Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
МАГІСТРА

на тему *Метод шифрованої передачі даних між хмарними підпросторами*

Галузь знань *12 – Інформаційні технології*
Шифр і назва галузі знань

Спеціальність *122 – Комп'ютерні науки*
Шифр і назва спеціальності

Виконав: студент *2* курсу, група *КНм-20-1*		*В.І. Заровний*	
	Підпис	Ініціали, прізвище	
Керівник:	*к.ф-м.н., доцент кафедри КН*	*В.Ц. Міхалевський*	
	Підпис	Ініціали, прізвище	
Нормоконтроль:	*к.т.н., доцент кафедри КН*	*Р.О. Багрій*	
	Підпис	Ініціали, прізвище	

До захисту допускаю:
Зав. кафедри КН, д.т.н., професор Підпис *О.В. Бармак* Ініціали, прізвище

_25_ _листопада_ 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Факультет *інформаційних тегнологій*
Кафедра *комп'ютерних наук*
Освітній ступінь *магістр*
Галузь знань *12 – Інформаційні технології*
Спеціальність *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук

_____
(підпис)
д.т.н., професор О.В. Бармак
«1» вересня 2021 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

1. Тема кваліфікаційної роботи магістра: *«Метод шифрованої передачі даних між хмарними підпросторами»*

2. Завдання видано студенту *Заровному Владиславу Ігоровичу*
(прізвище, ім'я, по батькові)

3. Керівник роботи *к.ф-м.н., доцент Міхалевський Віталій Цезарійович*
(прізвище, ім'я, по батькові)

4. Затверджені наказом університету від «25» серпня 2021 р. № 102

5. Зміст пояснювальної записки (перелік задач) та вихідні дані:

Мета роботи – розробка інформаційної системи шифрованої передачі даних між хмарними підпросторами. Об'єктом дослідження є інформаційна технологія передачі шифрованих даних при різних алгоритмах шифрування та підбору значень між хмарними підпросторами. Предметом дослідження є набори даних, отриманих із параметрів переданих у зовнішній інтерфейс бібліотеки, це можуть бути різнотипні дані, такі як об'єкт, список, рядок, число тощо.

Реферат

Кваліфікаційна робота магістра присвячена розробці інформаційної системи шифрованої передачі даних між хмарними підпросторами. Розробка такої технології дозолить іншим користувачам розглядати цей продукт як зовнішню бібліотеку для шифрування даних, що передаються між хмарними сервісами, та забезпечити контроль та цілісність переданої інформації.

Актуальність теми. В сучасному світі хмарних розробок все більше сервісів переносяться на хмарні платформи, що дозволяє їх масштабувати в залежності від навантаження на сервіси. Даний підхід вимагає мікросервісної архітектури проектів, де багато сервісів взаємодіють між собою шляхом передачі даних через інтернет. Для передачі даних використовується HTTP, що уможливлює перехоплення даних, які передаються між сервісами, та може використовуватися зловмисниками. Важливо, щоб у разі перехоплення даних зловмисники не мали можливості зрозуміти, які саме дані передавалися між сервісами. Це особливо є актуальним у сфері онлайн оплати. Шифрування даних кредитних карток та інформації про оплату унеможливлює спроби шахраїв використовувати дані карток. Тому виникає необхідність розробки комплексного підходу у визначені алгоритмів та створення інформаційної системи для передачі даних між хмарними підпросторами.

Метою дослідження є розробка інформаційної технології шифрованої передачі даних між хмарними підпросторами.

Для досягнення поставленої мети поставлені наступні **задачі**:

– дослідження необхідності використання шифрування даних для передачі їх між хмарними сервісами;

– розробка інформаційної системи передачі даних між хмарними підпросторами.

При цьому необхідно вирішити такі **підзадачі**:

– дослідження предметної області – різних підпросторів збереження даних;

– розробка алгоритмів та структури системи передачі даних;

– оцінка ризиків передачі шифрованих та нешифрованих даних;

– надання алгоритму використання ключів для шифрування;

– формування переліку програмних кодів як бібліотеки для використання іншими системами;

– надання рекомендаційних підходів для шифрування даних;

– тестування розробленої системи.

Об’єктом дослідження є інформаційна технологія передачі даних з використанням алгоритмів шифрування/дешифрування між різними хмарними підпросторами.

Предметом дослідження є структури даних, отримані для шифрування, імплементація алгоритмів шифрування та інтерфейс підключення для зовнішньої бібліотеки.

Достовірність роботи системи перевіряється проведенням системного аналізу та перевіркою системи за допомогою тестування на різних типах даних з використанням різним алгоритмів шифрування даних.

Наукова новизна одержаних результатів:

– вперше запропоновано метод, що дозволяє обрати найоптимальніший алгоритм шифрування для передачі його через хмарні сервіси;

– набули подальшого розвитку існуючі методи шифрування даних в частині отримання та обробки даних;

– розроблено алгоритм вибору шифрування даних, який пришвидшує та уточнює процес шифрування даних;

– вдосконалено процес прийняття рішення стосовно передачі даних між хмарними сервісами.

Практична значимість дослідження полягає в тому, що результати досліджень мають допустимий рівень швидкодії, а розглянуті набори даних мають застосування в реальному житті. Даний підхід має результат у вигляді

зашифрованих даних, тобто видозмінених даних, які передавалися на початку, у вказану бібліотеку. Зашифровані дані надалі можна використовувати для передачі між різними носіями. Також головним моментом є те, що ці дані повинні бути дешифруванні за допомогою ключа, який використовувався при шифруванні. У такому випадку результатом повинні бути дані, які відповідають тим, що були передані у бібліотеку. Подальшу роботу даного шифрування повинна перевірити система з безпеки, яка буде намагатися їх дешифрувати за допомогою різних підходів. В ситуації, коли система не зможе розшифрувати дані, це буде означати, що це є доцільний та безпечний алгоритми шифрування і його можна використовувати для передачі різних персональних даних.

Апробація результатів кваліфікаційної роботи магістра та публікації.

Основні наукові та практичні результати доповідалися на конференції: доповідь на тему «Метод шифрованої передачі даних між хмарними підпросторами» на XIII Всеукраїнській науково-практичній конференції «Актуальні проблеми комп'ютерних наук АПКН-2021» (Хмельницький, 15-16 жовтня 2021 року, Хмельницький національний університет).

За темою кваліфікаційної роботи магістра автором виконана одна наукова публікація: Заровний В.І. Метод шифрованої передачі даних між хмарними підпросторами / Заровний В.І., Скрипник Т.К. // Збірник наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021». – Хмельницький, 2021. – С. 335-337.

Структура та обсяг роботи. Кваліфікаційна робота магістра складається з завдання, реферату, змісту, вступу, 4 розділів, висновків, переліку посилань із 29 найменування та додатків. Загальний обсяг кваліфікаційної роботи магістра становить 110 сторінок, з них 81 сторінка основного тексту та 29 сторінок додатків.

Ключові слова: шифрування даних, метод шифрування даних, хмарні підпростори, паралельне опрацювання даних.

Зміст

Перелік скорочень **4**
Вступ **5**
Розділ 1 ..... **11**
Аналіз методів шифрованої передачі даних **11**
1.1 Опис предметної області 11
1.2 Програмні засоби для шифрування 12
1.3 Процес шифрування та дешифрування даних 19
1.4 Постановка задачі 25
Висновки до розділу 1 25
Розділ 2 ..... **27**
Математичні моделі алгоритмів **27**
2.1 Криптографічні системи 27
2.2 Симетричні криптосистеми 32
2.3 Асиметричні криптосистеми 34
2.4 Паралельні обчислення 44
Висновки до розділу 2 48
Розділ 3 ..... **50**
Розробка системи шифрованої передачі даних **50**
3.1 Опис структури та інтерфейсу 50
3.2 Вибір оптимального режиму шифрування 53
3.3 Розподіл вхідних даних на блоки 58
Висновки до розділу 3 60
Розділ 4 ..... **62**
Дослідження ефективності метода шифрованої передачі даних між хмарними підпросторами **62**
4.1 Системи підбору алгоритму для опрацювання даних 62
4.2 Розробка моделі перетворення даних на паралельних обчисленнях 67
4.2.1 Приклад моделі шифрування 69

4.2.2 Процедура захисту даних від змін 70
4.2.3 Процедура верифікації блоків з даними 71
4.2.4 Особливості використання паралельного обрахунку даних 71
4.3 Функціональне дослідження системи шифрування даних 72
4.3.1 Апробація прототипів моделі 72
4.3.2 Опис тестового середовища 73
4.3.3 Обмеження експерименту 73
4.3.4 Результати експериментів 73
4.3.5 Підсумок результатів тестування моделі 75
Висновки до розділу 4 76
Загальні висновки 79
Перелік посилань 82
Додатки

Перелік скорочень

Скорочення, термін, позначення	Пояснення
АІС	Автоматизована інформаційна система
СПД	Система передачі даних
СФ	Система фільтрації
АШ	Алгоритм шифрування
РНБ	Розподіл на блоки

Вступ

Сьогодні майже в кожного є доступ до мережі Internet і це дає багато переваг кожному користувачу мережі. Зараз все більше сервісів знаходяться в режимі онлайн та кожен користувач може ними скористатися. Це може бути як онлайн шопінг, так і оплата за комунальні послуги.

Все більше людей залучаються до мережі Internet та максимально використовують послуги, які надають їм різні IT компанії.

Ми можемо оплатити різні покупки товарів за допомогою Internet, але для цього користувач повинен передавати свої персональні дані. Як ця людина може бути впевнена, що її дані є захищеними і коли він водить, для прикладу, номер своєї кредитної карти, що він не буде викрадений зловмисниками та використаний у незаконних цілях? Кожна IT компанія, яка надає онлайн рішення, повинна гарантувати захист даних всіх своїх користувачів, що вони не будуть викрадені, а якщо і будуть викрадені, то щоб ніхто не зміг зрозуміти, що це за дані.

Тут на допомогу приходять алгоритми шифрування, які розробники додають при розробці конкретного продукту. Алгоритми шифрування є дуже важливими та необхідними для того, щоб захистити персональні дані користувача. Проблема в тому, що кожен алгоритм шифрування призначений під конкретні цілі і так часто буває, що розробники не знають для якої цілі який алгоритм використовувати.

З усіма цими питаннями може впоратись одна зовнішня бібліотека, яка допоможе вирішити всі нюанси при шифруванні даних, а саме бібліотека, яка буде вміщувати в собі перелік всіх поширених алгоритмів, включати новий алгоритм та сама зможе визначити, до якого типу відноситься дані та як яким чином їх треба зашифрувати.

Актуальність теми. В сучасному світі хмарних розробок все більше сервісів переносяться на хмарні платформи, що дозволяє їх масштабувати в

залежності від навантаження на сервіси. Даний підхід вимагає мікросервісної архітектури проектів, де багато сервісів взаємодіють між собою шляхом передачі даних через інтернет. Для передачі даних використовується HTTP, що уможливлює перехоплення даних, які передаються між сервісами, та може використовуватися зловмисниками. Важливо, щоб у разі перехоплення даних зловмисники не мали можливості зрозуміти, які саме дані передавалися між сервісами. Це особливо є актуальним у сфері онлайн оплати. Шифрування даних кредитних карток та інформації про оплату унеможливлює спроби шахраїв використовувати дані карток. Тому виникає необхідність розробки комплексного підходу у визначені алгоритмів та створення інформаційної системи для передачі даних між хмарними підпросторами.

Наразі у світі є декілька стандартів шифрування, кожний з яких призначений для конкретної цілі. Кожен з цих алгоритмів шифрування зарекомендував себе як чудовий варіанти захисту даних, але цими варіантами потрібно вміти користуватися і потрібно мати до них доступ. В залежності від того, який тип даних передається через мережу, потрібно використовувати той чи інший засіб шифрування. Всі ці питання можна вирішити за допомогою зовнішньої бібліотеки, яка буде вміщувати в собі базові алгоритми шифрування і тоді розробник, розуміючи тип даних та рівень захисту, який необхідний при передачі, може з легкістю використати цю бібліотеку.

Метою дослідження є розробка інформаційної технології шифрованої передачі даних між хмарними підпросторами.

Для досягнення поставленої мети, поставлені наступні **задачі**:

– дослідження необхідності використання шифрування даних для передачі їх між хмарними сервісами;

– розробка інформаційної системи передачі даних між хмарними підпросторами.

При цьому необхідно вирішити такі **підзадачі**:

– дослідження предметної області – різних підпросторів збереження даних;

– розробка алгоритмів та структури системи передачі даних;

– оцінка ризиків передачі шифрованих та нешифрованих даних;

– надання алгоритму використання ключів для шифрування;

– формування переліку програмних кодів як бібліотеки для використання іншими системами;

– надання рекомендаційних підходів для шифрування даних;

– тестування розробленої системи.

Об’єктом дослідження є інформаційна технологія передачі даних з використанням алгоритмів шифрування/дешифрування між різними хмарними підпросторами.

Предметом дослідження є структури даних, отримані для шифрування, імплементація алгоритмів шифрування та інтерфейс підключення для зовнішньої бібліотеки.

Система шифрування дозволяє зашифрувати дані для відправки мережею. Шифрування - обернене перетворення інформації з метою приховування від неавторизованих осіб з наданням, в цей же час, авторизованим користувачам доступу до неї. Головним чином, шифрування служить для дотримання конфіденційності інформації, що передається. Важливою особливістю будь-якого алгоритму шифрування є використання ключа, який підтверджує вибір конкретного перетворення із сукупності можливих для даного алгоритму.

Система буде представлена у вигляді бібліотеки API (прикладний програмний інтерфейс) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено — це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

У роботі буде представлено новий алгоритм шифрування створений з урахуванням підходів інших алгоритмів шифрування. Повинен бути детальний аналіз наявних алгоритмів шифрування та побудова нового алгоритму з урахуванням найкращих підходів. До аналізу будуть взяті наступні алгоритми:

MD5 - 128-бітний алгоритм хешування. Призначений для створення «відбитків» або дайджестів повідомлення довільної довжини та наступної перевірки їх автентичності.

CRC32 - алгоритм знаходження контрольної суми, призначений для перевірки цілісності даних. CRC є практичним додатком завадостійкого кодування, заснованому на певних математичних властивостях циклічного коду.

SHA1 - алгоритм криптографічного хешування. Описаний в RFC 3174. Для вхідного повідомлення довільної довжини (максимум 2^{64} - 1 біт , алгоритм генерує 160-бітове хеш-значення, зване також дайджестом повідомлення. Використовується в багатьох криптографічних додатках і протоколах. Також рекомендований як основний для державних установ у США.

SHA-2 - сімейство криптографічних алгоритмів - однонапрямлених хеш-функцій. Хеш-функції призначені для створення «відбитків» або «дайджестів» повідомлень довільної бітової довжини. Застосовуються в різних додатках або компонентах, пов'язаних із захистом інформації.

Аналіз вказаних алгоритмів дозволить краще розробити метод шифрування, з урахуванням нюансів кожного підходу.

Достовірність роботи системи перевіряється проведенням системного аналізу та перевіркою системи за допомогою тестування на різних типах даних з використанням різним алгоритмів шифрування даних.

Є кілька ключових факторів, що дозволяють визначити правильність виконання поставлених задач системою шифрування. Також зашифровані дані повинні перевірятися на криптостійкість - це характеристика шифру, визначальна його стійкість до дешифрування. Звичайно ця характеристика визначається періодом часу, необхідним для дешифрування. Добре розроблена

система шифрування має легко використовуватися як інтерфейс для будь-якої системи, яка планує під’єднати цю бібліотеку. Шифрування можна назвати правильним, якщо після передачі довільної структури даних відбулася видозміна цих даних за допомогою ключового слова, після чого їх можна повернути до попереднього нормального виду (дешифрувати) за допомогою того самого ключового слова.

Наукова новизна одержаних результатів:

– Вперше запропоновано метод, що дозволяє обрати найоптимальніший алгоритм шифрування для передачі його через хмарні сервіси;

– Набули подальшого розвитку існуючі методи шифрування даних в частині отримання та обробки даних;

– Розроблено алгоритм вибору шифрування даних, який пришвидшує та уточнює процес шифрування даних;

– Вдосконалено процес прийняття рішення стосовно передачі даних між хмарними сервісами.

Практична значимість дослідження полягає в тому, що результати досліджень мають допустимий рівень швидкодії, а розглянуті набори даних мають застосування в реальному житті.

Даний підхід має результат у вигляді зашифрованих даних, тобто видозмінених даних, які передавалися на початку, у вказану бібліотеку.

Зашифровані дані надалі можна використовувати для передачі між різними носіями, також головним моментом є те, що ці дані повинні бути дешифруванні, за допомогою ключа, який використовувався при шифруванні, у такому випадку результатом повинні бути дані які відповідають тим, що були передані у бібліотеку.

Подальшу роботу даного шифрування, повинна перевірити система з безпеки, яка буде намагатися їх дешифрувати, за допомогою різних підходів. В ситуації коли система не зможе розшифрувати дані, це буде означати, що це є

доцільний, та безпечний алгоритми шифрування і його можна використовувати для передачі різних персональних даних.

Апробація результатів кваліфікаційної роботи магістра та публікації.

Основні наукові та практичні результати доповідалися на конференції: доповідь на тему «Метод шифрованої передачі даних між хмарними підпросторами» на XIII Всеукраїнській науково-практичній конференції «Актуальні проблеми комп'ютерних наук АПКН-2021» (Хмельницький, 15-16 жовтня 2021 року, Хмельницький національний університет).

За темою кваліфікаційної роботи магістра автором виконана одна наукова публікація: Заровний В.І. Метод шифрованої передачі даних між хмарними підпросторами / Заровний В.І., Скрипник Т.К. // Збірник наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021». – Хмельницький, 2021. – С. 335-337.

Структура та обсяг роботи. Кваліфікаційна робота магістра складається з завдання, реферату, змісту, вступу, 4 розділів, висновків, переліку посилань із 29 найменування та додатків. Загальний обсяг кваліфікаційної роботи магістра становить 110 сторінок, з них 81 сторінка основного тексту та 29 сторінок додатків.

Розділ 1
Аналіз методів шифрованої передачі даних

1.1 Опис предметної області

Встановлення безпечного зв’язку між сервером та клієнтом - це важка та важлива справа. Потрібно розуміти, що в сьогоденні все важче створювати алгоритми, до яких не зможуть знайти ключ. Сучасні технології налічують багато алгоритмів шифрування, які підходять для конкретних типів даних або для конкретних ситуацій. Цікавий той факт, що більшість алгоритмів мають властивість старіти, це означає, що до певного алгоритму, на протязі довго часу, можуть підібрати спосіб, як його дешифрувати без знань ключа і це також вносить корективи в створення безпечного з’єднання між клієнтом та сервером.

В залежності від типу шифрування ми можемо сказати про швидкодію даного підходу. Загалом є два типи шифрування - це симетричне та асиметричне шифрування. Для того, щоб розібратися, яке шифрування краще використовувати, потрібно дотримуватися декількох правил:

1. Швидкість виконання шифрування даних.

В залежності від того, який алгоритм шифрування ми використовуємо, ми можемо зрозуміти, яка швидкість буде виконання завдань. Тут можна сказати, що асиметричне шифрування буде повільніше, тому що воно використовує інші алгоритми, які працюють довше. Що стосується симетричного шифрування, то воно працює швидше та використовує малий об’єм обрахунків, так як використовується лише один ключ для шифрування та дешифрування даних [9].

2. Аутентифікація користувача:

Асиметричне шифрування, на відміну від симетрично, використовує пару ключів, які математично між собою пов’язані, один з яких публічний, один приватний. За рахунок цього ми можемо зрозуміти і провести аутентифікацію людини, яка буде дешифрувати ці дані, тобто ми можемо сказати, що дані

необхідно зашифрувати за допомогою публічного ключа і завжди лише одна людина зможе їх дешифрувати, людина яка має приватний ключ [6].

3. Обсяг даних для шифрування:

Для швидкості шифрування потрібно розуміти, який обсяг даних буде шифруватися Коли ми працюємо з великими об’ємами даних дуже важливим вважається симетричний алгоритми шифрування, так як він працює набагато швидше ніж асиметричний, коли ж обсяг даних невеликий, рекомендовано використовувати асиметричне шифрування, так як воно вважається більш надійним [6].

З огляду на ці правила, стає зрозуміло, що для того щоб розробник вибрав коректний алгоритми шифрування, йому необхідно враховувати багато різних факторів та знати особливості кожного алгоритму.

При знанні вихідних параметрів, можна все ж таки визначити найкращий алгоритми шифрування, або навіть їх комбінацію, для того щоб отримати максимально вигоду з даної операції.

Ці всі критерії та правила написані, щоб зрозуміти природу поводження алгоритмів шифрування та специфіку отримання інформації для її перевірки.

1.2 Програмні засоби для шифрування

Сьогодні є досить багато систем у вільному доступі, які дозволяють організовувати шифрування даних та дають можливість самостійно розробити або запрограмувати систему із вже існуючих алгоритмів та пристосувати її до особливостей конкретної розробки.

Є великий список бібліотек, які можна додавати до кожного проекту в залежності від мови програмування, яка використовується. Ці бібліотеки дозволяють без глибоких математичних знань швидко підключати алгоритми шифрування та їх застосовувати.

Одна з найбільш популярних криптографічних бібліотек, яка вміщує в собі досить великий спектр функцій, називається «Bouncy Castle».

Ця бібліотека розрахована на використання у двох мовах програмування, таких як Java та C#.

Саме підтримка цих двох мов дозволяє застосовувати цю бібліотеку на більше, ніж 26 платформах. API цієї бібліотеки реалізовано за допомогою мови програмування C++ та має відкрити вихідний код. Особливості цього підходу дозволяють підтримувати 32 та 64 розрядну архітектуру систем, це дозволяє запускати її та використовувати на таких операційних системах як Android, Macintosh, Linux та Windows. Самі ключі генерується за допомогою випадкових чисел, які в собі генерує сама бібліотека та використовує при шифрування. Сама бібліотека немає обмежень стосовно довжини ключа RSA та вона сама автоматично вибирає найбільш кращий алгоритм шифрування для захисту системи за допомогою симетричних ключів при обміні.

Програмна система KeePass. Ще одніжю з найпопулярніших та широко доступних систем шифрування є система KeePass.

Це система була розроблена для Windows та забезпечує зручне зберігання усіх паролів, які використовуються в даній операційні системі. Ця система застосовує такі алгоритми як Twofish та Advanced Standart. Це зручний спосіб зберігання всіх ваших паролей, так як потім ця система надає можливість експортувати їх в різні формати та переносити між операційними системами. KeePass - це безкоштовний менеджер паролів з відкритим кодом, який допомагає безпечно керувати паролями. Ви можете зберігати всі свої паролі в одній базі даних, яка блокується за допомогою майстер-ключа. Тому вам потрібно запам'ятати лише один єдиний майстер-ключ, щоб розблокувати всю базу даних. Файли баз даних шифруються за допомогою найкращих і найбезпечніших алгоритмів шифрування, відомих на даний момент (AES-256, ChaCha20 та Twofish). Але ця система має також свої мінуси, тому що знаючи пароль до цієї системи, можна отримати доступ до всіх паролів користувача, тому зловмисники

часто намагаються шляхом різних підходів отримати цей пароль. На рисунку 1.1 зображено інтерфейс програми.

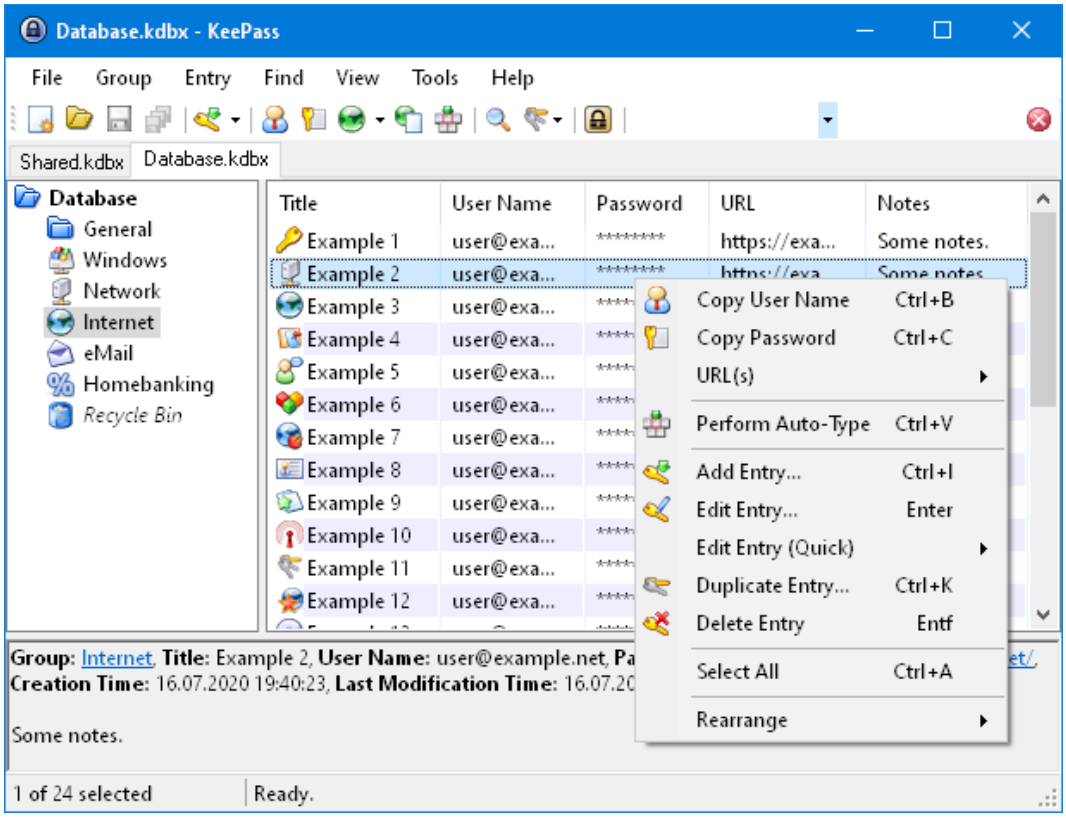
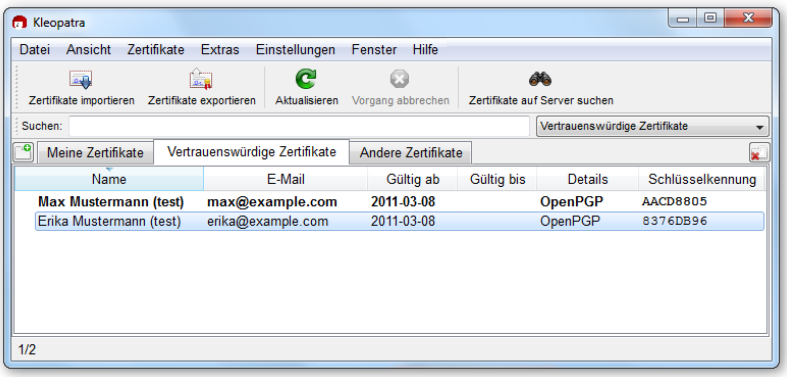


Рисунок 1.1 – Інтерфейс програми KeyPass

Themis є ще однією загальнодоступною бібліотекою. Це є основною бібліотекою для шифрування, вона використовується у багатьох мовах програмування та дозволяє захищати ваші дані. Щоб зрозуміти її інтеграції з іншими мовами програмування, можна подивитися на приклад Go. Дана мова програмування використовує цю бібліотеку уже вбудовану в саму мову та дозволяє нею користуватися з OpenSSL та BoringSSL. Головним алгоритмом в даній бібліотеці є застосування асиметричного алгоритму, це дозволяє обробляти та шифрувати дані за допомогою пари ключів, це публічний та приватний ключ та після певний реалізації вже розповсюджувати їх по різних користувачах, які повинні мати доступ до цієї інформації. Також потрібно розуміти, що асиметричний підхід має вплив на швидкодію при шифруванні даних, тому ця бібліотека може доставляти інші типи шифрування, які можна

використовувати при реалізації певних функціональностей по збереженню доступу до даних.

Ще однією доступною бібліотекою для шифрування даних є Kleopatra. Ця бібліотека була розроблена спеціальна під Windows та має найбільше значення ключа, а саме 4096 біт. Бібліотека є в публічному доступі та вважається open-source проектом, тобто вона тримається за рахунок добровольців, які займаються її підтримкою та додають новий функціонал в залежності від нових вимог до захисту даних. З огляду на це, зараз система має широкий спектр організації захисту даних та дозволяє створювати зручне покрокове генерування ключів, які потім можна буде прив'язати до багатьох компонентів операційної системи Windows або навіть до облікового запису. При встановленні програма додатково встановлює цілий пакет додаткових програм, які дозволяють не лише займатися шифруванням даних, але і організовують сховище згенерованих ключів. В даному сховищі можна перевіряти термін дії цих ключів та інформації стосовно них, для кого воно було видано, на який термін та яка ідентифікація ключа. З головних недоліків даного програмного забезпечення є обов'язковість встановлення всіх додаткових пакетів для того, щоб воно працювало. Окрім цього потрібно враховувати, що дане програмне забезпечення розрахована лише на використання в операційній системі Windows. Тому це унеможливлює використання даної програми на інших операційних систем, проте всі ці мінуси компенсується за рахунок широкого спектру функцій та гнучких інструментів, якими забезпечує дана програма користувача. На рисунку 1.2 зображено інтерфейс програми Kleopatra.



До програм у

Рисунок 1.2 – Інтерфейс програми Kleopatra

вільному доступі також можна віднести GEAT. Дана програма була створена також за рахунок добровольців та написана на мові Java. Це дозволяє використовувати GEAT на різних операційних системах, де встановлена Java віртуальна машина, що розширює область використання цієї програми.

Дана програма дозволяє обрати та згенерувати ключ, за допомогою якого потім будуть шифруватися дані, зручний інтерфейс дозволяє після генерації ключа обрати файл, який необхідно зашифрувати, або який необхідно розшифрувати. За рахунок цього функціоналу ним може легко користуватися навіть звичайний користувач, який не має знань про програмування, та може використовувати зашифровані файли для своїх особистих цілей та передачі файлів між іншими користувачами. Це програмне забезпечення також має свої мінуси у зв'язку з тим, що шифрування фотографії в даній програмі використовується за допомогою старих підходів, це займає тривалий час. Існує багато аналогів, які справляється набагато швидше з цим, також одним з недоліків даної програми є неможливість роботи на інших платформах, так як вона написана на Java і вимагає, щоб була встановлена віртуальна машина Java, без неї вона не буде працювати. Також після детального аналізу даної бібліотеки було визначено, що максимальний розмір ключа, який використовується для генерації, обмежено до 1024 біт. На рисунку 1.3 зображено інтерфейс програми

Рисунок 1.3 – Інтерфейс програми GEAT

GEAT.

Програма Криптософт Storage - це програма створена для забезпечення максимальний конфіденційності та цілісності всієї інформації за допомогою різних криптографічних перетворень, тобто виконання різних типів шифрування та розшифрування. Додатково відбувається здійснення керування доступом до даної інформації, що зберігається на носіях, таких як жорсткий диск комп'ютера, хмарні носії, флеш-накопичувач. У відповідності до сертифіката відкритого ключа дане програмне забезпечення працює у складі будь-якої інформаційної системи, дозволяє проводити контроль цілісності різного програмного забезпечення,та є розробкою компанії IT Engineering. Мінусом цього програмного забезпечення є те, що це платне програмне забезпечення, для того, щоб ним користуватися повністю, необхідно його купити. Також надається безкоштовно демо-версія, але ця версія включає обмежений пакет функціональності. На рисунку 1.4 зображено інтерфейс програми Криптософт

Рисунок 1.4 – Інтерфейс програми Криптософт Storage

Storage.

Програмне забезпечення для шифрування дисків, VeraCrypt. Програмне забезпечення розроблена для створення безпеки та конфіденційності усіх даних що знаходяться на дисках. Дане програмне забезпечення можете використувати розпаралелювання свого шифрування, тобто використовувати всі доступні ядра на процесорі для того, щоб пришвидшити роботу даного процесу. VeraCrypt включає в себе в себе різні алгоритми шифрування. На рисунку 1.5 зображено інтерфейс програми VeraCrypt.

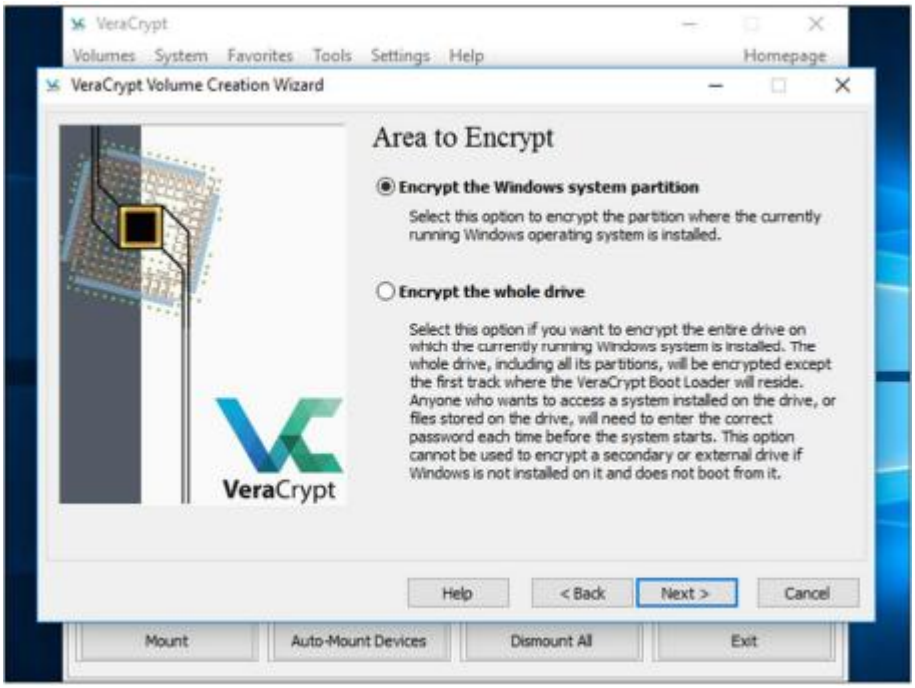


Рисунок 1.5 – Інтерфейс програми VeraCrypt

Саме VeraCrypt є головним підходом для шифрування даних, який користується популярністю серед користувачів операційної системи Windows.

1.3 Процес шифрування та дешифрування даних

Криптографія - це наука про математичні підходи, які дозволяють забезпечити автентифікацію, конфіденційність, та цілісність інформації. Початок криптографії почався з практичної необхідності передачі даних, які би були захищеними від зовнішнього втручання. Для математичного аналізу у криптографії використовуються такі підходи як математика, теорія імовірності та алгебра.

Та інформація, яку ми можемо прочитати, яку ми можемо осмислити без будь-яких додаткових інструментів, називається відкритим текстом або відкритою інформацію, а метод перетворення цієї відкрити інформації таким чином, щоб приховати його суть, називається шифрування. Шифрування відкриту інформацію дозволяє перетворити в незрозумілу форму. Ця незрозуміла форма називається шифротекстом. Шифрування дозволяє приховати інформацію від тих, хто не повинен нею володіти навіть попри те, що вони можуть побачити сам шифротекст.

Протилежний процес, який дозволяє перетворити шифротекст в зрозумілу інформацію, називається розшифруванням.

Раніше криптографія використовувалася для того, щоб забезпечити захист та конфіденційність самого повідомлення, роблячи та перетворюючи інформацію з повідомлення в незрозумілу форму для того, щоб ніхто не міг її прочитати, окрім отримувача. Це допомагало захистити інформацію від злочинців, які мали намір перехопити повідомлення та отримати необхідну інформацію.

Зараз в сучасному світі криптографія проснулася набагато вперед та використовуються в досить багатьох аспектах нашого життя. Вона дозволяє створювати електронні підписи, дозволяє перевіряти цілісність повідомлення, встановлювати безпекової з’єднання та визначати, хто повинен отримати повідомлення та хто його відправив. Зараз навіть зв’язок в Internet між

користувачем та сервером повинен бути захищений за допомогою шифра, що забезпечує захист від зловмисників.

Важливим аспектом будь-якого сервера може являтися база даних, дані якої повинні бути зашифровані, так як зараз кіберзлочинці завжди намагається достукатися саме до бази даних, тому що вона вміщує дорогоцінну інформацію, яку потім вони можуть використовувати для отримання вигоди. Бази даних можуть вміщувати в собі будь-яку інформацію, що стосується персональних даних, інформацію стосовно обліку цінних паперів та інформацію про компанію. Таким чином, шифрування цих даних є важливим аспектом кожного процесу створення бази даних та є вкрай обов'язковим.

Можна розглянути одну з найбільш розповсюджених проблем вразливостей. Розглянемо ситуацію, коли кібер злочинець може підміняти запити клієнта до сервера та відправляти замість них свої. В даному випадку, ми можемо взяти школу, в якій є своя система безпеки, пожежна система, температура приміщень, вологість. Вся ця інформація передається на сервер, за яким сидить оператор і у випадку виникнення критичної ситуації по цих датчиках, він виконує певні дії. Кіберзлочинець шляхом підміни інформації стосовно температури приміщень чи стану газової безпеки може відправити хибні сигнали на сервер і таким чином приховати дійсне становище речей. Також прикладом серйозних порушень може бути ваша сигналізація, яка має багато датчиків, закритих дверей, закритих вікон та всю цю інформацію відправляє на спеціальний хост, який її обробляє. Якщо зловмисники зможуть замість цих датчиків закритих дверей відправляти свою інформацію про те, що двері закриті, вони зможуть спокійно відкрити двері та проникнути в вашу квартиру. З огляду на це все є вкрай важливим шифрувати сигнали, які передаються між клієнтом та сервером, щоб уникнути ситуації, коли неавторизований користувач, тобто кіберзлочинність, буде відправляти якісь інші сигнали, які не є правдивими. Для того, щоб зробити усунення проблеми, необхідно використати криптографічні ключі при шифруванні сигналів.

Незважаючи на таку критичну необхідність шифрування даних, особливо в такій частині як бази даних, часто цей процес може ігноруватися зі сторони розробників, так як шифрування та дешифрування будь-якої частини операції, може сповільнювати швидкодію.

Не дивлячись на те, що всі доступи до бази даних можуть бути захищені, у разі відсутності шифрування самих даних може відбутися витік інформації за допомогою корумпованих IT спеціалістів, які можуть на цьому заробляти гроші.

До того ж, коли інформація в базі даних не зашифрована та пов’язана з фінансовими звітами, інтелектуальною власністю або інформацію стосовно запуска вашого проекту, ви можете отримати величезні збитки, так як ця інформація може бути отримана та використана в шкідливих цілях. Коли ж інформація буде зашифрована, то навіть отримавши її з бази даних, зловмисники не зможуть з нею скористатися, так як це буде набір незрозумілих символів без сенсу, який називається шифротекст.

Часто проблеми з безпекою виникають через некомпетентність працівників, які не дотримується стандартних кроків для забезпечення безпеки даних. Різні тестування, які проводяться на базі даних, та функціональні тестування лише перевіряють виконання завдань, вони не бачать виконання інших завдань, які можуть бути шкідливими, тому при розгортанні, для прикладу веб-сайту, необхідно проводити повну перевірку безпеки сайту та бази даних для того, щоб уникнути викрадення даних. Можливим підходом для викрадення даних можуть бути викрадені резервні копії баз даних. Загалом існує два види загроз для інформації, це зовнішня та внутрішня. Здебільшого IT компанії переймається за внутрішні загрози, так як можуть мати недовіру до своїх працівників та думати, що хтось з них може викрасти цю інформацію, але у гонитві за внутрішніми загрозами компанії часто забуваються за зовнішні. Часто щоб себе захистити компанії укладають контракти з своїми працівниками, для того щоб уникнути злиття даних. У разі все ж таки, коли працівник викрадає дані та це буде доведено, він повинен заплатити суму, яка буде відповідати

втраті компанії через цю ситуацію. Ця проблема вирішується суворим контролем за виконанням кроків шифрування даних та штрафами у разі відсутності дотримання всіх правил. Для прикладу, за правилами всі резервні копії баз даних повинні бути зашифровані найсуворішими методами шифрування. Якщо всі працівники будуть виконувати поставлені кроки, то навіть викрадення цих резервних копій ні до чого не призведе, так як копії будуть зашифровані. Також для уникнення проблем потрібне правильне керування доступом до інформації, а саме мати різні акаунти для різних типів доступу та не давати права для повного доступу усім користувачам. Для кожного користувача або команди людей необхідно мати окремий акаунт з доступом до інформації. Так у разі зникнення інформації можна буде відслідкувати останні дії конкретного користувача, які покажуть, хто саме це зробив, або ж додатково, ці користувачі можуть мати обмежені права, що унеможливить майбутнє викрадення інформації.

Правильний поділ обов’язків між адміністратором та користувачем буде гарантувати доступ до інформації лише для досвідченої команди. Таким чином користувачі, які будуть намагатися викрасти інформацію, отримають обмеження для того, щоб це зробити. Також чудовим підходом є обмеження кількості користувачів, які можуть працювати з базою даних. Таким чином кіберзлочинці отримують додаткові проблеми, коли будуть намагатися проникнути в базу даних.

Такий підхід використовується у бізнесах, які пов’язані з фінансами, так як фінансова інформація є досить конфіденційною та важливою для кожної системи і втручання в ці дані або їх отримання буде критично, тому важливо добре дбати про конфіденційність даних, давати правильні права доступу, для користувачів обмежити доступ, обмежити кількість людей, яка може мати доступ, та займатися шифруванням даних у відповідності до кроків, які встановлює безпека компанії.

Іншим типом проблеми можуть бути ключі шифрування. Необхідне правильне керування ключами шифрування, так як коли зловмисники розуміють,

що вони не можуть отримати доступ до бази даних, або коли вони навіть отримали дані, але вони є зашифрованими, зловмисники намагатимуться отримати доступ до ключів, які дозволять їм розшифрувати ці дані.

При поганому керуванню ключами шифрування вони можуть бути викрадені, так як часто ці ключі зберігаються на жорстких дисках, на робочих столах або інформація про ключове слово може бути просто записана на робочому місці працівника. Тому необхідно контролювати людей, які мають дані ключі, та проводити для них інструктаж, який буде розповідати, де необхідно ці ключі зберігати та в яких випадках ними потрібно користатися.

Цифрування даних - це спосіб, при якому інформація перетворюється в шефрований текст і може бути розшифрована до початкового вигляду лише за допомогою ключа. Тобто особа, яка володіє цим ключем, може розшифрувати отримані дані, тому є вкрай важливим тримати цей ключ в захисті, щоб зловмисники не отримала його. Без нього навіть викрадений зашифрований текст не буде мати жодного сенсу для зловмисників.

Процес схеми обробки даних показано на рисунку 1.6.

Рисунок 1.6 – Схема обробки даних

Процес перетворення зрозумілих даних в незрозумілі та позбавлені сенсу за допомогою математичних перетворень називається шифруванням. Дешифрування - це обернений процес, який займається тим, що перетворює зашифровані дані у початкові, зрозумілі. Алгоритми шифрування використовують математичні функції для шифрування чи дешифрування даних, таким чином в переважній більшості, використовується дві пов’язані функції, одна яка відповідає за шифрування та інша, яка відповідає за дешифрування.

У сучасних підходах шифрування також використовують спеціальні символи, так званий ключ, який додається до даних при шифрування, та необхідний для того, щоб ці дані розшифрувати. При правильному використанні ключа, процес дешифровки даних є доволі простим, коли у вас немає даного ключа, цей процес стає неможливим, так як потребує великої кількості операцій, які можуть розтягнутися на сотні років.

Шифрування даних використовується для запобігання доступу до важливої інформації та унеможливлює процес перехоплення даних. Шифрування даних є важливим аспектом у кожній архітектурі кібербезпеки та застосовується до різних потреб захищеності даних, починаючи від військових даних, секретних документів країн, до особистих персональних даних та кредитних карток. Для шифрування використовується програмні засоби, які в собі вміщують уже розроблені математичні алгоритми та схеми, які шифрують дані. Теоретично, це можна обійти тільки з великими обсягами обчислювальної потужності та доволі великим часом для вирахування встановлено ключа, цей час може тривати сотнями років. Шифрування використовується не лише в Internet для передачі даних, воно також може використовуватися в мобільних телефонах, в транзакціях банкоматів і навіть в передачі інформації про вашу сигналізацію. У всіх цих областях дані можуть бути перехоплені зловмисниками, але різниця полягає в тому, що навіть перехоплені дані не мають ніякої цінності, тому що вони не є зрозумілими.

1.4 Постановка задачі

Метою дослідження є розробка інформаційної технології шифрованої передачі даних між хмарними підпросторами.

Для досягнення поставленої мети, поставлені наступні **задачі**:

– дослідження необхідності використання шифрування даних для передачі їх між хмарними сервісами;

– розробка інформаційної системи передачі даних між хмарними підпросторами.

При цьому необхідно вирішити такі **підзадачі**:

– дослідження предметної області – різних підпросторів збереження даних;

– розробка алгоритмів та структури системи передачі даних;

– оцінка ризиків передачі шифрованих та нешифрованих даних;

– надання алгоритму використання ключів для шифрування;

– формування переліку програмних кодів як бібліотеки для використання іншими системами;

– надання рекомендаційних підходів для шифруванні даних;

– тестування розробленої системи.

Висновки до розділу 1

У даному розділі була отримана інформація стосовно методів шифрованої передачі даних, які використовуються у сьогоденні.

Стала зрозуміла необхідність у шифруванні даних для того, щоб забезпечити безпеку та надійність їх передачі. Отримані знання про методи та підходи, які використовуються при шифруванні та дешифруванні даних,

дозволяють краще розуміти сучасні вимоги до створення будь-якого програмного забезпечення, яке займається обміном даних.

Аналіз небезпек, які існують через відсутність зашифрованих каналів звязку, вказує на те, що правильний підбір алгоритмів шифрування та правильне створення структури доступу до даних дозволяє забезпечити коректну та безперебійну роботу кожної системи.

Розглянуте програмне забезпечення, яке дає можливість шифрувати дані, дозволяє використовувати вже наявні підходи для створення безпеки даних.

Розділ 2

Математичні моделі алгоритмів

2.1 Криптографічні системи

Криптографічні системи дозволяють зашифрувати всі передані дані. З даних носіїв для відновлення вихідних даних можна скористуватися ключами. Сучасні алгоритми шифрування виконують шифрування за допомогою одного або двох ключів. Дані ключі призначені для того, щоб займатися розшифруванням даних. У випадку коли використовується один ключ, ключ дозволяє шифрувати та дешифрувати дані, але при цьому, якщо хтось заволодіє цим ключем, він зможе повністю володіти всіма даними. У другому випадку, коли ми використовуємо два різних ключа, важливим є публічний ключ, який доступний для всіх, і кожен з них може ним скористатися. Але щоб розшифрувати дані, необхідно використовувати приватний ключ, який може бути доступний лише для однієї людини, це дозволяє розуміти, хто повинен бути кінцевим отримувачем цих даних та унеможливить процес розшифрування даних, так як ключ є лише в однієї особи.

Наразі є велика кількість методів шифрування, в основному вони поділяються на симетричні та асиметричні, в залежності від структури ключів, також методи шифрування можуть по-різному обробляти вхідні дані за допомогою шифрів, потокових шифрів та мати різну криптостійкість, тобто стійкість до злому ключа. Ключі шифрування та дешифрування можна порівнювати із звичайним паролем, який ви використовуєте для своєї емейл-скриньки або входу в телефон. Ключ шифрування генерується таким чином, щоб вони всі були унікальними за допомогою різних алгоритмів, тоді після перешифрування даних за допомогою ключа, дані будуть зашифровані у незрозуміло форму і також повернуті назад до оригінального тексту за допомогою цього ключа дешифратором.

Зазвичай ключ є двійковим або фактичним паролем, спосіб генерації ключа розповідає алгоритму, яким чином використовувалося шифрування даних, та що необхідно зробити для того, щоб його розшифрувати. Так як всі бібліотеки шифрування є загальнодоступними, будь-який злодій може, отримавши приватний ключ, розшифрувати ці дані, тому захист приватного ключа є головним аспектом у кібербезпеці.

У симетричній системі шифрування навпаки, якщо стане відомо хоча б один ключ, це буде означати, що зловмисник зможе і шифрувати, і дешифрувати дані. Приклад роботи симетричної криптосистеми показано на рисунку 2.1.

Рисунок 2.1 – Робота симетричної криптосистеми

Використовуючи шифрування симетричними ключами, відбувається мінімальний вплив на швидкодію систем, тобто користувач може навіть не замітити того, що процес отримав додаткові функції, такі як шифрування даних, так як для нього це не буде видно. Допоки ключі тримаються в секреті, це є швидким те безпечним способом комунікації, але як тільки один ключ стає відомим, вся ця комунікація може бути скомпрометована. Проте знаючи ключ, зловмисник зможе не лише розшифрувати дані, але й навпаки зашифрувати нові дані та відправляти їх на сервер, при цьому сервер буде думати, що це інший користувач, який має даний ключ.

Протоколом SSL зазвичай використовується шифрування з симетричними ключами, так як воно не впливає на швидкодію зв’язку. Також даний протокол використовується для авторизації користувача через мережу TCP/IP. Окрім

цього SSL використовує також і публічний ключ, тобто загалом ми можемо сказати, що в цьому протоколі використовується змішаний тип шифрування, це тоді коли комунікація відбувається за допомогою симетричного шифрування, а встановлення зв’язку та авторизація користувача за допомогою асиметрично шифрування, яке використовує публічний та приватний ключ.

У світі найчастіше використовують реалізації шифрування з використанням публічного ключа, який запатентований компанією RSA. Шифрування за допомогою публічного та приватного ключа, так зване асиметричне шифрування, стало відомим ще з середини 1970 років, коли почалися публічні дослідження асиметричних криптосистем. Після цього ці підходи стали широко доступними для комп’ютерних та всіх інших мереж і зараз їх можна використовувати для аутентифікації об’єкта, для створення свого підпису та аутентифікації електронної пошти.

Кожен публічний ключ може створити зашифроване повідомлення, яке зможе розшифрувати лише приватний ключ, публічний ключ може бути доступний для будь-кого, тобто будь-хто може зашифрувати для вас дані, але прочитати їх зможе лише та людина, яка має приватний ключ. Такі криптосистеми зазвичай використовується для створення приватного цифрового підпису. Це звичайне асиметричне шифрування можуть поєднувати з симетричним для того, щоб зробити менший вплив на швидкодію та забезпечити кращий захист даних.

На рисунку 2.2 зображено, як виглядає процес шифрування за допомогою асиметричного підходу.

Рисунок 2.2 – Робота симетричної криптосистеми

У порівнянні із симетричним шифруванням, асиметричне потребує більшого обсягу ресурсів для обчислення і тому може впливати на швидкодію певної системи, тому існує можливість використовувати комбінований підхід з симетричним ключем та публічним ключем асиметричного методу для кращої та швидшої взаємодії.

Зараз протокол SSL використовує саме цей змішаний підхід. Шифрування із використанням вашого приватного ключа допомагає аутентифікації вас як відправника повідомлення, тобто створити ваш приватний цифровий підпис, що є важливим для відправки електронних повідомлень, так для прикладу Firefox у майбутньому зможе перевіряти ваші електронні листи на наявність цього приватного ключа для того, щоб зрозуміти, що ці листи не були підроблені з моменту підписання.

Цифровий підпис є видом електронного підпису, який відбувається як результат криптографічного перетворення даних. Це дозволяє математичним методом підтвердити й ідентифікувати дані, які мають відправитись. Наприклад, коли ви відправляєте повідомлення, текст повідомлення не буде змінено, але до нього буде добавлена додатковий блок, який буде вміщувати цей приватний підпис, далі після отримання цього повідомлення його можна буде розділити на два окремих етапи. На першому етапі за допомогою математичних функцій та різного програмного забезпечення відбувається перевірка, яка називається відбитком та має такі властивості:

– фіксована довжина;

– унікальність відбитку;

– неможливість відновити повідомлення.

У разі зміни інформації цей відбиток буде оновлено і тому буде зрозуміло, чи підписаний він насправді вами.

На другому етапі вже інформація самого повідомлення буде шифруватися за допомогою математичних алгоритмів та персонально ключа, розшифрування цифрового підпису та отримання інформації стосовно її відправника можна лише за допомогою сертифіката відкритого ключа автора. Таким чином перевірка цих параметрів допомагає захистити документ від модифікації зловмисниками після підписання автора.

Процес пошуку ключа для того, щоб розшифрувати дані називається взломом алгоритма шифрування. Щоб здійснити взлом симетрично алгоритму необхідно знайти ключ, яким шифрувалися дані, а для взлому асиметрично алгоритму, який використовує публічний ключ, необхідно отримати секретну інформацію, яка передавалася між двома отримувачами.

При використанні симетричного алгоритму шифрування міцність шифрування визначається довжиною ключа, який використовується. Чим більше біт становить ключ, тим вище вважається рівень захищеності даних. Ключ шифрування можна вважати повністю захищеним, якщо метод атаки з метою злому не є швидшим за підбір кожного можливого ключа.

Зрозумівши процес злому алгоритма, ми можемо зрозуміти процес перехоплення та навіть розкриття важливої інформації.

Різні шифри можуть вимагати ключі різної довжини, тобто різного захисту. Для прикладу, симетричні ключі можуть мати довільно множину значень, проте як асиметричні ключі можуть мати лише підмножину значень. Це базується на проблемі математично підходу, який використовується при асиметричному шифруванні, тому для максимального захисту ваших даних рекомендовано використовувати шифрування змішане, яке може розділяти дані на різні частини та шифрувати їх різними алгоритмами. Тому, коли зловмисник навіть і

перехопить дані, він отримує лише частину тексту, яку буде важко розшифрувати.

2.2 Симетричні криптосистеми

Для кращого розуміння симетричних криптосистем, необхідно розглянути класична модель Шеннона, а саме систему, у якій є три наступні учасника, кожен з яких виконує своє завдання.

Перший учасник має намір за допомогою відкрито каналу зв’язку передати захищені дані. Дні будуть зашифровані, використовуючи ключ Y, та самі дані, які будуть позначені X. Учасник буде передавати ці дані у зашифрованому вигляді.

Другим учасником буде отримувач, завдання якого отримати переслане повідомлення та за допомогою свого ключа, розшифрувати його, щоб прочитати відправлений текст. Також передбачається, що другий учасник уже має власний ключ, який було отримано заздалегідь по каналу зв’язку.

Третій персонаж є аналогом злочинця, завданням якого є отримати дані, які передавалися, та видозмінити їх таким чином, щоб відправити отримувачу та мати з цього вигоду. Це демонструє класичний метод, коли зловмисники перехоплюють дані та замість них відправляють свої. Приклад таких дій було розглянуто у попередньому розділі, у прикладі з сигналізацією.

Для передачі даних буде використовуватися симетричний алгоритм. Симетричні алгоритми є досить популярними. Одним з найпопулярніших стандартів даного алгоритму є шифрування даних DAS, яке було розроблено компанією IBM.

Рисунок 2.3 – Робота симетричної криптосистеми

Приклад роботи алгоритму продемонстровано на рисунку 2.3.

Сам процес шифрування даних можна поділити на декілька кроків.

Першим кроком будем форматування вхідних даних. Передані вихідні дані необхідно перетворити в число за допомогою процедури та потім всі символи, які були перетворені в числа, необхідно розділити на 2 блоки, кожен з яких повинен становити 64 біта.

Саме ці два блока надалі будуть параметрами для функції шифрування та використовуватися для попередньої обробки.

Наступним кроком є попередня обробка. Головна задача попередньої обробки полягає в тому, що права частина обраховується логічною сумою по модулю два правого та лівого блоків, які передавалися як вхідні параметри для шифрування.

Роботу алгоритму зображено на рисунку 2.4.

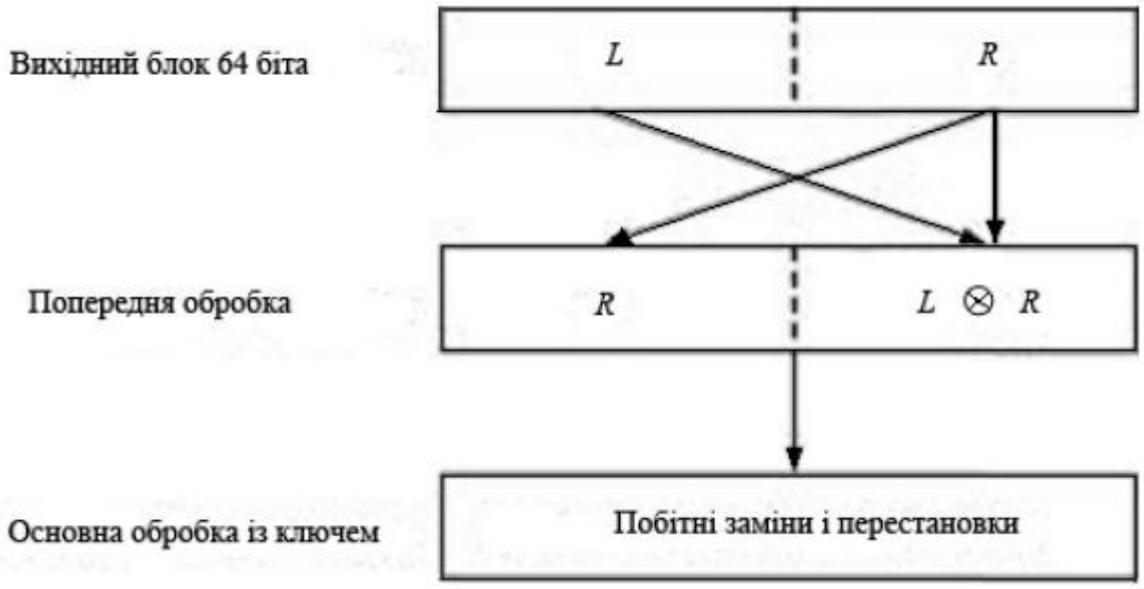


Рисунок 2.4 – алгоритм DES

Фінальним кроком є саме шифрування, тобто основна обробка із ключем. Обробка виконується у вигляді фіксованої послідовності довжиною 64 біта, з яких сам ключ буде становити лише 8 біт, а інші 56 будуть генеруватися випадковим чином. За допомогою такого підходу, відбувається організація двійкової послідовності та виконується побітна заміна і перестановка отриманих чисел.

Алгоритм шифрування DES є досить популярним для використання та є в різних продуктах, які вимагають безпеки даних. Додатково цей алгоритм може бути покращений за допомогою триразового шифрування та використанням двох різних ключів, але є і обмеження у такому підході, так як довжина ключа збільшується від 56 до 112 біт та потребує у три рази більше часу на виконання даних операції у порівнянні зі звичайним DES.

В даному підході, обмін секретними даними відбувається за принципом «кожен з кожним», тобто коли у системі є велика кількість користувачів, які обмінюються між собою зашифрованими даними. Це викликає потребу у великій кількості ключів та місці, де їх можна зберігати для відслідковування який ключ до якого користувача відноситься, що викликає за собою необхідність у додатковому сховищі, захищеному від зловмисників, яке буде містити дані ключі.

У сучасному світі ця проблема також вирішується за допомогою асиметричних криптосистем, що працюють за допомогою відкритих ключів, які стають доступними для всіх користувачів відразу.

2.3 Асиметричні криптосистеми

Симетричні системи використовують у функціях дешифрування та шифрування один і той самий ключ, тому такий підхід має недоліки:

– так як передається один і той самий ключ, він повинен передаватися через захищені канали для того, щоб бути впевненим, що ніхто інший не отримає даного ключа, крім того, хто повинен розшифрувати дані;

– також є серйозними вимоги до створення даних ключів, тому що потрібно розуміти, що для кількості користувачів кількість ключів є квадратною.

Для вирішення даних недоліків, системи, які генерують шифрування, можуть використовувати комбінацію з асиметричним шифруванням або шифруванням з відкритим ключем.

Асиметричні криптосистеми генерують унікальний відкритий ключ для дешифрування даних за рахунок закритого ключа, який не є доступним.

Головною ідеєю даного підходу є використання односторонньої функції, що дозволяє значно ускладнити процес взлому самого шифру. Роботу асиметричного алгоритму шифрування даних зображено на рисунку 2.5.

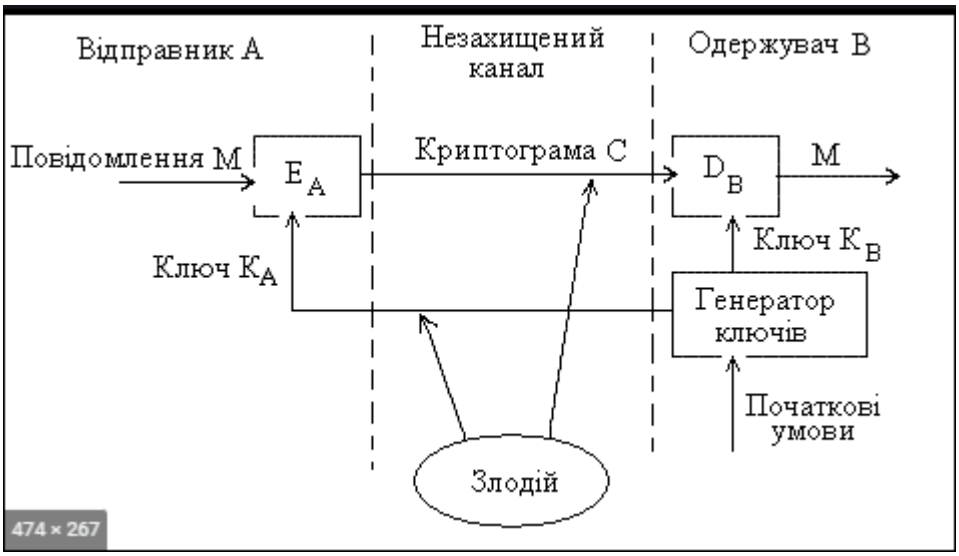


Рисунок 2.5 – Робота асиметричної криптосистеми

Дана система характеризується тим, що у кожного користувача є відкритий та закритий ключ. Саме відкритий ключ можна спокійно розповсюджувати між іншими людьми, так як без знання закрито ключа він немає ніякої цінності.

Даний підхід вирішує наступні проблеми:

– відсутність необхідності в захищених каналах або інших методах передачі закритого ключа, який може розшифрувати дані;
– відсутність квадратичної залежності між кількістю користувачів та кількістю ключів, які необхідні для цих користувачів.

Першим та наразі найпопулярнішим асиметричним алгоритмом є RSA, назва якого складається з прізвищ його винахідників.

Мінусом асиметричного шифрування є те, що процес дешифрування даних є досить складним, адже він за допомогою публічного ключа визначає закритим, що є доволі затратним в обчисленні, дешифруванням даних є пошук дільників натуральних чисел, які були виведені з добутка двох простих чисел, через це процедура визначення закритого ключа є часозатратною. Принцип роботи криптоалгоритму RSA зображено на рисунку 2.6.

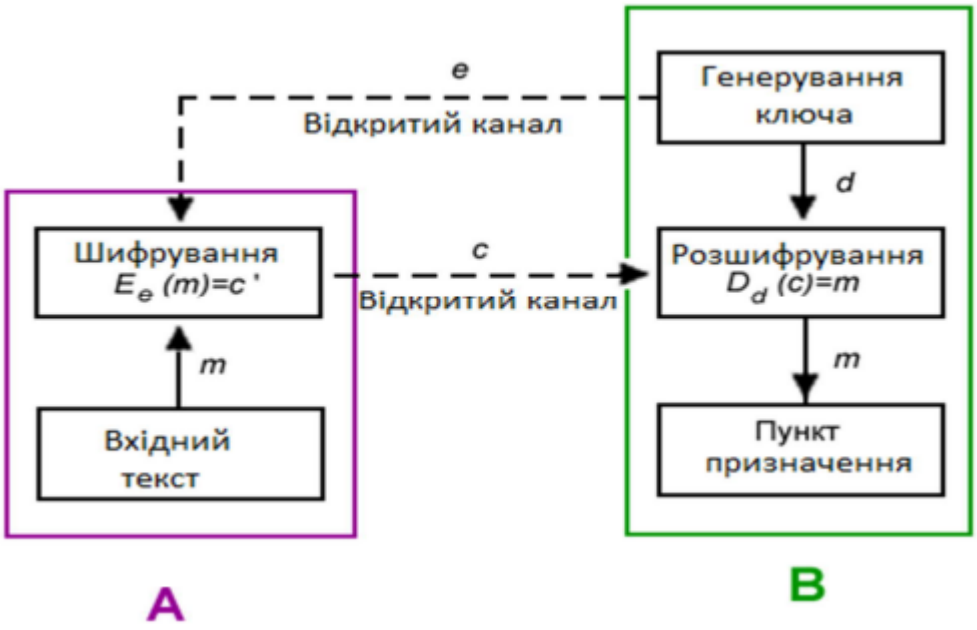


Рисунок 2.6 – Принцип роботи алгоритму RSA

Алгоритм RSA вже на протязі довго часу глибоко вивчався та перевірявся і правдиво визначено, що він є доволі стійким до взломів при наявній достатній довжині ключа. Цей алгоритм використовується у дуже багатьох системах, де

необхідне шифрування даних, де для забезпечення максимальної безпеки ці системи використовують 1024 бітний ключ, який вважається неможливо взламати.

Звичайно, в сучасному світі, потужності процесорів постійно зростають, що відкриває можливості для того, щоб можна було взламати ключ певним підбором значень, але не варто забувати про те, що збільшення можливості процесорів дозволяє збільшити довжину самого ключа, який обирається для шифрування даних.

Сам принцип роботи алгоритму шифрування зроблено таким чином, що він містить в собі процес як генерації ключів так і шифрування та дешифрування даних.

Для генерації ключів використовуються наступні дії:

– необхідно обрати два прості числа, назвемо їх p та q;

– далі треба визначити їх добуток n, де $n=p\times q$;

– далі обов’язковим є використання функції Ейлера, а саме $j(n)=(p-1)(q-1)$;

– обираємо ціле число e таке, щоб e було більше за одиницю, менше за результат функції Ейлера та взаємно просте з нею.

– на основі розширеного алгоритму Евкліда знаходиться число d таке, що $ed \pmod{f(n)}=1$ або $d=e^{-1} \pmod{f(n)}$.

Модулем називається число n, а числа e і d - відкритою та приватною експонентами відповідно. Відкритою частиною ключа є пара чисел (n,e), а пара (n, d) – секретною.

Для шифрування повідомлення ми можемо припустити, що користувачу необхідно відправити іншому повідомлення, для цього йому необхідно використати схему перетворення, тобто узгоджений протокол перетворення, де текст A, необхідно перетворити в число, а далі буде обчислюватися зашифрований текст, використовуючи відкритий ключ іншого користувача, який повинен отримати дане повідомлення.

Це може бути зроблено досить швидко, навіть у випадку, коли текст перевищує 500-бітну розрядність. Розшифрування виконається наступним чином:

$$a=(A')^d \bmod n. \tag{2.1}$$

Після використання даної операції відтворюється вихідне повідомлення:

$$(A')^d \equiv (a^e)^d \equiv a^{ed} \pmod{n}, \tag{2.2}$$

З умови

$$ed \equiv 1 \pmod{\varphi(n)}, \tag{2.3}$$

випливає твердження, що ed ≡ k0 j (n) + 1 для деякого цілого k0. Також необхідно добавити, що за теоремою Ейлера:

$$a^{\varphi(n)} \equiv 1 \pmod{n}, \tag{2.4}$$

Можна вивести наступну формулу:

$$a^{k0\varphi(n)+1} \equiv a \pmod{n}, (A')^d \equiv a \pmod{n}. \tag{2.5}$$

Результатом роботи шифрування є те, що нам необхідно знати таку пару чисел (e,n) та (d,n). Для шифрування перша пара буде презентувати собою відкритий ключ, в той час як друга пара - це закритий ключ.

Знаючи відкритий ключ, можна визначити і закритий ключ, це необхідно для того, щоб розшифрувати дані.

Для розшифровки даних використовується операція, яка займається тим, що знаходить множники. Для чого потрібно розкласти n і саме на виконання цієї процедури витрачається найбільше часу, саме з цією великою обчислювальною складністю і полягає криптостійкість, пов’язана з RSA.

Видатний вчений теоретик Міхаель Рабін розробив власну систему, яка була представником першої асиметричної криптосистеми. Система, що базувалася на важкості визначення квадратичного залишку.

Вона як і інші асиметричні системи використовувала пару відкритих та закритих ключів, де відкритий ключ використовували для того, щоб зашифрувати дані, де він міг бути доступний для всіх, та закритий ключ, для того, щоб могти розшифрувати ці дані.

Основною перевагою даної криптосистеми є те, що текст може бути розшифрований лише при умові, що шифрування здатне до активної факторизації відкритого ключа.

Саме це є доказом того, що система є стійкою до атак та відповідає підходу «все або нічого», алгоритм працює тільки тоді, коли відбувається повне розкладання цілого числа на прості множники, що є головним та важливим елементом.

Схема шифрування Міхаеля Рабіна розташовано на рисунку 2.7.

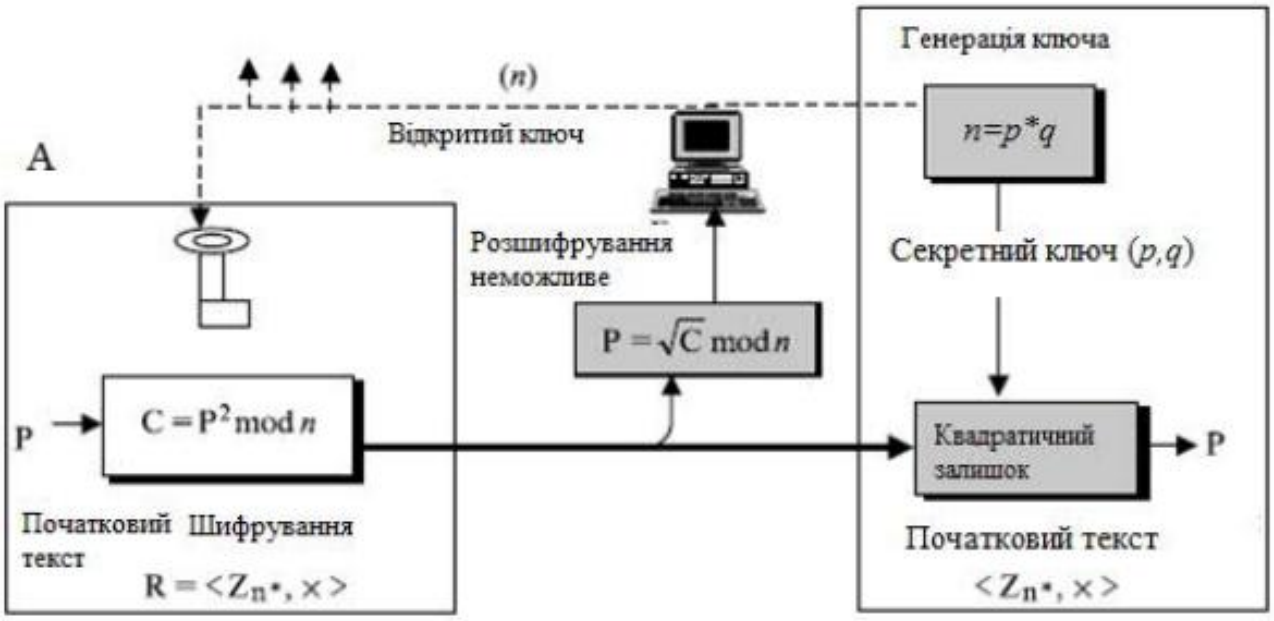


Рисунок 2.7 – Схема криптосистеми Міхаеля Рабіна

Для того, щоб згенерувати ключ, необхідно обрати два випадкових числа, які при цьому будуть великі та прості, p та q. Також необхідно дотримуватися умови де:

– числа повинні бути великими;
– числа повинні бути простими;
– повинна виконуватися умова: p ≡ q ≡ 3 mod 4.

Даний підхід дозволяє сильно пришвидшити процедуру знаходження ключа за модулем p та q. Далі відбувається процес схожий до симетрична шифрування, де необхідно знайти добуток цих чисел, число n буде відкритим ключем числа, а числа p і q, будуть закритим ключем.

Як результат, початкове повідомлення шифрується за допомогою відкритого числа та використовує таку формулу:

$$A' = A^2 \bmod n. \tag{2.6}$$

Саме використанням операції, яка підносить до квадрату за модулем ці числа, забезпечується покращення швидкодії даної системи Рабіна. Якщо її порівняти із швидкістю шифрування вже відомого метода RSA, то, навіть у найгіршому випадку, система шифрування Рабіна буде швидшою.

Також при шифруванні тексту для зручності можна вводити додаткові змінні а саме c1 і c2.

$$c_1 = {}^{\circ}A', \mod {}^{\circ}p,\ c_2 = {}^{\circ}A', \mod {}^{\circ}q, \tag{2.7}$$

Для визначення тексту необхідно знайти квадратичний залишок відповідно за модулями p та q.

$$x^2 \equiv c_1 (\mod p),\ y^2 \equiv c_2 (\mod q) \tag{2.8}$$

Як результат буде отримано 4 системи рівнянь:

$$\begin{cases} A_1 \equiv x(\mod p); \\ A_1 \equiv y(\mod q); \end{cases} \begin{cases} A_2 \equiv x(\mod p); \\ A_2 \equiv -y(\mod q); \end{cases} \begin{cases} A_3 \equiv -x(\mod p); \\ A_3 \equiv y(\mod q); \end{cases} \begin{cases} A_4 \equiv -x(\mod p); \\ A_4 \equiv -y(\mod q); \end{cases} \tag{2.9}$$

Саме один результат із цих систем, який ґрунтується на китайський теореми про залишки, і буде шуканим текстом.

Недоліком є те, що розшифрування, окрім правильного результату, дає ще три хибних результати, що впливає і на швидкодію, і є головною проблемою та незручністю використання системи Рабіна. Це є головною проблемою, яка стає на шляху розповсюдженого використання даної системи. Для вирішення цієї проблеми необхідно детально дослідити математичну модель, де виникають ці зайві дії та неправильні результати.

Необхідно розуміти, що якщо вихідний текст являє собою звичайне повідомлення, то тоді не буде важко визначити і текст, який був зашифрований. Але повідомлення може містити лише набір певних бітів, що використовувався при створенні ключа. Це буде означати, що визначення тексту, який було зашифровано, стає проблемою.

Для того, щоб вирішити цю проблему зі знаходженням тексту стало за основу перед самим текстом, який необхідно розширювати, додавати певний заголовок, який дозволить шифруванню зрозуміти, що саме з цього моменту починається реальний текст.

В класичній системи Міхаеля Рабіна блок доступного тексту обмежується величиною публічного ключа, тому коли відбувається відправка великих текстових повідомлень їх необхідно розділяти на блоки та шифрувати окремо.

Системою, яка володіє приблизно такою обчислювальною складністю як факторизація, є система Ель Гамаль. Вона включає в себе алгоритми просування та додатково алгоритми цифрового підпису.

У 1985 р. Ель Гамаль запропонував схему, яка була удосконаленим варіантом алгоритму Діффі Гельмана. Цей алгоритм використовувався для аутентифікації та шифрування криптосистем.

Кроки для шифрування використовувалася в такій послідовності:

1) генеруємо просте число p;

2) вибираємо ціле число g – що є первісним коренем p;

3) вибираємо випадкове ціле число a щоб відповідало 1<a<r ;

4) обчислюємо h=g a mod p;

5) відкритим ключем будуть три числа (p, g, h) і приватним ключем число a.

Текст повідомлення який повинен бути менше від числа p, шифрується таким чином:

1) вибирається сесійний ключ – випадкове ціле число r таке, що 1<r;

2) вираховуються числа c1=g r mod p і c2=h r A mod p.

Пара чисел (c1, c2) є шифротекстом. Головним недоліком криптосистеми Ель-Гамаля вважається те, що довжина шифротексту вдвічі більша від вихідного текста. Знаючи закритий ключ, вихідне повідомлення можна обчислити із шифртексту (c1, c2) за такою формулою:

$$A = c_2 (c_1^{-1})^a \bmod p \qquad (2.10)$$

Шифр Ель-Гамаля можна назвати шифром багатозначної зміни, так як туди вводиться випадкова величина r.

Через те, що величина r є випадковою, такий підхід можна назвати схемою імовірного шифрування. Через те, що це імовірний спосіб шифрування, який базується на випадкових величинах, можна вважати, що цей алгоритм є більш стійкий, оскільки він не має певного шаблонного процесу шифрування, що створює певні проблеми при намаганні взламати його.

Головним недоліком даного алгоритму є те, що у порівнянні із початковим текстом, зашифрований текст має два чи більше символів в схемі Гамаля. Необхідно постійно використовувати різні довжини для ключів при шифруванні повідомлень, тому що якщо використовувати однакові ключі для шифротексту, то можна буде легко обчислити початковий текст, якщо відомо модуль числа, в якому відбувалося шифрування.

Схема шифрування криптосистеми Ель-Гамаля зображено на рисунку 2.8.

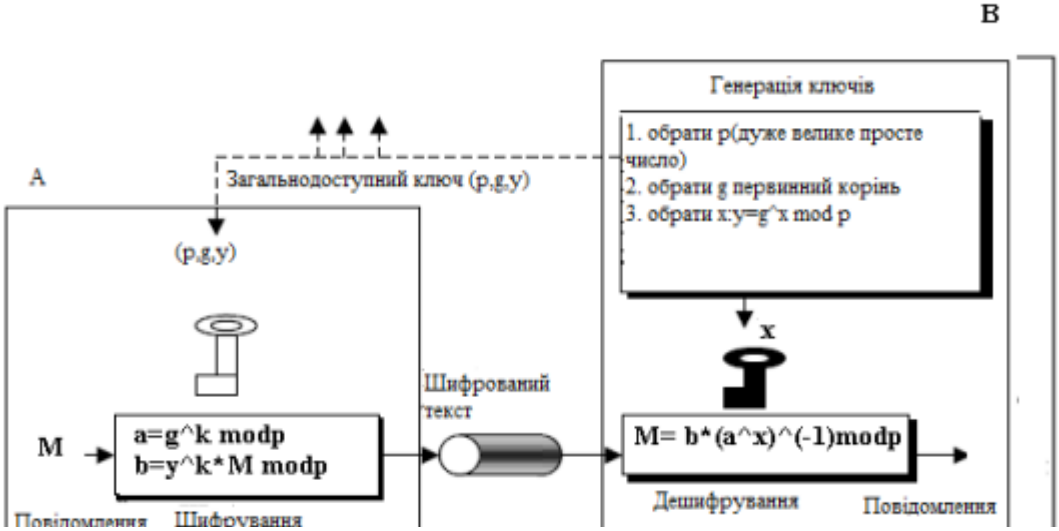


Рисунок 2.8 – Схема криптосистеми Ель-Гамаля

Проаналізовані 3 алгоритми виконують операції у двійковій або десятковій системі числення. Це ті системи, у яких відсутня можливість розпаралелення процесу обчислення, що впливає також на швидкодію шифрування даних.

Ідеальним співвідношенням шифрування тексту є комбінація з двох підходів з симетричного та асиметричного шифрування. Це дозволить працювати з великим набором текстів та покращити швидкодію його, але при цьому і буде забезпечувати великий рівень захисту та можливість працювати через публічний ключ. Наразі асиметричне шифрування є найбільш популярним, так як воно є найбільш захищеним та є простим в експлуатації, адже наявність публічного ключа, який можна розповсюджувати без захищених каналів та який може існувати один на велику кількість користувачів, є великою перевагою даного підходу.

2.4 Паралельні обчислення

Паралельне обчислення - це така організація процесу обчислень, при якій рішення завдання зводиться до обрахунку декількох окремих операцій в єдиний проміжок часу та за допомогою обчислювальних пристроїв в рамках однієї системи.

Доведено, що використання паралельних обчислень є доцільним лише у випадку, коли ефективність даних обчислень A >= 0.5. При цьому необхідно розуміти, що максимальне пришвидшення обчислень може складати від 2 до 10 разів.

До того ж, у разі виникнення падіння ефективності обрахунків, за допомогою різних спеціальних методів та систем можна визначити конкретне місце падіння швидкодії обчислення і це дозволить вирішити проблему та пришвидшити сам процес.

Сам процес паралельного обчислення виконується за допомогою двох способів. Перший спосіб - це чистий паралелізм, та інший спосіб - це конвеєризація. Є і винятки, тому що враховуючи кількість блоків з даними та кількість лінійних процедур, які використовуються при обчисленні складних задач, в алгоритмах заперечуваного шифрування навіть декомпозиція цих операцій на різні мікрооперації не принесе необхідного рівня прискорення обчислень. Схема паралельного обчислення зображено на рисунку 2.9.

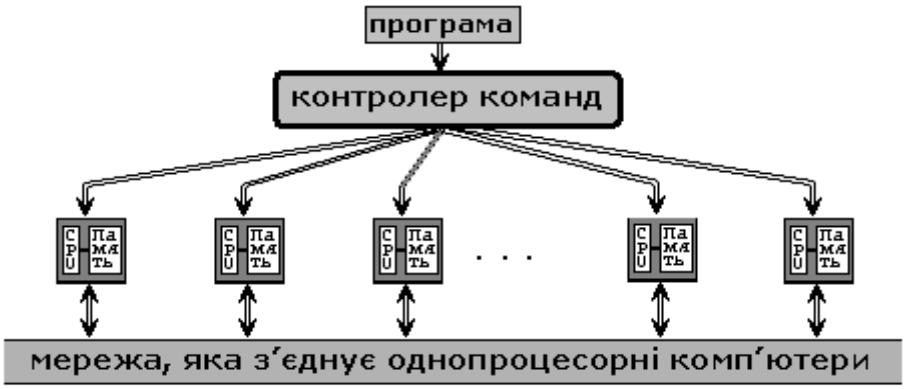


Рисунок 2.9 – Схема паралельного обчислення

Також слід зазначити, що для реалізації паралельного шифрування даних, використовуються методи, які дозволяють розділяти та склеювати дані для того, щоб виділити незалежні елементи даних для попередньої обробки. Цей метод можна зустріти в різних алгоритмах сортування, які займаються тим, що беруть загальний об’єм масиву, розділяють його на різні частини та співпрацюють з конкретними частинами у паралельних потоках, далі після виконання певних операцій вони поєднуються. Як результат, метод виділення незалежних елементів даних не потребує подальшого їх поділу на частини, оскільки після завершення їх обробки результати з кожного потоку будуть об’єднуватися у кінцеве значення.

Розподіл даних на блоки та організація вихідного тексту зображена на рисунку 2.10.

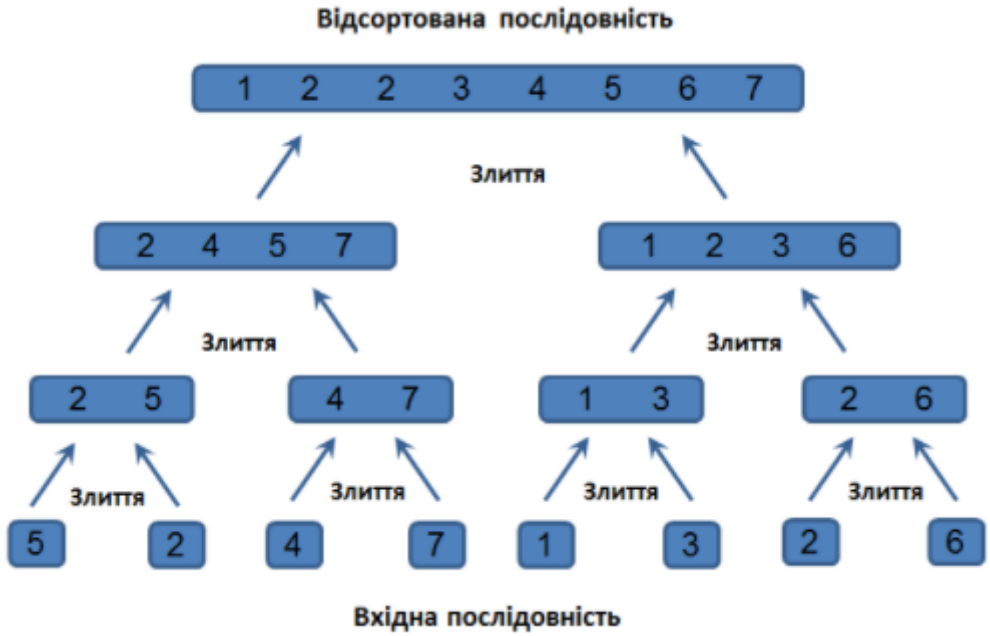


Рисунок 2.10 – Розподіл даних на блоки

Як і у випадку з алгоритми сортування, алгоритм шифрування передбачає попередню обробку даних та підготовку їх до такої структури, що дозволить найкраще обробити їх у паралельному режимі. При цьому операції необхідно розділяти на послідовні та паралельні, так як у разі наявності великої кількості послідовних операцій час, необхідний для виконання всіх обрахунків, може збільшуватися, а у паралельній послідовності можна втратити моменти, які необхідно було виконати перед фінальним об’єднанням даних.

Також і для пришвидшення обчислення даних при створенні шифрування використовується розподілена система.

Розподілена система - це та система, яка виконує обчислення без прив’язки до географічного розміщення вузлів, які повинні використовуватися при обчисленнях. Необхідно розуміти той момент, що реалізація даної системи є набагато складнішою, ніж її практичне застосування. Для того, щоб визначити чи дійсно необхідно її використовувати, необхідно перевірити даний випадок за допомогою наступних умов:

– дані, які необхідно зашифрувати, повинні знаходитися фізично в іншому географічному положенні або іншій мережі, відмінній від поточної системи;

– розуміння, що швидкість обчислень за допомогою засобів локальних вузлів неможливо збільшити.

Всі розподільні системи використовують обмін даними для комунікації вузлів через мережу інтернет або інші доступні мережі канали зв’язку. В основі цієї системи лежить модель «клієнт-сервер», яка є синхронна або асинхронна в залежності від складності та вимог системи. Як результат, було створено дві найбільш поширені моделі для розподілу даних:

– модель з поділом програмної логіки між вузлами;

– організація багатоланкової архітектури шини.

Для того, щоб попереджувати відмови розподілених систем забезпечувати зростання ефективності розподілу та виконання усіх обчислень, які необхідні для шифрування даних, реалізується шлях балансування навантаження на мережі занального транспортну та прикладного рівня. У даному розділі була отримана інформація стосовно методів шифрування і передачі даних, які використовуються на сьогодні.

Стала зрозуміла необхідність у шифрування даних для того, щоб забезпечити безпеку та надійність їх передачі. Отримані знання про методи та підходи, які використовуються при шифруванні та дешифруванні даних, дозволяють краще розуміти сучасні вимоги до створення будь-якого програмного забезпечення, яке займається обміном даних.

Аналіз небезпек, які існують через відсутність зашифрованих каналів, вказує на те, що правильний підбір алгоритмів шифрування та правильне створення структури і доступу до даних дозволяє забезпечити правильну та безперебійну роботу кожної системи.

Розглянуте програмне забезпечення, яке дає можливість шифрувати дані, дозволяє використовувати вже наявні підходи для створення безпеки даних.

Схема комунікації клієнт-сервер зображена на рисунку 2.11.

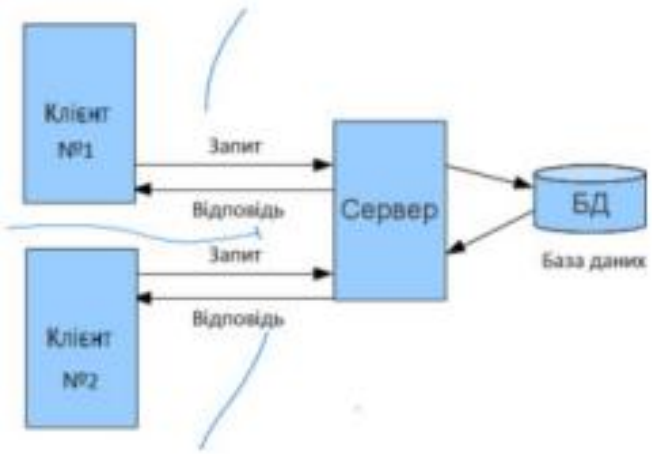


Рисунок 2.11 – Схема комунікації «клієнт-сервер»

Також використовуються відповідні протоколи для вирівнювання навантаження DNS, IP-адреса, Elastic Load Balancer, Nginx, HAProxy, NLB-кластер, LVS, Google Compute Engine, CloudStack, Least Connections, Sticky Sessions.

Висновки до розділу 2

Було отримано знання стосовно математичних моделей алгоритмів шифрування даних. Детальний розбір математичної моделі декількох алгоритмів дозволяє мати уявлення про процес шифрування та дешифрування даних. Була визначена різниця між симетричними та асиметричними криптосистемами, що дозволяє в майбутньому краще підібрати шлях для збереження цілісності та безпеки даних.

Розуміння математичної моделі при створенні шифрування дозволяє її покращити та додатково створити своє бачення того, як необхідно розробляти систему шифрування та, у випадку атак, розробити систему найбільш стійкою до них.

Розглянуті схеми найбільш захищених та популярних криптосистем дозволяють мати уявлення про сучасні вимоги та тенденції у створенні шифруванні та передачі даних, що позитивно впливає на риторику самої теми та дозволяє мати уявлення про те, як можна розвинути ці схеми.

Знання про паралельні обчислення дозволить максимально пришвидшити процес обчислення та шифрування даних, наскільки це можливо, а також розуміти аспекти, на які необхідно звернути увагу.

Розділ 3
Розробка системи шифрованої передачі даних

3.1 Опис структури та інтерфейсу

Система шифрованої передачі даних представлена у вигляді зовнішнього інтерфейсу бібліотеки.

Цей підхід передбачає використання системи у вигляді підключеної бібліотеки до проекту. Саме реалізація системи у вигляді бібліотеки дозволяє забезпечити зручне підключення даної системи та дозволяє зберегти функціональність самої бібліотеки від зовнішнього втручання.

Сама система шифрування складається з чотирьох головних компонентів, кожен з яких має свою ключову роль:

1). «API»: головний компонент, який об’єднує всі інші компоненти системи. Ціль даного модуля є відображення точки входу, яка дозволяє комунікувати з функціональністю бібліотеки, іншими словами, це так званий патерн «Міст», який дозволяє створити загально точку входа, а саме інтерфейс, та за допомогою нього комунікувати з усім іншим функціоналом бібліотеки. За допомогою цього модуля відбувається зв’язок між проектом користувача, який імплементував дану систему, та функціональністю самої бібліотеки. Цей підхід дозволяє приховати недоступну для користувача функціональність та використовувати різні інтерфейси в залежності від мови програмування та дозволеної для використання функціональності. Принцип роботи API зображено на рисунку 3.1.

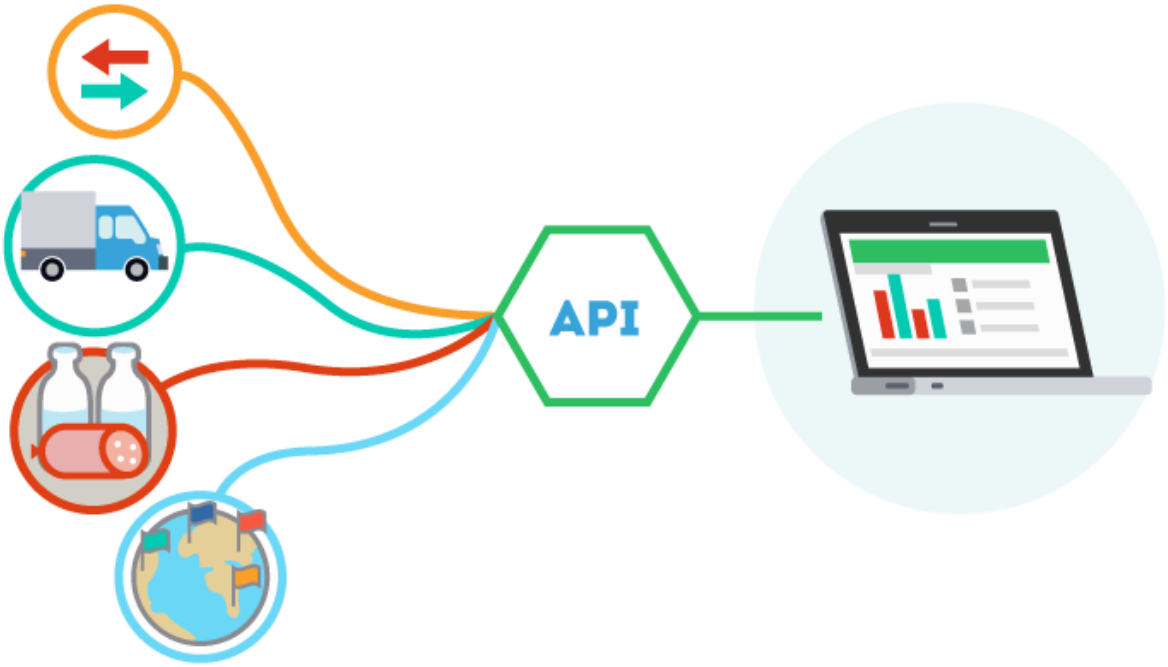


Рисунок 3.1 – Принцип роботи модуля «API»

2). «Модуль алгоритмів»: ідея даного модуля полягає в тому, що він містить список доступних алгоритмів, які будуть вибиратися для шифрування даних. Цей модуль нараховує всі можливі алгоритми шифрування та дешифрування для того, щоб інша функціональність могла з ним співпрацювати та обирати необхідний алгоритм для використання, а також перевіряти чи він є доступний у даній версії бібліотеки. Модуль алгоритмів зображено на рисунку 3.2.

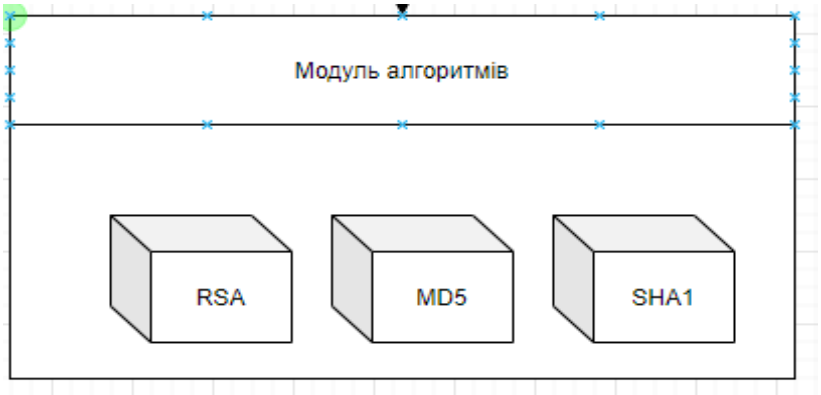


Рисунок 3.2 –Модуль алгоритмів

3). «Модуль бізнес-логіки»: даний модуль містить в собі всю бізнес логіку пов’язану з роботою бібліотеки, перевірку вихідних параметрів та перевірку їх відповідності шаблону. Модуль містить базову бізнес логіку, яка визначає, який саме алгоритми шифрування необхідно використовувати для даного типу даних та саме він займається підбором даного алгоритму. Також вона перевіряє розмір об’єкта, перевіряє його цілісність та назначає головний процес шифрування/дешифрування даних. У майбутньому модуль можна з легкістю розширити для додавання нового функціоналу, пов’язаного з шифруванням та дешифруванням даних.

4). «Інтернет модуль»: займається перевіркою версії алгоритмів шифрування, які знаходяться в бібліотеці. Так за допомогою цього модуля відбувається перевірка, чи є сама остання версія шифрування в даній бібліотеці чи вже наявна інша версія цього алгоритма, що дозволить тримати алгоритми шифрування завжди актуальними та використовувати саму останню версію даного алгоритму.

Загальна структура зв’язків, зображена на рисунку 3.3.

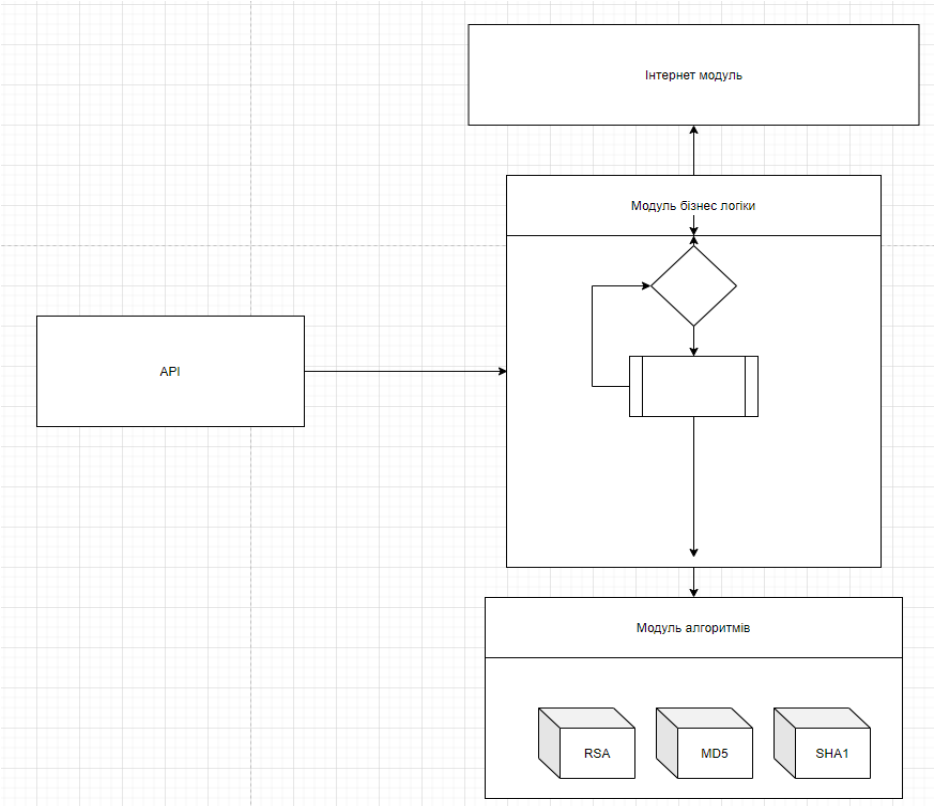


Рисунок 3.3 – Загальна структура зв’язків

Вихідними даними в залежності від вхідних параметрів є певний текст. При передачі зашифровано тексту, повинен повернутися розшифрований тест і при передачі звичайних даних, повертається зашифрований текст. Вся ця комунікація відбувається за допомогою модуля «API».

Вхідні параметри повинні містити в собі інформацію стосовно бажаного шифрування. У разі відсутності цієї інформації, система автоматично визначає тип шифрування в залежності від вхідних параметрів. Вихідними параметрами можуть бути будь-які об’єкти та структури, що несуть в собі інформацію. Далі в залежності від конфігурацій, які передаються додатковими параметрами, буде працювати логіка самої бібліотек.

При отриманні даних вони передаються на «модуль бізнес-логіки», який з урахуванням різних параметрів буде визначати, який алгоритм шифрування або дешифрування використовувати до отриманих даних.

Для підбору алгоритма штрафування система використовує наступні умови:

– яка має бути операція (шифрування / дешифрування);
– присутність бажаного алгоритми для виконання операції, у разі відсутності система сама визначає, який алгоритм використовувати;
– перевірка присутності самого ключа дешифрування або шифрування;
– пошук ключа зі свого локального сховища;
– відбувається перевірка розміру об’єкту;
– визначення типу об’єкта;
– актуалізація версії бібліотеки шифрування для використання;
– шифрування або дешифрування даних.

3.2 Вибір оптимального режиму шифрування

Вибір оптимального режиму шифрування є ключовим моментом у роботі бібліотеки для шифрування та дешифрування даних.

Бібліотека налічує декілька алгоритмів шифрування, що дозволяє вибрати найоптимальніший з них для конкретної ситуації.

Шляхом проходження вказаних перевірок відбувається відбір серед алгоритмів, які найкраще підійдуть до певної системи та певного типу об’єктів.

Для підбору бібліотека використовує список алгоритмів, при виборі алгоритма необхідно враховувати всі його особливості застосування та випадки, де його необхідно використовувати. Далі наведено список алгоритмів, які використовуються в даній бібліотеці, та особливості кожного з них:

1). MD5 - 128- бітний алгоритм хешування. Призначений для створення «відбитків» або дайджестів повідомлення довільної довжини та наступної перевірки їх автентичності. Схема роботи MD5 зображено на рисунку 3.4.

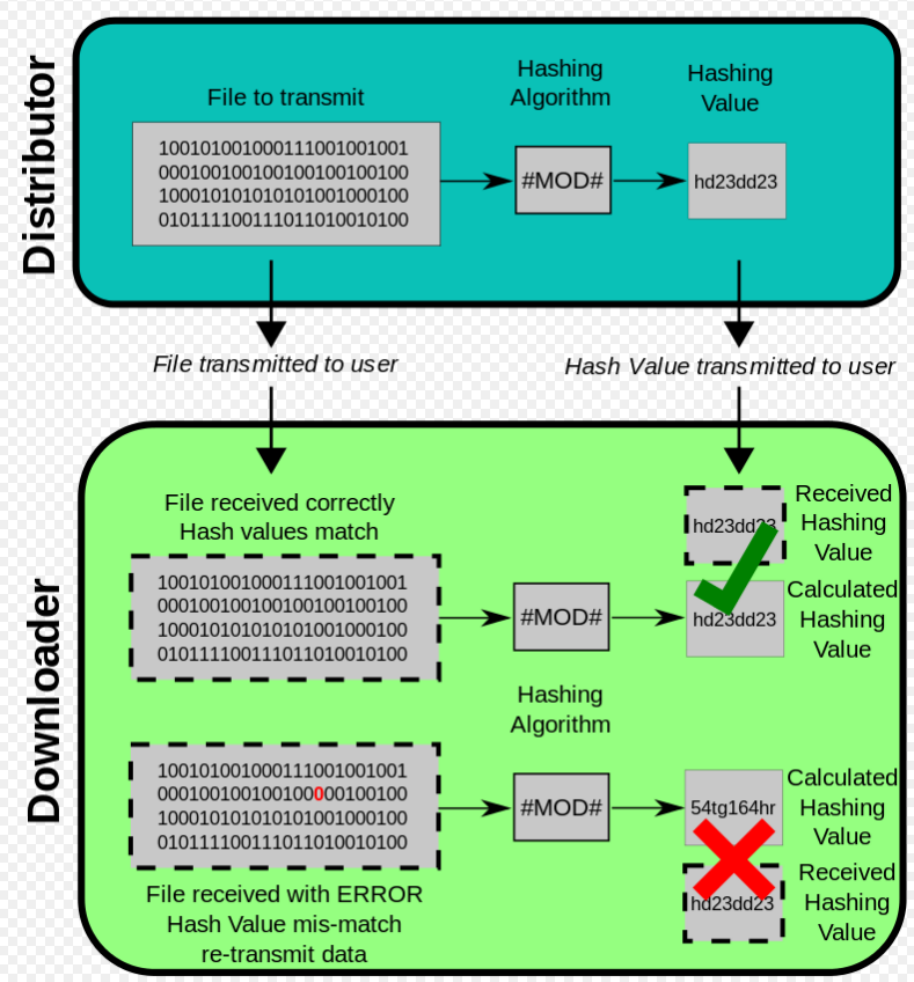


Рисунок 3.4 – Схема роботи MD5

2). CRC32 - алгоритм знаходження контрольної суми, призначений для перевірки цілісності даних. CRC є практичним додатком завадостійкого кодування, заснованим на певних математичних властивостях циклічного коду. Алгоритм працює за рахунок зіставлення двійкового многочлена бітів повідомлення. Формула зіставлення двійкових многочленів:

$$\sum_{n=0}^{N-1} a_n x^n \quad (3.1)$$

3). SHA1 - алгоритм криптографічного хешування. Описаний в RFC 3174. Для вхідного повідомлення довільної довжини (максимум 2^{64} - 1 біт , алгоритм генерує 160-бітове хеш-значення, яке називається також дайджестом повідомлення. Використовується в багатьох криптографічних додатках і протоколах. Також рекомендований як основний для державних установ у США. За основу шифрвання SHA1 використовує чотири нелінійні операції та чотири константи, зображені на рисунку 3.5.

$F_t(m,l,k) = (m \wedge l) \vee (\neg m \wedge k)$	K_t = 0x5A827999	0≤t≤19
$F_t(m,l,k) = m \oplus l \oplus k$	K_t = 0x6ED9EBA1	20≤t≤39
$F_t(m,l,k) = (m \wedge l) \vee (m \wedge k) \vee (l \wedge k)$	K_t = 0x8F1BBCDC	40≤t≤59
$F_t(m,l,k) = m \oplus l \oplus k$	K_t = 0xCA62C1D6	60≤t≤79

Рисунок 3.5 – Нелінійні операції та константи SHA1

Для визначення необхідного алгоритму потрібно розуміти їхню різницю між собою. Для порівняння береться два алгоритма, які мають однакове походження, а саме вони є покращеною версією MD4 алгоритма шифрування, хоча дивлячись на те, що вони мають однакове походження, їх необхідно використовувати в різних випадках. Коли ми говоримо про швидкодію виконання алгоритм шифрування, RSA1 є більш стійким до атак та краще шифрує дані, але його головним недоліком є те, що довжина дайджест даного алгоритму є більшою, ніж довжина дайджест алгоритму MD5. За рахунок цього, при шифруванні даних, алгоритм RSA1 виконує більше операцій, ніж алгоритм MD5. Як наслідок, алгоритм RSA1 буде шифрувати дані приблизно на 25 відсотків повільніше, ніж це буде робити аналогічний MD5. Як результат, коли

ми опрацьовуємо дані середнього або великого об’єму необхідно використовувати MD5, тому що швидкодія на 25 відсотків краще. Має велике значення, коли ми говоримо про великий об’єм файлів. Саме цей алгоритм шифрування дозволить зберегти багато часу та при цьому надійно захищає дані від атак.

Коли ми говоримо про об’єкти, які мають малий об’єм, тоді ми можемо використовувати алгоритм RSA1, тому що на малому об’єм різниця між використанням буде несуттєвою, але даний алгоритм є надійнішим за MD5.

CRC32 часто використовуються у вбудованих системах в якості захисту від випадкового пошкодження даних, але не зупиняє атаки на зашифровані тексти, які передавалися каналами зв'язку.

CRC32 не є алгоритм шифрування, але є дуже важливою частиною для перевірки даних, які були отримані для перевірки, чи вони не були змінені під час передачі.

Приклад місця, де його можна використовувати, це перевірка даних при ініціалізації системи для захисту від пошкодження прошивки.Також CRC можна використовувати і при перевірці можливого ушкодження програми або її невдалого завантаження.

Головною перевагою даного алгоритма є те, що він є максимально швидким та не використовує велику кількість операцій, яку використовують алгоритми шифрування.

Як результат, використання даного підходу є найкращим варіантом, коли нам необхідно лише перевірити дані, перевірити, що дані є цілісними та не були змінені після передачі. Це є важливим елементом у спілкуванні між хмарними сервісами, адже запобігає опрацюванню неправильних, змінених або пошкоджених даних.

Ключовим елементом бібліотеки шифрованої передачі даних є «Модуль бізнес-логіки», який відповідає за повну функціональність даної бібліотеки.

Головною функцією цього модуля є визначення алгоритмів шифрування або дешифрування, які використовуються після отримання вихідних параметрів за допомогою модуля «API».

Після отримання вхідних параметрів, таких як об’єкт, тип операції виконання, а саме шифрування чи дешифрування, необхідність використання останньої версії бібліотеки роботи в режимі онлайн чи офлайн, а також необов’язковий параметр на виконання спеціального алгоритму.

Після отримання всіх вхідних параметрів бібліотека починає виконання наступних операції:

1). Перш за все визначається, до якої категорії відноситься даний об’єкт за рахунок його об’єму. До якої категорії відноситься об’єкт, малий, середній або великий об’єм файлу встановлюється за рахунок глобальної змінної, яка є в конфігураціях бібліотеки. Дані конфігурації є обов’язковими при підключенні бібліотеки, це дозволяє налаштувати систему шифрування та дешифрування даних в залежності від потреб самого користувача, який встановлює дану систему;

2). Після встановлення, до якої категорії належить файл, відбувається наступний елемент, а саме перевірка алгоритмів, які можуть задовольнити операції по даній категорії файлів. Якщо це середня та велика категорія, відбувається вибір між алгоритмам шифрування MD5 та алгоритмом перевірки CRC32. В залежності від вихідних параметрів буде обрано конкретний алгоритм шифрування. Якщо файл належить до категорії об’єктів з малим об’ємом, буде відбуватися вибір алгоритму шифрування між CRC32 або SHA1. В залежності від вхідного параметра буде описано, якщо об’єкт повинен бути зашифрований або шифрований у надійний алгоритм шифрування, тоді буде використовуватися алгоритми шифрування SHA1, якщо ж необхідно просто перевірити цілісність даного файлу, буде використовуватись алгоритм CRC32;

3). Наступним елементом після вибору алгоритму шифрування та операцій, які необхідно провести з даними, відбувається процес актуалізації

бібліотеки в залежності від конфігурації бібліотеки, буде отримано дані чи необхідно актуалізувати версію бібліотек шифрування чи робота системи буде знаходитися в режимі офлайн. Якщо відбувається актуалізація бібліотек, система зв’язується за допомогою мережі інтернет з носієм бібліотек алгоритмів шифрування та актуалізує дані, а саме, робить запит на отримання останньої версії алгоритмів шифрування. Якщо він відповідає тій версії, яка знаходиться в самій системі, відбувається операція над наданим об’єктом, якщо версія відрізняється, тоді відбувається автоматичне скачування останньої бібліотеки та використання її при операціях з об’єктом.

Після виконання всіх вище зазначених операцій відбувається формування фінальної відповіді, яка буде повертатися за допомогою модуля «API». У фінальній відповіді буде згенеровано об’єкт, якийсь буде містити інформацію про тип операцій, використаний алгоритм для цієї операції та безпосередньо сам текст який буде результатом виконаної операції в залежності від запиту до бібліотеки.

3.3 Розподіл вхідних даних на блоки

Сучасні симетричні алгоритми шифрування забезпечують високий рівень обчислювальної стійкості та швидкості перетворення даних (від 2,6-3,9 Мбіт/с та вище). Досягнення останнього показника виконується за допомогою використання простих базових механізмів перетворення даних: оператори перестановки (P-boxes), оператори підстановки (S-boxes), сумування за модулем 2 (XOR), циклічний зсув, заміна, розкладання та об'єднання блоків. Їх коректне застосування дозволяє виконувати вимоги з розсіювання інформації та перемішування інформації, які висуваються до шифрів даного класу. Крім того, комбінування вказаних елементів дозволяє збільшити стійкість шифрів до атак з мінімальним навантаженням на обчислювальну систему. На відміну від P-boxes, S-boxes, XOR, циклічного зсуву та заміни, розкладання та об'єднання блоків є

універсальними механізмами для обробки масивів даних будь-якого розміру. Застосування елементів розкладання та об'єднання інформації дозволяє зменшити кінцевий розмір блоків з даними і збільшити швидкість їх перетворення. Окрім того, за вказаних обставин з’являється можливість використання інших елементів блокового шифрування (незалежно від операцій,

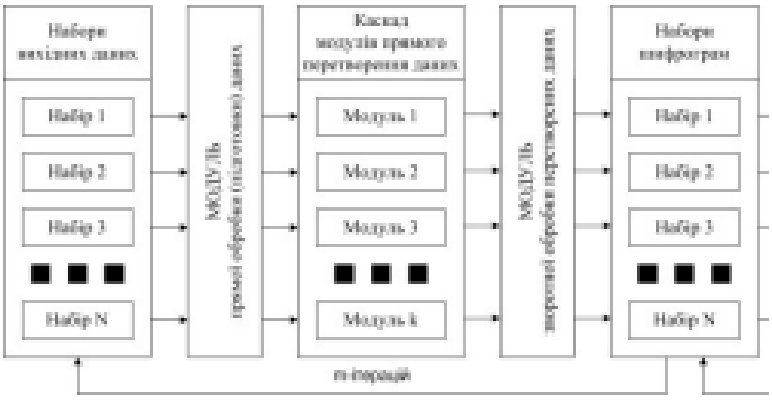


Рисунок 3.6 – Нелінійні операції та константи SHA1

які лежать в основі алгоритму перетворення даних). Вказане твердження стало основою для створення блокової моделі перетворення даних.

Під паралельними обчисленнями розуміють таку організацію процесу обчислень, при якому вирішення завдання зводиться до виконання декількох окремих операцій в єдиний проміжок часу за допомогою кількох обчислювальних пристроїв в рамках однієї обчислювальної системи. Встановлено, що можливість використання паралельних обчислень є доцільною лише у випадку коли ефективність обчислень після їх впровадження 40 становить $E \geq 0,5$. При цьому максимально можливе прискорення обчислень може складати від 2 до 10 разів. Разом з тим, у разі виникнення проблем з ефективністю технології паралельних обчислень, використання методів дозволяє виконати аналіз методики обчислень та знайти місце падіння продуктивності. Реалізація паралельних обчислень виконується в 2 способи: чистий паралелізм і конвеєризація. Враховуючи кількість блоків з даними та лінійність процедур, які

реалізують обчислення складних задач в алгоритмах заперечуваного шифрування, декомпозиція процедур на мікрооперації не принесе необхідного рівня прискорення обчислень. Також, зазначається, що для реалізації паралельної обробки даних використовують методи: розділення та склеювання даних, виділення незалежних елементів даних, попередньої обробки даних. Метод розділення та склеювання даних передбачає поділ масиву даних на частини з наступної обробкою в паралельному режимі (кількість частин відповідає кількості обчислювачів системи). При цьому скорочення часу обчислень можливо досягти лише за виконання однієї з умов: результат обробки частини даних є відомий на час початку обчислень, розмір блоку даних є незначним по відношенню до загального розміру масиву даних. Метод виділення незалежних елементів даних не потребує поділу даних на частини, оскільки по завершенню їх обробки результати кожного потоку об’єднуються у кінцеве значення. Метод попередньої обробки даних ґрунтується на підготовці масиву інформації до вигляду, який найбільше підходить для обробки у паралельному режимі. При цьому операції поділяються на послідовні та паралельні. У разі наявності значної кількості послідовних операцій, час необхідний для їх обробки $Tp \rightarrow \infty$, як у монопоточному режимі, або може бути гіршим.

Висновки до розділу 3

Отримавши всі необхідні знання стосовно алгоритмів, які використовується у системі шифрування та дешифрування даних, було визначено ряд критеріїв для відбору найкращого алгоритму в залежності від вихідних параметрів.

Даний підхід дозволив опрацьовувати вхідні параметри за рахунок найшвидшого алгоритму, який найкраще підходить для шифрування та дешифрування даних.

Реалізація даної системи допомогла зрозуміти різницю між алгоритмами, створити структуру, яка буде отримувати вхідні параметри та налаштовувати конфігурації для подальшої обробки даних, а також формування відповіді, яка буде доцільною для великої категорій користувачів даної системи.

Розуміння параметрів, необхідних для перевірки вхідного об’єкта, та його опрацювання, дозволяє згенерувати звітність та розуміння іншим розробникам, як саме користуватися даною бібліотекою та які конфігурації застосовувати для її роботи.

Отримані знання стосовно роботи алгоритмів, дані алгоритми та підходи надалі можна буде використовувати для побудови інших систем, які будуть спілкуватися між собою за рахунок закритих каналів зв’язку, що буде вимагати використання алгоритмів шифрування та розуміння побудови системи зв’язків між сервісами, щоб уникнути атак на дані та захистити їх.

Розділ 4

Дослідження ефективності метода шифрованої передачі даних між хмарними підпросторами

4.1 Системи підбору алгоритму для опрацювання даних

Для надання можливості вибору алгоритму шифрування та дешифрування при використанні системи застосовується модуль логіки опрацювання вхідних даних.

Спеціальна enum під назвою AlgorithmType, яка містить в собі список алгоритмів, необхідних для шифрування або дешифрування даних. Також цей об’єкт є параметром для класу RequestEntity та дає можливість будь-якому користувачу при передачі даних обрати алгоритм для виконання операції. У випадку, якщо алгоритм не буде зазначено, система автоматично визначить його на базі критеріїв, які прописані в сервісі. Реалізація даного об’єкту зображено на

```
public enum AlgorithmType {
    MD5,
    SHA1,
    CRC32
}
```

Рисунок 4.1 – Реалізація AlgorithmType

рисунку 4.1.

Даний об’єкт надалі використовується для формування запиту до бібліотеки шляхом передачі цього обєкту до RequestEntity. При розширенні підтримуваних алгоритмів назви нових алгоритмів повинні добавлятися саме в цей об’єкт.

Для опису категорія об’єктів для опрацювання реалізовано enum під назвою ObjectCategory, даний тип використовується для визначення категорії розміру об’єкта для опрацювання. Ця категорія є частиною критеріїв для визначення найоптимальнішого алгоритма та буде враховуватися при його виборі. Цей об’єкт є параметром класу RequestEntity та дозволяє користувачу бібліотеки самому визначати, до якої категорії віднести цей об’єкт. У випадку відсутності цього визначення, система автоматично буде встановлювати, до якої категорії віднести цей об’єкт, базуючись на його розмірі. Реалізацію даного об’єкта зображено на рисунку 4.2.

```
public enum ObjectCategory {
    LOW,
    MEDIUM,
    LARGE
}
```

Рисунок 4.2 – Реалізація ObjectCategory

Для визначення параметрів при роботі користувача з бібліотекою, реалізовано клас RequestEntity. Даний клас містить в собі інформацію про об’єкт, з яким необхідно виконати операцію. Він також містить інформацію про те, який алгоритм для даної операції використовувати та до якої категорії відноситься об’єкт для опрацювання. У разі відсутності інформації щодо типу алгоритму або категорії об’єкту, дані параметри будуть встановлені автоматично системою.

Клас використовується як головний елемент спілкування між користувачем та бібліотекою і є обов‘язковим параметром при використанні зовнішнього інтерфейсу бібліотеки. Імплементація даного класу зображена на рисунку 4.3.

```
public class RequestEntity<T> {

    T objectToEncrypt;
    AlgorithmType algorithmType;
    ObjectCategory objectCategory;

    public T getObjectToEncrypt() {
        return objectToEncrypt;
    }

    public void setObjectToEncrypt(T objectToEncrypt) {
        this.objectToEncrypt = objectToEncrypt;
    }

    public AlgorithmType getAlgorithmType() {
        return algorithmType;
    }

    public void setAlgorithmType(AlgorithmType algorithmType) {
        this.algorithmType = algorithmType;
    }

    public ObjectCategory getObjectCategory() {
        return objectCategory;
    }

    public void setObjectCategory(ObjectCategory objectCategory) {
        this.objectCategory = objectCategory;
    }
}
```

Рисунок 4.3 – Реалізація RequestEntity

Для забезпечення зовнішнього інтерфейсу використання бібліотеки та опису його функцій використовується інтерфейс EncriptService. Даний інтерфейс містить в собі ключові методи для спілкування з бібліотекою, а саме doEncrypt. Метод, що виконує операцій над об’єктом, який передається, та містить параметри у вигляді RequestEntity. Результатом роботи даного методу є ResponseEntity, який формує шаблон відповіді та містить в собі інформацію про об’єкт, який передавався, ключ та результати виконаних операцій з переданим об’єктом. Реалізація інтерфейсу зображено на рисунку 4.4.

Рисунок 4.4 – Реалізація EncryptService

```
public interface EncryptService {

    ResponseEntity doEncrypt(Object objectToEncrypt, AlgorithmType algorithmType);

    ResponseEntity doEncrypt(Object objectToEncrypt);
}
```

Для створення бібліотеки та отримання можливості її версійного супроводу та визначення назви артефакту, до проекту було додано утиліту для побудови проекту. Maven - це засіб автоматизації роботи з програмними проєктами, який спочатку використовувався для Java проєктів. Використовується для управління та складання програм. Дана утиліта дає можливість описати, як повинен виглядати файл бібліотеки, що дозволяє його скомпілювати, а також містить в собі інформацію стосовно додаткових бібліотек, які повинні бути підключені до даної системи. Реалізація конфігурації для побудови бібліотеки зображено на рисунку 4.5.

```
<?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>groupId</groupId>
    <artifactId>cryptoProject</artifactId>
    <version>1.0-SNAPSHOT</version>


</project>
```

Рисунок 4.5 – Реалізація конфігурації бібліотеки

У систему за допомогою збиральника проекту Maven було додано функціональність для роботи з алгоритмом шифрування MD5.

Дана бібліотека містить в собі функціональність, що дає можливість використовувати його для операцій з різними об’єктами. Реалізація підключення даної бібліотеки зображено на рисунку 4.6.

```
<!-- https://mvnrepository.com/artifact/com.joyent.util/fast-md5 -->
<dependency>
    <groupId>com.joyent.util</groupId>
    <artifactId>fast-md5</artifactId>
    <version>2.7.1</version>
</dependency>
```

Рисунок 4.6 – Реалізація підключення MD5

Для використання функціональності SHA1, її бібліотеку було додано в систему за допомогою збиральника проекту Maven. Дана бібліотека дозволяє використовувати функціонал алгоритму, а саме, виконання операцій з різними об’єктами. Реалізація підключення бібліотеки зображено на рисунку 4.7.

```
<!-- https://mvnrepository.com/artifact/org.webjars.npm/crc-32 -->
<dependency>
    <groupId>org.webjars.npm</groupId>
    <artifactId>crc-32</artifactId>
    <version>0.4.0</version>
</dependency>
```

Рисунок 4.7 – Реалізація підключення SHA1

Для того щоб мати змогу використовувати функціональність алгоритму CRC32, додано бібліотеку даного алгоритму за допомогою збірника проекту Maven, що дає змогу використовувати у роботі функціональність даного алгоритму. Реалізація підключення даної бібліотеки зображена на рисунку 4.8.

```
<!-- https://mvnrepository.com/artifact/li.rudin.mavenjs/sha1 -->
<dependency>
    <groupId>li.rudin.mavenjs</groupId>
    <artifactId>sha1</artifactId>
    <version>1.0.0</version>
</dependency>
```

Рисунок 4.8 – Реалізація підключення CRC32

Реалізація даного програмного коду дозволяє використовувати систему за призначенням та маштабувати бібліотеку у відповідності до нових вимог, серед яких використання нових алгоритмів шифрування та нові критерії для визначення операцій.

4.2 Розробка моделі перетворення даних на паралельних обчисленнях

В ході розробки метода шифрування передачі даних між хмарними підпросторами було вирішено наступні завдання:

Виконано опис та побудову функціональних блоків моделі, також було описано алгоритми для обробки даних, порівняно та обчислено теоретичний рівень продуктивності системи.

Шляхом практичних експериментів продемонстровано наявність затримок або швидкодії між ітераціями обробки та шифрування вхідних даних. Встановлено, що причиною затримок є значна кількість блоків з даними, які є результатом поділу файлів на менші частини.

Так як дані обробляються у послідовному режимі, це означає, що під час обробки одного набору даних всі інші перебувають в режимі очікування.

Використовувані алгоритми шифрування ґрунтується на асиметричних схемах перетворення даних, тому дана особливість може бути застосована для визначення часу лише при операції шифрування, в той час як процедури дешифрування даних використовують лінійні операції з важкими обчисленнями коренів, обчислення дискретного логарифму тощо.

Використання даних функції та підходів значно впливає на час функціонування вихідної модель. Як результат, кінцевий час для виконання обробки всіх даних навіть у багатопотоковому режимі може бути скоригуваним і оціненим виразом 4.1, при цьому самі механізми шифрування не зазнають жодних змін.

$$S_p = \frac{T_0}{T_p} = \frac{1}{k} \cdot BQ \cdot f_T(f_{1,2}) \approx 1 - wp \cdot \left(1 - \frac{1}{p}\right) \quad (4.1)$$

де S – коефіцієнт прискорення програмного коду,

To – час виконання програмного коду в однопотоковому режимі,

Tp – час виконання програмного коду в багатопотоковому режимі,

k – кількість обчислювальних потоків,

BQ –кількість блоків з даними,

fT – функція оцінки часу виконання програмного коду,

f1,2 – процедури шифрування та дешифрування,

wp – відносна частка паралельно виконуваних операцій,

p – кількість ядер центрального процесора.

Для того, щоб не впливати на алгоритми шифрування, забезпечити надійний рівень захисту даних та привести вихідну модель до вигляду, який дозволяє використання паралельних обчислень, було створено узагальнену

Рисунок 4.9 – Використання паралельних обчислень

модель, яка частково відображена на рисунку 4.9.

Такий підхід до створення та структуризації системи обробки даних дозволяє за рахунок невеликих змін в самій структурі скоротити кількість вільних ядер і, як наслідок, задіяти їх для збільшення швидкості перетворення даних у багатопотоковому режимі.

4.2.1 Приклад моделі шифрування

Для демонстрації запропонованої моделі паралельного обрахунку, яка реалізує механізм шифрування даних з відкритим ключем, було представлено наступну систему.

Дана система враховує те, що застосовано паралельне виконання обчислень на рівні програмного коду. Для використання такого типу обчислень, зокрема, було добавлено блоки балансування навантаження та контроль. Тепер є блок коду, який виконує оцінку характеристик вхідних даних та дозволяє виконати обробку результатів обчислень, попередньо поділивши їх на різні блоки.

Блоки зображені на рисунку 4.10 та 4.11.

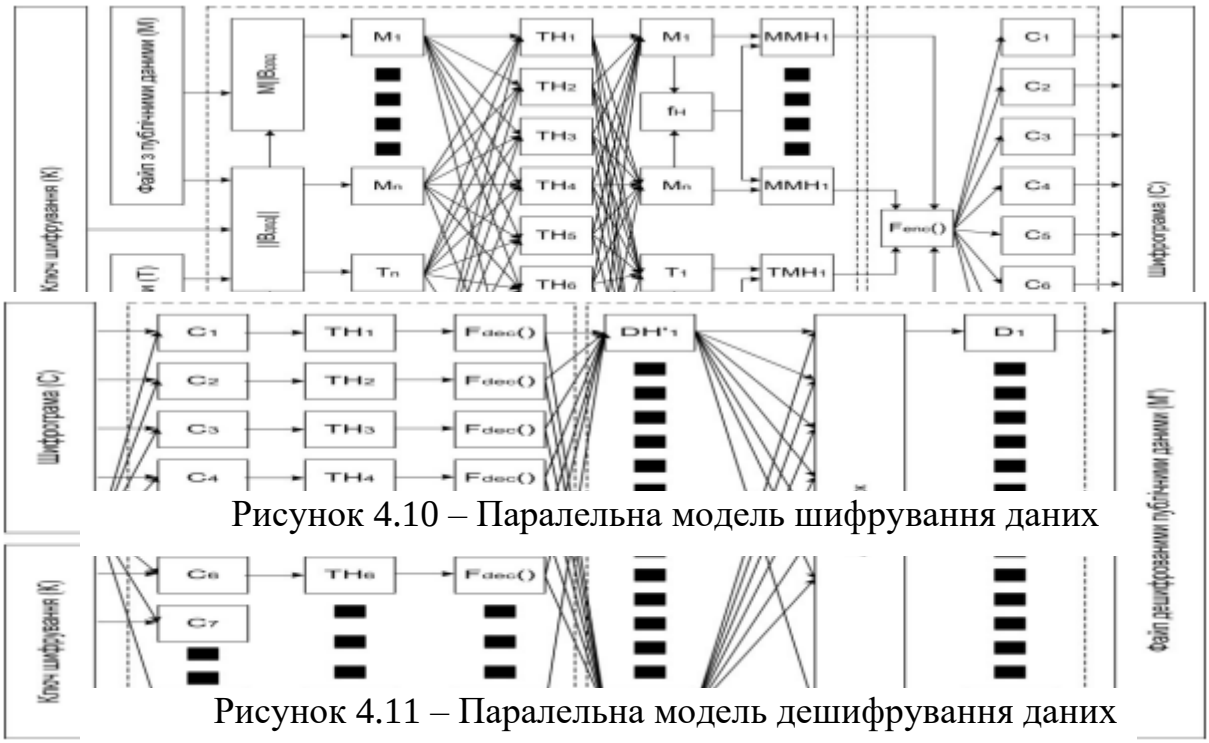


Рисунок 4.10 – Паралельна модель шифрування даних

Рисунок 4.11 – Паралельна модель дешифрування даних

Блок оцінки завантаженості системи дозволяє визначити, скільки ядер можна використовувати для розпаралелення процесу шифрування даних. Його основним завданням є визначення технічних характеристик систем для того, щоб зрозуміти, яку кількість ядер можна використовувати для пришвидшення вказаної моделі.

Блок для обробки вхідних даних призначений для підготовки закритих та відкритих даних, їх доопрацювання.

Він містить в собі додаткові підблоки завантаження даних у систему порівнянням вихідних файлів або даних один з одним та декомпозиції вихідних даних на блоки, що дозволить потім їх окремо опрацьовувати у паралельному режимі. Блок перетворення даних є моделлю, яка ґрунтується на процедурах алгоритмів шифрування. Вона включає в себе блоки шифрування даних.

1) Визначити розмір файлу з відкритими даними $\|F_M\|$.

2) Визначити розмір файлу з закритими даними $\|F_T\|$.

3) Визначити різницю між розмірами файлів $dF_S = \|F_M\|, \|F_T\|$.

4) Визначити розмір блоку з даними, які будуть перетворені програмою.

5) Визначити максимальний розмір вихідних даних $D_{\max} = f_{\max}(\|F_M\|, \|F_T\|)$.

6) Визначити загальний розмір вихідних даних за допомогою виразу $D_{\max} = f_{\max}(\|F_M\|, \|F_T\|) + AD_S$.

Поділ даних на окремі блоки для подальшого опрацювання передбачає фізичне розділення вихідних даних на окремі блоки за визначеним розміром та їх попередньою обробкою. Дана процедура виконує декомпозицію вхідних даних на *к* фрагментів, які повинні бути приблизно рівнозначного розміру.

4.2.2 Процедура захисту даних від змін

Важливим аспектом опрацювання даних є процедура їх захисту. Процедура захисту даних від змін передбачає попереднє опрацювання даних, що дозволяє перевіряти цілісність даних для подальшого використання та відтворення. Ця процедура є універсальною, тому при використанні описаної вище декомпозиції дозволяє застосовувати її для визначення цілісності даних не лише окремих блоків, але й всіх об’єктів разом. Даний процес включає наступні кроки.

– Генерація динамічної або статистичної мітки, яка є кінцевим розміром одного блоку;

– Імплементація додаткової мітки у кожен блок розпаралеленого об’єкту;

– Виконання контролю за блоками, щоб кожен блок відповідав вказаному розміру та містив у собі додаткову мітку.

4.2.3 Процедура верифікації блоків з даними

Процедура верифікації необхідна для того, щоб у результаті використання операцій над даними після їх розпаралелення, можна було зібрати всі результуючі блоки у єдине ціле та надати фінальний результат у вигляді опрацьованих даних.

Дана процедура містит у собі функціональність, яка дозволяє отримати мітку з розпаралелених блоків та, з урахуванням цих міток, зробити склейку блоків, що дозволить у кінцевому результаті отримати загальний об’єкт.

4.2.4 Особливості використання паралельного обрахунку даних

У зв’язку з вищезазначеними даними, для ефективного використання моделі з паралельними обчисленнями і отримання найбільш оптимального рівня

пришвидшення операцій, необхідно використовувати ряд апаратних вимог, зокрема, використання спеціального апаратного забезпечення.

Необхідна наявність центральних процесорів, які підтримують паралельні обчислення та використовують технологію мультиядерності.

Для того, щоб отримати максимальні результати пришвидшення, рекомендовано використання апаратного забезпечення під відомим брендом Intel або іншим його конкурентом процесором M1. Потужностей указаних процесорів буде достатньо для обробки даних розміром приблизно 100 блоків.

Також необхідно розуміти, що ефективне використання 4-х або 2-х ядерних процесорів у межах одного обчислювального пристрою буде вимагати використання всіх доступних каналів пам’яті ОЗП. Як результат, стає зрозуміло, що від типу пристрою та його апаратного забезпечення сильно залежить кінцева продуктивність системи. На теперішній час найкращими апаратними складовими для того, щоб отримати максимальну продуктивність системи, було визначено тип пам’яті ОЗП DDR4 з тактовою частотою, яка складає 3.2 ГГц.

Як результат, типова конфігурація системи, яка дозволить використати максимальні можливості спроектованої системи на базі різних обчислювальних розподілів на блоки та паралельного використання, буде включати в себе процес з 18 фізичними ядрами, наприклад, Intel Core I9 та оперативною пам’яттю 48 гігабайт типу DDR4 і частотою 3,2 ГГц.

4.3 Функціональне дослідження системи шифрування даних

4.3.1 Апробація прототипів моделі

Ефективність вище вказаних моделей була перевірена під час серії експериментів, де індикатором ефективності запропонованого рішення є зменшення часу на виконання операцій шифрування та дешифрування над тестовою моделлю.

4.3.2 Опис тестового середовища

Для тестування моделей обрано обладнання з більш високими технічними характеристиками, яке включає в себе x64-based PC, Intel64 Family 6 Model 142 Stepping 10 Genuine Intel ~1801 МГц, 8 ГБ RAM, HDD 500 ГБ, ОС Windows 10 Pro (10.0.18362 – Multiprocessor Free, IDLE Python v3.7, бібліотека line_profiler).

4.3.3 Обмеження експерименту

Для проведення експериментів використані вихідні дані:

1) формати вхідних даних: текстові, *.exe, *.zip;

2) розмір вхідних даних: 1КБ ÷ 20 МБ;

3) розмір ключа шифрування: 1024 (з точки зору продуктивності) та 8192 (з точки зору безпеки) біти;

4) розмір хешу даних: 1 ÷ 2 байти;

5) формат захисної мітки довільного змісту (наприклад, «!end!») та розміру від 3 байт.

4.3.4 Результати експериментів

4.3.4.1 Результати тестування звичайної моделі

Було проведено експерименти для звичайного режим обчислень за допомогою базової моделі. Результати дешифрування та шифрування даних зображено у таблицях 4.12 та 4.13.

Номер експерименту	Розмір ключа, біт	
	1024	8192
	Час шифрування, с	
1	1,882	4,507
2	1,755	4,714
3	1,701	4,763
4	1,622	4,775
5	1,829	4,735
6	1,748	4,675
7	1,692	4,781
8	1,672	4,774
9	1,744	4,741

Рисунок 4.12 – Результати шифрування даних у звичайній моделі

Номер експерименту	Тип даних			
	Відкриті	Закриті	Відкриті	Закриті
	Розмір ключа, біт			
	1024		8192	
	Час дешифрування, с			
1	386,036	7801,760	1456,720	29440,232
2	396,448	8012,200	663,536	13410,088
3	355,436	7183,320	697,820	14102,984
4	378,704	7653,600	736,000	14874,564
5	303,936	6142,480	820,056	16573,380
6	402,164	8127,720	804,964	16268,320
7	415,784	8403,000	757,724	15313,608
8	409,260	8271,120	831,276	16800,152
9	326,080	6590,080	919,624	18585,600
10	267,840	5413,040	937,528	18947,472

Рисунок 4.13 – Результати дешифрування даних у звичайній моделі

4.3.4.2 Результати тестування моделі з використанням розподілу даних

Були проведені експерименти за допомогою модифікації моделі обробки даних, а саме, розподілу її на окремі блоки та використання визначених конфігурацій для паралельного обчислення даних. Результати експериментів продемонстровано в таблицях 4.14 та 4.15.

Номер експерименту	Розмір ключа, біт	
	1024	8192
1	6,405	8,108
2	6,655	6,874
3	7,155	8,179
4	7,108	7,889
5	6,389	7,326
6	6,608	7,873
7	7,303	7,873
8	7,264	7,624
9	7,108	7,998
10	6,936	6,874

Рисунок 4.14 – Результати шифрування даних у паралельній моделі

Номер експерименту	Тип даних			
	Відкриті	Закриті	Відкриті	Закриті
	Розмір ключа, біт			
	1024		8192	
	Час дешифрування, с			
1	163,300	3300,300	309,980	6265,390
2	162,600	3286,330	321,360	6309,970
3	136,190	2752,580	317,680	6761,950
4	130,240	2632,320	309,360	6799,400
5	150,540	3042,420	265,570	8034,680
6	154,580	3124,240	318,980	11429,71
7	150,960	3051,040	317,910	8566,110
8	151,750	3066,870	309,030	5799,390
9	135,230	2732,990	306,360	6437,790
10	150,180	3035,200	295,740	5688,390

Рисунок 4.15 – Результати дешифрування даних у звичайній моделі

4.3.5 Підсумок результатів тестування моделі

В подальшому на базі отриманих експериментальних даних для визначення ефективності рішення було визначено, що результати обчислень показують прискорення, при цьому середній коефіцієнт прискорення становить до 4 разів в порівнянні з початковим та стандартним варіантом моделі. Для того, щоб описати отримані вище результати було виконано розрахунок відносної

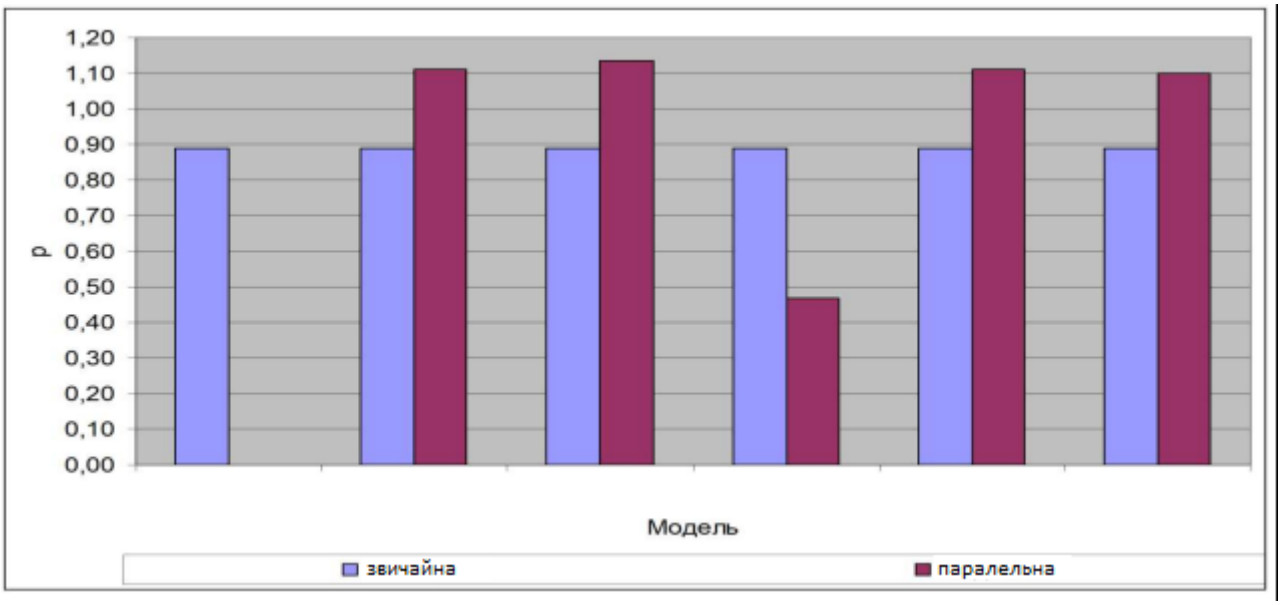


Рисунок 4.16 – Розрахунок відносної частки паралельно виконуваних

частки паралельно виконуваних моделей за рахунок формули 4.1 та зображено на рисунку 4.16.

Подальші експерименти з використанням даної тестової моделі показали продуктивність обробки даних при шифруванні їх за допомогою використання підходу з паралельного шифрування та поділу на блоки.

При цьому, сумарний рівень прискорення моделі складав в 2,5 раз у порівнянні зі швидкістю роботи шифрування без використання модифікації.

Причиною цього є те, що частка паралельного виконування операцій складає 60-90 відсотків згідно з формулою 4.1. Це свідчить про зростання ефективності запропонованого підходу на 14-40 відсотків. Як результат, ці показники є очікуваними та позитивними.

За результатами експериментів з модифікацію для паралельного обчислення та розбиття на блоки, було встановлено, що сумарний коефіцієнт прискорення тестової моделі зріст у декілька разів. Разом з тим відносна частка паралельно виконуваних операцій складала 110-120 відсотків, що призводить до зниження ефективності використання паралельних обчислень за рахунок наднормового використання ресурсів. Незважаючи на це, результати є найбільш наближеними до максимально можливого рівня прискорення з використанням паралельного обчислення, тим самим показуючи, що ефективність використання паралельних обчислень залежить від кількості операцій в програмі та поточного навантаження на апаратний пристрій.

Згідно вище сказаних даних, підхід з розпаралеленям даних для опрацювання на блоки дозволяє значно пришвидшити цей процес, але незважаючи на цей підхід, під час дешифрування даних продуктивність суттєво знижується через використання асиметричних алгоритмів математичних моделей.

Висновки до розділу 4

Згідно з початковою метою та завданням, яке було поставлено на початку розділу, було виконано огляд паралельного методу та методи оцінки продуктивності, які використовуються при підготовці даних для їх опрацювання.

В результаті цього було вирішено наступні завдання:

1). Побудовано базову модель обчислень з подальшою реалізацію алгоритмів шифрування на базі підготовки даних за допомогою паралельно обчислювальної системи;

2). Вказана система використовує примітивний метод балансування навантаження, який полягає у визначенні апаратних можливостей та використання їх для розподілу вхідних даних на блоки. В результаті проведених експериментів подібний підхід показав зростання швидкості шифрування даних в 2,5 рази у порівнянні з швидкістю роботи звичайного алгоритму.

Розроблено версію з використанням блоків, яка ґрунтується на паралельних обчисленнях з попередньою обробкою вихідних даних, а саме, розподілом їх на блоки. Як результат, це також призвело до скорочення часу обробки моделей у порівнянні з швидкістю асиметричного підходу.

Проведено дослідження причин зниження ефективності використання паралельних обчислень при досить великих файлах, коли завантажені системи становить більше, ніж сто відсотків.

Основною причиною даного зниження ефективності є зростання показника відносної кількості операцій, паралельно виконуваних програмою. Даний показник є наслідком завантаженості системи або розпаралелення об’єкту на занадто велику кількість блоків при використанні малої кількості ядер процесора.

З метою оцінки ефективності запропонованих рішень, було виконано порівняння швидкості їх роботи зі швидкостями роботи звичайно асиметричного шифрування. Встановлено, що новий підхід має високий рівень швидкодії у порівнянні зі звичайним підходом. Та разом з тим, швидкість під час

дешифрування даних є близька до швидкості дешифрування даних зі звичайним підходом за рахунок того, що використовуються асиметричні алгоритми.

Загальні висновки

За допомогою отриманої інформації стосовно методів шифрованої передачі даних, які використовуються на сьогодні, стала зрозуміла необхідність у шифруванні даних для того, щоб забезпечити безпеку та надійність їх передачі. Отримані знання про методи та підходи, які використовуються при шифруванні та дешифруванні даних, дозволяють краще розуміти сучасні вимоги до створення будь-якого програмного забезпечення, яке займається обміном даних.

Аналіз небезпек, які існують через відсутність зашифрованих каналів зв’язку, вказує на те, що правильний підбір алгоритмів шифрування та правильне створення структури доступу до даних дозволяє забезпечити коректну та безперебійну роботу кожної системи.

Розглянуте програмне забезпечення, яке дає можливість шифрувати дані, дозволяє використовувати вже наявні підходи для створення безпеки даних.

Було отримано знання стосовно математичних моделей алгоритмів шифрування даних. Детальний розбір математичної моделі декількох алгоритмів дозволяє мати уявлення про процес шифрування та дешифрування даних. Була визначена різниця між симетричними та асиметричними криптосистемами, що дозволяє в майбутньому краще підібрати шлях для збереження цілісності та безпеки даних.

Розуміння математичної моделі при створенні шифрування дозволяє її покращити та додатково створити своє бачення того, як необхідно розробляти систему шифрування та, у випадку атак, розробити систему найбільш стійкою до них.

Розглянуті схеми найбільш захищених та популярних криптосистем дозволяють мати уявлення про сучасні вимоги та тенденції у створенні шифрування та передачі даних, що позитивно впливає на риторику самої теми та дозволяє мати уявлення про те, як можна розвинути ці схеми.

Знання про паралельні обчислення дозволить максимально пришвидшити процес обчислення та шифрування даних, а також розуміти аспекти, на які необхідно звернути увагу.

Отримавши всі необхідні знання стосовно алгоритмів, які використовується у системі шифрування та дешифрування даних, було визначено ряд критеріїв для відбору найкращого алгоритму в залежності від вихідних параметрів.

Даний підхід дозволив опрацьовувати вхідні параметри за рахунок найшвидшого алгоритму, який найкраще підходить для шифрування та дешифрування даних.

Реалізація даної системи допомогла зрозуміти різницю між алгоритмами, створити структуру, яка буде отримувати вхідні параметри, та налаштовувати конфігурації для подальшої обробки даних, а також формування відповіді, яка буде доцільною для великої категорій користувачів даної системи.

Розуміння параметрів, необхідних для перевірки вхідного об’єкта, та його опрацювання дозволяє згенерувати звітність та розуміння іншими розробниками як саме користуватися даною бібліотекою та які конфігурації застосовувати для її роботи.

Отримані знання стосовно роботи алгоритмів надалі можна буде використовувати дані алгоритми та підходи для побудови інших систем, які будуть спілкуватися між собою за рахунок закритих каналів зв’язку, що буде вимагати використання алгоритмів шифрування та розуміння побудови системи, зв’язків між сервісами, щоб уникнути атак на дані та захистити їх.

Побудовано базову модель обчислень з подальшою реалізацію алгоритмів шифрування на базі підготовки даних за допомогою паралельно-обчислювальної системи.

Вказана система використовує примітивний метод балансування навантаження, який полягає у визначенні апаратних можливостей та використання їх для розподілу вхідних даних на блоки. В результаті проведених

експериментів подібний підхід показав зростання швидкості шифрування даних в 2,5 рази у порівнянні із швидкістю роботи звичайного алгоритму.

Розроблено версію з використанням блоків, яка ґрунтується на паралельних обчисленнях з попередньою обробкою вихідних даних, а саме, розподілом їх на блоки. Як результат, це також призвело до скорочення часу обробки моделей у порівнянні зі швидкістю асиметричного підходу.

Проведено дослідження причин зниження ефективності використання паралельних обчислень при досить великих файлах, коли завантаженість системи є понаднормовою.

Перелік посилань

1. Воробьев И.В. Защита информации в персональных ЭВМ. М.: Мир, 2008. 312 с.

2. Мафтик С. Механизмы защиты в компьютерных сетях. М.: Мир, 2013. 219 с.

3. Ростовцев А.Г., Маховенко Е.Б. Теоретическая криптография. М.: Профессионал, 2005. 480 с.

4. Рябко Б.Я., Фионов С.В. Криптографические методы защиты информации: Учебное пособие для вузов. М.: Телеком, 2005. 229 с.

5. Панасенко С.П. Алгоритмы шифрования. М.: Академический Проект, 2006. 529 с. 76

6. Ella D. L. Шифрування та Дешифрування [Електронний ресурс] / Deon Lackey Ella. – 2016. – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Archive/Security/Шифрування_та_Дешифрування.

7. Петров В.П., Петров С.В. Информационная безопасность человека и общества. М.: ЭНАС, 2015. 334 с.

8. Бранстед Д.К., Смид М.Э., Стандарт шифрования данных: прошлое и будущее. М.: МИИЭР, 2008. 513 с.

9. Шнаер Б. Практическая криптография. М.: Вильямс, 2007. 424 с.

10. Задірака В.К., Олексюк О.С. Комп’ютерна криптологія. Тернопіль, Київ, 2002. 504 с.

11. Міхаель Рабін [Електронний ресурс] Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9C%D1%96%D1%85%D0%B0%D0%B5%D0%BB%D1%8C_%D0%A0%D0%B0%D0%B1%D1%96%D0%BD

12. Схема Ель-Гамаля [Електронний ресурс] Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_%D0%95%D0%BB%D1%8C-%D0%93%D0%B0%D0%BC%D0%B0%D0%BB%D1%8F

13. Алгоритм шифрування MD5 [Електронний ресурс] Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%94%D1%96%

14. Протокол Діффі Геллмана [Електронний ресурс] Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%94%D1%96%D1%84%D1%84%D1%96_%E2%80%94_%D0%93%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD%D0%B0

15. Криптографія [Електронний ресурс] Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D1%96%D1%8F

16. MD5 [Електронний ресурс] Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MD5

17. Циклічний надлишковий код [Електронний ресурс] Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB%D1%96%D1%87%D0%BD%D0%B8%D0%B9_%D0%BD%D0%B0%D0%B4%D0%BB%D0%B8%D1%88%D0%BA%D0%BE%D0%B2%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4

18. SHA1 [Електронний ресурс] Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SHA-1

19. Петухов М. Симметричный алгоритм шифрования IDEA. Долгопрудный: ГУ МФТИ, 2017. 14 с.

20. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. Москва: Триумф, 2012. 816 с. 167

21. Moriai S., Yiqun L. Y. Cryptanalysis of Twofish (II). URL: https://www.schneier.com/wp-content/uploads/2015/01/twofish-analysis-shiho.pdf (дата звернення: 01.09.2020).

22. Menezes A. J., Oorschot P. V., Vanstone S. A. Handbook of Applied Cryptography: CRC Press, 1996. 816 p.

23. Anderson R., Biham E., Knudsen L. (2000) Serpent and Smartcards. Proceedings of the CARDIS'98: 3th International Conference on Smart Card Research and Applications in Belgium, (Louvain-la-Neuve, September 14-16, 2000), pp. 246-253.

24. Krishnamurthy G. N., Ramaswamy Dr. V. Encryption Quality Analysis and Security Evaluation of CAST-128 Algorithm and its Modified Version using Digital Images. International Journal of Network Security & Its Applications (IJNSA). 2009. Vol. 1, No. 1. P. 28-33.

25. Vanhoef M., Piessens F. (2015) All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS. 24th USENIX Security Symposium in USA (Washington, August 12–14, 2015), pp. 97-112.

26. Prohibiting RC4 Cipher Suites. URL: https://tools.ietf.org/html/rfc7465 (дата звернення: 01.09.2020).

27. Advanced encryption standart (AES). FIPS 197. [Чинний від 2001-11-26]. Гейтерсберг: NIST, 2001. 51 с.

28. AES (Advanced Encryption Standard). URL: https://ru.bmstu.wiki/AES_(Advanced_Encryption_Standard) (дата звернення: 12.06.2016).

29. Заровний В.І. Метод шифрованої передачі даних між хмарними підпросторами / Заровний В.І., Скрипник Т.К. // Збірник наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021». – Хмельницький, 2021. – С. 335-337.

Додатки

Додаток А
Програмні коди

```
package com.crypto.project.entity;

import com.crypto.project.enums.AlgorithmType;
import com.crypto.project.enums.ObjectCategory;

public class RequestEntity<T> {

    T objectToEncrypt;
    AlgorithmType algorithmType;
    ObjectCategory objectCategory;

    public T getObjectToEncrypt() {
        return objectToEncrypt;
    }

    public void setObjectToEncrypt(T objectToEncrypt) {
        this.objectToEncrypt = objectToEncrypt;
    }

    public AlgorithmType getAlgorithmType() {
        return algorithmType;
    }

    public void setAlgorithmType(AlgorithmType algorithmType) {
        this.algorithmType = algorithmType;
    }

    public ObjectCategory getObjectCategory() {
        return objectCategory;
    }

    public void setObjectCategory(ObjectCategory objectCategory) {
        this.objectCategory = objectCategory;
    }
}
```

```
package com.crypto.project.interfaces;

import com.crypto.project.entity.RequestEntity;
import com.crypto.project.entity.ResponseEntity;
import com.crypto.project.enums.AlgorithmType;

public interface EncryptService {

    ResponseEntity doEncrypt(RequestEntity objectToEncrypt, AlgorithmType algorithmType);

    ResponseEntity doEncrypt(RequestEntity objectToEncrypt);
}
```

```
package com.crypto.project.main;

import com.crypto.project.enums.AlgorithmType;
import com.crypto.project.enums.ObjectCategory;

public class Main {

    public static void main(String[] args) {
    }

}
```

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by Fernflower decompiler)
//

package com.twmacinta.util;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
```

```
import java.io.InputStream;
import java.io.UnsupportedEncodingException;

public class MD5 {
    MD5State state;
    MD5State finals;
    static byte[] padding = new byte[]{-128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    private static boolean native_lib_loaded = false;
    private static boolean native_lib_init_pending = true;
    private static final char[] HEX_CHARS = new char[]{'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'};

    public synchronized void Init() {
        this.state = new MD5State();
        this.finals = null;
    }

    public MD5() {
        if (native_lib_init_pending) {
            _initNativeLibrary();
        }

        this.Init();
    }

    public MD5(Object ob) {
        this();
        this.Update(ob.toString());
    }

    private void Decode(byte[] buffer, int shift, int[] out) {
        out[0] = buffer[shift] & 255 | (buffer[shift + 1] & 255) << 8 | (buffer[shift + 2] & 255) << 16 | buffer[shift + 3] <<
24;
        out[1] = buffer[shift + 4] & 255 | (buffer[shift + 5] & 255) << 8 | (buffer[shift + 6] & 255) << 16 | buffer[shift + 7]
<< 24;
        out[2] = buffer[shift + 8] & 255 | (buffer[shift + 9] & 255) << 8 | (buffer[shift + 10] & 255) << 16 | buffer[shift + 11]
<< 24;
        out[3] = buffer[shift + 12] & 255 | (buffer[shift + 13] & 255) << 8 | (buffer[shift + 14] & 255) << 16 | buffer[shift +
15] << 24;
        out[4] = buffer[shift + 16] & 255 | (buffer[shift + 17] & 255) << 8 | (buffer[shift + 18] & 255) << 16 | buffer[shift +
19] << 24;
```

```
        out[5] = buffer[shift + 20] & 255 | (buffer[shift + 21] & 255) << 8 | (buffer[shift + 22] & 255) << 16 | buffer[shift +
23] << 24;
        out[6] = buffer[shift + 24] & 255 | (buffer[shift + 25] & 255) << 8 | (buffer[shift + 26] & 255) << 16 | buffer[shift +
27] << 24;
        out[7] = buffer[shift + 28] & 255 | (buffer[shift + 29] & 255) << 8 | (buffer[shift + 30] & 255) << 16 | buffer[shift +
31] << 24;
        out[8] = buffer[shift + 32] & 255 | (buffer[shift + 33] & 255) << 8 | (buffer[shift + 34] & 255) << 16 | buffer[shift +
35] << 24;
        out[9] = buffer[shift + 36] & 255 | (buffer[shift + 37] & 255) << 8 | (buffer[shift + 38] & 255) << 16 | buffer[shift +
39] << 24;
        out[10] = buffer[shift + 40] & 255 | (buffer[shift + 41] & 255) << 8 | (buffer[shift + 42] & 255) << 16 | buffer[shift +
43] << 24;
        out[11] = buffer[shift + 44] & 255 | (buffer[shift + 45] & 255) << 8 | (buffer[shift + 46] & 255) << 16 | buffer[shift +
47] << 24;
        out[12] = buffer[shift + 48] & 255 | (buffer[shift + 49] & 255) << 8 | (buffer[shift + 50] & 255) << 16 | buffer[shift +
51] << 24;
        out[13] = buffer[shift + 52] & 255 | (buffer[shift + 53] & 255) << 8 | (buffer[shift + 54] & 255) << 16 | buffer[shift +
55] << 24;
        out[14] = buffer[shift + 56] & 255 | (buffer[shift + 57] & 255) << 8 | (buffer[shift + 58] & 255) << 16 | buffer[shift +
59] << 24;
        out[15] = buffer[shift + 60] & 255 | (buffer[shift + 61] & 255) << 8 | (buffer[shift + 62] & 255) << 16 | buffer[shift +
63] << 24;
    }


    public void Update(MD5State stat, byte[] buffer, int offset, int length) {
        this.finals = null;
        if (length - offset > buffer.length) {
            length = buffer.length - offset;
        }

        int index = (int)(stat.count & 63L);
        stat.count += (long)length;
        int partlen = 64 - index;
        int i;
        if (length >= partlen) {
            if (native_lib_loaded) {
                if (partlen == 64) {
                    partlen = 0;
                } else {
                    for(i = 0; i < partlen; ++i) {
```

```
            stat.buffer[i + index] = buffer[i + offset];
         }

         this.Transform_native(stat.state, stat.buffer, 0, 64);
      }

      i = partlen + (length - partlen) / 64 * 64;
      int transformLength = length - partlen;
      int transformOffset = partlen + offset;

      for(int var11 = 65536; transformLength > 65536; transformOffset += 65536) {
         this.Transform_native(stat.state, buffer, transformOffset, 65536);
         transformLength -= 65536;
      }

      this.Transform_native(stat.state, buffer, transformOffset, transformLength);
   } else {
      int[] decode_buf = new int[16];
      if (partlen == 64) {
         partlen = 0;
      } else {
         for(i = 0; i < partlen; ++i) {
            stat.buffer[i + index] = buffer[i + offset];
         }

         this.Transform(stat, stat.buffer, 0, decode_buf);
      }

      for(i = partlen; i + 63 < length; i += 64) {
         this.Transform(stat, buffer, i + offset, decode_buf);
      }
   }

   index = 0;
} else {
   i = 0;
}

if (i < length) {
   for(int start = i; i < length; ++i) {
      stat.buffer[index + i - start] = buffer[i + offset];
```

```
        }
    }

}

public void Update(byte[] buffer, int offset, int length) {
    this.Update(this.state, buffer, offset, length);
}

public void Update(byte[] buffer, int length) {
    this.Update(this.state, buffer, 0, length);
}

public void Update(byte[] buffer) {
    this.Update(buffer, 0, buffer.length);
}

public void Update(byte b) {
    byte[] buffer = new byte[]{b};
    this.Update(buffer, 1);
}

public void Update(String s) {
    byte[] chars = s.getBytes();
    this.Update(chars, chars.length);
}

public void Update(String s, String charset_name) throws UnsupportedEncodingException {
    if (charset_name == null) {
        charset_name = "ISO8859_1";
    }

    byte[] chars = s.getBytes(charset_name);
    this.Update(chars, chars.length);
}

public void Update(int i) {
    this.Update((byte)(i & 255));
}

private byte[] Encode(int[] input, int len) {
```

```
    byte[] out = new byte[len];
    int j = 0;

    for(int i = 0; j < len; j += 4) {
        out[j] = (byte)(input[i] & 255);
        out[j + 1] = (byte)(input[i] >>> 8 & 255);
        out[j + 2] = (byte)(input[i] >>> 16 & 255);
        out[j + 3] = (byte)(input[i] >>> 24 & 255);
        ++i;
    }

    return out;
}

public synchronized byte[] Final() {
    if (this.finals == null) {
        MD5State fin = new MD5State(this.state);
        int[] count_ints = new int[]{(int)(fin.count << 3), (int)(fin.count >> 29)};
        byte[] bits = this.Encode(count_ints, 8);
        int index = (int)(fin.count & 63L);
        int padlen = index < 56 ? 56 - index : 120 - index;
        this.Update(fin, padding, 0, padlen);
        this.Update(fin, bits, 0, 8);
        this.finals = fin;
    }

    return this.Encode(this.finals.state, 16);
}

public static String asHex(byte[] hash) {
    char[] buf = new char[hash.length * 2];
    int i = 0;

    for(int var3 = 0; i < hash.length; ++i) {
        buf[var3++] = HEX_CHARS[hash[i] >>> 4 & 15];
        buf[var3++] = HEX_CHARS[hash[i] & 15];
    }

    return new String(buf);
}
```

```
public String asHex() {
    return asHex(this.Final());
}

public static final synchronized boolean initNativeLibrary() {
    return initNativeLibrary(false);
}

public static final synchronized boolean initNativeLibrary(boolean disallow_lib_loading) {
    if (disallow_lib_loading) {
        native_lib_init_pending = false;
    } else {
        _initNativeLibrary();
    }

    return native_lib_loaded;
}

private static final synchronized void _initNativeLibrary() {
    if (native_lib_init_pending) {
        native_lib_loaded = _loadNativeLibrary();
        native_lib_init_pending = false;
    }
}

private static final synchronized boolean _loadNativeLibrary() {
    try {
        String prop = System.getProperty("com.twmacinta.util.MD5.NO_NATIVE_LIB");
        if (prop != null) {
            prop = prop.trim();
            if (prop.equalsIgnoreCase("true") || prop.equals("1")) {
                return false;
            }
        }

        prop = System.getProperty("com.twmacinta.util.MD5.NATIVE_LIB_FILE");
        File f;
        if (prop != null) {
            f = new File(prop);
            if (f.canRead()) {
                System.load(f.getAbsolutePath());
```

```
            return true;
        }
    }

    String os_name = System.getProperty("os.name");
    String os_arch = System.getProperty("os.arch");
    if (os_name == null || os_arch == null) {
        return false;
    }

    os_name = os_name.toLowerCase();
    os_arch = os_arch.toLowerCase();
    File arch_lib_path = null;
    String arch_libfile_suffix = null;
    if (os_name.equals("linux") && (os_arch.equals("x86") || os_arch.equals("i386") || os_arch.equals("i486") || os_arch.equals("i586") || os_arch.equals("i686"))) {
        arch_lib_path = new File(new File(new File("lib"), "arch"), "linux_x86");
        arch_libfile_suffix = ".so";
    } else if (os_name.equals("linux") && os_arch.equals("amd64")) {
        arch_lib_path = new File(new File(new File("lib"), "arch"), "linux_amd64");
        arch_libfile_suffix = ".so";
    } else if (os_name.startsWith("windows ") && (os_arch.equals("x86") || os_arch.equals("i386") || os_arch.equals("i486") || os_arch.equals("i586") || os_arch.equals("i686"))) {
        arch_lib_path = new File(new File(new File("lib"), "arch"), "win32_x86");
        arch_libfile_suffix = ".dll";
    } else if (os_name.startsWith("windows ") && os_arch.equals("amd64")) {
        arch_lib_path = new File(new File(new File("lib"), "arch"), "win_amd64");
        arch_libfile_suffix = ".dll";
    } else if (os_name.startsWith("mac os x") && os_arch.equals("ppc")) {
        arch_lib_path = new File(new File(new File("lib"), "arch"), "darwin_ppc");
        arch_libfile_suffix = ".jnilib";
    } else if (os_name.startsWith("mac os x") && (os_arch.equals("x86") || os_arch.equals("i386") || os_arch.equals("i486") || os_arch.equals("i586") || os_arch.equals("i686"))) {
        arch_lib_path = new File(new File(new File("lib"), "arch"), "darwin_x86");
        arch_libfile_suffix = ".jnilib";
    } else if (os_name.startsWith("mac os x") && os_arch.equals("x86_64")) {
        arch_lib_path = new File(new File(new File("lib"), "arch"), "darwin_x86_64");
        arch_libfile_suffix = ".jnilib";
    } else if (os_name.equals("freebsd") && (os_arch.equals("x86") || os_arch.equals("i386") || os_arch.equals("i486") || os_arch.equals("i586") || os_arch.equals("i686"))) {
        arch_lib_path = new File(new File(new File("lib"), "arch"), "freebsd_x86");
```

```
            arch_libfile_suffix = ".so";
        } else if (os_name.equals("freebsd") && os_arch.equals("amd64")) {
            arch_lib_path = new File(new File(new File("lib"), "arch"), "freebsd_amd64");
            arch_libfile_suffix = ".so";
        } else {
            arch_libfile_suffix = ".so";
        }

        String fname = "MD5" + arch_libfile_suffix;
        if (arch_lib_path != null) {
            f = new File(arch_lib_path, fname);
            if (f.canRead()) {
                System.load(f.getAbsolutePath());
                return true;
            }
        }

        f = new File(new File("lib"), fname);
        if (f.canRead()) {
            System.load(f.getAbsolutePath());
            return true;
        }

        f = new File(fname);
        if (f.canRead()) {
            System.load(f.getAbsolutePath());
            return true;
        }
    } catch (SecurityException var7) {
    } catch (UnsatisfiedLinkError var8) {
        var8.printStackTrace();
    }

    return false;
}

public static byte[] getHash(File f) throws IOException {
    if (!f.exists()) {
        throw new FileNotFoundException(f.toString());
    } else {
        Object close_me = null;
```

```
        try {
            long buf_size = f.length();
            if (buf_size < 512L) {
                buf_size = 512L;
            }

            if (buf_size > 65536L) {
                buf_size = 65536L;
            }

            byte[] buf = new byte[(int)buf_size];
            MD5InputStream in = new MD5InputStream(new FileInputStream(f));

            while(in.read(buf) != -1) {
            }

            in.close();
            return in.hash();
        } catch (IOException var7) {
            if (close_me != null) {
                try {
                    ((InputStream)close_me).close();
                } catch (Exception var6) {
                }
            }

            throw var7;
        }
    }
}

public static boolean hashesEqual(byte[] hash1, byte[] hash2) {
    if (hash1 == null) {
        return hash2 == null;
    } else if (hash2 == null) {
        return false;
    } else {
        int targ = 16;
        if (hash1.length < 16) {
            if (hash2.length != hash1.length) {
```

```
            return false;
          }

          targ = hash1.length;
        } else if (hash2.length < 16) {
          return false;
        }

        for(int i = 0; i < targ; ++i) {
          if (hash1[i] != hash2[i]) {
            return false;
          }
        }

        return true;
      }
    }
}
```

```
/* crc32.js (C) 2014-present SheetJS -- http://sheetjs.com */
/* vim: set ts=2: */
var CRC32;
(function (factory) {
        if(typeof DO_NOT_EXPORT_CRC === 'undefined') {
                if('object' === typeof exports) {
                        factory(exports);
                } else if ('function' === typeof define && define.amd) {
                        define(function () {
                                var module = {};
                                factory(module);
                                return module;
                        });
                } else {
                        factory(CRC32 = {});
                }
        } else {
                factory(CRC32 = {});
        }
}(function(CRC32) {
```

```
CRC32.version = '0.4.0';
/* see perf/crc32table.js */
function signed_crc_table() {
	var c = 0, table = new Array(256);

	for(var n =0; n != 256; ++n){
		c = n;
		c = ((c&1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
		c = ((c&1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
		c = ((c&1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
		c = ((c&1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
		c = ((c&1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
		c = ((c&1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
		c = ((c&1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
		c = ((c&1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
		table[n] = c;
	}

	return typeof Int32Array !== 'undefined' ? new Int32Array(table) : table;
}

var table = signed_crc_table();
/* charCodeAt is the best approach for binary strings */
var use_buffer = typeof Buffer !== 'undefined';
function crc32_bstr(bstr) {
	if(bstr.length > 32768) if(use_buffer) return crc32_buf_8(new Buffer(bstr));
	var crc = -1, L = bstr.length - 1;
	for(var i = 0; i < L;) {
		crc =  table[(crc ^ bstr.charCodeAt(i++)) & 0xFF] ^ (crc >>> 8);
		crc =  table[(crc ^ bstr.charCodeAt(i++)) & 0xFF] ^ (crc >>> 8);
	}
	if(i === L) crc = (crc >>> 8) ^ table[(crc ^ bstr.charCodeAt(i)) & 0xFF];
	return crc ^ -1;
}

function crc32_buf(buf) {
	if(buf.length > 10000) return crc32_buf_8(buf);
	for(var crc = -1, i = 0, L=buf.length-3; i < L;) {
		crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
		crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
		crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
```

```
		crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
	}
	while(i < L+3) crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
	return crc ^ -1;
}

function crc32_buf_8(buf) {
	for(var crc = -1, i = 0, L=buf.length-7; i < L;) {
		crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
		crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
		crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
		crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
		crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
		crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
		crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
		crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
	}
	while(i < L+7) crc = (crc >>> 8) ^ table[(crc^buf[i++])&0xFF];
	return crc ^ -1;
}

/* much much faster to intertwine utf8 and crc */
function crc32_str(str) {
	for(var crc = -1, i = 0, L=str.length, c, d; i < L;) {
		c = str.charCodeAt(i++);
		if(c < 0x80) {
			crc = (crc >>> 8) ^ table[(crc ^ c) & 0xFF];
		} else if(c < 0x800) {
			crc = (crc >>> 8) ^ table[(crc ^ (192|((c>>6)&31))) & 0xFF];
			crc = (crc >>> 8) ^ table[(crc ^ (128|(c&63))) & 0xFF];
		} else if(c >= 0xD800 && c < 0xE000) {
			c = (c&1023)+64; d = str.charCodeAt(i++) & 1023;
			crc = (crc >>> 8) ^ table[(crc ^ (240|((c>>8)&7))) & 0xFF];
			crc = (crc >>> 8) ^ table[(crc ^ (128|((c>>2)&63))) & 0xFF];
			crc = (crc >>> 8) ^ table[(crc ^ (128|((d>>6)&15)|((c&3)<<4))) & 0xFF];
			crc = (crc >>> 8) ^ table[(crc ^ (128|(d&63))) & 0xFF];
		} else {
			crc = (crc >>> 8) ^ table[(crc ^ (224|((c>>12)&15))) & 0xFF];
			crc = (crc >>> 8) ^ table[(crc ^ (128|((c>>6)&63))) & 0xFF];
			crc = (crc >>> 8) ^ table[(crc ^ (128|(c&63))) & 0xFF];
		}
```

Додаток Б

Ксерокопії наукових публікацій, виконаних при роботі над кваліфікаційною роботою магістра

Перелік наукових публікацій:

1. Заровний В.І. Метод шифрованої передачі даних між хмарними підпросторами / Заровний В.І., Скрипник Т.К. // Збірник наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021». – Хмельницький, 2021. – С. 335-337.

Міністерство освіти і науки України
Хмельницький національний університет

ЗБІРНИК НАУКОВИХ ПРАЦЬ

за матеріалами XIII Всеукраїнської науково-практичної конференції
«Актуальні проблеми комп'ютерних наук АПКН-2021»

15-16 жовтня 2021

Хмельницький 2021

Галкіна Р. І., Багрій Р. О., Скрипник Т. К.
Застосування адаптивного підходу для реалізації системи опитувань та тестувань 306

Гринь С. С., Пивовар О. С., Таранчук А. А.
Забезпечення прихованості дії та криптографічного захисту аналогових сигналів в хаотичній системі зв'язку 309

Данчук С. В., Багрій Р. О.
Технологія автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики 312

Длугунович Н. А.
Інформаційна технологія фінансового моделювання для розвитку малого підприємництва 316

Дрозд А. І., Форкун Ю. В.
Метод розподілу обчислювальних ресурсів для обробки розподілених потоків даних 319

Дудар О. В., Міхалевський В. Ц., Скрипник Т. К.
Інформаційна система для забезпечення підтримки екологічної рівноваги 321

Єфімчук А. С., Скрипник Т. К., Мазурець О. В., Молчанова М. О.
Автоматизований розподіл процесів при управлінні ІТ-проєктами в складних критично-безпекових умовах 324

Житкевич В. В., Медведчук В. Ю.
Метод відновлення пошкоджених растрових зображень 332

Заровний В. І., Скрипник Т. К.
Методи шифрованої передачі даних між хмарними підпросторами 335

Кудрявцев В. В., Форкун Ю. В.
Аналіз та застосування методів оптимізації швидкодії та відмовостійкості програмних продуктів 338

Курдибаха А. В., Мазурець О. В., Собко О. В., Молчанова М. О.
Інформаційна технологія оцінювання діяльності сімейного лікаря за даними прийомів 340

Лаврентій А. А, Петровський С. С.
Метод оцінювання наповненості дистанційних курсів предметів у школі 349

Левченко Т. В., Блажук В. Д., Молчанова М. О., Собко О. В.
Метод оптимізації транспортних перевезень засобами біологічної метаевристики 352

УДК 004.4

Заровний В. І., Скрипник Т. К.

Хмельницький національний університет

МЕТОДИ ШИФРОВАНОЇ ПЕРЕДАЧІ ДАНИХ МІЖ ХМАРНИМИ ПІДПРОСТОРАМИ

В сучасному світі хмарних розробок, все більше і більше проектів переносяться на хмарні сервіси. Підхід передачі даних між сервісами, є вразливим для перехоплення даних, та використовуватися зловмисниками. З огляду на цю особливість, є дуже важливим, мати змогу шифрувати дані, щоб зловмисники, навіть коли їх перехоплять, не мали можливості зрозуміти, які саме дані передавалися між сервісами. Для того, щоб зробити усунення проблеми, необхідно використати криптографічні ключі при шифруванні сигналів. У роботі представлено новий алгоритм шифрування, створений з урахуванням підходів інших алгоритмів шифрування у вигляді бібліотеки. Бібліотека містить детальний аналіз наявних алгоритмів шифрування, та дозволяє розробникам, користуватися всіма перевагами шифрування даних, не поглиблюючись в призначення кожного алгоритму, так як вся ця відповідальність перекладається на розроблену бібліотеку. Здатність бібліотеки автоматично визначати алгоритми шифрування, пришвидшує процес розробки, з урахуванням безпеки даних, та дозволяє мінімізувати випадки неправильного підбору алгоритму шифрування.

In today's world of cloud development, more and more projects are being transferred to cloud services. The data transfer approach between services is vulnerable to data interception and being used by attackers. Given this feature, it is very important to be able to encrypt the data so that attackers, even when intercepted, do not have the opportunity to understand what data was transmitted between services. In order to fix the problem, it is necessary to use cryptographic keys when encrypting signals. The paper presents a new encryption algorithm, created taking into account the approaches of other encryption algorithms in the form of a library. The library contains a detailed analysis of existing encryption algorithms, and allows developers to enjoy all the benefits of data encryption without delving into the purpose of each algorithm, as all this responsibility is transferred to the developed library. The ability of the library to automatically detect encryption algorithms, speeds up the development process, taking into account data security, and minimizes cases of incorrect selection of the encryption algorithm.

В сучасному світі хмарних розробок, все більше і більше проектів переносяться на хмарні сервіси, де вони можуть, як завгодно масштабувати, в залежності від навантаження. Даний підхід вимагає мікро сервісної архітектури проектів, де сервіси спілкуються між собою, шляхом передачі даних через інтернет. Підхід, є вразливим для перехоплення даних, які передаються між сервісами, та використовуватися зловмисниками. З огляду на цю особливість, є дуже важливим, мати змогу шифрувати дані, щоб зловмисники, навіть коли їх перехоплять, не мали можливості зрозуміти, які саме дані передавалися між сервісами. Це особливо є

актуальним у сфері онлайн оплати, яка також стає все більш розповсюдженою. Шифрування даних кредитних карток, та інформації про оплату, унеможливлює спроби шахраїв, використовувати персональні дані користувачів. Система шифрування дозволяє зашифрувати дані, для відправки мережею.

Можна розглянути одну з найбільш розповсюджених проблем вразливостей. Розглянемо ситуацію коли кібер злочинець може підміняти запити клієнта, до сервера, та відправляти замість них свої. В даному випадку, ми можемо брати школу, в якій є своя система безпеки, пожежна система, температура приміщень, вологість. Вся ця інформація передається на сервер, за яким сидить оператор, і у випадку виникнення критичної ситуації по цих датчиках, він виконує певні дії. Кіберзлочинець шляхом підміни інформації, стосовно температури приміщень, чи стану газової безпеки, може відправити хибні сигнали на сервер і таким чином приховати дійсне становище речей.

Також прикладом серйозних порушень може бути ваша сигналізація яка має багато датчиків, закритих дверей, закритих вікон, та всю інформацію відправляє, на спеціальний хост який її обробляє. Якщо зловмисники зможуть відправляти свою інформацію про те, що двері закриті, вони зможуть спокійно відкрити двері, та проникнути в вашу квартиру. З огляду на це все, є вкрай важливим шифрувати сигнали, які передаються між клієнтом та сервером, щоб уникнути ситуації, коли не авторизований користувач, тобто кіберзлочинність, буде відправляти інші сигнали, які не є правдивими.

Для того, щоб зробити усунення проблеми, необхідно використати криптографічні ключі при шифруванні сигналів.

Шифрування – це перетворення інформації з метою приховування від неавторизованих осіб, з наданням, в цей же час, авторизованим користувачам доступу до неї. Головним чином, шифрування служить для дотримання конфіденційності інформації, що передається. Важливою особливістю будь-якого алгоритму шифрування є використання ключа, який підтверджує вибір конкретного перетворення із сукупності можливих для даного алгоритму.

Принцип захисту шифрування від зловмисника на рисунку 1.

Система представлена у вигляді бібліотеки API (прикладний програмний інтерфейс). Спрощено — це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення.

У роботі представлено новий алгоритм шифрування, створений з урахуванням підходів інших алгоритмів шифрування. Додатково бібліотека містить детальний аналіз наявних алгоритмів шифрування, та можливості їх використання. До аналізу були взяті наступні алгоритми:

MD5, CRC32, SHA1, SHA-2. Запропоноване рішення можна використовувати в будь-якій сфері, де використовується передача даних.

Головними перевагами, є такі сфери як передача даних між хмарними сервісами, а саме передача даних між мікро сервісами, використання у передачі, персональних даних. Аналіз вказаних алгоритмів дозволить краще розробити метод шифрування, з урахуванням нюансів кожного підходу.

Рисунок 1 – Захист шифрування від зловмисників

Достовірність роботи системи перевіряється проведенням системного аналізу, та перевіркою системи, за допомогою тестування.

Є кілька ключових факторів що дозволяють визначити правильність виконання поставлених задач, системою шифрування. Також зашифровані дані повинні перевірятися на криптостійкість - це характеристика шифру, визначальна його стійкість до дешифрування. Звичайно ця характеристика визначається періодом часу, необхідним для дешифрування. Добре розроблена система шифрування, має легко використовуватися як інтерфейс для будь-якої системи яка планує під'єднати цю бібліотеку. Шифрування можна назвати правильним, якщо після передачі довільної структури даних відбулася видозміна цих даних, за допомогою ключового слова, після чого, їх можна повернути до попереднього нормального виду (дешифрувати), за допомогою того самого ключового слова.

Результатом виконаної роботи, є бібліотека, яка дозволяє розробникам, користуватися всіма перевагами шифрування даних, не поглиблюючись в призначення, та швидкодію, кожного алгоритму шифрування, так як вся ця відповідальність перекладається на розроблену бібліотеку. Здатність бібліотеки автоматично визначати алгоритми шифрування, пришвидшує процес розробки, з урахуванням безпеки даних, та дозволяє мінімізувати випадки неправильного підбору алгоритму шифрування.

Перелік посилань

1. Бібліографія правила оформлення С.5 [Електронний ресурс]. – Режим доступу: https://physics.lnu.edu.ua/wp-content/uploads/bibliografia.pdf
2. Рубрикація тексту [Електронний ресурс]. – Режим доступу: https://studfile.net/preview/5585989/page:5/
3. Наукова публікація: поняття, функції, основні види [Електронний ресурс]. – Режим доступу: https://lektsii.org/12-26321.html
4. Робота над написанням наукових статей, монографій, наукових доповідей і повідомлень [Електронний ресурс]. – Режим доступу: https://studfile.net/preview/6872312/

Додаток В

Презентаційний матеріал

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
НА ТЕМУ:

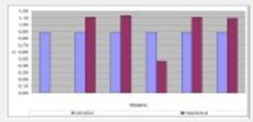
МЕТОД ШИФРОВАНОЇ ПЕРЕДАЧІ ДАНИХ МІЖ ХМАРНИМИ ПІДПРОСТОРАМИ

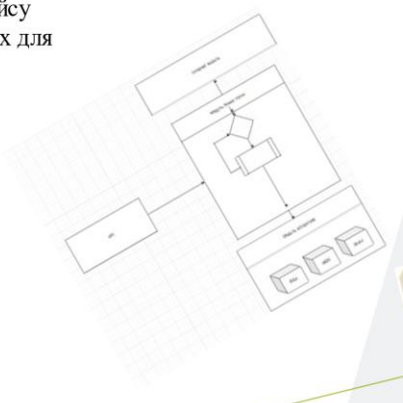
Виконав:
студент 2 курсу, група КНм-20-1 В.І.Заровний

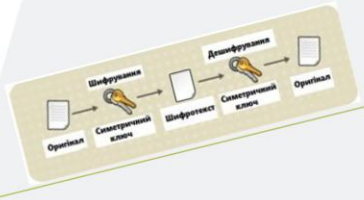
Керівник:
к.ф-м.н., доцент кафедри КН В.Ц.Міхалевський

План

1. Мета роботи,
2. Постановка задачі,
3. Новизна
4. Опис структури та інтерфейсу
5. Модель перетворення даних для паралельних обчисленнь
6. Результати обчислень
7. Висновки



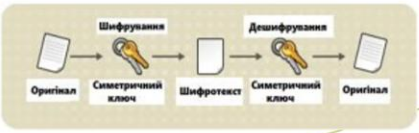




Мета роботи

В сучасному світі хмарних розробок все більше сервісів переносяться на хмарні платформи, що дозволяє їх масштабувати в залежності від навантаження на сервіси. Даний підхід вимагає мікросервісної архітектури проектів, де багато сервісів взаємодіють між собою, шляхом передачі даних через інтернет.

Для передачі даних використовується HTTP, що уможливлює перехоплення даних, які передаються між сервісами, та може використовуватися зловмисниками. Важливо, щоб у разі перехоплення даних зловмисники не мали можливості зрозуміти, які саме дані передавалися між сервісами. Це особливо є актуальним у сфері онлайн оплати.



Постановка задачі

Для досягнення поставленої мети, поставлені наступні **задачі**:

- дослідження необхідності використання шифрування даних для передачі їх між хмарними сервісами ;
- розробка інформаційної системи передачі даних між хмарними підпросторами .

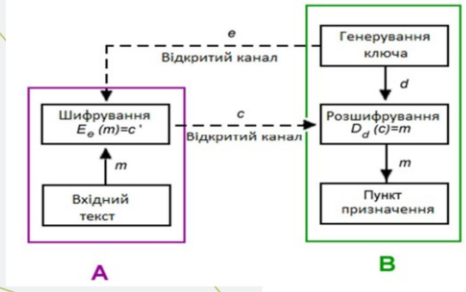
При цьому необхідно вирішити такі **підзадачі**

- дослідження предметної області – різних підпросторів збереження даних ;
- розробка алгоритмів та структури системи передачі даних ;
- оцінка ризиків передачі шифрованих та нешифрованих даних ;
- надання алгоритму використання ключів для шифрування ;
- формування переліку програмних кодів як бібліотеки для використання іншими системами ;
- надання рекомендаційних підходів для шифруванні даних ;
- тестування розробленої системи .

Об'єктом дослідження є інформаційна технологія передачі шифрованих даних при різних алгоритмах шифрування та підбору значень між хмарними підпросторами

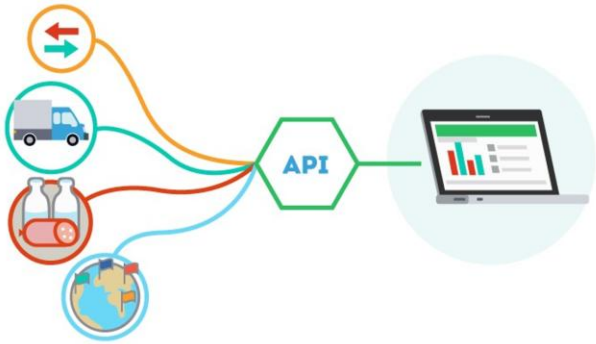
Предметом дослідження є набори даних, отриманих із параметрів переданих у зовнішній інтерфейс бібліотеки, це можуть бути різнотипні дані, такі як об'єкт, список, рядок, число, тощо.

Новизна



- Запропоновано метод, що дозволяє обрати найоптимальніший алгоритм шифрування для передачі його через хмарні сервіси
- Набули подальшого розвитку існуючі методи шифрування даних в частині отримання та обробки даних
- Розроблено алгоритм вибору шифрування даних, який пришвидшує та уточнює процес шифрування даних
- Вдосконалено процес прийняття рішення стосовно передачі даних між хмарними сервісами

Опис структури та інтерфейсу

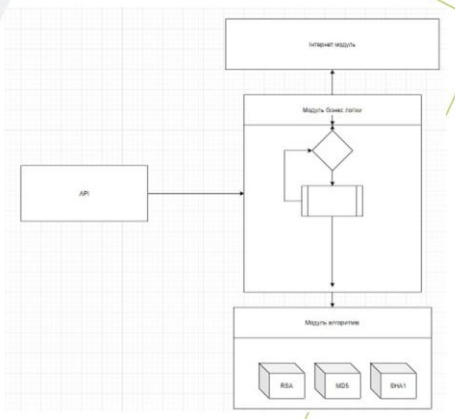


Система шифрованої передачі даних, представлена у вигляді зовнішнього інтерфейсу бібліотеки.

Цей підхід передбачає використання системи у вигляді бібліотеки. Реалізація системи у вигляді бібліотеки, дозволяє забезпечити зручне підключення, зберегти функціональність від зовнішнього втручання.

Структура системи

- API модуль
- Бізнес логіка
- Інтернет модуль
- Модуль алгоритмів



Структура системи

«API модуль»: головний компонент, який об'єднує всі інші компоненти системи. Ціль даного модуля є відображення точки входу, яка дозволяє комунікувати з функціональністю бібліотеки

«Модуль алгоритмів»: ідея даного модуля заключається в тому, що він містить список доступних алгоритмів, які будуть вибиратися для шифрування даних. Цей модуль налічує всі можливі алгоритми шифрування, та дешифрування, для того щоб інша функціональність могла з ним співпрацювати

«Бізнес модуль»: даний модуль містить в собі всю бізнес логіку пов'язану з роботою бібліотеки, перевірку вихідних параметрів, та перевірка їх відповідності шаблону. Модуль містить базову бізнес логіку, яка визначає який саме алгоритми шифрування необхідно використовувати

«Інтернет модуль»: займається перевіркою версії алгоритмів шифрування, які знаходяться в бібліотеці, так за допомогою цього модуля, відбувається перевірка, чи є сама остання версія шифрування в даній бібліотеці,

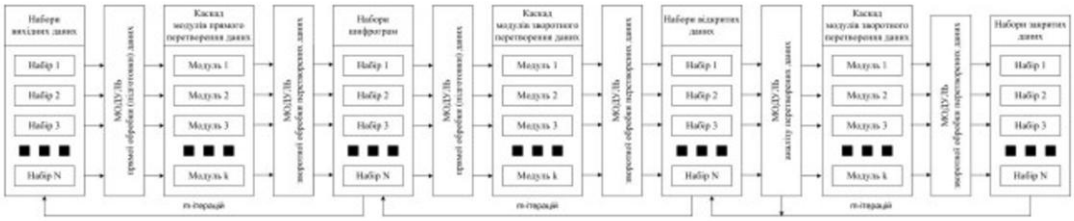
Вибір алгоритма шифрування

Бібліотека налічує декілька алгоритмів шифрування, що дозволяє вибрати найоптимальніший з них, для конкретної ситуації.

Шляхом проходження вказаних перевірок відбувається відбір серед алгоритмів, які найкраще підійдуть до певної системи та певного типу об'єктів.

Модель перетворення даних для паралельних обчисленнь

Для того щоб не впливати на алгоритми шифрування, забезпечити надійний рівень захисту даних та привести вихідну модель до вигляду який дозволяє використання паралельних обчислень, було створено узагальнену модель яка частково відображена на рисунку



Модель перетворення даних для паралельних обчисленнь

Використання даних функції та підходів значно впливає на час функціонування вихідної модель. Як результат кінцевий час для виконання обробки всіх даних навіть у багато потоковому режимі може бути коригувальний і оцінений виразом 4.1 при цьому самі механізми шифрування не зазнають жодних змін.

$$S_p = \frac{T_0}{T_p} = \frac{1}{k} \cdot BQ \cdot f_T(f_{1,2}) \approx 1 - wp \cdot \left(1 - \frac{1}{p}\right)$$

де S – коефіцієнт прискорення програмного коду,

To – час виконання програмного коду в однопотоковому режимі,

Tp – час виконання програмного коду в багатопотоковому режимі,

k – кількість обчислювальних потоків,

BQ –кількість блоків з даними,

fT – функція оцінки часу виконання програмного коду,

f1,2 – процедури шифрування та дешифрування,

wp – відносна частка паралельно виконуваних операцій,

p – кількість ядер центрального процесора.

Результати експерементів

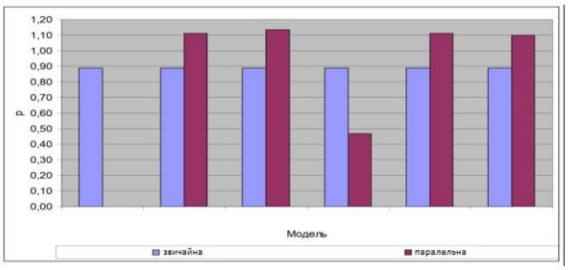
Було проведено експерименти для звичайного режим обчислень за допомогою базової моделі. Результати дешифрування та шифрування даних Зображено у таблицях

Номер експерименту	Тип даних			
	Відкриті	Закриті	Відкриті	Закриті
	Розмір ключа, біт			
	1024		8192	
	Час дешифрування, с			
1	386,036	7801,760	1456,720	29440,232
2	396,448	8012,200	663,536	13410,088
3	355,436	7183,320	697,820	14102,984
4	378,704	7653,600	736,000	14874,564
5	303,936	6142,480	820,056	16573,380
6	402,164	8127,720	804,964	16268,320
7	415,784	8403,000	757,724	15313,608
8	409,260	8271,120	831,276	16800,152
9	326,080	6590,080	919,624	18585,600
10	267,840	5413,040	937,528	18947,472

Були проведені експерименти за допомогою модифікації моделі обробки даних а саме розподілу її на окремі блоки та використання визначених конфігурацій для паралельного обчислення даних. Результати експериментів продемонстровано в таблицях

Номер експерименту	Тип даних			
	Відкриті	Закриті	Відкриті	Закриті
	Розмір ключа, біт			
	1024		8192	
	Час дешифрування, с			
1	163,300	3300,300	309,980	6265,390
2	162,600	3286,330	321,360	6309,970
3	156,190	2752,580	317,680	6761,950
4	130,240	2632,320	309,360	6799,400
5	150,540	3042,420	265,570	8034,680
6	154,580	3124,240	318,980	11429,71
7	150,960	3051,040	317,910	8566,110
8	151,750	3066,870	309,030	5799,390
9	135,230	2732,990	306,360	6437,790
10	150,180	3035,200	295,740	5688,390

Результати експерементів



- В подальшому на базі отриманих експериментальних даних для визначення ефективності рішення було визначено що результати обчислень показують прискорення , при цьому середній коефіцієнт вище в порівнянні з початковим та стандартним варіантом моделі, Для того щоб описати отримані вище результати було виконано розрахунок відносної частки паралельно виконуваних моделей за рахунок формули для коефіцієнта прискорення програмного коду

Висновок

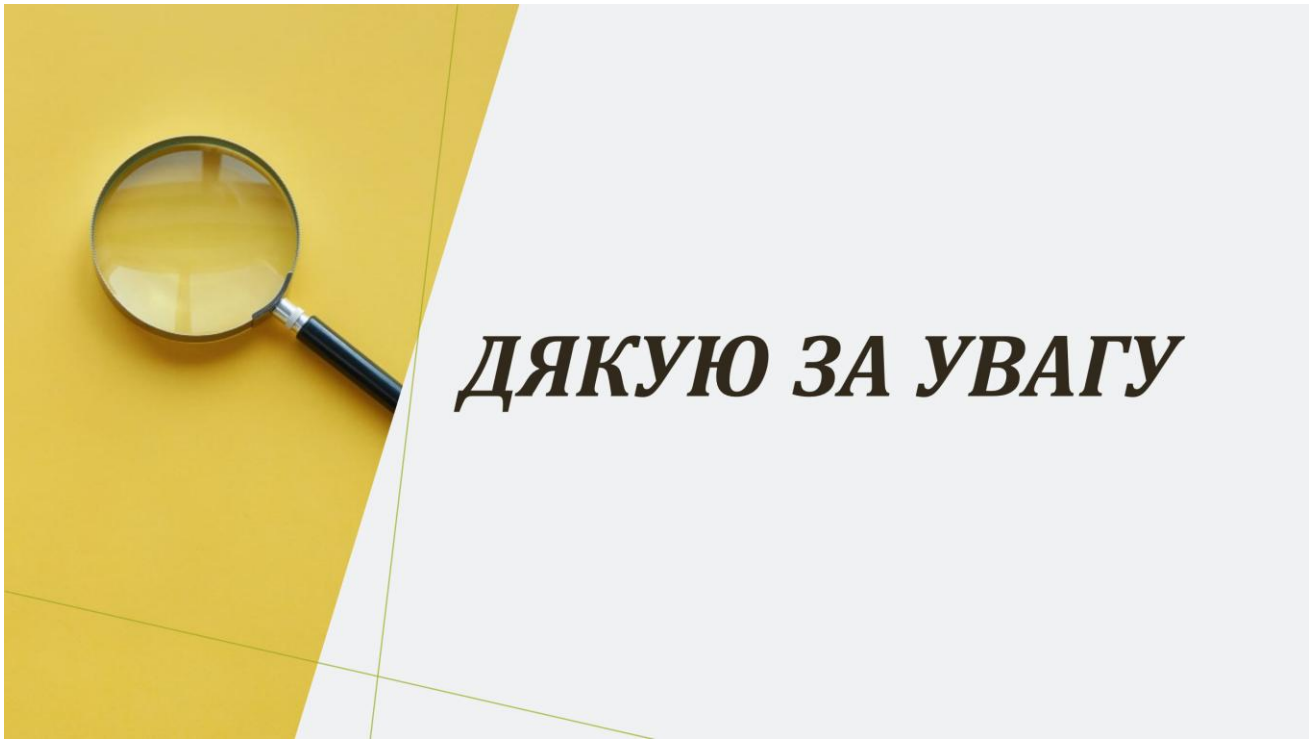
Аналіз небезпек які існують через відсутність зашифрованих каналів звязку, вказує на те, що правильний підбір алгоритмів шифрування, та правильне створення структури доступу до даних, дозволяє забезпечити коректну та безперебійну роботу кожної системи.

Проведено дослідження причин зниження ефективності використання паралельних обчислень при досить великих файлах коли завантаженість системи є понаднормовою..

примітивний метод балансування навантаження який полягає у визначенні апаратних можливостей та використання їх для розподілу вхідних даних на блоки. В результаті проведених експериментів подібний підхід показав зростання швидкості шифрування даних в 2,5 рази у порівнянні з швидкістю роботи

Розроблено версію з використанням блоків яка грунтується на паралельних обчисленнях з попередньою обробкою вихідних даних, а саме розподілом їх на блоки. Як результат це також призвело до скорочення часу обробки моделей у порівнянні з швидкістю асиметричного підходу.

Отримані знання стосовно роботи алгоритмів, надалі можна буде використовувати дані алгоритми та підходи для побудови інших систем які будуть спілкуватися між собою за рахунок закритих каналів зв'язку


ДЯКУЮ ЗА УВАГУ

Tue Nov 23 19:14:26 EET 2021, Петровський Сергій Степанович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 27.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 6%**

	Документ		Суммарное совпадение по Базе Данных	
ID: 97109 Название: Метод шифрованої передачі даних між хмарними підпросторами Добавлено в БД: 2021-11-23 Авторы: В.І. Заровний Руководители: В.Ц. Міхалевський Консультанты: Опоненты:	Символы	Лексемы	Символы	Лексемы
	100848	621	30004 (30%)	186 (30%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы
95888	Название: ЗВІТ з науково-дослідної практики Добавлено в БД: 2021-09-29 Авторы: Заровний В.І. Руководители: Скрипник Т.К. Консультанты: Опоненты:	27663 (27.0%)	160 (26.0%)

UNICHECK

Ім'я користувача:
Кафедра КН

ID перевірки:
1009320316

Дата перевірки:
23.11.2021 20:00:49 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
25.11.2021 22:51:17 EET

ID користувача:
100005671

Назва документа: **Заровний_В_КНм_20_1_Маг_робота_Без_додатка**

Кількість сторінок: **84** Кількість слів: **14921** Кількість символів: **115549** Розмір файлу: **1.96 MB** ID файлу: **1009344561**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

7%
Схожість

Найбільша схожість: **2.04%** з Інтернет-джерелом (http://llt.multycourse.com.ua/ua/print_page/module/22)

5.15% Джерела з Інтернету	261	Сторінка 86
4.09% Джерела з Бібліотеки	100	Сторінка 91

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	2
Підозріле форматування	13 сторінок

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНИХ НАУК

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ МАГІСТРА ДО ЗАХИСТУ ЗА РЕЗУЛЬТАТАМИ АНАЛІЗУ ЗВІТУ ПОДІБНОСТІ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Метод шифрованої передачі даних між хмарними підпросторами

Автор: Заровний В.І., група КНм-20-1

Спеціальність: 122 – Комп'ютерні науки

Освітня програма: освітньо-професійна

Науковий керівник: к.ф-м.н., доц. кафедри КН Міхалевський В.Ц.

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) за програмою Anti-Plagiarism виявлені 27% запозичень вказують на документ автора роботи Заровного В.І. та містять ЗВІТ з науково-дослідної практики.

2) За програмою UNICHECK виявлені 7% запозичень є фрагментарними – містять поширені конструкції, загальновідомі терміни, скорочення та визначення.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ідентичності/ схожості, складає 27% і 7% відповідно, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ В. Ц. Міхалевський

Гарант ОП _____ Р. О. Багрій

Завідувач кафедри КН _____ О. В. Бармак

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МОН УКРАЇНИ

кафедра комп'ютерних наук

ВІДГУК ОПОНЕНТА

на кваліфікаційну роботу магістра

гр. КНм-20-1 Заровного Владислава Ігоровича за темою: *Метод шифрованої передачі даних між хмарними підпросторами*

1. Актуальність обраної теми

В кваліфікаційній роботі магістра було розроблено метод шифрованої передачі даних між хмарними підпросторами. Тема роботи є актуальною на даний час, актуальність детально обґрунтована дослідженнями процесів шифрованої передачі даних між хмарними підпросторами та порівняльним аналізом існуючих алгоритмів шифрування запропонованими технологічними підходами.

2. Відповідність роботи предметній області спеціальності 122 Комп'ютерні науки та загальним вимогам до наукових робіт

Предметна область метода шифрованої передачі даних між хмарними підпросторами дозволяє формалізацією, систематизацією, збір та аналіз інформації про використання алгоритмів шифрування для передачі даних, а розроблена система описує, аналізує та оптимізує архітектурні рішення для визначення найоптимальнішого алгоритму шифрування для отриманих даних.

Кваліфікаційна робота відповідає вимогам стандарту до роботи магістра за спеціальністю 122 Комп'ютерні науки.

3. Повнота розкриття мети та завдань дослідження

Завдання дослідження повністю розкривають мету роботи. Запропоновано механізми оптимізації процесу вибору алгоритму шифрування, розроблено систему забезпечення шифрованої передачі даних між хмарними підпросторами.

4. Наявність наукової новизни

Наукова новизна підтверджена розробкою схеми управління зовнішньою бібліотекою, вдосконаленням та модернізацією принципу застосування алгоритмів шифрування, подальшим розвитком методології паралельного шифрування при розподілу даних на блоки.

5. Зміст кожного розділу роботи

В першому розділі проаналізована предметна область та сформована постановка задачі на розробку системи шифрованої передачі даних.

В другому розділі проаналізовано існуючі технології та запропоновано і описано систему шифрованої передачі даних.

В третьому розділі розроблено програму в середовищі IntelliJ Idea за допомогою мови програмування Java, надано схему архітектури розробленої системи.

В четвертому розділі протестовано і проведено порівняння результатів роботи при прямолінійному використані блоку даних для шифрування та розпаралеленому підході з поділом блоку даних на підблоки і шифрування окремих підблоків. Наведено порівняльні діаграми і зроблена оцінка отриманих результатів.

6. Ступінь розкриття теми роботи

Тема роботи розкрита повністю. Достовірність результатів підтверджена процесом тестування на основі вхідних параметрів для шифрування.

7. Якість оформлення кваліфікаційної роботи

Пояснювальна записка кваліфікаційної роботи магістра оформлена відповідно до норм. Мовних граматичних, синтаксичних помилок не виявлено.

8. Недоліки кваліфікаційної роботи

Явних недоліків в роботі не виявлено. Можна було б провести узагальнення роботи системи шляхом удосконалення інтерфейсу користувача.

9. Загальний висновок (допускається чи не допускається до захисту), якої оцінки заслуговує кваліфікаційна робота.

Рекомендую допустити кваліфікаційну роботу до захисту.

Робота заслуговує на оцінку « *добре* ».

Опонент _____ д.т.н., проф. Валерій Мартинюк

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МОН УКРАЇНИ

кафедра комп'ютерних наук

ВІДГУК НАУКОВОГО КЕРІВНИКА

на кваліфікаційну роботу магістра

гр. КНм-20-1 Заровного Владислава Ігоровича за темою: *Метод шифрованої передачі даних між хмарними підпросторами*

1. Актуальність теми

Актуальність теми обгрунтована в достатній мірі: все більше сервісів переносяться на хмарні платформи, що дозволяє їх масштабувати в залежності від навантаження на сервіси. Дані, що передаються через інтернет, піддаються ризику бути перехопленими зловмисниками, тому виникає необхідність розробки комплексного підходу у визначенні алгоритмів та створенні інформаційної системи для передачі даних між хмарними підпросторами шляхом шифрування даних.

2. Відповідність роботи предметній області спеціальності 122 Комп'ютерні науки та загальним вимогам до наукових робіт

В кваліфікаційній роботі магістра було розроблено метод шифрованої передачі даних між хмарними підпросторами. Тема кваліфікаційної роботи магістра відповідає предметній області спеціальності 122 Комп'ютерні науки та вимогам до кваліфікаційної роботи магістра: 1) Виконавцем спланована і реалізовано процес розробки комп'ютерної систем та програмного забезпечення, проведено тестування та порівняльний аналіз. 2) Виконавець обрав інформаційне середовище розробки та дослідження, що дозволило знайти правильне і ефективне рішення, а запропонований метод дозволяє обрати найоптимальніший алгоритм шифрування для передачі його через хмарні сервіси.

Проаналізовано, оцінено та порівняно різні технології виробничих процесів з метою встановлення пріоритетів у відповідності з критеріями продуктивності та якості, що визначені завданням. Розроблено алгоритм вибору шифрування даних, який пришвидшує та уточнює процес шифрування даних. Вдосконалено процес прийняття рішення стосовно передачі даних між хмарними сервісами.

3. Професійні та особистісні якості магістранта

Магістрант володіє в достатній мірі професійними якостями дослідника: 1) Має здатність збирати, формалізувати, систематизувати і аналізувати потреби та вимоги до комп'ютерної системи, що розробляється. 2) Має здатність формалізувати предметну область проєкту у вигляді відповідної інформаційної моделі.

Серед особистісних якостей магістранта слід виділити відповідальність, ініціативність, цілеспрямованість, здатність навчатися, нестандартність мислення.

4. Ступінь самостійності під час виконання кваліфікаційної роботи

Студент більшу частину роботи виконав самостійно. Особисто магістрантом досліджено предметну область, проведено порівняльний аналіз переваг та недоліків існуючих програмних засобів для шифрування/дешифрування даних; підібрано алгоритм для опрацювання даних; розроблено модель перетворення даних на паралельних обчисленнях; поставлено експеримент, отримано результати, проведено оцінку ефективності розробленої моделі для вирішення завдання роботи.

5. Наукова новизна та оригінальність запропонованих підходів

У виконаній роботі наукова новизна присутня в достатній мірі. Інноваційний підхід проявлено в розробці алгоритму вибору шифрування даних, який пришвидшує та уточнює процес шифрування даних.

Результати дослідження доповідались на 1-й конференції та оприлюднені в 1-х тезах.

6. Ступінь оволодіння методами дослідження

Магістрант в достатній мірі оволодів методами дослідження, які були використані у роботі: порівняння, аналізу, класифікації, узагальнення.

7. Повнота та якість розкриття теми роботи

Тема роботи розкрита достатньо в рамках поставлених завдань: розроблено інформаційну технологію шифрованої передачі даних між хмарними підпросторами; досліджено необхідність використання шифрування даних для передачі їх між хмарними сервісами.

Вивчено і досліджено структури даних, отримані для шифрування, імплементація алгоритмів шифрування та інтерфейс підключення для зовнішньої бібліотеки.

8. Логічність, послідовність, аргументованість, літературна грамотність викладу матеріалу

Магістрант матеріал виклав логічно, послідовно, аргументовано. Наводилися наявні розробки, ставилося задача та послідовно розв'язувалася. Для аргументації отриманих рішень проводилося теоретичне обгрунтування та порівняльний аналіз експериментів.

Літературна та граматична якість матеріалу на достатньому рівні.

9. Можливість практичного застосування кваліфікаційної роботи, окремих її частин

Результати кваліфікаційної роботи магістра можуть застосовуватися на практиці після налаштувань під конкретно запропоновану модель передачі даних між підпросторами. Практична цінність роботи полягає в тому, що результати досліджень мають допустимий рівень швидкодії, а розглянуті набори даних мають застосування в реальному житті.

10. Висновок про можливість допуску кваліфікаційної роботи до захисту, на яку оцінку заслуговує робота

Рекомендую допустити кваліфікаційну роботу магістра до захисту.

Робота заслуговує на оцінку « добре ».

Науковий керівник _____ к.фіз.-мат.н., доц. Віталій Міхалевський