МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
Аліна САВЧЕНКО
«_____» _____ 2022р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИЦІ ОСВІТНЬОГО СТУПЕНЯ **"МАГІСТР"**
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
"ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: «Модифікований метод асиметричного шифрування інформації у системах обміну даними»

Виконавець: студентка УС-212(М) Горовая Наталія Миколаївна
(студент, група, прізвище, ім'я, по батькові)

Керівник: доктор технічних наук професор Віноградов М.А.

Нормоконтролер: Ігор РАЙЧЕВ

Київ – 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Кібербезпеки, комп'ютерної та програмної інженерії*

Кафедра *Комп'ютерних інформаційних* технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«____»_____2022р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студентки

Горової Наталії Миколаївни

(прізвище, ім'я, по батькові)

1.Тема роботи: «Модифікований метод асиметричного шифрування інформації у системах обміну даними» затверджена наказом ректора від 28.09.2022 р. №1774

2. Термін виконання роботи: з 26.09.2022 по 21.11.2022.

3. Вихідні дані до роботи: розробка програми генерації асинхронних ключів для захисту вразливої інформації.

4. Зміст пояснювальної записки: вступ, аналітичний огляд і постановка завдання, розгляд завдання шифрування данних, дослідження технологій та засобів, розробка програмного продукту генерації асинхронних ключів, оцінка якості технології, висновки.

5. Перелік обов'язкового графічного матеріалу: загальний перелік існуючих систем та обробка інформації створеним програмним продуктом. Використання структури проблематики питань шифрування.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Отримання завдання на дипломну роботу, створення плану дипломної роботи та побудова плану-графіку виконання робіт.	26.09.2022 – 28.09.2022	
2.	Огляд та аналіз наукової літератури по темі дипломної роботи та написання Розділу 1.	29.09.2022 – 09.10.2022	
3.	Написання Розділу 2 дипломної роботи.	10.10.2022 –20.10.2022	
4.	Написання Розділу 3 дипломної роботи. Завершення створення пояснювальної записки дипломної роботи.	21.10.2022 – 31.10.2022	
5.	Оформлення та друк пояснювальної записки.	01.11.2022 – 07.11.2022	
6.	Створення презентації, доповіді та підготовка до захисту дипломної роботи.	08.11.2022 – 15.11.2022	
7.	Підготовка матеріалів дипломної роботи для передачі секретарю ДЕК (папка, конверт, диск із файлом диплому, рецензія, відгук).	16.11.2022 – 18.11.2022	

7. Дата видачі завдання: «26» вересня 2022 р.

Керівник дипломної роботи _____ Микола ВІНОГРАДОВ
(підпис керівника) (П.І.Б.)

Завдання прийняла до виконання _____ Наталія ГОРОВАЯ
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Модифікований метод асиметричного шифрування інформації у системах обміну даними» містить: 93 сторінок, 6 рисунків, 4 таблиці, 22 літературних джерел.

Ключові слова: ШИФРУВАННЯ, ГЕНЕРАЦІЯ, АВТЕНТИФІКАЦІЯ, ЗАХИСТ, ІНФОРМАЦІЯ.

Актуальність. У сучасній екосистемі глобальної економіки даних компанії збирають і зберігають велику кількість конфіденційної інформації про людей у своєму IT-середовищі. Недбалість у захисті конфіденційних даних часто наражає їх на підвищені ризики витоку даних

Метою дипломної роботи є покращення безпеки вразливих ресурсів.

Для реалізації мети були поставлені такі **завдання**:

1. проаналізувати наукову літературу в області шифрування з метою вивчення;

2. проаналізувати існуючі способи шифрування;

3. розробити програму, що реалізує генерацію асинхронних ключів;

4. запропонувати можливі способи використання шифрування в обміні вразливими даними.

Об’єкт дослідження: утиліти для генерації асинхронних ключів.

Предмет дослідження: система генерації асинхронних ключів для захисту вразливої інформації.

Методи дослідження, технічні та програмні засоби: розробка програмних бібліотек, порівняльний аналіз, обробка літературних джерел.

Отримані результати та їх новизна: запропоновано утиліту шифрування та генерації ключів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

AES	Advanced Encryption Standard
ISO	International Organization for Standardization
DES	Data Encryption Standard
VPN	Virtual Private Network
RSA	Rivest, Shamir, Adleman
FPE	format-preserving encryption

ВСТУП

У сучасній екосистемі глобальної економіки даних компанії збирають і зберігають велику кількість конфіденційної інформації про людей у своєму IT-середовищі. Недбалість у захисті конфіденційних даних часто наражає їх на підвищені ризики витоку даних, витрати на які , згідно з останніми даними , зросли до 4,24 мільйона доларів збитку. Порушення даних відрізняється від розкриття даних. Порушення вимагає, щоб зловмисник отримав доступ до даних, а розкриття даних означає, що дані не були належним чином захищені.

Крім високої вартості витоку даних, негативна рецензія ЗМІ, шкода репутації, простою роботи та втрата даних є додатковими згубними наслідками, які виникають після більшості інцидентів злому. Збої процесів, які залишають дані відкритими в першу чергу, зрештою спричиняють більшість порушень даних.

У проведеному дослідженні була проаналізована наукова та методична література в області шифрування та захисту інформації. З проведеного аналізу літератури були зроблені висновки про принципи роботи та ризики безпеки в інформаційному просторі, та про методи шифрування, їх переваги та недоліки. Таким чином, обробивши теоретичну інформацію про безпеку інформаційного простору, зрозумівши принципи роботи шифрування та створивши програмний продукт для генерації асиметричних ключів та шифрування повідомлень було запропоновано використання утиліти для обміну вразливою інформацією. В подальшому можлива перспектива розробки автономного простору обміну повідомленнями та додавання можливості шифрування та передачі файлів.

РОЗДІЛ 1
РИЗИКИ В ІНФОРМАЦІЙНОМУ ПРОСТОРІ

На сьогоднішній день, все більше і більше вразливої інформації зберігається в всесвітній мережі інтернету(платіжні дані, документи тощо), оскільки час не стоїть на місці, з розвитком технологій поширюються і спроби несанкціонованого доступу до інформації, тому технології захисту необхідно ускладнювати та удосконалювати постійно для кращого захисту вразливої інформації.

1.1. Конфіденційні дані

Конфіденційні дані – це інформація, яку потрібно захистити від несанкціонованого доступу, щоб мінімізувати можливу шкоду для окремих осіб і підприємств. Коли конфіденційні дані потрапляють у чужі руки, конфіденційність людей може бути скомпрометована, особисті дані викрадені або від їх імені може бути вчинено шахрайство. Коли комерційні таємниці, інтелектуальна власність чи інші конфіденційні дані компанії потрапляють у чужі руки, підприємства страждають від втрати конкурентної переваги [1].

Хоча наслідки розкриття конфіденційних даних компанії можуть бути серйозними, ці наслідки обмежені бізнес-рівнем. Оприлюднення індивідуальних даних впливає на людей, що робить належний захист такого типу інформації особливо актуальною проблемою для будь-якого бізнесу.

Велика кількість правил конфіденційності даних спрямована на захист конфіденційних даних, що належать окремим особам. Значна частина витрат, пов’язаних із витоком даних, пов’язана зі штрафними санкціями, судовими процесами та компенсаційними виплатами особам, які постраждали.

Кожен нормативний акт може дещо відрізнятися тим, що він визначає як конфіденційні персональні дані, але деякі спільні риси включають:

• Захищена медична інформація, яка включає історії хвороби, результати аналізів і інформацію про страхування осіб

• Особиста інформація, яка може ідентифікувати особу або використовувати її для визначення особи (наприклад, ім’я, дата народження, номер соціального страхування, номер водійського посвідчення, інформація про банківський рахунок, адреса)

• Біометричні дані, такі як відбитки пальців і сканування сітківки ока

Отже, як саме відбувається розкриття конфіденційних даних? Враховуючи складні IT-середовища, до яких перейшли більшість сучасних компаній, можливо, не надто дивно, що справи йдуть не так, коли намагаються захистити конфіденційну інформацію. Потенційними причинами є відсутність контролю та помилки співробітників. Корисно розділити методи оприлюднення даних залежно від того, чи перебувають дані в стані спокою чи знаходяться в русі.

1.1.1. Статичні конфіденційні дані

Коли конфіденційні дані перебувають у стані спокою, вони зберігаються в системі та наразі не доступні та не використовуються. Ця інформація може бути розкрита одним із таких способів:

• До даних не застосовується шифрування, що означає, що кожен, хто має доступ до файлу чи бази даних, у якій вони зберігаються, може легко переглядати конфіденційну інформацію.

• Помилки неправильної конфігурації, як-от налаштування хмарних сховищ, що містять конфіденційні дані, як загальнодоступних через Інтернет (у 2021 році медичні дані 50 000 пацієнтів були відкритими для загального доступу в базі даних, яку можна було легко завантажити в Інтернеті)

• Збої в контролі доступу , які надають надмірний доступ до конфіденційних даних користувачам, яким вони не потрібні.

1.1.2. Динамічні конфіденційні дані

Дані під час передачі проходять у вашій мережі між різними системами або між вашою мережею та Інтернетом. Приклади включають, коли дані надсилаються електронною поштою, коли дані переміщуються з локальної мережі в хмару, і дані обмінюються між програмами. Деякі причини розкриття конфіденційних даних під час передачі:

• Відсутність шифрування для даних, що передаються, робить їх відкритими для будь-кого, хто може перехопити ці дані під час їх переміщення.

• Поганий контроль політики та недостатня видимість даних дозволяють користувачам завантажувати та/або ділитися даними з несхваленими або неперевіреними пристроями.

• Співробітники використовують незахищені з’єднання для надсилання електронних листів із конфіденційними даними, які можуть перехопити зловмисники.

1.2. Види атак

Кіберзлочинність побудована навколо ефективного використання вразливостей,

і служби безпеки завжди знаходяться в невигідному становищі, оскільки вони повинні захищати всі можливі точки входу, тоді як зловмисникові потрібно знайти та використати лише одну слабкість або вразливість. Ця асиметрія дуже вигідна будь-якому зловмиснику, в результаті чого навіть великі підприємства намагаються запобігти монетизації кіберзлочинцями доступу до їхніх мереж – мереж, які зазвичай мають підтримувати відкритий доступ і підключення, намагаючись захистити корпоративні ресурси [2].

Зловмисники безпосередньо використовують кілька різних атак, щоб викрити та отримати доступ до конфіденційних даних, наприклад:

1) Шкідливе програмне забезпечення:

Якщо ви коли-небудь бачили антивірусне сповіщення, що з'являлося на екрані, або якщо ви помилково клацали зловмисне вкладення електронної пошти, тоді ви мали справу зі зловмисним програмним забезпеченням. Зловмисники люблять використовувати зловмисне програмне забезпечення, щоб закріпитися на комп'ютерах користувачів і, відповідно, в офісах, у яких вони працюють, оскільки це може бути дуже ефективним. «Зловмисне програмне забезпечення» стосується різних форм шкідливого програмного забезпечення, наприклад вірусів і програм-вимагачів. Коли зловмисне програмне забезпечення потрапляє на ваш комп'ютер, воно може спричинити різного роду хаос, починаючи від контролю над вашим комп'ютером, відстежуючи ваші дії та натискання клавіш і негласно надсилаючи різноманітні конфіденційні дані з вашого комп'ютера чи мережі на домашню базу зловмисника [2].Зловмисники використовуватимуть різноманітні методи, щоб занести зловмисне програмне забезпечення на ваш комп'ютер, але на певному етапі це часто вимагає від користувача вжити заходів для встановлення зловмисного програмного забезпечення. Це може включати натискання посилання для завантаження файлу або відкриття вкладення, яке може виглядати нешкідливим (як-от документи Word або PDF-файл), але насправді містить програму встановлення зловмисного програмного забезпечення.

2) Фішинг:

Фішинг (Password harvesting fishing, Phishing) – це великий розділ хакерської

соціальної інженерії. Фішинг направлений на те, щоб зібрати якомога більше інформації про об’єкт атаки, направляючи йому деякі послання, що викликають інтерес, а всередині містять шкідливе програмне забезпечення або інші шкідливі матеріали.

Звичайно, є ймовірність того, що ви просто не відкриєте випадкове вкладення або не натиснете посилання в будь-якому електронному листі, який прийде вам на шляху – у вас має бути вагома причина, щоб вжити заходів. Зловмисники це теж знають. Коли зловмисник хоче, щоб ви встановили зловмисне програмне забезпечення або розкрили конфіденційну інформацію, він часто вдається до тактики фішингу або прикидається вашим партнером чи кимсь іншим, кому ви довіряєте, щоб змусити вас виконати дії, яких ви зазвичай не робите. Оскільки вони покладаються на людську цікавість і імпульси, фішингові атаки важко зупинити. Під час фішингової атаки зловмисник може надіслати вам електронний лист, який нібито надійшов від людини, якій ви довіряєте, наприклад від вашого начальника або компанії, з якою ви співпрацюєте. Електронний лист виглядатиме легітимним і матиме деяку терміновість (наприклад, у вашому обліковому записі виявлено шахрайську діяльність). В електронному листі буде вкладений файл, який потрібно відкрити, або посилання, яке потрібно натиснути. Відкривши зловмисне вкладення, ви встановите шкідливе програмне забезпечення на свій комп’ютер. Якщо ви клацнете посилання, ви можете перейти на веб-сайт, який виглядає легітимно, і попросить вас увійти, щоб отримати доступ до важливого файлу, за винятком того, що веб-сайт насправді є пасткою, яка використовується для захоплення ваших облікових даних під час спроби входу. Щоб протистояти спробам фішингу, важливо розуміти важливість перевірки відправників електронної пошти та вкладень/посилань.

3) Атака SQL ін'єкції:

SQL означає мову структурованих запитів; це мова програмування, яка використовується для спілкування з базами даних. Багато серверів, які зберігають важливі дані для веб-сайтів і служб, використовують SQL для керування даними у своїх базах даних. Атака SQL-ін’єкцій спеціально націлена на такий сервер,

використовуючи шкідливий код, щоб змусити сервер розкрити інформацію, яку він зазвичай не робив би. Це особливо проблематично, якщо сервер зберігає приватну інформацію про клієнтів із веб-сайту, таку як номери кредитних карток, імена користувачів і паролі (облікові дані) або іншу особисту інформацію, яка є спокусливою та прибутковою ціллю для зловмисника [3].Атака SQL-ін’єкції працює, використовуючи будь-яку з відомих уразливостей SQL, які дозволяють серверу SQL запускати шкідливий код. Наприклад, якщо SQL-сервер вразливий до ін’єкційної атаки, зловмисник може перейти до вікна пошуку веб-сайту та ввести код, який змусить SQL-сервер сайту скинути всі збережені імена користувачів і паролі для сайту.

4) Міжсайтовий сценарій (XSS):

Під час атаки SQL-ін’єкції зловмисник переслідує вразливий веб-сайт, щоб націлитися на його збережені дані, наприклад облікові дані користувача або конфіденційні фінансові дані. Але якщо зловмисник бажає безпосередньо націлитися на користувачів веб-сайту, він може вибрати атаку міжсайтового сценарію. Подібно до атаки з ін’єкцією SQL, ця атака також передбачає введення шкідливого коду на веб-сайт, але в цьому випадку сам веб-сайт не атакується. Натомість шкідливий код, який впровадив зловмисник, запускається лише у браузері користувача, коли він відвідує атакований веб-сайт, і переслідує безпосередньо відвідувача, а не веб-сайт. Одним із найпоширеніших способів, як зловмисник може розгорнути атаку за допомогою міжсайтових сценаріїв, є введення шкідливого коду в коментар або сценарій, який може запускатися автоматично. Наприклад, вони можуть вставити посилання на шкідливий JavaScript у коментар до блогу. Міжсайтові скриптові атаки можуть суттєво зашкодити репутації веб-сайту, піддаючи ризику інформацію користувачів без будь-яких ознак того, що взагалі сталося щось зловмисне. Будь-яка конфіденційна інформація, яку користувач надсилає на сайт, наприклад облікові дані, дані кредитної картки чи інші особисті дані, може бути викрадена за допомогою міжсайтових сценаріїв, навіть якщо власники веб-сайтів навіть не помітять, що проблема виникла.

5) Відмова в обслуговуванні (DoS):

Якщо ви наповнюєте веб-сайт більшим трафіком, ніж він був створений для роботи, ви перевантажите сервер веб-сайту, і веб-сайту буде майже неможливо надати свій вміст відвідувачам, які намагаються отримати до нього доступ.Звісно, це може статися з нешкідливих причин, скажімо, якщо поширюється масштабна новина, і веб-сайт газети перевантажується трафіком від людей, які намагаються дізнатися більше. Але часто таке перевантаження трафіком є зловмисним, оскільки зловмисник переповнює веб-сайт надзвичайною кількістю трафіку, щоб фактично закрити його для всіх користувачів. У деяких випадках ці DoS-атаки виконуються багатьма комп'ютерами одночасно. Цей сценарій атаки відомий як розподілена атака на відмову в обслуговуванні (DDoS). Цей тип атаки може бути навіть важчим для подолання через те, що зловмисник одночасно з'являється з багатьох різних IP-адрес у всьому світі, що робить визначення джерела атаки ще складнішим для мережевих адміністраторів.

6) Перехоплення сесії та атаки "людина посередині":

Коли ви перебуваєте в Інтернеті, ваш комп'ютер виконує багато невеликих зворотних транзакцій із серверами по всьому світу, повідомляючи їм, хто ви, і запитує певні веб-сайти чи служби. Натомість, якщо все йде як слід, веб-сервери мають відповісти на ваш запит, надавши вам інформацію, до якої ви маєте доступ. Цей процес, або сеанс, відбувається незалежно від того, чи ви просто переглядаєте веб-сторінки, чи входите на веб-сайт за допомогою свого імені користувача та пароля [4]. Сеансу між вашим комп'ютером і віддаленим веб-сервером надається унікальний ідентифікатор сеансу, який має залишатися приватним між двома сторонами; однак зловмисник може захопити сеанс, захопивши ідентифікатор сеансу та видаючи себе за комп'ютер, який надсилає запит, дозволяючи йому увійти як нічого не підозрюючи користувач і отримати доступ до несанкціонованої інформації на веб-сервері. Існує кілька методів, які зловмисник може використати для викрадення ідентифікатора сеансу, наприклад атака міжсайтового сценарію, яка використовується для викрадення ідентифікаторів сеансу. Зловмисник також може захопити сеанс, щоб вставити себе між запитуючим комп'ютером і віддаленим сервером, видаючи себе за іншу сторону в сеансі. Це дозволяє їм перехоплювати

інформацію в обох напрямках і зазвичай називається атакою "людина посередині".

7) Повторне використання облікових даних:

Сьогодні користувачі мають стільки логінів і паролів, які потрібно запам’ятати, що виникає спокуса повторно використовувати облікові дані тут чи там, щоб трохи полегшити життя. Незважаючи на те, що найкращі практики безпеки універсально рекомендують мати унікальні паролі для всіх ваших програм і веб-сайтів, багато людей все одно використовують свої паролі повторно – факт, на який покладаються зловмисники. Коли зловмисники отримають колекцію імен користувачів і паролів зі зламаного веб-сайту чи служби (які легко отримати на будь-якій кількості веб-сайтів чорного ринку в Інтернеті), вони знають, що якщо вони використають ті самі облікові дані на інших веб-сайтах, є шанс, що вони зможуть щоб увійти. Незалежно від того, наскільки спокусливим може бути повторне використання облікових даних для вашої електронної пошти, банківського рахунку та вашого улюбленого спортивного форуму, цілком можливо, що одного разу форум буде зламано, що дасть зловмиснику легкий доступ до вашої електронної пошти та банківського рахунку. Коли мова заходить про повноваження, різноманітність є важливою. Доступні менеджери паролів, які можуть бути корисними, коли справа доходить до керування різними обліковими даними, які ви використовуєте. Це лише вибір поширених типів і методів атак (перейдіть за цим посиланням, щоб дізнатися більше про вразливості веб-додатків). Він не має на меті бути вичерпним, і зловмисники розвиваються та розробляють нові методи за потреби; однак усвідомлення та пом’якшення цих типів атак значно покращить вашу безпеку.

1.3. Стандарти захисту інформації

Розкриття конфіденційних даних є сферою безпеки. Правильне дотримання кількох фундаментальних принципів має велике значення для пом’якшення небажаних наслідків, таких як порушення або втрата даних.

Щоб зробити заходи кібербезпеки чіткими, потрібні письмові норми. Ці норми відомі як стандарти кібербезпеки: загальні набори приписів для ідеального виконання певних заходів. Стандарти можуть включати методи, рекомендації, еталонні рамки тощо. Це забезпечує ефективність безпеки, полегшує інтеграцію та взаємодію, дозволяє значуще порівнювати показники, зменшує складність і забезпечує структуру для нових розробок.

Стандарт безпеки – це опублікована специфікація, яка встановлює спільну мову та містить технічну специфікацію або інші точні критерії та призначена для постійного використання, як правило, настанови або визначення. Метою стандартів безпеки є підвищення безпеки систем інформаційних технологій (IT), мереж і критичної інфраструктури. Добре написані стандарти кібербезпеки забезпечують узгодженість між розробниками продуктів і служать надійним стандартом для придбання продуктів безпеки [5].

Стандарти безпеки, як правило, надаються для всіх організацій, незалежно від їх розміру, галузі та сектору, в якому вони працюють. Цей розділ містить інформацію про кожен стандарт, який зазвичай визнається важливим компонентом будь-якої стратегії кібербезпеки.

1.3.1. ISO

ISO означає Міжнародну організацію зі стандартизації. Міжнародні стандарти змушують речі працювати. Ці стандарти надають специфікації світового рівня для продуктів, послуг і комп’ютерів, щоб забезпечити якість, безпеку та ефективність. Вони відіграють важливу роль у сприянні міжнародній торгівлі.

Стандарт ISO офіційно встановлено 23 лютого 1947 року. Це незалежна неурядова міжнародна організація. Сьогодні до її складу входять 162 національні органи зі стандартизації та 784 технічні комітети та підкомітети, які опікуються розробкою стандартів. ISO опублікувала понад 22336 міжнародних стандартів і відповідних документів, які охоплюють майже всі галузі, від інформаційних

технологій до безпеки харчових продуктів, до сільського господарства та охорони здоров'я.

Серія ISO 27000 - це сімейство стандартів інформаційної безпеки, розроблене Міжнародною організацією зі стандартизації та Міжнародною електротехнічною комісією, щоб забезпечити всесвітньо визнану основу для найкращого управління інформаційною безпекою. Це допомагає організації захищати свої інформаційні активи, такі як відомості про співробітників, фінансову інформацію та інтелектуальну власність. Необхідність серії ISO 27000 виникає через ризик кібератак, з якими стикається організація. Кількість кібератак зростає з кожним днем, що робить хакерів постійною загрозою для будь-якої галузі, яка використовує технології [6].

Серію ISO 27000 можна розділити на багато типів. Вони є:

- ISO 27001 - цей стандарт дозволяє нам довести, що клієнти та зацікавлені сторони будь-якої організації керують найкращим захистом своїх конфіденційних даних та інформації. Цей стандарт передбачає процесний підхід для встановлення, впровадження, експлуатації, моніторингу, підтримки та вдосконалення програмного продукту.
- ISO 27000. Цей стандарт містить пояснення термінології, яка використовується в ISO 27001.
- ISO 27002. Цей стандарт містить вказівки щодо стандартів організаційної безпеки інформації та практики управління інформаційною безпекою. Він включає вибір, впровадження, функціонування та управління засобами контролю, враховуючи середовище ризиків інформаційної безпеки організації.
- ISO 27005. Цей стандарт підтримує загальні концепції, визначені в 27001. Він розроблений, щоб надати рекомендації щодо впровадження інформаційної безпеки на основі підходу до управління ризиками. Щоб повністю зрозуміти ISO/IEC 27005, необхідні знання концепцій, моделей, процесів і термінології, описаних у ISO/IEC 27001 та ISO/IEC 27002. Цей стандарт підходить для всіх типів організацій, таких як неурядові організації, державні установи та комерційні підприємства.
- ISO 27032 – це міжнародний стандарт, який чітко зосереджується на

кібербезпеці. Цей стандарт містить вказівки щодо захисту інформації за межами організації, наприклад у співпраці, партнерстві чи інших угодах про обмін інформацією з клієнтами та постачальниками.

1.3.2. Закон про IT

Основною метою Закону про інформаційні технології, також відомого як ITA-2000, або Закону про IT, є створення правової інфраструктури в Індії, яка має справу з кіберзлочинністю та електронною комерцією. Закон про IT базується на Типовому законі ООН про електронну комерцію 1996 року, рекомендованому Генеральною Асамблеєю ООН. Цей акт також використовується для перевірки зловживання кібермережею та комп'ютером в Індії. Він був офіційно прийнятий у 2000 році та доповнений у 2008 році. Він був розроблений, щоб дати поштовх електронній комерції, електронним транзакціям і пов'язаній діяльності, пов'язаній з комерцією та торгівлею. Це також полегшує електронне урядування за допомогою надійних електронних записів.

Закон про IT 2000 року складається з 13 глав, 94 розділів і 4 розкладів. Перші 14 розділів, які стосуються цифрових підписів, та інші розділи стосуються органів сертифікації, які мають ліцензію на видачу сертифікатів цифрових підписів, розділи з 43 по 47 передбачають штрафи та компенсацію, розділи з 48 по 64 стосуються оскарження у вищому суді, розділи з 65 по 79 стосуються правопорушення, а решта розділів з 80 по 94 стосуються різних актів.

1.3.3. Закон про авторське право

Закон про авторське право 1957 року, до якого внесено зміни Законом про авторське право 2012 року, регулює предмет закону про авторське право в Індії. Цей Закон застосовується з 21 січня 1958 року. Авторське право – це юридичний термін, який описує право власності на контроль над правами авторів «оригінальних

авторських творів», які зафіксовані в матеріальній формі вираження. Оригінальний авторський твір – це розповсюдження певних творів творчого вираження, включаючи книги, відео, фільми, музику та комп'ютерні програми. Закон про авторське право був прийнятий, щоб збалансувати використання та повторне використання творчих робіт проти бажання творців мистецтва, літератури, музики та монетизувати їхню роботу, контролюючи, хто може робити та продавати копії роботи.

Закон про авторське право охоплює:

- Права власників авторських прав;
- Твори підлягають охороні;
- Тривалість авторського права;
- Хто може претендувати на авторські права.

Закон про авторське право не поширюється на таке:

- Ідеї, процедури, методи, процеси, концепції, системи, принципи або відкриття;
- Твори, які не зафіксовані в матеріальній формі (наприклад, хореографічний твір, який не нотований чи записаний, або імпровізаційна промова, яка не була записана);
- Знайомі символи або малюнки;
- Назви, імена, короткі фрази та гасла;
- Звичайні варіації друкарського орнаменту, літер або забарвлення.

1.3.4. Патентне право

Патентне право – це право, яке стосується нових винаходів. Традиційне патентне право захищає матеріальні наукові винаходи, такі як друковані плати, нагрівальні спіралі, автомобільні двигуни чи блискавки. З плином часу патентне право використовується для захисту широкого спектру винаходів, таких як бізнес-практика, алгоритми кодування або генетично модифіковані організми. Це право забороняти іншим виробляти, використовувати, продавати, імпортувати, спонукати інших до порушення та пропонувати продукт, спеціально адаптований для застосування патенту [7].

Загалом, патент – це право, яке може бути надано, якщо винахід:

- Не природний об'єкт або процес;
- Новий;
- Корисний;
- Не банальний.

1.3.5. IPR (Intellectual property rights)

Права інтелектуальної власності – це права, які дозволяють творцям або власникам патентів, торгових марок або творів, захищених авторським правом, отримувати вигоду від власних планів, ідей чи інших нематеріальних активів або інвестицій у створення. Ці права інтелектуальної власності викладені в статті 27 Загальної декларації прав людини. Він передбачає право на захист моральних і матеріальних інтересів, що випливають з авторства наукових, літературних чи художніх творів. Ці права власності дозволяють власнику здійснювати монопольне використання предмета протягом певного періоду.

1.4. Постановка задачі

Під інформаційною безпекою розуміється захищеність інформації та підтримує інфраструктури від випадкових або навмисних впливів природного або штучного характеру, що можуть призвести нанесенням шкоди власникам або користувачам інформації і підтримуючої інфраструктури.

На практиці найважливішими є три аспекти інформаційної безпеки:

- доступність (можливість за розумний час отримати необхідну інформаційну послугу);
- цілісність (актуальність і несуперечність інформації, її захищеність від руйнування і несанкціонованого зміни);
- конфіденційність (захист від несанкціонованого ознайомлення).

Формування режиму інформаційної безпеки − проблема комплексна. Заходи для її рішення можна поділити на чотири рівні:

- законодавчий (закони, нормативні акти, стандарти і т.п.);
- адміністративний (дії загального характеру, що починаються керівництвом організації);
- процедурний (конкретні міри безпеки, що мають справу з людьми);
- програмно-технічний (конкретні технічні заходи).

При формуванні режиму інформаційної безпеки слід враховувати сучасний стан інформаційних технологій. Майже всі організації чекають від інформаційних систем в першу чергу корисною функціональності. Комп'ютерні системи купуються не заради захисту даних; навпаки, захист даних будується заради вигідного використання комп'ютерних систем. Для отримання корисної функціональності природно звернутися до найбільш сучасним рішенням в області інформаційних технологій. Значить, кажучи про захист, слід мати на увазі перш за все сучасні апаратні та програмні платформи.

Висновки до розділу 1

Отож, основним напрямком розвитку ринку залишаються ризики несанкціонованого доступу до даних. За допомогою шифрування можна нейтралізувати досить велику частку загроз, і вибір на користь шифрування виправдовує себе і є зручним та досить простим способом в застосуванні для організації доступу до корпоративних ресурсів, до порталів і хмарних сервісів.

РОЗДІЛ 2
ШИФРУВАННЯ ДАНИХ

2.1. Для чого потрібне шифрування

Щодня через Інтернет надсилається величезна кількість особистих даних: електронні листи з деталями про наше особисте життя, паролі, які ми вводимо на екранах входу, податкові документи, які ми завантажуємо на сервери.

Інтернет-протоколи надсилають приватні дані пакетами за тими самими маршрутами, що й дані всіх інших, і, на жаль, зловмисники знайшли способи перегляду даних, що ширяють Інтернетом.

Ось тут і з’являється шифрування: шифрування даних означає, що ми шифруємо вихідні дані, щоб приховати значення тексту, але водночас дозволяємо розшифрувати дані за допомогою секретного ключа.

Шифрування дозволяє двом людям (або комп’ютерам) обмінюватися особистою інформацією через відкриті мережі.

2.2. Як працює шифрування

Шифрування є критично важливим інструментом, який власники бізнесу, як і ви, використовують для захисту даних, встановлення довіри клієнтів і дотримання нормативних вимог. Усвідомлюєте ви це чи ні, шифрування використовується майже в кожній цифровій бізнес-взаємодії.

Шифрування даних – це механізм безпеки, який перетворює дані вашої компанії у відкритому вигляді в закодовану інформацію, яка називається зашифрованим текстом.

Ось кілька основних термінів шифрування, які слід знати:

• Також відомі як шифри, алгоритми – це правила або інструкції для процесу шифрування. Довжина ключа, функціональність і особливості використовуваної системи шифрування визначають ефективність шифрування.

• Дешифрування − це процес перетворення нерозбірливого зашифрованого тексту в читабельну інформацію.

• Ключ шифрування – це рандомізований рядок бітів, який використовується для шифрування та дешифрування даних. Кожен ключ унікальний, а довші ключі важче зламати. Типова довжина ключа становить 128 і 256 біт для закритих ключів і 2048 для відкритих ключів.

Існує два види систем криптографічних ключів: симетричні та асиметричні.

Зашифрований текст або числа можна розшифрувати лише за допомогою унікального ключа, який надається під час шифрування.

Шифрування даних можна використовувати як для даних у русі, так і для даних у стані спокою. Ви можете поєднати цей механізм захисту даних із службами автентифікації, щоб гарантувати доступ до ваших бізнес-даних лише авторизованим користувачам.

Шифрування – це спосіб зробити дані (повідомлення або файли) нечитабельними, забезпечуючи доступ до цих даних лише авторизованій особі. Шифрування використовує складні алгоритми для шифрування даних і розшифровує ці дані за допомогою ключа, наданого відправником повідомлення. Шифрування гарантує, що інформація залишається приватною та конфіденційною, незалежно від того, зберігається вона чи передається. Будь-який несанкціонований доступ до даних побачить лише хаотичний масив байтів [8].

Дані, які потрібно зашифрувати, називають відкритим текстом. Відкритий текст потрібно передати за допомогою певних алгоритмів шифрування, які в основному є математичними обчисленнями, які виконуються з необробленою інформацією. Існує кілька алгоритмів шифрування, кожен з яких відрізняється програмою та індексом безпеки.

Окрім алгоритмів, потрібен також ключ шифрування. За допомогою згаданого ключа та відповідного алгоритму шифрування відкритий текст перетворюється на зашифрований фрагмент даних, також відомий як зашифрований текст. Замість того, щоб надсилати відкритий текст одержувачу, зашифрований текст надсилається через незахищені канали зв'язку.

Коли зашифрований текст досягає призначеного одержувача, він/вона може використати ключ дешифрування, щоб перетворити зашифрований текст назад у вихідний читабельний формат, тобто відкритий текст. Цей ключ розшифровки має постійно зберігатися в таємниці та може бути або не схожий на ключ, який використовується для шифрування повідомлення. Зрозуміємо те ж саме на прикладі.

Зрозуміємо процес роботи на прикладі: жінка хоче надіслати своєму хлопцю особистий текст, тому вона шифрує його за допомогою спеціального програмного забезпечення, яке шифрує дані в те, що здається нерозбірливим. Потім вона надсилає повідомлення, а її хлопець, у свою чергу, використовує правильну розшифровку, щоб перекласти його [9].

Якщо хтось задається питанням, чому організаціям потрібно практикувати шифрування, майте на увазі ці чотири причини:

1) Автентифікація: шифрування з відкритим ключем доводить, що вихідний сервер веб-сайту володіє закритим ключем і тому йому законно призначено сертифікат SSL. У світі, де існує так багато шахрайських веб-сайтів, це важлива функція.

2) Конфіденційність: шифрування гарантує, що ніхто не зможе прочитати повідомлення або отримати доступ до даних, крім законного одержувача або власника даних. Цей захід запобігає доступу та читанню особистих даних кіберзлочинцями, хакерами, інтернет-провайдерами, спамерами та навіть державними установами.

3) Відповідність нормативним вимогам: у багатьох галузях і державних установах діють правила, які вимагають від організацій, які працюють з особистою інформацією користувачів, зберігати цю інформацію в шифруванні.

Вибірка нормативних стандартів і стандартів відповідності, які забезпечують дотримання шифрування, включає HIPAA, PCI-DSS і GDPR.

4) Безпека: шифрування допомагає захистити інформацію від порушень даних, незалежно від того, чи перебувають вони в стані спокою чи в дорозі. Наприклад, навіть якщо корпоративний пристрій загубили або вкрали, дані, що зберігаються на ньому, швидше за все, будуть у безпеці, якщо жорсткий диск правильно зашифровано. Шифрування також допомагає захистити дані від зловмисних дій, як-от атак типу "людина посередині", і дозволяє сторонам спілкуватися, не боячись витоку даних.

Шифрування даних зазвичай буває двох типів:

• Симетричне шифрування: використовується один симетричний ключ як для шифрування, так і для дешифрування даних. Ключ надається всім авторизованим користувачам для надання доступу до даних.

• Асиметричне шифрування: використовує два окремі ключі для шифрування та дешифрування даних. Один ключ стає загальнодоступним (спільним для всіх), а інший залишається приватним (відомим лише генератору ключа). Відкритий ключ використовується для шифрування даних, а закритий – для їх розшифровки.

Тепер ми зануримося в два найпоширеніші типи шифрування, які використовуються для захисту Інтернет-комунікацій: симетричне шифрування та шифрування з відкритим ключем.

Різні методи шифрування залежать від типу використовуваних ключів, довжини ключа шифрування та розміру зашифрованих блоків даних.

2.3. Методи симетричного шифрування

Симетричне шифрування – це будь-який метод, у якому той самий ключ використовується як для шифрування, так і для дешифрування даних. Шифр Цезаря є одним із найпростіших методів симетричного шифрування, і, звичайно, одним із найлегших для зламу.

З тих пір криптологи винайшли багато інших методів симетричного

шифрування, включно з тими, які використовуються сьогодні для шифрування даних, таких як паролі.

У системі симетричного ключа кожен, хто отримує доступ до даних, має однаковий ключ. Ключі, які шифрують і розшифровують повідомлення, також повинні залишатися секретними для забезпечення конфіденційності. Хоча це можливо, але безпечне розподілення ключів для забезпечення належного контролю робить симетричне шифрування непрактичним для широкого комерційного використання.

2.3.1. Шифр Цезаря

Техніка шифру Цезаря є одним із найперших і найпростіших методів техніки шифрування. Це просто тип шифру підстановки, тобто кожна літера певного тексту замінюється літерою з фіксованою кількістю позицій у алфавіті. Наприклад, зі зсувом на 1 A буде замінено на B, B стане C тощо. Метод, очевидно, названий на честь Юлія Цезаря, який, очевидно, використовував його для спілкування зі своїми чиновниками.

Таким чином, щоб зашифрувати заданий текст, нам потрібне ціле значення, відоме як зсув, яке вказує на кількість позицій, на яку кожна літера тексту була переміщена вниз.

Шифрування можна представити за допомогою модульної арифметики, спочатку перетворивши літери на числа за схемою A = 0, B = 1,…, Z = 25. Шифрування літери зсувом n можна описати математично так:

Фаза шифрування зі зсувом n: $E_n(x) = (x + n) mod\ 26$

Фаза дешифрування зі зсувом n: $D_n(x) = (x - n) mod\ 26$

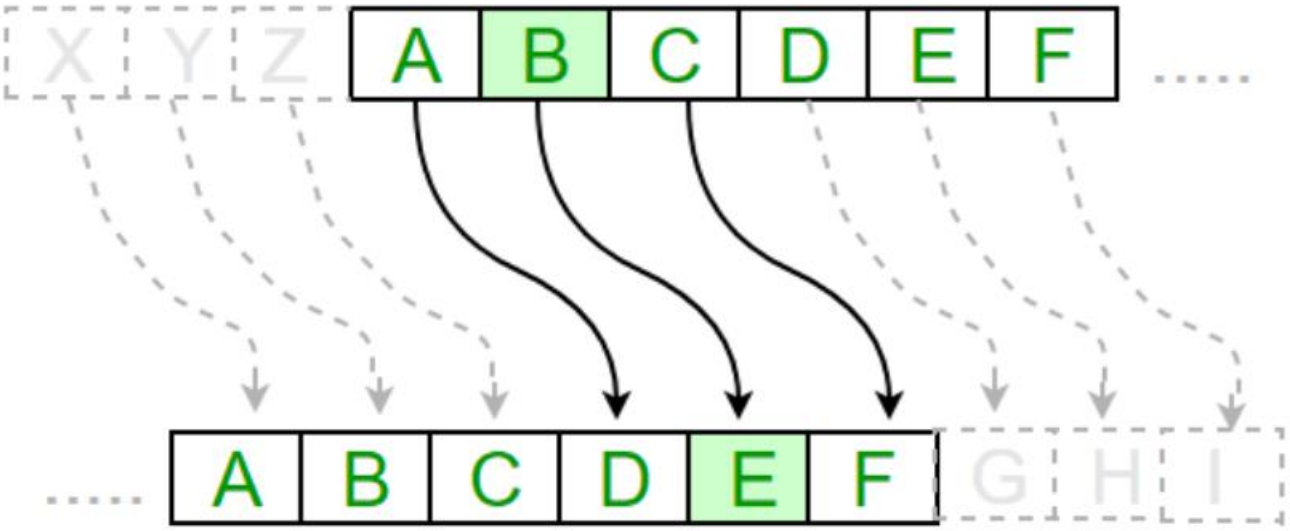

Рис.2.1. Зсув для методу Цезаря

Приклади:

1) Текст : ABCDEFGHIJKLMNOPQRSTUVWXYZ

Зсув : 23

Шифр : XYZABCDEFGHIJKLMNOPQRSTUVW

2) Текст : ATTACKATONCE

Зсув : 4

Шифр : EXXEGOEXSRGI

Алгоритм для шифру Цезаря:

Вхідні дані:

1) Рядок малих літер, який називається Текст.

2) Ціле число від 0 до 25, що позначає необхідний зсув.

Процедура:

- Переходьте по заданому тексту по одному символу за раз.
- Для кожного символу трансформуйте заданий символ згідно з правилом залежно від того, шифруємо чи розшифровуємо текст.
- Повернути новий згенерований рядок.
- Програма, яка отримує текст (рядок) і значення Shift (ціле число) і повертає зашифрований текст.

З кодом програми, яка отримує текст (рядок) і значення Shift (ціле число) і повертає зашифрований текст на мовіJavaможна побачити в Додатку А.

Для розшифрування ми можемо написати іншу функцію decrypt, подібну до encrypt, яка застосує заданий зсув у протилежному напрямку, щоб розшифрувати вихідний текст.

Таким чином, ми можемо використати ту саму функцію для розшифровки, натомість ми змінимо значення shift таким чином, щоб shift = 26-shift

2.3.2. Шифр Віженера

Французькі криптологи винайшли шифр Віженера в середині 1500-х років. Шифр вважався особливо надійним, і автор Льюїс Керолл навіть назвав його «незламним» у 1868 році. Він справді був набагато сильнішим, ніж шифр Цезаря, але, як ми побачимо, його точно можна зламати [10].

У шифрі Віженера використовується ціле слово як клавіша зсуву, на відміну від шифру Цезаря з одним зсувом.

Уявіть, що ми хочемо зашифрувати фразу VERSAILLESта використати клавішу ShiftCHEESE.

По-перше, нам потрібно повторити клавішу Shift, щоб вирівняти кожну з літер у фразі:

Таблиця 1.

Зсув для методу Віженера

Оригінал:	V	E	R	S	A	I	L	L	E	S
Клавіша Shift:	C	H	E	E	S	E	C	H	E	E

Тепер замінимо кожну букву оригінального тексту відповідно до таблиці Віженера:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X

Рис.2.2. Таблиця шифрування Віженера

Для першої літери «V» вибираємо рядок, який починається на «V». Тоді, оскільки відповідна літера клавіші shift – «C», ми переходимо до стовпця, який має заголовок «C». Буква на перетині рядка "V" і стовпця "C" - це "X". Таким чином, ми шифруємо "V" як "X". Буква на перетині рядка "E" і стовпця "H" − "L", тому ми шифруємо "E" як "L". Якщо ми продовжимо, ми отримаємо зашифрований текст "XLVWSMNSIW".

Таблиця 2.

Результат шифрування методом Віженера

Оригінал:	V	E	R	S	A	I	L	L	E	S
Клавіша Shift:	C	H	E	E	S	E	C	H	E	E
Зашифровано:	X	L	V	W	S	M	N	S	I	W

Розглянемо розшифрування цим методом. Якщо ми отримуємо зашифроване повідомлення «NVYZJI» від нашого союзника, і ми знаємо, що вони використали шифр Віженера з клавішею перемикання «CHEESE». Знову вибудовуємо зашифроване повідомлення за допомогою клавіші shift:

Таблиця 3.

Зсув для методу Віженера

Зашифровано:	N	V	Y	Z	J	I
Клавіша Shift:	C	H	E	E	S	E

Тепер ми можемо зробити зворотну заміну за таблицею:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X

Рис.2.3. Таблиця дешифрування Віженера

Ми починаємо з вибору рядка для першої літери в клавіші shift "C". Потім ми скануємо цей рядок, поки не знайдемо першу зашифровану літеру "N". Коли ми знаходимо «N», ми дивимося вгору, щоб побачити заголовок цього стовпця «L». Таким чином, розшифрування "N" − це "L". Для наступної літери ми вибираємо рядок "H", знаходимо в рядку "V" і дивимося вгору, щоб побачити, що ми в стовпці "O". Якщо ми зробимо це для кожної клавіші shift і зашифрованої літери, ми розшифруємо всю фразу як "LOUVRE".

Таблиця 4.

Результат дешифрування методом Віженера

Зашифровано:	N	V	Y	Z	J	I
Клавіша Shift:	C	H	E	E	S	E
Оригінал:	L	O	U	V	R	E

Шифр Віженера є різновидом багатоалфавітного шифру, і його важче зламати, ніж шифр Цезаря, оскільки в ньому використовується ціле слово зсуву.

Якби перехоплювач не мав уявлення, що таке клавіша перемикання, і хотів грубою силою пройти до дешифрування, йому потрібно було б спробувати всі можливі у світі слова перемикання, а можливо, навіть вигадані слова! Для простого смертного це може зайняти все життя. Це набагато більше роботи, ніж грубе форсування шифру Цезаря, де нам просто потрібно було перевірити 26 різних обсягів зміни.

У 1800-х роках люди нарешті знайшли різні способи використання частотного аналізу для злому шифру. Наприклад, у довгому повідомленні коротке слово на зразок "THE" може бути перекладено тими самими трьома зашифрованими літерами кілька разів (але не кожного разу), і це розкриває можливу довжину для клавіші shift.

Тепер, коли ми використовуємо потужні комп’ютери, шифр Віженера відносно легко розшифрувати, оскільки комп’ютер може швидко перевірити мільйони слів і легко знайти витік інформації в частотному аналізі.

2.3.3. Розширений стандарт шифрування (AES)

Advanced Encryption Standard – це симетричний алгоритм шифрування, який шифрує блоки даних по 128 біт за раз. Для шифрування цих блоків даних він використовує ключі довжиною 128, 192 і 256 бітів. 256-бітний ключ шифрує дані за 14 раундів, 192-бітний ключ за 12 раундів і 128-бітний ключ за 10 раундів. Кожен раунд складається з кількох кроків заміни, транспозиції, змішування відкритого

тексту тощо.

Ви можете використовувати AES для безпеки Wi-Fi, шифрування мобільних додатків, шифрування файлів і безпеки рівня безпечних сокетів/транспортного рівня (SSL/TLS). Насправді ваш веб-браузер зараз використовує AES для шифрування підключення до цього веб-сайту.

Крім безпеки, шифрування AES дуже привабливе для тих, хто з ним працює. чому Оскільки процес шифрування AES відносно простий для розуміння. Це забезпечує просте впровадження , а також дуже швидке шифрування та дешифрування .

Крім того, AES вимагає менше пам’яті, ніж багато інших типів шифрування (наприклад, DES), що робить його справжнім переможцем, коли справа доходить до вибору бажаного методу шифрування.

Нарешті, коли для дії потрібен додатковий рівень безпеки, ви можете поєднати AES з різними протоколами безпеки, як-от WPA2, або навіть іншими типами шифрування, як-от SSL.

Навіть якщо він не зовсім «давній», розширений стандарт шифрування старий .

Розроблений у 1998 році двома бельгійськими криптографами Вінсентом Рейменом і Джоан Демен, AES існує вже більше 20 років. Спочатку його називали Rijndael - поєднання імен його розробників [11].

Завдяки своїй непроникності шифрування AES вже 18 років служить стандартом шифрування. Це тому, що в 2002 році Національний інститут стандартів і технологій (NIST) замінив застарілий стандарт шифрування даних (DES) на AES.

Ось кілька помітних прикладів того, де розробники можуть використовувати шифрування AES:

- VPN (віртуальні приватні мережі). Оскільки робота VPN полягає в безпечному з’єднанні вас з іншим сервером в Інтернеті, можна розглядати лише найкращі методи шифрування, щоб ваші дані не витоку. VPN, які використовують розширений стандарт шифрування з 256-бітними ключами, включають NordVPN, Surfshark і ExpressVPN .
- Wi-Fi. Правильно – бездротові мережі також використовують шифрування

AES (зазвичай разом із WPA2). Це не єдиний тип шифрування, який можуть використовувати мережі Wi-Fi, проте більшість інших методів шифрування набагато менш безпечні.

- Мобільні додатки. Багато популярних програм (наприклад, Snapchat і Facebook Messenger) використовують шифрування AES, щоб безпечно надсилати інформацію, як-от фотографії та повідомлення.
- Інструменти архівування та стиснення. Усі основні програми стиснення файлів використовують AES для запобігання витоку даних. Ці інструменти включають 7z, WinZip і RAR.
- Системні компоненти ОС. Деякі компоненти операційної системи (наприклад, файлові системи) використовують розширений стандарт шифрування для додаткового рівня безпеки.
- Бібліотеки мов програмування. Бібліотеки таких мов кодування, як Java, Python і C++, реалізують шифрування AES.
- Менеджери паролів. Це програми, які несуть багато конфіденційної інформації. Ось чому такі менеджери паролів, як LastPass і Dashlane, не пропускають важливий етап впровадження AES.

2.3.4. Потрійний стандарт шифрування даних (DES)

TripleDES – це симетрична техніка шифрування та вдосконалена форма стандарту шифрування даних (DES), який шифрує блоки даних за допомогою 56-бітного ключа. Потрійний DES застосовує алгоритм шифрування DES тричі до кожного блоку даних. Ви можете використовувати Triple DES для шифрування PIN-кодів банкоматів і паролів UNIX. Такі популярні програми, як Microsoft Office і Mozilla Firefox, також використовують Triple DES.

3DES – це методшифрування , який був похідним від оригінального стандарту шифрування даних (DES). Він став популярним наприкінці дев’яностих, але з тих пір втратив популярність через зростання більш безпечних алгоритмів, таких як AES-256 і XChaCha20 [12].

Незважаючи на те, що у 2023 році його підтримку буде припинено, у деяких ситуаціях він все ще реалізований. Оскільки він заснований на одному з перших широко опублікованих і вивчених алгоритмів, DES, все одно важливо дізнатися, що таке 3DES і як він працює.

Хоча він офіційно відомий як алгоритм потрійного шифрування даних (3DEA), найчастіше його називають 3DES. Це пояснюється тим, що алгоритм 3DES тричі використовує шифр Data Encryption Standard (DES) для шифрування своїх даних.

DES – це алгоритм із симетричним ключем, заснований на мережі Фейстеля. Як шифр із симетричним ключем, він використовує той самий ключ для процесів шифрування та дешифрування. Мережа Feistel робить обидва ці процеси майже однаковими, що призводить до ефективнішого алгоритму.

DES має як 64-бітний блок, так і розмір ключа, але на практиці ключ забезпечує лише 56-бітний захист. 3DES було розроблено як більш безпечну альтернативу через малу довжину ключа DES. У 3DES алгоритм DES запускається три рази з трьома ключами; однак він вважається безпечним, лише якщо використовуються три окремі ключі.

Як тільки слабкі сторони звичайного DES стали більш очевидними, 3DES був використаний у широкому діапазоні програм. Це була одна з найбільш поширених схем шифрування до появи AES.

Деякі приклади його реалізації включали:

- Microsoft Office
- Firefox
- Платіжні системи EMV

Багато з цих платформ більше не використовують 3DES, оскільки є кращі альтернативи.

Національний інститут стандартів і технологій (NIST) опублікував роботу пропозиції про те, що всі форми 3DES будуть застарілими до 2023 року та заборонені з 2024 року.

2.3.5. Blowfish

Спочатку розроблений для заміни DES, Blowfish – це процес симетричного алгоритму, який розділяє повідомлення на 64-розрядні сегменти та шифрує їх окремо. Blowfish відомий як швидкий, гнучкий і непорушний. Він є загально доступним, тому його можна використовувати безкоштовно, що робить його ще більш привабливим. Ви можете використовувати Blowfish для захисту транзакцій на своїх платформах електронної комерції. Він також може бути ефективним у захисті інструментів шифрування електронної пошти підприємств, систем керування паролями та програмного забезпечення резервного копіювання. Blowfish – це симетричний блоковий шифр, який можна використовувати як додаткову заміну DES або IDEA. Він має ключ змінної довжини, від 32 біт до 448 біт, що робить його ідеальним як для домашнього використання, так і для експорту. Blowfish був розроблений у 1993 році Брюсом Шнайером як швидка безкоштовна альтернатива існуючим алгоритмам шифрування. З тих пір його ретельно проаналізували, і він повільно набуває визнання як надійний алгоритм шифрування. Blowfish не запатентований і не має ліцензії, і доступний безкоштовно для будь-якого використання. Blowfish є одним із найшвидших блокових шифрів , які широко використовуються, за винятком випадків зміни ключів. Кожен новий ключ вимагає попередньої обробки, еквівалентної шифруванню близько 4 кілобайт тексту, що дуже повільно порівняно з іншими блоковими шифрами. Це запобігає його використанню в деяких програмах, але не є проблемою в інших, наприклад SplashID.

На Blowfish не поширюються жодні патенти, тому він доступний для будь-кого. Це сприяло його популярності в криптографічному програмному забезпеченні.

2.3.6. Twofish

Twofish – це симетричний безліцензійний метод шифрування, який шифрує

блоки даних по 128 біт. Це більш універсальний наступник методів шифрування Blowfish і Threefish. Twofish завжди шифрує дані в 16 циклів незалежно від розміру ключа шифрування. Хоча воно повільніше, ніж шифрування AES, ви можете використовувати Twofish для захисту своїх рішень для шифрування файлів і папок.

Twofish є наступником Blowfish і, як і його попередник, використовує симетричне шифрування, тому потрібен лише один 256-бітний ключ. Цей метод є одним із найшвидших алгоритмів шифрування та ідеально підходить як для апаратного, так і для програмного середовища. Коли він був випущений, він став фіналістом конкурсу Національного інституту технологій і науки (NIST) на пошук заміни алгоритму шифрування стандарту шифрування даних (DES). Зрештою, алгоритм Rjindael був обраний замість алгоритму шифрування Twofish. Подібно до Blowfish, у цьому симетричному алгоритмі шифрування використовується блоковий шифр. агато організацій задають питання: чи безпечний Twofish, якщо NIST не хоче використовувати його для заміни DES? Відповідь − так, Twofish надзвичайно безпечний у використанні. Причина, по якій NIST не бажає використовувати Twofish, полягає в тому, що він повільніший порівняно з алгоритмом шифрування Rjindael. Однією з причин, чому Twofish настільки безпечний, є те, що він використовує 128-бітний ключ, який майже не сприйнятливий до атак грубою силою. Обсяг обчислювальної потужності та часу, необхідних для грубої форсування зашифрованого повідомлення із 128-бітним ключем, робить будь-яку інформацію, яка розшифровується, непридатною, оскільки розшифровка одного повідомлення може тривати десятиліття.

Багато організацій задають питання: чи безпечний Twofish, якщо NIST не хоче використовувати його для заміни DES? Відповідь − так, Twofish надзвичайно безпечний у використанні. Причина, по якій NIST не бажає використовувати Twofish, полягає в тому, що він повільніший порівняно з алгоритмом шифрування Rjindael. Однією з причин, чому Twofish настільки безпечний, є те, що він використовує 128-бітний ключ, який майже не сприйнятливий до атак грубою силою. Обсяг обчислювальної потужності та часу, необхідних для грубої форсування зашифрованого повідомлення із 128-бітним ключем, робить будь-яку

інформацію, яка розшифровується, непридатною, оскільки розшифровка одного повідомлення може тривати десятиліття.

Однак це не означає, що Twofish несприйнятливий до всіх атак. Частина алгоритму шифрування Twofish використовує попередньо обчислену заміну, залежну від ключа, для отримання зашифрованого тексту. Попереднє обчислення цього значення робить Twofish вразливим до атак по бічному каналу, але залежність ключа від заміни допомагає захистити його від атак по бічному каналу. На Twofish було здійснено кілька атак, але творець алгоритму Брюс Шнайер стверджує, що це не були справжні атаки криптоаналізу. Це означає, що практичного зриву алгоритму Twofish ще не відбулося.

Хоча, як і Advanced Encryption Standard (AES), Twofish не є найпоширенішим алгоритмом шифрування, він все ще має багато застосувань. Найбільш відомі продукти, які використовують Twofish у своїх методах шифрування:

- PGP (Pretty Good Privacy): PGP – це алгоритм шифрування, який використовує Twofish для шифрування електронних листів. Дані електронного листа зашифровано, але відправник і тема не зашифровані.
- GnuPG: GnuPG – це реалізація OpenPGP, яка дозволяє користувачам шифрувати та надсилати дані під час спілкування. GnuPGP використовує системи керування ключами та модулі для доступу до каталогів відкритих ключів. Ці каталоги відкритих ключів надають відкриті ключі, опубліковані іншими користувачами в Інтернеті, тому, якщо вони надсилають повідомлення, зашифроване своїм закритим ключем, будь-хто, хто має доступ до каталогу відкритих ключів, може розшифрувати це повідомлення.
- TrueCrypt: TrueCrypt шифрує дані на пристроях за допомогою прозорих для користувача методів шифрування. TrueCrypt працює локально на комп’ютері користувача та автоматично шифрує дані, коли вони залишають локальний комп’ютер. Прикладом може бути користувач, який надсилає файл зі свого локального комп’ютера до зовнішньої бази даних. Файл, надісланий до бази даних, буде зашифрований, коли він покине локальний комп’ютер.
- KeePass: KeePass – це програмне забезпечення для керування паролями, яке

шифрує збережені паролі та створює паролі за допомогою Twofish.

2.3.7. Шифрування зі збереженням формату (FPE)

Шифрування зі збереженням формату – це симетричний алгоритм, який зберігає формат і довжину ваших даних під час їх кодування. Наприклад, якщо номер телефону клієнта 813-204-9012, FPE змінить його на інший, скажімо, 386-192-4019. Таким чином, формат і довжина залишаються незмінними, але символи змінюються, щоб зберегти вихідні дані.

Ви можете використовувати FPE для захисту програмного забезпечення та інструментів керування хмарою. Amazon Web Services (AWS) і Google Cloud, одні з найбільш надійних хмарних платформ, використовують цей метод для хмарного шифрування.

Шифрування із збереженням формату (FPE) – це процес шифрування даних таким чином, щоб вихідні дані (зашифрований текст) залишалися в тому самому форматі, що й вхідні дані (відкритий текст). Значення «формату» різне . Зазвичай використовуються лише кінцеві набори символів; цифровий, буквений або буквено-цифровий». Наприклад:

- 9-значний номер соціального страхування зашифровано в 9-значний рядок зашифрованого тексту
- 16-значний номер кредитної картки зашифровано в 16-значний рядок зашифрованого тексту
- 10-значний номер телефону зашифровано в 10-значний рядок зашифрованого тексту

Це особливо критично для організацій, які використовують застарілі системи, чиї кодові бази та моделі даних не можна змінити або надто обтяжливі та ризиковані для оновлення. Уявіть собі базу даних, яка зберігає конфіденційні дані про охорону здоров’я, яка була введена в дію майже 20 років тому і продовжує працювати. Більшість організацій мало схильні до ризиків, пов’язаних з реструктуризацією такої ключової виробничої системи.

Національний інститут стандартів і технологій (NIST) рекомендував два методи (під назвами FF1 і FF3-1) для реалізації шифрування зі збереженням формату. І FF1, і FF3-1 призначені для обробки 128-бітного блокового шифру (наприклад, AES) і обробки його вхідних і вихідних даних, щоб забезпечити збереження даних у потрібному форматі, але при цьому їх можна розшифрувати до вихідного значення. Однак створити ефективну стратегію захисту даних не так просто, як завантажити популярний алгоритм шифрування та пропрацювати його базову реалізацію. Алгоритми FPE мають бути ретельно реалізовані, щоб забезпечити конфіденційність, цілісність і доступність зашифрованих даних, зберігаючи при цьому критичні вимоги до формату, щоб не викликати збоїв у роботі.

FPE можна застосовувати в широкому діапазоні випадків використання. Ось кілька прикладів:

- Перевірка платіжної картки: у секторі роздрібної торгівлі та електронної комерції дані платіжної картки необхідно збирати та зберігати для здійснення платежів. Крім того, працівникам може знадобитися переглянути та перевірити останні чотири цифри інформації платіжної картки клієнта. FPE полегшує надання працівникам лише необхідної інформації, залишаючи інші дванадцять цифр захищеними.
- Застарілі бази даних: телекомунікаційна система може мати безліч застарілих систем, які потребують використання зашифрованих даних для безпеки. Однак організація може не мати можливості реструктуризувати свої бази даних для зберігання зашифрованих даних. З FPE структура баз даних може залишатися незмінною.

Хоча це лише два потенційних варіанти використання FPE, будь-які можливості шифрування конфіденційних даних у обмеженому середовищі зберігання можуть бути чудовим кандидатом для FPE. Окрім очевидної переваги безпеки шифрування даних, які раніше були відкритими, FPE потребує значно менше зусиль і ресурсів для впровадження порівняно з іншими рішеннями, оскільки не потребує модифікації рівня зберігання. Якщо зашифрований номер

кредитної картки все ще виглядає як номер кредитної картки, для бази даних не має значення, що його потрібно розшифрувати, перш ніж він стане придатним для використання.

2.4. Методи асиметричного шифрування

Коли йдеться про слово шифрування, ми думаємо про нього як про техніку, яка захищає дані за допомогою криптографічного ключа, і в цьому немає нічого поганого. Однак більшість людей не усвідомлюють, що існують різні типи методів шифрування. Асиметричне шифрування, також відоме як криптографія з відкритим ключем, є прикладом одного типу.

На відміну від симетричного шифрування , асиметричне шифрування шифрує та розшифровує дані за допомогою двох окремих, але математично пов’язаних криптографічних ключів. Ці ключі відомі як відкритий ключ і приватний ключ. Разом вони називаються пара відкритих і закритих ключів.

Асиметричне шифрування використовує два різні, але пов’язані ключі. Один ключ, відкритий ключ, використовується для шифрування, а інший, закритий ключ, призначений для дешифрування. Як випливає з назви, приватний ключ має бути закритим, щоб лише автентифікований одержувач міг розшифрувати повідомлення.

В основі асиметричного шифрування лежить криптографічний алгоритм. Цей алгоритм використовує протокол генерації ключів (різновид математичної функції) для генерації пари ключів. Обидва ключа математично пов'язані один з одним. Це співвідношення між ключами відрізняється від одного алгоритму до іншого.

Алгоритм в основному є комбінацією двох функцій – функції шифрування та функції дешифрування.

Асиметрична система ключів, також відома як система відкритих/приватних ключів, використовує два ключі. Один ключ залишається в таємниці – приватний ключ – тоді як інший ключ стає широко доступним для всіх, хто його потребує. Цей ключ називається відкритим ключем. Приватний і відкритий ключі математично пов’язані між собою, тому відповідний закритий ключ може розшифрувати лише ту

інформацію, зашифровану за допомогою відкритого ключа.

2.4.1. RSA

RSA – асиметричний алгоритм шифрування, заснований на факторизації добутку двох великих простих чисел. Лише той, хто знає ці цифри, може успішно розшифрувати повідомлення.

RSA часто використовується для захисту передачі даних між двома точками зв'язку. Однак його ефективність знижується при шифруванні великих обсягів даних. Незважаючи на це, цей метод шифрування є дуже надійним у передачі конфіденційних даних завдяки його відмінним математичним властивостям і складності.

За допомогою шифрування RSA повідомлення шифруються за допомогою коду, що називається відкритим ключем , яким можна відкрито ділитися. Завдяки певним математичним властивостям алгоритму RSA, коли повідомлення було зашифровано відкритим ключем, його можна розшифрувати лише за допомогою іншого ключа, відомого як закритий ключ. Кожен користувач RSA має пару ключів, що складається з його відкритого та закритого ключів. Як випливає з назви, закритий ключ повинен зберігатися в секреті.

Схеми шифрування з відкритим ключем відрізняються від шифрування з симетричним ключем, де і шифрування, і процес дешифрування використовують той самий закритий ключ. Ці відмінності роблять шифрування з відкритим ключем, наприклад RSA, корисним для спілкування в ситуаціях, коли не було можливості безпечно розповсюдити ключі заздалегідь.

Шифрування RSA часто використовується в поєднанні з іншими схемами шифрування або для цифрових підписів, які можуть підтвердити автентичність і цілісність повідомлення. Зазвичай він не використовується для шифрування цілих повідомлень або файлів, оскільки він менш ефективний і потребує більше ресурсів, ніж шифрування із симетричним ключем.

Щоб підвищити ефективність, файл зазвичай шифрується за допомогою

алгоритму симетричного ключа, а потім симетричний ключ шифрується за допомогою шифрування RSA. Відповідно до цього процесу лише суб'єкт, який має доступ до закритого ключа RSA, зможе розшифрувати симетричний ключ.

Без можливості доступу до симетричного ключа вихідний файл не можна розшифрувати. Цей метод можна використовувати для захисту повідомлень і файлів, не займаючи надто багато часу або споживаючи надто багато обчислювальних ресурсів.

Шифрування RSA можна використовувати в різних системах. Він може бути реалізований у OpenSSL, wolfCrypt, cryptlib та ряді інших криптографічних бібліотек.

Як одна з перших широко використовуваних схем шифрування з відкритим ключем, RSA заклала основи для більшості наших безпечних комунікацій. Він традиційно використовувався в TLS, а також був оригінальним алгоритмом для шифрування PGP . RSA все ще можна побачити в ряді веб-браузерів, електронної пошти, VPN, чату та інших каналів зв'язку.

RSA також часто використовується для встановлення безпечних з'єднань між клієнтами VPN і серверами VPN. За такими протоколами, як OpenVPN, TLS можуть використовувати алгоритм RSA для обміну ключами та встановлення безпечного каналу.

Отримавши відкритий ключ одержувача, відправник може використовувати його для шифрування даних, які він хоче зберегти. Після того, як його було зашифровано відкритим ключем, його можна розшифрувати лише закритим ключем із тієї самої пари ключів. Навіть той самий відкритий ключ не можна використовувати для розшифровки даних.

Коли одержувач отримує зашифроване повідомлення, він використовує свій закритий ключ для доступу до даних. Якщо одержувач хоче повернути повідомлення безпечним способом, він може зашифрувати своє повідомлення відкритим ключем сторони, з якою спілкується. Знову ж таки, після того, як інформацію було зашифровано за допомогою відкритого ключа, єдиний спосіб отримати доступ до інформації – через відповідний закритий ключ.

Таким чином, раніше невідомі сторони можуть використовувати шифрування RSA для безпечної передачі даних між собою. Значні частини каналів зв'язку, які ми використовуємо в нашому онлайн-житті, були побудовані на цій основі.

2.4.2. Криптографія еліптичної кривої (ECC)

Криптографія з еліптичною кривою – це новий тип криптографії з відкритим ключем, який є сильнішим за шифрування RSA. Він використовує коротші клавіші, що робить його швидшим. Він асиметричний, тобто ви можете використовувати його в протоколах SSL/TLS для посилення безпеки веб-зв'язку. Ви також можете використовувати ECC для одностороннього шифрування електронних листів, а також цифрових підписів у таких криптовалютах, як біткойн.

Криптографія з еліптичною кривою (ECC) є одним із найпотужніших, але найменш зрозумілих типів криптографії, які широко використовуються сьогодні.

За допомогою ECC ви можете використовувати менші ключі, щоб отримати той самий рівень безпеки. Маленькі ключі важливі, особливо в світі, де все більше криптографії виконується на менш потужних пристроях, таких як мобільні телефони. Хоча помножити два прості числа легше, ніж розкласти добуток на складові частини, коли прості числа стають дуже довгими, навіть сам крок множення може зайняти деякий час на малопотужному пристрої. Хоча ви, ймовірно, могли б продовжувати підтримувати захист RSA, збільшивши довжину ключа, що супроводжується зниженням продуктивності шифрування на клієнті. Здається, ECC пропонує кращий компроміс: високий рівень безпеки з короткими швидкими ключами.

Криптографія з еліптичною кривою (ECC) – це метод шифрування з відкритим ключем, заснований на теорії еліптичних кривих, який можна використовувати для створення швидших, менших і ефективніших криптографічних ключів.

ECC є альтернативою криптографічному алгоритму Rivest-Shamir-Adleman (RSA) і найчастіше використовується для цифрових підписів у криптовалютах, таких як Bitcoin та Ethereum, а також для одностороннього шифрування

електронних листів, даних і програмного забезпечення.

Еліптична крива не є еліпсом або овальною формою, але вона представлена як петля, що перетинає дві осі, які є лініями на графіку, що використовуються для вказівки положення точки. Крива є повністю симетричною або дзеркально відображеною вздовж осі x графіка.

Системи шифрування з відкритим ключем, такі як ECC, використовують математичний процес для об'єднання двох різних ключів, а потім використовують вихід для шифрування та дешифрування даних. Один – відкритий ключ, який відомий усім, а інший – закритий ключ, який відомий лише відправнику й одержувачу даних.

ECC генерує ключі за допомогою властивостей рівняння еліптичної кривої замість традиційного методу генерації як добутку великих простих чисел. З криптографічної точки зору точки вздовж графіка можна сформулювати за допомогою такого рівняння:

y?=x? + ax + b

ECC подібний до більшості інших методів шифрування з відкритим ключем, таких як алгоритм RSA та Діффі-Хеллмана. Кожен із цих механізмів криптографії використовує концепцію односторонньої функції, або люка. Це означає, що математичне рівняння з відкритим і закритим ключами можна використовувати для легкого переходу з точки A в точку Б. Але, не знаючи закритого ключа та залежно від використовуваного розміру ключа, дістатися з точки Б в точку A складно, якщо не неможливо, досягти.

ECC базується на властивостях набору значень, для яких можна виконувати операції з будь-якими двома членами групи, щоб створити третій член, який виходить із точок, де лінія перетинає осі, як показано зеленою лінією та трьома синіми крапки на діаграмі нижче, позначені A, B і C. Множення точки на кривій на число створює іншу точку на кривій (C). Візьміть точку C і перемістіть її до дзеркальної точки на протилежній стороні осі x, утворюючи точку D. Звідси лінія повертається до нашої вихідної точки A, створюючи перетин у точці E. Цей процес

можна завершити n кількість разів у межах визначеного максимального значення. n _це значення закритого ключа, яке вказує, скільки разів має бути виконано рівняння, закінчуючи кінцевим значенням, яке використовується для шифрування та дешифрування даних. Максимальне визначене значення рівняння стосується розміру використовуваного ключа.

ECC може забезпечити рівень безпеки, який вимагає менше обчислювальних ресурсів для шифрування та дешифрування даних порівняно з альтернативними методами, такими як RSA. Наприклад, ECC із використанням 256-бітного ключа потребує 3072-бітного ключа RSA для досягнення еквівалентного захисту. Оскільки ECC забезпечує еквівалентну безпеку з меншою обчислювальною потужністю та використанням ресурсів акумулятора, ніж RSA, він широко використовується для мобільних додатків і пристроїв Інтернету речей (IoT) з обмеженими ресурсами центрального процесора (CPU).

ECC пропонує кілька переваг порівняно з RSA:

- Він працює на пристроях з низьким ресурсом ЦП і пам'яті.
- Він шифрує та розшифровує швидше.
- Більший розмір ключа можна використовувати без значного збільшення розміру ключа або вимог до ЦП і пам’яті.

Вважається, що ECC є дуже безпечним, якщо розмір використовуваного ключа достатньо великий. Уряд США вимагає використання ECC із розміром ключа 256 або 384 біт для внутрішнього зв’язку, залежно від рівня чутливості інформації, що передається.

Але ECC не обов’язково є більш-менш безпечним порівняно з такими альтернативами, як RSA. Основною перевагою ECC є ефективність шифрування та дешифрування даних.

Висновки до розділу 2

На хмарних серверах зберігається величезна кількість даних, які передаються щодня. Практично неможливо виконувати повсякденні операції без зберігання чи

передачі цих величезних обсягів даних. Програмне забезпечення для шифрування даних гарантує, що дані захищені та безпечно передаються з одного каналу на інший.

Так, зашифровані дані можна зламати. Однак, залежно від рівня шифрування даних, рівень складності зростав.

Перш ніж розпочати впровадження шифрування даних, потрібно зрозуміти та визначити свої потреби безпеки. Рівень шифрування залежатиме від рівня безпеки, необхідного вам і вашій організації. Вибрати правильні засоби шифрування, які відповідають вашим потребам. Створити та реалізувати стратегію шифрування.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ГЕНЕРАТОРА АСИМЕТРИЧНИХ КЛЮЧІВ

3.1. Аналіз розробки продукту шифрування

Розробка та реалізація стратегії шифрування є спільними зусиллями ваших IT, операційних і управлінських команд. Ось чотири кроки для створення ефективної стратегії шифрування для захисту даних:

1. Класифікуйте дані. Перший крок – визначити, які дані шифрувати. Зрозумійте та класифікуйте різні типи даних , які ви надсилаєте та зберігаєте (наприклад, номери кредитних карток, інформацію про клієнтів, дані компанії), залежно від того, наскільки вони чутливі, як часто вони використовуються та як вони регулюються.

2. Визначте правильне рішення для шифрування даних. Використовуйте програмні засоби шифрування, щоб зашифрувати свої бази даних або окремі файли, які містять конфіденційну інформацію. Ви також можете покластися на стандартні програми та інструменти безпеки, такі як безпека електронної пошти, платіжні шлюзи та хмарна безпека, оскільки вони також містять функції шифрування бази даних, необхідні для захисту конфіденційних даних.

3. Застосуйте надійні методи керування ключами шифрування. Слідкуйте за своїми ключами шифрування, щоб навіть якщо хтось інший їх заволодіє, він не зможе отримати доступ до ваших даних. Це можна зробити, використовуючи рішення для керування ключами для зберігання та керування ключами шифрування.

4. Зрозумійте обмеження шифрування. Шифрування лише допомагає захистити конфіденційні дані від хакерів. Також не менш важливо мати надійні засоби кібербезпеки, такі як брандмауери та захист кінцевих точок.

3.2. Програмна реалізація генератора асиметричних ключів

Java Cryptography API дозволяє шифрувати та розшифровувати дані в Java, а також керувати ключами, підписувати та автентифікувати повідомлення, обчислювати криптографічні хеші та багато іншого. Термін криптографія часто скорочується до crypto , тому іноді ви побачите посилання на Javacrypto замість Java Cryptography. Хоча ці два терміни стосуються однієї теми.

Java cryptography API надається за допомогою того, що офіційно називається JavaCryptographyExtension. Розширення JavaCryptographyExtension також іноді називають абревіатурою JCE .

Розширення Java Cryptography Extension є частиною платформи Java протягом тривалого часу. Спочатку JCE зберігався окремо від Java, оскільки США мали певні обмеження на експорт технологій шифрування. Тому найсильніші алгоритми шифрування не були включені в стандартну платформу Java. Ви могли б отримати ці потужніші алгоритми шифрування для JavaJCE, якби ви були компанією в США, але решті світу довелося обійтися слабшими алгоритмами (або застосувати власні криптоалгоритми та підключити до JCE).

Архітектура криптографії Java (JCA) – це назва внутрішнього дизайну API криптографії Java. JCA структурована навколо деяких центральних класів загального призначення та інтерфейсів. Реальна функціональність цих інтерфейсів надається постачальниками. Таким чином, ви можете використовувати Cipherклас для шифрування та дешифрування деяких даних, але конкретна реалізація шифру (алгоритм шифрування) залежить від конкретного постачальника, який використовується.

API шифрування Java розділено між такими пакетами Java:

- java.security;
- java.security.cert;
- java.security.spec;
- java.security.interfaces;
- javax.crypto;
- javax.crypto.spec;
- javax.crypto.interfaces;

Основні класи та інтерфейси цих пакетів:

- Provider;
- SecureRandom;
- Cipher;
- MessageDigest;
- Signature;
- Mac;
- AlgorithmParameters;
- AlgorithmParameterGenerator;
- KeyFactory;
- SecretKeyFactory;
- KeyPairGenerator;
- KeyGenerator;
- KeyAgreement;
- KeyStore;
- CertificateFactory;
- CertPathBuilder;
- CertPathValidator;
- CertStore.

Клас Provider(java.security.Provider) є центральним класом в API криптографії Java. Щоб використовувати Java crypto API, вам потрібен Providerнабір. Java SDK поставляється з власним постачальником криптографії. Якщо ви не встановите явного постачальника криптографії, використовується стандартний постачальник Java SDK. Однак цей провайдер може не підтримувати алгоритми шифрування, які ви хочете використовувати. Тому вам, можливо, доведеться встановити власного постачальника криптографії. Один із найпопулярніших постачальників криптографії для API криптографії Java називається Bouncy Castle. Ось приклад, який встановлює BouncyCastleProvider:

```
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import java.security.Security;
public class ProviderExample {
public static void main(String [] args) {
Security.addProvider(new BouncyCastleProvider());
   }
}
```

Клас Cipher(javax.crypto.Cipher) представляє криптографічний алгоритм. Шифр можна використовувати як для шифрування, так і для дешифрування даних.

Ось як створити Cipher примірник Java:

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
```

У цьому прикладі створюється Cipher примірник, який використовує внутрішній алгоритм шифрування AES.

Метод Cipher.getInstance(...) приймає рядок, який визначає, який алгоритмшифруваннявикористовувати, атакожкількаіншихконфігураційалгоритму. У наведеному вище прикладі CBC частина – це режим, у якому може працювати алгоритм AES. PKCS5Padding Частина – це те, як алгоритм AES має обробляти останні байти даних для шифрування, якщо дані не відповідають розміру блоку 64 або 128 біт. межа.

Перед використанням Cipher примірника його необхідно ініціалізувати. Ви ініціалізуєте Cipher примірник, викликаючи його init () метод. Метод init () приймає два параметри:

1) Режим шифрування / дешифрування

2) Ключ

Перший параметр визначає, чи Cipher повинен шифрувати або розшифровувати дані. Другий параметр визначає ключ, за допомогою якого потрібно шифрувати або розшифровувати дані.

Ось Cipher.init() приклад Java:

```
byte [] keyBytes   = new byte []{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
String algorithm  = "RawBytes";
SecretKeySpec key = new SecretKeySpec(keyBytes, algorithm);
cipher.init(Cipher.ENCRYPT_MODE, key);
```

Щоб ініціалізувати Cipher примірник для дешифрування даних, треба використовувати Cipher.DECRYPT_MODE:

```
cipher.init(Cipher.DECRYPT_MODE, key);
```

Після правильної ініціалізації Cipher ви можете почати шифрувати або дешифрувати дані. Ви робите це, викликаючи методи Cipher update() або .doFinal() Цей update() метод використовується, якщо ви шифруєте або розшифровуєте частину більшого фрагменту даних. Метод doFinal() викликається, коли ви шифруєте останню частину великого фрагмента даних або якщо блок, який ви передаєте, doFinal()представляє повний блок даних для шифрування.

Ось приклад шифрування деяких даних за допомогою цього doFinal()методу:

```
byte [] plainText = "abcdefghijklmnopqrstuvwxyz".getBytes("UTF-8");
byte [] cipherText = cipher.doFinal(plainText);
```

Щоб розшифрувати дані, ви повинні передати зашифрований текст (зашифровані дані) у метод doFinal() або .doUpdate().

Щоб зашифрувати або розшифрувати дані, потрібен ключ. Є два типи ключів - залежно від того, який тип алгоритму шифрування ви використовуєте:

Симетричні ключі використовуються для симетричних алгоритмів шифрування. Алгоритми симетричного шифрування використовують один і той самий ключ для шифрування та дешифрування.

Асиметричні ключі використовуються для асиметричних алгоритмів шифрування. Алгоритми асиметричного шифрування використовують один ключ для шифрування, а інший – для дешифрування. Прикладами алгоритмів асиметричного шифрування є алгоритми шифрування відкритий ключ – закритий ключ.

Якимось чином сторона, якій потрібно розшифрувати дані, повинна знати ключ, необхідний для розшифровки даних. Якщо сторона, яка розшифровує дані, не збігається зі стороною, яка їх шифрує, якимось чином цим двом сторонам потрібно домовитися про ключ або обмінятися ключем. Це називається обміном ключами.

Ключі мають бути важко вгадати, щоб зловмисник не міг легко вгадати ключ шифрування. У прикладі з попереднього розділу про Cipher клас використовувався дуже простий, жорстко закодований ключ. На практиці це не дуже гарна ідея. Якщо їх ключ легко вгадати, зловмиснику легко розшифрувати зашифровані дані та, можливо, створити підроблені повідомлення.

Важливо, щоб ключ було важко вгадати. Таким чином, ключ повинен складатися з випадкових байтів. Чим більше випадкових, тим краще, а чим більше байтів, тим важче вгадати, оскільки існує більше можливих комбінацій.

Ви можете використовувати KeyGeneratorклас Java для створення більшої кількості випадкових ключів шифрування.

Ось KeyGenerator приклад Java:

```
KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
SecureRandom secureRandom = new SecureRandom();
int keyBitSize = 256;
keyGenerator.init(keyBitSize, secureRandom);
SecretKey secretKey = keyGenerator.generateKey();
```

Отриманий SecretKey примірник можна передати Cipher.init() методу ось так:

cipher.init(Cipher.ENCRYPT_MODE, secretKey);

Алгоритми асиметричного шифрування використовують пару ключів, що складається з відкритого та закритого ключів для шифрування та дешифрування даних. Щоб створити асиметричну пару ключів, ви можете використовувати KeyPairGenerator(java.security.KeyPairGenerator). Трохи KeyPairGenerator детальніше це описано в підручнику JavaKeyPairGenerator, але ось простий KeyPairGeneratorприклад Java:

SecureRandom secureRandom = new SecureRandom();

KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");

KeyPair keyPair = keyPairGenerator.generateKeyPair();

Java KeyStore – це база даних, яка може містити ключі. Java KeyStore представлено класом KeyStore(java.security.KeyStore). А KeyStore може містити такі типи ключів:

- Приватні ключі
- Відкриті ключі + сертифікати
- Секретні ключі

В асиметричному шифруванні використовуються закриті та відкритий ключі. Відкритий ключ може мати відповідний сертифікат. Сертифікат – це документ, який підтверджує особу особи, організації чи пристрою, які претендують на володіння відкритим ключем. Як доказ сертифікат зазвичай має цифровий підпис сторони, що перевіряє.

JavaKeytool – це інструмент командного рядка, який може працювати з файлами JavaKeyStore. Keytool може створювати пари ключів у файлі KeyStore, експортувати сертифікати та імпортувати сертифікати в KeyStore, а також виконувати кілька інших функцій.

Коли ви отримуєте зашифровані дані від когось іншого, як ви знаєте, що ніхто не змінював зашифровані дані на шляху до вас?

Загальне рішення полягає в тому, щоб обчислити дайджест повідомлення з даних перед його шифруванням, а потім зашифрувати дані та дайджест повідомлення та надіслати їх по мережі. Дайджест повідомлення – це хеш-значення,

обчислене на основі даних повідомлення. Якщо байт змінено в зашифрованих даних, дайджест повідомлення, розрахований на основі даних, також зміниться.

Отримавши зашифровані дані, ви розшифровуєте їх і обчислюєте з них дайджест повідомлення, а також порівнюєте розрахований дайджест повідомлення з дайджестом повідомлення, надісланим разом із зашифрованими даними. Якщо два дайджести повідомлень однакові, існує висока ймовірність (але не 100% гарантія), що дані не були змінені.

Ви можете використовувати JavaMessageDigest(java.security.MessageDigest) для обчислення дайджестів повідомлень. Ви викликаєте MessageDigest.getInstance() метод для створення MessageDigest примірника. Існує кілька різних алгоритмів дайджесту повідомлень. Вам потрібно сказати, який алгоритм ви хочете використовувати під час створення MessageDigest примірника. Більш детально це описано в підручнику JavaMessageDigest.

Ось приклад створення MessageDigestприкладу:

MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");

У цьому прикладі створюється MessageDigestпримірник, який використовує внутрішній криптографічний хеш-алгоритм SHA-256 для обчислення дайджестів повідомлень.

Щоб обчислити дайджест повідомлення деяких даних, ви викликаєте метод update() або .digest()

Метод update()можна викликати кілька разів, і дайджест повідомлення оновлюється внутрішньо. Коли ви передаєте всі дані, які хочете включити до дайджесту повідомлення, ви викликаєте digest()та отримуєте дані дайджесту повідомлення. Ось приклад update() кількаразового дзвінка з наступним digest() дзвінком:

```
MessageDigestmessageDigest = MessageDigest.getInstance("SHA-256");
byte [] data1 = "0123456789".getBytes("UTF-8");
byte [] data2 = "abcdefghijklmnopqrstuvxyz".getBytes("UTF-8");
messageDigest.update(дані1);
messageDigest.update(дані2);
```

byte [] digest = messageDigest.digest();

Ви також можете викликати digest()один раз, передаючи всі дані для обчислення дайджесту повідомлення. Ось як це виглядає:

MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");

byte [] data1 = "0123456789".getBytes("UTF-8");

byte [] digest = messageDigest.digest(data1);

Клас JavaMac використовується для створення MAC з повідомлення. Термін MAC є скороченням від MessageAuthenticationCode. MAC схожий на дайджест повідомлення, але використовує додатковий ключ для шифрування дайджесту повідомлення. Тільки маючи вихідні дані та ключ, ви можете перевірити MAC. Таким чином, MAC є більш безпечним способом захистити блок даних від модифікації, ніж дайджест повідомлення. Цей Масклас описано більш детально в підручнику JavaMac , але нижче наведено короткий вступ.

Ви створюєте Маспримірник Java, викликаючи Mac.getInstance()метод, передаючи як параметр назву алгоритму, який потрібно використовувати. Ось як це виглядає:

Macmac = Mac.getInstance("HmacSHA256");

Перш ніж ви зможете створити MAC з даних, ви повинні ініціалізувати Mac примірник за допомогою ключа. Ось приклад ініціалізації Mac примірника за допомогою ключа:

byte [] keyBytes = new byte []{0,1,2,3,4,5,6,7,8 ,9,10,11,12,13,14,15};

String algorithm= "RawBytes";

SecretKeySpec key = new SecretKeySpec(keyBytes, algorithm);

mac.init(key);

Після Mac ініціалізації примірника ви можете обчислити MAC з даних, викликавши метод update()або doFinal()Якщо у вас є всі дані для розрахунку MAC, ви можете doFinal()негайно викликати метод. Ось як це виглядає:

byte [] data = "abcdefghijklmnopqrstuvxyz".getBytes("UTF-8");

byte [] macBytes = mac.doFinal(дані);

Якщо у вас є доступ лише до даних в окремих блоках, викличте update() дані кілька разів і завершіть викликом doFinal(). Ось як це виглядає:

```
byte [] data = "abcdefghijklmnopqrstuvxyz".getBytes("UTF-8");
byte [] data2 = "0123456789".getBytes("UTF-8");
mac.update(дані);
mac.update(data2);
byte [] macBytes = mac.doFinal();
```

Клас Signature(java.security.Signature) використовується для даних цифрового підпису. Коли дані підписуються, з цих даних створюється цифровий підпис. Таким чином, підпис відокремлений від даних.

Цифровий підпис створюється шляхом створення дайджесту повідомлення (хеш) із даних і шифрування цього дайджесту повідомлення за допомогою закритого ключа пристрою, особи чи організації, яка має підписати дані. Дайджест зашифрованого повідомлення називається цифровим підписом.

Щоб створити Signature примірник, ви викликаєте Signature.getInstance(...) метод. Ось приклад створення Signature примірника:

```
Signaturesignature = Signature.getInstance("SHA256WithDSA");
```

Щоб підписати дані, ви повинні ініціалізувати Signature примірник у режимі підпису. Виробите це, викликаючи initSign(...) метод, який передає закритий ключ для підпису даних. Ось як виконується ініціалізація Signature примірника в режимі підпису:

```
signature.initSign(keyPair.getPrivate(), secureRandom);
```

Після Signature ініціалізації примірника його можна використовувати для підпису даних. Ви робите це, викликаючи update() передачу даних для підпису як параметра. Ви можете викликати update() метод кілька разів із додатковими даними для включення під час створення підпису. Коли всі дані будуть передані в update()метод, ви викликаєте sign()метод для отримання цифрового підпису. Ось як це виглядає:

```
byte [] data = "abcdefghijklmnopqrstuvxyz".getBytes("UTF-8");
signature.update(дані);
```

byte [] digitalSignature = signature.sign();

Щоб перевірити підпис, необхідно перевести Signature примірник у режим перевірки. Це робиться шляхом виклику initVerify(...)методу, який передає як параметр відкритий ключ для перевірки підпису. Тепер ініціалізація Signature примірника в режим перевірки виглядає так:

Signaturesignature = Signature.getInstance("SHA256WithDSA");

signature.initVerify(keyPair.getPublic());

Після ініціалізації режиму перевірки вивикликаєте update() метод із даними, які підписує підпис, і завершуєте викликом, verify() який повертає true або false залежно від того, чи можна перевірити підпис чи ні. Ось як виглядає перевірка підпису:

byte [] data2 = "abcdefghijklmnopqrstuvxyz".getBytes("UTF-8");

signature2.update(data2);

boolean verified = signature2.verify(digitalSignature);

Ось повний приклад створення та перевірки цифрового підпису за допомогою Signature класу:

SecureRandom secureRandom = newSecureRandom();

KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");

KeyPair keyPair = keyPairGenerator.generateKeyPair();

Signature signature = Signature.getInstance("SHA256WithDSA");

signature.initSign(keyPair.getPrivate(), secureRandom);

byte [] data = "abcdefghijklmnopqrstuvxyz".getBytes("UTF-8");

signature.update(дані);

byte [] digitalSignature = signature.sign();

Signature signature2 = Signature.getInstance("SHA256WithDSA");

signature2.initVerify(keyPair.getPublic());

signature2.update(data);

boolean verified = signature2.verify(digitalSignature);

System.out.println("перевірено = " + перевірено);

3.3. Реалізація утиліти для генерації ключів та шифрування повідомлень

На основі проаналізованої вище інформаціїї було розроблено утиліту для генерування асиметричних ключів та шифрування/дешифрування повідомлень. На Рис. 3.1 представлено скріншот першого вікна, на якому дві кнопки «Відправити» та «Отримати».

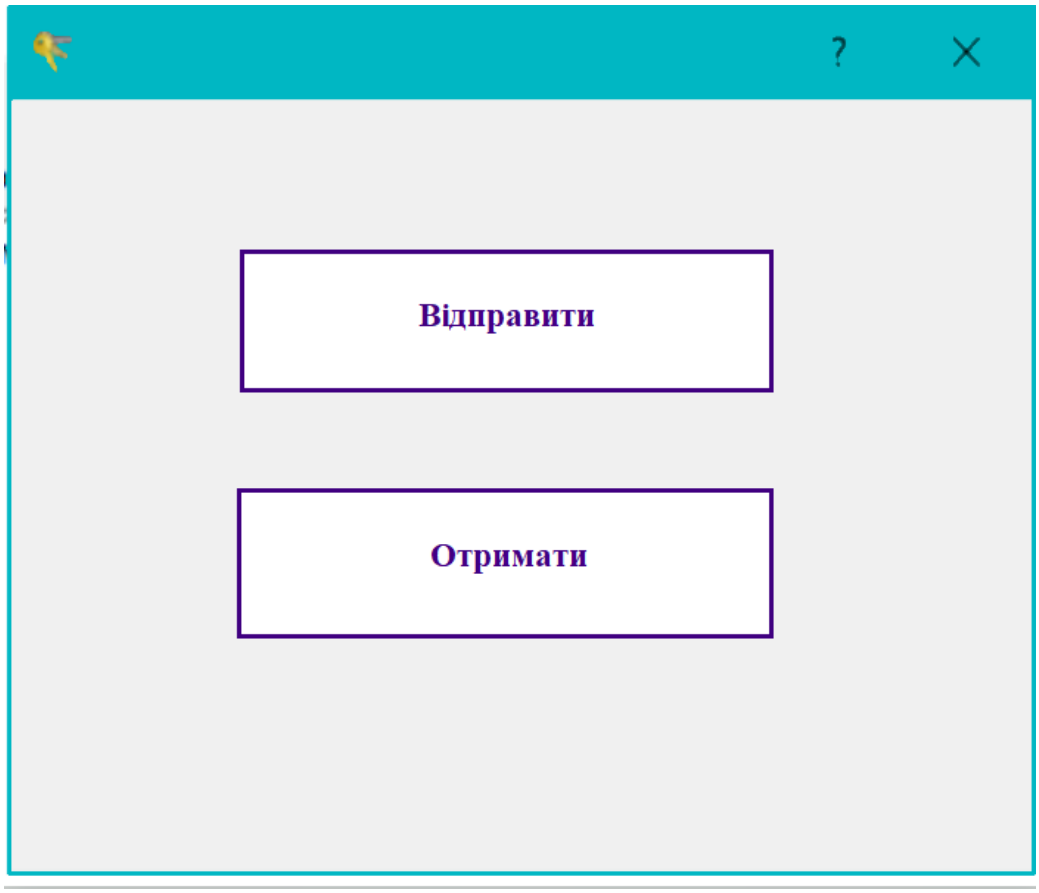

Рис.3.1. Скріншот першого вікна програми

При натисканні на кнопку «Отримати»(Рис.3.2) відкривається вікно з полями «Публічний ключ», «Приватний ключ», «Введіть отримане зашифроване повідомлення», «Розшифроване повідомлення», та кнопками «Генерувати ключі», «Поділитися публічним ключем», «Розшифрувати повідомлення». При натисканні на кнопку «Поділитися публічним ключем» пропонується передати ключ через будь-який месенджер, пошту, або скопіювати в файл. При натисканні на кнопку

«Розшифрувати повідомлення» відбувається розшифрування повідомлення приватним ключем, яке було зашифроване за допомогою публічного ключа.

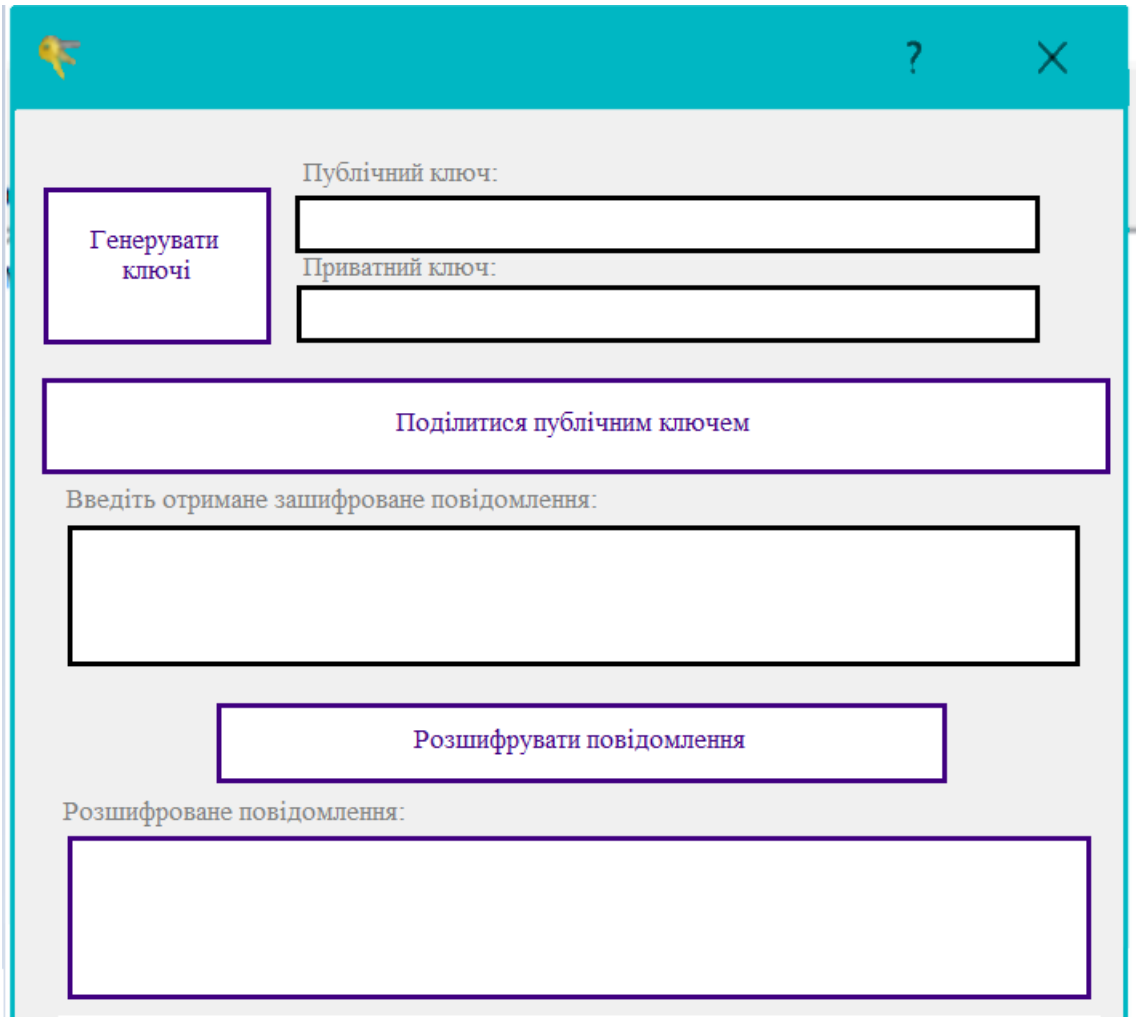

Рис.3.2. Вікно «Отримати»

При натисканні на кнопку «Генерувати ключі» відкривається вікно зображене на Рис. 3.3. Отримані ключі будуть відображені на формі «Отримати» в полях «Публічний ключ» та «Приватний ключ».

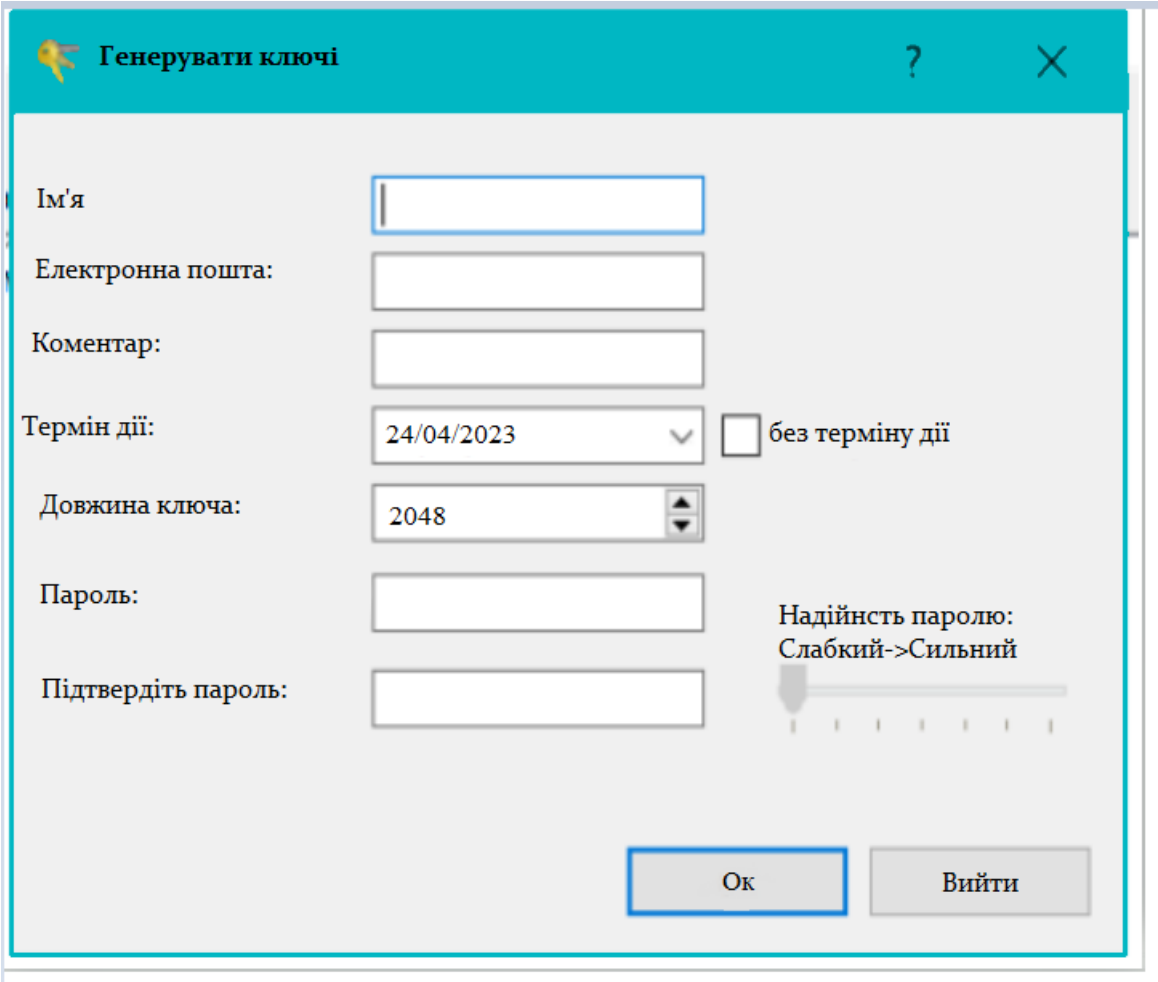

Рис. 3.3. Вікно генерації ключів

При натисканні на кнопку «Відправити» відкривається вікно(Рис. 3.4) з полями вводу публічного ключа, повідомлення, кнопками шифрування та відправлення повідомлення(для відправлення повідомлення пропонується передати його через довільний месенджер, пошту або скопіювати).

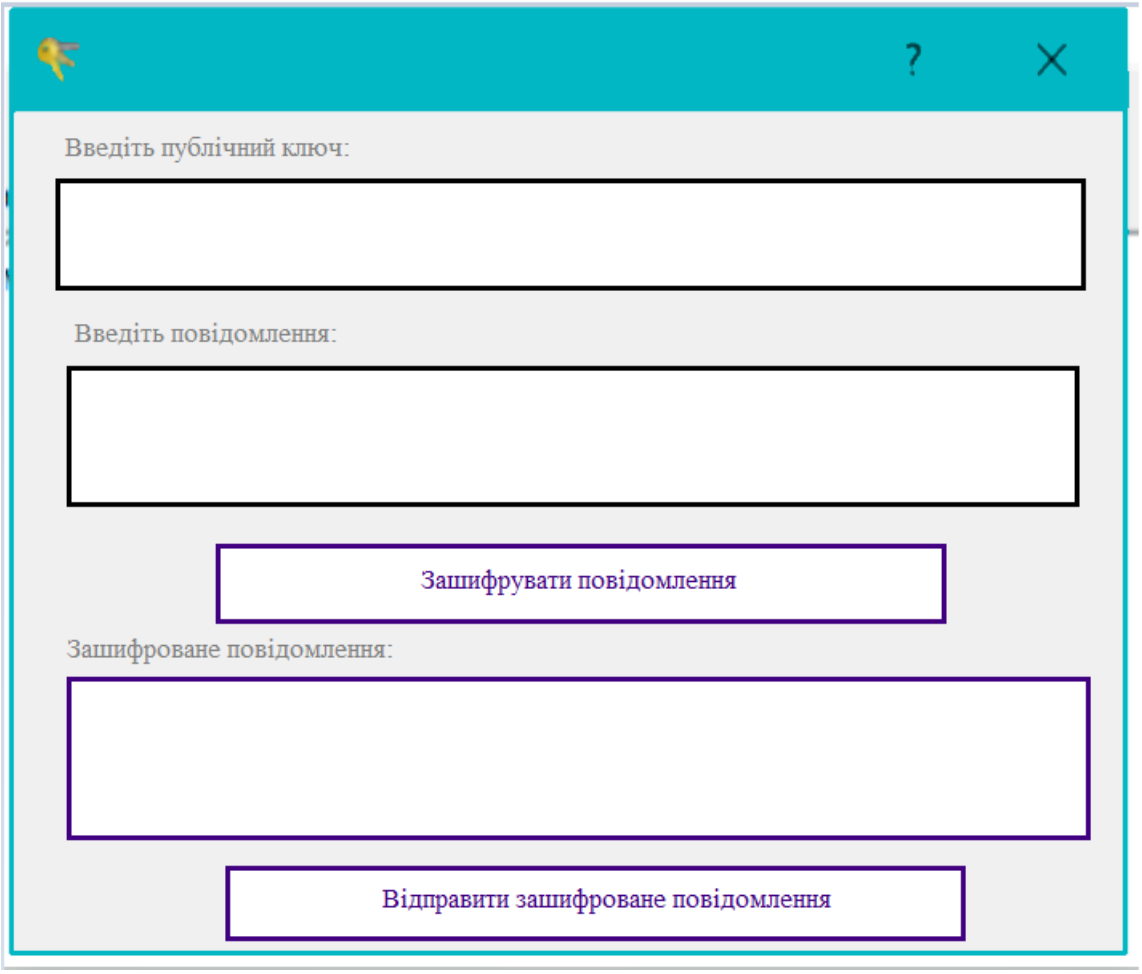

Рис. 3.4. Вікно «Відправити»

Зашифроване повідомлення можна відправити будь-яким чином і розшифрувати за допомогою форми «Отримати»

3.2. Застосування генератора асиметричних ключів в системі обміну даними

Таким чином, розроблений генератор можна використовувати для шифрування/дешифрування повідомлень, самі зашифровані повідомлення можна передавати в інших месенджерах, через пошту і т. д., а розшифровувати безпосередньо через програму. Хоча такий спосіб передачі може здатися незручним для користувача, але це може бути більш надійно, оскільки окрім шифрування повідомлень додається варіація способів їх передачі. На подальшу переспективу

звичайно можна додати можливість передачі прямо через додаток, але тоді потрібно буде подумати над додатковим захистом. Окрім цього цього, можна буде додати можливість шифрування та обміну файлами. Як бачимо, додаток можна використовувати в комплексі з іншими месенджерами, крім того є поле для подальших модифікацій додатку.

Висновки до розділу 3

На основі динамічних бібліотек Java було створено додаток, що дозволяє генерувати пару асиметричних ключів і обмінюватися зашифрованими повідомленнями з можливістю їх розшифрування. У додатку реалізовано форму генерації ключів, шифрування та розшифрування повідомлень.

Середовище розробки було обрано в силу універсальності додатків, які розробляються на ній. В якості мови програмування, на якій було розроблено додаток, було обрано Java, тому що ця мова набирає все більшої популярності серед розробників.

ВИСНОВКИ

У проведеному дослідженні була проаналізована наукова та методична література в області шифрування та захисту інформації. З проведеного аналізу літератури були зроблені висновки про принципи роботи та ризики безпеки в інформаційному просторі, та про методи шифрування, їх переваги та недоліки. На хмарних серверах зберігається величезна кількість даних, які передаються щодня. Практично неможливо виконувати повсякденні операції без зберігання чи передачі цих величезних обсягів даних. Програмне забезпечення для шифрування даних гарантує, що дані захищені та безпечно передаються з одного каналу на інший. Перш ніж розпочати впровадження шифрування даних, потрібно зрозуміти та визначити свої потреби безпеки. Рівень шифрування залежатиме від рівня безпеки, необхідного вам і вашій організації. Вибрати правильні засоби шифрування, які відповідають вашим потребам. Таким чином, обробивши теоретичну інформацію про безпеку інформаційного простору, зрозумівши принципи роботи шифрування та створивши програмний продукт для генерації асиметричних ключів та шифрування повідомлень було запропоновано використання утиліти для обміну вразливою інформацією. На основі динамічних бібліотек Java було створено додаток, що дозволяє генерувати пару асиметричних ключів і обмінюватися зашифрованими повідомленнями з можливістю їх розшифрування. У додатку реалізовано форму генерації ключів, шифрування та розшифрування повідомлень. Середовище розробки було обрано в силу універсальності додатків, які розробляються на ній. В якості мови програмування, на якій було розроблено додаток, було обрано Java, тому що ця мова набирає все більшої популярності серед розробників. В подальшому можлива перспектива розробки автономного простору обміну повідомленнями та додавання можливості шифрування та передачі файлів.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ISO/IEC 27005:200.335. Інформаційні технології. Методи захисту. Управління ризиками інформаційної безпеки [Текст]. – Київ: ДП "УкрНДНЦ", 200.335. – 60 с.

2. Керівництво з управління ризиками для систем інформаційних технологій. Рекомендації Національного інституту Стандартів і технологій (Guide for Conducting Risk Assessments. National Institute of Standards and Technology) [Текст]. – Gaithersburg: National Institute of Standards and Technology, 200.332. – 95 с.

3. Introducing OCTAVE Allegro: Improving the Information Security Risk Assessment Process [Текст] / R. A.Caralli, J. F. Stevens, L. R. Young, L. R. Wilson.– Бостон: Університет Карнегі-Меллон, 2007. – 0.3354 с.

4. Vipul Ved Prakash, Benjamin Trott, “Asymmetric Cryptography in Perl”, O'Reilly, 2001.

5. R.Coleridge, “The Cryptography API, or How to Keep a Secret”, MSDN, 1996.

6. D.Esposito, “Supporting CryptoAPI in Real-World Applications”, MSDN, 1997.

7. S.K.Parmar, “An introduction to security”, Fred Cohen &Associates, 2000.

8. R.L/Rivest, A.Shamir and L.Adleman, “A method for obtaining digital signatures and public key cryptosystems”. Commun. ACM, vol.21, p. 120-126, 1978.

9. W.Diffie and M.E.Hellman, “New directions in cryptograpgy”, IEEE Trans.Inf.Theory, vol.IT-22, N6, p.644-654, Nov. 1976.

10. A.Menezes, P.van Oorschot, S.Vanstone, “Handbook of applied cryptography”, CRC Press, 1996.

11. Cyber Security for Retail Services: Strategies that Empower your Business, Drive Innovation and Build Customer Trust [Електронний ресурс] // Symantec White Paper. – 200.335. – Режим доступу до ресурсу: https://www.symantec.com/content/dam/symantec/docs/whitepapers/cybersecurity-retail-en.pdf.

12. Cyber risk in retail: Protecting the retail business to secure tomorrow’s growth [Електронний ресурс]. – 200.337 – Режим доступу до ресурсу: https://www2.deloitte.com/content/dam/Deloitte/pe/Documents/risk/us-risk200.337-retail-cyber-risk-report-04070.335.pdf

13. John R. Vacca (Ed.) Computer and Information Security Handbook, 3rd Ed., Morgan Kaufmann, 50 Hampshire Street, 5th Floor, Cambridge, MA 02139, United States, 2017. - 1280 p.

14. Yuri Diogenes, Erdal Ozkaya. Cybersecurity - Attack and Defense Strategies Packt Publishing Ltd., Livery Place, 35 Livery Street, Birmingham B3 2PB, UK., 2018. - 384 p. 2018

15. Alghamdi M., Al-Ghamdi M.I. A.-M. Cybersecurity Best Practices Guide For IIROC Dealer Members, 2021 Electronic resource. Access mode:

16. Joseph Migga Kizza. Guide to Computer Network Security, Fifth Edition, Springer International Publishing AG 2020

17. Jie Wang, Zachary A. Kissel Introduction to Network Security Theory and Practice. John Wiley & Sons Singapore Pte Ltd, 2015. - 440 p.

18. Charles P. Pfleeger, Shari Lawrence Pfleeger, Jonathan Margulies. Security in Computing Fifth Ed. Pearson Education, Inc., 2015. - 944 p.

19. Karnel Erickson Networking Hacking – 2019

20. IEEE 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems. – New York: IEEE, 2000. – 29 p.

21. Jackson J.R. Networks of Waiting Lines. Operations Research, 1957, no. 5, pp. 518 – 521.

22. Jagerman D., Melamed B., Willinger W. Stochastic Modeling of Traffic Process, Frontiers in Queuing: Models, Methods and Problems // CRC Press. – 1996.

ДОДАТКИ

Додаток А

```
import java.io.FileInputStream;

import java.io.FileOutputStream;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

public class CodingFiles {

Cipher ecipher;

Cipher dcipher;

public CodingFiles() {

// TODO Auto-generated constructor stub

}
```

```
public SecretKey initCoding()

{

try{

SecretKey key = KeyGenerator.getInstance("DES").generateKey();

ecipher = Cipher.getInstance("DES");

dcipher = Cipher.getInstance("DES");

ecipher.init(Cipher.ENCRYPT_MODE, key);

dcipher.init(Cipher.DECRYPT_MODE, key);

return key;

}

catch(Exception e)

{

}
```

```
return null;

}

public void iniCoding(SecretKey key)

{

try{

ecipher = Cipher.getInstance("DES");

dcipher = Cipher.getInstance("DES");

ecipher.init(Cipher.ENCRYPT_MODE, key);

dcipher.init(Cipher.DECRYPT_MODE, key);

}

catch(Exception e)

{
```

```
}

}

public boolean fileEncode(String sourse,String result)

{

try{

FileInputStream inFile = new FileInputStream(sourse);

int bytesAvailable = inFile.available();

byte [] bytesReaded = new byte [bytesAvailable];

inFile.read(bytesReaded,0,bytesAvailable);

inFile.close();

byte [] br_enc = ecipher.doFinal(bytesReaded);

FileOutputStream outFile = new FileOutputStream(result);

outFile.write(br_enc);
```

```
outFile.close();

return true;

}

catch(Exception e)

{}

return false;

}

public boolean fileDecode(String sourse,String result)

{

try{

FileInputStream enc_inFile = new FileInputStream(sourse);

int enc_bytesAvailable = enc_inFile.available();
```

```
byte [] enc_bytesReaded = new byte [enc_bytesAvailable];

enc_inFile.read(enc_bytesReaded,0,enc_bytesAvailable);

enc_inFile.close();

byte [] br_dec =dcipher.doFinal(enc_bytesReaded);

FileOutputStream dec_outFile = new FileOutputStream(result);

dec_outFile.write(br_dec);

dec_outFile.close();

return true;

}

catch(Exception e)

{}

return false;

}
```

```
}

import java.io.BufferedInputStream;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.security.KeyFactory;

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.PrivateKey;

import java.security.PublicKey;

import java.security.SecureRandom;

import java.security.Signature;

import java.security.spec.X509EncodedKeySpec;
```

```
public class DigitalSignature {

public static void saveToFile (byte [] info, String filename) {

try {

FileOutputStream fos = new FileOutputStream(filename);

fos.write(info);

fos.close();

}

catch (Exception e){}

}

public static byte [] readFromFile (String fileName) {

byte [] info;

try {

FileInputStream fis = new FileInputStream(fileName);
```

```
info = new byte [fis.available()];

fis.read(info);

fis.close();

}

catch (Exception e) {info = new byte [0];}

return(info);

}

public static boolean CreateDigitalSignatureForFile(String puth)

{

try

{

KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA", "SUN");
```

```
SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");

keyGen.initialize(1024, random);

KeyPair pair = keyGen.generateKeyPair();

PrivateKey priv = pair.getPrivate();

PublicKey pub = pair.getPublic();

Signature dsa = Signature.getInstance("SHA1withDSA", "SUN");

dsa.initSign(priv);

FileInputStream fis = new FileInputStream(puth);

BufferedInputStream bufin = new BufferedInputStream(fis);

byte [] buffer = new byte [1024];

int len;

while (bufin.available() != 0)

{
```

```
len = bufin.read(buffer);

dsa.update(buffer, 0, len);

}

bufin.close();

byte [] realSig = dsa.sign();

saveToFile (realSig,puth+".sig");

byte [] key = pub.getEncoded();

saveToFile (key,puth+".pubkey");

//byte [] priv_key = priv.getEncoded();

//saveToFile (priv_key,"privkey_"+puth);

return true;

}
```

```
        catch (Exception e){}

        return false;

        }

        public static boolean TestedByDigitalSignature(String puth, String sign_puth, String
pubkey_puth){

        try{

        byte [] encKey = readFromFile(pubkey_puth);

        X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(encKey);

        KeyFactory keyFactory = KeyFactory.getInstance("DSA", "SUN");

        PublicKey pubKey = keyFactory.generatePublic(pubKeySpec);

        byte [] sigToVerify = readFromFile(sign_puth);

        Signature sig = Signature.getInstance("SHA1withDSA", "SUN");

        sig.initVerify(pubKey);

        FileInputStream datafis = new FileInputStream(puth);
```

```
BufferedInputStream bufin = new BufferedInputStream(datafis);

byte [] buffer = new byte [1024];

int len;

while (bufin.available() != 0)

{

len = bufin.read(buffer);

sig.update(buffer, 0, len);

}

bufin.close();

boolean verifies = sig.verify(sigToVerify);

return verifies;

}
```

```
catch(Exception e){}

return false;

}

}
```

Приложение Г

Код класса FInterfaceForm

```
import java.awt.Button;

import java.awt.Event;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import javax.crypto.SecretKey;
```

```
import javax.swing.JButton;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

public class FInterfaceForm extends JFrame{

/**

*

*/

private static final long serialVersionUID = 1L;

Button bt_enc = new Button("Поділитися публічним ключем");

Button bt_dec = new Button("Розшифрувати повідомлення");

Button bt_testdsig = new Button("Генерація ключів");
```

```
public FInterfaceForm() {

// TODO Auto-generated constructor stub

this.setLayout(null);

this.setBounds(200, 200, 420, 90);

this.setTitle("Java sercurity");

bt_enc.setBounds(0, 0, 200, 25);

bt_dec.setBounds(200, 0, 200, 25);

bt_dsig.setBounds(0, 25, 200, 25);

bt_testdsig.setBounds(200, 25, 200, 25);

this.add(bt_enc);

this.add(bt_dec);

this.add(bt_dsig);

this.add(bt_testdsig);
```

```
}

@SuppressWarnings("deprecation")

@Override

public boolean action(Event evt, Object arg1) {

// TODO Auto-generated method stub

if(evt.target instanceof Button)

{

if(evt.target.equals(bt_enc))

{

try{

if( jfc.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {

JFileChooser jfc_s = new JFileChooser();
```

```
        jfc_s.setDialogTitle("Введіть повідомлення");

        if( jfc_s.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {

        JFileChooser jfc_s_key = new JFileChooser();

        jfc_s_key.setDialogTitle("Поділитися ключем");

        if( jfc_s_key.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {

        CodingFiles cf = new CodingFiles();

        SecretKey sk = cf.initCoding();

        FileOutputStream fos = new
FileOutputStream((jfc_s_key.getSelectedFile()).getAbsolutePath());

        ObjectOutputStream oos = new ObjectOutputStream(fos);

        oos.writeObject(sk);

        oos.flush();

        oos.close();
```

```
if(cf.fileEncode((jfc.getSelectedFile()).getAbsolutePath(),
(jfc_s.getSelectedFile()).getAbsolutePath()))

JOptionPane.showMessageDialog(new JButton("Ok"), "Ключі згенеровано
успішно", JOptionPane.WARNING_MESSAGE);

else

JOptionPane.showMessageDialog(new JButton("Ok"),"Щось не так!",
JOptionPane.WARNING_MESSAGE);

}

}

}

}catch(Exception e){}

}

if(evt.target.equals(bt_dec))

{

try{
```

```
JFileChooser jfc = new JFileChooser();

jfc.setDialogTitle("Публічний ключ.");

if( jfc.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {

JFileChooser jfc_o = new JFileChooser();

jfc_o.setDialogTitle("Введіть ключ для розшифрування”)

if( jfc_o.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {

JFileChooser jfc_s = new JFileChooser();

jfc_s.setDialogTitle("Как сохранить расшифрованый файл?");

if( jfc_s.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {

CodingFiles cf = new CodingFiles();

FileInputStream fis = new
FileInputStream((jfc_o.getSelectedFile()).getAbsolutePath());

ObjectInputStream oin = new ObjectInputStream(fis);
```

```
        SecretKey ts = (SecretKey) oin.readObject();

        cf.iniCoding(ts);

        if(cf.fileDecode((jfc.getSelectedFile()).getAbsolutePath(),
(jfc_s.getSelectedFile()).getAbsolutePath()))

        {

        JOptionPane.showMessageDialog(new JButton("Ok"), "Файл
розшифровано","Вітаємо!", JOptionPane.WARNING_MESSAGE);

        }

        else

        {

        JOptionPane.showMessageDialog(new JButton("Ok"), "невірний ключ!",
JOptionPane.WARNING_MESSAGE);

        }

        oin.close();

        }
```

```
}

}

}catch(Exception e){}

}

if(evt.target.equals(bt_dsig))

{

try{

if( jfc.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {

if(DigitalSignature.CreateDigitalSignatureForFile((jfc.getSelectedFile()).getAbsolute
Path()))

JOptionPane.showMessageDialog(new JButton("Ok"), "Ключі
згенеровано","Вітаємо!", JOptionPane.WARNING_MESSAGE);

else
```

```
JOptionPane.showMessageDialog(new JButton("Ok"), “Щось пішло не так.!",
JOptionPane.WARNING_MESSAGE);

}

}catch(Exception e){}

}

if(evt.target.equals(bt_testdsig))

{

try{

JFileChooser jfc = new JFileChooser();

jfc.setDialogTitle("Генерація ключів.");

if( jfc.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {

if( jfc_dg.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {

JFileChooser jfc_key = new JFileChooser();

jfc_key.setDialogTitle("Введіть публічний ключ.");
```

```
if( jfc_key.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {

if(DigitalSignature.TestedByDigitalSignature((jfc.getSelectedFile()).getAbsolutePath
(), (jfc_dg.getSelectedFile()).getAbsolutePath(),
(jfc_key.getSelectedFile()).getAbsolutePath()))

JOptionPane.showMessageDialog(new JButton("Ok"), "Успішно!",
JOptionPane.WARNING_MESSAGE);

else

JOptionPane.showMessageDialog(new JButton("Ok"),"Danger!!!",
JOptionPane.WARNING_MESSAGE);

}

}

}

}catch(Exception e){}

}

}

return super.action(evt, arg1);

}

}
```