МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Пояснювальна записка до дипломного проєкту

магістра
(освітній ступінь)

на тему «Дослідження методів і засобів забезпечення конфіденційності інформації у web середовищі»

ХАІ.603.667п1.121.156325.20О

Виконав: студент 6 курсу групи № 667п1
Спеціальність 121 – Інженерія програмного забезпечення
(код та найменування)
Освітня програма Хмарні обчислення та Інтернет речей
(найменування)
Шаповалов М.Є.
(прізвище й ініціали студента)
Керівник Пудовкіна Л.Ф.
(прізвище та ініціали)
Рецензент Іващенко Г.С.
(прізвище та ініціали)

Харків – 2020

Міністерство світи і науки України
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу
(повне найменування)
Кафедра інженерії програмного забезпечення
(повне найменування)
Рівень вищої освіти другий (магістерський)

Спеціальність 121 – інженерія програмного забезпечення
(код та найменування)
Освітня програма хмарні обчислення та Інтернет речей
(найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (ініціали та прізвище)
“_____” _____ 2020 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Шаповалову Михайлу Євгенійовичу
(прізвище, ім’я, по батькові)

1. Тема дипломного проєкту *Дослідження методів і засобів забезпечення конфіденційності інформації у web середовищі*
керівник дипломного проекту *Пудовкіна Лариса Федорівна, к.т.н, професор*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “__” ___2020 року № __

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи: *прототипу програмного продукту для забезпечення конфіденційності даних у web середовищі*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
огляд та аналіз існуючих методів забезпечення безпеки у web середовищі; розроблення методу для забезпечення шифрування та підтримки цілісності інформації, контролю доступу та перевірки аутентифікації; розроблення програмного додатку для забезпечення конфіденційності інформації у web середовищі

5. Перелік графічного матеріалу
РПЗ – стор. 99, рисунків – 20 шт., таблиць – 10 шт., презентація – 16 слайдів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Пудовкіна Л.Ф., проф. каф. 603		
2	Пудовкіна Л.Ф., проф. каф. 603		
3	Пудовкіна Л.Ф., проф. каф. 603		

8. Нормоконтроль _____ __В.А. Постернакова__ «____» _____2020 р.
(підпис) (ініціали та прізвище)

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1	Отримання і затвердження теми диплому	03.09.2019	
2	Аналіз предметної області	04.09.2019	
3	Постановка задачі	20.11.2019	
4	Проведення теоретичних досліджень	22.11.2019	
5	Розробка прототипу ПЗ	02.09.2020	
6	Підготовка пояснювальної записки	22.10.2020	
7	Оформлення пояснювальної записки до дипломного проекту	10.11.2020	
8	Передзахист дипломного проекту	24.11.2020	
9	Захист дипломного проекту	04.12.2020	

Студент _____ __Шаповалов М.Є.____
(підпис) (прізвище та ініціали)

Керівник роботи _____ __Пудовкіна Л.Ф.__
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту містить 99 стор., 20 рис., 1 додаток, 26 джерел.

Об'єктом дослідження є процес забезпечення конфіденційності інформації у web середовищі.

Предметом дослідження є моделі та методи забезпечення конфіденційності інформації у web середовищі.

Метою роботи є підвищення ефективності безпеки інформації у web середовищі шляхом розроблення прототипу програмного продукту для забезпечення конфіденційності даних у web середовищі за рахунок індивідуального керування параметрами безпеки файлів.

Для досягнення поставленої мети необхідно вирішити наступні задачі: провести огляд та аналіз існуючих методів забезпечення безпеки у web середовищі; розробити метод для забезпечення шифрування та підтримки цілісності інформації, контролю доступу та перевірки аутентифікації; розробити програмний додаток для забезпечення конфіденційності інформації у web середовищі.

Наукова новизна. Удосконалено метод забезпечення конфіденційності інформації у web середовищі який на відміну від існуючих використовує незалежні параметри безпеки, включаючи докази цілісності та аутентичності задля індивідуального керування власником даних, що дає можливість підвищити ефективність безпеки інформації у web середовищі.

Практична значимість отриманих результатів. В результаті був розроблений програмний додаток для забезпечення конфіденційності інформації у web середовищі, який можна використовувати також як криптографічний елемент при комерційній реалізації системи забезпечення безпеки даних.

ШИФРУВАННЯ ДАНИХ, ВЕБ-СЕРЕДОВИЩЕ, КОНФІДЕЦІЙНІСТЬ ДАНИХ, ЦІЛІСНІСТЬ ІНФОРМАЦІЇ, БЕЗПЕКА ДАНИХ

ABSTRACT

Explanatory note to the graduate work contains 99 pp., 20 fig., 1 app., 26 sources.

The object of research is the process of ensuring the confidentiality of information in the web environment.

The subject of the study are models and methods of ensuring the confidentiality of information in the web environment.

The aim of the work is to increase the efficiency of information security in the web environment by developing a prototype software product to ensure the confidentiality of data in the web environment through individual management of file security settings.

To achieve this goal it is necessary to solve the following tasks: to review and analyze existing methods of security in the web environment; develop a method to ensure encryption and maintain the integrity of information, access control and authentication; develop a software application to ensure the confidentiality of information in the web environment.

Scientific novelty. Improved method of ensuring the confidentiality of information in the web environment, which, unlike existing ones, uses independent security parameters, including evidence of integrity and authenticity for individual management of the data owner, which allows to increase the effectiveness of information security in the web environment.

The practical significance of the obtained results. As a result, a software application was developed to ensure the confidentiality of information in the web environment, which can also be used as a cryptographic element in the commercial implementation of data security.

DATA ENCRYPTION, WEB ENVIRONMENT, DATA CONFIDENTIALITY, INFORMATION INTEGRITY, DATA SECURITY

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IaaS - Infrastructure as a Service.

PaaS - Platform as a Service.

SaaS - Software as a Service.

SLA - Service Layer Agreement.

TC - Trusted Computing.

VPC - Virtual Private Cloud.

VPN - Virtual Private Network.

ІОБД - інформаційно-орієнтована безпека даних.

ПЗ – програмне забезпечення.

ПП – програмний продукт.

РАЕ - розширений алгоритм Евкліда.

ТО - теорема про остачі.

ФКД – файл конфедаційних даних.

ВСТУП

Актуальність дослідження новітніх методів забезпечення конфіденційності інформації у web середовищі зумовлено тим, що при розповсюдженні цієї технології в корпоративному сегменті у споживачів виникає низка перешкод, що насамперед включає проблеми безпеки та конфіденційності. На додаток до традиційних ризиків безпеки, що виникають в обчислювальних системах, підключених до Інтернету, вони ще мають також специфічні проблеми безпеки та конфіденційності через віртуалізацію.

Об'єктом дослідження є процес забезпечення конфіденційності інформації у web середовищі.

Предметом дослідження є моделі та методи забезпечення конфіденційності інформації у web середовищі.

Метою роботи є підвищення ефективності безпеки інформації у web середовищі шляхом розроблення прототипу програмного продукту для забезпечення конфіденційності даних у web середовищі за рахунок індивідуального керування параметрами безпеки файлів.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

– провести огляд та аналіз існуючих методів забезпечення безпеки в у web середовищі;

– розробити метод для забезпечення шифрування та підтримки цілісності інформації, контролю доступу та перевірки аутентифікації;

– розробити програмний додаток для забезпечення конфіденційності інформації у web середовищі.

Методи дослідження: методи системного аналізу, методи забезпечення безпеки даних, методи шифрування інформації, методи розробки базуються на мові програмування C#.

Наукова новизна. Удосконалено метод забезпечення конфіденційності інформації у web середовищі який на відміну від існуючих використовує незалежні параметри безпеки, включаючи докази цілісності та аутентичності задля індивідуального керування власником даних, що дає можливість підвищити ефективність безпеки інформації у web середовищі.

Практична значимість отриманих результатів. В результаті був розроблений програмний додаток для забезпечення конфіденційності інформації у web середовищі, який можна використовувати також як криптографічний елемент при комерційній реалізації системи забезпечення безпеки даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Постановка мети й завдань дослідження

Метою роботи є підвищення ефективності безпеки інформації у web середовищі шляхом розроблення прототипу програмного продукту для забезпечення конфіденційності даних у web середовищі за рахунок індивідуального керування параметрами безпеки файлів.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

– провести огляд та аналіз існуючих методів забезпечення безпеки в у web середовищі;

– розробити метод для забезпечення шифрування та підтримки цілісності інформації, контролю доступу та перевірки аутентифікації;

– розробити програмний додаток для забезпечення конфіденційності інформації у web середовищі.

1.2 Загальна мета захисту інформації

Інформацією, що захищається є персональні дані користувачів і їх компаній, всі носії, на яких обробляються дані і процеси, за допомогою яких відбувається обробка.

Забезпечення безпеки веб-додатку передбачає перешкоди для несанкціонованих дій зловмисників, таких як зміни даних, зміни структури веб-додатки, виведення з ладу.

Для формування необхідних умов для забезпечення інформаційної безпеки оброблюваних даних веб-додатком необхідно виявити потенційного зловмисника і скласти модель загроз.

1.2.1 Модель порушника інформаційної безпеки

Модель порушника являє собою якісь дії зловмисників, які здатні реалізувати загрози інформаційної безпеки веб-додатку, їх цілі та послідовність можливих дій. Під порушником розглядається особа або група осіб. Всіх порушників можна розділити на два типи:

– Зовнішні порушники - особи, у яких немає прав доступу в контрольовану зону веб-додатку, де розміщується конфіденційна інформація;

– Внутрішні порушники - особи, у яких є права доступу в контрольовану зону, на якій розміщується конфіденційна інформація. Можливості зловмисника залежать від його посади і прав доступу до конфіденційної інформації.

До зовнішніх порушників можна віднести:

– представників конкуруючих організацій;

– противників різних організацій і проведених ними заходів;

– хакер-одинак;

– об'єднана група хакерів;

– спецслужби різних держав.

Внутрішніми порушниками можуть бути:

– адміністратори веб-додатку або веб-серверу;

– користувачі, що володіють правами адміністраторів, у створеному клієнтами веб-додатку;

– користувачі, що володіють доступом до конфіденційної інформації;

– зареєстровані користувачі веб-додатку, які мають доступ в особистий кабінет;

– користувачі, які заподіюють шкоду ненавмисно.

Виходячи з особливостей функціонування веб-середовища, користувачі, що працюють з ним, мають різні права доступу до інформаційних ресурсів, в залежності від ролі самого користувача.

1.2.2 Основні загрози інформаційної безпеки

Метою порушників може бути крадіжка конфіденційних даних, порушення цілісності або зупинка процесів веб-програми, що приводить його в неробочий стан.

Метою крадіжки персональних даних може бути отримання клієнтської бази компанії клієнта, отримання персональних даних клієнтів.

Метою порушення цілісності веб-додатку є зміна особистого рахунку клієнта або зміна даних інших користувачів.

Метою порушення доступності веб-додатку є зниження його репутації, відмова клієнтів від використання веб-додатку (конкуренція) або отримання прибутку.

Можливі способи реалізації загроз інформаційної безпеки:

– Несанкціонований доступ до інтерфейсу управління: інтерфейси управління зазвичай доступні через загальнодоступні мережі для авторизованих клієнтів і можливих неавторизованих зловмисників, тоді як звичайні центри обробки даних зазвичай доступні тільки авторизованим адміністраторам безпосередньо або через приватні мережі. Більш того, доступ до управління зазвичай здійснюється через веб-додаток або сервісні технології, тому інтерфейс управління веб-середовищем, ймовірно, схильний до вразливості цих технологій.

– Проблема відновлення даних: через природу віртуалізації і спільного використання даних на апаратному рівні області пам'яті і зберігання, які були орендовані попередніми клієнтами, можуть бути перерозподілені для нових клієнтів. Можливо, що ці нові клієнти можуть відновлювати дані з цих областей пам'яті і зберігання, які можуть містити конфіденційну інформацію, що належить попереднім клієнтам.

– Вразливість образу шаблону віртуальної машини (VM): нова віртуальна машина зазвичай створюється шляхом клонування образу шаблону попередньо сконфігурованої віртуальної машини, так як це економить час і зусилля. Таким

чином, багато клієнтів будуть орендувати віртуальні машини з однаковими конфігураціями. Зловмисник може збирати інформацію про образи шаблонів веб систем, ставши клієнтом веб-додатку з правами адміністратора. Як тільки зловмисник має доступ до образів шаблонів, він може шукати вразливості в цих образах, які також використовуються іншими клієнтами.

– Можливість витоку даних: інша проблема вразливості, пов'язана з образами шаблонів віртуальних машин, полягає в тому, що провайдери можуть використовувати шаблони, створені іншими клієнтами для нових клієнтів. Ці шаблони можуть містити секретні бекдори, створені зловмисником, що прикидаєьтся клієнтом, і дозволяють зловмисникові отримати доступ до віртуальних машин інших клієнтів.

– Ін'єкційні вразливості: оскільки більшість веб сервісів використовують служби веб-додатків, можна вводити зловмисні коди в веб систему, використовуючи вразливості в таких службах веб-додатків, щоб зламати веб-сервери, які обслуговують ці служби. Існує багато прикладів способів взлому з використанням зловмисних кодів, таких як коди SQL, команда ОС або коди JavaScript. Як тільки веб-сервер зламаний, він може використовуватися в якості стартового майданчика для зловмисника для взламування інших цілей в системі, і ці цілі включають бази даних і операційні системи.

– Метрики безпеки і труднощі моніторингу: клієнти повинні мати можливість вимірювати і контролювати ситуацію безпеки своїх веб послуг і ресурсів. Однак надання таких можливостей клієнтам як і раніше є проблемою, тому що доступні традиційні стандартні інструменти ще не підходять для веб середовища. Оскільки веб середовище має складні і динамічні ієрархічні сервіси, які можуть включати в себе різних провайдерів, веб середовище вимагає нових розподілених можливостей моніторингу, відповідних цим характеристикам.

– Складність в управлінні цифровими ключами і випадковими числами: в веб системі існують різні типи ключів і випадкових чисел, необхідні для

криптографічних операцій. Управління та зберігання різних ключів в веб середовищі - непрості завдання, тому що немає повної фізичної ізоляції між ресурсами зберігання, виділеними для різних клієнтів. Ефективність генерації випадкових чисел в основному залежить від апаратного годинника, використовуваного генератором випадкових чисел. Відсутність такої ефективності може бути випробувано в веб середовищі, де в різних сеансах кілька клієнтів використовують одні і ті ж ресурси генерації одночасно. Це може призвести до перевантаження ресурсів генерації випадкових чисел, або може привести до отримання слабких чисел. Таким чином, впровадження стандартних механізмів безпеки, таких як модуль апаратної безпеки, який спирається на ефективний ресурс генерації випадкових чисел, в веб системи є проблемою безпеки.

– Проблема функціональної сумісності: ця проблема пов'язана з тим, як різні провайдери дозволяють власникам даних безперешкодно переміщати свої дані від одного провайдера до іншого або від провайдера назад в свої локальні ресурси, коли їм це потрібно. Без функціональної сумісності між провайдерами, власник даних може заблокувати певного провайдера і не зможе легко перейти до інших провайдерів або оптимізувати послуги між різними провайдерами.

– Спостереження за шаблонами активності: шаблони активності одного веб клієнта можуть спостерігатися або іншими клієнтами, або провайдером. Це спостереження може бути кроком для атаки безпеки або може бути використано для виявлення ділових дій, які не можуть бути виявлені в звичайних обставинах. Наприклад, обмін інформацією між двома компаніями може свідчити про планування злиття.

– Необхідність взаємних можливостей проведення аудиту, підзвітності і надійність: довіра повинна будуватися між провайдерами і клієнтами в веб середовищі. Прозорість через високий рівень можливостей проведення аудиту і підзвітності має важливе значення для створення довірливих відносин. Довірливі відносини будуть більш складними, якщо провайдери делегують деякі з веб

сервісів субконтрактам. Клієнти повинні знати, чи є які- небудь субконтракти, які також можуть відповідати за їх дані. Наприклад, Linkup надав онлайн-службу зберігання через іншого субпідрядного провайдера під назвою Nirvanix, перш ніж він закрився в результаті втрати значної кількості клієнтських даних. Ймовірно, субпідрядник ніс відповідальність за втрату клієнтських даних.

1.3 Існуючі напрямки захисту від загроз

Методи захисту інформації мають на увазі якісь дії з боку розробників веб-додатку, що сприяють максимальному захисту інформації. Основні методи захисту інформації від загроз перераховані в таблиці 1.1.

Таблиця 1.1 - Основні способи захисту від загроз

Методи захисту	Опис
Програмні	Програми, за допомогою яких відображаються хакерські атаки, виконується відновлення втрачених даних, а також створюються резервні копії інформації.
Апаратні	Пристрої для обробки інформації
Фізичні	За допомогою фізичних засобів захисту запобігає доступ сторонніх осіб на територію, на якій зберігаються конфіденційні дані.
Організаційні	Методи захисту, які мають на увазі регламентацію, управління і примус. До них відноситься розробка посадових інструкцій, різні бесіди зі співробітниками, заходи заохочення і покарання. Співробітники несуть відповідальність за зловживання посадовими повноваженнями, за витік або втрату даних.

Продовження таблиці 1.1

Методи захисту	Опис
Законодавчі	Нормативно-правові акти, які регулюють діяльність співробітників, що мають доступ до конфіденційних даних, і визначають міру відповідальності за втрату або крадіжку секретної інформації.
Психологічні	Заходи для створення особистої зацікавленості співробітників в цілості і достовірності даних. Кожен співробітник повинен відчувати себе важливою частиною системи і повинен бути зацікавлений в успіхах компанії.

Програмні, апаратні і фізичні засоби захисту в більшості своїй застосовуються для захисту від зовнішніх порушників, які не мають доступу до конфіденційних даних.

Для захисту від внутрішніх порушників застосовуються організаційні, законодавчі та психологічні заходи, що відносяться до типу співпраці між власником і особами, що мають доступ до конфіденційної інформації.

Для максимального захисту конфіденційних даних потрібно постійно стежити за новими загрозами інформаційній безпеки і при необхідності максимально оперативно усувати порушення вимог безпеки.

1.3.1 Загрози, пов'язані з передачею даних між користувачами

При аутентифікації користувачів і їх подальшою взаємодією з веб-сервером, передача даних здійснюється по протоколу HTTPS, за допомогою SSL-сертифіката, що дозволяє шифрувати всі передані дані між користувачами і сервером. У разі крадіжки переданих даних зловмисником, він отримає зашифровані дані, які йому будуть не цікаві.

Мінімізувати загрози здійснення несанкціонованого доступу до розділів веб-сервера, що вимагає авторизацію користувачів, за допомогою захисту від наступних атак можливо за допомогою:

– • SQL-injection - захистом проти ін'єкцій є точна фільтрація і перевірка вхідних даних від користувачів;

– XSS - для захисту проти цієї атаки використовується кілька способів, до яких відносяться екранування даних, білі списки, вказівка кодування сторінки, установка прапора HttpOnly, а також впровадження Content Security Policy;

– контроль доступу до функцій веб-сервера - система перевірки прав доступу користувачів до кожного функціонального компоненту;

– підробка міжсайтових запитів (CSRF) - всі важливі дані передаються за допомогою POST запитів, а також перевіряється джерело даних;

– прямі посилання на сторінки з конфіденційною інформацією - при запиті даних перевіряються права доступу користувача;

– крадіжка пароля адміністратора - при вході користувача з роллю «Адміністратор» відбувається двоетапна аутентифікація.

1.3.2 Захист конфіденційності і цілісності даних від провайдерів

Конфіденційність і цілісність є важливими вимогами для різних додатків, таких як e-government і EHR (Electronic Health Record). Клієнти не тільки турбуються про компрометацію конфіденційності і цілісності своїх даних від можливих зловмисників, а й від потенційних цікавих до цього провайдерів. На жаль, порушення безпеки, підраховані в 2011 році і перераховані в [6], показують, що великі компанії, такі як Google, EMC/RSA, Sony, UK National Healthcare System (NHS) і Amazon EC2, всі стикалися з інцидентами з безпекою. Дані клієнтів передаються стороннім провайдерам, які можуть бути надійними або ненадійними. Термін ненадійний може використовуватися для вказівки того, що провайдеру не можна повністю довіряти. Наприклад, ненадійні

постачальники послуг можуть не змінювати дані користувачів, але вони можуть пасивно порушувати конфіденційність даних або приховано змінювати протоколи для своєї фінансової вигоди. Іншими словами, сервер провайдера можна розглядати як чесний, але допитливий сервер. Отже, він заслуговує на довіру в наданні послуг з точки зору доступності даних, дотримання основних вимог контролю безпеки і обробки чесно дозволених запитів на збереження даних і повернення правильних результатів. Проте, можливі зловмисні дії всередині серверу можуть виконуватися зловмисним адміністратором або співробітником.

У зв'язку з більш широким впровадженням веб сервісів дослідники вивчають і розробляють нові методи, які зберігають конфіденційність і цілісність зовнішніх даних без повної залежності від провайдерів для забезпечення цих вимог безпеки. Рішення повинні надавати клієнтам більше контролю над захистом своїх даних і також захищати дані від провайдерів. Оскільки сервер провайдера, на якому розміщені аутсорсингові дані, може бути не повністю надійним, кілька дослідників запропонували методи вирішення таких ситуацій. Загалом, запропоновані ними рішення базуються на шифруванні даних до того, як дані будуть відправлені на сервер провайдера.

Хоча зашифровані дані захищені від несанкціонованого доступу, вони не можуть бути повністю корисні, якщо вони не дешифровані. Наприклад, авторизовані користувачі не можуть шукати ключові слова в зашифрованих даних, використовувати зашифровані дані в якості вхідних даних для операцій обчислення або порівняння. Оскільки дешифрування даних може розкривати їх контент серверам-провайдера, то принаймні більш безпечно буде розшифровувати дані тільки в надійних машинах, які контролюються користувачем, що був авторизований для доступу до цих даних.

На рисунку 1.1 показана базова архітектура шифрування даних для захисту конфіденційності, перш ніж відправляти їх до веб-серверу. Потім дані залишаються зашифрованими на веб-сервері, і тільки користувачі, авторизовані власником даних, можуть отримати облікові дані для доступу до зашифрованих

даних. Зашифровані дані можуть бути розшифровані тільки після їх завантаження на авторизований комп'ютер користувача. У такому сценарії конфіденційність даних не залежить від неявного припущення про довіру сервера або індивідуальних угод про рівень послуг (SLA). Замість цього, захист конфіденційності залежить від методів шифрування, що використовуються для захисту даних.

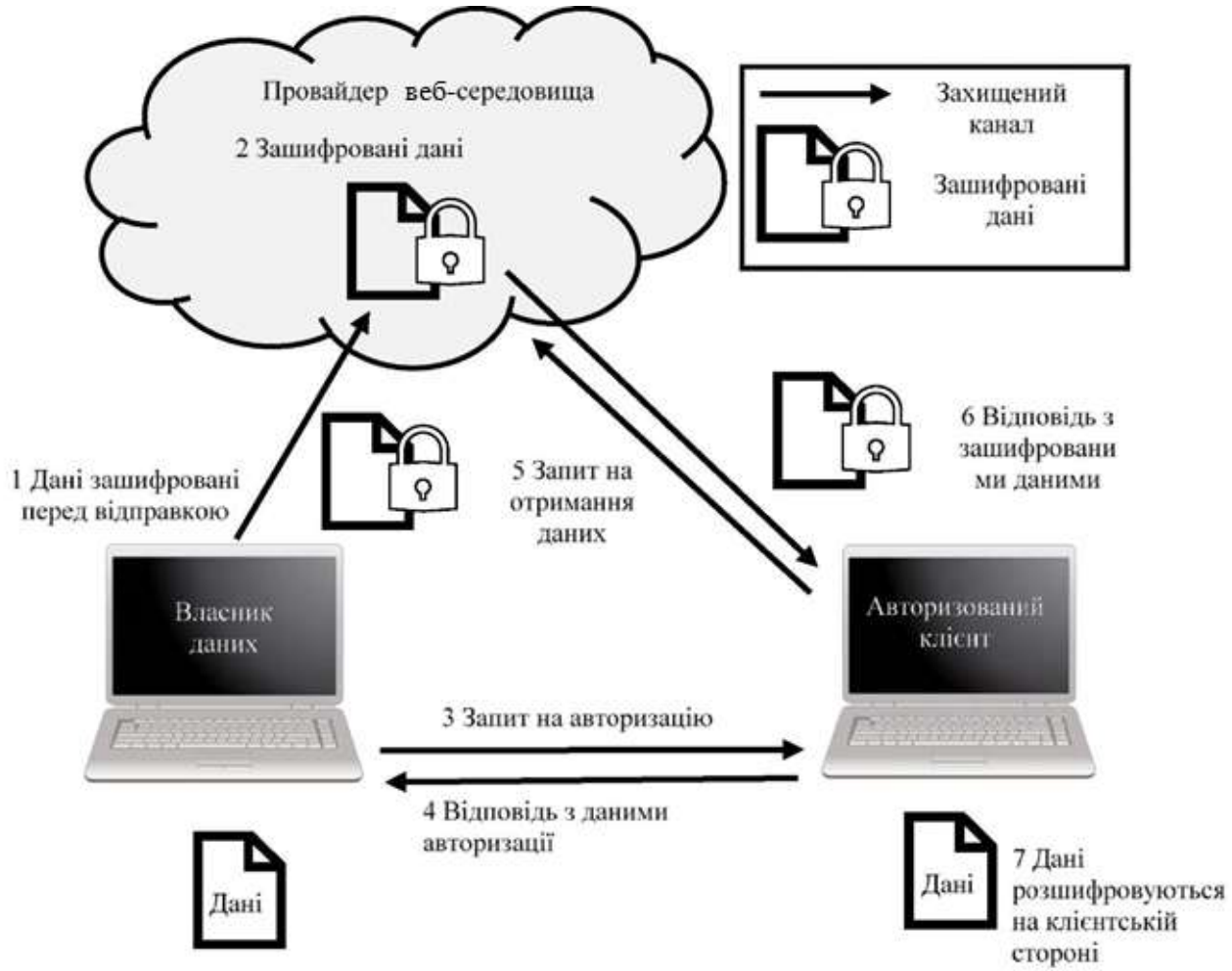

Рисунок 1.1 - Базова архітектура для збереження конфіденційності даних на веб-сервері

Решта проблем полягають в тому, як дозволити власнику даних і авторизованим користувачам ділитися і шукати зашифровані дані і використовувати їх для деяких обчислень відповідно до їх прав доступу. Всі ці

функції повинні виконуватися безпечним чином, без розкривання приватної інформації неавторизованим суб'єктам, включаючи провайдерів. Нові криптографічні методи, схеми довіренних обчислень і підходи орієнтовані на безпеку інформації можуть стати перспективним рішенням для подолання декількох проблем безпеки.

Нові технології криптографії необхідні, щоб відповідати моделі безпеки і підвищити безпеку інформації з меншим впливом на зручність її використання. Наприклад, шифрування з можливістю пошуку - це одна з областей, де дослідники розробляють можливість пошуку в зашифрованих даних без їх дешифрування. Лише авторизовані користувачі можуть запитувати і отримувати зашифровані дані, не відкриваючи ніяку приватну інформацію ні про дані, ні про запит, який може містити інформацію про дані. Іншим прикладом є методи шифрування, які дозволяють виконувати обчислення зашифрованих даних без їх дешифрування. В таких методах також захищаються результати обчислення, які зазвичай містять інформацію про захищені дані. Прикладом цих методів є гомоморфне шифрування. Існують і інші методи криптографії, такі як шифрування на основі ідентифікаційної інформації (IBE) і шифрування на основі атрибутів (ABE), які можуть поліпшити управління ключами і контроль доступу до веб-систем. Чоу, Голле (Chow, Golle) та співавтори вважають, що ці нові криптографічні рішення можуть бути найбільш підходящими інструментами для вирішення декількох проблем безпеки і надання клієнтам більш ефективного контролю над своїми даними у веб-сховищі, розглядаючи, як ці методи можуть бути поліпшені, щоб адаптувати їх до практичного веб-середовища.

IT-спільнота, зокрема Trusted Computing Group (TCG), намагається створити набір технологій, таких як автентифікація, шифрування даних, управління ідентифікацією та доступом, управління паролями, управління доступом до мережі та аварійним відновленням, щоб гарантувати, що комп'ютерні системи будуть виконувати бажаний тип операції. TCG застосовує схему довірених обчислень до Trusted Multi Tenant Infrastructure (TMI), щоб

встановити довіреність ненадійного середовища, наприклад, в послузі публічних веб-сховищ. Нова концепція спрямована на те, щоб дозволити клієнтам оцінювати довіреність провайдерів, застосовуючи набір апаратних і програмних технологій. Атестація віддаленого сервера - одна з цих технологій, яка дозволяє клієнтам перевіряти хости.

TCG починає будувати свої рішення на основі створення стандартного апаратного модуля - Trusted Platform Module (TPM), який виконує основні функції криптографії, такі як хеш-функція і RSA. Ці криптографічні функції необхідні для встановлення надійності в обчислювальному обладнанні. Іншими словами, TCG спрямована на надання стандартних апаратних і програмних технологій для надійних обчислень. Тому деякі з рішень безпеки, зокрема ті, які пропонуються для забезпечення віртуальної ізоляції між віртуальними машинами і віртуальною машиною від провайдера сервера, засновані на технологіях TCG. Ці технології як і раніше стикаються з декількома проблемами і не можуть виключно використовуватися для забезпечення універсального рішення всіх проблем з безпекою. Наприклад, якщо TPM, який є основним компонентом TCG, скомпрометовано, всі рішення буде порушено, як вказав Крістофер Тарновський (Christopher Tarnovsky) в 2010 році. Крім того, концепція TCG не надає користувачам достатній контроль над своїми даними з точки зору конфіденційності та політики безпеки.

У традиційних методах захисту даних безпека забезпечується сервером, який зберігає інформацію. Методи, які використовуються для захисту даних, а також управління захищеними даними контролюються адміністраторами сервера. Такий підхід можна класифікувати як системно-орієнтований підхід, який не підходить для захисту даних клієнтів в менш надійному веб-середовищі. Очікується, що підхід, орієнтований на інформацію, буде більш ефективним і адаптивним. Термін орієнтований на безпеку інформації в цілому вказує на те, що захист сфокусовано навколо даних. Ця термінологія може використовуватися по-різному. Підхід орієнтований на безпеку інформації полягає в захисті даних зсередини, таким чином дані, відповідно до їх значення

і класифікації, мають свої вимоги безпеки, вбудовані в фактичні дані, щоб забезпечити оптимальний захист даних на будь-якому етапі існування даних, незалежно від середовища, в якому зберігаються дані.

1.3 Висновки по розділу 1

Розглянуто обчислювальну парадигму, в якій, з одного боку, фізичні ресурси можуть спільно використовуватися клієнтами в Інтернеті, і з іншого боку, клієнти мають власний обчислювальний простір, використовуючи методи віртуалізації. Загальна концепція полягає в тому, що служби та ресурси, що надаються провайдером через широкосмугові мережі, в основному в Інтернеті, і клієнти використовують ресурси і послуги в міру необхідності і платять тільки за те, що вони споживають. Ці послуги можуть постачатися в різних моделях, що базуються на типі послуги, що надається. Основними трьома послугами є: програмне забезпечення як послуга (SaaS), платформа як послуга (PaaS) і інфраструктура як послуга (IaaS). Загалом, існує п 'ять відомих моделей розгортання послуг, а саме: приватне веб-сховище, громадське веб-сховище, публічне веб-сховище, гібридне веб-сховище і віртуальне приватне веб-сховище. В даний час модель публічних веб-сховищ є найпопулярнішою комерційною моделлю. З одного боку, ця технологія дає великі переваги, такі як економічна ефективність, економія часу і масштабованість. З іншого боку, ця технологія стикається з цілою низкою труднощів і проблем, коли проблеми конфіденційності та безпеки можуть вважатися найбільш складними. Таким чином, дослідницькі тенденції полягають у захисті конфіденційності даних і цілісності у веб-сховище навіть від самих провайдерів і в наданні клієнтам функцій можливості більш строго контролювати свою політику безпеки даних у веб-сховище. Запропоновані рішення для підвищення безпеки даних у веб-сховище мають різні напрямки: деякі зосереджені на використовуваних інструментах, в першу чергу криптографічних алгоритмах, для підвищення безпеки даних у веб-сховищі, інші зосереджені на більш комплексних рішеннях,

що поєднують різні методи безпеки, засновані головним чином на двох підходах: надійні обчислення та підхід орієнтований на безпеку інформації.

2 МЕТОДИ ЗАБЕЗПЕЧЕННЯ КОНФІДЕНЦІЙНОСТІ ДАНИХ У WEB СЕРЕДОВИЩІ

2.1 Методи керування доступом

Дані користувачів веб-ресурсів, в основному, зберігаються у віртуальних сховищах провайдерів. В публічних SaaS та DaaS моделях користувачі володіють лише даними, які знаходяться на зберіганні. Все обладнання та програмне забезпечення, залучене до зберігання та обробки інформації, знаходиться у власності сервіс провайдерів. В інших моделях, таких як публічні IaaS и PaaS моделі, користувач має доступ до обробки даних та до програмного забезпечення, при цьому доступу до апаратного забезпечення немає. Відповідно, з перспективи користувача веб сервісу, найбільш цінним активом є його дані, особливо ті дані, що містять інформацію делікатного характеру і вимагають особливого ставлення, а саме: дані урядового характеру, охорони здоров'я та фінансового спектру. Раніше користувачі розміщували делікатні дані на своїх ПК, зараз більш привабливі перспективи до аутсорсингу своїх даних до веб-серверу. Як і будь-які інші послуги в мережі Інтернет, веб-сервіси також зазнають атак на системи безпеки. Компрометація конфіденційності та приватності даних споживачів веб-послуг, може привести в свою чергу до довгострокових ефектів і будь-які втрати можуть бути достатньо важким для відновлення. Наприклад, коли кілька паролів з адміністративних облікових записів UK's National Healthcare System (NHS) було взламано в червні 2011 року, система NHS була закрита органами охорони здоров'я для захисту записів пацієнтів. Це показує, що для таких випадків конфіденційність даних є більш важливою ніж нормальне функціонування системи як такої. Внутрішні ризики можуть бути від користувачів-зловмисників, які використовують той самий веб сервіс, і пом'якшення ризиків, в такому випадку, залежить повністю від

провайдера і виходить з-під контролю власника даних. З точки зору власників даних, веб середовище невидиме і власник даних не впевнений в тому як його/її дані захищені від ризиків пов'язаних з безпекою. Наприклад, виходячи з природи файлів, їх може бути переміщено через сервіс провайдерів, невідомих для власників даних, або в різних країнах, з різною юрисдикцією щодо конфіденційності даних. Як наслідок з цих проблем, клієнти веб-сервісу відчувають обмежений контроль над своїми даними і їм бракує впевненості щодо безпеки даних. З іншої точки зору, провайдери мають надмірний контроль над даними клієнтів, особливо щодо їх безпеки та приватності. Отже, власники даних стурбовані безпекою та приватністю їх даних і мають бажання зберегти свої дані в безпеці, навіть від провайдерів послуг зберігання даних. Крім того, вони надають перевагу власноруч управляти політикою безпеки своїх даних в безпечному режимі, як ніби ці дані зберігалися на їх ПК.

Традиційна концепція безпеки зазвичай зосереджується навколо технологій і пристроїв, що використовуються для зберігання та обробки даних. Ця концепція може бути важко адаптованою для забезпечення необхідного відповідного рівня безпеки, особливо для делікатних та конфіденційних даних. Наукова спільнота нещодавно звернула увагу на питання безпеки та конфеденійності даних на веб-серверах, запропоновані рішення в основному спрямовані на забезпечення безпеки операційних систем (ОС), що лежать в основі та віртуальних машин (VM), що хостять веб-сервіси. Таким чином, більшість рішень як і раніше засновані на традиційній концепції безпеки і в основному фокусуються на будь-який ОС-орієнтованій або VM-орієнтованій безпеці. Деякі з цих рішень засновані на концепті надійних обчислень, який було запропоновано і розроблено групою Trusted Computing Group (TCG). TCG прагне розробити набір стандартів і технологій, таких як Trusted Platform Module (TPM), які можуть зберігати клієнтські дані і додатки, оброблювані в межах веб-інфраструктури, який убезпечений навіть від системних адміністраторів сервера. TCG, на основі їх технологій, фокусується на наданні набору інструментів, які

можуть бути використані для надання допомоги клієнтам, щоб оцінити надійність провайдерів, стежити за дотриманням політики, а також створювати можливість прозорості щодо фізичного розташування даних. Проте, клієнти веб-сервісу повинні спочатку довіряти TCG технологіям з точки зору оцінки надійності провайдерів та якщо TPM, який є основним компонентом щоб побудувати цю довіру, буде скомпрометовано, постраждає все рішення. У 2010 році, Крістофер Тарновський стверджував, що йому вдалося скомпрометувати TPM. Отже, дослідження і запропоновані рішення на основі на TPM, можливо, варто піддати переоцінці. Навіть якщо TPM та інші інструменти TCG є гарантією безпеки, фокус цих інструментів не на тому щоб надати користувачам бажаний контроль за безпекою та приватністю їх даних. Замість цього, з точки зору клієнта, реалізація концепції TC дозволяє здійснювати клієнтам моніторинг або аудит операцій, в тому числі над політикою контролю доступу, для серверу через довіренні інструменти, які можуть забезпечити докази відповідності концепції TC для користувачів. Нова концепція була запропонована декількома дослідниками для вирішення конкретних питань безпеки даних, переміщаючи фокус з забезпечення безпеки для даних клієнтів на дані, як такі. Ця концепція все ще розвивається, і існують різні думки щодо її застосування.

2.1.1 Класифікація існуючих підходів

Існуючі підходи можуть бути класифіковані за двома критеріями: перша класифікація заснована на тому, на якому рівні забезпечується безпека, а друга класифікація - на тому, хто несе відповідальність за забезпечення безпеки.

На правій частині рисунку 2.1, проілюстровано рівні які можуть бути передбачені функцію безпеки по відношенню до даних. В цілому, рішення сфокусоване на забезпечення безпеки поза рівнем даних класифікуются як системно- центровані. Якщо рішення сфокусовано на окремому визначеному рівні, його класифіковано відповідно до цього специфічного рівня. Наприклад, рішення спрямовані на поліпшення безпечної ізоляції між віртуальною

машиною та гіпервізором можуть бути класифіковані як VM-орієнтовані рішення в області безпеки. З іншого боку, рішення, спрямовані на забезпечення безпеки даних усередині самих даних, як показано на лівій частині рисунку 2.1, класифікуються як інформаційно-орієнтовані підходи. Рівні, показані на рисунку 2.1, є типовими рівнями. Там може бути більше або менше рівнів в практичній системі, на основі фактичних потреб та реалізації.

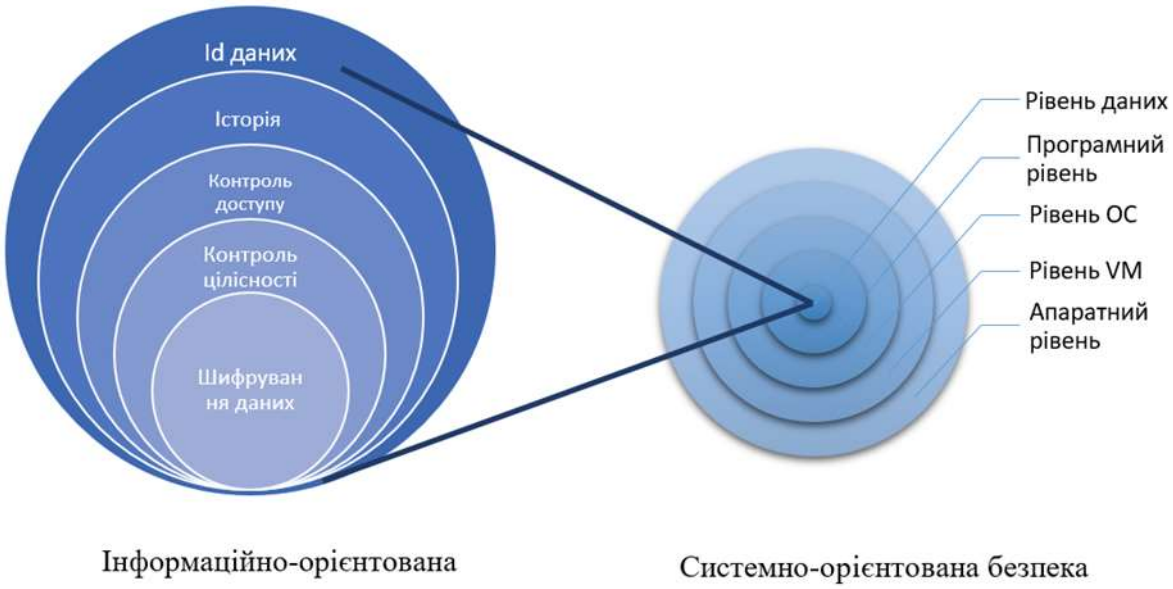

Рисунок 2.1 – Класифікація рівнів забезпечення безпеки даних

Приклад другої класифікації показано на рисунку 2.2. Існує три рівні відповідальності безпеки:

- Рівень сервіс провайдерів, де забезпечується безпека та здійснюється підтримка хмарних провайдерів.

- Рівень довіренних обчислень, де забезпечується безпека та організатором виступає третя сторона.

- Рівень даних, на якому забезпечується безпека та організаторами виступають власники даних.

Для комплексного рішення питання безпеки, всі ці три класифікації безпеки повинні бути інтегровані і адаптовані до моделі веб-сховища. Проте, залежність одного рівня безпеки від іншого рівня, повинна бути зведена до мінімуму. Так, наприклад, функції безпеки, що забезпечуються на рівні інформаційно-орієнтованої безпеки не повинні покладатися на інші рівні безпеки, зокрема, рівень гіпервізора та апаратнний рівень, так як ці два рівні знаходяться під контролем провайдера в буд-якій моделі. Крім того, з точки зору відповідальності, управління безпекою інформаційно-орієнтованого рівня має здійснюватися лише власником даних, так як дані у веб-сховище належать власнику даних незалежно від моделі, що хостить їх.

Рисунок 2.1 – Рівні безпеки даних

З точки зору вивчення різноманітності поглядів в розумінні концепції інформаційно-орієнтованої безпеки даних (ІОБД), дослідники погоджуються зосередитися на інформації при розробці необхідного раціонального рівня безпеки. Однак у дослідників різні погляди на те, як забезпечити необхідний рівень. Чи надавати його за межами даних (додатки, обладнання і т. д., як показано в правій частині рисунка 2.1. протягом його життєвого циклу) або всередині даних (як показано в лівій частині того ж рисунка)? Перші два

посилання в списку орієнтовані на забезпечення безпеки за межами даних, оскільки концепція ІОБД застосовується до бізнес-процесів з використанням традиційних обчислювальних систем. Їх мета полягає в тому, що якщо система має кілька рівнів політик безпеки і механізмів, які розробляються незалежно, то система зможе застосувати належний рівень безпеки до інформації відповідно до її класифікації.

При застосуванні концепції ІОБД до веб-системи мова йде про захист даних від самої веб-системи. Отже, з одного боку, в деяких дослідженнях концепція ІОБД застосовується до веб-сховища, все ще фокусуючись на забезпеченні належного рівня безпеки за межами даних, і вони намагаються захистити свої механізми безпеки і захищати дані від веб-системи, яка розміщує дані на основі концепції Trusted Computing (TC). З іншого боку, в інших дослідженнях концепція ІОБД базується на забезпеченні функцій безпеки через даних, тобто всередині даних, тому вона стає самоописовою, самозахищеною і самообороняємою. Однак в [7] ІОБД залежить від технологій ТС для оцінки надійності середовища, тому для забезпечення безпеки даних потрібен зовнішній захист даних. Орієнтація на безпеку інофрмації і збереження конфіденційності даних у веб-сховище представлена шляхом забезпечення конфіденційності даних і конфіденційності доступу до даних з використанням або підходу ТС, або підходу ІОБД, де дані самозахищені. Що стосується відповідальності за безпеку, більшість уявлень про безпеку враховують, що власник даних відповідає за оцінку вимог безпеки даних і забезпечення безпеки даних до того, як захищені дані будуть відправлені до веб-сховища. Однак після того, як дані перемістилися до веб-сховища, дослідники не згодні з тим, хто несе відповідальність, тобто власник даних, провайдер або третя сторона, за підтримку і управління безпекою даних відповідно до концепції орієнтованої на безпеку інформації.

У цій роботі підхід ІОБД визначається як такий, що базується на забезпеченні вимог безпеки з середини даних, тому захист даних і вся інформація, необхідна для захисту, прив'язані до даних. Більш того, вимоги до безпеки - відповідальність власників даних. Підхід ІОБД буде відповідати

критеріям класифікації концепції ІОБД в двох вимірах: на якому рівні забезпечується безпека і хто несе за це відповідальність (див. рисунок 2.1 і рисунок 2.2).

На даний момент концепція ІОБД знаходиться на етапі зародження, проте стрімко розвивається. Систематичний перегляд літератури в напрямку пошуку реалізованих систем інформаційно-орієнтованої безпеки не дало плідних результатів. Проте можна відзначити роботу колег з Індії, які практично дослідили інформаційно- орієнтовану систему безпеки на прикладі розподіленої системи охорони здоров’я [22].

Дослідники пропонують рішення, що здатне забезпечити конфіденційність, контроль доступу та цілісність. Дані спочатку сегментуються і кожен сегмент шифрується за допомогою статичних симетричних наборів ключів, а потім за допомогою динамічного симетричного ключа в момент передачі даних. Статичний ключ створюється за допомогою технології AES. Наступний симетричний ключ розроблений за допомогою методу ECDH. Це процес динамічне генерує ключ, тому не потрібно постійно зберігати ключ. Кожного разу цей ключ виводиться на обох кінцях.

Далі використовується асиметрична техніка розподілу ключів для генерації симетричного ключа використовуючи KDF.

2.1.2 Контроль доступу до даних

Власник даних зазвичай є найкращим суб'єктом для оцінки вимог безпеки своїх власних даних. Наприклад, якщо дані підключені до бізнес-моделі, вимоги до безпеки даних повинні бути результатом аналізу їх використання в бізнес-процесі. Як правило, дані повинні оброблятися відповідно за їх значенням, оскільки наступний принцип безпеки проголошує: «цінність того, що захищається, впливає на заходи, прийняті для його захисту». Отже, дані класифікуються на основі оцінки власника даних та вимог до безпеки.

Наприклад, дані можуть бути просто класифіковані як цілком таємні, секретні або конфіденційні. Виходячи з цього, політики контролю доступу та властивості безпеки, необхідні для обробки даних, надаються відповідно до цих вимог безпеки. Класифікація даних на концептуальному рівні може бути представлена як інформація, приєднана до даних, або представлена на вимогу необхідних заходів безпеки, які застосовуються до даних.

Політика контролю доступу - це основна функція безпеки, в якій вимоги безпеки задаються для кожного набору даних відповідно до політик безпеки, які можуть включати будь-які потенційні юридичні зобов'язання. Ці політики включають правила і обмеження, які контролюють доступ, використання і потік даних. Основа концепції полягає в контролі того, хто або що може отримати доступ до даних і з якими наданими правами, наприклад, читати, писати, які відповідають традиційній концепції політики контролю доступу. Питання полягає не тільки у визначенні цих політик, але і в забезпеченні дотримання цих політик безпечним чином, який зберігає конфіденційність користувачів при їх доступі або спільному використанні даних. У підході ІОБД політики контролю доступу та механізм їх застосування визначається власником даних, а хмарна система несе відповідальність за дотримання механізму примусового виконання. Завдання полягає в тому, як політики контролю доступу та їх примусове застосування можуть бути поширеними і легко модифікованими для адаптації до динамічних змін вимог до використання і спільного використання даних в хмарному обчислювальному середовищі. Оскільки у кожного набору даних є свої політики доступу, виникають інші складні проблеми щодо взаємодії різних політик контролю доступу та права власності на новий набір даних, який створюється в хмарі від обробки декількох наборів даних з різними політиками і власниками даних.

Конфіденційність вмісту набору даних захищена від будь-яких неавторизованих об'єктів, навіть провайдерів. Щоб самозахистити конфіденційність даних, вони шифруються за допомогою достатньої технології шифрування, і вони залишаються зашифрованими до тих пір, поки вони не

будуть переведені в повністю надійний домен і не будуть розшифровані авторизованим користувачем. Тільки авторизовані користувачі можуть мати доступ до секретного ключа, який прикріплений до набору даних, а також захищений. Зашифровані дані упаковуються з їх параметрами безпеки, такими як захищений секретний ключ і політика використання даних. Тому кожен із самозахищених наборів даних має свої політики доступу, і ці політики консультуються, коли авторизований користувач запитує доступ до захищених даних. За допомогою самозахисту, де політики доступу впроваджені в дані, доступ до захищених даних може бути виконаний в будь-якому місці, якщо існує механізм для виконання політик вбудованого доступу. Механізм контролю доступу буде рудиментарним в його реалізації в локації. Тільки власник даних може керувати політиками контролю доступу та іншими параметрами і функціями, пов'язаними з самозахистом, характерним для кожного набору даних.

Дуже важливо, щоб цілісність набору даних могла бути перевірена на будь- якому етапі. При зберіганні в загальнодоступному домені набір даних піддається модифікації неавторизованими об'єктами, яким може бути навіть сам провайдер, випадково або навмисно протягом життєвого циклу набору. Тому власники даних і авторизовані користувачі вимагають, щоб ця функція гарантувала, що дані не будуть некоректно змінені. У підході ІОБД інформація для перевірки цілісності даних вбудована в дані і також захищена. Перевірка не залежить від інформації, наданої поза самими даними. Отже, пряме підтвердження цілісності даних у веб-сховище без необхідності завантаження, це великий виклик, особливо якщо дані динамічно змінюються.

Конфіденційність даних означає, що тільки уповноважені сторони мають можливість доступу до захищених даних з відповідними правами і привілеями, визначеними власником даних. Ризик компрометації конфіденційності даних зростає в хмарній моделі через збільшення числа сторін, включаючи інших користувачів в одній хмарі. Крім того, управління даними делегується провайдеру хмари, а хмарам не вистачає апаратного поділу між користувачами.

Це призводить до збільшення ризику компрометації конфіденційності, оскільки дані стають неконтрольованими власниками даних і доступні для інших сторін.

Приватність має різні визначення в літературі. Приватність може використовуватися як синонім конфіденційності, в той час як деякі дослідники розглядають приватність як іншу концепцію. Концепція приватності розглядається по-різному навколо слова, і це відображено в різних законах про приватність. Приватність видається більш широкою концепцією, ніж конфіденційність, оскільки вона охоплює те, що на основі закону або культури вважається приватною інформацією, що стосується організацій і людей, навіть якщо ця інформація сама по собі не є конфіденційною. Ця концепція широти може бути знайдена в одному з визначень конфіденційності, наданих Американським інститутом сертифікованих громадських бухгалтерів і Канадським інститутом дипломованих бухгалтерів в стандарті загальноприйнятих принципів конфіденційності: «права і обов'язки окремих осіб і організацій щодо збирання, використання, утримання та розкриття особистої інформації». Наприклад, ім'я або місцезнаходження звичайної особи зазвичай не вважається конфіденційним, але в деяких випадках воно може вважатися приватним. Тому захист приватності йде далі, ніж захист фактичних даних від несанкціонованого доступу. Він також запобігає витокам будь-якої інформації, яка розглядається як приватна інформація під час обробки даних. Наприклад, прикріплені політики контролю доступу вважаються приватною інформацією, і провайдер не повинен розкривати їх. Застосування такого виду захисту до процесу управління доступом призводить до підходу до доступу до конфіденційності, який може використовуватися для запобігання витоку навіть часткової інформації про дані під час процесу доступу. Тому будь-яке рішення безпеки має враховувати конфіденційність даних і приватність клієнтів у відповідності з різними правилами та вимогами.

Доступність даних відноситься до послуг, наявних і доступних для авторизованих користувачів, коли вони їм потрібні. Крім того, ці служби можуть

бути пов'язані з обробкою критично важливих даних і запуском важливих додатків, які повинні бути доступні весь час і здатні обслуговувати велику кількість користувачів. Доступність послуг залежить від постійної безперервної роботи інфраструктур і мережевих ресурсів. Тому провайдер повинен мати надійні і надлишкові стратегії для забезпечення доступності системи. Провайдер також несе відповідальність за обслуговування своїх сервісів, навіть якщо є внутрішня або зовнішня загроза, націлена на доступність системи. Іншою проблемою доступності є доступність даних клієнта для доступу власником даних або авторизованими користувачами. Авторизований користувач повинен мати можливість отримувати дані з веб-сховища в будь- який час. Як згадувалося раніше, на етапах життєвого циклу даних, постачальник веб-сховища повинен мати ефективну стратегію резервного копіювання та архівування для захисту доступності даних. З точки зору власника даних, дані є найважливішим активом, яким він володіє у веб-сховищі. Пропонований ІОБД підхід в цій роботі дає власнику даних більш істотну відповідальність і контроль над захистом своїх даних, які повинні зберігатися в веб-середовищі.

2.2 Підхід до ІОБД

Вважатимемо, що дані захищені перш ніж покинути довірений домен власника даних, а їх параметри безпеки прив'язані до даних як частина їх метаданих. В ідеальній ситуації тільки облікові дані, які використовуються для відміни або доступу через захист, зберігаються за межами даних, а також власником даних і авторизованими користувачами відповідно до їх прав доступу. Будь-які інші параметри безпеки даних повинні бути прив'язані до даних. Наприклад, в тих випадках, коли дані динамічно оновлюються, користувач, який звертається до даних, повинен перевірити, чи дані є найсучаснішими. Ця вимога має бути надана з фактичних даних або метаданих, прикріплених до даних, наприклад, функція історії є під-рівнем від рівня даних, як показано на рисунку 2.1. Як інший приклад, доказ достовірності джерела

даних також може бути додано разом з доказом цілісності, що зазвичай включає цифровий підпис власника даних. Будь-яка додаткова функція може бути додана як новий під-рівень під рівнем даних або включена в якості одного зі зразкових підрівнів, проілюстрованих в лівій частині рисунку 2.1. Крім того, всі ці метадані безпеки надаються на рівні даних і створюються власником даних. Вони також залишаються захищеними протягом усього життєвого циклу даних. У окремого набору даних є свої метадані безпеки, незалежні від інших наборів даних. У наступному списку наведено критерії, які повинні задовольняти будь-яке надане рішення безпеки на основі підходу ІОБД:

- Кожен набір даних є самоописовим, самозахищеним і самозбереженним (тобто захищеним набором даних). Отже, вимоги та функції безпеки кожного набору даних надаються зсередини і не залежать від установок поза набором даних, крім деяких базових процесів обробки даних.

- Захист даних не залежить від провайдера або довіреної третьої сторони.

- Тільки власник даних відповідає за створення і управління цими вимогами і функціями безпеки для кожного набору даних з моменту його створення до кінця життєвого циклу набору даних.

- Всі операції, пов'язані з доступом до захищених даних, встановлені авторизованими користувачами, і примусове застосування політик безпеки для даних виконуються без шкоди для конфіденційності користувачів або конфіденційності даних.

Основи ІОБД підходу показано на рисунку 2.3. Створення даних виконується в надійному домені з боку власника даних. На етапі створення дані класифікуються на основі вимог безпеки і, відповідно, метадані безпеки прив'язані до даних, наприклад, їх політика контролю доступу, перевірка цілісності, запис їх історії і можливості відстеження. На етапі створення, створюється безпечний набір даних і він інкапсулює вміст захищених даних і відповідні метадані. Контролер управління даними і відстеження дозволяє власнику даних управляти і відстежувати набір даних, наприклад, розташування, використання та історію доступу, з моменту створення, поки він

не буде видалений з веб-середовища. Крім того, передача між надійним доменом, в якому знаходиться користувач, і доменом-сховищем зазвичай здійснюється через Інтернет і захищена основним механізмом захисту, забезпечуваним підходом ІОБД або протоколом інтернет-безпеки. У захищеного набору даних протягом його життєвого циклу у веб-сховищі є можливість, виконавши додані до нього коди, інформувати власника даних, якщо є будь-які зміни його стану. У той же час захищений набір даних включає в себе виконуваний код, готовий для прийому і відповіді на запити власника даних для відстеження інформації і команд управління. Навіть якщо зв'язок між доменом-сховищем і надійним доменом не працює, оскільки набір даних містить всю необхідну інформацію для виконання основних політик безпеки, пов'язаних з даними, захищений набір даних може бути постійно доступний авторизованим користувачам без участі власника даних.

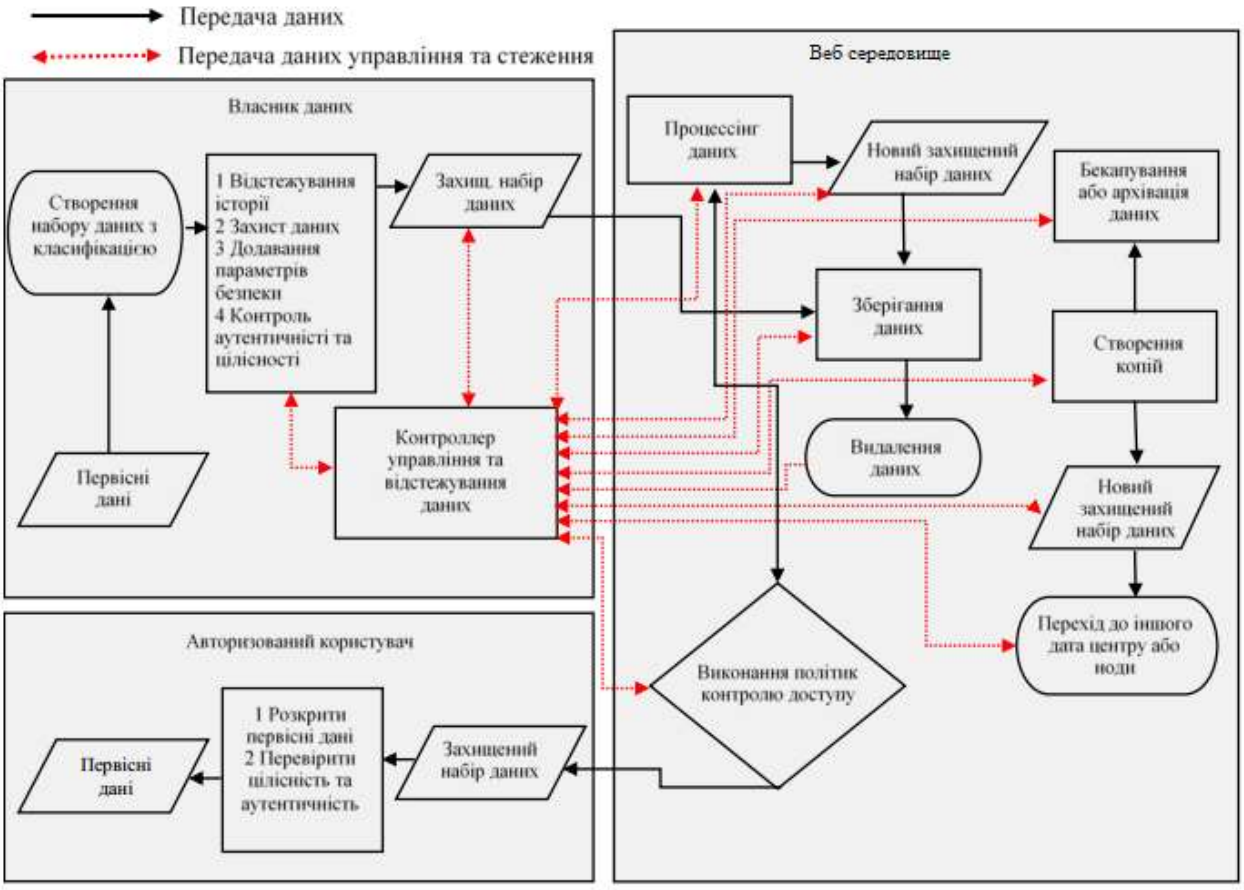

Рисунок 2.3 - Основи ІОБД підходу

Як показано на рисунку 2.3, примусове застосування політик доступу буде виконуватися ресурсами веб-сервера, але на основі параметрів безпеки, прикріплених до набору захищених даних. Правило провайдера в примусовому порядку дозволяє виконання цих політик доступу, не розкриваючи деталі політики контролю доступу або вміст набору даних. Власник даних може перевірити цілісність вмісту захищених даних, включаючи його історію записів.

Коли захищений набір даних повинен бути отриманий авторизованим користувачем, веб-сервер перевірятиме права доступу користувачів до даних з використанням пов'язаних параметрів безпеки. Авторизованим користувачам дозволено завантажувати захищений набір даних. Потім авторизовані користувачі можуть перевірити цілісність і оригінальність завантажених даних, використовуючи додані докази цілісності та аутентичності. Ця концептуальна основа ІОБД дозволяє, з одного боку, оптимізувати конфігурацію безпеки параметрів безпеки для кожного набору даних, а з іншого - знижує складність загального управління безпекою. Це особливо важливо, коли відповідальність за безпеку також оптимально розподілена між потенційними сторонами веб-системи (тобто провайдерами, користувачами і третьою стороною) відповідно до трьох рівнів відповідальності безпеки, проілюстрованих на рисунку 2.2.

2.3 Метод забезпечення конфіденційності інформації у web середовищі

Вимоги щодо забезпечення конфіденційності інформації у web-середовищі згідно пропонованому ІОБД підходу:

- дані зашифровані та доступні лише для авторизованих користувачів.
- дані доступні для пошуку без загрози для їх конфіденційності.
- дані відповідають вимогам самозахисту та необхідним параметрам безпеки.
- параметри контролю доступу приховані від провайдерів та інших користувачів.

- провайдер серверу не знає кількість або особистість користувачів, що авторизовані та мають право доступу до даних.
- несанкціоновані об'єкти, в тому числі постачальники послуг, не можуть отримати доступ до даних або отримати інформацію про дані від авторизованих процесів що проводяться на даних.
- дані містять всю необхідну інформацію для перевірки їх цілосності для авторизованих користувачів, які мають доступ до даних.
- взаємодія між власниками та авторизованими користувачами має бути мінімальною, особливо щодо ключових цілей управління.

Всі параметри, необхідні для досягнення функцій безпеки прикріплюються до файлу даних і в результаті є файл з конфіденційних даних (ФКД-файл). Функції безпеки включають у себе захист конфеденційності, перевірку цілесності та аутентифікацію.

2.3.1 Управління доступом та розділення ключа

У бездротових мережах, де пристрої можуть мати обмежені ресурси, особливо обчислювальні ресурси і ресурси зберігання, рішення на основі теореми про остачі (ТО) можуть бути використані в декількох схемах, безпосередньо при аутентифікації та управлінні доступом до схем [11]. ТО використовує кілька додатків в криптографії та зв'язок захищеними сітями через свою арифметичну природу, що не споживає високих обчислювальних ресурсів. Таким чином, вона підходить для обчислювальних і комунікаційних додатків з обмеженим ресурсами. Розділення секрету є однією із основних функцій безпеки з використанням ТО в криптографії. Це дозволяє виконувати поділ виводячи з нього декілька спільних, які можуть бути відновлені тільки з певним заздалегідь підготовленим набором значень. Багато додатків засновані на цій схемі розділення секрету, такі як порогова криптографія та електронне голосування. Наприклад, бездротова сенсорна мережа застосована в аутентифікації на основі ТО. В іншому прикладі схема управління доступом запропонована і заснована на

ТО для бездротового мультимедійного сенсорного датчика мережі. ТО також використовується для зменшення рівня споживаної енергії при передачі пакетів між вузлами бездротових сенсорних мереж, які призводять до збільшення тривалості життя та енергоефективності мережі. Спеціальні мобільні мережі та інші бездротові системи, де ТО запропоновано використовувати для забезпечення безпеки з меншим впливом на системні ресурси. Існує також непряме використання ТО в додатках для прискорення та зниження споживання ресурсів при генерації ключів.

Для описаного підходу ІОБД ТО використовуватиме криптосистема з публічним ключем для безпечного обміну даними захищених користувачів, що зберігаються у веб-середовищі серед авторизованих користувачів [16]. Дані, які іноді згадуються як ресурси або файли, захищені симетричним методом шифрування, де як секретний ключ використовується K_s. Ресурсом може бути набір даних або файл, який містить дані будь-якого типу, в тому числі текст, аудіо, зображення або відео. Для поширення секретного ключа для авторизованих користувачів, його зашифровано з використанням методу публічного шифрування відкритого ключа користувача. Шифрування також включає в себе інше значення, C_r, яке буде використовуватися в якості відповіді на запит сервера, коли користувач робить запит щодо доступу до ресурсу. Лише авторизовані користувачі, які мають відповідний приватний ключ, можуть розшифрувати шифротекст C_r та K_r для отримання C_r, щоб отримати доступ до ресурсу *r*. Параметри C_r та K_s зчеплені для формування $C_r||K_s$ та розглядаються як єдине значення. Це значення шифрується за допомогою публічного ключа $E_{K_{pub\ i}}$. кожного авторизованого користувача u_i, в результаті шифротекст $(E_{K_{pub\ i}}(C_r \,||\, K_S))$ цього користувача u_i, де *i=1, 2, ... k,* а k - кількість авторизованих користувачів, що мають доступ до ресурсу *r*.

Щоб застосувати ТО до запропонованого рішення, для користувачів u_1, u_2, ..., u_k, кожен авторизований користувач пов'язаний з унікальним відносним простим числом $n_i = n_1, n_2, ..., n_k$, де k - це кількість авторизованих користувачів.

Всі n_i, $1 \le i \le k$, є відносні прості числа. Потім, шифротекст $C_r \,||\, K_s$ генерується для кожного користувача для виведення наступної спільної конгруентності:

$$\begin{aligned} X_r &\equiv (E_{K_{pub\ 1}}(C_r \,||\, K_S)) \bmod n_1, \\ X_r &\equiv (E_{K_{pub\ 2}}(C_r \,||\, K_S)) \bmod n_2, \\ &\dots \\ X_r &\equiv (E_{K_{pub\ k}}(C_r \,||\, K_S)) \bmod n_k. \end{aligned} \tag{2.1}$$

Вирішення цієї конгруенції X_r, таке, що $0 \le X_r < n = n_1 n_2 \dots n_k$, є загальним значенням для ресурсу *r* і він приєднаний до ресурсу, а ресурс і загальна значення зберігаються разом в веб-сервері. Коли авторизований користувач u_i надсилає запит щодо доступу до ресурсу *r*, веб-сервер надсилає йому розділене значення X_r. Коли користувач отримує X_r, він використовує відповідний private key для розшифровки X_r, щоб отримати значення $C_r \,||\, K_s$, як показано в рівнянні (2.2) нижче:

$$\left(C_r \,||\, K_s = D_{K_{prv\ i}}(X_r \bmod n_i)\right), \tag{2.2}$$

де $D_{K_{prv\ i}}$ - це операція дешифрування з використанням приватного ключа користувача u_i та n_i, як відносного простого числа специфічного для даного користувача. Значення C_r потім вирушає назад до серверу користувачем, щоб довести володіння правом доступу до ресурсу. Потім сервер надсилає ресурс для користувача і ключ K_S, який використовується користувачем щоб розшифрувати файл та отримати його вміст.

З точки зору підвищення безпеки в запропонованому рішенні, власник даних повинен використовувати унікальний симетричний ключ, K_S, для шифрування кожного файлу перед відправкою його на зберігання у веб-сховище. Відповідно до ІОБД підходу, всі вимоги до безпеки прикріплені до фактичних

даних, отже, кожен симетричний ключ прикріплюється до його файлу. З рівняння (2.1), симетричний ключ K_S захищений двома рівнями: спочатку шифруванням його авторизованим користувачем, а потім за допомогою ТО, щоб знайти загальне значення X_r. Лише авторизовані користувачі можуть обчислити K_S з X_r за допомогою рівняння (2.2), якому потрібен відповідний n_i та приватний ключ авторизованого користувача. Для ефективного управління ключами, власник даних додає себе в якості користувача при розрахунку X_r для кожного файлу за допомогою рівняння (2.1). Отже, ані власник даних, ані користувачі не зобов'язані зберігати K_s, але їм потрібен тільки їх приватний ключ і призначене значення n_i. Таким чином, ключ K_s надійно та ефективно, спільно з авторизованими користувачами, захищено від несанкціонованого доступу, включаючи постачальника послуг хостингу файлу.

Секретний ключ K_s а також значення C_r є унікальними для кожного файлу. Якщо значення для одного файлу було скомпрометовано, інші файли залишаються в безпеці. Секретне значення C_r використовується авторизованим користувачем, щоб показати серверу, що він або вона має право отримати доступ до ресурсу r (тобто файлу). Власник даних надійно прикріплює секретне значення C_r до файлу, а також надсилає його всередині шифротекст X_r до сервера. Лише авторизовані користувачі можуть розрахувати секретне значення C_r з використанням розділеного значення X_r. Таким чином, лише авторизовані користувачі можуть знати і відкрити C_r серверу і довести, що вони мають право доступу до цього конкретного файлу. Секретне значення C_r є унікальним для кожного файлу, навіть якщо він використовується одним користувач для різних файлів. Таким чином, якщо один C_r для конкретного файлу скомпрометований, ніякі інші файли не будуть під загрозою. Крім того, ця функція корисна, якщо власник даних хоче змінити деталі щодо авторизованих користувачів для файлу, наприклад, щоб додати нового авторизованого користувача, власнику даних лише необхідно змінити X_r для цього файлу. Оскільки параметр C_r може залишатися таким же самим, немає необхідності для повторного надсилання нового C_r до серверу. Це, в свою чергу, дозволить знизити ймовірність

компрометації цього значення в процесі підтримки динамічного механізму оновлення списку авторизованих користувачів. Проте, при забороні користувачу доступу до файлу, до якого він/вона вже мають доступ, значення C_r має бути змінене з міркувань безпеки.

Запропоноване рішення поєднує в собі контроль доступу та спільного використання ключа в одному механізмі з використанням ТО. Веб-сервер зберігає зашифровані дані з відповідним секретним C_r і розділеним значенням X_r, що розраховується власником даних. Провайдер може прочитати розділене значення X_r, але не може дізнатися K_S, який було використано для шифрування даних. Лише авторизовані користувачі можуть виявити K_S від X_r. Провайдер також може зчитувати секретне значення C_r для кожного файлу. Проте, число або ідентичність користувачів які можуть отримати доступ до файлу, навіть якщо вони вже отримали доступ до файлу, залишаються прихованими від провайдера.

Для полегшення необхідних розрахунків, можна відзначити, що, коли авторизований користувач u_i має доступ до файлу r, власник даних вже повинен був надіслати його відносне просте n_i призначене для нього, яке відправляється один раз для кожного користувача. Тоді власник даних використовує його з відкритим ключем користувача для всіх файлів, правом доступу до яких володіє користувач. Для більшої безпеки, n_i повинен бути надісланий користувачеві надійно, наприклад, за допомогою шифрування його з відкритим ключем користувача. Таким чином, зловмисникам буде важче виявити шифротекст $\left(E_{K_{pub\ k}}(C_r \,||\, K_S)\right)$ від спільного значення X_r. Проте, навіть якщо n_i будь-якого користувача розкривається несанкціонованим об'єктам, файли цього користувача залишаються в безпеці до тих пір, поки не буде скомпрометовано приватний ключ користувача. Тому, перед тим, як користувач має право отримати доступ до файлу, власник даних файлу має відкритий ключ користувача при розрахунку X_r файлу і користувач отримав його/її n_i.

Повідомлення, якими обмінюються авторизовані користувачі і хостинг сервером загальних файлів показано на рисунку 2.4. Коли користувачеві u_i

потрібен доступ до файлу r, користувач надсилає запит на сервер, що містить ID файлу, ID_i власника файлу, нонс (будь-яке випадкове число) і сесійний ключ K_t. Все шифрується за допомогою відкритого ключа сервера $E_{K_{pub\ s}}$.

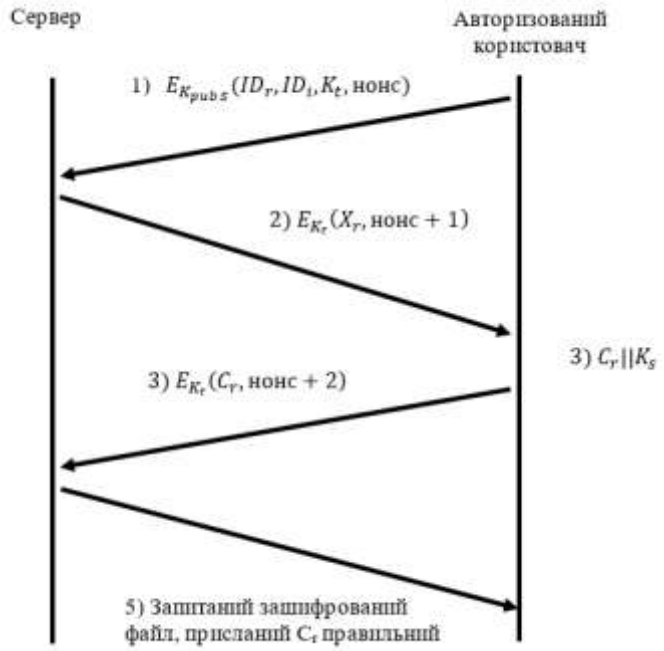

Рисунок 2.4 - Процедура контролю доступа

Результат можна представити у вигляді $E_{K_{pub\ i}}(ID_r ID_i, K_t, нонс)$. Сесійний ключ K_t створюється користувачем для безпечного обміну інформацією між користувачем і сервером. Сервер знаходить файл з ID_r, читає його спільне значення X_r, збільшує *нонс* до *(нонс +1)* і шифрує їх за допомогою сесійного ключа K_t. Результат, можна представити як $E_{K_t}(X_r, нонс+1)$, який повертається користувачу.

Після отримання інформації, представленої в повідомленні номер два на рисунку 2.4, з сервера, користувач обчислює секретний параметр C_r, використовуючи рівняння (2.2) і повертає C_r та *(нонс + 2)* на сервер і шифрується сесійний ключ K_t, тобто $E_{K_t}(C_r, нонс+2)$. Сервер перевіряє C_r і, якщо він збігається з тим, що прикріплений до файлу на який було надіслано запит, сервер

надсилає зашифрований файл користувачеві. Користувач може розшифрувати файл за допомогою ключа K_S, який було отримано раніше в результаті використання рівняння (2.2) на третьому етапі на рисунку 2.4. Всі повідомлення між користувачем і сервером є зашифровані і інкрментуючий нонс використовується для запобігання традиційних атак, такі як Man-In-The Middle (MIM). Для конфеденційного доступа, особистість авторизованого користувача приховано від сервера, отже, процедури уникає використання публічного ключа користувача, щоб зберегти його особистість в таємниці. На противагу цьому, користувач починає сесію з сервером з повідомленням зашифрованим за допомогою публічного ключа сервера для перевірки аутентичності сервера. Таким чином, якщо користувач здійснює зв'язок з певним конкретним сервером, тільки цей сервер може розшифрувати повідомлення для того щоб розкрити сесійний ключ K_t і *нонс*, а потім реагувати зашифрованим повідомленням яке містить *(нонс + 1)*.

Процедура доступу показує, що виконання політики контролю доступу та забезпечення авторизованого користувача секретний ключ для дешифрування даних здійснюється в одному механізмі. Цей метод зменшує додаткове навантаження як на сервері, так і користувача, з точки зору обчислення та управління ключами. До того ж, використання ТО в цьому механізмі, є ще одним фактором для зменшення додаткових витрат, оскільки операції ТО є простими модульними арифметичними операціями; зокрема, наприклад, не має модульних експонентних операцій. Ще одна важлива особливість що досягається за допомогою використання ТО, є те, що ТО дозволяє власнику даних приховувати число авторизованих користувачів. Шифротекст $\left(E_{K_{pub\ i}}(C_r \,||\, K_S)\right)$ кожного авторизованого користувача u_i для $i = 1, 2, \ldots k$ може бути представлений з використанням ТО з одним значенням X_r. Сервер не може дізнатися скільки користувачів представлені в цьому X_r, але без використання ТО власник даних повинен прикріпити всі шифротекст, щоб розкрити серверу кількість користувачів.

Щоб надати доступ до файлу r новому користувачу u_{k+1}, до ТО додається нова конгруентність, як показано в hівнянні (2.3). У такому випадку, власнику необхідно перерахувати загальне значення $X_r^{'}$.

$$\begin{aligned} X_r^{'} &\equiv (E_{K_{pub\ 1}}(C_r \,||\, K_S)) \bmod n_1, \\ X_r^{'} &\equiv (E_{K_{pub\ 2}}(C_r \,||\, K_S)) \bmod n_2, \\ &\ldots \\ X_r^{'} &\equiv (E_{K_{pub\ k}}(C_r \,||\, K_S)) \bmod n_k, \\ X_r^{'} &\equiv (E_{K_{pub\ k+1}}(C_r \,||\, K_S)) \bmod n_{k+1}. \end{aligned} \tag{2.3}$$

$K_{pub\ k+1}$ є відкритим ключем нового користувача. Рішення X_r вищевказаної одночасної конгруентності можна розрахувати з X_r, який вже додано до файлу, з меншою кількістю модульних розрахунків, якщо нова система конгруентності для $X_r^{'}$ має один і той же модуль попереднього X_r плюс новий модуль; іншими словами, якщо власник даних вирішує додати нового користувача для доступу до існуючого файлу і не збирається змінювати попередні значення, включаючи C_r та K_s, які були використані для обчислити X_r. Отже, щоб знайти $X_r^{'}$ більш ефективно, можна знайти рішення використовуючи наступний шлях:

$$\begin{aligned} X_r^{'} &\equiv X_r \bmod n_1 n_2 \ldots n_k, \\ X_r^{'} &\equiv (E_{K_{pub\ k+1}}(C_r \,||\, K_S)) \bmod n_{k+1}. \end{aligned} \tag{2.4}$$

Власник даних надсилає нове загальне значення $X_r^{'}$ з ID файлу та ID власника даних до серверу для заміни старого X_r. Механізм додавання нового користувача ефективний з двох причин; необхідна лише одна асиметрична операція шифрування для фіксованої довжини інформації, тобто, C_r, та він знаходить рішення лише для двох конгруенцій, див. рівняння (2.4). Хоча загальні

значення $X_r^{'}$ базуються на основі секретних значень, лише авторизовані користувачі мають доступ до них.

Щоб скасувати право доступу користувача u_k до файлу r, власник даних повинен змінити секретне значення C_r на $C_r^{'}$ та перерахувати X_r на $X_r^{''}$ для користувачів, що зберігають право доступу до файлу, від 1 до k-1 наступним чином:

$$\begin{aligned} X_r^{''} &\equiv (E_{K_{pub\ 1}}(C_r^{'} \,||\, K_S)) \bmod n_1, \\ X_r^{''} &\equiv (E_{K_{pub\ 2}}(C_r^{'} \,||\, K_S)) \bmod n_2, \\ &\dots \\ X_r^{''} &\equiv (E_{K_{pub\ k-1}}(C_r^{'} \,||\, K_S)) \bmod n_{k-1}. \end{aligned} \tag{2.5}$$

Власник даних надсилає на сервер нові значення $C_r^{'}$ та $X_r^{''}$ з ID файлу та ID власника файлу щоб замінити відповідні старі значення для файлу. Якщо користувач отримав доступ до даних перед скасуванням старих значень, то можливо, що користувач і сервер можуть змовитися для доступу до даних після скасування. Коли авторизований користувач отримав доступ до файлу, ключ цього файлу стає відомим для нього. Пізніше, якщо для цього користувача буде анульовано доступ до файлу і власник даних змінив лише C_r для цього файлу, користувач може домовлятися з сервером, так що сервер дозволяє йому отримати доступ до файлу, і користувач надає серверу старий ключ K_s, що досі розшифровує файл. Схема не розкриває особистість користувачів для сервера і може зменшити імовірність такого роду collusion attack, рішуче рекомендується змінити ключ K_s і повторно зашифрувати файл, особливо якщо відбулася зміна змісту. Потім замінити старий файл на новий, який було зашифровано з новим ключем і який має нові $C_r^{'}$ та $X_r^{''}$ прикріплені до нього. Таким чином, сервер і користувач, позбавлений права доступу, не можуть вступити в змову з метою отримання доступу до нового вмісту файлу.

2.4 Метод зашифрованого пошуку

Власник даних повинен вказати один секретний ключ K_w, який використовується при шифруванні кожного ключового слова. Власник файлу використовує Keyed Pseudorandom Function (PRF) для шифрування ключових слів [14]. Нехай $H_K(m)$ буде Hash-based Message Authentication Code (HMAC), який може бути використаний будь-якою криптографічною хеш-функцію, такою як MD5, SHA1, SHA256 або SHA512, з ключем K та вхідне повідомлення m [15]. Припустимо, w_i це і-ий ключ в списку ключових слів, R це випадкове число, флаг - це постійний бітовий патерн з фіксованою довжиною 1 біт і зміщенням випадковим патерном бітів.

$$a_i = H_{K_w}(w_i),\ b_i = H_{a_i}(R),\ c_i = b_i \oplus (флаг \,||\, зміщення). \quad (2.6)$$

Від кожного ключового слова w_i, власник даних створює c_i та надсилає його з випадковим числом R до сервера як додаток до відповідного файлу. c_i є зашифрованою формою ключового слова w_i; таким чином, сервер не може відкрити w_i від c_i.

Для пошуку зашифрованих даних, що зберігаються в веб-середовищі, коли користувач u_i здійснює пошук ключового слова w_i, користувач має надіслати запит власнику даних для отримання можливості пошуку, що містить ключове слово w_i зашифроване за допомогою публічного ключа власника ($K_{prv\ o}$) і підписаного приватного ключа користувача ($K_{prv\ i}$). Власник даних потім відповідає з $T_w = H_{K_w}(w_i)$, зашифрованому за допомогою публічного ключа користувача ($K_{prv\ i}$). Користувач розшифровує повідомлення за допомогою свого приватного ключа та потім зашифровує T_w з публічним ключом сервера ($K_{prv\ S}$), перед відправкою його на сервер в якості пошукового запиту з ID власника, нонс і сесійний ключ K_t, як показано в рівнянні (2.7).

$$E_{K_{pub\ S}}(T_w, ID\ \text{власника}, K_t, \text{нонс}). \quad (2.7)$$

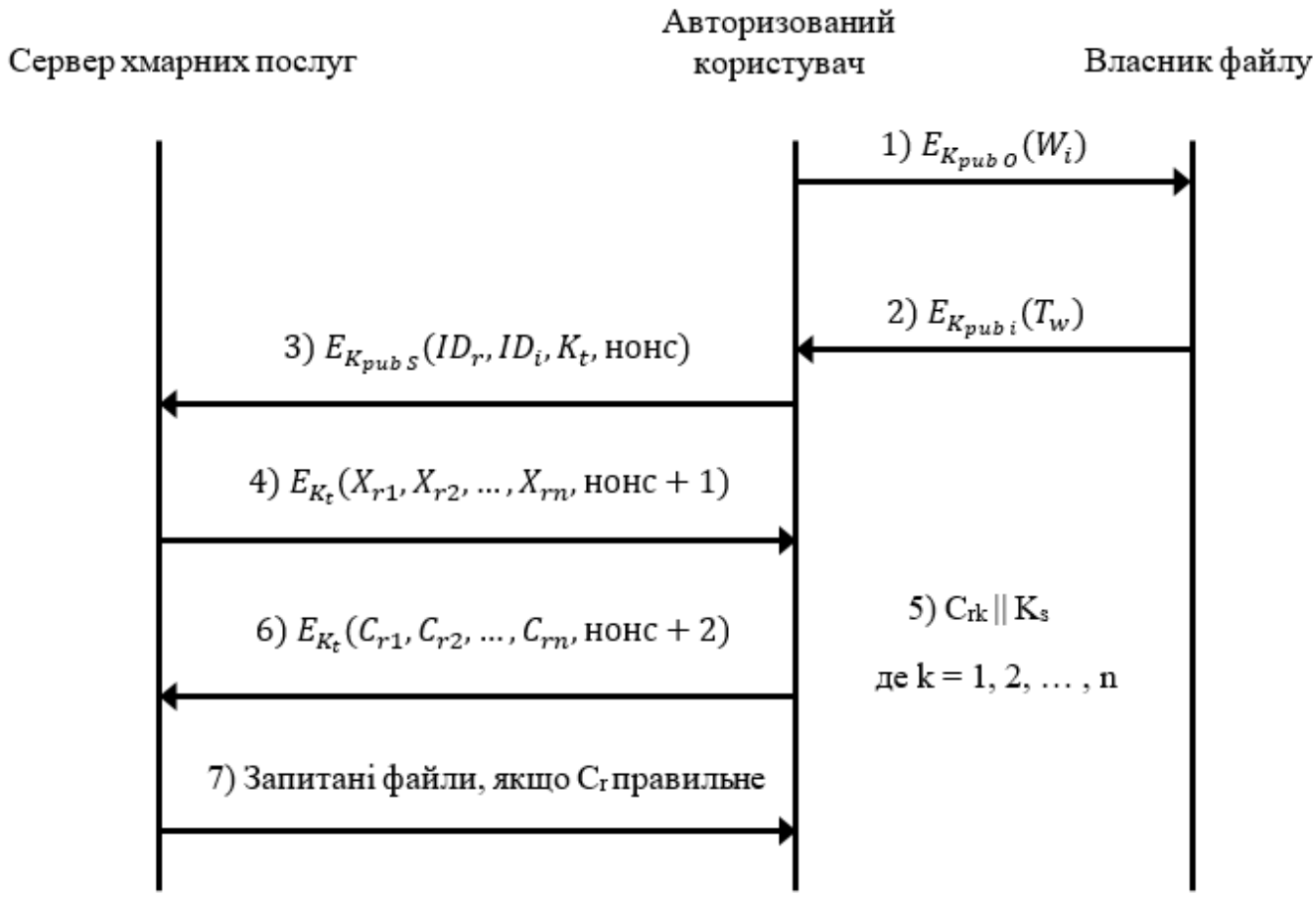

Рисунок 2.5 – Схема методу зашифрованого пошуку

Сервер повинен шукати тільки дані, пов'язані з ID власника, використовуючи T_w для обчислення $P = H_{Tw}(R)$, а потім для кожного c_i сервер перевіряє результат $P \oplus c_i$, щоб побачити, якщо перший біт дорівнює флагу. Потім ID файлу, пов'язане з цим c_i, додається до списку результатів пошуку разом із власними двома параметрами X_r та C_r.

$$P = H_{Tw}(R), \text{ if } P \oplus c_i = флаг \| зміщення \therefore \in \text{збіг}. \qquad (2.8)$$

Після пошуку всіх файлів, сервер надсилає лише те значення $X_r(X_{r_1}, X_{r_2}, ...X_{r_n})$, (де n - кількість файлів, які відповідають критеріям пошуку, та *(нонс + 1))* і всі зашифровані за допомогою сесійний ключ, K_t, назад клієнту.

$$E_{K_t}(X_{r_1}, X_{r_2}, ..., X_{r_n}, \text{нонс+1}). \quad (2.9)$$

Користувач отримує секретне значення C_{ri} для кожного загального значення X_{ri}, де i = 1, 2, ..., n, , а потім повертає секретні значення і (нонс + 2) на сервер зашифроване за допомогою сесійного ключа K_t.

$$E_{K_t}(C_{r_1}, C_{r_2}, ..., C_{r_n}, \text{нонс+2}). \quad (2.10)$$

Сервер перевіряє секретні значення (C_{r1}, C_{r2},..., C_{rn}) та надсилає клієнту лише ті файли, секретні значення яких збігається з тими, які були надіслані користувачем. Тепер користувач може розшифрувати файли за допомогою секретного ключа K_s, витягнутого з X_{ri} для кожного файлу. Рисунок 2.5 ілюструє безпечну процедуру пошуку.

2.5 Побудова ФКД-файлу з перевірками цілісності та автентичності

Концепція ФКД-файлу створює захищений контейнер, який включає в себе всі попередні модулі. ФКД-файл інкапсулює вихідний файл даних перед його відправкою до серверу і таким чином зберігає файл даних захищеним від несанкціонованого доступу, в тому числі серверу. Лише авторизовані користувачі можуть здійснювати пошук і отримати доступ до файлового вмісту, відповідно до доданих параметрів політики безпеки, представлених в C_r та X_r, які встановлюються та управляються, головним чином, власником файлу. Процес створення ФКД-файлу передбачено проводити в повністю довіреній машині, яка належить власнику даних. Перед створенням ФКД-файлу є чотири основні операції (рисунок 2.6):

1 шифрування вихідного вмісту файлу за допомогою алгоритму симетричного шифрування;

2 створити, за допомогою ТО, загальну значення X_r;

3 генерування секретних ключових слів для можливостей пошуку;

4 створення перевірок цілісності та автентичності для зашифрованого вихідного файлу даних. Зашифрований файл захешований, а потім отриманий дайджест значення має цифровий підпис за допомогою його шифрування використовуючи приватний ключ $K_{prv\ o}$ власника файлу.

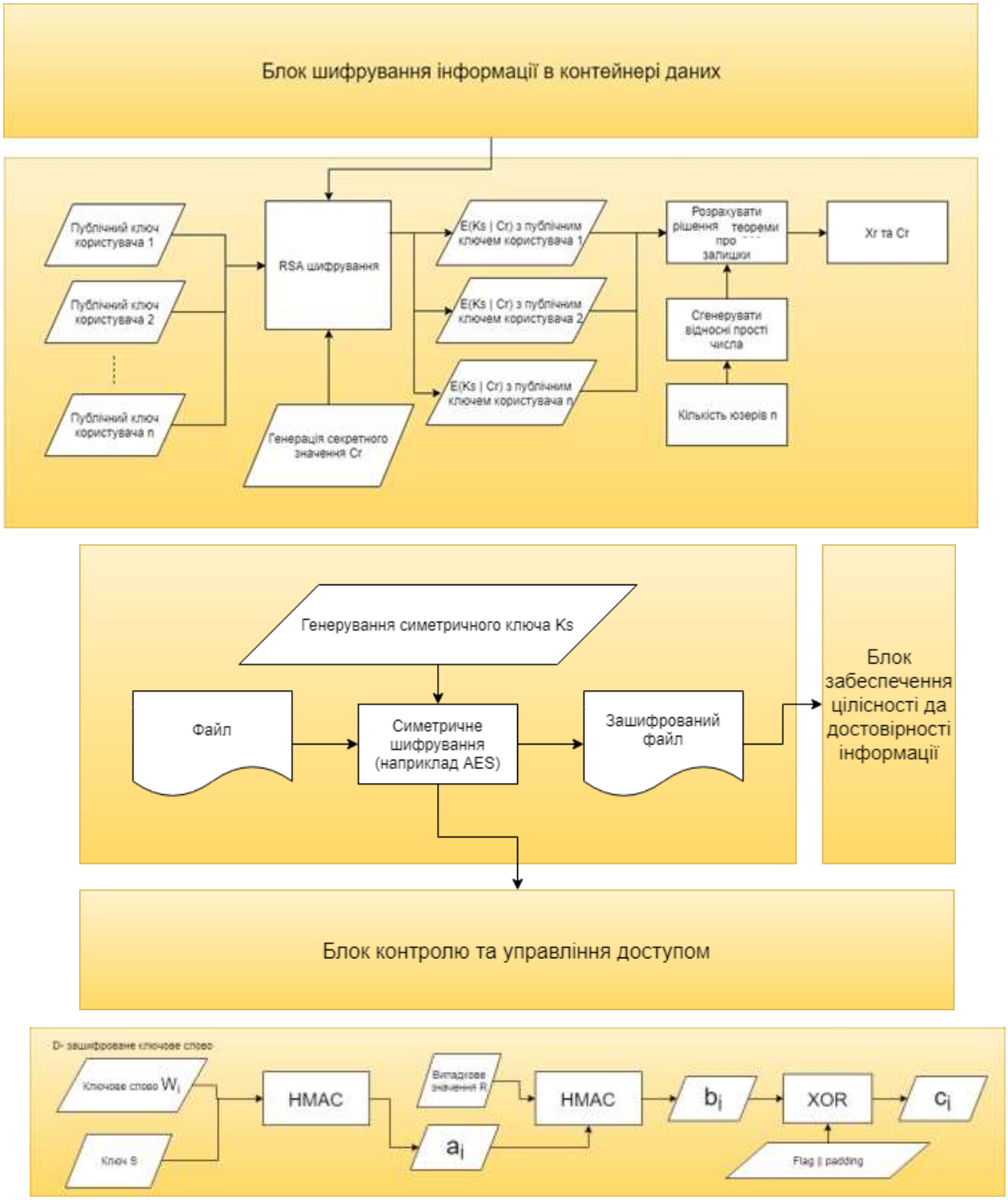

Рисунок 2.6 - Метод забезпечення конфіденційності інформації у web середовищі

Потім ФКД-файл створюється на основі цих чотирьох операцій. Рисунок 2.6 (блок F) показує вихідну структуру ФКД-файлу. Лише авторизовані користувачі мають можливість отримати вихідний файл з ФКД -файлу.

Всі зашифровані ключові слова, де c_i для $i=1,2, ..., z$ та z, як кількість ключових слів, прикріплюються до ФКД-файлу разом з асоційованими рандомними числами R_i.

ФКД-файл тепер може бути відправлений для зберігання у веб середовище. Приватність та цілісність вихідного файлу надійно зберігається і не залежать від веб серверу для забезпечення вимог безпеки, за винятком деяких операцій для їх виконання.

2.6 Переваги розробленого методу

У порівнянні з попередніми роботами, запропоноване рішення є більш безпечним, так як використовує відмінний симетричний ключ для шифрування кожного файлу та інше секретне значення для забезпечення дотримання параметрів контролю доступу для кожного файлу. Це поліпшення знаходить своє відображення в кількох аспектах безпеки та приватності, які наведено нижче.

Користувач не може отримати доступ до зашифрованих даних, перш ніж він/вона забезпечує надійність секретного значення для серверу.

Особистість користувача завжди прихована.

Якщо симетричний ключ K_s і/або секретне значення C_r одного файлу скомпрометовано, інші файли залишаються в безпеці.

Запропоноване рішення поєднує в собі контроль доступу та спільного використання секретного ключа в одному механізмі, з використанням ТО, що призводить до мінімізації обчислювальних та управлінських накладних витрат.

Запропоноване рішення є більш стійким до можливих атак користувачів, позбавлених права доступу, і серверу. Оскільки кожен файл шифрується з іншим симетричним ключем, сервер не може вступити в змову з користувачем, позбавленим права доступу, для того щоб розкрити вміст даних, якщо такий користувач не мав доступу перед позбавленням його таких прав. Особливо, якщо

такий користувач отримав доступ до ФКД-файлу і показав симетричний ключ K_s, то власник даних може видалити файл і використовувати новий ключ K_s для створення нового ФКД-файлу. Таким чином, якщо сервер запобігає збереженню копії цього ФКД- файлу, то він не може вступити в змову з користувачами, які були позбавлені права доступу.

При додаванні нового користувача до списку авторизованих користувачів, які можуть отримати доступ до файлу, власнику даних не потрібно змінювати секретне значення C_r для серверу. Має бути змінене лише загальне значення X_r. Отже, знижується можливість компрометації значення C_r. Крім того, для позбавлення користувача прав доступу, власнику даних лише необхідно змінити C_r цього файлу, тому що ніякі інші файли не використовують одне і те саме старе значення.

Коли власник даних додає себе в якості авторизованого користувача для всіх ФКД-файлів, власник даних і авторизовані користувачі мають залишати у безпеці лише свої пари публічних ключів.

Запропоноване технічне рішення забезпечує цілісність та перевірку аутентичності для кожного файлу і не вимагає зміни параметрів при позбавленні існуючого користувача прав доступу та надання доступу для нового користувача.

Використовуючи запропоноване рішення, всі параметри безпеки незалежно один від одного додаються до кожного ФКД-файлу. Це дозволяє як провайдеру, так і користувачеві працювати з кожним ФКД-файлом більш гнучко та ефективно. Наприклад, користувач або сервер може перемістити ФКД-файл, як правило, не піклуючись про свою безпеку або політику контролю доступу, так як ці функції вже є частиною цього процесу і не залежать від зовнішніх об'єктів. Наприклад, безпека ФКД-файлу не залежить від управління базами даних або доступом до системи, таким чином, не потрібно, звертати увагу на ці параметри при переміщенні даних на сервері або між серверами.

Узагальнюючи, варто зазначити, що обчислення ТО рішення більш ефективне у порівнянні з іншими методами криптографії, що використовують в

криптографічному методі контролю доступу, так як він використовує прості модульні операції без необхідності експоненціальних операцій.

2.7 Висновки по розділу 2

У цьому розділі наведено підхід до ІОБД і його застосування:

- вимоги безпеки забезпечуються всередині даних;
- власник даних відповідає за ці вимоги безпеки протягом усього терміну служби даних;
- вимоги безпеки не залежать від заходів безпеки, що надаються поза межами даних.

Ці концепції відрізняють підхід ІОБД від інших підходів для забезпечення безпеки і приватності даних. Інші підходи забезпечують заходи безпеки на рівні обладнання, платформи або програми, спираючись на провайдера і/або третю сторону, включаючи технології ТС. На противагу цьому, підхід ІОБД забезпечує рішення для забезпечення безпеки і приватності даних для покриття всього життєвого циклу даних. Відповідно, кожен набір даних є самоописовим, самозахисним і з усіма доданими до нього специфікаціями безпеки, і виконання цих специфікацій здійснюється безпечним чином.

Протягом усього життєвого циклу наборів даних власник даних може керувати своїми специфікаціями безпеки і відстежувати їх, включаючи історію використання. Найголовніше, що на першому етапі життєвого циклу даних, етапі створення набору даних, власник даних відповідає за побудову кожного набору даних з усіма вимогами безпеки перед відправкою його до веб-сховища. На тих етапах, що залишилися до етапу знищення набору даних, підтримується відповідність політикам безпеки власників даних. Очікується, що застосування такого підходу поліпшить приватність даних і безпеку у веб-сховищі. Хоча область охоплення обмежена неструктурованими даними, що зберігаються і розділяються в публічному веб-сховищі, пропоноване рішення охоплює найбільш поширені ситуації в веб-сервісах. Більш того, рішення може бути змінено для застосування в інших формах даних. Основні вимоги безпеки для

пропонованого рішення в основному пов'язані із захистом приватності і цілісності при доступі і пошуку зашифрованих даних, що зберігаються на веб-серверах. На основі концепцій ІОБД розглядаються методи безпеки, які можуть задовольняти ці вимоги. Методи, що відповідають вимогам безпеки, будуть адаптовані і використані в запропонованому рішенні ефективним і практичним способом.

Розроблено метод забезпечення конфіденційності інформації у web середовищі на основі ІОБД підходу. Метод розроблений на основі модулів, кожен з яких забезпечує набір сервісів безпеки, які в основному знаходяться в управлінні власника даних. Перший модуль призначений для шифрування даних перед їх аутсорсингом. Другий модуль створює можливості для більш ефективного та дієвого способу управління політикою контролю доступу, які виражені секретним та спільним значенням, які призначені для спільного використання і доступу до контролю даних. Третій модуль використовується для забезпечення можливості безпечного пошуку для зашифрованих даних через генерацію зашифрованих ключових слів. Четвертий модуль виділяє перевірки цілісності та автентичності. Результати всіх попередніх модулів використовуються для створення ФКД-файлу. Використання ФКД-файлу є результатом пошуку рішення що підвищує безпеку і конфіденційність та відповідає веб середовищу. Використання ФКД-файлу, як частини запропонованого рішення, здатне зберегти приватність, цілісність, автентичність даних які було розміщено на веб-сервері, навіть від самого провайдеру з мінімальним впливом на функціональність зашифрованих даних, які відповідають можливостям пошуку та поширення для авторизованих користувачів. Функції безпеки, запропоновані в ФКД-файлі, залежать від криптографічних алгоритмів, які використовуються в описаному підході і знаходяться під управлінням власника даних.

3 РОЗРОБЛЕННЯ ПРОТОТИПУ ПРОГРАМНОГО ПРОДУКТУ ДЛЯ ЗАБЕЗПЕЧЕННЯ КОНФІДЕНЦІЙНОСТІ ДАНИХ У WEB СЕРЕДОВИЩІ

3.1 Вихідні дані на розробку прототипу програмного продукту

Засоби реалізації і середовище різняться для сервера і клієнтської сторони. Для реалізації було використано мову C#, що базувалася на Eclipse IDE. Віртуальна серверна платформа, що представляє сервер сховища даних, була встановлена на персональному комп'ютері з наступними характеристиками:

- процесор: I5-7400 3.0GHz/8GT/s/6MB;
- оперативна пам'ять: 16 ГБ;
- загальна сховище на жорсткому диску: 256 ГБ (SSD).

В якості платформи віртуалізації для створення віртуального сервера обрано VMware VSphere 6, тому що VMware є одним зі світових лідерів в області віртуалізації. Віртуальний сервер був налаштований з такими специфікаціями:

- ОС: Windows Server 2012 R2 Enterprise 64-розрядної версії;
- віртуальна пам'ять: 8 ГБ;
- віртуальний простір для зберігання: 100 ГБ;
- процесор: 2 ядра Intel I5-7400 3.0 ГГц.

Інші віртуальні машини були встановлені на тому ж виділеному апаратному сервері для створення віртуального середовища, що імітує середу віртуалізації, яка використовується у web. Кілька клієнтів використовують один і той же апаратний сервер, використовуючи свої віртуальні машини, що працюють на одному і тому ж апаратному сервері. Ці віртуальні машини ізольовані віртуально з використанням платформи віртуалізації.

Інший фізичний комп'ютер використовувався для моделювання як сторони власника даних, так і авторизованого користувальницького боку. Це був ноутбук з наступними характеристиками:

- процесор: Intel CORE i3 CPU 2.2 ГГц;
- пам'ять: 16 ГБ;
- ОС: Windows 10 64 біт.

На рисунку 3.1 показано налаштування середовища, повідомлень, переданих між об'єктами, і основних операцій з кожного боку.

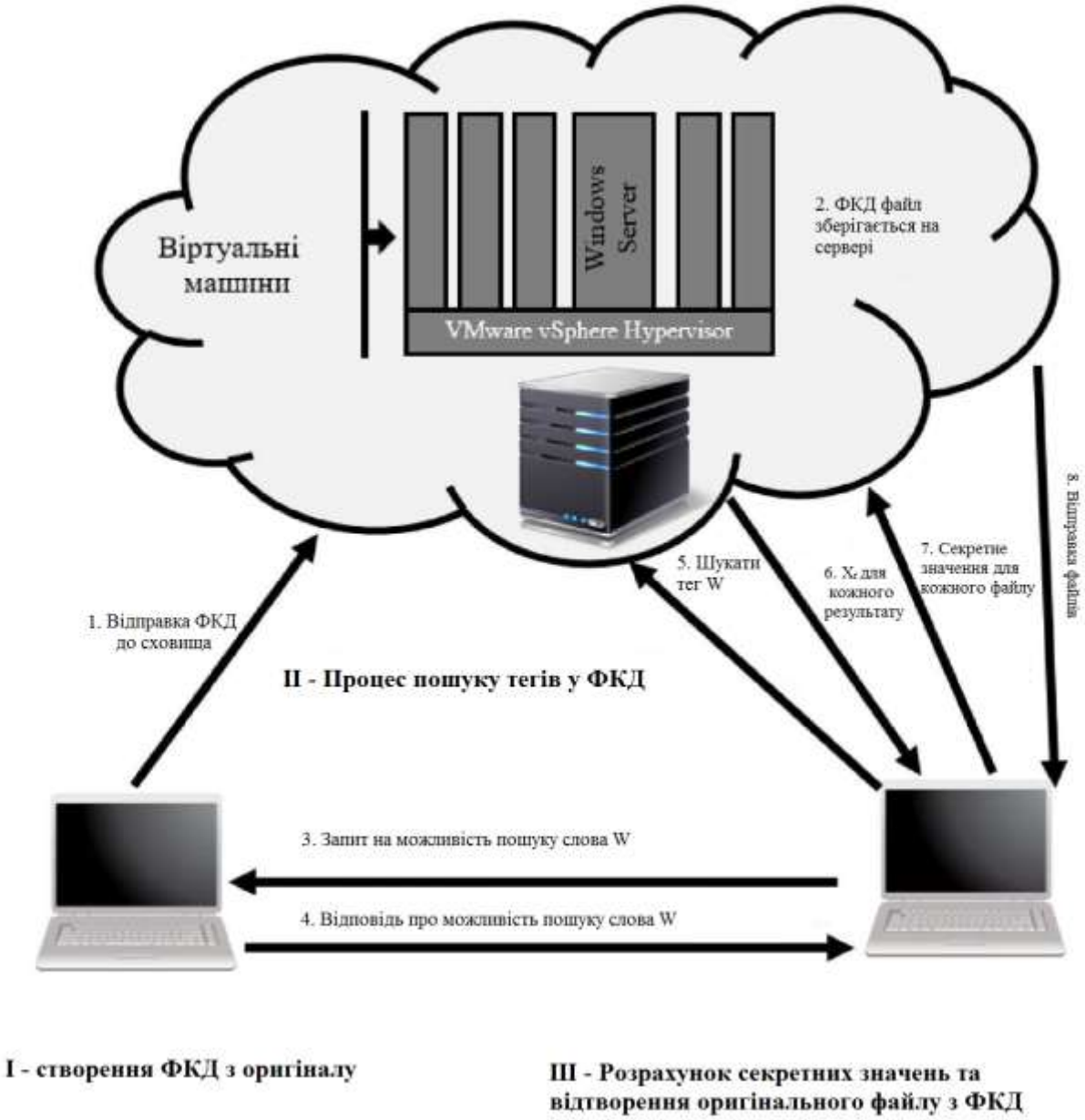

Рисунок 3.1 - Середовище для запропонованого рішення

З боку власника даних було реалізовано і протестовано операції зі створення файлу конфедаційних даних (ФКД) перед його аутсорсингом на веб-сервер. Для авторизованого користувача реалізація та тестування проводилися для операцій шляхом обчислення значення $C_r||K_s$ із загального значення X_r, шляхом дешифрування даних з ФКД, що містить зашифровані дані. На стороні сервера було виконано пошук ключових слів у ФКД. На етапі реалізації одним із завдань був вимір службових даних сховища і накладних витрат на обчислення для виконання операцій для забезпечення безпеки даних.

3.2 Клієнтська сторона

Клієнтська сторона може бути власником даних або авторизованим користувачем. Багато функцій, такі як асиметричний алгоритм шифрування, алгоритм симетричного шифрування та обчислення хеша файлу, використовуються як власником даних, так і авторизованим користувачем. Пошук рішення теореми про остачі (ТО) і шифрування ключових слів виконується тільки власником даних.

Функція шифрування даних за допомогою симетричного алгоритму шифрування є найбільш важливою операцією. Для надійного захисту дані повинні бути зашифровані, навіть якщо вони залишаться у власності власника даних. Тому перша дія після класифікації даних як конфіденційних даних - це шифрування даних з використанням адекватного алгоритму шифрування і сили ключа. У загальному випадку алгоритм симетричного шифрування використовується замість асиметричного шифрування для шифрування великої кількості даних, оскільки він більш ефективний. У цій реалізації файли зашифровуються за допомогою Advanced Encryption Standard (AES), що забезпечується програмою AES Crypt з відкритим вихідним кодом для платформ Windows з використанням 256-бітної довжини ключа, сильнішої довжини ключа для цього алгоритму.

ТО використовується для розподілу двох спільних секретних значень для кожного файлу серед авторизованих користувачів: симетричний ключ K_s, який використовується для шифрування певного файлу і секретного значення C_r, використовуваного для забезпечення дотримання політик контролю доступу для цього певного файлу. У реалізації довжина значення C_r повинна бути зафіксована для всіх файлів, тому легко відрізнити її від симетричного ключа K_s, коли два значення об'єднані як одне значення $C_r||K_s$. Ці два значення зашифровуються одним з асиметричних алгоритмів шифрування $E_{K_{pub\ i}}$ з публічним ключем кожного авторизованого користувача U_i, в результаті шифротекст $a_i = \left(E_{K_{pub\ i}} (C_r \,||\, K_S) \right)$ для *i=1, 2, ... k,* де k - кількість авторизованих користувачів для певного файлу. Отриманий в результаті шифротекст для кожного користувача, якого було авторизовано для доступу до файлу, буде вихідним для операції ТО для обчислення загального значення X_r для цього файлу. Потім для реалізації потрібно знайти рішення одночасної конгруентності ТО. Отже, наприклад, якщо використовуємо алгоритм RSA для шифрування значення $C_r||K_s$, мінімальна довжина результату шифротексту буде становити не менше 1024 біт. Для реалізації платформа .NET надає бібліотеку класів System.Numerics.Biglnteger, що обробляє довгі цілі значення довжиною понад 64 біти. Крім того, для ТО потрібна функція для генерації відносних простих чисел $n_i = n_1, n_2,..., n_k$, де k - кількість авторизованих користувачів. Генерація відносних простих чисел може бути виконана один раз для всіх користувачів, а потім визначена унікально для кожного користувача. Програма, яка використовується для вирішення одночасної конгруентності ТО, базується на алгоритмі Гарнера та розширеному алгоритмі Евкліда (РАЕ) і використовується для обчислення модульного мультиплікативного звороту. Вихідні коди для цих двох алгоритмів доступні у вигляді програми з відкритим вихідним кодом, однак оригінальні вихідні коди реалізовані для підтримки цілих чисел з довжиною в 64 біти. Для реалізації запропонованого рішення вихідні коди модифікуються на основі бібліотеки класів System.Numerics.Biglnteger для підтримки цілих чисел більше

64 бітів, класифікованих на C# як BigIntegers. Отже, результуюча програма для вирішення ТО може обробляти цілі числа довжиною понад 64 біти. CalculateKTO.cs і EuclidHelper.cs - це два файли класів C#, які містять оновлені програмні коди для пошуку рішення ТО і для обчислення модульного мультиплікативного звороту. Вихідні коди цих двох файлів класів C# описані в Додатку А.

Функція CalculateKTO.Execute (Biglnteger [] input_a, Biglnteger [] input_n) використовується для знаходження рішення ТО, де *a* і *n* - вхідні масиви BigInteger значень a_i і n_i. У функції Execute, модульний мультиплікативний зворот обчислюється з використанням функції EuclidHelper.ExecuteInverse (BigInteger input_p, BigInteger input_g), як показано на рисунку 3.2. На рисунку 3.3 показано вихідний код для вирішення ТО, що підтримує BigIntegers. Поверненим значенням зворотної функції є $g^{-1}(mod\ p)$. Найбільший спільний дільник (gcd) двох цілих чисел можна обчислити, використовуючи функцію BigInteger p.gcd (BigInteger g), включену в бібліотеку класу BigInteger.

```
    //Ця функція виконає розширений алгоритм Евкліда, щоб знайти GCD для значень input_a та
    public static System.Numerics.BigInteger[]
        ExecuteEEA(System.Numerics.BigInteger input_a, System.Numerics.BigInteger input_b)
    {
        System.Numerics.BigInteger[] result = new BigInteger[3];
        System.Numerics.BigInteger q;

        if (input_b.Equals(0 as System.Numerics.BigInteger)) {
            result[0] = input_a;
            result[1] = 1 as System.Numerics.BigInteger;
            result[2] = 0 as System.Numerics.BigInteger;
        }
        else
        {
            // В іншому випадку зробити рекурсивний виклик
            q = input_a / input_b;
            result = ExecuteEEA(input_b, a.remainder(input_a));

            System.Numerics.BigInteger s = result[2] * q;
            System.Numerics.BigInteger temp = result[1] - s; // - ans[2]*q;

            result[1] = result[2];
            result[2] = temp;
        }

        return result;
    }
}
```

Рисунок 3.2 - Код для обчислення модульного мультиплікативного звороту

```
using System;

public class CalculateKTO
{

    public static System.Numerics.BigInteger Execute(System.Numerics.BigInteger[] input_a, System.Numerics.BigInteger[]
    {
        if (input_a == null || input_n == null) {
            Console.WriteLine("Жоден з аргументів не може бути пустим");
            return null;
        }
        if (input_a.Length < 2 || input_n.Length < 2) {
            Console.WriteLine("Довжина кожного елемента повинна бути більше двох");
            return null;
        }
        if (input_a.Length != input_n.Length) {
            Console.WriteLine("Довжина обох аргументів повинна дорівнювати");
            return null;
        }
    System.Numerics.BigInteger x = input_a[0];
    System.Numerics.BigInteger nInverse, y, z;
    System.Numerics.BigInteger ni = System.Numerics.BigInteger.One;

    int i = 0;
    while (i < input_a.Length - 1) {
        ni = ni * input_n[i];
        //перевіримо чи n відносно просте
        if (!System.Numerics.BigInteger.One.Equals(input_n[i + 1].gcd(ni))) {
            Console.WriteLine("The n's are not relatively prime->" + input_n[i + 1] + "," + ni);
            return null;
        }
        nInverse = cryptoBig.inverse(n[i + 1], ni);
        if (nInverse.compareTo(0) == -1) {
            nInverse = nInverse + input_n[i + 1];
        }
        z = input_a[i + 1] - x;
        z = z * nInverse;
        y = z.remainder(input_n[i + 1]);
        if (y.compareTo(0) == -1) {
            y = y + n[i + 1];
        }
        x = x + ni * y;
        i++;
    }
    return x;
    }
}
```

Рисунок 3.3 - Код для віднаходження рішення ТО

3.2.1 Шифрування асиметричного ключа

В реалізації було використано RSA в якості асиметричного алгоритму шифрування ключа для шифрування значення Cr||Ks для кожного авторизованого користувача. Довжина Ks може бути 128, 192 або 256 біти і це для секретного значення Cr має бути обрано так, що загальна довжина Cr||Ks буде менше 1024 бітів. Це гарантує, що зашифроване значення Cr||Ks обмежене 1024 бітами. Більш того, обмеження довжини значення Cr||Ks призведе до зменшення

обчислювальних витрат операцій шифрування і дешифрування RSA. Платформа .NET надає пакет який містить кілька бібліотек класів, які включають в себе функції, які використовуються для генерації пар ключів RSA і RSA-операцій шифрування/дешифрування. З цих функцій було запрограмовано два файли C#-класів: клас GenerateRSAkey.cs і клас RSAfurBytes.cs. Клас GenerateRSAkey.cs призначений для створення пари ключів RSA і зберігання їх у файлі. Клас RSAforBytes.cs містить дві функції: rsaEncrypt (byte [] data, String PubKeyFile) для шифрування масиву байтів і rsaDecrypt (byte [] data, String PrivKeyFile) для дешифрування цього масиву байтів. Коди двох класів перераховані в Додатку А.

На рисунку 3.4 показано деякі результати генерації пар ключів RSA і шифрування значення Cr||Ks з використанням згенерованих ключів. Значення Cr||Ks було вибране рівне «1044433225255972339988823253747 9». Це ціле значення було перетворене з Biglnteger в масив байтів, тому що функція rsaEncrypt запрограмована для шифрування даних в форматі масиву байтів. Перетворення результувало у 14 байтах, а шифрування цього значення для десяти користувачів зайняло 15 мілісекунд, як показано в знімку екрана на рисунку 3.4. Отримане зашифроване значення для кожного користувача має довжину 128 байтів (1024 біти), оскільки використовуваний відкритий ключ генерується з довжиною 1024 біти. Кожен зашифрований вихід повинен бути перетворений назад в Biglnteger. Потім отримані зашифровані значення Biglnteger будуть використовуватися в якості вхідних даних для функції CRTforBiglnteger.findSolution (Biglnteger [] a, Biglnteger [] n), де масив *a []* містить значення a_i і *n []*, містить значення n_i; як представлено в рівнянні (2.1) де $a_i = \left(E_{K_{pub\ i}}(C_r \,||\, K_s)\right)$ та n_i - відносне просте число для користувача u_i для *i = 1, 2, ... k*, де *k* - кількість авторизованих користувачів для цього певного файлу. Потім отримане рішення ТО для X_r з findSolution буде прикріплене до захищених даних як частина ФКД файлу.

Результати роботи RSA для 10 користувачів: час виконання - 15 мс; довжина Cr||Ks - 14 байт; довжина зашифрованого - 128 байт.

```
//Кінець блоку шифрування тегів

// Блок шифрування RSA
Stopwatch swRSA = new Stopwatch();
swRSA.Start();
BigInteger C_K = System.Numerics.BigInteger.Parse("10444332252559723399888232537479");
long c = 1044433225; //
Console.WriteLine("Значення Cr " + c);

sbyte[] data = C_K.toByteArray();
/* RSA шифрування масиву байтів Cr||Ks з кожним публічнім ключем авторизованого користувача */
sbyte[] rsaEncoded1 = RSAforBytes.rsaEncrypt(data, "public.key");
sbyte[] rsaEncoded2 = RSAforBytes.rsaEncrypt(data, "public2.key");
sbyte[] rsaEncoded3 = RSAforBytes.rsaEncrypt(data, "public3.key");
sbyte[] rsaEncoded4 = RSAforBytes.rsaEncrypt(data, "public4.key");
sbyte[] rsaEncoded5 = RSAforBytes.rsaEncrypt(data, "public5.key");
sbyte[] rsaEncoded6 = RSAforBytes.rsaEncrypt(data, "public6.key");
sbyte[] rsaEncoded7 = RSAforBytes.rsaEncrypt(data, "public7.key");
sbyte[] rsaEncoded8 = RSAforBytes.rsaEncrypt(data, "public8.key");
sbyte[] rsaEncoded9 = RSAforBytes.rsaEncrypt(data, "public9.key");
sbyte[] rsaEncoded10 = RSAforBytes.rsaEncrypt(data, "public10.key");

swRSA.Stop();
 //Кінець RSA шифрування
Console.WriteLine("RSA шифрування зайняло:" + swRSA.elapsed() + " мс");
Console.WriteLine("Довжина Cr||Ks:" + data.Length + " байт");
Console.WriteLine("Довжина зашифрованого Cr||Ks:" + rsaEncoded1.Length + " байт");
// Кінець шифрування
// Транформування зашифрованого масиву даних в BigInteger
BigInteger transformed1 = new BigInteger(rsaEncoded1);
BigInteger transformed2 = new BigInteger(rsaEncoded2);
BigInteger transformed3 = new BigInteger(rsaEncoded3);
BigInteger transformed4 = new BigInteger(rsaEncoded4);
BigInteger transformed5 = new BigInteger(rsaEncoded5);
```

Рисунок 3.4 - Шифрування RSA для десяти користувачів

3.2.2 Шифрування ключових слів

В реалізації ключові слова для пошуку зашифровуються з використанням коду повідомлень на основі Hash з алгоритмом хешування SHA256. Клас C# HMACSHA256.C# містить функцію encode (String key, String data), яка обчислює значення HMAC-SHA256 для строкових вхідних даних з ключем, який вводиться у вигляді рядка. Функція encode повертає зашифрований рядок у форматі байтового масиву довжиною 256 біти. На рисунку 3.5 показано код функції кодування. Ключ, використовуваний для цієї функції, однаковий для всіх ключових слів.

Рисунок 3.6 показує програмний код для шифрування п'яти ключових слів з використанням функції кодування. Шифрування зайняло 187 мілісекунд.

```
public class HMACSHA256
{
    public static sbyte[] ExecuteEncode(string key_input, string data_input)
    {
        Mac hash_algo = Mac.getInstance("HmacSHA256"); // Вибір хеш алгоритму
        SecretKeySpec secret_key = new SecretKeySpec(key_input.GetBytes(), "HmacSHA256");
        hash_algo.init(secret_key); //Генерація на основі переданого ключа

        return hash_algo.doFinal(data_input.GetBytes()); //Складання хеша
    }
}
```

Рисунок 3.5 - Вирахування коду повідомлень на основі Hash для ключового слова

```
public static void Main(string[] args)
{
    string User = "Shapovalov "; //Id юзера
    //додаємо ключові слова
    string keyword1 = "Information";
    string keyword2 = "Conf";
    string keyword3 = "Network";
    string keyword4 = "KhAI";
    string keyword5 = "Oriented";
    int kw_count = 5; //Кількість зашифрованих тегів
    Stopwatch sw = new Stopwatch();
    sw.Start();
    //Кодуємо алгоритмом SHA-256
    sbyte[] encodedKeyword = HMACSHA256.encode("keyforKeywords", keyword1);
    sbyte[] encodedKeyword2 = HMACSHA256.encode("keyforKeywords",keyword2);
    sbyte[] encodedKeyword3 = HMACSHA256.encode("keyforKeywords",keyword3);
    sbyte[] encodedKeyword4 = HMACSHA256.encode("keyforKeywords",keyword4);
    sbyte[] encodedKeyword5 = HMACSHA256.encode("keyforKeywords",keyword5);
    sw.Stop();
    Console.WriteLine("Розрахунок SHA-256 хешу тегів зайняв:" + sw.Elapsed + " мс");
```

Рисунок 3.6 - Вирахування HMAC-SHA256 для п'яти ключових слів

3.2.3 Перевірка цілісності та достовірності

Перевірка цілісності та достовірності обчислюється шляхом хешування зашифрованого файлу і подальшого шифрування отриманого дайджесту

особистим ключем власника даних. Функція createFileHash (String filename, String method) в класі Hash.cs використовується для обчислення дайджесту файлу з використанням одного з хеш-алгоритмів, таких як SHA-I, SHA-256 і MD5. CreateFileHash приймає в якості вхідних даних ім'я файлу і ім'я хеш-алгоритму (наприклад, SHA-1, SHA-2 або MD5).

Після обчислення дайджесту даного файлу функція повертає значення дайджесту в форматі масиву байтів. Довжина отриманого дайджесту залежить від використовуваного алгоритму.

Наприклад, при використанні SHA-256 підсумкова довжина дайджесту становить 256 біти. Потім отриманий дайджест зашифровано з використанням функції шифрування RSA rsaEncrypt (байт [] data, String PrivKeyFile), де ключ є закритим ключем власника даних (тобто цифровим підписом власника даних). Значення цифрового підпису буде 1024 біти при використанні 1024-бітного закритого ключа і 2048 біт при використанні 2048-бітного ключа. Рисунок 3.7 містить програмний код для обчислення значення FSignature в програмному коді, який використовується для перевірки цілісності та достовірності.

```
// Розрахунок вхідного дайджесту файла і сигнатури власника даних
string private_key_file = "prkey_dataowner.key"; //приватний ключ власника даних
Stopwatch swDigest = new Stopwatch();
swDigest.Start();
//Підрахунок дайджесту файла
sbyte[] fileDigest = Hash.createFileHash(InputFile, "SHA-256");
// Підпис файлу
sbyte[] fileSignature = RSAforBytes.rsaEncrypt(fileDigest, private_key_file);
```

Рисунок 3.7 - Код для обчислення доказів цілісності та достовірності

3.2.4 Створення захищеного файлу власником даних

Створення ФКД-файлу в основному пов'язане з раніше описаними операціями, а саме: симетричним шифруванням, пошуком рішення ТО, HMAC-SHA256, шифруванням RSA і підписом зашифрованого дайджесту повідомлень

вхідного файлу. Всі виходи цих операцій і пов'язаної з ними інформацією включаються до складу вмісту файлу ФКД. Клас CreateSecureFile.cs створено і розроблено для використання у створенні файлу ФКД з зашифрованого файлу. Перед запуском програми CreateSecureFile вихідний файл повинен вже бути зашифрований програмою AES Crypt і визначений наступними параметрами, з прикладами присвоєних значеннь параметру, вказаного для кожного з параметрів:

1 Визначене наперед ім'я власника або ID.

string User = "Shapovalov"; //Id юзера

2 Визначені наперед рядки, число і ключ шифрування ключових слів пошуку.

//додаємо ключові слова

string keywordl = "Information";

string keyword2 = "Conf';

int kw_count = 2; //Кількість зашифрованих тегів

//Кодуємо алгоритмом SHA-256

sbyte[] encodedKeyword = HMACSHA256.encode ("keyforKeywords", keywordl);

sbyte[] encodedKeyword2 = HMACSHA256.encode ("keyforKeywords", keyword2);

3 Визначені наперед окремо значення $C_r||K_s$ і C_r.

Biglnteger C_K = System.Numerics.BigInteger.Parse ("10444332252559723399888232537479");

long c = 1044433225;

4 Визначені наперед всі відносні прості значення (тобто елементи масиву Biglnteger n []) авторизованими користувачами цього файлу.

Biglnteger prime1 = System.Numerics.BigInteger.Parse ("215143111331331"); // Користувач 1

BigInteger prime2 = System.Numerics.BigInteger.Parse ("103143111331"); // Користувач 2

5 Визначені наперед файли, що містять публічний ключ для кожного з авторизованих користувачів.

sbyte [] rsaEncoded1 = RSAforBytes.rsaEncrypt(data, "public.key"); // Користувач 1

sbyte [] rsaEncoded2 = RSAforBytes.rsaEncrypt(data, "public2.key"); // Користувач 2

6 Визначений наперед файл, що містить приватний ключ власника даних.

string private_key_file = "prkey_dataowner.key"; //приватний ключ власника даних

7 Визначені наперед алгоритми хешування (наприклад, SHA-1 або SHA-256) для дайджесту зашифрованого файлу.

sbyte [] fileContainingDigest = Hash.createFileHash (InputFile, "SHA-256");

Після визначення вищевказаних параметрів програма готова до виконання. Виконання спочатку обчислює HMAC-SHA256 ключових слів, як показано на рисунку 3.6 вище. Потім вона шифрує значення $C_r||K_s$ з відкритим ключем кожного користувача, приклад якого показано на рисунку 3.7. Отримані зашифровані значення з відносними цілими числами користувачів використовуються для знаходження загального значення за допомогою функції CRTforBiglnteger.fmdSolution. Наприклад, обчислення ТО-рішення для п'яти користувачів, де довжина їх відносних простих чисел становить 1025 біти, а значення $C_r||K_s$ становило 1024 біта, зайняло близько 5,1 мілісекунди, а отриманий X_r був 5122 біти.

Потім програма запропонує користувачеві ввести зашифрований файл, який буде перетворений в файл ФКД з прикріпленими параметрами безпеки. Програма обчислює дайджест вхідного файлу і підписує дайджест особистим ключем власника даних. При використанні розробленої програми для обчислення дайджесту файлу розміром 40,7 МБ і створення цифрового підпису

файлу, коли алгоритм хеша SHA-256 і 1024-бітний закритий ключ RSA були використані для цього конкретного файлу й зайняло це 780 мілісекунд.

Після обчислення всіх необхідних параметрів програма запропонує користувачеві ввести шлях, ім'я та розширення для вихідного файлу ФКД з розширенням «dcs» за замовчуванням. Потім програма починає створювати ФКД-файл в якості вихідного. Програмний код можна знайти в Додатку А.

На мові програмування C# для вставки нового потоку біт на початку файлу потрібно створити новий файл з новою інформацією метаданих файлу і вихідним вмістом файла. Тому для створення ФКД-файлу потрібно створити новий файл з параметрами, в якості метаданих і зашифрованого вмісту файлу. Це додасть обчислювальних витрат, особливо для великих файлів, в копіюванні вмісту зашифрованого файлу в файл ФКД.

3.2.5 Операції на стороні авторизованого користувача

Авторизований користувач вимагає, в основному, чотирьох операцій, тобто обчислення значення $C_r||K_s$ із загального значення X_r, відтворення вихідного зашифрованого файлу з файлу ФКД, перевірки цілісності та перевірки аутентичності і, нарешті, дешифрування зашифрованого файлу для створення файлу у вигляді нешифрованого тексту. Всі ці операції реалізовані в одному файлі класу ReadSecureFileB.cs. Користувач повинен визначити наступні параметри. Приклади присвоєних значень параметру наведені для кожного з параметрів:

1 Унікальний відносно простий користувач

```
BigInteger n = new BigInteger("3812938192391");
```

2 Файл, що містить закритий ключ користувача.

```
string pr_key_file = "pr.key";
```

3 Файл, що містить відкритий ключ власника даних.

```
String publ_key_file = "publicUser.key";
```

Прості і закриті ключі користувача використовуються для обчислення $C_r||K_s$ із значення X_r, а відкритий ключ власника даних використовується для перевірки цілісності та аутентичності файлу ФКД.

Після визначення значення $C_r||K_s$ авторизований користувач u_i тепер може використовувати C_r для завершення процедури контролю доступу та K_s для дешифрування вихідного зашифрованого файлу після його завантаження. Оскільки комунікаційна частина процедури контролю доступу не розглядається в цій реалізації, то код читає X_r безпосередньо з файлу ФКД, поки сервер повинен це зробити і відправити його користувачеві під час процедури контролю доступу. Основна увага в цій частині реалізації була присвячена вимірюванню накладних витрат обчислень, викликаних обчисленням $C_r||K_s$ на стороні користувача. Як показано в таблиці 3.2 вище програмний код виконує в основному дві функції. Перша функція пов'язана з алгоритмом ТО, який заснований на простій модульної функції (тобто $X_r \bmod n_i$), і, отже, розрахунок може бути виконаний дуже швидко. Наприклад, використовуючи цю функцію, обчислюючи значення X_r з довжиною 5121 біти, для п'яти користувачів, та n з довжиною 1024 біти займає близько 96,565 мікросекунд. Друга функція - це операція розшифровки ТО для отриманого зашифрованого значення з першої функції. Ця операція дешифрування зайняла близько 516 мілісекунд.

Файл ФКД містить інформацію, таку як значення X_r, докази цілісності і аутентичності, на додаток до зашифрованих даних у файлі. Коли клас ReadSecureFileB.cs запущений, програма попросить користувача ввести вхідний файл ФКД, а потім ім'я і розширення для вихідного файлу. Програмний код для зчитування інформації з вхідного файлу ФКД і подальшого виведення зашифрованих даних у файлі.

Код, призначений для того, щоб дозволити авторизованому користувачу або провайдеру хмари читати з ФКД-файлу, ім'я власника даних, X_r, C_r і підписаний дайджест файлу зашифрованих даних. Однак тільки авторизовані користувачі можуть отримати ключ K_s, який потім може використовуватися для

дешифрування зашифрованого файлу. Крім того, авторизований користувач може також перевірити цілісність і достовірність файлу даних.

Після отримання зашифрованого файлу даних з файлу ФКД і читання прикріпленого підпису до файлу, авторизований користувач може перевірити цілісність і аутентичність файлу зашифрованих даних.

Процес перевірки призводить до обчислювальних накладних витрат, а кількість накладних витрат залежить від розміру файлу. Наприклад, перевірка цілісності та аутентичності файлу розміром 40,7 МБ займає близько 720 мілісекунд.

Після перевірки цілісності та аутентичності зашифрованого файлу даних зашифрований файл даних може бути дешифрований за допомогою програмного забезпечення AES Crypto з ключем Ks для цього файлу. Велика частина обчислювальних витрат пов'язана з розшифровкою зашифрованого файлу, особливо для великих файлів.

3.3 Реалізація на стороні сервера

На стороні сервера основна реалізована операція - це процес пошуку в файлі ФКД прикріплених зашифрованих ключових слів. Клас search. cs включає код, необхідний для пошуку файлів ФКД. Для прискорення процесу пошуку використовується алгоритм пошуку рядків Кнут-Моррис-Пратт. Цей алгоритм є одним з найбільш ефективних алгоритмів пошуку, і він може знайти точну відповідність шаблонів в ряді цифрових даних. Клас KMPMatch.cs містить реалізацію алгоритму з відкритим вихідним кодом. KMPMatch.cs забезпечує функцію: indexOf (bytes[] data_input, bytes[] pattern_input).

Функція використовує алгоритм пошуку рядків для пошуку першого входження певного образу масиву байтів (тобто byte[] pattern) в заданому діапазоні масиву байтів (тобто byte[] data). Ця функція використовується в класі search.cs для пошуку заданого зашифрованого ключового слова в файлах ФКД. Реалізований алгоритм пошуку, заснований на алгоритмі пошуку рядків,

призначений для пошуку тільки з боку файлу ФКД, який містить масив байтів, що містять зашифровані ключові слова. Наприклад, для п'яти зашифрованих ключових слів діапазон буде 164 байта, оскільки перші 4 байта файлу ФКД представляють кількість ключових слів і кожне зашифроване ключове слово зайняло 32 байти. Обмеження діапазону пошуку дуже важливе для скорочення часу, необхідного для пошуку, особливо для великих файлів. Якщо процес пошуку не обмежується діапазоном, що містить зашифровані ключові слова, пошук по всьому файлу ФКД буде витрачати ресурси і час на обчислення, оскільки в зашифрованому файлі даних немає зашифрованих ключових слів з можливістю пошуку.

Перед запуском класу search.cs необхідно вказати шлях до каталогу, який містить файли ФКД для пошуку, як показано нижче:

String path = "C:\\some_fold";

Код, запитує рядок ключового слова, а потім обчислює значення HMAC- SHA256 для ключового слова. Отримане значення поміщається в масив байтів. Цей код включений в клас search.cs для експериментів. У реальному житті цей код запускається на стороні власника даних. Хмарний сервер отримує тільки зашифровані ключові слова (тобто масив байтів), які використовуються для пошуку файлів ФКД, що містять ключові слова.

Після введення ключового слова і обчислення його HMAC-SHA256, search.cs відкриває кожен з файлів ФКД і зчитує кількість зашифрованих ключових слів, вбудованих в файл. З цілого числа ключових слів, прикріплених до ФКД-файлу, код обчислює діапазон пошуку, а потім виконує пошук ключового слова з використанням алгоритму пошуку рядків. Якщо образ відповідає, код виводить на екран шлях і ім'я файлу ФКД, що містить це ключове слово. Код повторює ту ж процедуру для всіх файлів ФКД в зазначеній директорії. Для всіх файлів код показує час, необхідний для пошуку кожного з файлів.

3.4 Реалізація на стороні клієнта

У цьому пункті обговорюються результати і проблеми впровадження в відношенні накладних витрат з точки зору зберігання і обчислень. Накладні витрати визначаються як час, витрачений на обчислення. Результати засновані на припущенні, що всі значення параметрів вже згенеровані, такі як пари ключів RSA, відносні прості числа, значення C_r і K_s. В цьому пункті основна увага приділяється стороні власника даних і авторизованій користувальницькій стороні, оскільки серверна сторона вже розглянута в попередньому пункті.

Існують проблеми з впровадженням і накладні витрати при створенні файлу ФКД на машині власника даних. Спочатку файл даних буде зашифрований з використанням AES (використовується інша хмара з симетричними алгоритмами шифрування), і отриманий файл зашифрованих даних стане основною частиною файлу ФКД. У таблиці 3.1 показані накладні витрати на зберігання і час виконання шифрування файлів різних типів даних з використанням алгоритму AES з розміром ключа 256 біти.

Таблиця 3.1 - Зберігання та тимчасові накладні витрати для шифрування AES

Назва файлу	Розмір в байтах	Розмір після шифрування в байтах	Збільшення розміру в байтах	AES шифрування в секундах
Sample.docx	14,175	14,485	310	<1
visio.vsd	45,029	45,341	312	<l
photo.jpg	64,819	65,130	318	<1
video1.wmv	25,327,026	25,327,338	312	1.6
video2.mov	41,670,503	41,670,812	309	2.3
video3.avi	136,318,116	136,318,429	313	14.8
video4.mov	598,787,322	598,787,634	311	86.7

Накладні витрати на зберігання складають близько 300 байт для всіх файлів. Збільшується обчислювальне навантаження зі збільшенням розміру файлу, щоб досягти більш ніж однієї хвилини для файлу з розміром 571 МБ. Ця операція є основою для будь-якого методу, заснованого на шифруванні даних, перш ніж дані будуть відправлені до серверу на збереження. Крім того, важливо вибрати сильний алгоритм шифрування і довший ключ для досягнення більш високої безпеки. Однак власники даних мають гнучкість, відповідно до їх вимог безпеки, щоб зменшити обчислювальні накладні витрати, використовуючи більш короткий ключ або слабший алгоритм шифрування з меншими обчислювальними витратами. Загалом, процес шифрування несе в собі найбільші обчислювальні витрати на стороні власника даних в запропонованому в цій роботі методі.

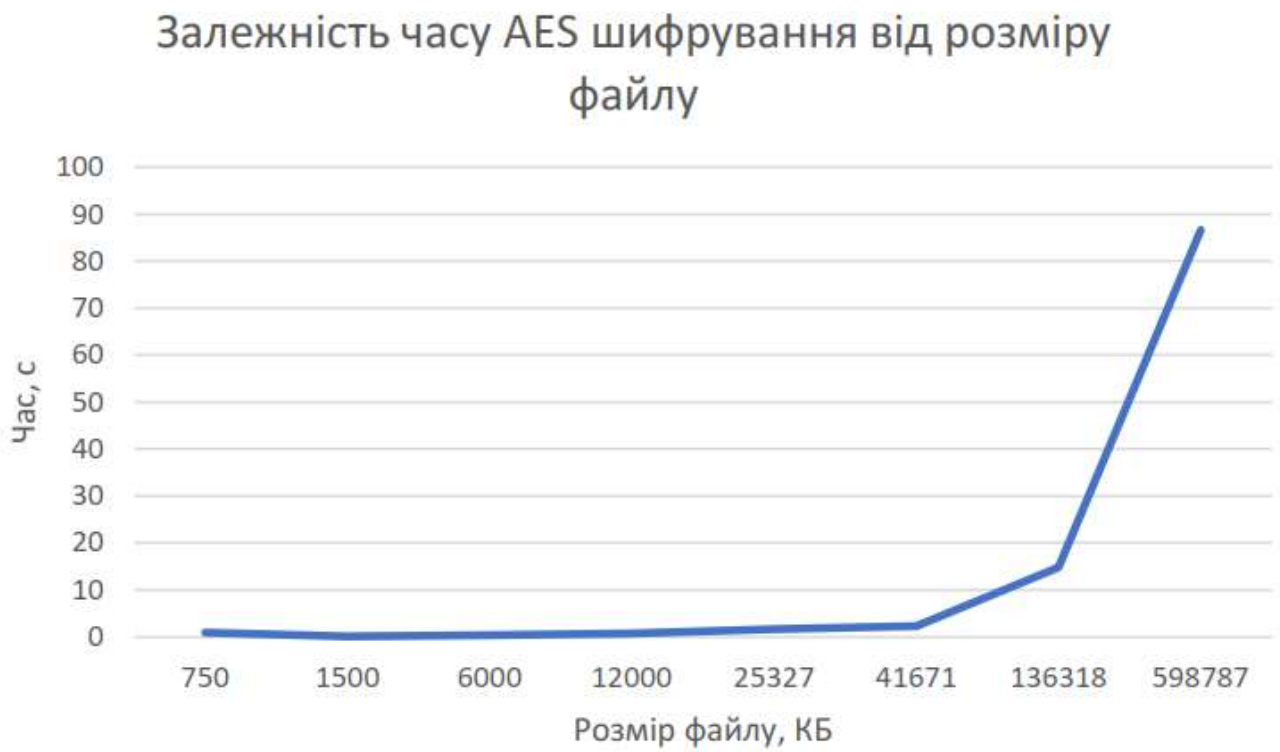

Рисунок 3.8 - Залежність часу AES шифрування від розміру файлу

Обчислення ТО-рішення для визначення значення X_r є суттєвою частиною запропонованого рішення. На основі цієї цінності засновані політики обміну ключами і контролю доступу. На розрахунок ТО не впливає розмір файлу, але на нього впливає кількість користувачів і алгоритм асиметричного шифрування, використовуваний для шифрування значення $C_r||K_s$. У таблиці 3.1 показані витрати часу на зберігання і час при обчисленні ТО-рішення для різних користувачів з двох до шести. Ці обчислення виконувалися, коли значення $C_r||K_s$ було зашифровано з 1024-бітовим RSA і кожне відносне просте число було 1022-бітним. Результати показують, що сама операція ТО вимагає незначного часу виконання. Наприклад, для шести користувачів обчислення ТО-рішення зайняло близько 6 мілісекунд. Час виконання обчислення значення обумовлено головним чином процесом шифрування ТО і збільшується зі збільшенням кількості користувачів. Однак, оскільки процес шифрування ТО виконується для фіксованого значення довжини $C_r||K_s$, загальний час виконання пошуку все ще незначний і задовільний. Наприклад, загальний час становив 31 мілісекунду для шести користувачів, а загальний час збільшувалася приблизно на 2 мілісекунди для кожного додаткового користувача. Більшість витрат, викликаних операцією ТО - це накладні витрати на зберігання, які збільшуються зі збільшенням числа користувачів. На рисунку 3.9 показано, що на підставі результатів таблиці 3.2 розмір X_r лінійно збільшується приблизно на 1000 біт для кожного додаткового користувача, коли довжина ключа ТО дорівнює 1024 біти. Наприклад, для шести користувачів розмір дорівнював 6142 біт.

Таблиця 3.2 - Зберігання та тимчасові накладні витрати для розрахунку загального значення Xr

№ користувача	ТО (1024 біти) шифрування для $C_r\|\|K_s$ в мс	Підрахунок ТО-рішення X_r в мс	Загальний час підрахунку X_r в мс	Довжина X_r в бітах
1	21	3	24	2049

Закінчення таблиці 3.2

№ користувача	ТО (1024 біти) шифрування для Cr||Ks в мс	Підрахунок ТО-рішення Xr в мс	Загальний час підрахунку Xr в мс	Довжина Xr в бітах
2	22	4	26	3073
3	23	4	27	4097
4	24	5	29	5118
5	25	6	31	6142

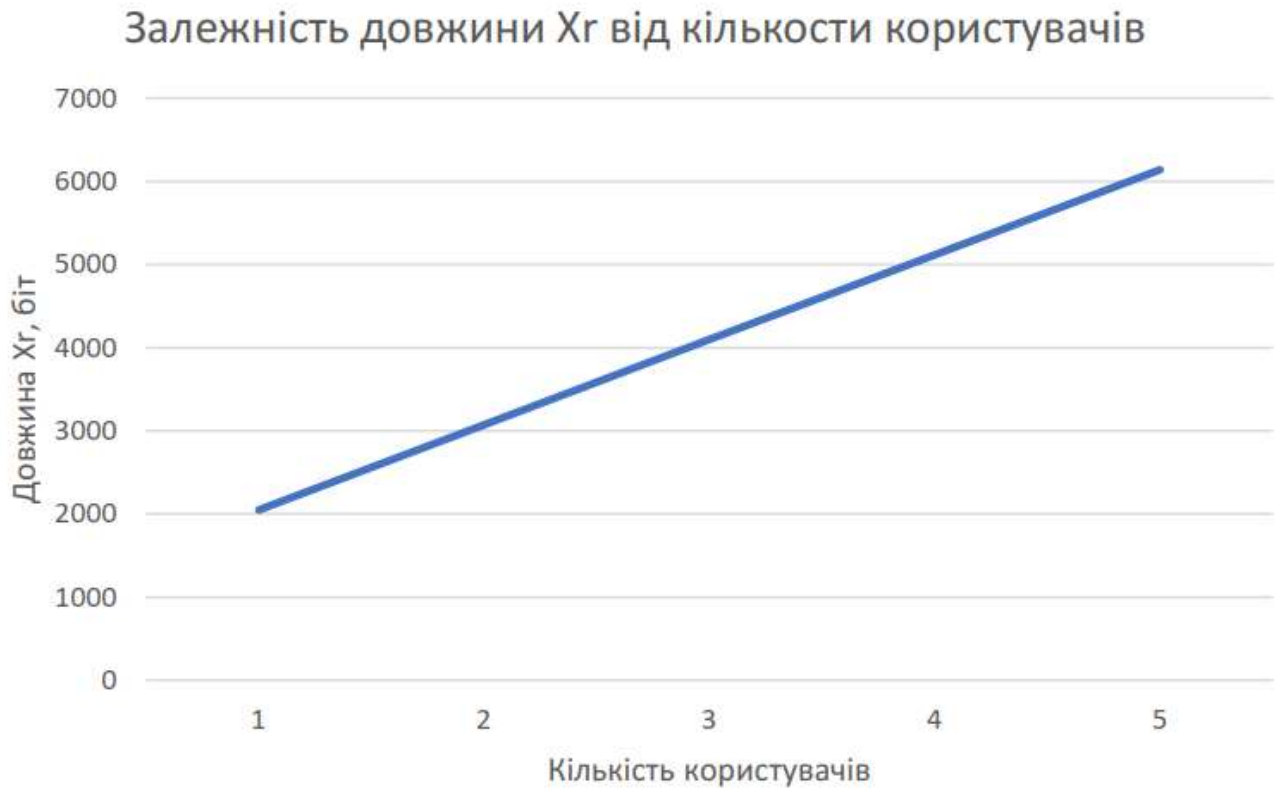

Рисунок 3.9 - Розмір X_r в бітах у зв'язку з кількістю користувачів

Додавання зашифрованих ключових слів в файл ФКД необхідне для можливості пошуку. У таблиці 3.3 показані накладні витрати на зберігання і час виконання, пов'язані з додаванням від одного до п'яти зашифрованих ключових

слів. Результати показують помірні обчислювальні накладні витрати, близько 200 мілісекунд, а накладні витрати на зберігання збільшується на 256 біт для кожного додаткового ключового слова. Виконується тільки один раунд HMAC, в той час як пропоноване рішення вимагає двох раундів і операції XOR. Причиною цього є скорочення обчислювальних витрат, а також використання алгоритму пошуку рядків для пошуку на сервері. Коли операція XOR використовується при шифруванні ключових слів, алгоритм пошуку рядків не може застосовуватися, оскільки при використанні операції процес пошуку XOR буде включати операцію XOR перед пошуком того ж шаблону, який суперечить механізму пошуку рядків. Більш того, використання одного раунду HMAC з SHA-256 або SHA-512 має забезпечувати достатній рівень безпеки. Реалізація дозволяє користувачам використовувати HMAC-SHA512 для більшої безпеки, але накладні витрати на зберігання збільшаться в два рази. Наприклад, при використанні HMAC-SHA256 для п'яти ключових слів загальна необхідна пам'ять становить 1280 біт. При використанні HMAC-SHA512 для п'яти ключових слів необхідне сховище буде 2560 біт. Власник даних повинен віднайти компроміс між рівнем безпеки, необхідним для ключових слів і накладними витратами на сховище і час виконання. Також необхідно ретельно визначити кількість необхідних ключових слів, щоб зменшити накладні витрати, не втрачаючи зручність пошуку.

Таблиця 3.3 - Накладні витрати на сховище і час виконання для додавання зашифрованих ключових слів

№ ключового слова	Тривалість HMAC-SHA256 в мс	Загальний розмір виходу в байтах
1	196	256
2	196	512
3	201	768
4	197	1024
5	199	1280

Додавання доказів цілісності та аутентичності в файл ФКД має важливе значення, але це призводить до додаткових витрат обчислення і зберігання. У таблиці 3.3 показані накладні витрати на зберігання і час виконання при розрахунку цілісності та аутентичності файлів різних типів і розмірів. Хеш-функція, яка використовується для дайджесту зашифрованих файлів була SHA-256, а ключ RSA для шифрування дайджесту був 1024-біт. Через збільшення розміру файлу збільшується обчислювальне навантаження, а накладні витрати на зберігання фіксуються на 1024 біт, що є результатом шифрування 256-бітного дайджесту файлу RSA з довжиною в 1024 біт.

Таблиця 3.4 - Зберігання та тимчасові накладні витрати для підтвердження цілісності та достовірності

Вхідне ім'я файлу	Розмір в байтах	Час підрахунку дайджесту та підпису в мс	Накладні витрати на зберігання в байтах
Sample.docx.aes	14,485	51	128
visio.vsd.aes	45,341	51	128
photo.jpg.aes	65,130	133	128
video1.wmv.aes	25,327,338	1910	128
video2.mov.aes	41,670,812	775	128
video3.avi.aes	136,318,429	8714	128
video4.mov.aes	598,787,634	50977	128

Для створення ФКД-файлу необхідно, щоб всі параметри були вже обчислені. Загальні накладні витрати на зберігання залежать від кількості користувачів, яких авторизовано для доступу до файлу, кількості прикріплених ключових слів і специфікацій використовуваних алгоритмів. У таблиці 4.5 показано, що загальні накладні витрати при створенні ФКД-файлу складають 952

байта для файлів різного розміру. Файли ФКД були створені з п'ятьма прикріпленими зашифрованими ключовими словами і повинні використовуватися п'ятьма авторизованими користувачами. Алгоритми використовувалися зі специфікаціями, визначеними в реалізаціях, тобто RSA-1024, AES-256, SHA-256. У разі великих файлів цими накладними витратами на зберігання може бути знехтувано.

Таблиця 3.5 - Загальні накладні витрати на зберігання для створення файлу ФКД для п'яти користувачів і п'яти ключових слів

Вхідне ім'я файлу	Розмір в байтах	Розмір ФКД-файлу в байтах	Накладні витрати на зберігання в байтах
Sample.docx.aes	14,485	15,437	952
visio.vsd.aes	45,341	46,293	952
photo.jpg.aes	65,130	66,082	952
video1.wmv.aes	25,327,338	25,328,290	952
video2.mov.aes	41,670,812	41,671,764	952
video3.avi.aes	136,318,429	136,319,381	952
video4.mov.aes	598,787,634	598,788,586	952

Процес створення файлу ФКД вимагає копіювання вмісту зашифрованого файлу в новий файл ФКД. Ця операція вимагає додаткового час, як показано в таблиці 3.6. Цього часу можна уникнути, якщо одночасно шифрування AES і обчислення зашифрованого файлового дайджесту виконуються одночасно при створенні ФКД- файлу, тобто, вихідний файл даних зашифрований в блоках, а потім кожен блок систематизується і використовується для створення файлу

ФКД за один раунд. Тому на цей раз це пов'язано з реалізацією і не розглядається як накладні витрати щодо запонованого рішення.

Основні операції, що виконуються власником даних для створення файлу ФКД - це шифрування AES, обчислення значення X_r, обчислення HMAC для ключових слів, обчислення дайджесту зашифрованого файлу і шифрування дайджесту. Таблиця 3.7. показує загальний час виконання цих операцій для різних розмірів файлів. Значення X_r було розраховано для п'яти користувачів і було додано п'ять зашифрованих ключових слів.

Таблиця 3.6 - Тимчасові накладні витрати для копіювання вмісту зашифрованого файлу в файл ФКД

Вхідне ім'я файлу	Розмір в байтах	Копіювання зашифрованого вмісту файлу в ФКД-файл в мс
Sample.docx.aes	14,485	33
visio.vsd.aes	45,341	42
photo.jpg.aes	65,130	90
video1.wmv.aes	25,327,338	252
video2.mov.aes	41,670,812	391
video3.avi.aes	136,318,429	1284
video4.mov.aes	598,787,634	79560

Загальний час виконання становид менше 4 секунд для файлів розміром менше 50 МБ і близько 20 секунд для файлу розміром 130 МБ. Для файлу в 571 МБ загальний час виконання становив близько 1,7 хвилини. З результатів, показаних в таблиці 4.7, ясно, що обчислення параметрів, необхідних для контролю доступу, спільного використання ключів і пошуку, має менший вплив на обчислення. На противагу цьому, шифрування AES, за яким слід обчислювати цілісність та достовірність, має найбільший вплив на обчислення. Тому в запропонованому рішенні операція симетричного шифрування бере на себе

більшість обчислювальних накладних витрат, особливо для великих файлів. Однак будь-яке рішення, засноване на шифруванні даних перед відправкою даних в хмару, вимагає принаймні одного раунду шифрування даних.

Авторизований користувач повинен провести три операції; обчислення значення $C_r||K_s$ із значення X_r, перевірку цілісності та аутентичності зашифрованого файлу даних і дешифрування файлу зашифрованих даних. Таблиця 3.8 показує час виконання операцій для різних розмірів файлів ФКД. Для розшифровки AES потрібно найбільший час виконання, за яким слід перевірити цілісність та автентичність зашифрованого файлу даних. Обчислення значення $C_r||K_s$ займає невеликий час виконання. Відтворення вихідного файлу зашифрованих даних з файлу ФКД можна виконати на стороні сервера, а потім сервер відправляє вихідний файл зашифрованих даних з його підписаним дайджестом замість файлу ФКД. Отже, накладні витрати на копіювання зашифрованого вмісту з файлу ФКД в новий відтворений файл виключаються на стороні користувача.

Таблиця 3.7 - Загальний час, необхідний для створення ФКД-файлу

Розмір вхідного файлу в байтах	AES в секундах	Підрахунок Xr для 5 користувачів, сек	Підрахунок дайджесту файлу та підпису, сек	HMAC-SHA256 для 5 ключових слів, сек	Загальний час, сек
14,074	<1	0.029	0.003	0.199	-1
44,032	<1	0.029	0.004	0.199	-1
65,812	<1	0.029	0.005	0.199	-1
26,246,026	1.6	0.029	0.474	0.199	2.302
42,750,493	2.3	0.029	0.720	0.199	3.248
137,237,016	14.8	0.029	5.050	0.199	20.078
598,787,322	86.7	0.029	17.897	0.199	104.825

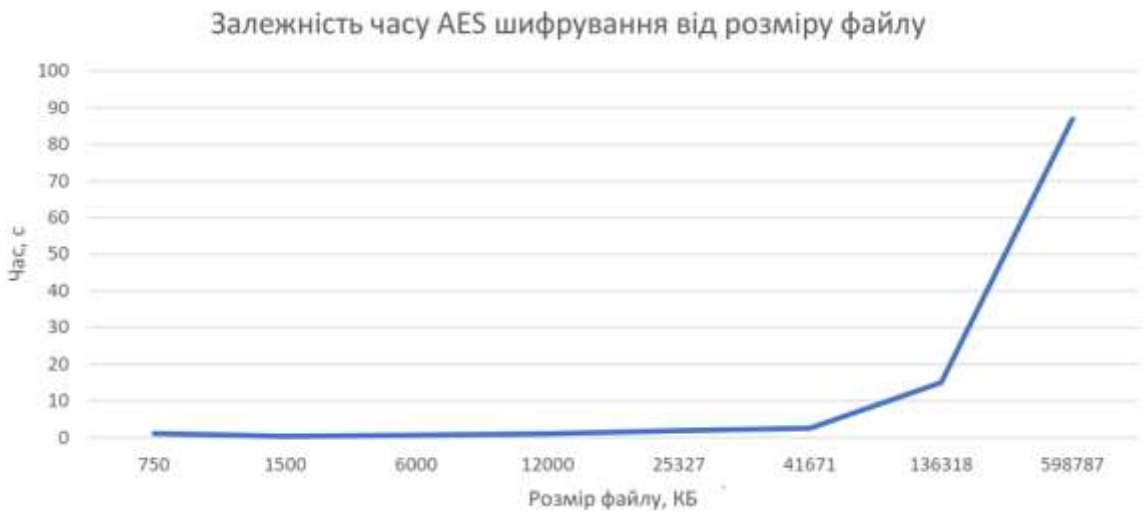

Рисунок 3.10 - Залежність сумарного часу від розміру файла

Таблиця 3.8 - Виконання операцій на стороні користувача для різних файлів ФКД

Вхідна назва ФКД-файлу	Розмір, байт	Створення оригінального зашифрованого файлу, мс	Перевірка цілісності та аутентичності, мс	Віднаходження Cr||Ks із Xr, мс	AES дешифрування, сек
Samp1e.docx.dcs	15,337	214	3	29	<1
visio.vsd.dcs	45,290	220	4	29	<1
photo.jpg.dcs	67,082	350	5	29	<1
video1.wmv.dcs	26,247,290	1448	474	29	3.5
video2.mov.dcs	42,751,754	1457	720	29	2
video3.avi.dcs	137,238,282	11520	5050	29	32
video4.mov.dcs	598,788,586	13198	17897	29	32.7

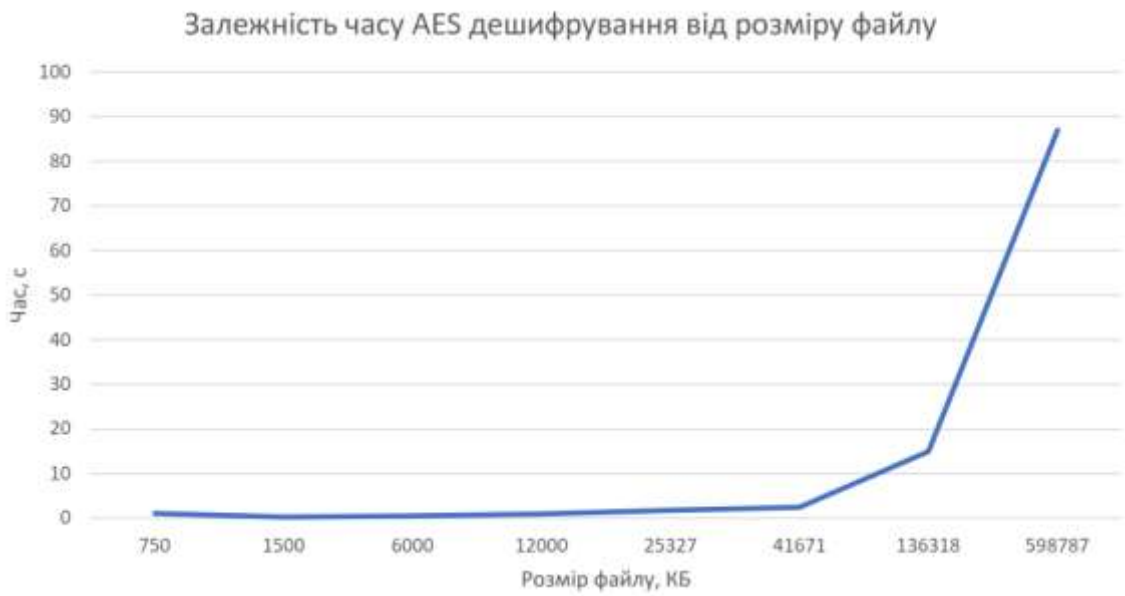

Рисунок 3.11 - Графік залежності від часу

3.5 Перевірка ефективності пропонованих рішень

Для визначення ефективності планується порівняти швидкість роботи розробленого ПП з аналогом - розподіленою програмною системою охорони здоров’я [22]. В таблиці 3.9 приведені результати роботи аналогічного ПП.

Таблиця 3.9 – Результати роботи аналогічного ПП

Розмір файлу (КБ)	Час шифрування (мс)
750	344
1500	421

Далі розмір двох файлів доводиться до приблизно такого ж розміру, а потім апроксимуються до точок 750 КБ, 1500 КБ. Результати порівняння предствлені на рисунку 3.13.

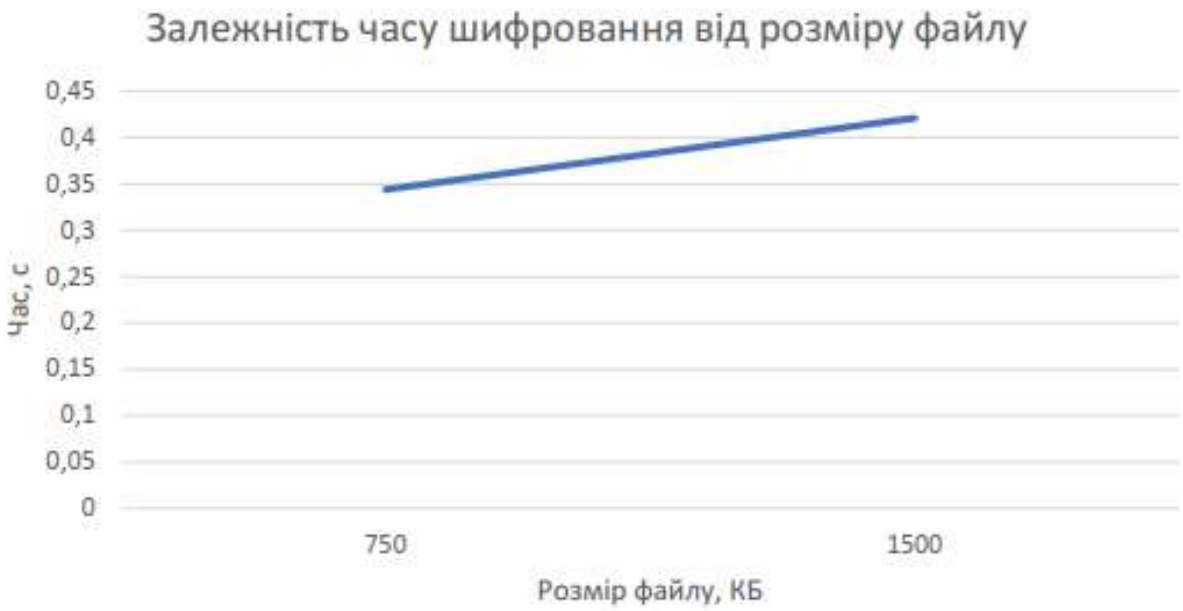

Рисунок 3.12 - Швидкість роботи аналогічного ПП

Рисунок 3.13 - Порівняння швидкості рботи для малих файлів

3.5 Висновки по розділу 3

В розділі основна увага приділялася навантаженню на зберігання і обчислення, викликаним використанням запропонованого рішення з боку

клієнта і сервера. Для обох сторін була встановлена експериментальна платформа, і для реалізації основних операцій звпропонованого рішення була використана платформа Rider від JetBrains для мови програмування C#. Для операцій власника даних, основний код для створення файлу ФКД з зашифрованого файлу був записаний у файлі класу C# з ім'ям CreatSecureFile.cs. Цей клас обчислює загальне занчення X_r для будь-яких користувачів, викликавши функцію findSolution з CRTfurBiginteger.cs, яку було реалізовано для обчислення ТО-рішення при використанні цілих чисел розміром більше 64 біти. Крім того, було створено CreatSecureFile для обчислення інших параметрів безпеки, а саме зашифрованих ключових слів для можливості пошуку, перевірки цілісності та достовірності.

Було реалізовано читання параметрів безпеки в ФКД-файлі і відтворення вихідної зашифрованої файлової форми. Також файл містить операції, необхідні на стороні авторизованого користувача, а саме, виявлення секретних значень C_r і K_s з X_r; і перевірки цілісності та аутентичності зашифрованого файлу. Відкрите вихідне програмне забезпечення AES Crypto використовувалося для шифрування вихідного файлу на стороні власника даних, а потім для його дешифрування на стороні користувача.

Файл класу Search.cs був реалізований на сервері для пошуку заданого ключового слова в файлах ФКД. Для підвищення продуктивності при створенні ФКД- файлу була додана інформація про те, скільки ключових слів і яка їх довжина. Отже, функція пошуку може обчислювати діапазон всередині ФКД-файлу, який містить зашифровані ключові слова, тому процес пошуку буде обмежений цим діапазоном. Таким чином, час пошуку не буде залежати від довжини файлу. Крім того, використовувався алгоритм пошуку KMP для підвищення швидкості пошуку.

Результати проведених операцій показують, що запропоноване рішення є простим і не вимагає складних операцій. Крім того, накладні витрати на зберігання при створенні файлу ФКД є низькими в порівнянні з наданими

функціями. Цими функціями є: створення ФКД-файлу з можливостями пошуку, політикою прихованого контролю доступу та цілісністю і достовірністю, незалежно від того, де він зберігається в хмарі. Запропоноване рішення може бути практично реалізовано з мінімальними витратами на обчислення і зберігання. Доведено, що запропоноване рішення дуже ефективне, так як воно не вимагає складних методів деривації ключів, і файл даних не потрібно шифрувати більше одного разу.

ВИСНОВКИ

В дипломному проекті досліджується інформаційно-орієнтована безпека даних, як важливий підхід до підвищення безпеки та приватності даних, що зберігаються клієнтами у веб-системах, особливо в середовищі публічних веб-сховищ, де дані можуть часто переміщатися з одного сервера на інший і можуть бути доступні іншим кінцевим об'єктам. Основна увага приділяється захисту фактичних даних від можливих ризиків безпеки у веб-середовищі без повної впевненості в тому, що провайдер або третя сторона є надійними. На відміну від підходів, спрямованих на забезпечення безпеки обладнання і додатків, що обробляють дані у веб-сховищі, які вимагають щоб власник даних довіряв провайдеру або третій стороні для забезпечення необхідних вимог безпеки та приватності.

Запропоновано підхід до досліджується інформаційно-орієнтованої безпеки даних, метою якого є підвищення безпеки і приватності даних у веб-сховищі. Запропонована структура заснована на визначенні вимог безпеки безпосередньо у фактичних даних, тому дані є самоописовими, самозахищеними та самоохороняємими протягом усього їх життєвого циклу. Власник даних зберігає повну відповідальність за визначення і управління безпекою та приватністю своїх даних, переданих у веб-сховище. Крім того, приватність даних і приватність користувачів, які звертаються до даних, залишаються захищеними навіть від провайдерів. Захист даних не залежить від надійності провайдера і не вимагає надійності третьої сторони. В результаті підвищення рівня безпеки та приватності, що досягається за допомогою запропонованого підходу, переваги використання веб-сервісів можуть бути досягнуті без втрати контролю над безпекою та приватністю даних власником даних або оприлюднення даних за можливих порушень безпеки і приватності, напротивагу зберіганню даних у власному приватному обчислювальному середовищі користувача. Наведено очікувані переваги застосування підходу ІОБД: всі заходи безпеки і вимоги

надаються з самого набору даних; власник даних відповідає за конфігурацію, управління і моніторинг даних та їх характеристик безпеки протягом усього терміну служби даних; безпека і приватність даних і користувачів, які мають доступ до даних, не повинні покладатися на провайдерів або довірених третіх осіб.

Запропоновано метод, заснований на створенні файлу, який називається ФКД-файл, власником даних для кожного файлу даних, який повинен бути переданий у веб-сховище. Кожен ФКД-файл містить зашифрований вихідний файл даних і набір параметрів безпеки, які використовуються для забезпечення дотримання політик контролю доступу, пошуку ключових слів і перевірки цілісності та аутентичності файлу даних. Виключно, власник даних встановлює і управляє цими параметрами безпеки для кожного ФКД-файлу протягом усього його життєвого циклу. ФКД-файл зберігає захищений файл даних у веб-середовищі, і тільки авторизовані користувачі можуть отримати доступ до зашифрованого файлу даних в своїх довірених доменах. Отже, дані залишаються захищеними від можливих загроз безпеки всередині і поза веб-середовища, навіть від можливих шкідливих дій самого провайдера.

Кожен ФКД-файл має свої власні політики контролю доступу, прикріплені до нього, які приховані від провайдера, що несе відповідальність за дотримання політик доступу. Крім того, кількість і особистості авторизованих користувачів, які можуть мати доступ до даних теж приховані від провайдера.

Для обчислення загального значення, що містить два секретних значення використано теорему про остачі. У запропонованому рішенні ТО використовується для розподілу двох секретних значень; одним значенням є секретний ключ для дешифрування файлу даних та інше секретне значення для аутентифікації авторизованих користувачів в процедурі доступу без виявлення особистостей користувачів. В результаті запропоноване рішення поєднує в собі загальний секретний ключ і примусове застосування політик контролю доступу в одному механізмі, що зводить до мінімуму накладні витрати обчислень і управління ключами. Кожен ФКД-файл має своє загальне значення секретного

ключа, що додається, та яке включає певні політики доступу і симетричний ключ цього ФКД-файлу. Ні власник даних, ні користувачі не повинні зберігати або замінювати цей ключ, так як кожен ключ буде надійно прикріплений до відповідного файлу. Тільки авторизовані користувачі можуть отримати доступ і дешифрувати файл даних в ФКД-файлі. Сервер, який зберігає ФКД-файли, не може виявити секретне значення. Якщо один ФКД-файл стає скомпрометованим, інші ФКД-файли не будуть порушені, тому що кожен файл має свої унікальні секретні значення. Тільки власник даних може змінювати політики контролю доступу для кожного ФКД-файлу шляхом безпечної і ефективної зміни його секретного значення. Реалізація та результати експерименту, представлені у дипломному проекту, показують, що обчислення секретного значення має низькі обчислювальні витрати.

В ФКД-файлі також можна безпечно здійснювати пошук, прикріплюючи до нього зашифровані ключові слова. Тому пошук можна здійснювати навіть у нетекстових файлах по секретним ключовим словам. Ключові слова зашифровані і прикріплені до ФКД-файлу власником даних, і тільки авторизовані користувачі можуть шукати ключові слова в ФКД-файлах. Прикріплені ключові слова зашифровуються власником даних, щоб приховати їх від неавторизованих осіб, включаючи провайдера. Ця можливість пошуку не залежить від бази даних, що зберігається у провайдера, і вимагає лише звичайного процесу шифрування і простого методу пошуку. На етапі імплантації процедура пошуку призначена для прискорення процесу пошуку, використовуючи ефективний алгоритм пошуку рядків Кнута-Морріса-Пратта. Результати виконання реалізації показує задовільну ефективність пошуку, в той час як ключові слова залишаються надійно захищеними.

Пропонований підхід забезпечує платформу, незалежну від обчислювального середовища, і підходить для складної і динамічної природи веб-середовища, особливо у випадку публічного веб-сховища. ФКД-файл має свій зашифрований файл даних, а вимоги безпеки до нього прив'язані як параметри безпеки, які також відсилають до метаданих безпеки. Кожен ФКД-

файл має свої незалежні параметри безпеки, включаючи докази цілісності та аутентичності, прикріплені до нього, і може управлятися індивідуально власником даних. Тому власники даних можуть бути впевнені в безпеці своїх ФКД-файлів, навіть якщо файли передаються між різними провайдерами та різними географічними локаціями. Незалежно від веб-сервера, в якому знаходиться ФКД-файл у веб-середовищі, авторизовані користувачі можуть здійснювати пошук, доступ та перевірку цілісності, оскільки вони вбудовані в файл і не залежать від обчислювального середовища. Більш того, коли провайдер створює копії для резервного копіювання або поліпшення доступу для декількох користувачів, або з яких-небудь інших причин, параметри безпеки вихідного ФКД-файлу підтримуються скопійованими версіями.

Результати, отримані при тестуванні реалізованих операцій запропонованого методу, показують, що реалізація, а отже і відповідне рішення, проста і не вимагає складних операцій, що вимагають великих обчислювальних ресурсів, а також високого обслуговування. Крім того, запропоноване рішення може бути практично використане для будь-якого типу файлу даних з помірними накладними витратами з точки зору зберігання і обчислювальних ресурсів та дозволяє власнику даних гнучко вибирати алгоритми шифрування і хешування на основі необхідних сильних сторін безпеки і додатків.

ПЕРЕЛІК ПОСИЛАНЬ

1 A break in the clouds: towards a cloud definition / M.Lindner, J. Caceres, R. Luis, V. Luis. // ACM SIGCOMM Computer Communication Review. - 2009. - С. 50-55.

2 Mell P. The NIST Definition of Cloud Computing [Електронний ресурс] / P. Mell, T. Grance // National Institute of Standards and Technology. - 2011. - Режим доступу до ресурсу: https://csrc.nist.gov/publications/detail/sp/800-145/final

3 Yeun C. Cloud computing security management / C. Yeun, S. Almulla. // Engineering Systems Management and Its Applications (ICESMA). - 2010. - 2nd International Conference on Engineering System Management & Applications

4 Mutavdzic R. Cloud computing architectures for national, regional and local government / Ratko Mutavdzic. // MIPRO. - 2010.

5 Patidar S. A Survey Paper on Cloud Computing / S. Patidar, D. Rane, P. Jain. // Advanced Computing & Communication Technologies (ACCT). - 2012.

6 Lin D. Data protection models for service provisioning in the cloud / D. Lin, A. Squicciarini. // The ACM Symposium on Access Control Models and Technologies (SACMAT). - 2010. - С. 183-192.

7 Controlling data in the cloud: outsourcing computation without outsourcing control / [J. Staddon, R. Masuoka, J. Molina та ін.]. // ACM Conference on Computer and Communications Securit. - 2009. - №9. - С. 85-90.

8 Privacy preserving cloud data access with multi-authorities / J.Taeho, L. Xiang- Yang, W. Zhiguo, W. Meng. // INFOCOM. - 2013 - 2013 Proceedings IEEE.

9 Chen L. Adaptive Data Replicas Management Based on Active Data-centric Framework in Cloud Environment / L. Chen, D. Hoang. // High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC). - 2013 - №10.

10 Kangsheng S. Historical development of the Chinese remainder theorem / Shen Kangsheng. // Archive for History of Exact Sciences. - 1988. - №38. - С. 285-305.

11 Bogomolny A. Chinese Remainder Theorem [Електронний ресурс] / Alexander Bogomolny // Interactive Mathematics Miscellany and Puzzles - Режим доступу до ресурсу: https://www.cut-the-knot.org/blue/chinese.shtml

12 Sorenson J. Genetic algorithms for the extended GCD problem / V. Piehl, J. Sorenson, and N. Tiedeman // Journal of Symbolic Computation. - 1997.

13 Fast Parallel Garner Algorithm for Chinese Remainder Theorem / [L. Yongnan, X. Limin, L. Aihua та ін.]. // International Conference on Network and Parallel Computing (NPC). - 2017. - №9. - С. 164-171.

14 Wassenberg J. Fast keyed hash/pseudo-random function using SIMD multiply and permute / J. Wassenberg, J. Alakuijala. // Google Research. - 2016. - №2016.

15 Khali H. A system-level architecture for hash message authentication code / H. Khali, R. Mehdi, A. Araar. // ICECS. - 2005. - №12.

16 Ferguson N. Practical Cryptography / N. Ferguson, B. Schneier., 2003.

17 big-O notation [Електронний ресурс] // U.S. National Institute of Standards and Technology. - 2004. - Режим доступу до ресурсу: https://xlinux.nist.gov/dads/HTML/bigOnotation.html

18 Distributed Searchable Symmetric Encryption / [C. Bosch, P. Andreas, B. Leenders та ін.]. // Privacy, Security and Trust (PST). - 2014. - №12.

19 Zonghua Y. Research on asymmetric searchable encryption / Y. Zonghua, W. Yudong. // AIP Conference Proceedings. - 2017.

20 Безпека даних в хмарних середовищах: матеріали V міжнар. наук.-практ. конф. з інформаційних систем та технологій, 1-2 грудня 2017 р., Київ / відп. ред. А. В. Писаренко. - К.: Вид-во ТОВ «Інжиніринг», 2017. - 28 с.

21 Smoot S.R. Private Cloud Computing / Stephen R. Smoot, Nam-KeeTan. - Morgan Kaufmann Publishers B, 2011. - 424 с.

22 A cloud security framework for a data centric WSN application / S.Sayantani, D. Rounak, D. Suman, N. Sarmistha. // International Conference on Distributed Computing and Networking. - 2016. - №17.

23 Rountree D. The Basics of Cloud Computing: Understanding the Fundamentals of Cloud Computing in Theory and Practice / Derrick Rountree, IleanaCastrillo. - Newnes, 2013. - 172 с.

24 Qing Li Applications integration in a hybrid cloud computing environment: modelling and platform / Qing Li, Zeyuan W., Weihua Li, Jun Li, Cheng Wang, Ruiyang Du // Enterprise Information Systems. - 2013. - 7 (3). - С. 237-271

25 NIST Special Publication 500-293, US Government Cloud Computing Technology Roadmap, Release 1.0 (Draft), Volume II Useful Information for Cloud Adopters, 2011. - 85 с.

26 Mell P. Effectively and Securely Using the Cloud Computing Paradigm / P. Mell, T. Grance. - NIST. Information Technology Laboratory. - 2009. - Том 10 - С. 7

ДОДАТОК А
Листінг фрагментів коду

```
using System;
using System.Diagnostics;
public class CreateSecureFile {
public static void Main(string[] args) {
string User = "Shapovalov"; //Id юзера
//додаємо ключові слова
string keyword1 = "Information"; string keyword2 = "Conf";
int kw_count = 2; //Кількість зашифрованих тегів
Stopwatch sw = new Stopwatch();
sw.Start();
//Кодуємо алгоритмом SHA-256
sbyte[] encodedKeyword = HMACSHA256.encode("keyforKeywords", keyword1);
sbyte[] encodedKeyword2      = HMACSHA256.encode("keyforKeywords",keyword2);
sw.Stop();
Console.WriteLine("Розрахунок SHA-256 хешу тегів зайняв:" + sw.Elapsed + " мс");
//Кінець блоку шифрування тегів
// Блок шифрування RSA Stopwatch swRSA = new Stopwatch();
swRSA.Start();
BigInteger C_K = System.Numerics.BigInteger.Parse("10444332252559723399888232537479");
long c = 1044433225; Console.WriteLine("Значення Cr " + c);
sbyte[] data = C_K.toByteArray();
/* RSA шифрування масиву байтів Cr||Ks з кожним публічнім ключем авторизованого користувача */
sbyte[] rsaEncoded1 = RSAforBytes.rsaEncrypt(data, "public.key");
sbyte[] rsaEncoded2 = RSAforBytes.rsaEncrypt(data, "public2.key");
sbyte[] rsaEncoded3 = RSAforBytes.rsaEncrypt(data, "public3.key");
sbyte[] rsaEncoded4 = RSAforBytes.rsaEncrypt(data, "public4.key");
sbyte[] rsaEncoded5 = RSAforBytes .rsaEncrypt(data, "public5.key");
sbyte[] rsaEncoded6 = RSAforBytes.rsaEncrypt(data, "public6.key");
sbyte[] rsaEncoded7 = RSAforBytes.rsaEncrypt(data, "public7.key");
sbyte[] rsaEncoded8 = RSAforBytes.rsaEncrypt(data, "public8.key");
```

```
sbyte[] rsaEncoded9 = RSAforBytes.rsaEncrypt(data, "public9.key");
sbyte[] rsaEncoded10 = RSAforBytes.rsaEncrypt(data, "public10.key");
swRSA.Stop();
//Кінець RSA шифрування
Console.WriteLine("RSA шифрування зайняло:" + swRSA.elapsed() + "мс");
Console.WriteLine ("Довжина Cr||Ks:" + data.Length + " байт");
Console.WriteLine(" Довжина зашифрованого Cr||Ks:" + rsaEncoded1.Length + " байт");
// Кінець шифрування
// Транформування зашифрованого масиву даних в Biglnteger
BigInteger transformed1 = new BigInteger(rsaEncoded1);
BigInteger transformed2 = new BigInteger(rsaEncoded2);
BigInteger transformed3 = new BigInteger(rsaEncoded3);
BigInteger transformed4 = new BigInteger(rsaEncoded4);
BigInteger transformed5 = new BigInteger(rsaEncoded5);
// Відносні прості числа
BigInteger prime 1 = System.Numerics.BigInteger.Parse
("215143111331331113151515157719135111111111 1111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 11
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 11
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 11
111111111111111111111111");
BigInteger prime2 = System.Numerics.BigInteger.Parse
("215143111331331113151515157719135111111111 1111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 11
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 11
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 11
111111111111111111111113");
BigInteger prime3 = System.Numerics.BigInteger.Parse
("215143111331331113151515157719135111111111 1111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 11
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 11
111111111111111111111115");
```

```
BigInteger prime4 = System.Numerics.BigInteger.Parse
("215143111331331113151515157719135111111111 1111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111 11
1111111111111111111111111111111111111111111111111111111111111111111 11
1111111111111111111111111111111111111111111111111111111111111111111 11
11111111111111111111117");
BigInteger primen5 = System.Numerics.BigInteger.Parse
("215143111331331113151515157719135111111111 1111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111 11
1111111111111111111111111111111111111111111111111111111111111111111 1
1111111111111111111111111111111111111111111111111111111111111111111 11
11111111111111111111119");
//знайти рішення для ТО
Stopwatch swTO = new Stopwatch();
swTO. Start();
BigInteger [] a = new BigInteger [] {transformed1, transformed2, transformed3, transformed4,
transformed5};
BigInteger [] n = new BigInteger [] {prime1, prime2, prime3, prime4, prime5};
// Пошук рішення ТО Xr для цього файлу
BigInteger x = CRTforBigInteger.findSolution(a, n);
swTO. Stop();
Console.WriteLine("Довжина кожного n в бітах: " + n[0].bitLength());
Console.WriteLine("Підрахунок Xr зайняв: " + swTO.Elapsed() + " мс" );
//Перевірка, чи Xr більше 0
if (x.compareTo(System.Numerics.BigInteger.Zero) > 0) {
Console.WriteLine("Рішення ТО Xr: " + x);
Console.WriteLine(" Довжина Xr в бітах:" + x.bitLength());
}
else {
Console.WriteLine("Значення Xr негативне");
}
//вкажіть зашифрований файл для створення ФКД-файлу
Console.WriteLine ("Введіть ім'я файла з розширенням: ");
string InputFile;
Console.ReadLine(InputFile);
```

```
File InFile = new File(InputFile); //input file
if (!InFile.exists()) {
Console.WriteLine("File does not exist.");
Environment.Exit(O);
}
// Розрахунок вхідного дайджесту файла і сигнатури власника даних
string private_key_file = "prkey_dataowner.key"; //приватний ключ власника даних
Stopwatch swDigest = new Stopwatch();
swDigest.Start();
//Підрахунок дайджесту файла
sbyte[] fileContainingDigest = Hash.createFileHash(InputFile, "SHA-256");
// Підпис файлу
sbyte[] fileSignature = RSAforBytes.rsaEncrypt(fileContainingDigest, private_key_file);
Console.WriteLine("Довжіна підпису файла у байтах: " + FSignature.Length);
//Зупинка таймеру обчислювання дайджеста та сігнатури
swDigest.Stop();
Console.WriteLine("Підрахунок дайджеста та сігнатури зайняв:" + swDigest.Elapsed() + " ms");
// Кінець розрахунків дайджеста та сігнатури генерування ФКД-файла
Console.Write("Введіть назву та росширення вихідного ФКД-файла: ");
BufferedReader outc = new System.IO.StreamReader(System.in);
string outfileExtension = outc.ReadLine();
File file = new File(outfileExtension); //output file
Stopwatch swOutput = new Stopwatch();
swOutput.Start();
RandomAccessFile outputFile = new RandomAccessFile(file, "rw");
outputFile.writeInt(kw_count); //кількість ключових слів
outputFile.write(encodedKeyword);
outputFile.write(encodedKeyword2);
outputFile.write(encodedKeyword3);
outputFile.write(encodedKeyword4);
outputFile.write(encodedKeyword5);
outputFile.writeUTF(User); // запис Id користувача
sbyte[] xr = x.toByteArray(); //Конвертація BigInteger Xr в масив байтів
int xlength = xr.Length; outputFile.writeLong(Cr); // Cr
outputFile.writeInt(xlength); // довжина Xr
```

```
outputFile.write(xr); //спільне Xr
outputFile.write(FSignature); //додавання цифрового підпису зчитування
RandomAccessFile in = new RandomAccessFile(InFile, "r"); // Перезапис байтів з вхідного у вихідний sbyte[] buf = new sbyte[1024];
int len;
while ((len = in.read(buf)) > 0)
{
f.write(buf, 0, len);
}
Console.WriteLine("Нова довжина ФКД-файлу:" + f.length());
f.seek(0);
f.close();
in.close();
Console.WriteLine("ФКД-файл зроблено");
swOutput.Stop();
Console.WriteLine("ФКД-файл сгенеровано за:" +swOutput.Elapsed() + " мс");
Console.WriteLine(DateTime); // show end date and time
}
private static string DateTime {
get
{
DateFormat df = new SimpleDateFormat("yyyy-MMdd_ hh:mm:ss");
df.TimeZone = TimeZone.getTimeZone("PST");
return df.format(DateTime.Now);
}
}
public class HMACSHA256 {
public static sbyte[] ExecuteEncode(string key_input, string data_input)
{
Mac hash_algo = Mac.getInstance("HmacSHA256"); // Вибір хеш алгоритму
SecretKeySpec secret_key = new SecretKeySpec(key_input.GetBytes(), "HmacSHA256");
hash_algo.init(secret_key); //Генерація на основі переданого ключа
return hash_algo.doFinal(data_input.GetBytes()); //Складання хеша
}
}
```

```
using System;
public class CalculateTO
{
// Розрахунок рішення для теореми про остачі
public static System.Numerics.BigInteger Execute(System.Numerics.BigInteger[]
input_a, System.Numerics.BigInteger[] input_n) {
if (input_a == null || input_n == null) {
Console.WriteLine("Жоден з аргументів не може бути пустим");
return null;
}
if (input_a.Length < 2 || input_n.Length < 2) {
Console.WriteLine("Довжина кожного елемента повинна бути більше двох");
return null;
}
if (input_a.Length != input_n.Length) {
Console.WriteLine(" Довжина обох аргументів повинна дорівнювати");
return null;
}
System.Numerics.BigInteger x = input_a[0];
System.Numerics.BigInteger nInverse, y, z;
System.Numerics.BigInteger ni = System.Numerics.BigInteger.One;
int i = 0;
while (i < input_a.Length - 1) {
ni = ni * input_n[i]; //перевіримо чи n відносно просте
if (!System.Numerics.BigInteger.One.Equals(input_n[i + 1].gcd(ni))) {
Console.WriteLine("The n's are not relatively prime->" + input_n[i + 1] + "," + ni);
return null;
}
nInverse = cryptoBig.inverse(n[i + 1], ni);
if (nInverse.compareTo(0) == -1) { nInverse = nInverse + input_n[i + 1]; }
z = input_a[i + 1] - x;
z = z * nInverse;
y = z.remainder(input_n[i + 1]);
if (y.compareTo(0) == -1) { y = y + n[i + 1]; }
x = x + ni * y;
```

```
i++;
}
return x; // рішення ТО}
}
public class EuclidHelper
{
//повертає g в ступені (-1) (mod p)
public static System.Numerics.BigInteger
ExecuteInverse(System.Numerics.BigInteger input_p, System.Numerics.BigInteger input_g)
{
System.Numerics.BigInteger[] result = ExtendedEuclid(input_p, input_g);
if (result[2].compareTo(0 as System.Numerics.BigInteger) != -1) { return input_p + result[2];
}
return result[2];
}
//Ця функція виконає розширений алгоритм Евкліда, щоб знайти GCD для значень input_a та
input_b
public static System.Numerics.BigInteger[] ExecuteEEA(System.Numerics.BigInteger   input_a,
System.Numerics.BigInteger input_b) {
System.Numerics.BigInteger[] result = new BigInteger[3];
System.Numerics.BigInteger q;
if (input_b.Equals(0 as System.Numerics.BigInteger)) {
 result[0] = input_a;
result[1] = 1 as System.Numerics.BigInteger;
result[2] = 0 as System.Numerics.BigInteger;
}
else {
// В іншому випадку зробити рекурсивний виклик q = input_a / input_b;
result = ExecuteEEA(input_b, a.remainder(input_a));
System.Numerics.BigInteger s = result[2] * q;
System.Numerics.BigInteger temp = result[1] - s; // - ans[2]*q;
result[1] = result[2];
result[2] = temp;
}
return result;}}
```