким можна закодувати 16 повідомлень, тобто $q^n = 4^2 = 16 (n = 2)$:

$$n_{сер} = \sum_{i=1}^{16} p(x_i) n_i = 0{,}22 * 1 + (0{,}1 + 0{,}1 + 0{,}08 + 0{,}07 + 0{,}07 + 0{,}06 +$$

$$+0{,}06 + 0{,}05 + 0{,}05) * 2 + (0{,}04 + 0{,}03 + 0{,}02 + 0{,}02 + 0{,}01 + 0{,}01) *$$

$$* 3 = 0{,}22 + 0{,}64 * 2 + 0{,}14 * 3 = 0{,}22 + 1{,}28 + 0{,}42 = 1{,}92 < 2.$$

Отже, створений код є справді оптимальним, оскільки $n_{сер} < n$.

Оптимальність кодування слід визначати порівнянням ентропії, яка припадає на одне повідомлення, із середньою довжиною кодової комбінації, що повинні бути дуже близькими за значеннями. Ентропія повинна бути меншою від середньої довжини кодової комбінації або бути рівною їй.

Отже,

$$H = -\sum_{i=1}^{16} p(x_i) \log p(x_i) = -(0{,}22 \log 0{,}22 + 0{,}1 \log 0{,}1 + 0{,}1 \log 0{,}1 +$$

$$+0{,}08 \log 0{,}08 + 0{,}07 \log 0{,}07 + 0{,}06 \log 0{,}06 + 0{,}06 \log 0{,}06 +$$

$$+0{,}05 \log 0{,}05 + 0{,}05 \log 0{,}05 + 0{,}04 \log 0{,}04 + 0{,}03 \log 0{,}03 +$$

$$+0{,}02 \log 0{,}02 + 0{,}02 \log 0{,}02 + + 0{,}02 \log 0{,}02 + 0{,}01 \log 0{,}01) =$$

$$= 0{,}4806 + 2 * 0{,}3322 + 0{,}2915 + 2 * 0{,}2686 + 2 * 0{,}2435 +$$

$$+2 * 0{,}2161 + 0{,}1857 + 0{,}1517 + 3 * 0{,}1129 + +0{,}0664 =$$

$$= 3{,}6354 \text{ біт/повідомлення.}$$

Враховуючи це, варто звернути увагу на те, що ентропія завжди залежить від алфавіту оптимального коду, яким повідомлення має кодуватись, тобто слід урахувати кількість інформації, яка розміщується в одному елементі кодової комбінації. Для оптимального коду $q = 4$ в одному елементі кодової комбінації отримаємо 2 біти інформації.

Таким чином,

H = 3,6354 біт/повідомлення < $2n_{сер}$ = 3,84 біт/повідомлення < $2n$ = 2 * 2 = 4 біт/повідомлення, тобто код є оптимальним.

Друга універсальна методика утворення ОНК базується на зазначеній у кодовому алфавіті з кількістю якісних значень q. Відповідно до цієї методики виконуються такі методи побудови коду Хаффмана. Вона, як і методика Шеннона-Фано, передбачає створення ОНК процедури:

- розміщують множину з N повідомлень, що кодуються, згідно порядку спадання ймовірностей;
- останні N_0 повідомлень $(2 \leq N_0 \leq q)$ поєднують у нове повідомлення з імовірністю, яке дорівнюється сумі ймовірностей об'єднуваних повідомлень;
- утворену множину $(N - N_0 + 1)$ повідомлень розміщують відповідно до порядку спадання ймовірностей;
- поєднують останні q повідомлення та організовують множину повідомлень у порядку спадання ймовірностей. Так діють до тих пір, поки ймовірність чергового поєднаного повідомлення не дорівнюватиме одиниці;
- зводять кодове дерево, розпочинаючи з кореня, а гілкам цього дерева надають якісні ознаки кодового алфавіту q.

Кодові комбінації ОНК — це послідовність якісних ознак, які зустрічаються на шляху від кореня до вершини кодового дерева.

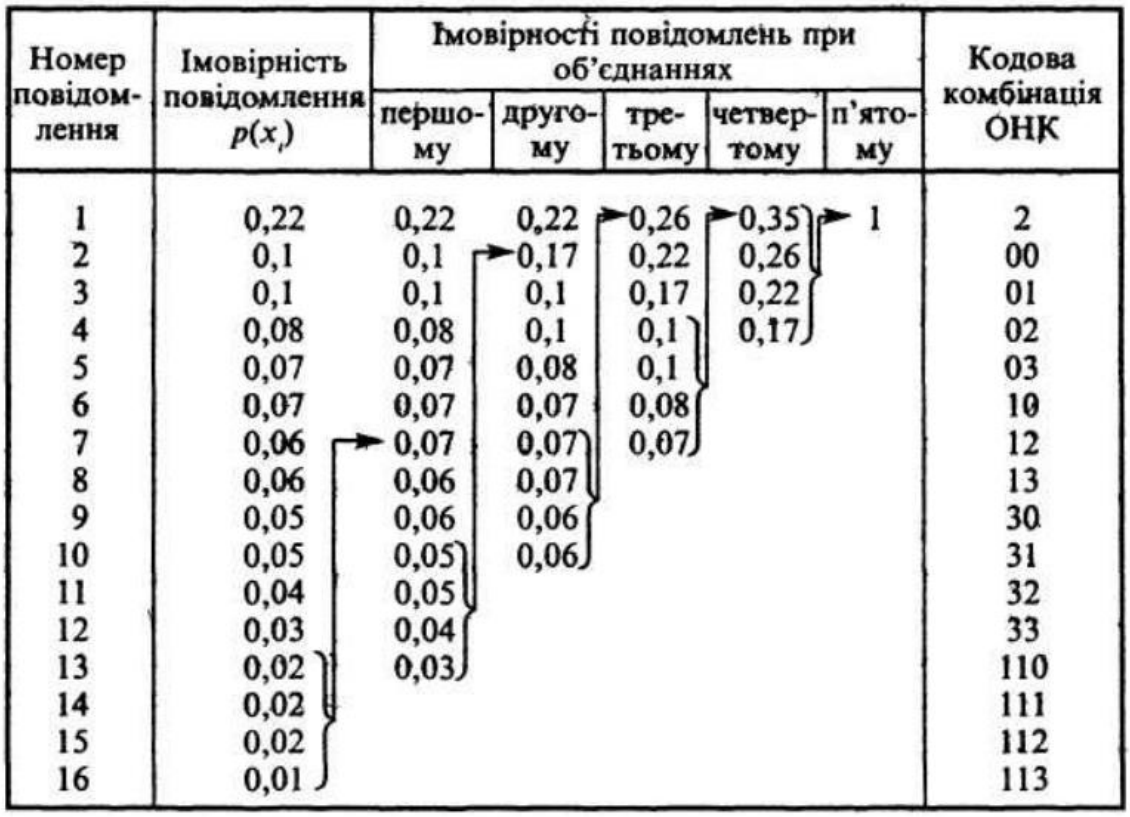
Створення ОНК відповідності до другої універсальної методики розглянемо щодо передачі 16 повідомлень за допомогою комбінаціям четверикового коду, які задано в попередньому прикладі. Послідовність цієї побудови буде такою:

- множину з N = 16 повідомлень розмістимо в порядку спадання ймовірностей;
- оскільки q = 4, поєднуємо останні чотири повідомлення й будуємо нове умовне повідомлення з імовірністю, яка дорівнює сумі ймовірностей поєднаних повідомлень;

- розмістимо побудовану множину з 16 - 4 + 1 = 13 повідомлень в порядку спадання ймовірностей;
- знов поєднуємо останні чотири повідомлення, потім впорядковуємо множину повідомлень у порядку спадання ймовірностей. Цю процедуру маємо повторити ще три рази, доки при останньому поєднанні сумарна ймовірність не матиме значення одиниці (рис. вище);
- створюємо кодове дерево. Його гілкам надаємо якісні ознаки кодового алфавіту від 0 до 3;
- до рисунку нижче додаємо кодові комбінації ОНК. Такі комбінації визначаються послідовністю якісних ознак, що будуть зустрічатися на шляху від кореня до певної вершини кодового дерева;
- з метою перевірки оптимальності коду відносно довжини кодових комбінацій, треба визначити середню довжину $n_{сер}$ кодової комбінації ОНК та ентропію, які містяться в одному повідомленні:

$$n_{сер} = 0,22 * 1 + (0,1 + 0,1 + 0,08 + 0,07 + 0,07 + 0,06 + 0,06 + 0,05 +$$
$$+ 0,05 + 0,04 + 0,03) * 2 + (0,02 + 0,02 + 0,02 + 0,01) * 3 =$$
$$= 0,22 + 0,71 * 2 + 0,07 * 3 = 1,85 < 2$$

H = 0,6354 біт/повідомлення < $2n_{сер}$ = 3,7 біт/повідомлення < $2n$ = 4 біт/повідомлення.



Номер повідомлення	Імовірність повідомлення $p(x_i)$	Імовірності повідомлень при об'єднаннях					Кодова комбінація ОНК
		першому	другому	третьому	четвертому	п'ятому	
1	0,22	0,22	0,22	0,26	0,35	1	2
2	0,1	0,1	0,17	0,22	0,26		00
3	0,1	0,1	0,1	0,17	0,22		01
4	0,08	0,08	0,1	0,1	0,17		02
5	0,07	0,07	0,08	0,1			03
6	0,07	0,07	0,07	0,08			10
7	0,06	0,07	0,07	0,07			12
8	0,06	0,06	0,07				13
9	0,05	0,06	0,06				30
10	0,05	0,05	0,06				31
11	0,04	0,05					32
12	0,03	0,04					33
13	0,02	0,03					110
14	0,02						111
15	0,02						112
16	0,01						113

Рисунок 2.5 – Таблиця, що побудована за другою універсальною методикою ОНК

Таким чином, цей код також вважається оптимальним і має кращі показники за оптимального коду, створеного за першою універсальною методикою, оскільки $n_{сер2} < n_{сер1}$ (1,85<1,92).

Ці універсальні методики мають неоднозначність, але перша з них дозволяє точніше створювати ОНК. Недоліком другої універсальної методики утворення ОНК можна вважати громіздкість (особливо зі збільшенням кількості повідомлень N та алфавіту q коду), і це можна пояснити необхідністю побудови кодового дерева.

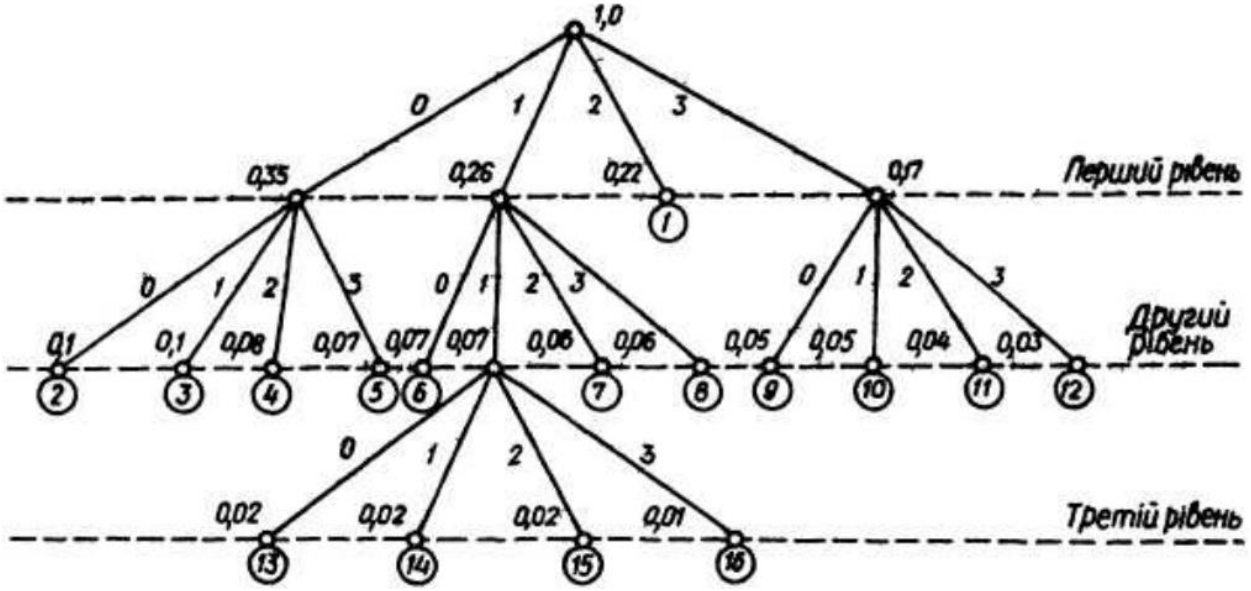


Рисунок 2.6 – Кодове дерево, побудовано за другою універсальною методикою ОНК

Відзначимо, що перевагою другої універсальної методики утворення ОНК з $q > 2$ при $N < q^n$ можуть бути вагоміші при більш ретельнішому виборі кількості найменш імовірних повідомлень, які поєднуються ще на першому етапі ($2 \leq N_0 \leq q$). На всіх послідуючих етапах ця кількість буде дорівнювати q.

2.8 Код Хаффмана

На нашу думку, оптимальним варіантом для кодування та декодування файлів було обрано код Хаффмана.

Алгоритм Хаффмана — адаптивний жадібний алгоритм оптимального префіксного кодування алфавіту з мінімальною надмірністю. Він був розроблений аспірантом Массачусетського технологічного інституту Девідом Хаффманом під час написання курсової роботи та опублікований в статті «A Method for the Construction of Minimum-Redundancy Codes» в 1952 року. Наразі він застосовується в багатьох програмах стиснення даних без втрат.

У відмінність від алгоритму Шеннона-Фано, цей алгоритм є завжди оптимальним і для вторинних алфавітів з більш ніж двома символами.

Цей метод кодування складається з двох основних етапів:

- побудови оптимального кодового дерева;
- побудови відображення код-символу на основі побудованого дерева.

На вході класичний алгоритм Хаффмана одержує таблицю частот, з якими зустрічаються символи у повідомленні. Потім відповідно до цієї таблиці будується дерево кодування Хаффмана (Н-дерево):

- список вільних вузлів складається з символів вхідного алфавіту. Кожен лист володіє вагою, що дорівнює або ймовірності, або кількості входжень символу у стиснене повідомлення;
- обираються два вільних вузли дерева з найменшими вагами;
- будується їхній батьківський вузол з вагою, що дорівнює їх сумарній вазі;
- вузол-батько вноситься в список вільних вузлів, а два його нащадки прибираються з цього списку;
- одній дузі, що виходить з вузла батька, ставиться у відповідність біт 1, іншій — біт 0;
- кроки, які починаються з другого, будуть повторюватися до тих пір, поки не зостанеться лише один вільний вузол в списку вільних вузлів. Він і буде розглядатися як корінь дерева.

Допустимо, у нас є наступна таблиця частот:

Таблиця 2.1 – Приклад даних «частота-символ»

Частоти	15	7	6	6	5
Символи	A	B	C	D	E

Цей процес можна представити як створення дерева, корінь якого є символом з сумою ймовірностей поєднаних символів, що був отриманий при поєднанні символів з останнього кроку, його n0 нащадки є символами з попереднього кроку тощо.

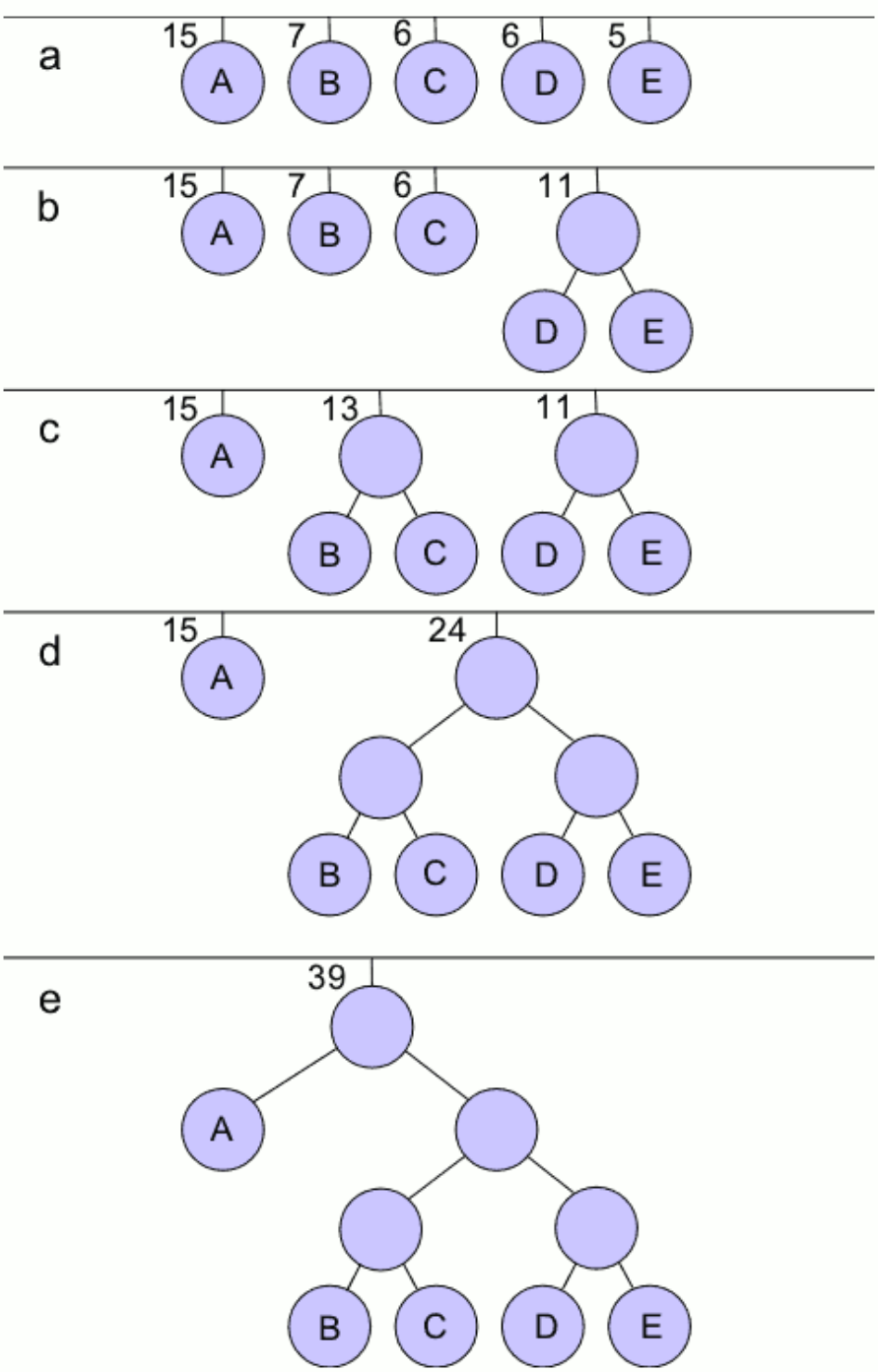


Рисунок 2.7 – Приклад кодового дерева

Для того, щоб визначити код для кожного із символів, які містяться у повідомлення, варто пройти шлях від листка дерева, що відповідає поточному символу, до його кореня, та накопичувати біти при пересуванні по гілках дерева (перша гілка в шляху відповідає молодшому біту). Послідовність бітів, що

одержана таким чином, є кодом даного символу, що був записаним у зворотному порядку.

Для даної таблиці символів коди Хаффмана виглядатимуть таким чином:

Таблиця 2.2 – Код, що сформований із кодового дерева

Символ	A	B	C	D	E
Код	0	100	101	110	111

Вони будуть однозначно декодовані при читанні їх з потоку, тому що жоден з одержаних кодів не являється префіксом іншого. Крім того, найчастіший символ повідомлення «A» закодований найменшою кількістю біт, а найбільш рідкісний символ «E» — найбільшою.

2.9 Бібліотека «CImg.h»

Така бібліотека нам стане в пригоді для того, щоб мова програмування C++, завдяки якій ми розроблюємо проект могла працювати із зображеннями, відео та звуками, де напряму неможливо буде зчитати текстовий файл методами мови програмування C++.

```
#include "CImg.h"
using namespace cimg_library;

int main() {
  CImg<unsigned char> image("lena.jpg"), visu(500,400,1,3,0);
  const unsigned char red[] = { 255,0,0 }, green[] = { 0,255,0 }, blue[] = { 0,0,255 };
  image.blur(2.5);
  CImgDisplay main_disp(image,"Click a point"), draw_disp(visu,"Intensity profile");
  while (!main_disp.is_closed() && !draw_disp.is_closed()) {
    main_disp.wait();
    if (main_disp.button() && main_disp.mouse_y()>=0) {
      const int y = main_disp.mouse_y();
      visu.fill(0).draw_graph(image.get_crop(0,y,0,0,image.width()-1,y,0,0),red,1,1,0,255,0);
      visu.draw_graph(image.get_crop(0,y,0,1,image.width()-1,y,0,1),green,1,1,0,255,0);
      visu.draw_graph(image.get_crop(0,y,0,2,image.width()-1,y,0,2),blue,1,1,0,255,0).display(draw_disp);
      }
    }
  return 0;
}
```

Рисунок 2.8 – Приклад використання бібліотеки у мові програмування C++ для відкриття зображення у вікні та його подальше опрацювання

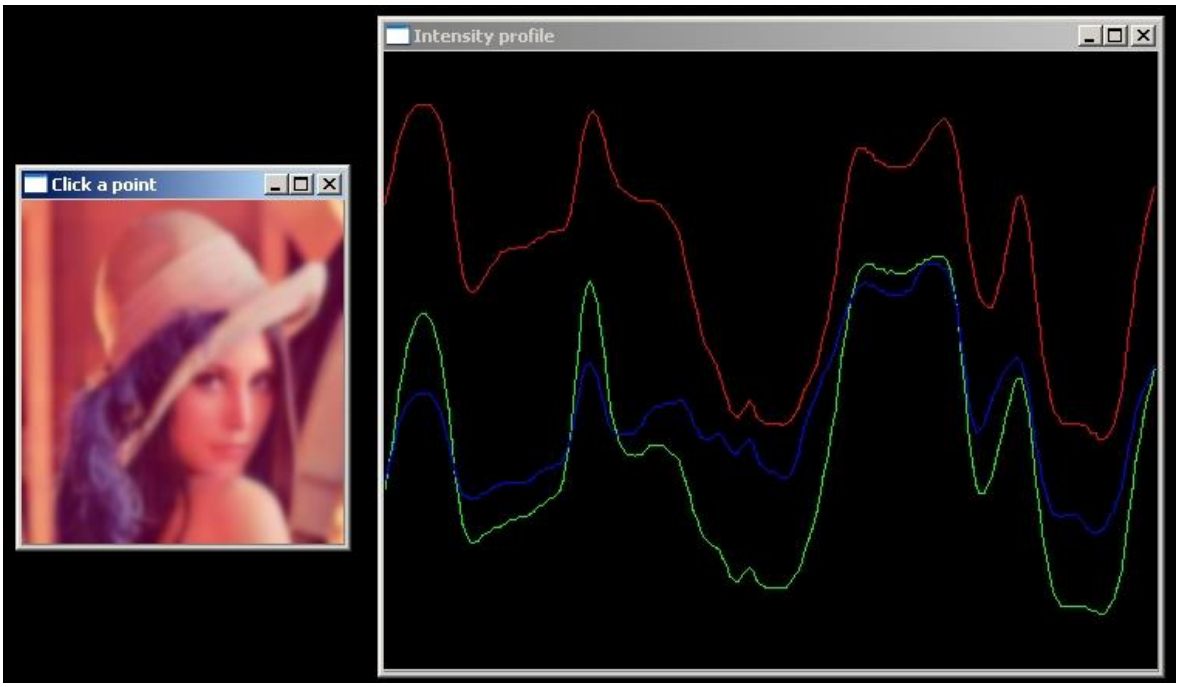


Рисунок 2.9 – Демонстрація роботи коду програми, що зображено на рисунку 2.8

Бібліотека CImg - це відкритий та невеликий C++ інструментарій для обробки зображень, який був розроблений з урахуванням цих властивостей:

- користь бібліотеки CImg визначає методи та класи керування зображеннями у вашому власному коді на C++. Ви можете застосовувати CImg для завантаження або збереження різних форматів файлів, до доступу до піксельних значень, до відображення, перетворення або фільтрування зображень, до малювання примітивів (текст, обличчя, криві, 3D-об'єкти

тощо), до обчислення статистики, керування взаємодією користувачів із зображеннями тощо;

- CImg визначає один клас зображення, що може являти собою набори даних, що мають до 4-х розмірів (від 1d скалярних сигналів до 3d-гіперспектральних об'ємних зображень), з типом пікселів шаблону (bool, char, int, float тощо). Колекції зображень і послідовності теж обробляються;
- CImg є автономним та безпечним для потоків і портативним. Вона повністю працює на різних операційних системах, таких операційних системах як Windows, BSD, Unix, MacOS X, тощо і сумісна з різними компіляторами C ++ (Visual C ++, g ++, clang ++, icc тощо);
- CImg складається з одного файлу заголовка CImg.h, який має бути доданий у ваш C++ h-файл. Він відображає тільки чотири різних класи, інкапсульовані в просторі імен cimg_library. Він може бути скомпільований з використанням мінімального набору стандартних C++ і системних бібліотек. Немає потреби в екзотичних або складних залежностях;
- CImg може застосовувати функціональні можливості зовнішніх інструментів або бібліотек, таких як ImageMagick, FFMPEG, Board, GraphicsMagick, FFTW3, libjpeg, Lapack, Magick ++, libcurl, libpng, libtiff, OpenCV, OpenEXR. Більш того, простий механізм плагіну дозволяє будь-якому користувачеві безпосередньо підвищувати можливості бібліотеки відповідно до своїх потреб.

2.10 Libavcodec

Libavcodec є універсальною бібліотекою для кодування і декодування аудіофайлів та відеофайлів, яка також є частиною пакета FFmpeg. Вона написана на мові C та здатна декодувати більшість мультимедіа форматів. За допомогою зворотного програмування була створена значна частина libavcodec.

До того ж ця бібліотека кодує та декодує субтитри і декілька фільтрів бітового потоку. Нас зацікавило кодування та декодування такого вигляду, щоб можна було використати код Хаффмана для шифрування даних.

Спільна архітектура надає різноманітні послуги, які починаються від бітів потоку вводу-виводу до оптимізації DSP, робить її придатною для реалізації надійних і швидких кодеків, а також для експериментів.

2.11 Застосування хеш-функцій

2.11.1 Криптографічні хеш-функції

Серед численної кількості хеш-функцій прийнято виділяти криптографічно стійкі, що використовуються в криптографії, оскільки додаткові вимоги на них накладаються. Для того, щоб хеш-функція вважалася криптографічно стійкою, вона має відповідати трьом основним вимогам, на яких засновано більшість застосувань хеш-функцій в криптографії, а саме:

- бути незворотної: для заданого значення хеш-функції m має бути обчислювально неможливо знайти блок даних *X*, для якого *H(X) = m*;
- бути стійкою до колізій першого роду: для заданого повідомлення *M* має бути обчислювально неможливим підібрати інше повідомлення *N*, для якого *H(N) = H(M)*;
- мати стійкість до колізій другого роду: має бути обчислювально неможливим підібрати пару повідомлень (M, M'), що мають однаковий хеш.

Викладені вимоги залежні один від одного:

- оборотна функція нестійка до колізій першого та другого роду;
- функція, нестійка до колізій першого роду, як і нестійка до колізій другого роду; тоді зворотне невірно.

Необхідно підкреслити, що ще не доведене існування незворотних хеш-функцій, для яких теоретично неможливе обчислення будь-якого прообразу

заданого значення хеш-функції. Взагалі, знаходження зворотного значення є лише обчислювально складним завданням.

Атака «днів народження» дозволяє знаходити колізії для хеш-функції з довжиною значень n бітів в середньому за приблизно $2^{n/2}$ обчислень хеш-функції. Якщо обчислювальна складність знаходження колізій для неї близька до 2n/2, то n-бітну хеш-функцію слід вважати криптостійкою.

Як приклад, якщо використовується 64-бітовий хеш на рівні близько $1.8*10^{19}$ різних виходів і якщо вони всі рівноймовірні (найкращий випадок), тоді необхідні лише $5.1*10^{9}$ спроб для отримання колізії, із використанням грубої сили. Це число відоме як межа днів народження (англ. birthday bound) і для n-бітових кодів її можна обрахувати як $2^{n/2}$.

Таблиця 2.3 – Ймовірність появи колізії при різному числі бітів

Бітів	Приблизна можлива кількість виходів(H)	Приблизна ймовірність випадкової колізії (p)		
		10^{-18}	10^{-15}	10^{-12}
16	$6,6*10^{4}$	2	2	2
32	$4,3*10^{9}$	2	2	2
64	$1,8*10^{19}$	6,1	$1,9*10^{2}$	$6,1*10^{3}$
128	$3,4*10^{38}$	$2,6*10^{10}$	$8,2*10^{11}$	$2,6*10^{13}$
256	$1,2*10^{77}$	$4,8*10^{29}$	$1,5*10^{31}$	$4,8*10^{32}$
384	$3,9*10^{115}$	$8,9*10^{48}$	$2,8*10^{50}$	$8,9*10^{51}$
512	$1,3*10^{154}$	$1,6*10^{68}$	$5,2*10^{69}$	$1,6*10^{71}$

Продовження таблиці 2.3

Бітів	Приблизна можлива кількість виходів(H)	Приблизна ймовірність випадкової колізії (p)		
		10^{-9}	10^{-6}	0,1%
16	$6,6*10^{4}$	2	2	11
32	$4,3*10^{9}$	2,9	93	$2,9*10^{3}$
64	$1,8*10^{19}$	$1,9*10^{5}$	$6,1*10^{6}$	$1,9*10^{8}$
128	$3,4*10^{38}$	$8,2*10^{14}$	$2,6*10^{16}$	$8,3*10^{17}$
256	$1,2*10^{77}$	$1,5*10^{34}$	$4,8*10^{35}$	$1,5*10^{37}$
384	$3,9*10^{115}$	$2,8*10^{53}$	$8,9*10^{54}$	$2,8*10^{56}$
512	$1,3*10^{154}$	$5,2*10^{72}$	$1,6*10^{74}$	$5,2*10^{75}$

Продовження таблиці 2.3

Бітів	Приблизна можлива кількість виходів(H)	Приблизна ймовірність випадкової колізії (p)		
		1%	25%	50%
16	$6,6*10^{4}$	36	$1,9*10^{2}$	$3,0*10^{2}$
32	$4,3*10^{9}$	$9,3*10^{3}$	$5,0*10^{4}$	$7,7*10^{4}$
64	$1,8*10^{19}$	$6,1*10^{8}$	$3,3*10^{9}$	$5,1*10^{9}$
128	$3,4*10^{38}$	$2,6*10^{18}$	$1,4*10^{19}$	$2,2*10^{19}$
256	$1,2*10^{77}$	$4,8*10^{37}$	$2,6*10^{38}$	$4,0*10^{38}$
384	$3,9*10^{115}$	$8,9*10^{56}$	$4,8*10^{57}$	$7,4*10^{57}$
512	$1,3*10^{154}$	$1,6*10^{76}$	$8,8*10^{76}$	$1,4*10^{77}$

Продовження таблиці 2.3

Бітів	Приблизна можлива кількість виходів(H)	Приблизна ймовірність випадкової колізії (p)
		75%
16	$6,6*10^{4}$	$4,3*10^{2}$
32	$4,3*10^{9}$	$1,1*10^{5}$
64	$1,8*10^{19}$	$7,2*10^{9}$
128	$3,4*10^{38}$	$3,1*10^{19}$
256	$1,2*10^{77}$	$5,7*10^{38}$
384	$3,9*10^{115}$	$1,0*10^{58}$
512	$1,3*10^{154}$	$1,9*10^{77}$

Дуже значимим для криптографічних хеш-функцій є те, що при щонайменшій зміні аргументу значення функції істотно змінювалося (лавинний ефект). До того ж, значення хешу не має давати витоку інформації, навіть про окремі біти аргументу. Ця вимога є запорукою криптостійкості алгоритмів хешування, що хешують пароль користувача для отримання ключа.

Хешування часто застосовується в алгоритмах електронно-цифрового підпису, де не саме повідомлення шифрується, а лише його хеш-код, що зменшує час обчислення, а також підвищує криптостійкість. Також в більшості випадків, значення їх хеш-кодів замість паролів зберігаються.

2.11.2 Геометричне хешування

Геометричне хешування (англ. Geometric hashing) широко застосовується у комп'ютерній графіці й обчислювальній геометрії як метод для вирішення завдань на площині або в тривимірному просторі, наприклад, для знаходження однакових зображень або для пошуку найближчих пар в множині точок. Хеш-функція в

даному методі зазвичай одержує на вхід якийсь метричний простір і відокремлює його, утворюючи сітку з клітин. Таблиця у цьому випадку є масивом з двома або більше індексами, яка має назву файл сітки (англ. Grid file). Геометричне хешування також використовується в телекомунікаціях під час роботи з багатовимірними сигналами.

2.11.3 Прискорення пошуку даних

Хеш-таблиця є структурою даних, що дає можливість зберігати пари виду (хеш-код, ключ) і підтримує операції пошуку, вставки та видалення елементів. Завданням хеш-таблиць являється прискорення пошуку, наприклад, у випадку записів до текстових полів в базі даних може обчислюватись їх хеш код і дані можуть розміщуватись у розділі, що відповідає цьому хеш-коду. В такому випадку, при пошуку даних слід спочатку обчислити хеш-код тексту, а потім відразу стане зрозуміло, в якому розділі їх треба шукати, тобто, шукати треба буде не по всій базі, а лише по одному її розділу (це істотно прискорює пошук).

Розміщення слів у словнику за алфавітом вважається побутовим аналогом хешування в даному випадку. Перша літера слова є його хеш-кодом, і при пошуку переглядаємо не весь словник, а лише потрібну літеру.

2.12 Open SSL

OpenSSL являється відкритим програмним продуктом, який був розроблений як універсальна бібліотека для криптографії, що використовує протоколи Transport Layer Security та Secure Sockets Layer. Застосовується, зокрема, в бібліотеці cUrl для здійснення роботи за протоколом HTTPS. Є доступним для більшості UNIX-подібних операційних систем (включаючи Linux, Solaris або OpenSolaris, QNX4, Mac OS X, QNX6 і чотирьох операційних систем BSD з відкритим сирцевим кодом), а також для Microsoft Windows та OpenVMS.

Реалізує наступні алгоритми шифрування:

- шифри: SEED, IDEA, AES, Blowfish, CAST-128, DES, RC2, RC4, RC5, GOST 28147-89, Triple DES, Camellia;
- криптографічні хеш-функції: MD5, MD2, SHA-1, SHA-2, RIPEMD-160, MDC-2, GOST R 34.11-94;
- асиметричні алгоритми шифрування: DSA, RSA, Еліптична криптографія, Протокол Діффі-Геллмана, GOST R 34.10-2001.

2.13 SHA-256

SHA-256 є односпрямованою функцією для створення цифрових відбитків фіксованої довжини (256 біт, 32 байт) з вхідних даних розміром до 2,31 ексабайт (2^{64} біт) і являється окремим випадком алгоритму з сімейства криптографічних алгоритмів SHA-2 (Secure Hash Algorithm Version 2), що були опубліковані АНБ США в 2002 році.

Хеш-функції сімейства SHA-2 створені на основі структури Меркле-Дамгарда.

Оригінал тексту після доповнення поділяється на блоки, кожен блок - на 16 слів. Алгоритм пропускає кожен блок повідомлення через цикл з 64 ітераціями. На кожній ітерації 2 слова перетворюються, функцію перетворення задають інші слова. Результати обробки кожного блоку складаються, сума являє собою значення хеш-функції. Немає можливості обробляти блоки паралельно, тому як проводиться ініціалізація внутрішнього стану результатом обробки попереднього блоку.

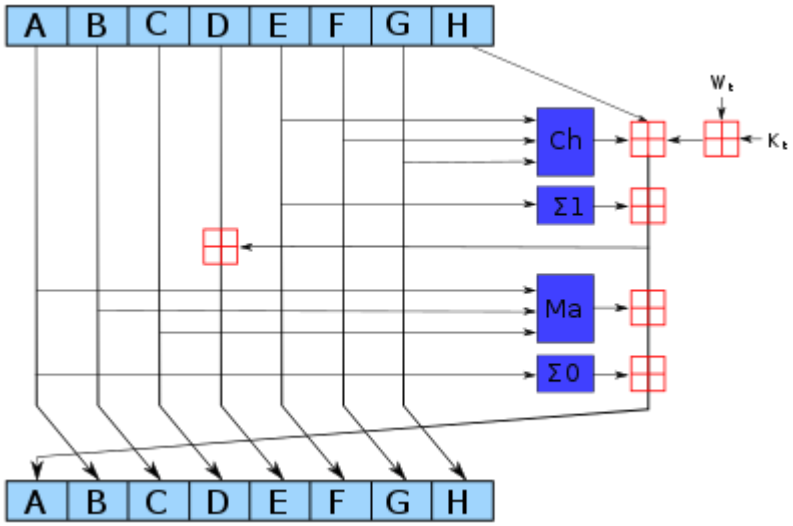


Рисунок 2.10 – Одна ітерація обробки блоку даних

Алгоритм використовує такі бітові операції:

- ‖ — Конкатенація,
- + — Додавання,
- And — Побітове «І»,
- Or — Побітове «АБО»,
- Xor — Виключне «АБО»,
- Shr (Shift Right) — Логічний зсув вправо,
- Rotr (Rotate Right) — Циклічний зсув вправо.

Характеристики SHA-256:

- довжина дайджесту повідомлення є 256 біт;
- довжина внутрішнього стану є 256 біт;
- довжина блоку становить 512 біт;
- максимальна довжина повідомлення складає $2^{64} - 1$ біт;
- довжина слова є 32 біт;
- кількість ітерацій в циклі 64 біт.

Наразі відомі методи для побудови колізій до 31 ітерації. З огляду на алгоритмічної схожості SHA-2 з SHA-1 і наявності в останньої потенційних

вразливостей прийнято рішення, що SHA-3 буде базуватися на зовсім іншому алгоритмі.

Висновки до розділу

У даному розділі проаналізовані рішення, які були обрані нами для побудови системи захисту програми від користувацького втручання.

Для передачі даних між сервером ліцензування та користувачем вирішено використати 3g модем як той, що може підключатися як флеш накопичувач для компактності, та не буде працювати у мережі з іншими підключеннями із-за потенційної можливості перехоплення у домашній мережі хеш значення.

Передача буде відбуватися за протоком TCP із стеку протоколів TCP/IP із-за того, що обраний нами протокол гарантує передачу даних без втрат. Це досягається тим, що при передаванні пакету відбувається запит на перевірку, чи дійшов пакет до місця призначення, якщо ні – пакет надсилається знову. Це є перевагою для нас на відміну від протоколу UDP, що наявний у даному стеку протоколів та не передає дані у цілісному вигляді, орієнтуючись більше на швидкість передачі.

Також із оптимальних кодів між кодами Шенона-Фано та Хаффмана – ми обираємо останній, так як він на відміну від першого жадібний до використання пам’яті, що крім кодування також і зможе стискати дані. Кодове дерево коду Хаффмана і буде нашим ключем кодування.

Зовнішні бібліотеки CImg.h та Libavcodec ми будемо використовувати для того, щоб перетворити зображення та відеофайли у такий вигляд, щоб можна було зручно кодувати ці дані кодом Хаффмана.

Хеш функцію SHA-256 ми використовуємо для перевірки на зміни закодованого файлу, як додатковий елемент захисту. Якщо закодоване повідомлення буде змінено – програма не буде працювати. До того дана хеш функція успішно використовується в блокчейні, маючи дуже багато перетворень, які роблять малоймовірним появу колізії. Для зручності, ми будемо

використовувати у програмі алгоритм SHA-256 наданою бібліотекою OpenSSL, що є відкритою бібліотекою з багатьма рішеннями для кодування та шифрування даних.

Для конфіденційності передача даних відбувається через VPN, а на стороні сервера створюється спеціальне налаштування роутеру, де блокуються усі вхідні та вихідні підключення крім тих, що приходять зі сторони клієнта і саме для звернення до бази даних для отримання хеш суми.

Використано пристрій, у даному випадку модем, який є необхідним для захищеного з’єднання з сервером ліцензування. Застосовано алгоритми кодування, такі як код Хаффмана та SHA-256 хешування, використано системний час як метод захисту від конкретного втручання та використання зовнішні бібліотеки, з метою розширення можливості захисту, окрім текстових, файлів на зображення та відеофайли відповідно.

Наступний розділ буде присвячений розробці системи захисту програми від користувацького втручання та підтвердження роботи поданої системи.

3 ПРОЕКТУВАННЯ СИСТЕМИ ЗАХИСТУ ПРОГРАМИ ВІД КОРИСТУВАЦЬКОГО ВТРУЧАННЯ

3.1 Функціональна схема системи захисту програми

3.1.1 Кодування

Структурна схема системи захисту програми від користувацького втручання зображено для кодування та запуску програми розробника-новачка зображена у додатку.

Для початку ми відкриваємо кодер та вказуємо йому які файли нам слід закодувати для захисту. Це має бути обрано розробником-початківцем. Після цього починається відлік часу, який буде записано в окремий файл.

Далі починається процес кодування кодом Хаффмана. Якщо це текстовий файл, то одразу починається кодування кодом Хаффмана, результат якого збережеться окремо у файлі із форматом «.co».

На рисунку 3.1 показано алгоритм кодування повідомлення для тексту, зображення та відеофайла.

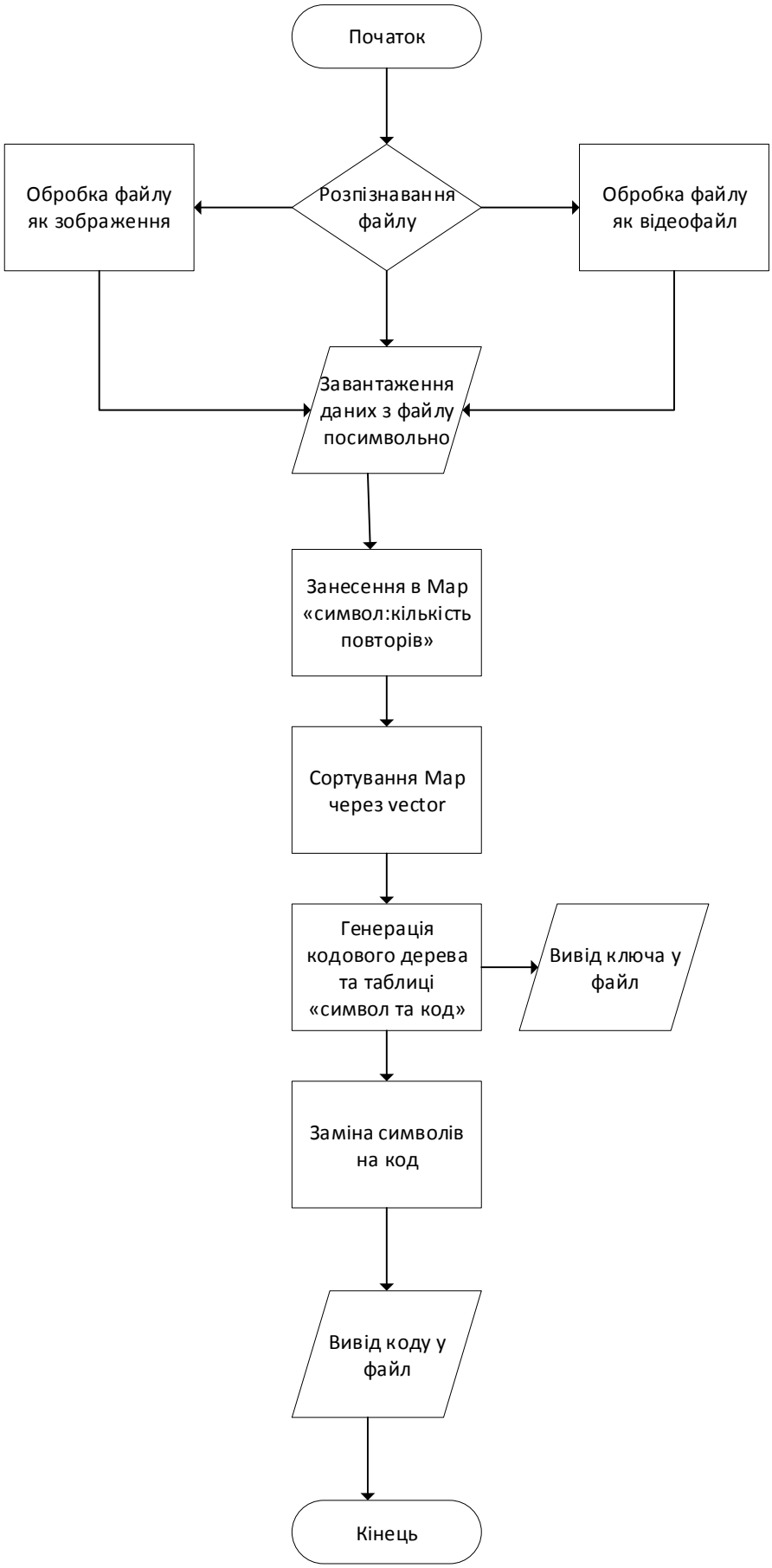


Рисунок 3.1 - Алгоритм кодування файлу кодом Хаффмана

Для роботи із зображеннями тут нам як раз знадобиться зовнішня бібліотека CImg.h для декодування зображення попіксельно у масив даних за схемою «Червоний-Зелений-Синій-Прозорість» (RGBA), і тільки потім кодується за кодом Хаффмана, результат зберігається в окремому файлі.

Для роботи із відеофайлом нам знадобиться зовнішня бібліотека libavcodec, що є частиною пакету FFmpeg, за допомогою якої ми декодуємо відеофайл як масив зображень за допомогою зовнішнього кодера\декодера ffmpeg. Після цього ми працюємо з цим масивом зображень за допомогою зовнішньої бібліотеки CImg.h і повторюємо обробку даних як це було у випадку із зображеннями. Звуковий файл ми зберігаємо окремо як масив із записаними в нього частотами. І лише після усього цього усі ці дані програма кодує окремо у файл за допомогою коду Хаффмана.

Після процесу кодування в окремий файл під назвою key.txt записується ключ разом із згенерованим кодом для вставки у проект програмісту-початківця.

Далі обчислюється хеш сума SHA-256 закодованого файлу та отримується наше значення, що теж зберігається в окремому файлі. Відлік системного часу завершується та результат зберігається окремо у файлі.

Після цього вимикаються усі мережеві адаптери крім нашого спеціального для роботи з 3g модемом. Та на сервері створюється запит на створення нового значення в базі даних, де записується хеш сума для подальшого звернення разом з ім’ям файлу та датою створення закодованого файлу кодом Хафмана. Передача відбувається через протокол TCP/IP через тунель VPN з доступом тільки до серверу бази даних. Після цього ми закриваємо з’єднання та повертаємо усі відключені мережеві адаптери.

3.1.2Декодування

Структурна схема системи захисту програми від користувацького втручання для декодування та запуску програми розробника-новачка зображена у додатку.

Розглянемо схему декодування та її елементи більш точно.

Спочатку ми відкриваємо наш виконуваний файл програми розробника-новачка формату EXE.

Далі вимикаються усі мережеві адаптери, крім нашого спеціального для 3g модему, та створюється запит, що передається до серверу ліцензування за протоколом TCP/IP з використанням VPN тунелю із доступом тільки до серверу ліцензування. Це необхідно для того, щоб ідентифікувати за допомогою хеш суми нашу програму.

Після цього ми із серверу ліцензування отримуємо ключ та закриваємо з'єднання через TCP/IP протокол через тунель VPN. Тільки після цього повертаємо вимкнені мережеві адаптери. Алгоритм даної операції зображено на рисунку 3.2.

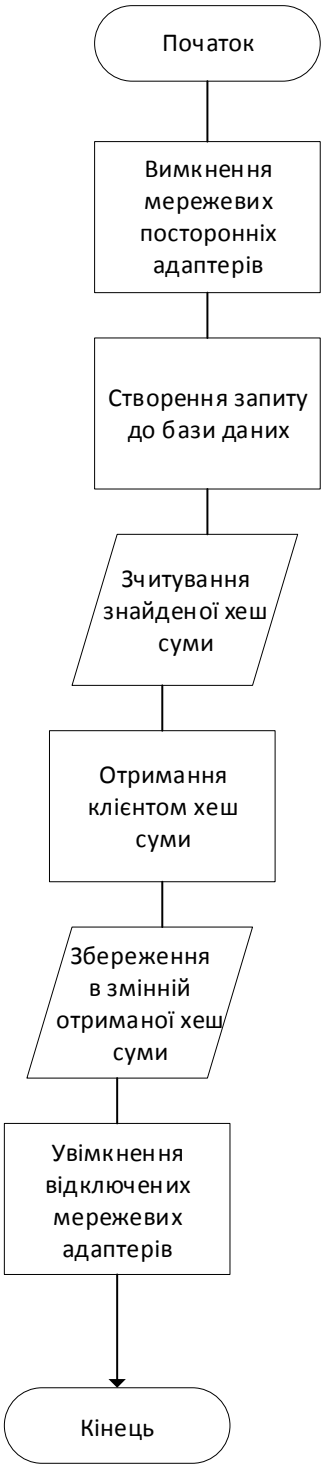


Рисунок 3.2 – Алгоритм отримання ключа через 3g роутер

Далі починається відлік системного часу, який необхідний для того, щоб запобігти можливості робити переривання програми для її подальшого вивчення зловмисником.

Після даної операції ми починаємо операцію взяття хеш суми SHA-256

закодованого файлу та порівнюємо його з отриманим хешем із нашого серверу ліцензування. Якщо він однаковий, то програма захисту продовжує роботу, якщо ні – то виконується аварійне завершення з помилкою. Це допомагає виявити зміни проектних файлів, якщо зловмисник тим чи іншим чином таки отримав ключ кодування та змінив ці дані.

Наступний крок це декодування проектного файлу за допомогою коду Хаффмана, ключ який зберігається в пам’яті виконуваного файлу програміста-початківця, і тільки після цих дій відлік системного часу припиняється та починає виконуватися власне програма розробника-початківця за умовою, якщо програма не переривалася зловмисником з метою вивчення поведінки програми захисту. Ця перевірка відбувається за допомогою порівняння часу, який отримано на момент кодування та отримання хешу, та що зберігається у пам’яті виконуваного файлу.

3.2 Розрахунок коду Хаффмана

3.2.1 Кодування

Кодування Хаффмана є простим алгоритмом для побудови кодів змінної довжини, що мають мінімальну середню довжину.

Алгоритм починається складанням списку символів алфавіту в порядку убування їх ймовірностей. Потім від кореня будується дерево, листям якого служать ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символу з найменшими ймовірностями, додаються наверх часткового дерева, будуть видалені зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, обраних на цьому кроці символів. Коли список скорочується до одного допоміжного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву і побудовою кодів всіх символів.

Нижче буде покроково продемонстрований приклад кодування повідомлення «Привіт! У мене все добре, а у вас?» не враховуючи пробіли, великі та малі літери та знаки:

- Виписуємо усі літери та рахуємо ймовірність їх появи. Усього у нашому повідомленні 24 символи. Будуємо таблицю з кількістю появ символі у повідомлені та ймовірність появи цих символів.

Таблиця 3.1 – Ймовірності появ літер у повідомленні

Літери	Кількість появ n_i	Імовірності $P(a_i)$
П	1	0,04
Р	2	0,08
И	1	0,04
В	3	0,13
І	1	0,04
Т	1	0,04
У	2	0,08
М	1	0,04
Е	4	0,17
Н	1	0,04
С	2	0,08
Д	1	0,04
О	1	0,04
Б	1	0,04
А	2	0,08

- Будуємо таблицю ймовірностей у порядку спадання кожної літери та створюємо допоміжні символи для кожної літери, через які ми будемо проводити «об’єднання» символів;

Таблиця 3.2 – Допоміжні символи та ймовірності у порядку спадання

Символи a_i	Літери	Імовірності $P(a_i)$
a_1	Е	0,17
a_2	В	0,13
a_3	Р	0,08
a_4	У	0,08
a_5	С	0,08
a_6	А	0,08
a_7	П	0,04
a_8	И	0,04
a_9	І	0,04
a_{10}	Т	0,04
a_{11}	М	0,04
a_{12}	Н	0,04
a_{13}	Д	0,04
a_{14}	О	0,04
a_{15}	Б	0,04

- Будуємо кодове дерево та з кожним кроком об’єднуємо два символи найменшої ймовірності в один до тих пір, поки ймовірність не буде дорівнювати одиниці. Нижні гілки умовно позначимо як 0, а верхні тоді позначимо як 1 для подальшої генерації коду за допомогою кодового дерева;

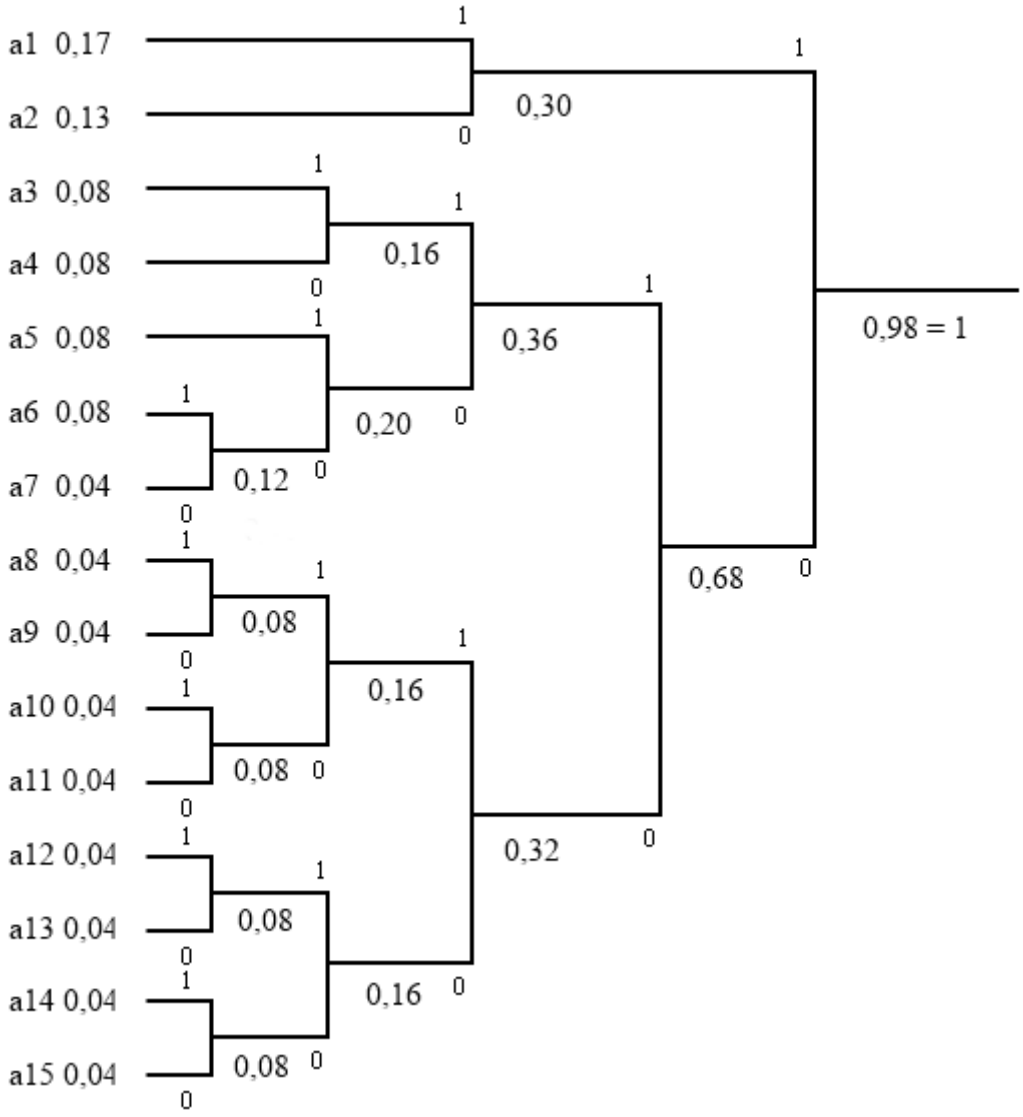


Рисунок 3.3 – Кодове дерево з використанням даних із таблиці 3.2.

- Виписуємо у таблицю код із нулів та одиниць для кожного символу, як результат побудованого дерева. Код визначаємо за вітками кодового дерева.

Таблиця 3.3 – Закодовані символи

Символи a_i	Літери	Код
a_1	Е	11
a_2	В	10
a_3	Р	0111
a_4	У	0110

Продовження таблиці 3.3

a_5	С	0101
a_6	А	01001
a_7	П	01000
a_8	И	00111
a_9	І	00110
a_{10}	Т	00101
a_{11}	М	00100
a_{12}	Н	00011
a_{13}	Д	00010
a_{14}	О	00001
a_{15}	Б	00000

Так як код Хаффмана є префіксним, то в жодному із 15 символів ми не отримаємо повторень при об’єднанні ми не отримаємо жодним чином код іншого символу, тому нам взагалі не треба використовувати розділення кодової послідовності щоб зрозуміти, який символ закодовано.

Із даних, отриманих у таблиці вище, ми можемо тепер закодувати наше повідомлення «Привіт! У мене все добре, а у вас?» не враховуючи знаки, пробіл, велика чи мала літера от так:

Повідомлення: «привітуменевседобреаувас»

Закодоване повідомлення:

«0100001110011110001100010101100010011000111110010111000100000100
000011111010010110100100101010»

Згенерований ключ декодування разом із кодом для вставки можна побачити на рисунку 3.4. Розробник початківець має додати цей код разом із логікою у свій проект для захисту своєї програми. Також там описано функцію декодування для відкриття програми користувачем за умови, якщо хеш сума закодованого файлу і хеш сума що надіслана із серверу рівні.

Також по між цього, на рисунку 3.5 можна побачити згенерований файл закодованого повідомлення «привітуменевседобреаувас».

```
class coder_pr
{
private:
    //here generated key
    std::key_buff< char, int >  =  {{'Е', 11}, {'В', 10}, {'Р', 0111},
                                   {'У', 0110}, {'С', 0101}, {'А', 01001},
                                   {'П', 01000}, {'И', 00111}, {'І', 00110},
                                   {'Т', 00101}, {'М', 00100}, {'Н', 00011},
                                   {'Д', 00010}, {'О', 00001}, {'Б', 00000}, };
    // check server's hash with hash what generated before launch programm
    std::string sts;
    std::string get.Key()
    {
        ifstream co("sample_file.co");
        std::string to_find;
        char character;
        while (!co.eof()) // прочитали его и заполнили им строку
        {
            co.get(character);
            sts.push_back(character);
        }
        co.close();
        for (auto iter = key_buff.begin(); iter != key_buff.end(); ++iter)
        {
            to_find = std::to_string(iter->second);
            word = std::to_string(iter->first);
            while (possition == -1)
            {
                int possition = sts.to_find(to_find);
                std::string progress_rep = sts;
                progress_rep.replace(possition-1,possition+to_find.length(),word);
                sts = progress_rep;
            }
        }
        return sts;

    }
public:
```

Рисунок 3.4 – Згенерований файл з ключем та кодом, що необхідно додати у проект розробника-початківця

```
0100001110011110001100010101100010011000111110010111000100000100000011111010010110100100101
```

Рисунок 3.5 – Згенерований файл із закодованим повідомленням

3.2.2 Декодування

Для того, щоб декодувати закодоване кодом Хаффмана повідомлення нам треба використати таблицю символів та кодів вище та замінити код із нулів та одиниць на символи назад. Ми це можемо зробити без жодних проблем, так як код Хаффмана префіксний.

Повідомлення:

«01000011100111100011000101011000100110001111100101110001000001000000111110100101101001001010101»

Декодоване повідомлення: «привітуменевседобреаувас»

3.3 Розрахунок хешу SHA-256

SHA-256 працює подібно до MD4, MD5, та SHA-1: Повідомлення, яке хешується має доповнювати це ж повідомлення так, щоб результат був кратний 512 бітам, а потім зібрати у блоки повідомлень по 512-біт M(1), M(2),…,M(N) .

Блоки повідомлень обробляються по одному в один проміжок часу: Починаючи з фіксованого початкового значення хешу H(0) , послідовно обчислюючи

$$H^{(i)} = H^{(i-1)} + C_{M^{(i)}}(H^{(i-1)}), \quad (3.1)$$

де *C* це SHA-256 функція стиснення і “+” означає додавання за модулем 2^{32}. $H^{(N)}$ це хеш від *M*.

Розрахунок хешу повідомлення починається з підготовки повідомлення:

- Переносимо повідомлення звичайним способом: Припустимо довжина повідомлення M, у бітах, це *l*. Додавши біт “1” у кінець повідомлення, і потім *k* нульових бітів, де *k* найменше невід'ємне рішення рівняння $l + 1 + k \equiv 448 \, mod \, 512$. До цього додаємо 64-бітний блок, який дорівнює номеру *l*, що написано у двійковій системі числення. Для прикладу, 8-бітне ASCII повідомлення “abc” має довжину 8 * 3 = 24, так що він доповнений одиницею, потім 448 - (24 + 1) = 423 це нульові біти, і потім його довжина перетвориться на 512-бітне повідомлення.

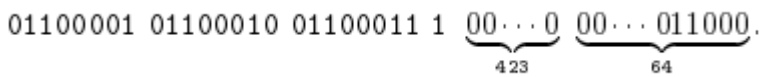


Рисунок 3.6 – Підготовлене до хешування 512 бітне повідомлення, яке тепер кратне 512 біт.

- Розбиваємо повідомлення на N 512-бітних блоків $M^{(1)}$, $M^{(2)}$, … , $M^{(N)}$. Перші 32 біти блоку з повідомленням *i* позначимо як $M_0^{(i)}$, наступні 32 біти як $M_1^{(i)}$, і так далі аж до $M_{15}^{(i)}$. Ми використовуємо найбільш значущу умовність, тому в кожному 32-бітному слові лівий біт зберігається в найбільш значущому положенні біта.

Обчислення хешу виконується наступним чином:

Ми створюємо цикл від одиниці до максимального числа блоків у доповненому повідомленні (N). Всередині циклу ми ініціалізуємо регістри *a, b, c, d, e, f, g, h* із *(i-1)*-им проміжним хеш-значенням, що дорівнює початковому значенню хешу, якщо $i = 1$. Далі ми у ці регістри записуємо значення хешу від повідомлення минулої операції ($N^{(i-1)}$).

Далі ми застосовуємо функцію стиснення SHA-256 для оновлення регістрів a, b,…, h, створивши новий цикл всередині існуючого, що рахує кількість кроків (64 у даному випадку). В середині вже даного циклу ми рахуємо функції $Ch(e,f,g), Maj(a,b,c), \sum_0(a), \sum_1(e), W_j$, де

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z) \quad (3.2)$$

$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \quad (3.3)$$

$$\sum\nolimits_0(x) = S^2(x) \oplus S^{13}(x) \oplus S^{22}(x) \quad (3.4)$$

$$\sum\nolimits_1(x) = S^6(x) \oplus S^{11}(x) \oplus S^{25}(x) \quad (3.5)$$

$$\sigma_0(x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x) \quad (3.6)$$

$$\sigma_1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x) \quad (3.7)$$

Потім ми виходимо з циклу у циклі та обчислюємо *i*-ті проміжні хеш-значення $H^{(i)}$, додавши *(i-1)*-ті проміжні хеш-значення $H^{(i-1)}$ до регістрів.

І потім ми отримуємо вираз, що є хеш від *M*.

$$H^{(N)} = (H_1^{(N)}, H_2^{(N)}, \dots, H_8^{(N)}) \tag{3.8}$$

Розширені блоки повідомлень $W_0, W_1, \dots, W_{63}$ обчислюються наступним чином за допомогою таблиці повідомлень SHA-256:

$$W_j = M_j^{(i)}, \tag{3.9}$$

де для j від 0 до 15 включно, а для j від 16 до 63 включно буде виконуватися така рівність:

$$W_j \leftarrow \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16} \tag{3.10}$$

Послідовність постійних слів, $K_0, \dots, K_{63}$ використано у SHA-256. В шістнадцятирічній системі вони визначаються як:

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

Рисунок 3.7 – Перші тридцять два біти дробової частини коренів куба перших шістдесяти чотирьох простих чисел

Закодувавши повідомлення «abc», і виходячи із вказаного вище тексту ми отримаємо наступні дані у рисунку 3.8.

Далі ми рахуємо H_i, виконуючи операцію додавання за модулем першого та останнього елементів блоку даних кожного регістру і отримуємо хеш повідомлення «abc».

H_1 = 6a09e667 + 506e3058 = ba7816bf

H_2 = bb67ae85 + d39a2165 = 8f01cfea

H_3 = 3c6ef372 + 04d24d6c = 414140de

H_4 = a54ff53a + b85e2ce9 = 5dae2223

H_5 = 510e527f + 5ef50f24 = b00361a3

H_6 = 9b05688c + fb121210 = 96177a9c

H_7 = 1f83d9ab + 948d25b6 = b410ff61

H_8 = 5be0cd19 + 961f4894 = f20015ad

Виходячи з цього ми отримали результат хешування 24-бітного повідомлення “abc”:

«ba7816bf 8f01cfea 414140de 5dae2223 b00361a3 96177a9c b410ff61 f20015ad».

У шістнадцятирічній системі числення хеш буде виглядати ось так:

«61626380 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000018»

	a	b	c	d	e	f	g	h
init:	6a09e667	bb67ae85	3c6ef372	a54ff53a	510e527f	9b05688c	1f83d9ab	5be0cd19
t = 0	5d6aebcd	6a09e667	bb67ae85	3c6ef372	fa2a4622	510e527f	9b05688c	1f83d9ab
t = 1	5a6ad9ad	5d6aebcd	6a09e667	bb67ae85	78ce7989	fa2a4622	510e527f	9b05688c
t = 2	c8c347a7	5a6ad9ad	5d6aebcd	6a09e667	f92939eb	78ce7989	fa2a4622	510e527f
t = 3	d550f666	c8c347a7	5a6ad9ad	5d6aebcd	24e00850	f92939eb	78ce7989	fa2a4622
t = 4	04409a6a	d550f666	c8c347a7	5a6ad9ad	43ada245	24e00850	f92939eb	78ce7989
t = 5	2b4209f5	04409a6a	d550f666	c8c347a7	714260ad	43ada245	24e00850	f92939eb
t = 6	e5030380	2b4209f5	04409a6a	d550f666	9b27a401	714260ad	43ada245	24e00850
t = 7	85a07b5f	e5030380	2b4209f5	04409a6a	0c657a79	9b27a401	714260ad	43ada245
t = 8	8e04ecb9	85a07b5f	e5030380	2b4209f5	32ca2d8c	0c657a79	9b27a401	714260ad
t = 9	8c87346b	8e04ecb9	85a07b5f	e5030380	1cc92596	32ca2d8c	0c657a79	9b27a401
t = 10	4798a3f4	8c87346b	8e04ecb9	85a07b5f	436b23e8	1cc92596	32ca2d8c	0c657a79
t = 11	f71fc5a9	4798a3f4	8c87346b	8e04ecb9	816fd6e9	436b23e8	1cc92596	32ca2d8c
t = 12	87912990	f71fc5a9	4798a3f4	8c87346b	1e578218	816fd6e9	436b23e8	1cc92596
t = 13	d932eb16	87912990	f71fc5a9	4798a3f4	745a48de	1e578218	816fd6e9	436b23e8
t = 14	c0645fde	d932eb16	87912990	f71fc5a9	0b92f20c	745a48de	1e578218	816fd6e9
t = 15	b0fa238e	c0645fde	d932eb16	87912990	07590dcd	0b92f20c	745a48de	1e578218
t = 16	21da9a9b	b0fa238e	c0645fde	d932eb16	8034229c	07590dcd	0b92f20c	745a48de
t = 17	c2fbd9d1	21da9a9b	b0fa238e	c0645fde	846ee454	8034229c	07590dcd	0b92f20c
t = 18	fe777bbf	c2fbd9d1	21da9a9b	b0fa238e	cc899961	846ee454	8034229c	07590dcd
t = 19	e1f20c33	fe777bbf	c2fbd9d1	21da9a9b	b0638179	cc899961	846ee454	8034229c
t = 20	9dc68b63	e1f20c33	fe777bbf	c2fbd9d1	8ada8930	b0638179	cc899961	846ee454
t = 21	c2606d6d	9dc68b63	e1f20c33	fe777bbf	e1257970	8ada8930	b0638179	cc899961
t = 22	a7a3623f	c2606d6d	9dc68b63	e1f20c33	49f5114a	e1257970	8ada8930	b0638179
t = 23	c5d53d8d	a7a3623f	c2606d6d	9dc68b63	aa47c347	49f5114a	e1257970	8ada8930
t = 24	1c2c2838	c5d53d8d	a7a3623f	c2606d6d	2823ef91	aa47c347	49f5114a	e1257970
t = 25	cde8037d	1c2c2838	c5d53d8d	a7a3623f	14383d8e	2823ef91	aa47c347	49f5114a
t = 26	b62ec4bc	cde8037d	1c2c2838	c5d53d8d	c74c6516	14383d8e	2823ef91	aa47c347
t = 27	77d37528	b62ec4bc	cde8037d	1c2c2838	edffbff8	c74c6516	14383d8e	2823ef91
t = 28	363482c9	77d37528	b62ec4bc	cde8037d	6112a3b7	edffbff8	c74c6516	14383d8e
t = 29	a0060b30	363482c9	77d37528	b62ec4bc	ade79437	6112a3b7	edffbff8	c74c6516
t = 30	ea992a22	a0060b30	363482c9	77d37528	0109ab3a	ade79437	6112a3b7	edffbff8
t = 31	73b33bf5	ea992a22	a0060b30	363482c9	ba591112	0109ab3a	ade79437	6112a3b7
t = 32	98e12507	73b33bf5	ea992a22	a0060b30	9cd9f5f6	ba591112	0109ab3a	ade79437
t = 33	fe604df5	98e12507	73b33bf5	ea992a22	59249dd3	9cd9f5f6	ba591112	0109ab3a
t = 34	a9a7738c	fe604df5	98e12507	73b33bf5	085f3833	59249dd3	9cd9f5f6	ba591112
t = 35	65a0cfe4	a9a7738c	fe604df5	98e12507	f4b002d6	085f3833	59249dd3	9cd9f5f6
t = 36	41a65cb1	65a0cfe4	a9a7738c	fe604df5	0772a26b	f4b002d6	085f3833	59249dd3
t = 37	34df1604	41a65cb1	65a0cfe4	a9a7738c	a507a53d	0772a26b	f4b002d6	085f3833
t = 38	6dc57a8a	34df1604	41a65cb1	65a0cfe4	f0781bc8	a507a53d	0772a26b	f4b002d6
t = 39	79ea687a	6dc57a8a	34df1604	41a65cb1	1efbc0a0	f0781bc8	a507a53d	0772a26b
t = 40	d6670766	79ea687a	6dc57a8a	34df1604	26352d63	1efbc0a0	f0781bc8	a507a53d
t = 41	df46652f	d6670766	79ea687a	6dc57a8a	838b2711	26352d63	1efbc0a0	f0781bc8
t = 42	17aa0dfe	df46652f	d6670766	79ea687a	decd4715	838b2711	26352d63	1efbc0a0
t = 43	9d4baf93	17aa0dfe	df46652f	d6670766	fda24c2e	decd4715	838b2711	26352d63
t = 44	26628815	9d4baf93	17aa0dfe	df46652f	a80f11f0	fda24c2e	decd4715	838b2711
t = 45	72ab4b91	26628815	9d4baf93	17aa0dfe	b7755da1	a80f11f0	fda24c2e	decd4715
t = 46	a14c14b0	72ab4b91	26628815	9d4baf93	d57b94a9	b7755da1	a80f11f0	fda24c2e
t = 47	4172328d	a14c14b0	72ab4b91	26628815	fecf0bc6	d57b94a9	b7755da1	a80f11f0
t = 48	05757ceb	4172328d	a14c14b0	72ab4b91	bd714038	fecf0bc6	d57b94a9	b7755da1
t = 49	f11bfaa8	05757ceb	4172328d	a14c14b0	6e5c390c	bd714038	fecf0bc6	d57b94a9
t = 50	7a0508a1	f11bfaa8	05757ceb	4172328d	52f1ccf7	6e5c390c	bd714038	fecf0bc6
t = 51	886e7a22	7a0508a1	f11bfaa8	05757ceb	49231c1e	52f1ccf7	6e5c390c	bd714038
t = 52	101fd28f	886e7a22	7a0508a1	f11bfaa8	529e7d00	49231c1e	52f1ccf7	6e5c390c
t = 53	f5702fdb	101fd28f	886e7a22	7a0508a1	9f4787c3	529e7d00	49231c1e	52f1ccf7
t = 54	3ec45cdb	f5702fdb	101fd28f	886e7a22	e50e1b4f	9f4787c3	529e7d00	49231c1e
t = 55	38cc9913	3ec45cdb	f5702fdb	101fd28f	54cb266b	e50e1b4f	9f4787c3	529e7d00
t = 56	fcd1887b	38cc9913	3ec45cdb	f5702fdb	9b5e906c	54cb266b	e50e1b4f	9f4787c3
t = 57	c062d46f	fcd1887b	38cc9913	3ec45cdb	7e44008e	9b5e906c	54cb266b	e50e1b4f
t = 58	ffb70472	c062d46f	fcd1887b	38cc9913	6d83bfc6	7e44008e	9b5e906c	54cb266b
t = 59	b6ae8fff	ffb70472	c062d46f	fcd1887b	b21bad3d	6d83bfc6	7e44008e	9b5e906c
t = 60	b85e2ce9	b6ae8fff	ffb70472	c062d46f	961f4894	b21bad3d	6d83bfc6	7e44008e
t = 61	04d24d6c	b85e2ce9	b6ae8fff	ffb70472	948d25b6	961f4894	b21bad3d	6d83bfc6
t = 62	d39a2165	04d24d6c	b85e2ce9	b6ae8fff	fb121210	948d25b6	961f4894	b21bad3d
t = 63	506e3058	d39a2165	04d24d6c	b85e2ce9	5ef50f24	fb121210	948d25b6	961f4894

Рисунок 3.8 – Опрацювання блоку даних повідомлення «abc»

Доказ успішно згенерованого файлу із вставленим хешем зображено на рисунку 3.9. Розробник початківець має додати цей код у свій проект для функціонування захисту.

```
// This hash is generated. Add this code to your main file for protect.

class merge_pr::coder_pr
{
    //here generated SHA-256 hash
    string hs = "ba7816bf 8f01cfea 414140de 5dae2223 b00361a3 96177a9c b410ff61 f20015ad";
    // check server's hash with hash what generated before launch program
    if (hs = check_hs)
    {
        getFile(); //decode with key and get file for opening program
    }
    // example logic what do when encrypted file is changed
    else
    {
        std::cout<<"Your files is changed! You cannot continue use program!"
        abort();
    }
};
```

Рисунок 3.9 – Згенерований файл з логікою порівняння хешу

3.4 Моделювання втручання в систему захисту

Наприклад, якщо якимось чином зловмисник перехопив ключ дешифрування та змінив дані закодованого файлу, дизасемблюючи програмний код, або просто щось змінив всередині, то його очікує неприємний сюрприз, пов’язаний із тим, що він не зможе на одному етапі захисту пройти перевірку хеш суми SHA-256, що отриманий через 3g модем. Це пов’язано з тим, що якщо зміниться хоча б один байт або символ у файлі – хеш, який буде отриманий із зламаного закодованого файлу, стане абсолютно іншим. У такому випадку передбачений аварійний вихід програми з помилкою, що показано на рисунку 3.10.

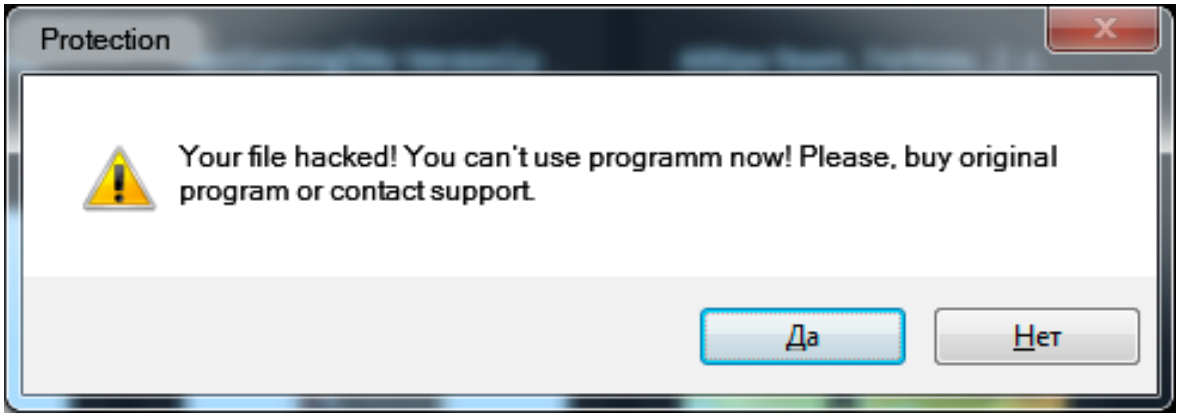


Рисунок 3.10 – Помилка ліцензії, яка виникає під час порівняння хешу

При тому, щоб дістати ключ дешифрування файлу – необхідно також використати дизасемблер, що має дати зловмиснику шанс відновити код, але у такому випадку програма через деякий час припиняє роботу, це означає, що спрацював захист по часу роботи. Він необхідний для того, щоб унеможливити діставання ключів та хешу методом дизасемблювання або переривання роботи програми, так як при кодуванні файлів розробником-початківцем програма захисту також записала окремо час, необхідний для декодування та перевірки хеш суми на рівність.

Також логіка програми не передбачає ситуацію, коли хеш так і не був отриманий саме через спеціально налаштований для цього 3g модем. В такому випадку у нас інша помилка, яка зображена на рисунку 3.12.

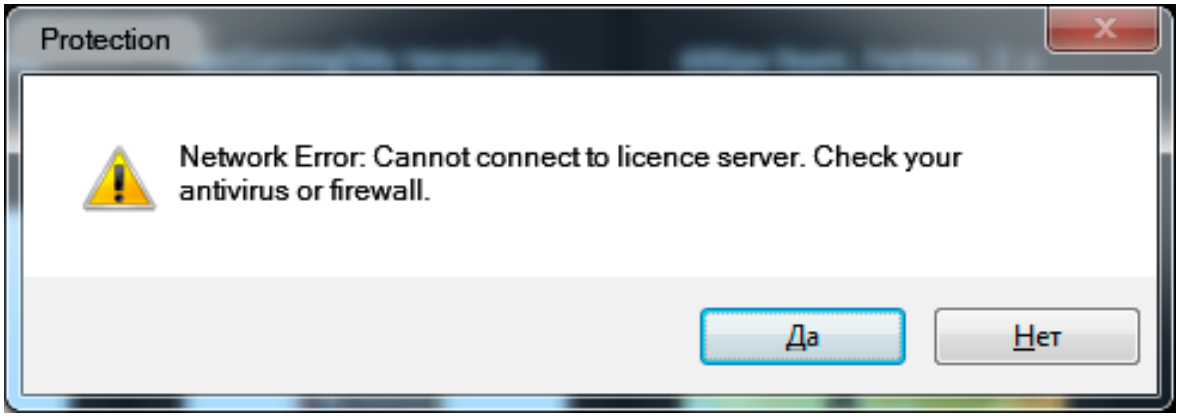


Рисунок 3.11 – Помилка з’єднання з сервером ліцензування для отримання хешу

До того, навіть якщо якимось чином втрутитися у захищене з’єднання і перехопити хеш суму, та використавши її обходу перевірки – це не означає те, що модифікований зловмисником захищений файл буде декодовано, бо тоді буде виявлено, що відновити дані буде неможливо із-за змін у файлі, що теж є доказом роботи системи захисту.

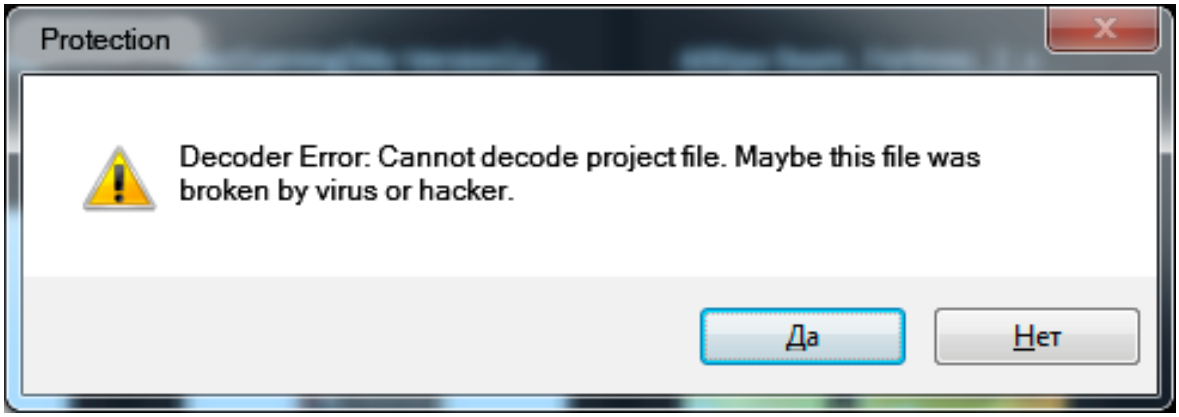


Рисунок 3.12 – Помилка декодування, що означає що закодований файл було змінено і його тепер неможливо декодувати за наявним ключем.

Висновки до розділу

В третьому розділі були показані структурна схема системи шифрування, структурна схема системи дешифрування, функціональні схем системи захисту програми, а саме кодування і декодування.

Також було розроблено спеціальний кодер для проектних файлів для розробника-початківця, який має його запустити, щоб закодувати файли свого проекту.

На виході він отримає закодований файл, файл із згенерованим ключем та його логікою для декодування під час запуску програми, що розроблена розробником-початківцем, файл із логікою перевірки хеш суми, а також файл з виміряним часом кодування та хешування для захисту від переривань.

До того, згенерована хеш сума має відправиться через модем на сервер ліцензування, щоб у випадку, коли звичайний користувач захоче відкрити програму і маючи при тому спеціальний 3g модем, він зміг би отримати з серверу хеш суму для продовження роботи системи захисту програми від користувацького втручання, що проводиться перед запуском програми, яку написав розробник-початківець.

Також у цьому розділі було доведено роботу кодера та декодера за алгоритмом Хаффмана і це порівняно з розрахунком коду Хаффмана вручну, результат яких співпадає, що доводить роботу програми.

Те саме стосується також і розрахунку хеш суми SHA-256, що було складно зробити вручну із-за наявності 64 кроків розрахунку як циклу, підготування повідомлення, з якого необхідно зробити розрахунок хеш суми, яке має бути кратним 512, використовуючи багато різних між собою логічних операцій.

ВИСНОВКИ

У дипломній роботі було поставлено мету спроектувати систему захисту програми від користувацького втручання, зробивши програму розробника-початківця захищеною від впливу користувачів та зловмисників.

У першому розділі перераховано та розглянуто багато методів захисту, наведені їх переваги та недоліки, а також їх можливості.

У другому розділі обрано та проаналізовано методи та зовнішні бібліотеки, а також обрано пристрій, який буде використовуватися для забезпечення захисту програми від користувацького втручання. Усі ці методи поодинці можуть захищати, але на жаль відомо багато способів як зламати усі їх, тому єдиним виходом залишається об'єднати усі ці методи захисту в одну систему для забезпечення кращого захисту.

Розробка системи захисту виконана як раз з врахуванням багатьох методів, що в комплексі дозволить покращити захист програми, і це:

- використання коду Хаффмана для кодування файлів;
- взяття хеш суми SHA-256 для перевірки закодованих файлів на зміни;
- використання строго налаштованого відліку системного часу для унеможливлення несанкціонованого вивчення програми через переривання;
- отримання хешу через мережу від серверу, з яким ми з'єднуємося через спеціально налаштований 3G модем за протоколом TCP/IP.

Результати 3-го розділу присвячені опису та демонстрації роботи системи захисту програми від користувацького втручання. А саме, розроблені структурні схеми, функціональні схеми та алгоритми програм, обрані та описані окремі методи захисту разом із додатковими бібліотеками, що необхідні були для роботи із зображеннями та відеофайлами для подальшої розробки, виконані необхідні розрахунки, що підтверджують результати роботи захисту. Також проведені різного роду втручання в роботу програми, які доказали, що розроблена нами система, на відміну від використання даних методів поодинці, гарно працює,

захищаючи програму від дизасемблювання та переривання, захищає від будь якої зміни закодованого файлу та не дає змогу відкривати закодовані файли напряму через редактор зображень або відеофайлів, так як ці файли закодовані кодом Хаффмана.

Перевагами проекту є комплексні рішення, можливість крім текстових файлів закодувати також зображення та відеофайли, що необхідно для повноти функціоналу програми нашому розробнику початківцю.

До недоліків можна віднести хіба підвищену ціну із-за використання 3g модему для організації спеціального захищеного з'єднання із сервером ліцензування. Також до недоліків можна віднести те, що роблячи такий сильний метод, програма, яку розробив розробник-початківець може запускатися з деякою затримкою.

Результати дипломного проекту можуть використовуватися для комерційного продажу у сфері захисту інформації для різного роду програм та навіть ігор.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Захист програмного забезпечення [Електронний ресурс]: матеріал з Вікіпедії. Доступ: https://uk.wikipedia.org/wiki/Захист_програмного_забезпечення
2. Хеш-функція [Електронний ресурс]: матеріал з Вікіпедії. Доступ: https://en.wikipedia.org/wiki/Hash_function
3. Атака «днів народження» [Електронний ресурс]: матеріал з Вікіпедії. Доступ: https://en.wikipedia.org/wiki/Birthday_attack
4. Електронні системи [Електронний ресурс]; уклад. Й.Й. Білинський, К.В. Огородник, М.Й. Юкиш. Доступ: https://web.posibnyky.vntu.edu.ua/firen/6bilynskyj_elektronni_systemy/index.htm
5. Жураковський Ю.П. Теорія інформації та кодування: Підручник/ Ю.П. Жураковський, В.П. Полторак. – К.: Вища шк., 2001. – 225 с.
6. Й.Й. Білинський Кодування із стисненням інформації / Й.Й. Білинський, К.В. Огородник, М.Й. Юкиш.// [Електронний ресурс]. Доступ: https://web.posibnyky.vntu.edu.ua/firen/6bilynskyj_elektronni_systemy/56.htm
7. Алгоритмы сжатия и компрессии [Электронный ресурс]. Доступ: http://www.compression-pointers.ru
8. Майним Bitcoin с помощью бумаги и ручки [Электронный ресурс]. Доступ: https://habr.com/ru/post/258181/
9. Кодирование Хаффмана [Электронный ресурс]: материал из Научной библиотеки. Доступ: http://sernam.ru/book_sel.php?id=7
10. Как выбрать 3G модем? [Электронный ресурс]. Доступ: https://3gstar.com.ua/vybrat-modem-a-14.html
11. Роутер или модем: что лучше? [Электронный ресурс]. Доступ: http://bezprovodoff.com/wi-fi/oborudovanie/router-ili-modem.html

12. Что такое модем? Виды модемов, принцип функционирования [Электронный ресурс]. Доступ: http://we-it.net/index.php/set/lokalnye-seti/18-chto-takoe-modem
13. Как настроить функцию фильтрации по IP адресам на беспроводном маршрутизаторе TP-Link [Электронный ресурс]. Доступ: https://www.tp-link.com/ru/support/faq/156/
14. Електронні засоби навчання [Електронний ресурс]. Доступ: http://www.znanius.com/3608.html
15. ГОСТ 19.701-90 Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения [Электронный ресурс]. Доступ: https://internet-law.ru/gosts/gos
16. Теория защиты программ от взлома [Электронный ресурс]. Доступ: http://infocity.kiev.ua/hack/content/hack262.phtml
17. Защита программ от взлома [Электронный ресурс]. Доступ: https://z-oleg.com/secur/articles/progprotect.php
18. Как сделать надежную защиту программы [Электронный ресурс]. Доступ: https://dou.ua/forums/topic/2699/

ДОДАТОК А

Лістинг згенерованого коду для розробника-початківця

```
// This key is generated. Add this code to your main file for protect.
//===============
// Add this include files if you do not use it now
#include <fstream>
#include <map>
#include <string>
#include <iostream>
//===============

class coder_pr
{
private:
        //here generated key
        std::key_buff< char, int > = {{'E', 11}, {'B', 10}, {'P', 0111},
                                                {'У', 0110}, {'C', 0101}, {'A', 01001},
                                                {'П', 01000}, {'И', 00111}, {'І', 00110},
                                                {'Т', 00101}, {'М', 00100}, {'Н', 00011},
                                                {'Д', 00010}, {'О', 00001}, {'Б', 00000}, };
        // check server's hash with hash what generated before launch programm
        std::string sts;
        std::string get.Key()
        {
                ifstream co("sample_file.co");
                std::string to_find;
                char character;
                while (!co.eof()) // прочитали его и заполнили им строку
                {
                        co.get(character);
                        sts.push_back(character);
                }
                co.close();
                for (auto iter = key_buff.begin(); iter != key_buff.end(); ++iter)
                {
                        to_find = std::to_string(iter->second);
```

```
                                word = std::to_string(iter-
35                              >first);
36                              while (possition == -1)
37                              {
                                        int possition =
38                                      sts.to_find(to_find);
                                        std::string
39                                      progress_rep = sts;
                                        progress_rep.replace(possition-
40                                      1,possition+to_find.length(),word);
                                        sts =
41                                      progress_rep;
42                              }
43                      }
44                      return sts;
45
46              }
47  public:
48                      std::string getFile { return sts };
49  };
```

ДОДАТОК Б

Згенерована хеш сума для розробника-початківця

```
1  // This hash is generated. Add this code to your main file for protect.
2
3  class merge_pr::coder_pr
4  {
5          //here generated SHA-256 hash
6          string hs = "ba7816bf 8f01cfea 414140de 5dae2223 b00361a3 96177a9c b410ff61 f20015ad";
7          // check server's hash with hash what generated before launch program
8          if (hs = check_hs)
9          {
                                                  getFile(); //decode with key and get file
10                                                 for opening program
11         }
12         // example logic what do when encrypted file is changed
13         else
14         {
                                                  std::cout<<"Your files is changed! You cannot
15                                                 continue use program!"
16                                                 abort();
17         }
18 };
```