МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ВАСИЛЬЧЕНКО ДАНИЇЛ НАЗАРОВИЧ

Допускається до захисту:
В.о. завідувача кафедри
прикладної математики та кібербезпеки,
доктор філософії з математики
_____А.В. Луценко
«___» _____ 20__р.

РОЗРОБКА СИСТЕМИ ЗАБЕЗПЕЧЕННЯ ЗАХИСТУ ПЕРСОНАЛЬНИХ ДАНИХ В МЕДИЧНІЙ ХМАРНІЙ СИСТЕМІ E-HEALTH

Спеціальність 105 «Прикладна фізика та наноматеріали»

Кваліфікаційна (магістерська) робота

Науковий керівник:
Загоруйко Л.В.,
к.т.н., доцент,
доцент кафедри прикладної
математики та кібербезпеки

_____
(підпис)

Оцінка : ___/ ___ / _____
(бали/за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця 2024

АНОТАЦІЯ

Васильченко Д.Н. Розробка системи забезпечення захисту персональних даних в медичній хмарній системі e-health. Спеціальність 105 «Прикладна фізика та наноматеріали», освітня програма «Технології інтернету речей», Донецький національний університет імені Василя Стуса, Вінниця 2024.

У кваліфікаційній роботі розроблено систему захисту персональних даних в медичній хмарній системі E-health на базі визначеного ефективного методу розподілу секретної інформації за критеріями складності алгоритму, досконалості, ідеальності визначених з відкритих даних та критеріями швидкодії і використання пам’яті визначених експериментами через програмну реалізацію протоколів розподілу секретної інформації, що пов’язано зі стрімким розвитком інформаційних технологій загалом, зміною форматів збереження і зростанням об’єму даних, та з процесами їх перетворень, редагувань тощо у контексті хмарних медичних даних.

Ключові слова: інформаційні технології, інформаційна система, e-Health, m-Health, медичні інформаційні системи, протокол, секрет

ABSTRACT

D. N. Vasylchenko. Development of a personal data protection system in the medical cloud system e-health. Specialty 105 "Applied Physics and Nanomaterials," Educational Program "Internet of Things Technologies," Vasyl' Stus Donetsk National University, Vinnytsia 2024.

In the qualification work, a personal data protection system is developed for the medical cloud system E-health based on an effective method of distributing confidential information. This method is assessed using criteria such as algorithm complexity, perfection, and ideality determined from open data, as well as performance criteria like speed and memory usage identified through experiments involving the software implementation of secret sharing protocols. This development is driven by the rapid evolution of information technologies, changes in data storage

formats, the increasing volume of data, and processes like transformation, editing, and processing, particularly in the context of cloud-based medical data systems. The system ensures compliance with data security standards, enhances user privacy, and mitigates risks associated with unauthorized access or data breaches in modern E-health environments.

Keywords: information technologies, information system, e-Health, m-Health, medical information systems, protocol, secret.

ЗМІСТ

ВСТУП.....5
РОЗДІЛ 1
АНАЛІЗ КОНТЕКСТУ ТА ПРОБЛЕМАТИКИ РОЗРОБКИ МЕТОДІВ ЗАХИСТУ ПЕРСОНАЛЬНИХ ДАНИХ В МЕДИЧНИХ ХМАРНИХ СИСТЕМАХ.....7
1.1 e-Health як частина системи охорони здоров'я України.....7
1.2 Світовий та український контекст кібератак на хмарні середовища медичної інфраструктури.....10
Висновок до розділу 1.....20
РОЗДІЛ 2.....21
ОГЛЯД МЕТОДІВ РОЗПОДІЛУ СЕКРЕТНОЇ ІНФОРМАЦІЇ ДЛЯ ЗАБЕЗПЕЧЕННЯ ЗАХИСТУ ПЕРСОНАЛЬНОЇ ІНФОРМАЦІЇ.....21
2.1 Протокол таємного обміну секретною інформацією за схемою Шаміра..21
2.2 Протокол таємного обміну секретною інформацією за схемою Асмута-Блума.....23
2.3 Протокол таємного обміну секретною інформацією за схемою Карніна-Гріна-Хелмана.....25
Висновок до розділу 2.....26
РОЗДІЛ 3.....27
ВИЗНАЧЕННЯ ЕФЕКТИВНОГО МЕТОДУ ДЛЯ ЗАБЕЗПЕЧЕННЯ ЗАХИСТУ ПЕРСОНАЛЬНИХ ДАНИХ.....27
3.1 Складність математичних алгоритмів і параметри досконалості і ідеальності протоколів таємного обміну секретною інформацією.....27
3.2 Використані програмні засоби, швидкодія та використання пам'яті протоколів.....28
Висновок до розділу 3.....39
ВИСНОВКИ.....40
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....41

ДОДАТКИ.....45
Код програми..... 45

ВСТУП

Актуальність теми дослідження. В умовах швидкого впровадження цифрових технологій у різні сфери діяльності людини, питання безпеки при обробці та зберіганні інформації стає критично важливим. Особливо це стосується сфери охорони здоров'я, де використання медичних інформаційних систем (МІС) як частини електронної системи охорони здоров'я e-Health стає необхідністю. Мета даного дослідження полягає у створенні ефективного методу, що гарантує безпеку обробки, передачі та зберігання особистих даних (ОД) пацієнтів, які використовують МІС для зв’язку з центральною базою даних e-Health, побудованих на засадах розподіленої хмарної системи.

Мета роботи полягає у визначенні ефективного протоколу розподілу секретної інформації для захисту персональних даних в систему e-Health.

Потрібно вирішити наступні завдання:

1. Проаналізувати літературні джерела, які найбільш актуальні при вирішенні задач, поставлених в магістерській роботі.
2. Проаналізувати методи захисту персональних даних в медичних інформаційних системах.
3. Визначити ефективний метод розподілу секретної інформації за критеріями складності алгоритму, досконалості, ідеальності та швидкодії.

Об’єктом дослідження медичні інформаційні системи (МІС), що використовуються для зберігання, обробки та передачі особистих даних пацієнтів в e-Health.

Предметом дослідження є методи та засоби забезпечення безпеки при передачі, обробці та зберіганні особистих даних пацієнтів у медичних інформаційних системах, побудованих за принципом розподіленої хмарної архітектури.

Теоретичне та практичне значення одержаних результатів полягає у визначенні ефективного методу забезпечення безпеки особистих даних пацієнтів у медичних інформаційних системах e-Health з використанням

розподіленої хмарної архітектури для розробки відповідної системи. Теоретично, воно вносить новаторські підходи до інформаційної безпеки, а практично може підвищити рівень захисту особистих даних, забезпечуючи безпеку пацієнтів та сприяючи розвитку нормативно-правової бази для медичних інформаційних систем e-Health.

Апробація результатів дослідження. Місце системи E-Health у системі охорони здоров'я України було представлено у віснику студентського наукового товариства Донецького Національного Університету імені Василя Стуса, 2023 [1].

Аналіз методів забезпечення захисту персональних даних в медичній хмарній системі E-Health було представлено у збірнику III Міжнародної науково-практичній конференції "Прикладні аспекти сучасних міждисциплінарних досліджень" [40].

Структура та обсяг кваліфікаційної (магістерської) роботи**.** Робота складається зі вступу, 3-х розділів, висновку, списку літературних джерел кількістю 40 найменувань. Основний зміст роботи становить 60 сторінки та містить 2 рисунки.

РОЗДІЛ 1
АНАЛІЗ КОНТЕКСТУ ТА ПРОБЛЕМАТИКИ РОЗРОБКИ МЕТОДІВ ЗАХИСТУ ПЕРСОНАЛЬНИХ ДАНИХ В МЕДИЧНИХ ХМАРНИХ СИСТЕМАХ

1.1 e-Health як частина системи охорони здоров’я України

Електронна система охорони здоров'я України (e-Health) — українська багатоскладова і багатосервісна інформаційно-телекомунікаційна система для забезпечення автоматизованого ведення обліку послуг медичного спрямування та систематизації керування медичною інформацією через електронний вигляд, аби оптимізувати паперову та іншу роботу.

Головними складовими частинами e-Health є:

1. Центральна база даних (ЦБД) — електронна інформаційна система для зберігання реєстрів, що передбачені українським законодавством, також програмні модулі, компоненту інформаційної системи Національної служби здоров’я України (НСЗУ), яка призначена для реалізації державних фінансових гарантій тощо.

2. Відкритий програмний інтерфейс (API) — забезпечує можливість встановлювати зв’язок між реєстрами, державними електронними інформаційними ресурсами, медичними інформаційними системами (МІС) та центральною базою даних e-Health для створення, перегляду, редагування, видалення та обміну медичною інформацією тієї чи іншої форми (документи про історію хвороби, результати аналізів, флюорографій тощо).

3. Медичні інформаційні системи (МІС) — електронна інформаційна система, що створена з метою взаємодіяти та автоматизовувати роботу з масивами даних медичних установ та зручно поєднувати їх з даними центральної бази даних e-Health. Реалізації МІС можуть бути різними, використання відповідної системи залежить від потреб власників медичних установ, їх персоналу, клієнтів [2].

Головні керівні ролі в електронній системі охорони здоров’я України e-Health обіймають:

1. Міністерство охорони здоров’я України (МОЗ) — відповідає за формування політики сфери національної охорони здоров’я, щодо e-Health МОЗ займається прописуванням і затвердженням правил її функціонування та роботи у контексті нормативно-правових актів — постанов, наказів, законів, розпоряджень законодавства України.

2. Державне підприємство «Електронне здоров'я» — є адміністратором центральної бази даних e-Health, який глобально займається її технічною підтримкою, оновленням функціоналу, налаштуванням взаємодії між МІС та ЦБД медичних установ [2,3].

3. Власники медичних установ — використовуючи МІС, вони забезпечують своїх клієнтів і персонал електронними кабінетами, взаємодіючи з центральною базою даних e-Health через зручний інтерфейс користувача.

4. Національна служба здоров’я України — основними завданнями відносно e-Health є ведення реєстрів, окреслення напрямків розвитку, захист персональних даних користувачів [2].

Електронна система охорони здоров'я e-Health використовує передові технології та комунікаційні засоби для підтримки ефективної роботи сфери охорони здоров'я громадян України. Прогнозується, що значні трансформації відбудуться в галузі віртуальних медичних консультацій, мобільних додатків для моніторингу на смартфонах та інших переносних пристроях. Кількість громадян України, які користуються цифровими інструментами для управління своєю медичною документацією, постійно зростає. Однак інтеграція різноманітних джерел даних в електронні медичні записи залишається складною задачею. Система компенсації відстає від темпів інновацій, що ускладнює адаптацію до криз у сфері охорони здоров'я, таких як Covid-19, та підтримку віддалених форм надання медичних послуг. Покращення досвіду отримання медичної допомоги та забезпечення високоякісних результатів у

галузі охорони здоров'я стає важливим завданням для досягнення довгострокового успіху.

За визначенням Всесвітньої організації охорони здоров'я (ВОЗ), концепція e-Health характеризується як "використання інформаційних та комунікаційних технологій для підтримки сфери охорони здоров'я та пов'язаних з нею сфер". У минулому це може бути обмежувалося використанням комп'ютерів для перегляду тестових результатів чи електронних медичних записів. Тепер цей підхід став значно більш широким, охоплюючи різноманітні застосування від мобільного здоров'я (m-Здоров'я) до телемедицини (див. Таблицю 1) [4].

Таблиця 1.1 — Термінології, які стосуються електронної охорони здоров'я відповідно до рекомендацій Всесвітньої організації охорони здоров'я [4]

Термін	Визначення
E-Health	Використання інформаційних та засобів комунікації для підтримки сфери охорони здоров'я та пов'язаних з нею сфер.
M-Health	Використання бездротових мобільних технологій у сфері охорони здоров'я.
Цифрове здоров’я	Широкий загальний термін, який охоплює e-Health (включаючи m-Health), а також нові напрямки, такі як використання передових наук у галузі обробки великих обсягів даних, геноміки та штучного інтелекту.

Продовження таблиці 1.1

Телемедицина	Використання телекомунікаційних і віртуальних технологій для консиліумів та консультацій із закордонними медичними установами

1.2 Світовий та український контекст кібератак на хмарні середовища медичної інфраструктури

Щодня наш світ переживає великий наплив інновацій у сфері комп'ютерних технологій, що дозволяє людям досягати найрізноманітніших цілей та розв'язувати складні завдання. Із появою вільного доступу до Інтернету споживачі активно користуються соціальними мережами, шукають інформацію, витрачають час у різних освітніх та розважальних сервісах, займаються програмуванням. Пандемія COVID-19 ще більше прискорила ці процеси, вимагаючи адаптації бізнесу та інших сфер діяльності під нові реалії роботи та надання послуг (рис. 1.1) [5].

Рис. 1.1. Динаміка електронної торгівлі до та після COVID-19 [5]

Це також впливає на галузь охорони здоров'я, де розробка медичних інформаційних систем на основі розподілених хмарних сервісів стає

тенденцією. Важливість оперативного та зручного використання інформації у медичних організаціях визнається як ключовий аспект забезпечення ефективності надання медичної допомоги.

Потреба в розв'язанні завдань, пов'язаних із зберіганням, структуруванням та обробкою обширних об'ємів інформації, визначає актуальність створення та впровадження в медичні установи нових технологічних рішень, таких як медичні інформаційні системи (МІС), які в Україні є одними з двох головних компонент e-Health, інший це центральна база даних, з якою всі МІС взаємодіють. Перехід до електронного формату даних дозволяє лікарям отримувати оперативний доступ до необхідної інформації про пацієнтів, що сприяє швидшому прийняттю рішень та прискорює процес встановлення діагнозу та обрання методів лікування. Згідно з чинним законодавством, медичні організації виступають як оператори особистих даних своїх пацієнтів, взявши активну участь у всіх етапах обробки цієї інформації. Існує необхідність в ефективному методі забезпечення безпеки даних пацієнтів в медичних інформаційних системах.

При розробці медичних інформаційних систем (МІС) як частини e-Health в Україні головною проблемою є необхідність впровадження надійного захисту конфіденційної інформації, оскільки завдання забезпечення безпеки є важливим аспектом сучасних медичних систем через вразливість таких даних. У період популяризації інформаційних систем, витоки даних через хакерські атаки стали однією з основних проблем щодо персональних даних користувачів таких систем.

Наприклад, у середині грудня 2023-го року в Україні зафіксовано численні кібератаки, зокрема на найбільшого мобільного оператора "Київстар", що призвело до значних порушень у роботі регіональної системи попередження про повітряні нальоти та деяких банківських послуг , створило труднощі для цивільних та військових [6].

Закон України "Про захист персональних даних" від 1 червня 2010 року № 2297-VI встановлює правила обробки персональних даних, включаючи роботу в межах МІС [7].

Через збільшення використання цифрових технологій, необхідність надання критичних послуг та зростання обсягів особливо чутливих даних медичних установ, що викликані як і науково-технічним прогресом загалом, так і досить конкретними кризовими подіями, такі як COVID-19, збільшується і кількість кібератак на таку критичну інфраструктуру, наприклад:

1. Атака за допомогою програмного забезпечення для вимагання на Universal Health Services (UHS) — Universal Health Services, одна з найбільших медичних організацій у США, стала жертвою атаки програмним забезпеченням для вимагання у вересні 2020 року. Атака призвела до перебоїв у роботі сотень її медичних закладів, включаючи лікарні та клініки. Також змусила організацію перейти на паперові записи, призвела до затримок у наданні медичних послуг, а відновлення систем зайняло кілька тижнів. Нападники, ймовірно, використовували програму Ryuk, відому своєю корисністю для фінансових махінацій [8,30].

2. Атака за допомогою програмного забезпечення для вимагання на Health Service Executive Ірландії (HSE) — у травні 2021 року Health Service Executive (HSE) Ірландії зазнала великої атаки за допомогою програмного забезпечення для вимагання. Атака призвела до відключення багатьох IT-систем HSE, що серйозно вплинуло на медичні послуги країни. Тисячі медичних прийомів були скасовані, а доступ до медичних записів став неможливим. Атака була пов'язана з бандою програм-вимагачів Conti, яка зажадала великий викуп. HSE відмовилась від виплати, і повне відновлення системи зайняло кілька тижнів [9].

3. Кібератака на медичну систему Флориди — у червні 2021 року медична організація, що надає послуги у кількох лікарнях Флориди, стала жертвою кібернападу. Це призвело до компрометації чутливих персональних медичних даних. Зловмисники отримали доступ до персональних медичних

записів та іншої конфіденційної інформації, ймовірно, вплинувши на сотні тисяч пацієнтів. Цей інцидент сприяв занепокоєнням щодо безпеки медичних даних [10].

4. Витік даних T-Mobile — атака не була безпосередньо на медичну установу, у серпні 2021 року компанія T-Mobile, що надає послуги кільком медичним організаціям, зазнала величезного витоку даних. Цей витік відкрив доступ до персональних даних мільйонів користувачів, включаючи медичні відомості. Витік даних включав чутливу персональну інформацію, таку як повні імена, адреси, дати народження та медичні дані для певного ряду деяких осіб, що створило загрозу шахрайства та порушення конфіденційності у медичних закладах [11].

5. Атака за допомогою програмного забезпечення для вимагання на Scripps Health — у травні 2021 року Scripps Health, велика медична організація в Південній Каліфорнії, зазнала атаки за допомогою програмного забезпечення для вимагання. Атака зупинила багато її IT-систем, включаючи електронні медичні записи та системи запису на прийом. Атака змусила Scripps скасувати операції, відкласти прийоми та перейти на паперові записи. Вона також призвела до значних операційних збоїв і викликала занепокоєння щодо безпеки пацієнтів та збереження даних [12].

6. Кібернапад на Міністерство охорони здоров'я та соціальних служб США (HHS) — На початку 2020 року Міністерство охорони здоров'я та соціальних служб США (HHS) стало мішенню для атаки за допомогою розподіленої відмови в обслуговуванні (DDoS), що збіглося з початком пандемії COVID-19. Нападники намагалися перевантажити та відключити системи HHS, коли США справлялися з пандемією. Хоча це не була крадіжка даних, атака спричинила перебої в роботі в критичний момент, коли Міністерство керувало федеральною відповіддю на пандемію. Це, ймовірно, була політично мотивована атака, пов'язана з чужоземними противниками, які намагалися порушити роботу США в умовах пандемії [13].

7. Атака на CNA Financial — хоча CNA Financial не є медичною установою, вона надає медичне страхування. У березні 2021 року компанія стала жертвою атаки за допомогою програмного забезпечення для вимагання, що вплинуло на її операції та послуги для клієнтів. Атака вплинула на кілька медичних організацій, застрахованих у CNA, що призвело до затримок в обробці заявок на страхування та адміністративних збоїв. Ця атака підкреслює вразливість медичних установ через сторонні постачальники послуг [14].

8. Витік даних Neiman Marcus — у листопаді 2022 року компанія Neiman Marcus, яка також має значну лінію медичних та оздоровчих послуг, зазнала витоку даних, в результаті якого були розкриті платіжні дані кількох клієнтів. Хоча це не був прямий напад на медичну організацію, цей витік розкрив фінансову та медичну інформацію осіб, які використовували послуги магазину, що підкреслює вразливість між секторами роздрібної торгівлі та охорони здоров'я [15].

Що ж до української електронної системи охорони здоров’я e-Health, то з початком повномасштабного вторгнення з боку держави-терориста Російської Федерації 24 лютого 2022-го року [16], то вона, як і інші системи критичної інфраструктури під постійним тиском кібератак. Як засвідчила заступник Міністра охорони здоров’я України з питань цифрового розвитку, цифрових трансформацій і цифровізації Марія Карчевич [17], у розмові з LB.ua (українське ЗМІ) [18] розповіла про загрози, які виникають для медичних інформаційних систем.

Зазначила, що зловмисники використовують різні методи, серед яких найпоширенішими є спроби підібрати паролі до акаунтів медпрацівників і користувачів МІС, а також DDoS-атаки і фішингові кампанії. Атаки відбуваються як вдень, так і вночі. Іноді служба безпеки змушена дзвонити, щоб терміново реагувати. Команда вживає всі необхідні заходи для забезпечення стабільності системи, але не може надто детально розповідати про свої дії.

За словами Карчевич, центральна база даних медичної системи має понад 10 рівнів захисту, при цьому дані про пацієнтів і персональні відомості зберігаються окремо, що є важливим елементом безпеки.

Найбільші вразливості, на її думку, можуть бути на локальному рівні. Тому Міністерство охорони здоров'я активно тестує приватні медичні інформаційні системи, які використовують медичні заклади по всій країні, щоб вони могли працювати безпечно у взаємодії з центральною базою.

Карчевич також пояснила, що вже відключено 13 медичних інформаційних систем через порушення вимог безпеки. Медичні заклади, які ними користувалися, переведені на інші платформи. При цьому всі дані пацієнтів залишились у безпеці, адже вони автоматично передаються до центральної бази даних електронної бази даних e-Health [19].

Сфера охорони здоров'я залишається привабливою ціллю для кіберзлочинців з кількох причин:

- Критичні дані: Медичні організації зберігають цінні особисті медичні дані (PHI) та іншу чутливу інформацію, що приваблює злочинців.
- Операційні вразливості: Багато медичних установ використовують застаріле програмне забезпечення та системи, що робить їх уразливими до атак.
- Програмне забезпечення для вимагання та шантаж: Злочинці часто знають, що медичні організації мають велике бажання швидко відновити роботу, тому вони частіше погоджуються на виплату викупу.

Збільшення кібернападів на медичні установи після COVID-19 підкреслює нагальну потребу в поліпшенні заходів безпеки в кіберпросторі та збільшенні інвестицій у захист інфраструктури медичних даних.

Важливо відзначити, що, крім даних, які можна використовувати для шантажу та вимагань, існує ще один вид цінної інформації - медичні знімки, які мають значний потенціал для розробки алгоритмів машинного навчання. Наприклад, набір даних, що включає в себе зображення комп'ютерної томографії, рентгенів та інші форми візуалізації, представляє виклик у плані доступності відкритих ресурсів. Таким чином, наявність об'ємної бази даних із

мільйонами зображень може служити додатковим засобом заробітку для зловмисників, які можуть продавати ці зображення зацікавленим компаніям. Водночас супутня інформація про особисті дані виступатиме підтвердженням власності зловмисника над цією цінною базою даних.

1.3 Проблематика захисту персональних даних в хмарному середовищі

Питання забезпечення безпеки особистих даних в інформаційних системах залишається актуальним сьогодні, особливо при обробці медичної інформації. Відповідно до вимог Українського закону "Про захист персональних даних", забезпечення захисту особистих даних пацієнтів у медичних організаціях стає ключовим завданням в рамках медичних інформаційних систем (МІС). Серед безлічі реалізацій МІС часто можна зустріти системи, які або взагалі не використовують механізми забезпечення безпеки, або застосовують неприйнятні методи, такі як симетричне шифрування даних, основним недоліком якого є проблема розподілу ключів.

Симетричне шифрування — це метод криптографії, при якому для шифрування та дешифрування даних використовується один і той самий секретний ключ. Хоча цей метод є досить швидким і ефективним, він має кілька суттєвих недоліків, які обмежують його застосування в певних випадках.

Одним із основних недоліків симетричного шифрування є проблема безпечного обміну ключами. Оскільки одна і та сама секретна ключова інформація використовується як для шифрування, так і для дешифрування, необхідно забезпечити її безпечну передачу між сторонами, які обмінюються зашифрованими даними.

Якщо ключ буде перехоплений зловмисниками під час передачі, вони зможуть дешифрувати всю інформацію, що була зашифрована цим ключем.

Відсутність належних методів передачі ключів виводить симетричне шифрування на другий план у багатьох застосуваннях, де важливе безпечне зберігання та передача секретної інформації.

Симетричне шифрування не є оптимальним для великих систем з великою кількістю користувачів. Кожен новий користувач вимагає індивідуального секретного ключа, якщо вони хочуть обмінюватися зашифрованими даними з іншими учасниками. Це призводить до проблеми масштабованості.

У випадку, коли користувачів багато, кількість ключів, які потрібно зберігати, обчислюється за формулою:

$$\frac{n(n-1)}{2} = K, \tag{1.1}$$

де:

- n — кількість учасників;
- K — кількість ключів.

Для кожної пари користувачів потрібен окремий ключ. У великих організаціях або системах з великою кількістю користувачів, це може бути дуже складно та неефективно в плані зберігання і управління ключами.

Безпека симетричного шифрування безпосередньо залежить від конфіденційності ключа. Якщо зловмисник отримує доступ до ключа, то він може розшифрувати всі дані, зашифровані за допомогою цього ключа. Це означає, що в разі компрометації ключа вся система шифрування стає вразливою.

Відсутність розподілу ключів і слабкість механізмів їх захисту може призвести до серйозних проблем у безпеці.

Якщо ключ зберігається у відкритому вигляді або навіть у менш захищеному місці, то зловмисники можуть його легко добути через атаки на систему зберігання чи за допомогою соціальної інженерії.

Симетричні алгоритми можуть бути вразливими до атаки повторного використання, коли зловмисник перехоплює зашифровані повідомлення і повторно відправляє їх, щоб здійснити якусь небажану дію. Це особливо актуально в системах, де важливим є не тільки конфіденційність, але й цілісність та автентичність повідомлення. Для запобігання такій атаці часто використовують механізми додаткової аутентифікації, такі як підписування

повідомлень чи використання часових міток (nonce), але це ускладнює сам процес шифрування та дешифрування.

У симетричному шифруванні для кожного користувача та кожної операції, пов'язаної з шифруванням/дешифруванням, потрібен єдиний ключ. Це обмежує можливість гнучкого управління правами доступу.

Відсутність можливості диференційованого доступу до різних частин даних або до різних функцій шифрування може призвести до труднощів в організації політик доступу в складних системах. Наприклад, якщо один ключ доступу є для великої кількості користувачів, то всі ці користувачі матимуть однаковий рівень доступу до даних, навіть якщо це не є бажаним.

При використанні симетричних алгоритмів для забезпечення як конфіденційності, так і цілісності даних одночасно, виникає потреба у додаткових засобах, оскільки стандартне шифрування не гарантує цілісності (не дає можливості перевірити, чи не було змінено зашифроване повідомлення).

Для цього використовуються додаткові механізми, такі як хешування (HMAC) чи цифрові підписи, що призводить до додаткових витрат ресурсів і ускладнення системи.

У разі відсутності цих додаткових механізмів, система може бути вразливою до атак, які модифікують зашифровані дані, не порушуючи саму конфіденційність.

Для досягнення високої безпеки симетричне шифрування потребує використання досить довгих та складних ключів. Однак навіть за умови наявності сильних ключів проблема зберігання та управління ними залишатиметься.

Ключі, що є дуже довгими та складними, можуть бути важкими для зберігання та передачі, а також потребують значних обчислювальних потужностей для обробки.

Крім того, слабо захищена система може дозволити зловмиснику отримати ці ключі шляхом атак, спрямованих на зберігання чи передачу даних.

У сучасних динамічних середовищах, де інформація швидко змінюється або оновлюється (наприклад, у хмарних системах або великих розподілених мережах), симетричне шифрування може бути недостатньо гнучким для швидкої зміни ключів.

У таких випадках знову виникає проблема: якщо ключ застаріває або має бути змінений, це може потребувати великих зусиль з боку адміністраторів системи або навіть призвести до збоїв у роботі.

Отже, симетричне шифрування, хоч і є швидким і ефективним методом захисту даних, має низку суттєвих обмежень. Проблеми з безпекою обміну ключами, масштабованістю, управлінням доступом, а також вразливості до атак, які не враховують зміни даних, роблять його менш привабливим для великих, розподілених або високозахищених систем. Це підвищує важливість використання гібридних систем шифрування, де симетричне шифрування комбінується з асиметричними методами для досягнення більш високого рівня безпеки і зручності в управлінні ключами [20].

Зазначимо також, що деякі компанії для зберігання даних використовують сервери, що розташовані за межами території України (Amazon, Azure і т.д.), що створює прогалину в інформаційній безпеці. Цікаво, що навіть найбільш захищені системи, які використовують асиметричне шифрування, можуть бути вразливими, оскільки дані в таких системах шифруються на локальних серверах, а потрапляють туди через канали зв'язку у незашифрованому вигляді.

Багато організацій затримуються з впровадженням хмарних обчислень через серйозні проблеми з юридичної точки зору, з якими стикаються постачальники хмарних послуг:

1. Забезпечення конфіденційності даних пов'язане з обов'язком захищати інформацію клієнта, що передається третій стороні. Порушення будь-яких угод з третьою стороною може серйозно підірвати конфіденційність даних клієнтів.

2. Щодо цілісності даних, виникають проблеми зі зберіганням вихідних даних, розташованих в різних фізичних місцях на серверах, що

обслуговуються різними організаціями по всьому світу. Без гарантії адекватного захисту даних, що зберігаються в центрах обробки даних, розташованих у різних куточках світу, виникає серйозна загроза для цілісності інформації.

3. Функція взаємодії різних сервісів у хмарі також впливає на конфіденційність даних, і це стає проблемою для адміністрування хмарної інфраструктури.

Проте, як підсумок, щодо області розподілених інформаційних систем, на сьогодні вони перебувають на передньому краї в розв'язанні завдань обстеження громадського здоров'я. Відсутність необхідності в наявності єдиного сервера та централізованого управління даними виявляється ключовим фактором, особливо в медичній галузі. Централізоване управління може накладати обмеження на швидкість обробки даних, тим самим сповільнюючи весь процес. Крім того, воно вважається менш безпечним з точки зору обробки та зберігання особистих даних.

РОЗДІЛ 2
ОГЛЯД МЕТОДІВ РОЗПОДІЛУ СЕКРЕТНОЇ ІНФОРМАЦІЇ ДЛЯ ЗАБЕЗПЕЧЕННЯ ЗАХИСТУ ПЕРСОНАЛЬНОЇ ІНФОРМАЦІЇ

2.1 Протокол таємного обміну секретною інформацією за схемою Шаміра

Для забезпечення безпеки обміну медичними файлами обстеження можна використовувати схему Шаміра. Ця схема дозволяє створити (k, p) порогову систему обміну секретом, де тільки будь-які k чи більше сторін (k ≤ p) можуть відновити секрет. Це означає, що, наприклад, якщо встановлено поріг k=3 та загальна кількість сторін p=5, то будь-які три або більше сторін можуть відновити секрет, але будь-які менше, ніж три, сторони не матимуть достатньої інформації для розкриття секрету. Це дозволяє забезпечити конфіденційність медичної інформації шляхом обміну файлами між довіреними сторонами, забезпечуючи при цьому високий рівень захисту навіть у випадку, якщо деякі ключі потраплять у руки несанкціонованих осіб (рис. 2.1, 2.2).

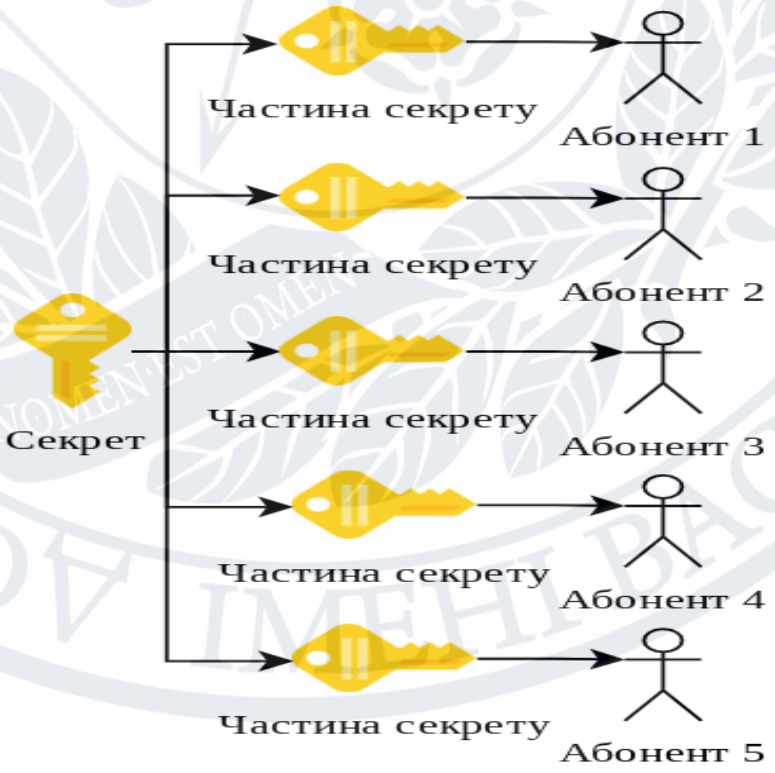

Рис. 2.1. Розділення секрету на частини за схемою Шаміра

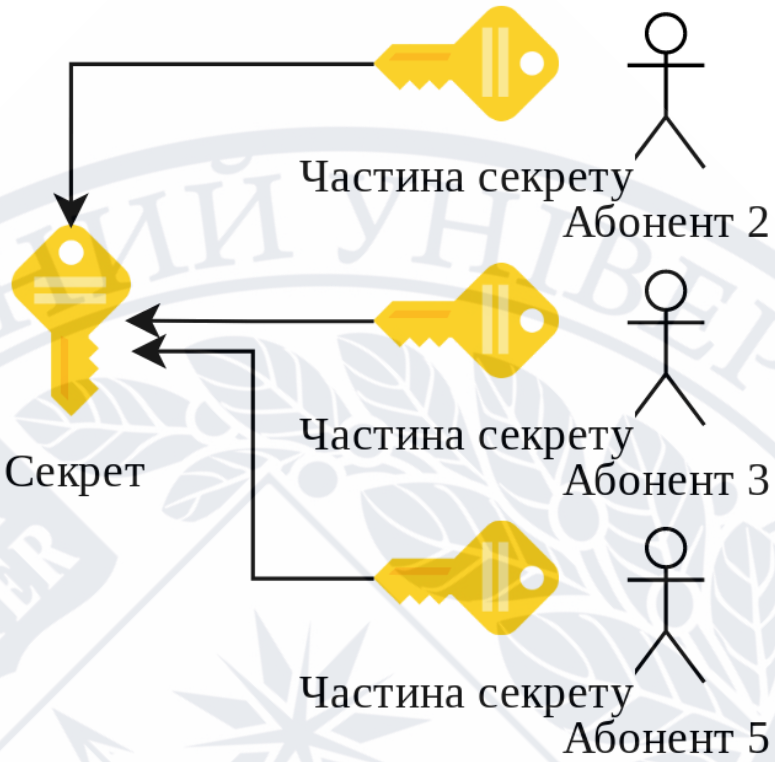

Рис. 2.2. Збирання частин секрету в один фрагмент за схемою Шаміра

У вказаному підході абоненти, тобто користувачі, виступають у ролі серверів, що зберігають фрагменти секрету, які в даному контексті є медичними файлами обстеження. Протокол розподілу секрету Шаміра – це криптографічний метод, призначений для розподілу секретної інформації серед кількох учасників таким чином, що лише певна кількість з них, що володіють інформаційним потоком, відома як поріг, необхідна для відновлення секрету. У центрі протоколу знаходиться інтерполяція полінома над скінченними полями. Секрет представлений як сталий член полінома, при цьому кожен учасник отримує частку, що відповідає точці на кривій полінома. Поліном будується таким чином, що знання будь-якої підмножини менше, ніж поріг, не надає жодної інформації про секрет. Таким чином, поєднуючи частки від достатньої кількості учасників, можна відновити початковий секрет. Ця основна властивість протоколу розподілу секрету Шаміра забезпечує стійкість та безпеку при розподілі конфіденційної інформації серед кількох сторін.

З математичної точки зору, поділ секрету виглядає так:

1. Ініціалізація:
   a. Учасники домовляються про просте число p:

$$p \in P, \tag{2.1}$$

$$p > s, \tag{2.2}$$

де:

- P − множина простих чисел.
- s − секретний ключ (значення).

b. Кожен учасник незалежно вибирає випадкові коефіцієнти $a_1, a_2, ..., a_{t-1}$ з множини цілих чисел за модулем *p:*

$$a_i \in Z_p, \tag{2.3}$$

$$0 \leq i \leq t - 1, \tag{2.4}$$

де:

- Z_p − множина цілих чисел за модулем *p.*
- t − поріг

c. Обчислити поліном:

$$f(x) = s + a_1 x + a_2 x^2 + ... + a_{t-1} x^{t-1} \ mod \ p \tag{2.5}$$

2. Розподіл:

Учасники обирають різні ненульові значення *x* для своїх часток та обчислюють відповідні значення *y (f(x))* за допомогою узгодженого полінома.

3. Відновлення:

a. Учасники збирають принаймні *t* часток від інших.

b. Відновлення секрету *s* за допомогою інтерполяції Лагранжа:

$$s = \sum_{i=1}^{t} (y_i \cdot \prod_{j \neq i, j=1}^{t} \frac{x - x_j}{x_i - x_j}) \tag{2.6}$$

де (x_i, y_j) − це частки, зібрані від учасників [21-22,40].

2.2 Протокол таємного обміну секретною інформацією за схемою Асмута-Блума

Протокол розподілу секрету Асмута-Блума розподіляє секрет серед учасників, представляючи його як пару цілих чисел за модулем відмінних

простих чисел. Використовуючи китайську теорему залишку , учасники обчислюють частки, що конгруентні секрету за модулем різних простих чисел. Ці частки розподіляються серед учасників, і будь-який підмножина з необхідною кількістю часток може відновити секрет, використовуючи теорему. Цей підхід забезпечує надійний розподіл і відновлення секрету серед кількох сторін.

Опис протоколу з математичної точки зору:

1. Ініціалізація:
   a. Учасники кількістю n колективно узгоджують набір різних простих чисел $p_1, p_2, ..., p_n$, кожне з яких більше за секрет *s:*

$$p_1, p_2, ..., p_n, \tag{2.7}$$

$$p_i > s, \tag{2.8}$$

$$p_i \in P, \tag{2.9}$$

де P − множина простих чисел.

   b. Користуючись китайською теоремою залишку, кожен учасник обчислює свої відповідні значення $u_1, u_2, ..., u_n$ такі, що:

$$u_i \equiv s \bmod p_i, \tag{2.10}$$

де u_i − частина секрету.

2. Розподіл:
   a. Учасники розподіляють свої відкриті ключі (p_i, u_i) іншим для забезпечення секретного розподілу.
3. Відновлення:
   a. Збираючи t або більше часток, учасники використовують китайську теорему залишку для відновлення секрету *s*:

$$s = \sum_{i=1}^{n} u_i \cdot \left(\prod_{j \neq i, j=1}^{n} p_j \right) \cdot \left(\prod_{j \neq i, j=1}^{n} (p_j)^{-1} \bmod p_i \right) \bmod P, \tag{2.11}$$

$$P = p_1 \cdot p_2 \cdot \ldots \cdot p_i, \tag{2.12}$$

$$t \leq n, \tag{2.13}$$

де t − поріг кількості часток секрету для відновлення початкового секрету [23-24,40].

2.3 Протокол таємного обміну секретною інформацією за схемою Карніна-Гріна-Хелмана

За допомогою протоколу розподілу секрету Карніна-Гріна-Хелмана, учасники можуть безпечно розділити конфіденційну інформацію шляхом генерації часток, які обмінюються між собою. Кожна частка обчислюється на основі випадкового числа, яке кожен учасник обирає незалежно, та загальних параметрів p та g. Після розподілу часток учасники можуть використовувати зібрані частки для відновлення початкового секретного значення за допомогою визначеного методу відновлення. Цей протокол забезпечує безпечний та ефективний спосіб розподілу та відновлення секретної інформації між довіреними сторонами.

1. Ініціалізація:

Визначається секрет S і налаштовуються криптографічні параметри, такі як просте число і випадкові вектори.

2. Розподіл:

Секрет ділиться на частки шляхом обчислення скалярних добутків між випадковими векторами, і ці значення розподіляються як частки серед учасників.

3. Відновлення секрету:

Використовуючи хоча б k часток, вирішується система лінійних рівнянь для відновлення вектора u, з якого пізніше відновлюється оригінальний секрет [25-26,40].

Отже, було розглянуто найпоширеніші алгоритми розподілу секретної інформації, на яких математичних основах вони побудовані, які можна використовувати у розподілених хмарних системах, якою і є медична система e-Health.

РОЗДІЛ 3
ВИЗНАЧЕННЯ ЕФЕКТИВНОГО МЕТОДУ ДЛЯ ЗАБЕЗПЕЧЕННЯ ЗАХИСТУ ПЕРСОНАЛЬНИХ ДАНИХ

3.1 Складність математичних алгоритмів і параметри досконалості і ідеальності протоколів таємного обміну секретною інформацією

Перед тим як розпочати тести, було сформульовано мету – визначити найбільш ефективний протокол для поділу секретної інформації серед методів, що мають відповідну функціональність. Для цього було проведено порівняльний аналіз за кількома ключовими параметрами.

Щоб визначити ефективність запропонованого методу захисту, було проаналізовано складність математичних алгоритмів, ресурсомісткість обчислень, а також ступінь досконалості й ідеальності для трьох схем, що підходять для розв'язання поставленого завдання. Вибір цих схем ґрунтувався на їхній відповідності до мети задачі, простоті реалізації та доступності у відкритому доступі бібліотек, які можна використовувати без додаткових налаштувань. Таблиця 2.1 містить порівняльний аналіз складності алгоритмів для етапів поділу секретної інформації та її відновлення.

Таблиця 2.1 − Аналіз складності математичних алгоритмів протоколів таємного обміну секретною інформацією

	Шаміра	Асмута-Блума	Карін-Грін-Хелмана
Розділення секрету	$O(N*K)$	$O(N)$	$O(K*N)$
Відновлення секрету	$O(K^2)$	$O(K^2)$	$O(K^3)$

Таблиця 2.2 містить результати аналізу схем за параметрами досконалості та ідеальності.

Таблиця 2.2 − Аналіз параметрів досконалості та ідеальності протоколів таємного обміну секретною інформацією

	Шаміра	Асмута-Блума	Карнін-Грін-Хелмана
Досконалість	+	+	+
Ідеальність	+	-	-

Під досконалістю мається на увазі, що навіть за наявності необмеженої кількості несанкціонованих користувачів інформація про секрет не може бути отримана.

Ідеальність означає, що якщо частка секрету дорівнює повному обсягу секрету, то схема є ідеальною.

Аналіз показав, що єдина схема, яка відповідає обом критеріям, це схема Шаміра.

3.2 Використані програмні засоби, швидкодія та використання пам’яті протоколів

Для програмної реалізації було обрано об’єктно-орієнтовану мову програмування Java версії 23, через ряд переваг цієї мови програмування:

1. Незалежність від платформи — завдяки можливості "написати один раз, а запускати де завгодно" на основі Java Virtual Machine (JVM), Java є дуже портативною. Реалізації протоколів розподілу секретів на Java можуть запускатися стабільно на різних операційних системах, що є корисним для протоколів, які можуть бути розгорнуті на різних типах пристроїв.

2. Вбудовані криптографічні бібліотеки — Java має надійні бібліотеки для криптографії (наприклад, Java Cryptography Architecture, Bouncy Castle), які підтримують генерацію випадкових чисел, хешування та модульну арифметику. Ці бібліотеки дозволяють ефективно та безпечно реалізувати необхідні криптографічні операції для розподілу секретів.

3. Автоматизоване управління пам’яттю та функції безпеки — Автоматичне прибирання пам'яті в Java допомагає запобігати витокам пам'яті, що є критичним під час роботи з конфіденційними даними, як-от

криптографічні ключі та частки секрету. Крім того, Java має вбудовані функції безпеки, такі як перевірка меж для масивів, що допомагає запобігати певним уразливостям та помилкам під час обробки великих чисел або керування частками.

4. Багатопоточність та підтримка конкурентності — Протоколи розподілу секретів, особливо у великих масштабах, можуть виграти від багатопоточності для розподілу навантаження, обробки багатьох користувачів або розподілу/обміну ключами між різними клієнтами чи вузлами. Java забезпечує потужні функції для багатопоточності, що може бути корисним для реалізації багатосторонніх обчислень та ефективного розподілу даних.

5. Повна підтримка обробки даних та модульної арифметики — — Протоколи розподілу секретів часто потребують складних математичних операцій, таких як модульне піднесення до степеня та поліноміальна арифметика (як у схемах Шаміра та Асмута-Блума). Підтримка роботи з великими цілими числами (через BigInteger) у Java дозволяє точно та ефективно реалізовувати модульну арифметику без турботи про переповнення чисел.

6. Об'єктно-орієнтоване програмування (ООП) — — об'єктно-орієнтований підхід Java дозволяє легко інкапсулювати та створювати модульний код, що робить зручним розробку багаторазових компонентів для кожного протоколу. Це корисно під час реалізації, підтримки та розширення складних алгоритмів розподілу секретів, оскільки протоколи можна розділити на зрозумілі, керовані класи та методи.

7. Широка спільнота та документація — Java має велику спільноту розробників, безліч бібліотек та детальну документацію. Ця екосистема підтримує швидку розробку, виправлення помилок та оптимізацію під час реалізації складних алгоритмів, таких як схеми розподілу секретів [27].

Як інтегроване середовище розробки було обрано IntelliJ IDEA з кількох причин:

1. Розширена допомога при написанні коду: IntelliJ IDEA забезпечує інтелектуальне доповнення коду, глибокий статичний аналіз і поради в

реальному часі. Це особливо корисно при реалізації криптографічних алгоритмів, де точність і акуратність мають велике значення. IDE допомагає виявити потенційні помилки на ранніх етапах, підкреслюючи їх, пропонуючи варіанти рефакторингу та миттєві виправлення.

2. Потужні інструменти для налагодження та тестування: налагодження є критично важливим при розробці криптографічних протоколів, таких як схеми розподілу секретів, адже навіть найменша помилка може призвести до серйозних уразливостей. Потужний відлагоджувач IntelliJ IDEA дозволяє легко крокувати по коду, перевіряти змінні та оцінювати вирази в реальному часі. Крім того, підтримуються фреймворки для юніт-тестування, як-от JUnit, що корисно для написання автоматизованих тестів для перевірки коректності реалізації протоколу.

3. Інтеграція з інструментами для складання: IntelliJ IDEA бездоганно інтегрується з популярними інструментами для складання, такими як Maven та Gradle. Ці інструменти допомагають керувати залежностями та обробляти процес компіляції, що спрощує організацію великих проєктів із багатьма залежностями. Це особливо корисно для криптографічних проєктів, де можуть бути потрібні зовнішні бібліотеки (як-от Bouncy Castle для криптографії).

4. Інтеграція з системами контролю версій: IntelliJ IDEA інтегрується з Git, Mercurial та іншими системами контролю версій. Це дозволяє легко керувати версіями коду, співпрацювати та відслідковувати зміни протягом часу. Робота з системами контролю версій є важливою при розробці складних алгоритмів, адже інколи необхідно повернути або порівняти зміни, особливо при роботі з критичними компонентами безпеки.

5. Підтримка всіх основних платформ: IntelliJ IDEA працює на різних платформах (Windows, macOS, Linux), що корисно, якщо ви розробляєте систему, яка має працювати на різних операційних системах. Ви можете писати та тестувати код на будь-якій платформі, знаючи, що він буде працювати однаково на всіх середовищах.

6. Багата екосистеми плагінів: IntelliJ IDEA має велику кількість плагінів, що розширюють її функціональність. Наприклад, можна додати підтримку різних мов програмування або інструментів, які можуть знадобитися в майбутньому, або навіть покращити IDE за допомогою специфічних бібліотек чи фреймворків, що стосуються криптографії та безпеки протоколів.

7. Орієнтація на безпеку: IntelliJ IDEA включає функції, орієнтовані на безпечне програмування. Наприклад, вона може виявляти потенційні уразливості безпеки в коді, такі як жорстко закодовані паролі або неправильне використання криптографічних функцій. Це особливо корисно при роботі з криптографічними протоколами, де безпека є головним пріоритетом.

8. Рефакторинг і оптимізація коду: IntelliJ IDEA спрощує рефакторинг коду за допомогою своїх розширених інструментів для рефакторингу. Оскільки протоколи розподілу секретів часто включають складні алгоритми та кодові бази, можливість змінювати та покращувати код без введення помилок може зекономити час і запобігти виникненню багів.

9. Інструменти для колаборації: Якщо ви працюєте в команді, IntelliJ IDEA інтегрується з інструментами для співпраці, такими як GitHub та GitLab, що дозволяє легко працювати з колегами, ділитися кодом і керувати запитами на злиття. Це корисно при командній розробці криптографічних проєктів, де багато людей можуть працювати над різними частинами реалізації протоколу [28].

Збирачем проекту виступає інструмент Maven, через свої якості:

1. Управління залежностями: однією з найбільших переваг Maven є його здатність керувати залежностями проєкту. Криптографічні протоколи часто вимагають зовнішніх бібліотек (наприклад, Bouncy Castle для криптографії). З Maven ви можете легко оголосити ці залежності в файлі pom.xml, і Maven автоматично завантажить та включить правильні версії. Це економить час і гарантує, що правильні бібліотеки завжди будуть включені у ваш проєкт.

2. Послідовність у різних середовищах: Maven гарантує, що ваш проєкт буде будуватися послідовно в різних середовищах. При розробці криптографічних протоколів важливо, щоб процес складання був

відтворюваним на будь-якій машині розробника або в будь-якому середовищі розгортання. Стандартизований процес складання Maven гарантує цю послідовність.

3. Автоматизація складання: Maven автоматизує процес складання, дозволяючи вам компілювати код, виконувати тести, створювати JAR-файли (формат файлу, який використовується для зберігання та розповсюдження програм на Java, що містить класи, бібліотеки, ресурси та метадані в одному архіві [30]) та пакувати застосунок одною командою. Це спрощує управління життєвим циклом проєкту, що є важливим при роботі з такими складними протоколами, як розподіл секретів, де необхідно забезпечити правильне складання та тестування всіх компонентів щоразу, коли ви вносите зміни.

4. Структура проєкту та принцип «конвенція замість конфігурації»: Maven забезпечує стандартизовану структуру проєкту, що допомагає організувати код, ресурси та файли конфігурацій. Це корисно при роботі з великими проєктами, такими як протоколи розподілу секретів, оскільки це спрощує навігацію по проєкту. Наприклад, Maven очікує певну структуру папок (наприклад, src/main/java для вихідного коду, src/test/java для тестів), що зменшує потребу в конфігурації та підвищує продуктивність.

5. Інтеграція з IDE: Maven безперешкодно інтегрується з популярними IDE, такими як IntelliJ IDEA, Eclipse [31] та NetBeans [32]. Ця інтеграція забезпечує автоматичну синхронізацію між IDE та Maven, що полегшує імпорт проєктів, управління залежностями та виконання складань безпосередньо з середовища розробки. Для розробників, які працюють над криптографічними протоколами, це значно спрощує робочий процес.

6. Тестування та безперервна інтеграція: Maven добре працює з фреймворками для тестування, такими як JUnit, що дозволяє легко визначати та виконувати юніт-тести. У криптографії тестування є критично важливим для забезпечення коректності та безпеки протоколу. Завдяки Maven ви можете виконувати автоматизовані тести як частину процесу складання. Крім того, Maven інтегрується з інструментами для безперервної інтеграції (CI), такими як

Jenkins, що дозволяє автоматизувати тестування та складання в CI-конвеєрі, що є важливим для підтримки якості складних криптографічних протоколів з часом.

7. Легкість у розповсюдженні та упаковці: Maven легко пакує ваш проєкт у JAR, WAR (це формат файлу, що використовується для зберігання веб-додатків на Java, включаючи сервіти, JSP-файли, статичні ресурси та конфігураційні файли, упаковані в один архів для розгортання на веб-сервері [33]) або інші формати, що дозволяє розповсюджувати реалізацію вашого протоколу розподілу секретів. За допомогою правильних налаштувань Maven також може генерувати документацію та звіти, що надають корисну інформацію про структуру проєкту та його залежності.

8. Масштабованість та гнучкість: конфігурація Maven є дуже гнучкою, що дозволяє налаштувати процес складання відповідно до ваших потреб. Наприклад, ви можете визначити власні плагіни або налаштувати додаткові завдання для спеціалізованих криптографічних потреб, таких як обробка генерації ключів або інтеграція з зовнішніми сервісами. Модульний дизайн Maven також робить його легко розширюваним.

9. Централізований репозиторій для бібліотек: Maven використовує централізований репозиторій бібліотек, що означає, що вам не потрібно вручну керувати JAR-файлами чи вирішувати конфлікти версій. З Maven ви просто вказуєте залежності, і Maven завантажить необхідні файли з центрального репозиторію, що забезпечує використання добре підтримуваних та актуальних бібліотек.

10. Документація та підтримка спільноти: Maven має розширену документацію та велику спільноту користувачів. Це спрощує пошук рішень для типових проблем, налаштування вашого проєкту або розуміння, як інтегрувати Maven у ваш процес складання. Коли ви працюєте над криптографічним проєктом, де проблеми можуть бути дуже технічними, наявність надійної документації та підтримки спільноти може зекономити багато часу та зусил [29].

Як робоча станція використовувався персональний комп'ютер на ОС Linux Debian 12 [34] з характеристиками: Intel Core i5-11800H, ОЗП 16 ГБ, SSD 500 Гб.

В рамках дослідження проведено 8 серій експериментів, що дозволяють оцінити залежність часу поділу та відновлення секрету від кількості серверів у пороговій схемі поділу, порогової кількості учасників, а також об’єм використовуваної пам'яті при використанні різних схем поділу секрету на основі аналізу параметрів (див. у Додатках).

Для аналізу швидкодії було використано внутрішній інструмент мови програмування Java − System.nanoTime(), оскільки відповідальний за забезпечення високоточного вимірювання часу в наносекундах, що ідеально підходить для точної оцінки тривалості коротких операцій, таких як поділ секрету чи його відновлення. Також він надає стабільні, точні та незалежні від годинника значення, він є кращим вибором для бенчмаркінгу алгоритмів і вимірювання тривалості виконання в продуктивно-критичних застосунках.

Як саме використовується інструмент System.nanoTIme():

1. Фіксація початкового часу:

Перед початком роботи алгоритму (наприклад, поділу секрету чи його відновлення) поточний час записується за допомогою System.nanoTime() і зберігається у змінній startTime.

2. Фіксація кінцевого часу:

Після завершення операції час знову записується за допомогою System.nanoTime() і зберігається у змінній endTime.

3. Обчислення тривалості:

- Вираховується різниця між кінцевим часом і початковим часом дає тривалість виконання операції в наносекундах.
- Для зручності значення переводиться в мілісекунди шляхом поділу на 1_000_000.

4. Виведення результату:

Обчислена тривалість виконання виводиться у консоль, щоб надати інформацію про продуктивність алгоритму [35].

Для аналізу використання пам’яті було використано інструмент Runtime, оскільки надає можливість моніторити та взаємодіяти з пам'яттю Java Virtual Machine (JVM) [36]. Він дозволяє отримати доступ до таких метрик пам'яті, як загальна пам'ять, вільна пам'ять та використана пам'ять, що робить його ідеальним інструментом для оцінки споживання пам'яті під час виконання програми. Вимірюючи різницю між використаною пам'яттю до та після виконання конкретної операції, можна оцінити, скільки пам'яті було використано цією операцією. Це особливо корисно для розуміння ефективності алгоритмів щодо пам'яті. Використання Runtime не потребує сторонніх інструментів для профілювання або бібліотек. Це легкий і вбудований інструмент у Java, що забезпечує сумісність та простоту використання.

Як саме використовується інструмент Runtime [37]:

1. Створюється об’єкту класу Runtime для взаємодії із віртуальною машиною мови програмування Java і отримується доступ до неї через метод getRuntime():

2. Примусовий виклик збирача сміття (Garbage Collector [38]):

Перед вимірюванням використаної пам'яті викликається runtime.gc(), щоб примусово запустити збирач сміття. Це дозволяє мінімізувати вплив залишкових об'єктів у пам'яті та надає точну базу для вимірювань.

3. Визначення кількості пам'яті, що використовується до запуску алгоритму:

Здійснюється шляхом віднімання вільної пам'яті (runtime.freeMemory()) від загальної пам'яті, виділеної для JVM (runtime.totalMemory()).

4. Відпрацювання методу розподілу секретної інформації.

5. Визначення кількості пам'яті, що використовується у результаті роботи алгоритму тим же ж способом.

6. Обчислення спожитої пам’яті:

Вираховується різниця між використаною пам'яттю після та до операції, що надає значення спожитої пам’яті.

7. Переведення у зручну одиницю вимірювання:

Обчислена пам’ять, що була використана методом, конвертується з байтового значення у кілобайтове для зручності.

Нехай N − кількість серверів у пороговій схемі поділу. Параметр K використовується для операції відновлення секрету і утворює необхідну порогову кількість учасників. У якості файлу (M) був використаний DICOM (ангіографія). Розмір склав 1333 кбайт. У таблиці 2.3 наведені результати, виходячи з яких видно переваги схеми Шаміра для різних значень K та N.

Таблиця 2.3 − Дослідження основних параметрів протоколів розділення секретної інформації DICOM файлу 1333 кілобайт

	Параметри												
Схеми	Швидкість дії над секретом (мс)	N=5	K=4	M= 1333	N=10	K=8	M= 1333	N=25	K=20	M= 1333	N=50	K=40	M= 1333
Шаміра	Розділення	3,95			4,02			5,04			11,5		
	Відновлення	0,80			0,83			1,6			3,8		
Асмута-Блу -ма	Розділення	72,02			115,2			696,5			802		
	Відновлення	1			1,6			3,03			16		
Карнін-Грін -Хелмана	Розділення	12,6			152,1			723			3702		
	Відновлення	1,5			80			417,7			2203		

Далі було проведено те саме дослідження, але з файлом більшого розміру, а саме 12516 кілобайт, аби перевірити чи краща швидкодія протоколу Шаміра відносно інших протоколів збережеться, у таблиці 2.4 наведені результати.

Таблиця 2.4 − Дослідження основних параметрів протоколів розділення секретної інформації txt файлу 12516 кілобайт

	Параметри												
Схеми	Швидкість дії над секретом (мс)	N=5	K=4	M=12516	N=10	K=8	M=12516	N=25	K=20	M=12516	N=50	K=40	M=12516
Шаміра	Розділення	8,3			8,5			10			23		
	Відновлення	2,2			2,3			4			8,5		
Асмута-Блу-ма	Розділення	160,7			115,2			696,5			1822		
	Відновлення	2,4			3,33			5,7			32		
Карнін-Грін-Хелмана	Розділення	27,4			271			1401			7001		
	Відновлення	2,1			142			809,2			4070		

Із результатів видно, що схема Шаміра є найкращою у питанні швидкодії.

Що стосується середніх значень використання пам’яті комп’ютера, було проведено дослідження результати якого показано у таблиці 2.5.

Таблиця 2.5 − Дослідження використання пам’яті у процесі обчислень

	Шаміра	Асмута-Блума	Карнін-Грін-Хелмана
Розділення секрету	*27 Кб*	*63 Кб*	*87 Кб*
Відновлення секрету	*120 Кб*	*333 Кб*	*89 Кб*

Таким чином, на підставі проведених експериментів було обрано схему поділу секрету Шаміра, оскільки виявилася найоптимальнішою за критеріями:

- швидкодія;
- використання пам’яті;
- властивість досконалості;
- властивість ідеальності.

ВИСНОВКИ

В магістерській роботі було досліджено сучасні методи поділу секрету, в основі яких лежать реалізації Шаміра, Асмута-Блума та Карнін-Грін-Хелмана.

В результаті роботи запропоновано метод забезпечення безпеки персональних даних на основі схеми розподілу секретної інформації Шаміра, що ляже в основу системи захисту у хмарній медичній інформаційній системі.

Для підтвердження правильності вибору схеми поділу секрету було проведено експерименти, під час яких було визначено оптимальну схему Шаміра. За результатами серій проведених досліджень через програмну реалізацію методів розподілу секретної інформації, схема Шаміра довела ефективність за критеріями швидкодії та використання пам’яті, у тому числі під час роботи з файлами великих розмірів та різних форматів, що підтверджується швидкою обробкою даних відносно інших протоколів, чим обумовлений вибір даної схеми для реалізації запропонованого у цій роботі методу захисту. Крім того, підтверджено, що схема Шаміра має властивості досконалості та ідеальності, що є важливим аспектом у контексті питань інформаційної безпеки. Застосування СРС з такими властивостями дозволяє забезпечувати безпеку даних на вищому рівні, оскільки є можливість створювати фрагменти секрету такого самого розміру, як і вихідний файл.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Васильченко Д. Н., Загоруйко Л. В. Місце системи E-Health у системі охорони здоров'я України. [Електронний ресурс] / Вісник студентського наукового товариства Донецького національного університету імені Василя Стуса. Випуск 15. Том 2 / ред. кол. М. І. Прихненко (голова) та ін. Вінниця: ДонНУ імені Василя Стуса, 2023. Вип. 15. Т. 2. 234 с. - Режим доступу: https://jvestnik-sss.donnu.edu.ua/article/view/14707

2. Електронна система охорони здоров'я в Україні [Електронний ресурс] — Режим доступу : https://ehealth.gov.ua/

3. Підключені до ЦБД Медичні Інформаційні Системи [Електронний ресурс] — Режим доступу : https://ehealth.gov.ua/pidklyucheni-do-ehealth-mis/

4. WHO guideline: recommendations on digital interventions for health system strengthening. Geneva: World Health Organization; 2019.

5. Дашко І. М., Михайліченко Л. В. Цифровізація економіки в умовах пандемії CovId-19 як стратегічна платформа розвитку економіки держави: ЗНУ, 2023. С. 2-7.

6. Major cyberattack on Ukrainian mobile operator disrupts banking services and air raid sirens [Електронний ресурс] — Режим доступу : https://edition.cnn.com/2023/12/12/europe/ukraine-cyber-attack/index.html

7. Global Data Privacy & Security Handbook [Електронний ресурс] — Режим доступу : https://resourcehub.bakermckenzie.com/en/resources/data-privacy-security/emea/ukraine/topics/key-data-privacy-and-security-laws

8. UHS Ransomware Attack Cost $67M in Lost Revenue, Recovery Effort [Електронний ресурс] — Режим доступу : https://www.techtarget.com/healthtechsecurity/news/366595382/UHS-Ransomware-Attack-Cost-67M-in-Lost-Revenue-Recovery-Efforts

9. Lessons Learned from the HSE Cyber Attack [Електронний ресурс] — Режим доступу:

https://www.hhs.gov/sites/default/files/lessons-learned-hse-attack.pdf

10. Cyberattacks At UF Hospitals In Leesburg, The Villages Under Investigation [Електронний ресурс] — Режим доступу: https://health.wusf.usf.edu/health-news-florida/2021-06-05/experts-look-into-possible-cyberattacks-at-leesburg-villages-hospitals

11. Deadline Passes on T-Mobile's $350 Million Settlement Days After Another Data Breach [Електронний ресурс] — Режим доступу: https://www.cnet.com/personal-finance/deadline-passes-on-t-mobiles-350-million-settlement-days-after-another-data-breach/

12. Scripps ransomware post-mortem reveals significant ripple effects for nearby hospitals [Електронний ресурс] — Режим доступу: https://www.fiercehealthcare.com/health-tech/scripps-ransomware-post-mortem-shows-cybersecurity-regional-problem

13. U.S. Health and Human Services Department Suffers Cyberattack [Електронний ресурс] — Режим доступу: https://www.securitymagazine.com/articles/91909-us-health-and-human-services-department-suffers-cyberattack

14. CNA Financial Paid $40 Million in Ransom After March Cyberattack [Електронний ресурс] — Режим доступу: https://www.bloomberg.com/news/articles/2021-05-20/cna-financial-paid-40-million-in-ransom-after-march-cyberattack

15. Data breach affects over 60,000 customers of luxury retailer Neiman Marcus [Електронний ресурс] — Режим доступу: https://www.bitdefender.com/en-gb/blog/hotforsecurity/data-breach-affects-over-60-000-customers-of-luxury-retailer-neiman-marcus/

16. Російське вторгнення в Україну (з 2022) [Електронний ресурс] — Режим доступу: https://uk.wikipedia.org/wiki/Російське_вторгнення_в_Україну_(з_2022)

17. Марія Карчевич [Електронний ресурс] — Режим доступу: https://moz.gov.ua/uk/dev-mariya-karchevich

18. Про LB.UA [Електронний ресурс] — Режим доступу: https://lb.ua/about

19. “На день може бути і сотні атак”, — Марія Карчевич про спроби зламати електронну медсистему країни [Електронний ресурс] — Режим доступу: https://lb.ua/health/2024/10/17/640312_na_den_mozhe_buti_i_sotni_atak-.html

20. Delfs, Hans; Knebl, Helmut (2007). "Symmetric-key encryption". *Introduction to cryptography: principles and applications*. Springer. ISBN 9783540492436.

21. Поділ секрету Шаміра [Електронний ресурс] — Режим доступу : https://exbase.io/uk/wiki/podil-sekretu-shamira

22. Shamir's Secret Sharing: Explanation and Visualization [Електронний ресурс] — Режим доступу : https://evervault.com/blog/shamir-secret-sharing

23. On the asymptotic idealness of the Asmuth-Bloom threshold secret sharing scheme [Електронний ресурс] — Режим доступу : https://www.sciencedirect.com/science/article/abs/pii/S0020025518304869

24. Constructing Ideal Secret Sharing Schemes based on Chinese Remainder Theorem [Електронний ресурс] — Режим доступу : https://eprint.iacr.org/2018/837.pdf

25. On Secret Sharing Systems [Електронний ресурс] — Режим доступу : https://www-ee.stanford.edu/~hellman/publications/45.pdf

26. Revisiting the Karnin, Greene and Hellman Bounds https://www.win.tue.nl/~berry/papers/icits08kgh.pdf

27. Java Documentation [Електронний ресурс] — Режим доступу : https://docs.oracle.com/en/java/

28. IntelliJ IDEA The Leading Java and Kotlin IDE [Електронний ресурс] — Режим доступу : https://www.jetbrains.com/idea/

29. Welcome to Apache Maven [Електронний ресурс] — Режим доступу : https://maven.apache.o

30. JAR File Overview [Електронний ресурс] — Режим доступу : https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jarGuide.html

31. Eclipse IDE [Електронний ресурс] — Режим доступу : https://eclipseide.org/

32. Apache Netbeans IDE [Електронний ресурс] — Режим доступу : https://netbeans.apache.org/front/main/index.html

33. Web Archive (WAR) files [Електронний ресурс] — Режим доступу : https://www.ibm.com/docs/en/radfws/9.7?topic=files-web-archive-war

34. Debian 12 “bookworm” released [Електронний ресурс] — Режим доступу : https://www.debian.org/News/2023/20230610

35. Class System [Електронний ресурс] — Режим доступу : https://docs.oracle.com/javase/1.5.0/docs/api/java/lang/System.html

36. Java Virtual Machine Technology Overview [Електронний ресурс] — Режим доступу : https://docs.oracle.com/en/java/javase/23/vm/java-virtual-machine-technology-overview.html

37. Class Runtime [Електронний ресурс] — Режим доступу : https://docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html

38. Java Garbage Collector [Електронний ресурс] — Режим доступу : https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

39. Scope of DICOM [Електронний ресурс] — Режим доступу : https://dicom.nema.org/medical/dicom/current/output/chtml/part01/chapter_1.html#sect_1.1

40. Прикладні аспекти сучасних міждисциплінарних досліджень [Електроний ресурс] — Режим доступу : https://jpasmd.donnu.edu.ua/issue/view/537

ДОДАТКИ

Код програми

```
package org.example;


import java.io.File;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Arrays;
import java.util.List;


public class SecretSharingTest {


//  Тестуємо алгоритм секретного поділу з використанням різних методів
   public void testAlgorithm(SecretSharingAlgorithm algorithm, String filePath, int n, int k) throws
IOException {
        File file = new File(filePath);
        BigInteger secret = BigInteger.valueOf(file.length());


        System.out.println("Тестування з файлом: " + filePath);
        System.out.println("Розмір файлу: " + file.length() + " байтів");
        System.out.println("Секрет: " + secret);


        // Розподіл секрету
        //---------------------------------------------------------------
        // ---Вимірювання пам'яті
        Runtime runtime = Runtime.getRuntime();
        // Очищення пам'яті перед вимірюванням
        runtime.gc();
        // Використана пам'ять до операції
        long startMemoryBeforeShares = runtime.totalMemory() - runtime.freeMemory();
        List<?> sharesForMemoryCheck = algorithm.createShares(secret, n, k);
        // Використана пам'ять після операції
        long endMemoryAfterShares = runtime.totalMemory() - runtime.freeMemory();
        long memoryUsedForShares = endMemoryAfterShares - startMemoryBeforeShares;
```

```
// ---Вимірювання часу
double startTime = System.nanoTime();
List<?> sharesForTimeCheck = algorithm.createShares(secret, n, k);
System.out.println(sharesForTimeCheck.getFirst());
double endTime = System.nanoTime();

double shareTimeMillis = (endTime - startTime) / 1_000_000;

System.out.println("Час поділу секрету (мс): " + shareTimeMillis);
System.out.println("Використана пам'ять для поділу секрету (Кілобайт): " + memoryUsedForShares / 1024.0);

// Відновлення секрету
//----------------------------------------------------------------
// Вибір k часток для відновлення
List<?> selectedShares = sharesForMemoryCheck.subList(0, k);

// ---Вимірювання пам'яті
Runtime runtime1 = Runtime.getRuntime();
// Очищення пам'яті перед вимірюванням
runtime1.gc();
// Використана пам'ять до операції
long startMemoryBeforeRecovery = runtime1.totalMemory() - runtime1.freeMemory();
BigInteger recoveredSecretForMemoryCheck = algorithm.recoverSecret(selectedShares, k);
// Використана пам'ять після операції
long endMemoryAfterRecovery = runtime1.totalMemory() - runtime1.freeMemory();

// ---Вимірювання часу
double startRecoverTime = System.nanoTime();
BigInteger recoveredSecretForTimeCheck = algorithm.recoverSecret(selectedShares, k);
double endRecoverTime = System.nanoTime();

double recoverTimeMillis = (endRecoverTime - startRecoverTime) / 1_000_000;
```

```
        long memoryUsedForRecovery = endMemoryAfterRecovery - startMemoryBeforeRecovery;


        System.out.println("Час відновлення секрету (мс): " + recoverTimeMillis);
        System.out.println("Використана пам'ять для відновлення секрету (Кілобайт): " +
memoryUsedForRecovery / 1024.0);


        // Перевірка коректності відновленого секрету
        System.out.println("Чи співпадають секрети? " + secret.equals(recoveredSecretForMemoryCheck));
        System.out.println("Секрет: " + secret);
        System.out.println("Відновлений секрет: " + recoveredSecretForMemoryCheck);
        System.out.println("-------------------------------------------------");
    }


    public static void main(String[] args) {
        try {
            SecretSharingTest tester = new SecretSharingTest();
            String filePath = "/home/kuk/Downloads/0012.DCM";


            List<int[]> testCases = Arrays.asList(
                    new int[]{5, 4},
                    new int[]{10, 8},
                    new int[]{25, 20},
                    new int[]{50, 40}
            );
            Runtime runtime = Runtime.getRuntime();


            // Тест розподілу секрету за схемою Шаміра
            System.out.println("***** Тест розподілу секрету за схемою Шаміра *****");
            System.out.println("************************************************");
            SecretSharingAlgorithm shamir = new ShamirSecretSharing();
            for (int i = 0; i < 100; i++) {
                runtime.gc();
                tester.testAlgorithm(shamir, filePath, testCases.get(0)[0], testCases.get(0)[1]);
```

```
        }

        // Тест розподілу секрету за схемою Асмута-Блума
        System.out.println("***** Тест розподілу секрету за схемою Асмута-Блума *****");
        System.out.println("**********************************************");
        SecretSharingAlgorithm<BigInteger[]> asmuthBloom = new AsmuthBloomSecretSharing();
        for (int i = 0; i < 100; i++) {
            runtime.gc();
            tester.testAlgorithm(asmuthBloom, filePath, testCases.get(0)[0], testCases.get(0)[1]);

        }

        System.out.println("***** Тест розподілу секрету за схемою Карнін-Грін-Хеллмана *****");
System.out.println("************************************************************");
        SecretSharingAlgorithm<BigInteger> karninGreeneHellman = new
KarninGreeneHellmanSecretSharing();
        for (int i = 0; i < 100; i++) {
            runtime.gc();
            tester.testAlgorithm(karninGreeneHellman, filePath, testCases.get(0)[0], testCases.get(0)[1]);

        }

    } catch (IOException e) {
        throw new RuntimeException(e);
    }
  }

}

package org.example;

import java.math.BigInteger;
```

```
import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.List;

public class ShamirSecretSharing implements SecretSharingAlgorithm<BigInteger> {
  private static final SecureRandom RANDOM = new SecureRandom();  // Генератор випадкових чисел
  private static final BigInteger PRIME = new
BigInteger("340282366920938463463374607431768211507");  // Велике просте число для модулярних
операцій

  @Override
  public List<BigInteger> createShares(BigInteger secret, int n, int k) {
    validateParameters(secret, n, k);  // Перевірка вхідних параметрів

    secret = secret.mod(PRIME); // Перевірка, щоб секрет знаходився в межах простого числа PRIME
    List<BigInteger> shares = new ArrayList<>();  // Список для збереження часток
    BigInteger[] coefficients = generateCoefficients(k - 1);  // Генерація коефіцієнтів полінома

    // Генерація часток
    for (int i = 1; i <= n; i++) {
      BigInteger x = BigInteger.valueOf(i);  // Визначення x для поточної частки
      shares.add(evaluatePolynomial(secret, coefficients, x));  // Обчислення значення полінома для x
    }
    return shares;  // Повернення згенерованих часток
  }

  @Override
  public BigInteger recoverSecret(List<BigInteger> shares, int k) {
    if (shares == null || shares.size() < k) {
      throw new IllegalArgumentException("Недостатньо часток для відновлення секрету");  //
Перевірка кількості часток
    }

    return lagrangeInterpolation(shares, k);  // Відновлення секрету за допомогою інтерполяції
Лагранжа
```

```
    }

    // Перевірка вхідних параметрів
    private void validateParameters(BigInteger secret, int n, int k) {
        if (secret == null || secret.signum() < 0) {
            throw new IllegalArgumentException("Секрет має бути невід'ємним числом"); // Перевірка, чи є секрет невід'ємним числом
        }
        if (n <= 0 || k <= 0) {
            throw new IllegalArgumentException("n та k мають бути додатними числами"); // Перевірка, чи є n і k додатними числами
        }
        if (k > n) {
            throw new IllegalArgumentException("k не може бути більшим за n"); // Перевірка, чи не більше k за n
        }
    }

    // Генерація випадкових коефіцієнтів для полінома
    private BigInteger[] generateCoefficients(int count) {
        BigInteger[] coefficients = new BigInteger[count]; // Масив для збереження коефіцієнтів
        for (int i = 0; i < count; i++) {
            do {
                coefficients[i] = new BigInteger(PRIME.bitLength(), RANDOM).mod(PRIME); // Генерація випадкового коефіцієнта в межах PRIME
            } while (coefficients[i].equals(BigInteger.ZERO)); // Перевірка, щоб коефіцієнт не був нульовим
        }
        return coefficients; // Повернення масиву коефіцієнтів
    }

    // Оцінка значення полінома в точці x за допомогою методу Горнера
    private BigInteger evaluatePolynomial(BigInteger secret, BigInteger[] coefficients, BigInteger x) {
        BigInteger result = secret; // Початковий результат - це сам секрет
        BigInteger power = BigInteger.ONE; // Потужність для обчислення полінома
```

```
        for (BigInteger coeff : coefficients) {
            power = power.multiply(x).mod(PRIME);  // Обчислення степеня x
            result = result.add(coeff.multiply(power)).mod(PRIME);  // Додавання коефіцієнта до результату полінома
        }
        return result;  // Повернення результату полінома
    }

    // Відновлення секрету за допомогою інтерполяції Лагранжа
    private BigInteger lagrangeInterpolation(List<BigInteger> shares, int k) {
        BigInteger secret = BigInteger.ZERO;  // Початкове значення секрету

        // Перебір кожної частки для обчислення її внеску в секрет
        for (int i = 0; i < k; i++) {
            BigInteger x_i = BigInteger.valueOf(i + 1);  // x_i - координата x для поточної частки
            BigInteger share_i = shares.get(i);  // y_i - значення частки

            BigInteger numerator = BigInteger.ONE;  // Чисельник для коефіцієнта Лагранжа
            BigInteger denominator = BigInteger.ONE;  // Знаменник для коефіцієнта Лагранжа

            // Обчислення чисельника і знаменника коефіцієнта Лагранжа
            for (int j = 0; j < k; j++) {
                if (i != j) {
                    BigInteger x_j = BigInteger.valueOf(j + 1);  // x_j - координата x іншої частки

                    // Обчислення чисельника і знаменника за допомогою різниці (x_i - x_j)
                    numerator = numerator.multiply(x_j.negate()).mod(PRIME);
                    denominator = denominator.multiply(x_i.subtract(x_j)).mod(PRIME);
                }
            }

            // Обчислення коефіцієнта Лагранжа для цієї частки
            BigInteger lagrangeCoeff = numerator.multiply(denominator.modInverse(PRIME)).mod(PRIME);
```

```
            // Додавання внеску цієї частки до секрету
            secret = secret.add(share_i.multiply(lagrangeCoeff)).mod(PRIME);
        }

        return secret;  // Повернення відновленого секрету
    }
}

package org.example;

import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.List;

public class AsmuthBloomSecretSharing implements SecretSharingAlgorithm<BigInteger[]> {
    private static final SecureRandom RANDOM = new SecureRandom();  // Генератор випадкових чисел

    @Override
    public List<BigInteger[]> createShares(BigInteger secret, int n, int k) {
        validateParameters(secret, n, k);  // Перевірка вхідних параметрів

        // Генерація взаємно простих модулів
        List<BigInteger> moduli = generateCoprimeModuli(n, k, secret);
        BigInteger m0 = moduli.get(0);  // Спеціальний модуль для секрету
        BigInteger productOfFirstK = calculateProduct(moduli.subList(1, k + 1));  // Обчислення добутку перших k модулів

        // Перевірка обмежень для використання КНР (Китайської теореми залишків)
        if (!(m0.multiply(BigInteger.TWO).compareTo(productOfFirstK) < 0)) {
            throw new IllegalStateException("Згенеровані модулі не задовольняють обмеження Asmuth-Bloom");
        }
```

```
        // Генерація часток
        List<BigInteger[]> shares = new ArrayList<>();
        BigInteger s = secret.mod(m0);  // Секрет по модулю m0
        for (int i = 1; i <= n; i++) {
            BigInteger mod = moduli.get(i);  // Поточний модуль
            BigInteger share = s.mod(mod);  // Частка для поточного модуля
            shares.add(new BigInteger[]{mod, share});  // Додавання частки до списку
        }

        return shares;  // Повернення згенерованих часток
    }

    @Override
    public BigInteger recoverSecret(List<BigInteger[]> shares, int k) {
        if (shares == null || shares.size() < k) {
            throw new IllegalArgumentException("Недостатньо часток для відновлення секрету");  // Перевірка, чи є достатньо часток
        }

        // Сортування часток за модулями (рекомендується для узгодженості)
        shares.sort((a, b) -> a[0].compareTo(b[0]));

        // Вибір перших k часток
        List<BigInteger[]> selectedShares = shares.subList(0, k);

        // Відновлення секрету за допомогою КНР
        return reconstructWithCRT(selectedShares);
    }

    // Перевірка вхідних параметрів
    private void validateParameters(BigInteger secret, int n, int k) {
        if (secret == null || secret.signum() < 0) {
            throw new IllegalArgumentException("Секрет має бути невід'ємним числом");  // Перевірка, чи є секрет невід'ємним числом
```

```
        }
        if (n <= 0 || k <= 0 || k > n) {
            throw new IllegalArgumentException("Невірні параметри: n та k мають бути додатними, і k <= n");  // Перевірка параметрів n та k
        }
    }

    // Генерація списку взаємно простих модулів
    private List<BigInteger> generateCoprimeModuli(int n, int k, BigInteger secret) {
        List<BigInteger> moduli = new ArrayList<>();
        BigInteger lowerBound = secret.multiply(BigInteger.TWO);  // Перевірка, що модулі більше за 2 * секрет
        BigInteger candidate = lowerBound.add(BigInteger.ONE);

        while (moduli.size() < n + 1) {  // Потрібно n + 1 модулів (включаючи m0)
            if (isPrime(candidate) && isPairwiseCoprime(candidate, moduli)) {
                moduli.add(candidate);  // Додавання модулю, якщо він простий та взаємно простий з іншими
            }
            candidate = candidate.add(BigInteger.ONE);  // Перевірка наступного кандидата
        }

        return moduli;  // Повернення списку модулів
    }

    // Перевірка, чи є число простим
    private boolean isPrime(BigInteger number) {
        return number.isProbablePrime(100);  // Використання ймовірного тесту простоти
    }

    // Перевірка, чи є число взаємно простим з усіма числами в списку
    private boolean isPairwiseCoprime(BigInteger number, List<BigInteger> moduli) {
        for (BigInteger mod : moduli) {
            if (!number.gcd(mod).equals(BigInteger.ONE)) {  // Перевірка на взаємну простоту
                return false;
```

```
            }
        }
        return true;  // Якщо всі модулі взаємно прості, повертається true
    }

    // Обчислення добутку чисел у списку
    private BigInteger calculateProduct(List<BigInteger> numbers) {
        BigInteger product = BigInteger.ONE;
        for (BigInteger number : numbers) {
            product = product.multiply(number);  // Множення всіх чисел
        }
        return product;  // Повернення результату
    }

    // Відновлення секрету за допомогою Китайської теореми залишків
    private BigInteger reconstructWithCRT(List<BigInteger[]> shares) {
        BigInteger mProduct = BigInteger.ONE;
        for (BigInteger[] share : shares) {
            mProduct = mProduct.multiply(share[0]);  // Множення всіх модулів
        }

        BigInteger secret = BigInteger.ZERO;
        for (BigInteger[] share : shares) {
            BigInteger mod = share[0];  // Поточний модуль
            BigInteger remainder = share[1];  // Залишок для поточної частки
            BigInteger mPartial = mProduct.divide(mod);  // Частина добутку без поточного модуля
            BigInteger mInverse = mPartial.modInverse(mod);  // Обчислення оберненого числа по модулю
            secret = secret.add(remainder.multiply(mPartial).multiply(mInverse)).mod(mProduct);  // Додавання
внеску поточної частки до секрету
        }

        return secret;  // Повернення відновленого секрету
    }
}
```

```
package org.example;

import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.List;

public class KarninGreeneHellmanSecretSharing implements SecretSharingAlgorithm<BigInteger> {
    private static final SecureRandom RANDOM = new SecureRandom();  // Генератор випадкових чисел
    private static final BigInteger PRIME = new
BigInteger("340282366920938463463374607431768211507"); // Велике просте число для модульної
арифметики

    @Override
    public List<BigInteger> createShares(BigInteger secret, int n, int k) {
        validateParameters(secret, n, k);  // Перевірка параметрів

        // Генерація випадкового вектора `u` розмірності `k`
        BigInteger[] u = generateRandomVector(k);

        // Обчислення скалярного добутку `u` та загальновідомого вектора `V0`
        BigInteger[] v0 = generateRandomVector(k);

        BigInteger scalarSecret = scalarProduct(u, v0).mod(PRIME);  // Обчислення секрету

        // Перевірка, чи обчислений секрет відповідає заданому
        if (!scalarSecret.equals(secret.mod(PRIME))) {
            throw new IllegalArgumentException("Заданий секрет не відповідає обчисленому скалярному
добутку (u, V0)");
        }

        // Генерація (n + 1) випадкових векторів Vi розмірності `k`
        List<BigInteger[]> vectors = new ArrayList<>();
        vectors.add(v0); // V0 відомий усім учасникам
        for (int i = 0; i < n; i++) {
            vectors.add(generateRandomVector(k));
        }
```

```
        // Обчислення часток як скалярних добутків u та Vi
        List<BigInteger> shares = new ArrayList<>();
        for (int i = 1; i <= n; i++) {
            BigInteger share = scalarProduct(u, vectors.get(i)).mod(PRIME);  // Обчислення частки
            shares.add(share);
        }

        return shares;  // Повернення списку часток
    }

    @Override
    public BigInteger recoverSecret(List<BigInteger> shares, int k) {
        if (shares == null || shares.size() < k) {
            throw new IllegalArgumentException("Недостатньо часток для відновлення секрету");  // Перевірка кількості часток
        }

        // Відновлення вектора `u` через розв'язання системи лінійних рівнянь
        BigInteger[][] matrix = generateRecoveryMatrix(k);
        BigInteger[] u = solveLinearEquations(matrix, shares.subList(0, k));

        // Обчислення секрету, використовуючи відновлений вектор `u` та загальний `V0`
        BigInteger[] v0 = generateRandomVector(k); // V0 повинен бути тим самим, що й під час створення
        return scalarProduct(u, v0).mod(PRIME);
    }

    // Перевірка вхідних параметрів
    private void validateParameters(BigInteger secret, int n, int k) {
        if (secret == null || secret.signum() < 0) {
            throw new IllegalArgumentException("Секрет має бути невід'ємним числом");  // Перевірка, чи секрет є невід'ємним
        }
        if (n <= 0 || k <= 0) {
            throw new IllegalArgumentException("n та k мають бути додатними числами");  // Перевірка значень n та k
        }
        if (k > n) {
            throw new IllegalArgumentException("k не може бути більшим за n");  // Перевірка, чи k не
```

```
перевищує n
        }
    }

    // Генерація випадкового вектора заданої розмірності
    private BigInteger[] generateRandomVector(int dimension) {
        BigInteger[] vector = new BigInteger[dimension];
        for (int i = 0; i < dimension; i++) {
            vector[i] = new BigInteger(PRIME.bitLength(), RANDOM).mod(PRIME);  // Випадкове число по
модулю PRIME
        }
        return vector;
    }

    // Обчислення скалярного добутку двох векторів
    private BigInteger scalarProduct(BigInteger[] u, BigInteger[] v) {
        if (u.length != v.length) {
            throw new IllegalArgumentException("Вектори мають бути однакової розмірності");  // Перевірка
розмірності векторів
        }

        BigInteger result = BigInteger.ZERO;
        for (int i = 0; i < u.length; i++) {
            result = result.add(u[i].multiply(v[i])).mod(PRIME);  // Додавання добутків координат
        }
        return result;  // Повернення результату
    }

    // Генерація матриці для відновлення секрету
    private BigInteger[][] generateRecoveryMatrix(int k) {
        BigInteger[][] matrix = new BigInteger[k][k];
        for (int i = 0; i < k; i++) {
            for (int j = 0; j < k; j++) {
                matrix[i][j] = new BigInteger(PRIME.bitLength(), RANDOM).mod(PRIME);  // Генерація
випадкових значень
            }
        }
        return matrix;  // Повернення матриці
```

```
    }

    // Розв'язання системи лінійних рівнянь
    private BigInteger[] solveLinearEquations(BigInteger[][] matrix, List<BigInteger> constants) {
        int n = matrix.length;
        BigInteger[] result = new BigInteger[n];

        // Гауссове виключення
        for (int i = 0; i < n; i++) {
            // Приведення діагонального елемента до 1
            BigInteger diagInverse = matrix[i][i].modInverse(PRIME);
            for (int j = 0; j < n; j++) {
                matrix[i][j] = matrix[i][j].multiply(diagInverse).mod(PRIME);
            }
            constants.set(i, constants.get(i).multiply(diagInverse).mod(PRIME));

            // Усунення елементів нижче діагоналі
            for (int j = i + 1; j < n; j++) {
                BigInteger factor = matrix[j][i];
                for (int k = 0; k < n; k++) {
                    matrix[j][k] = matrix[j][k].subtract(factor.multiply(matrix[i][k])).mod(PRIME);
                }
                constants.set(j, constants.get(j).subtract(factor.multiply(constants.get(i))).mod(PRIME));
            }
        }

        // Зворотна підстановка
        for (int i = n - 1; i >= 0; i--) {
            result[i] = constants.get(i);
            for (int j = i + 1; j < n; j++) {
                result[i] = result[i].subtract(matrix[i][j].multiply(result[j])).mod(PRIME);
            }
        }

        return result;  // Повернення розв'язаних значень
    }
}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>secret_sharing_protocols</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>23</maven.compiler.source>
    <maven.compiler.target>23</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>commons-io</groupId>
      <artifactId>commons-io</artifactId>
      <version>2.14.0</version>
    </dependency>
  </dependencies>


</project>
```