Міністерство освіти і науки України
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра економічної кібернетики та інформатики

МІЖДИСЦИПЛІНАРНА КУРСОВА РОБОТА
з спеціальності 124 «Системний аналіз» за першим (бакалаврським) рівнем вищої освіти на тему:

ЗАСОБИ ЗАХИСТУ ДАНИХ У ХМАРНИХ СЕРВІСАХ

Студента 4 курсу групи СА-41
спеціальності 124 «Системний аналіз»
Шуль Я. В.
(прізвище та ініціали) (підпис)

Керівник
_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали)

Національна шкала _____
Кількість балів_____ Оцінка:___
ECTS _____

Члени комісії: _____ _____
(прізвище та ініціали) (підпис)
_____ _____
(прізвище та ініціали) (підпис)
_____ _____
(прізвище та ініціали) (підпис)

м. Тернопіль – 2024 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра економічної кібернетики та інформатики

ЗАВДАННЯ НА МІЖДИСЦИПЛІНАРНУ КУРСОВУ РОБОТУ

група СА -41 курс4

Студента **Шуля Я. В.**

Тема міждисциплінарної курсової роботи:
Засоби захисту даних у хмарних сервісах

Основні розділи міждисциплінарної курсової роботи:

Вступ

1. Аналіз основних загроз безпеці даних у хмарних сервісах.
2. Оцінка ефективності сучасних методів захисту інформації.
3. Розробка рекомендацій щодо впровадження заходів безпеки.

Висновки

Рекомендована література: візьміть зі списку літератури
1.**Безпечна робота з Google Workspace**
2.**Fernet (симетричне шифрування)**
3.**Хмарна безпека та конфіденційність**

Додатки
Дата видачі завдання "____"_____ 2024р.
Термін представлення роботи на кафедру "____"_____ 2024р.

Керівник міжисциплінарної курсової роботи _____

ЗМІСТ

ВСТУП

У сучасному світі технології стрімко розвиваються, і використання хмарних сервісів стає невід'ємною частиною повсякденної роботи як приватних осіб, так і великих компаній. Зокрема, у сфері системної аналітики, де дані є основою для ухвалення рішень, питання безпеки даних набуває особливої актуальності. Обробка та зберігання даних в умовах хмарних платформ стають дедалі зручнішими та економічно вигіднішими. Однак, разом із зростанням популярності хмарних сервісів зростають і виклики, пов'язані із забезпеченням конфіденційності, цілісності та доступності даних у цих середовищах.

Основні проблеми, які виникають у хмарних середовищах, зводяться до ризиків несанкціонованого доступу, витоку інформації та загрози цілісності даних. Для системних аналітиків, які опрацьовують великий обсяг конфіденційних даних, наприклад, фінансових показників чи поведінкових характеристик користувачів, такі ризики можуть призвести до серйозних фінансових втрат такі як розривання договору з потенційним клієнтом або втрати довіри з боку партнерів. Сучасні хмарні сервіси забезпечують базовий рівень захисту, але користувачі також повинні мати інструменти для самостійного шифрування й контролю даних, щоб підвищити безпеку й відповідати стандартам інформаційної безпеки.

Ця робота досліджує можливості захисту даних у хмарних сервісах через застосування сучасних протоколів шифрування. Спираючись на симетричне шифрування, ми розглянемо бібліотеку `cryptography` в Python та її інструмент `Fernet`, який забезпечує просте й водночас надійне шифрування для середовищ із хмарним зберіганням. Розуміння і впровадження таких технологій стає необхідним для всіх фахівців, що працюють із даними, оскільки це дозволяє знизити ризики втрат і порушень конфіденційності, а також сприяє підвищенню рівня довіри до використання хмарних технологій у критично важливих галузях, як-от фінанси, охорона здоров'я та держуправління.

У цій курсовій роботі проаналізовано сучасний підходи до захисту даних у хмарних сервісах та практичне застосування інструментів шифрування для підвищення безпеки даних. У роботі буде розглянуто як теоретичні, так і практичні аспекти використання протоколів шифрування, зокрема під час зберігання зашифрованих файлів у хмарі. Практична частина дослідження зосереджена на створенні демонстраційного проекту, який покаже процес шифрування файлів і подальшого безпечного зберігання їх у хмарному середовищі. Специфічні завдання роботи включають дослідження ризиків зберігання інформації в хмарних середовищах, огляд інструментів для забезпечення безпеки даних, а також практичну реалізацію алгоритму шифрування і його оцінку.

Таким чином, проведене дослідження дозволить глибше зрозуміти, як працюють методи шифрування на практиці та як їх інтеграція може покращити захист даних у хмарних сервісах. Це допоможе зробити хмарні технології більш безпечними та надійними, забезпечуючи користувачів ефективними інструментами для збереження конфіденційності та цілісності їхніх даних.

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ МЕТОДІВ ЗАХИСТУ ДАНИХ У ХМАРНИХ СЕРВІСАХ

1.1. Розвиток технологій захисту даних у хмарних сервісах

Розвиток хмарних технологій та методів захисту даних у цьому середовищі є результатом багаторічної еволюції обчислювальних систем і засобів забезпечення безпеки. Спочатку хмарні обчислення були орієнтовані на забезпечення віддаленого зберігання даних і доступу до ресурсів, але через поширення хмарних платформ, обсяги даних почали швидко зростати, як і кількість загроз для їхньої безпеки.

Перші хмарні технології здебільшого використовувалися в середині 2000-х років, коли великі компанії, такі як Amazon з своїм Amazon Web Services, Google з Google Compute Engine та Microsoft запустила подібний сервіс Azure у 2010 році (рисунок 1.1).

Рисунок 1.1 - Логотипи відомих хмарних сервісів

Спочатку безпека була обмежена традиційними засобами захисту мережевої інфраструктури, такими як міжмережеві екрани та VPN. Однак, із зростанням складності систем і кількості користувачів, хмарні провайдери стикнулися з новими загрозами. У відповідь на ці виклики були впроваджені різноманітні протоколи і стандарти шифрування, які стали основою для сучасної безпеки в хмарних середовищах.

Перші спроби захисту даних включали прості методи шифрування, такі як SSL (Secure Sockets Layer), які забезпечували базовий рівень безпеки для передачі даних. Згодом SSL було замінено на вдосконалений протокол TLS (Transport Layer Security), який забезпечував вищу стійкість до атак, оскільки використовував більш надійні методи шифрування. TLS став базовим стандартом захисту даних при передачі через мережу, що було критичним для захисту даних у хмарних системах.

У 2010-х роках із зростанням використання хмарних платформ та їх проникненням у бізнес-середовище особлива увага приділялася безпеці зберігання даних. Саме тоді розпочалося впровадження стандартів шифрування AES (Advanced Encryption Standard) для захисту даних, які зберігалися в хмарних середовищах. Згодом шифрування на стороні користувача (client-side encryption) дозволило кінцевим користувачам шифрувати свої дані перед завантаженням у хмару, що підвищило рівень довіри до хмарних технологій. Це нововведення стало важливим кроком на шляху до захисту конфіденційності, оскільки навіть самі провайдери не мали доступу до ключів шифрування.

Також важливою віхою розвитку стало створення стандартів ISO/IEC 27017 та 27018, які визначають вимоги до безпеки інформації в хмарних середовищах та захисту персональних даних у хмарних сервісах. Вони стали міжнародним еталоном для хмарних провайдерів, які прагнуть забезпечити високий рівень захисту даних і дотримання вимог конфіденційності.

З розвитком технологій контейнеризації та впровадженням таких інструментів, як Docker та Kubernetes, з'явилися нові виклики щодо безпеки даних, оскільки контейнери забезпечують швидке розгортання та масштабування додатків, але разом із цим виникають ризики безпеки через розподіленість даних та високий рівень інтеграції. Щоб захистити дані в таких умовах, почали використовувати додаткові методи аутентифікації, управління доступом і моніторингу, що забезпечують цілісність та доступність даних в умовах контейнеризації.

На сьогодні, поряд із протоколами шифрування та стандартами захисту, широко використовуються й інші технології, такі як багатоетапна автентифікація (MFA - Multifactor Authentication), токени доступу та контроль прав доступу на основі ролей (RBAC - Role-Based Access Control). Також активно розвиваються механізми шифрування даних під час обробки (end-to-end encryption) та квантовий захист інформації, які будуть актуальними у майбутньому, забезпечуючи стійкість до більш складних атак.

Отже, історія розвитку захисту даних у хмарних технологіях ілюструє поступовий перехід від базових методів мережевої безпеки до комплексних протоколів і стандартів, спрямованих на забезпечення конфіденційності, цілісності та доступності інформації в хмарних середовищах.

1.2. Захист даних у хмарних середовищах

Захист даних у хмарних середовищах означає забезпечення конфіденційності, цілісності та доступності інформації, що зберігається та обробляється на віддалених серверах. Оскільки дані користувачів завантажуються на хмарні платформи, відповідальність за їх безпеку часто розподіляється між клієнтом і провайдером послуг. Умовно, заходи захисту можна поділити на три основні категорії: мережевий захист, шифрування даних та управління доступом.

Основним інструментом для забезпечення безпеки є шифрування даних під час передачі (TLS/SSL) та зберігання (AES, RSA). Таке шифрування запобігає несанкціонованому доступу до інформації, навіть якщо фізичний або логічний доступ до неї буде отримано зловмисником. Існує два основних підходи до шифрування(рисунок 1.2):

1. Шифрування на боці клієнта (client-side), коли дані шифруються до їх завантаження у хмару, і тільки клієнт має ключ для дешифрування;
2. Шифрування на боці сервера (server-side), коли за шифрування відповідає сам хмарний провайдер.

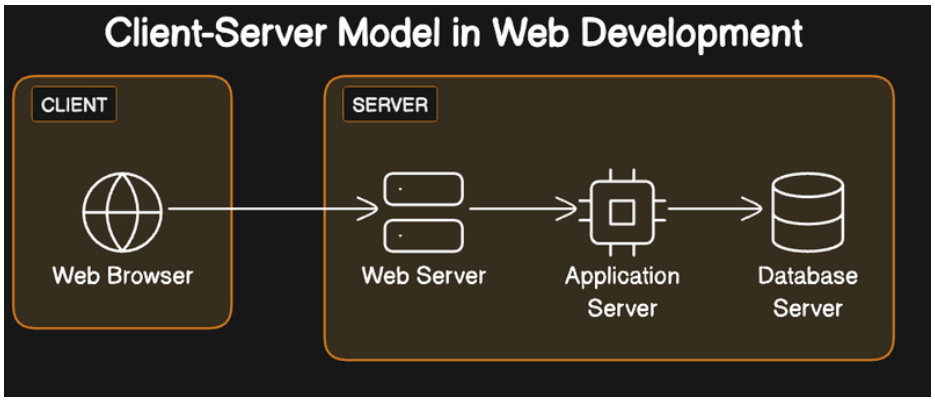


Рисунок 1.2 - Спрощений приклад взаємодії клієнта з сервером

Важливим аспектом є **управління доступом**, яке включає механізми автентифікації користувачів (MFA, токени доступу) та контроль прав доступу. Наприклад, модель контролю доступу на основі ролей (RBAC) дозволяє визначати, які користувачі можуть отримати доступ до певних ресурсів.

Ще однією складовою захисту даних у хмарі є **моніторинг та логування** дій користувачів і системи. Це дозволяє виявити та відстежити потенційні порушення безпеки, оскільки усі дії в хмарі можуть бути зареєстровані та проаналізовані.

1.3. Ризики та загрози безпеки у хмарних сервісах

Попри численні переваги хмарних сервісів, такі як гнучкість, масштабованість та економічність, вони супроводжуються суттєвими ризиками

та загрозами безпеки. Основні загрози стосуються конфіденційності, цілісності та доступності даних, що є критичними факторами для захисту інформації.

1. Несанкціонований доступ до даних

Оскільки хмарні сервіси доступні через мережу, користувачі та дані стають потенційними мішенями для хакерських атак. Злом облікових записів або використання вразливостей у програмному забезпеченні може дати зловмисникам можливість несанкціоновано отримати доступ до даних. Особливо вразливими є сервіси без багатофакторної автентифікації (MFA) та контролю доступу на основі ролей (RBAC).

2. Витік або втрата даних

Через централізацію зберігання та обробки інформації у хмарних середовищах ризик витоку або втрати даних зростає. Навіть у випадках, коли компанія має надійні внутрішні механізми захисту, у хмарі частина відповідальності покладається на провайдера. Недостатнє шифрування або збої в безпекових процесах провайдера можуть призвести до витоку критично важливих даних.

3. Атаки на інфраструктуру

Зловмисники можуть здійснювати атаки на інфраструктуру хмарних сервісів, такі як DDoS Attack (Distributed Denial-of-Service Attack), що впливають на доступність послуг. Хоча багато хмарних провайдерів пропонують захист від таких атак, потужніші або координовані атаки можуть вплинути на роботу хмарних сервісів, роблячи їх недоступними для користувачів.

4. Неправильне налаштування конфігурацій

Неправильне налаштування конфігурацій, такі як публічний доступ до конфіденційних даних або недостатньо налаштовані права доступу, можуть призвести до небезпечних ситуацій. Часто це є результатом людської помилки або недостатньої обізнаності про безпекові стандарти. Зокрема, відкриті бази

даних або файлові сховища без захисту паролем є типовими прикладами подібних помилок.

5. Загроза внутрішніх зловживань

Внутрішні користувачі, такі як співробітники чи партнери, можуть свідомо або випадково пошкодити дані чи надати несанкціонований доступ до них. Хмарні платформи з високою інтеграцією та численними користувачами особливо вразливі до внутрішніх зловживань, що ускладнює контроль за доступом.

6. Юридичні та регуляторні ризики

Регуляторні вимоги щодо захисту даних (як-от GDPR) вимагають дотримання конкретних стандартів для зберігання та обробки інформації. У випадках, коли дані розміщуються у хмарі, можуть виникати ризики, пов'язані з юрисдикцією, особливо якщо дані зберігаються в іншій країні з менш жорсткими вимогами щодо конфіденційності.

Таким чином, безпека хмарних сервісів вимагає комплексного підходу, де важливо врахувати як технологічні, так і організаційні аспекти захисту. Але не зважаючи на ці всі вразливості хмарні можливості лишаються одним з кращих виборів для великих бізнесів, а також і для звичайних користувачів.

1.4. Методології забезпечення захисту даних

Для надійного захисту даних у хмарних сервісах застосовують кілька основних методів. Найважливішими є шифрування, обмеження доступу, постійний моніторинг та регулярне створення резервних копій.

По-перше, **шифрування** даних робить їх недоступними для зловмисників, навіть якщо вони зможуть отримати до них доступ. Дані шифруються при передачі (наприклад, коли ви пересилаєте файли через інтернет) і коли зберігаються на сервері (тобто в хмарі). Для цього часто використовують спеціальні програми чи бібліотеки. Наприклад, у Python можна використовувати

бібліотеку cryptography, яка за допомогою коду перетворює текст або файли у зашифрований формат перед їхнім завантаженням у хмару (рисунок 1.3).

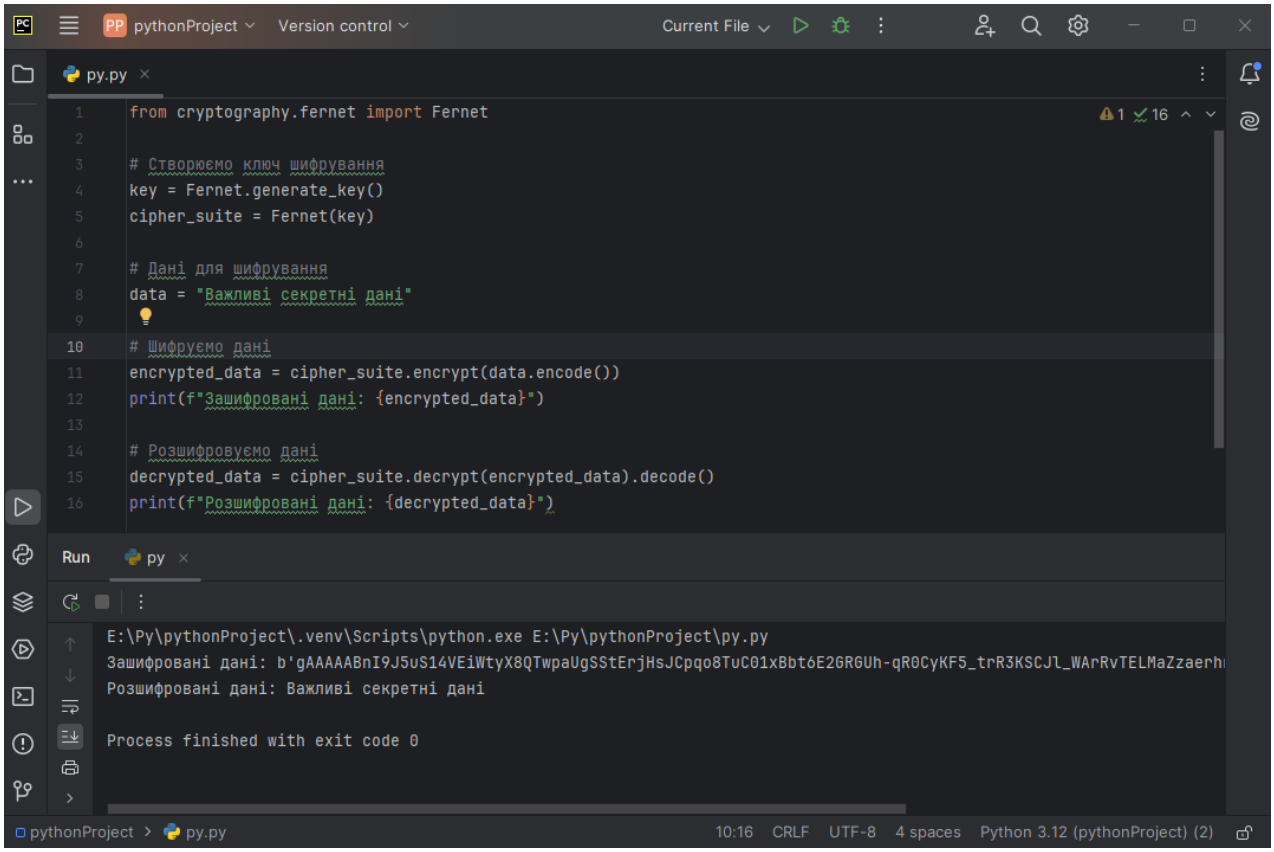


Рисунок 1.3 – Код показує базове шифрування та розшифрування тексту

Ще одним важливим способом захисту є **контроль доступу**. Це означає, що до кожного файлу мають доступ тільки ті користувачі, яким він дійсно потрібен. Для цього часто використовують багатофакторну автентифікацію (коли потрібно, наприклад, ввести пароль та код із телефону) та налаштовують різні рівні доступу, як у системі Azure AD (рисунок 1.4). Наприклад, працівник відділу кадрів не матиме доступу до фінансових даних, якщо це не потрібно для його роботи.

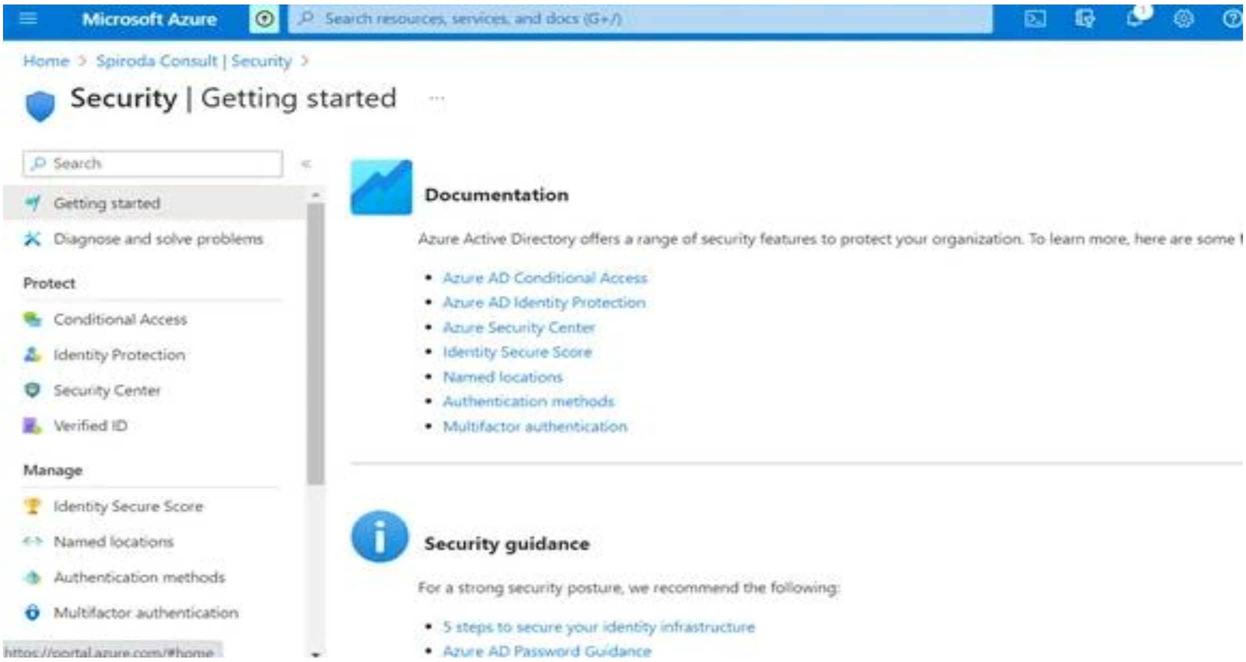


Рисунок 1.4 – Вікно налаштування безпеки Microsoft Azure

Постійний **моніторинг та аудит** дій у хмарному середовищі допомагає відстежувати, хто і що робить із даними. Наприклад, сервіс CloudTrail від AWS зберігає інформацію про кожну дію користувачів у системі (рисунок 1.5). Це допомагає виявити підозрілі дії, як-от великі завантаження або видалення файлів.

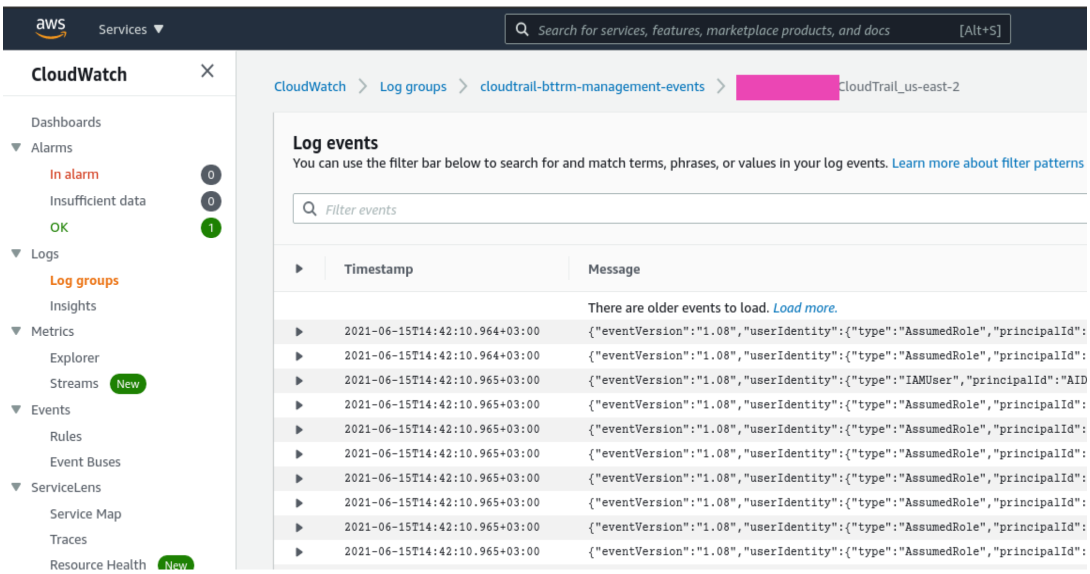


Рисунок 1.5 – Звіт CloudTrail з інформацією про logs груп

Резервне копіювання також є важливим елементом захисту. Це регулярне збереження копій даних, щоб їх можна було відновити у разі втрати чи пошкодження. Наприклад, Google Cloud пропонує просте рішення для автоматичного резервного копіювання даних (рисунок 1.6).

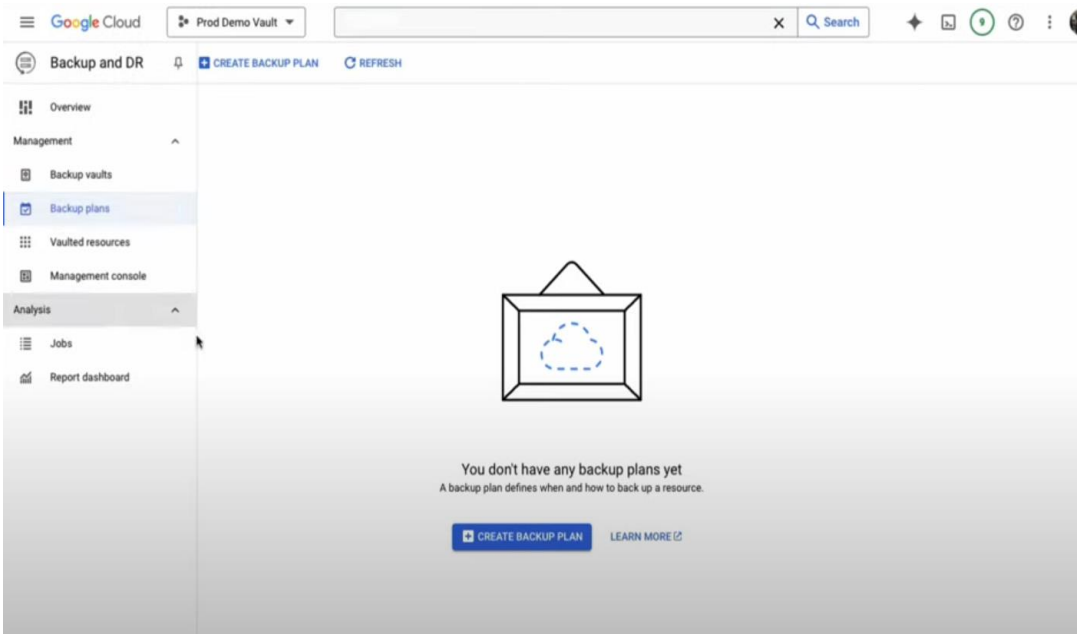


Рисунок 1.6 – Інтерфейс Backup and Recovery у Google Cloud

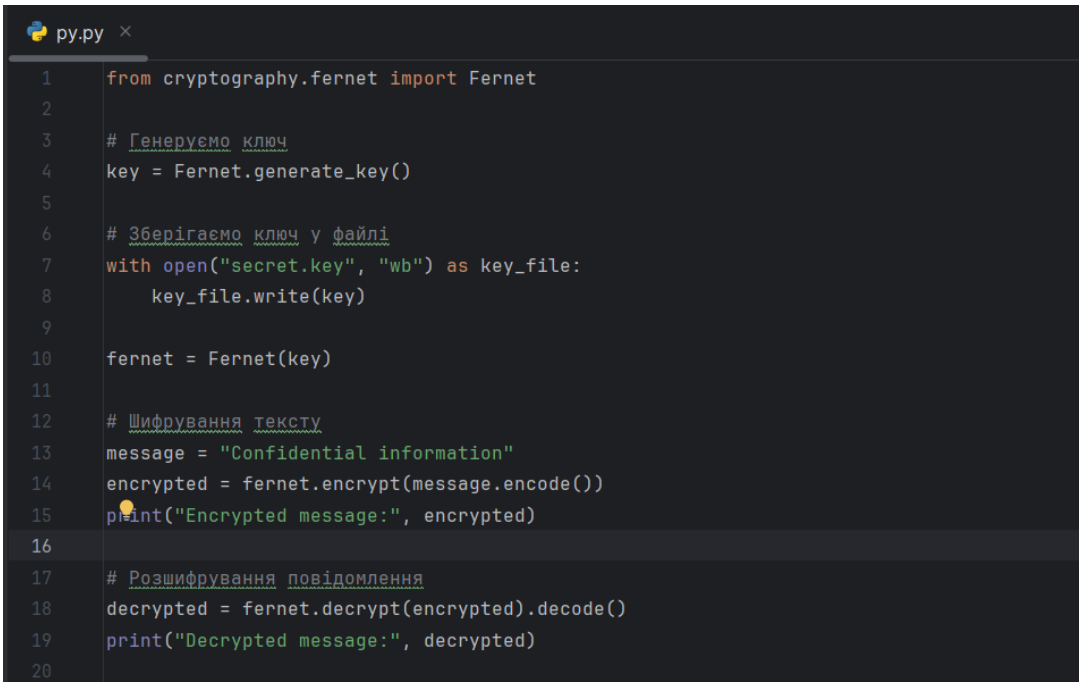
Отже, захист даних у хмарі базується на шифруванні, обмеженні доступу, моніторингу та резервному копіюванні. Ці кроки допомагають зберегти наші дані безпечними та доступними навіть у випадку проблем чи загроз.

РОЗДІЛ 2. ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ШИФРУВАННЯ ДАНИХ

2.1. Огляд бібліотеки cryptography у Python

Бібліотека `cryptography` у Python є потужним інструментом для роботи із зашифрованими даними, яка забезпечує простий і зручний спосіб реалізації шифрування та розшифрування інформації. Її використовують для забезпечення конфіденційності, цілісності та безпеки даних у програмах і системах, що працюють з конфіденційною інформацією. Бібліотека підтримує два основних підходи до шифрування, це симетричне та асиметричне шифрування. Симетричне шифрування використовує один ключ для шифрування і розшифрування даних, а асиметричне для шифрування використовується один ключ (публічний), а для розшифрування інший (приватний).

Найпопулярнішим інструментом у бібліотеці cryptography для симетричного шифрування є **Fernet**, який дозволяє безпечно шифрувати та розшифровувати дані за допомогою одного ключа. Ключ генерується автоматично або визначається користувачем і має надійний формат, що робить його захищеним від атак.



```python
from cryptography.fernet import Fernet

# Генеруємо ключ
key = Fernet.generate_key()

# Зберігаємо ключ у файлі
with open("secret.key", "wb") as key_file:
    key_file.write(key)

fernet = Fernet(key)

# Шифрування тексту
message = "Confidential information"
encrypted = fernet.encrypt(message.encode())
print("Encrypted message:", encrypted)

# Розшифрування повідомлення
decrypted = fernet.decrypt(encrypted).decode()
print("Decrypted message:", decrypted)
```

Рисунок 2.1 – приклад коду генерації ключа і його шифрування

Спочатку (рисунок 2.1) генерується секретний ключ, який використовується для створення об'єкта Fernet. Далі, за допомогою методу encrypt, повідомлення шифрується, а методом decrypt — розшифровується (рисунок 2.2).

```
E:\Py\pythonProject\.venv\Scripts\python.exe E:\Py\pythonProject\py.py
Encrypted message: b'gAAAAABnJnfJyv3qsb0ihZG7lbv0cL648GvmivKTEPeufKwjotMTnc3kL_4hIGAeH
Decrypted message: Confidential information

Process finished with exit code 0
```

Рисунок 2.2 – показ виводу із зашифрованим ключем

Бібліотека також підтримує **асиметричне шифрування** (наприклад, RSA), що часто використовується для обміну ключами в захищених мережах. cryptography надає можливості для створення публічних і приватних ключів,

підпису даних і їх перевірки, що може бути корисним для забезпечення автентичності і цілісності повідомлень.

Таким чином, cryptography є універсальним і зручним інструментом для роботи з даними, які потребують захисту. Її можливості підходять як для базових, так і для більш складних задач, пов'язаних із шифруванням і управлінням ключами, що робить її ідеальним вибором для роботи з конфіденційною інформацією у Python.

2.2. Принципи роботи Fernet для симетричного шифрування

Fernet — це метод симетричного шифрування, доступний користувачам модуля cryptography у Python, який спрощує шифрування та розшифрування тексту і надає простий інтерфейс для новачків у криптографії. Fernet використовує алгоритм Advanced Encryption Standard (AES) для кодування і декодування повідомлень.

Fernet забезпечує, що повідомлення, зашифроване з його допомогою, не може бути прочитане або змінене без відповідного ключа. Він також підтримує реалізацію ротації ключів через клас MultiFernet. Також задіює 128-бітний секретний ключ, який генерується випадковим чином. Цей ключ необхідний як для шифрування, так і для розшифрування даних. Під час шифрування Fernet виконує такі кроки:

- Генерація одноразового номера (nonce): 128-бітний випадковий вектор ініціалізації (IV), який забезпечує унікальність шифротексту навіть для однакових вхідних даних.
- Шифрування: Використовуючи алгоритм AES у режимі CBC (Cipher Block Chaining), дані шифруються з використанням секретного ключа та згенерованого IV.
- Додавання HMAC: Для забезпечення цілісності та автентичності до шифротексту додається HMAC (Hash-based Message Authentication Code), обчислений за допомогою алгоритму SHA256.

- Формування токена: Остаточний шифротекст містить версію, часову мітку, IV, зашифровані дані та HMAC, закодовані у форматі Base64.

Під час розшифрування Fernet виконує зворотні дії:

- Перевірка HMAC: Переконується, що шифротекст не був змінений, перевіряючи HMAC.
- Розшифрування: Використовуючи AES у режимі CBC з відповідним IV та секретним ключем, дані розшифровуються.
- Перевірка часової мітки: За потреби можна перевірити, чи не минув певний час з моменту шифрування, що може бути корисним для обмеження терміну дії повідомлення.

Таким чином, Fernet є ефективним інструментом для симетричного шифрування в Python, який забезпечує конфіденційність, цілісність та автентичність даних. Схожий приклад (рисунок 2.3).

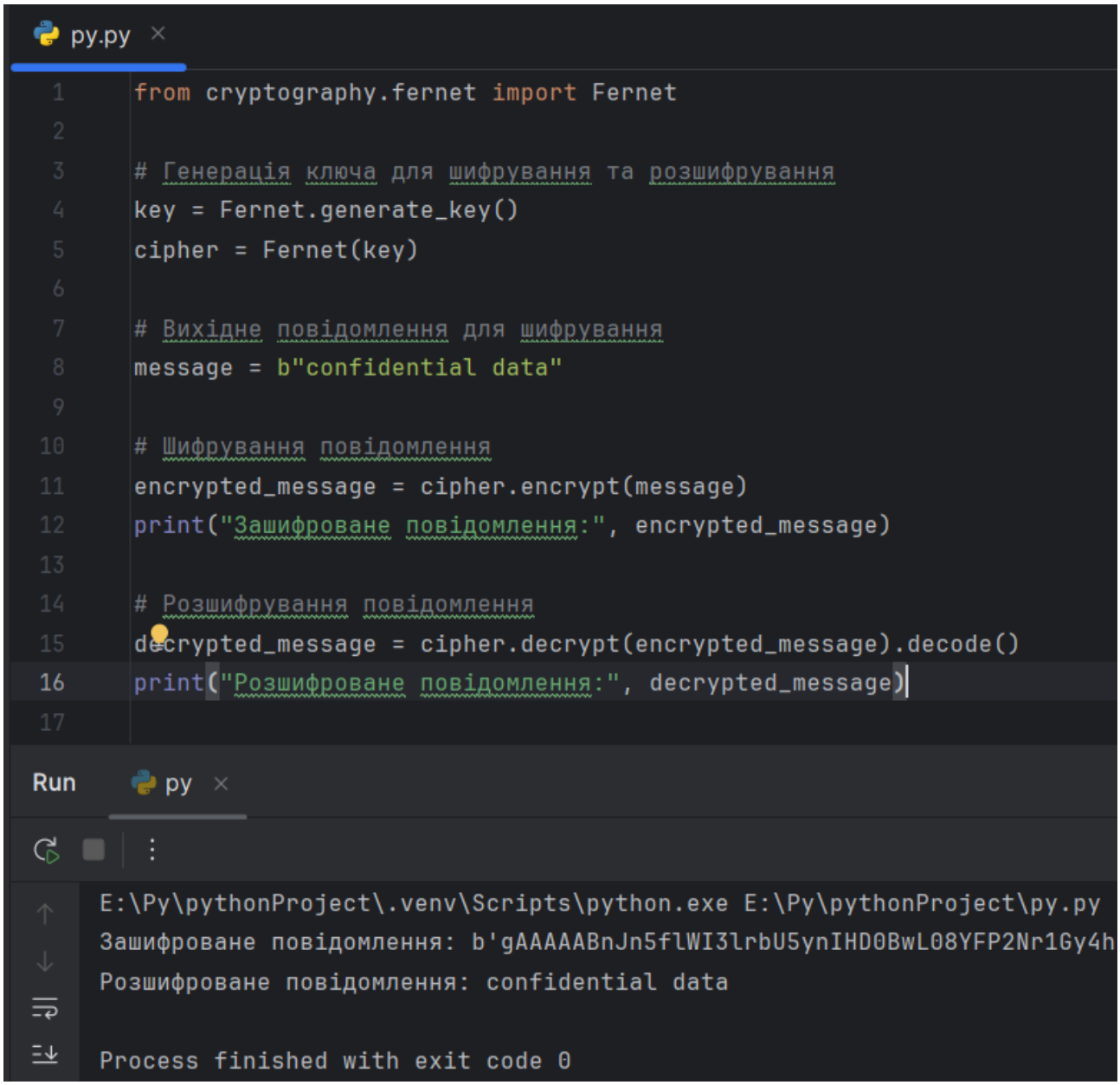


Рисунок 2.3 - код генерації, шифрування і розшифрування ключа

2.3. Використання сервісів для безпечного зберігання даних у хмарі

Для безпечного зберігання даних у хмарі насамперед важливо обрати надійного провайдера, який пропонує високий рівень захисту. Такі сервіси, як Google Диск, Dropbox та Microsoft OneDrive, забезпечують стабільну синхронізацію, інтеграцію з іншими додатками та гнучке налаштування безпеки, що робить їх зручними і водночас безпечними для користувачів.

Перед завантаженням даних у хмару варто подбати про їхнє шифрування. Це створює додатковий рівень захисту інформації, навіть якщо до файлів буде

отримано несанкціонований доступ. Використання бібліотеки cryptography у Python, наприклад, дозволяє легко шифрувати документи, знижуючи ризик витоку.

Ще один важливий аспект безпеки — це налаштування прав доступу. Хмарні сервіси дають змогу визначати, хто може переглядати, редагувати або видаляти файли, і таким чином обмежити доступ до інформації для небажаних користувачів. Це особливо важливо для компаній та команд, де одночасно працюють декілька користувачів.

Не менш корисною функцією є двофакторна автентифікація, яка додає додатковий рівень захисту, вимагаючи підтвердження входу через інший пристрій або код. Вона суттєво зменшує ймовірність того, що зловмисники зможуть отримати доступ до облікового запису, навіть маючи ваш пароль.

І, звісно ж, регулярне резервне копіювання залишається важливою частиною безпеки даних. Зберігання копій файлів на локальних пристроях або в іншому хмарному середовищі допомагає уникнути втрати інформації у разі технічних збоїв або інших непередбачуваних ситуацій.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРАКТИЧНОГО ПРОЄКТУ ЗАХИСТУ ДАНИХ

3.1. Підготовка та створення інформаційної структури даних

У цьому розділі я створив таблицю з умовними даними робітників, який буде використано для подальшого шифрування та зберігання у хмарному сервісі. Для цього скористався мовою програмування Python у IDEA PyCharm та її стандартними бібліотеками для створення csv файлу. В самому коді буде невеличкий опис того що робить код (рисунок 3.1).

py.py

```
import csv

# Визначення даних
data = [
    {"ID": 1, "Name": "John Doe", "Email": "john.doe@example.com"},
    {"ID": 2, "Name": "Jane Smith", "Email": "jane.smith@example.com"},
    {"ID": 3, "Name": "Bob Johnson", "Email": "bob.johnson@example.com"}
]

# Визначення назв полів
fieldnames = ["ID", "Name", "Email"]

# Створення та запис у файл CSV
with open("employees.csv", mode="w", newline="") as file:
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(data)

print("Файл 'employees.csv' успішно створено.")
```

Run py

```
E:\Py\pythonProject\.venv\Scripts\python.exe E:\Py\pythonProject\py.py
Файл 'employees.csv' успішно створено.

Process finished with exit code 0
```

Рисунок 3.1 - створення csv формат файлу за допомогою Python

3.2. Реалізація шифрування даних за допомогою cryptography.fernet

Після того коли було створено таблицю з даними тепер треба буде встановити саму бібліотеку cryptography для подальшої роботи над шифруванням. Якщо бібліотека cryptography ще не встановлена, її можна додати за допомогою менеджера пакетів pip, ввівши *pip install cryptography* у рядок терміналу в самому PyCharm.

Дальше пишеться код який для шифрування та розшифрування даних використовується симетричний ключ, який необхідно зберігати в безпечному місці (рисунок 3.2).

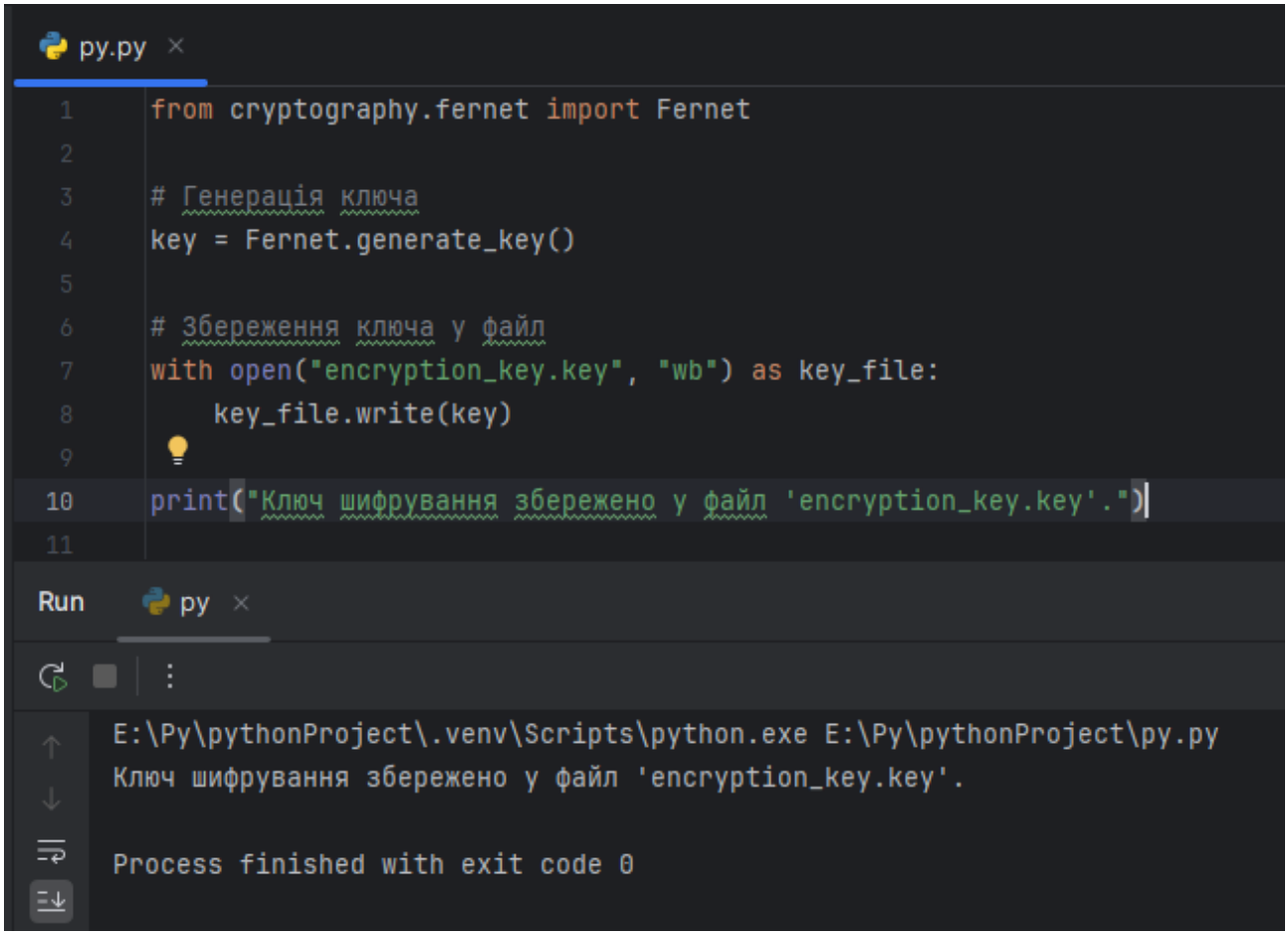


Рисунок 3.2 - python код для створення ключа і його збереження

Імпорт модуля Fernet: з бібліотеки cryptography.fernet.

Генерація ключа: метод Fernet.generate_key() створює 32-байтовий ключ у форматі Base64.

Збереження ключа: відкриває файл *encryption_key.key* у режимі запису байтів ("wb") та записуємо туди згенерований ключ. Він буде збережений там де і сам код.

Після генерації та збереження ключа переходимо до шифрування вмісту файлу *employees.csv*, який також находиться в тій самій папцій що й і сам зашифрований ключ *encryption_key.key* (рисунок 3.3).

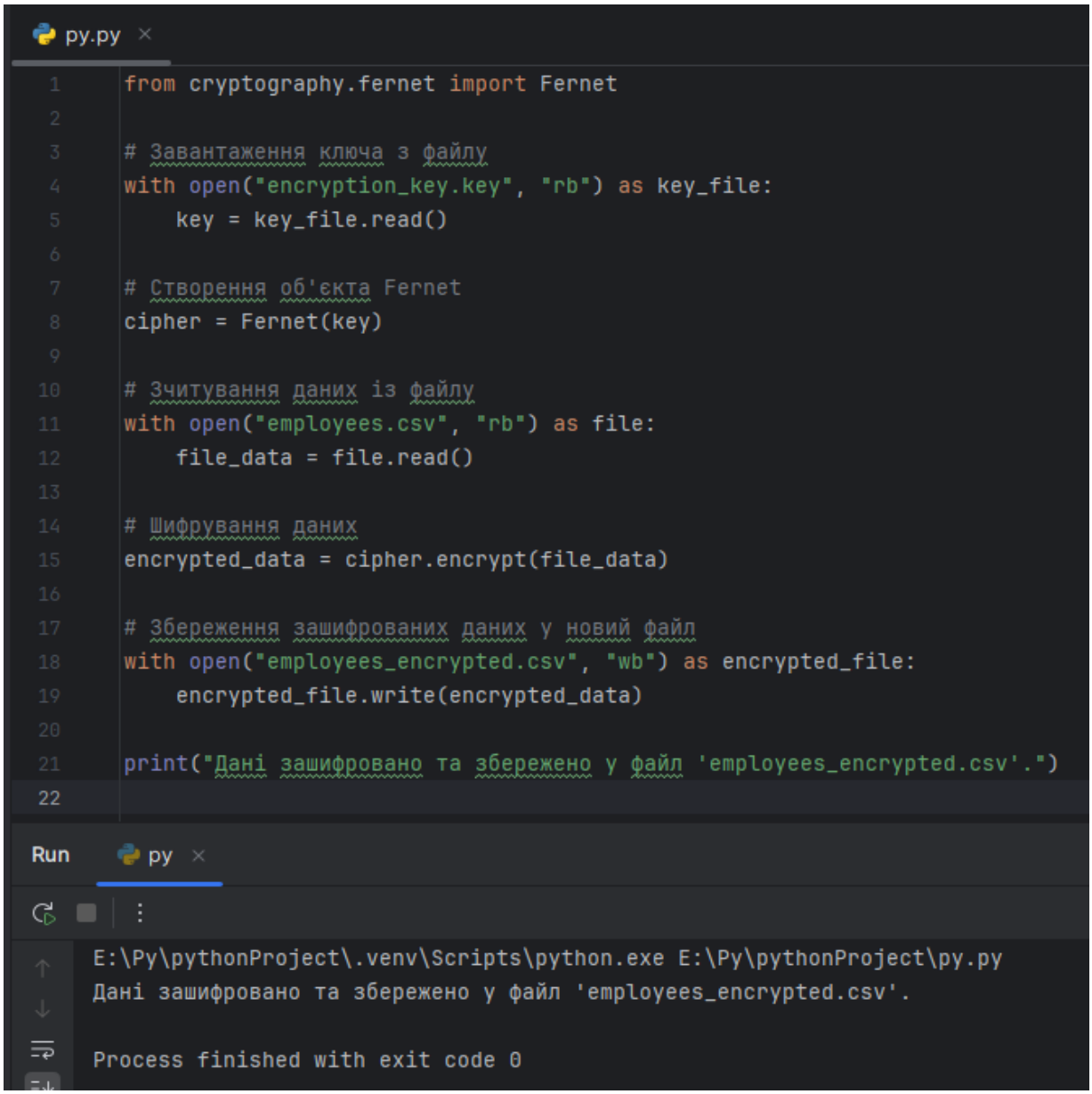


Рисунок 3.3 - шифрування таблиці

Завантаження ключа: відкриває файл *encryption_key.key* у режимі читання байтів ("rb") та зчитуємо ключ.

Створення об'єкта Fernet: ініціалізуєм об'єкт Fernet з використанням зчитаного ключа.

Зчитування даних із файлу: відкриває файл *employees.csv* у режимі читання байтів та зчитуємо його вміст.

Шифрування даних: метод encrypt() об'єкта Fernet шифрує зчитані дані.

Збереження зашифрованих даних: відкриває новий файл *employees_encrypted.csv* у режимі запису байтів та записує туди зашифровані дані. Після виконання цього коду у робочій директорії з'явиться файл *employees_encrypted.csv*, який міститиме зашифровані дані з початкового файлу *employees.csv*. Цей файл можна безпечно зберігати або передавати, оскільки без відповідного ключа його вміст неможливо прочитати (рисунок 3.4).

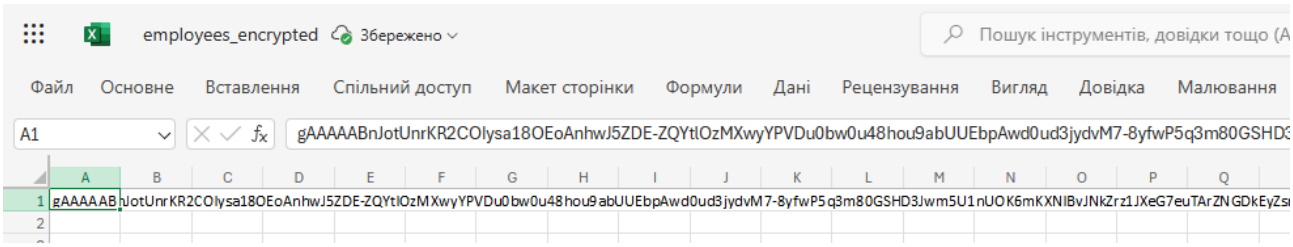


Рисунок 3.4 - зашифрована таблиця

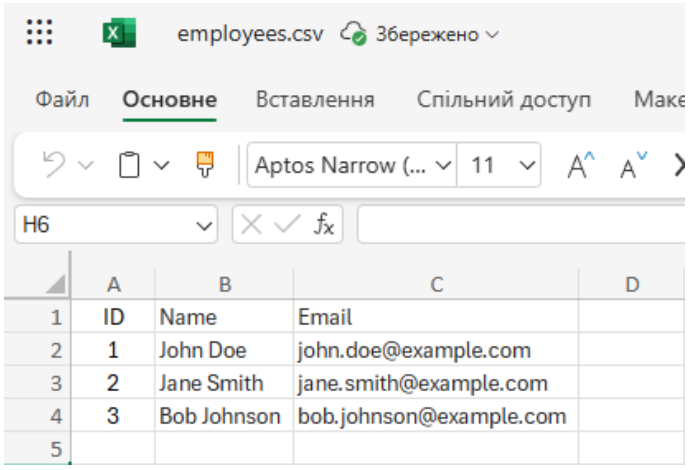


Рисунок 3.5 - не зашифрована таблиця

3.3. Зберігання зашифрованого файлу у хмарному сервісі

Після шифрування даних важливо забезпечити їх надійне зберігання. Хмарні сервіси, такі як Google Drive, надають зручні інструменти для зберігання та доступу до файлів.

Але для того щоб використовувати Google Drive API потрібно получити доступ до нього зареєструвавшись і створивши новий проек з певними дозволами (рисунок 3.6).

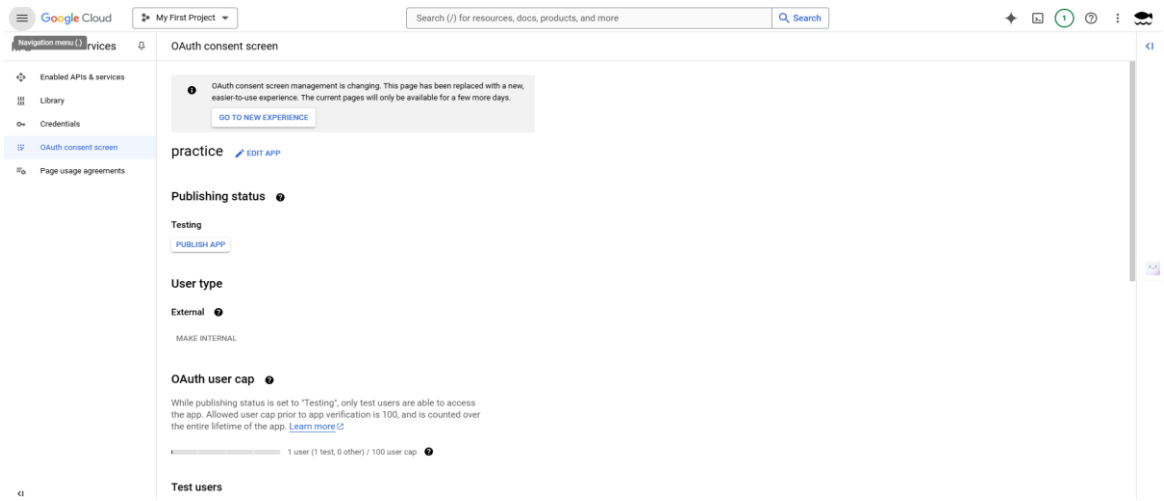


Рисунок 3.6 - сторінка проетку

Після того з GoogleCloude можна викачати json файл для того щоб за допомогою Python коду закинути зашифрований файл з робітниками на хмару.

Для роботи з Google Drive API у Python потрібно встановити бібліотеки *pip install google-auth google-auth-oauthlib google-auth-httplib2 google-api-python-client* у терміналі у PyCharm. Після налаштування доступу та встановлення бібліотек можна завантажити зашифрований файл на Google Drive:

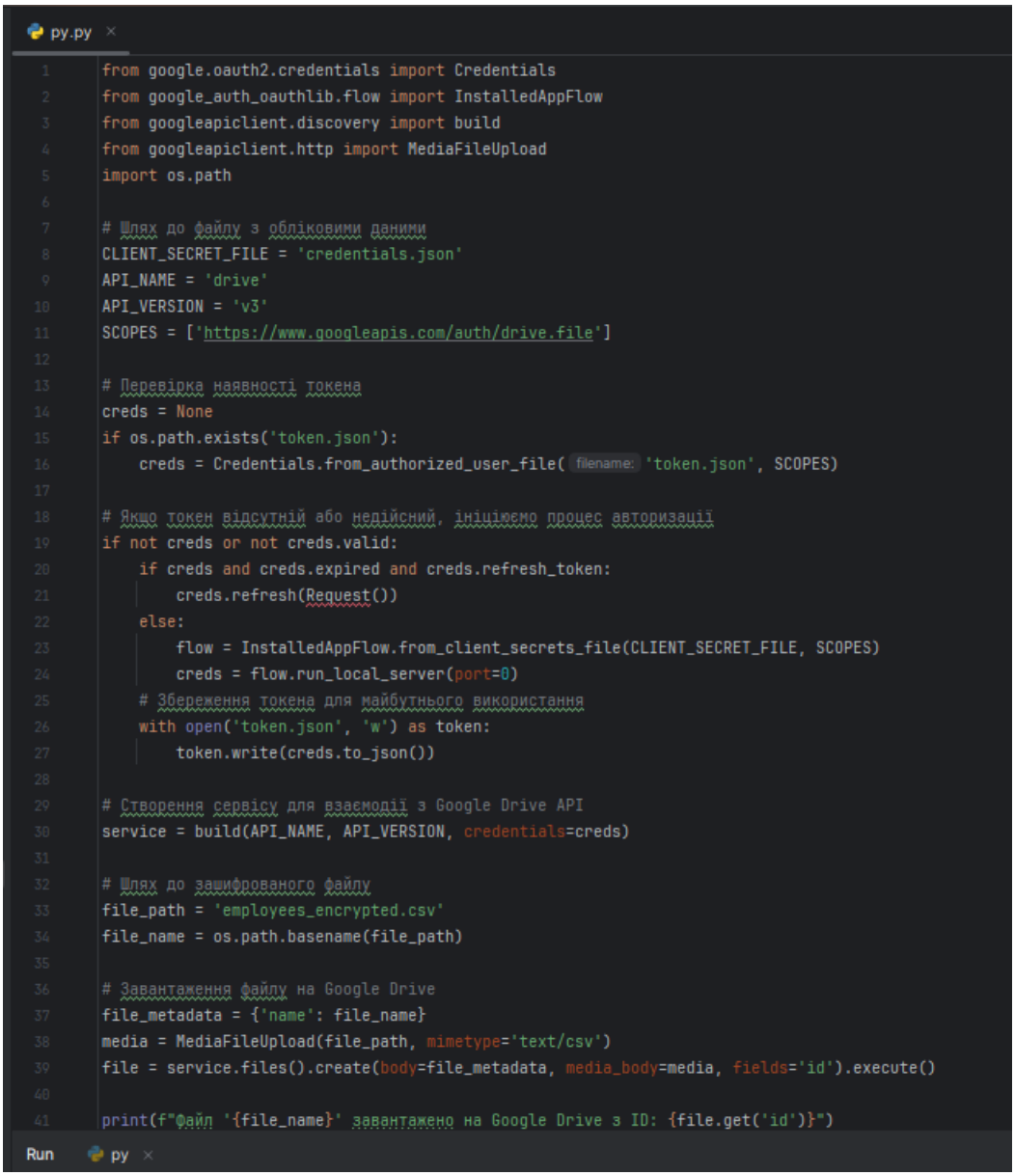


Рисунок 3.7 - код для перекидання зашифрованих даних на хмару

Код (рисунок 3.7) призначений для взаємодії з Google Drive API для завантаження файлів. Спочатку імпортуються необхідні бібліотеки для роботи з Google API та файловою системою. Далі встановлюються базові параметри: шлях до файлу з обліковими даними (credentials.json), назва та версія API ('drive' і 'v3'), а також області доступу (scopes) для роботи з файлами. Код перевіряє наявність збереженого токену доступу в файлі *token.json*. Якщо токен існує, він завантажується для використання. Якщо токен відсутній або недійсний,

запускається процес авторизації: якщо токен прострочений, але має *refresh token* - він оновлюється, інакше запускається повний процес авторизації через локальний веб-сервер. Після успішної авторизації токен зберігається в файл *token.json* для майбутнього використання. Потім створюється сервіс для роботи з Google Drive API використовуючи отримані облікові дані. Після всього, код готує метадані файлу (в даному випадку CSV файл), створює об'єкт MediaFileUpload для завантаження і виконує саме завантаження файлу на Google Drive. Після успішного завантаження виводиться повідомлення з ID завантаженого файлу (рисунок 3.8). Весь цей процес забезпечує безпечну авторизацію та завантаження файлів на Google Drive з можливістю повторного використання токену доступу (рисунок 3.9).

```
E:\Py\pythonProject\.venv\Scripts\python.exe E:\Py\pythonProject\py.py
Please visit this URL to authorize this application: https://accounts.google.com/o/oauth2/auth?response
Файл 'employees_encrypted.csv' завантажено на Google Drive з ID: 1KCVn__50OZR7aJ5yYFF6XZ65-WNqIIRn

Process finished with exit code 0
```

Рисунок 3.8 - показ усіпішності виконання коду з Рисунок 3.7

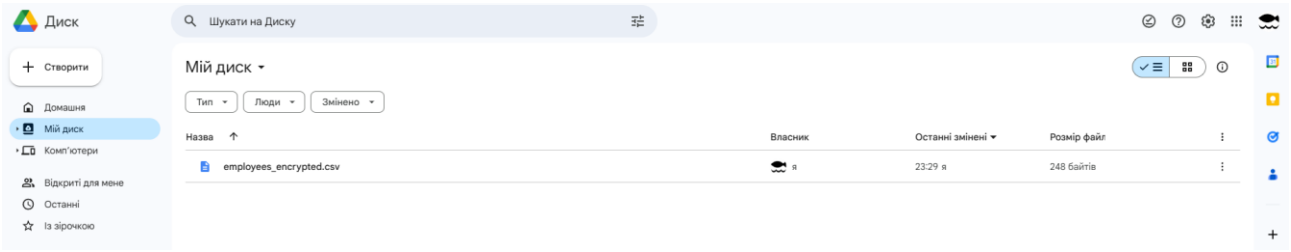


Рисунок 3.9 - запис даних Google Drive

Таким чином, зашифрований файл успішно завантажено на Google Drive, що забезпечує його безпечне зберігання та доступність з будь-якого пристрою.

3.4. Розшифрування даних та перевірка захисту

Після зберігання зашифрованого файлу у хмарному сервісі важливо переконатися, що дані можна безпечно розшифрувати та відновити до початкового стану. На прикладі(рисунок 3.10) розглянемо процес розшифрування даних за допомогою бібліотеки *cryptography.fernet* у Python.

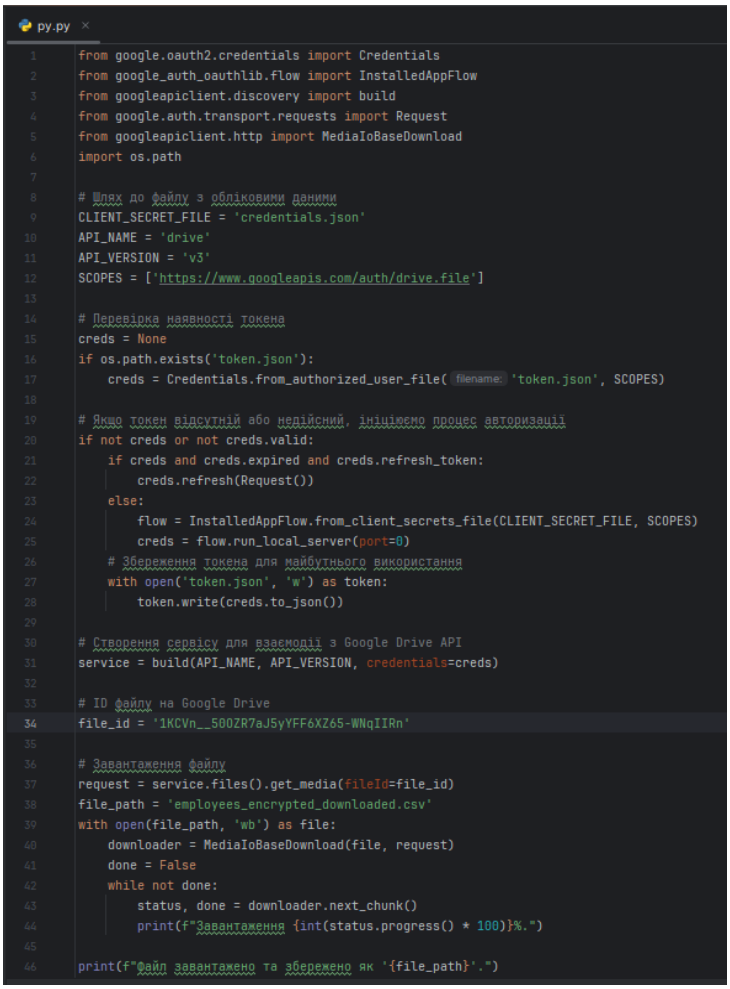


Рисунок 3.10 - завантаження зашифрованого файлу з хмарного сервісу

Код (рисунок 3.10) призначений для завантаження файлу з Google Drive на локальний комп'ютер. Спочатку відбувається імпорт необхідних бібліотек для роботи з Google API. Далі встановлюються основні параметри: шлях до файлу з обліковими даними (credentials.json), назва сервісу ('drive'), версія API (v3) та необхідні права доступу. Код перевіряє наявність збереженого токену в файлі

token.json - якщо він є, використовується цей токен. Якщо токен відсутній або недійсний, запускається процес авторизації: при наявності простроченого токену з можливістю оновлення - він оновлюється, інакше запускається повна авторизація через локальний сервер. Успішний токен зберігається в файл *token.json* для подальшого використання. Після авторизації створюється сервіс для роботи з Google Drive API. Далі код отримує медіа-дані файлу за його ID, створює локальний файл для збереження даних і запускає процес завантаження. Завантаження відбувається частинами (chunks), і прогрес виводиться у відсотках. Після завершення завантаження файл зберігається на комп'ютері під назвою '*employees_encrypted_downloaded.csv*', і виводиться повідомлення про успішне завершення операції (рисунок 3.11).

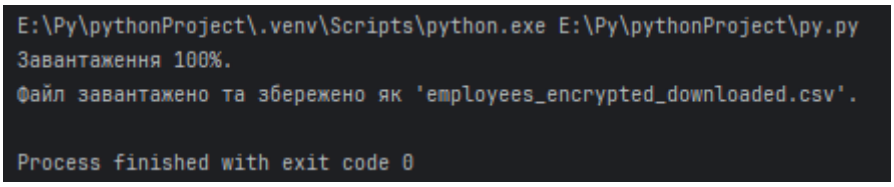


Рисунок 3.11 - вивід повідомлення про успішне завершення операції.

Після завантаження зашифрованого файлу на локальний комп'ютер можна приступити до його розшифрування.

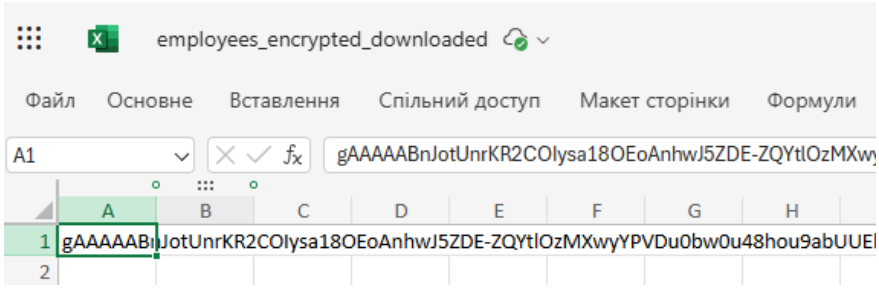


Рисунок 3.12 - зашифрований файл

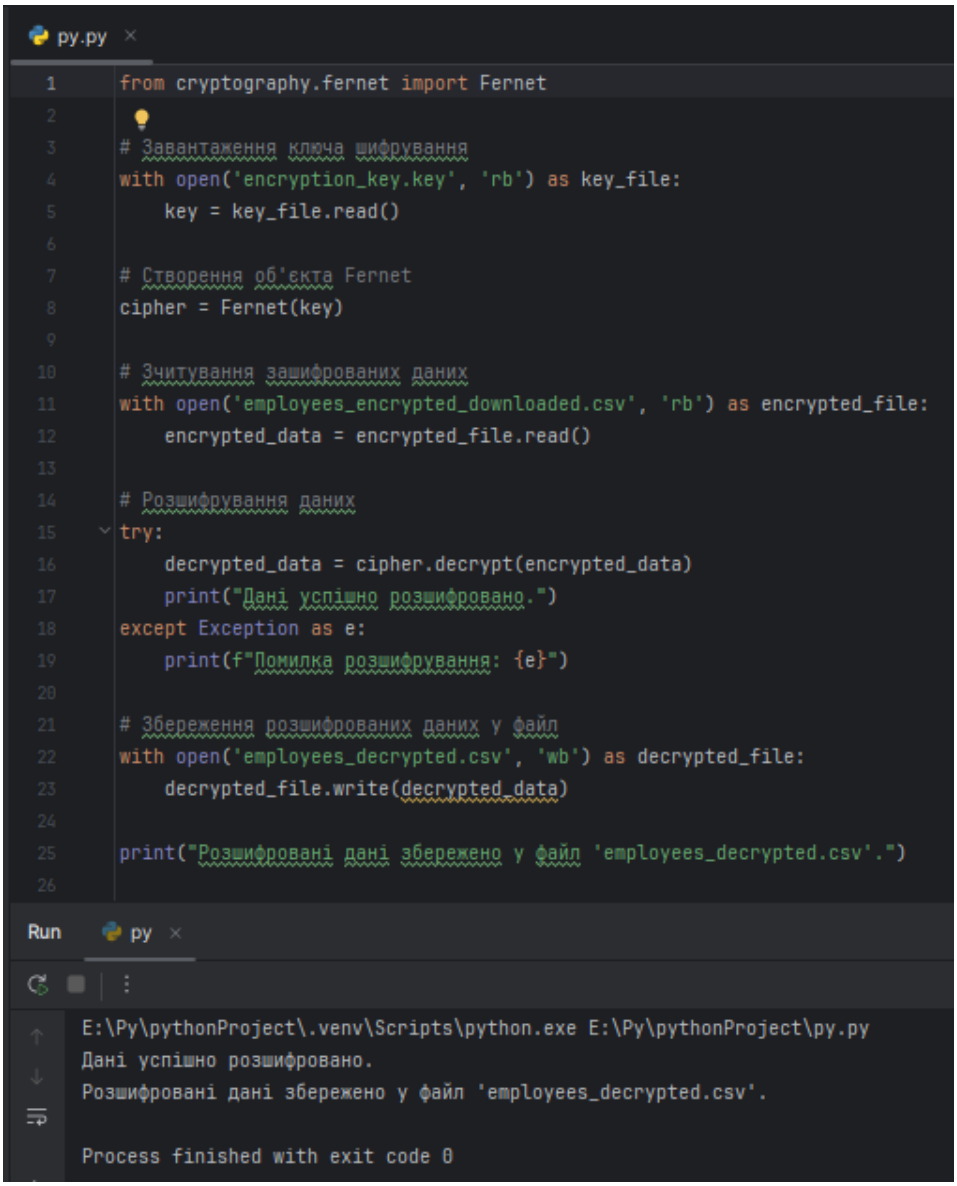


Рисунок 3.13 - код для розшифрування даних

Цей код виконує процес розшифрування завантаженого файлу за допомогою бібліотеки *cryptography.fernet*. Спочатку код зчитує ключ шифрування з файлу '*encryption_key.key*' у бінарному режимі. На основі цього ключа створюється об'єкт Fernet, який буде використовуватися для розшифрування. Далі код відкриває зашифрований файл '*employees_encrypted_downloaded.csv*' також у бінарному режимі та зчитує його вміст. Розшифрування даних відбувається за допомогою методу decrypt() об'єкта Fernet, при цьому весь процес обгорнутий у блок *try-except* для обробки можливих помилок розшифрування. Якщо розшифрування пройшло успішно,

розшифровані дані записуються у новий файл '*employees_decrypted.csv*' у бінарному режимі. Про кожен етап процесу виводяться інформаційні повідомлення, включаючи повідомлення про успішне розшифрування та збереження даних або про помилку, якщо вона виникла. Цей код є частиною процесу безпечної обробки даних, де важливо правильно зберігати та використовувати ключ шифрування для успішного відновлення оригінальних даних.

Після розшифрування даних важливо переконатися, що вони не були пошкоджені або змінені під час процесу шифрування та розшифрування. Для цього можна порівняти хеш-суми оригінального та розшифрованого файлів(рисунок 3.14).

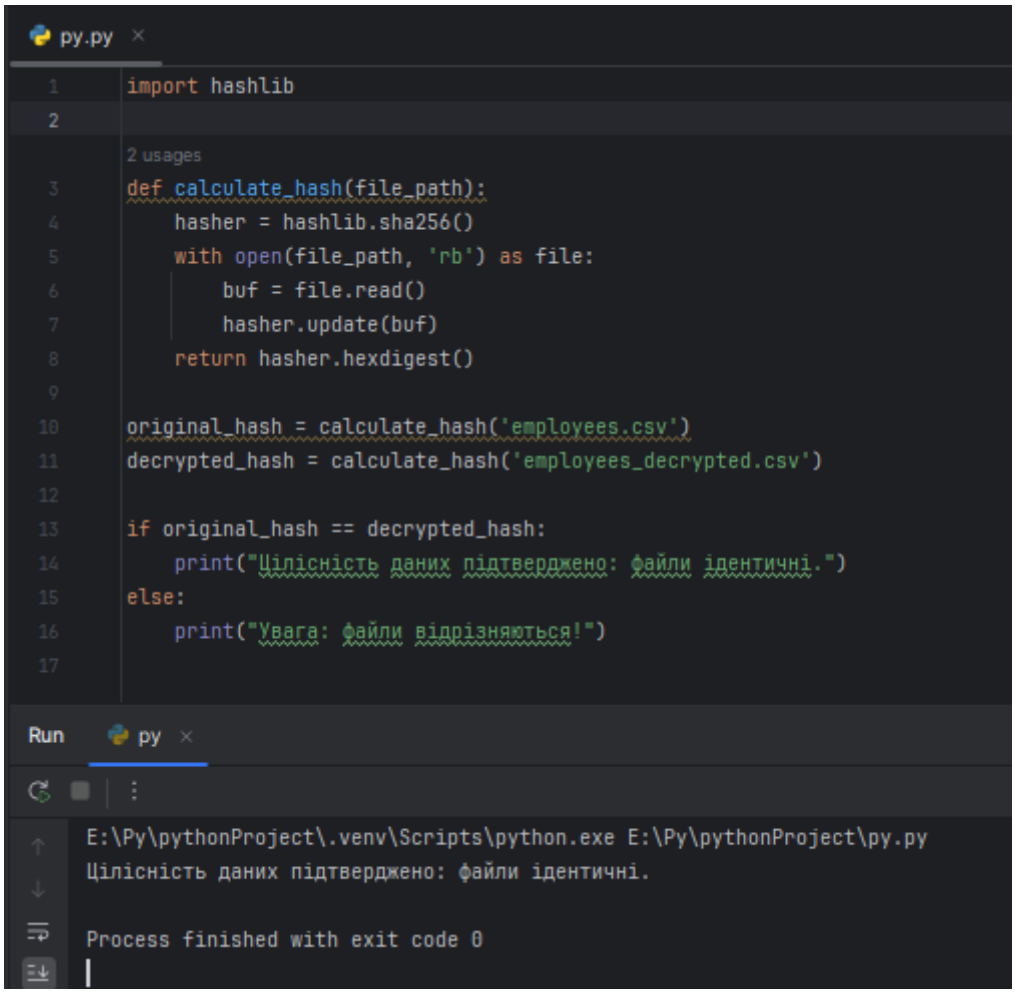


Рисунок 3.14 - код для перевірки цілісності даних

У цьому розділі продемонстровано процес розшифрування зашифрованих даних, завантажених з хмарного сервісу, та перевірили їх цілісність. Використання бібліотеки *cryptography.fernet* у Python забезпечує надійний механізм шифрування та розшифрування даних, а порівняння хеш-сум дозволяє переконатися у відсутності змін у даних під час їх обробки.

РОЗДІЛ 4. ОЦІНКА НАДІЙНОСТІ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1. Оцінка надійності шифрування.

У цьому проекті було використано симетричне шифрування за допомогою алгоритму Fernet, який є частиною бібліотеки cryptography у Python. Fernet забезпечує надійний рівень захисту завдяки використанню алгоритму шифрування AES-128 в режимі CBC (Cipher Block Chaining) з додаванням HMAC для автентифікації. Це означає, що навіть якщо зловмисник отримає доступ до зашифрованого файлу, він не зможе прочитати дані без наявності ключа, оскільки шифрований текст неможливо дешифрувати без цього ключа.

На тестових даних було продемонстровано, що Fernet забезпечує конфіденційність і цілісність інформації, оскільки будь-які зміни в зашифрованих даних призводять до помилки під час розшифрування. Додаткова перевірка цілісності через порівняння хешів підтвердила, що вихідний файл і розшифрований файл є ідентичними. Це дозволяє стверджувати, що обраний підхід забезпечує надійну ідентифікацію будь-яких змін у даних.

4.2. Переваги та обмеження розробленого інструментарію

Переваги обраного підходу:

1. Простота реалізації: Бібліотека cryptography дозволяє легко реалізувати шифрування даних без необхідності створювати складні алгоритми. Метод Fernet надає простий API для шифрування та розшифрування, що робить його зручним для новачків.
2. Надійний захист: Використання алгоритму AES-128 і HMAC забезпечує захист від широкого спектра атак. Навіть якщо дані зберігаються в хмарному середовищі, вони залишаються недоступними для сторонніх осіб, якщо тільки ключ не буде скомпрометований.
3. Мінімізація ризиків: Завдяки симетричному шифруванню весь процес захищений від змін в алгоритмі. Аудит цілісності даних через порівняння

хешів підтверджує, що дані залишаються незмінними після процесу шифрування та розшифрування.

Обмеження обраного підходу:

1. Необхідність управління ключем: Одним з головних обмежень симетричного шифрування є необхідність зберігати і захищати ключ. Якщо ключ буде втрачений або скомпрометований, доступ до зашифрованих даних буде втрачено, і весь захист стане недійсним. Це особливо важливо для хмарного зберігання, де ключ може бути доступний зловмисникам, якщо його не захистити належним чином.
2. Обмеження розміру ключа: Fernet використовує ключ фіксованого розміру (AES-128), що може бути обмеженням для деяких проектів. Хоча AES-128 забезпечує високий рівень безпеки, для деяких критичних застосунків рекомендують більш потужні алгоритми з більшими розмірами ключів, наприклад AES-256.
3. Продуктивність при великих обсягах даних: Симетричне шифрування може бути ресурсомістким, особливо при шифруванні великих обсягів даних. У реальних застосунках, які обробляють значні масиви даних, використання такого шифрування може вимагати додаткових ресурсів.

4.3. Напрями покращення захисту даних у майбутніх дослідженнях

Для підвищення рівня захисту даних у майбутніх дослідженнях можна розглянути кілька напрямків покращення:

1. Перехід на асиметричне шифрування для обміну ключами: У цій курсовій роботі використовувалося симетричне шифрування, яке вимагає передачі ключа між сторонами. Використання асиметричного шифрування для генерації та обміну ключами перед шифруванням даних знижує ризик компрометації ключа, оскільки обидві сторони будуть мати різні ключі.

2. Використання двофакторної автентифікації (2FA): Для додаткового захисту доступу до ключа шифрування та файлів можна інтегрувати двофакторну автентифікацію. Це забезпечить додатковий рівень безпеки, навіть якщо ключ буде скомпрометований, оскільки зловмисник також потребуватиме доступу до другого фактора.
3. Застосування шифрування з великим розміром ключа (AES-256): Використання алгоритму AES-256 замість AES-128 забезпечить вищий рівень захисту для критичних даних. Це особливо важливо для застосунків з підвищеними вимогами до безпеки, де можливий більш тривалий час для злому 256-бітного ключа.
4. Додаткова перевірка цілісності даних: Для гарантування, що дані не були змінені під час шифрування або транспортування, можна використовувати контрольні суми або хеші на рівні хмарного зберігання. Це дозволить автоматично перевіряти цілісність кожного файлу при його передачі в хмару.
5. Сегментація та шифрування великих файлів блоками: Для оптимізації продуктивності при шифруванні великих файлів можна розділити файли на менші блоки та зашифрувати кожен блок окремо. Це дозволить знизити навантаження на систему під час шифрування великих обсягів даних та підвищити ефективність зберігання у хмарі.

ВИСНОВКИ

Застосування шифрування для захисту даних у хмарному середовищі показало, що симетричне шифрування є ефективним способом зберегти конфіденційність інформації. Використання методу Fernet з бібліотеки cryptography у Python забезпечило належний рівень захисту, перетворюючи дані у зашифрований формат, який неможливо прочитати без спеціального ключа. Це гарантує, що навіть якщо файл потрапить до сторонніх осіб, його вміст залишиться недоступним.

Процедура розшифрування та перевірка цілісності даних підтвердили надійність обраного підходу, забезпечивши збереження вихідних даних без змін після шифрування та зберігання. Порівняння хешів розшифрованого та оригінального файлів підтвердило їх повну ідентичність, що свідчить про стабільність і точність методу.

Основними перевагами цього підходу є його простота в реалізації, високий рівень захисту та легка інтеграція з хмарними сервісами, такими як Google Drive. Проте метод також має певні недоліки. Зокрема, симетричне шифрування вимагає обережного зберігання ключа, адже його компрометація може призвести до втрати доступу до зашифрованих даних. Крім того, робота з великими обсягами інформації може впливати на продуктивність, оскільки шифрування значних даних є ресурсомістким процесом.

Для покращення рівня безпеки в майбутньому можна розглянути додаткові заходи, як-от використання асиметричного шифрування для безпечного обміну ключами, двофакторну автентифікацію для захисту ключа, збільшення розміру ключа шифрування та додаткові методи перевірки цілісності даних. Такі вдосконалення допоможуть підвищити захист даних і зробити цей підхід ще більш надійним у контексті зберігання конфіденційної інформації в хмарних сховищах.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google Workspace. (2024). Хмарні сервіси безпеки й захисту даних. [Електронний ресурс] - https://workspace.google.com/intl/uk/security/
2. Mell, P., & Grance, T. (2011). *The NIST Definition of Cloud Computing*. National Institute of Standards and Technology. [Електронний ресурс] - http://surl.li/fguzyb
3. Імена.UA. (2020). Історія розвитку хмарних технологій. [Електронний ресурс] – https://www.imena.ua/blog/cloud-computing-history/
4. Осьмак Д.П., Дмітрієв В.Є., Свертока В.В ЗАБЕЗПЕЧЕННЯ ЗАХИЩЕНОГО ДОСТУПУ ДО ХМАРНИХ СЕРВІСІВ НА БАЗІ РІШЕННЯ MICROSOFT CLOUD APP SECURITY. – 2021. – №4(48). – С. 1-7.
5. DQS Україна. (2024). Сертифікація за стандартом ISO/IEC 27017. [Електронний ресурс] – https://www.dqsglobal.com/uk-ua/sertifikujte/iso-27017
6. Knowledge Share. (2024). ToothPic: Secure Passwordless Authentication. [Електронний ресурс] – http://surl.li/ayhsqe
7. Auth0. (2024). Role-Based Access Control (RBAC). [Електронний ресурс] – https://auth0.com/docs/manage-users/access-control/rbac
8. Cloudflare. (2024). What is End-to-End Encryption? [Електронний ресурс] – https://www.cloudflare.com/learning/privacy/what-is-end-to-end-encryption/
9. Computer Weekly. (2024). A History of Cloud Computing. [Електронний ресурс] – https://www.computerweekly.com/feature/A-history-of-cloud-computing
10. Microsoft. (2024). Що таке безпека в хмарі? [Електронний ресурс] – https://www.microsoft.com/uk-ua/security/business/security-101/what-is-cloud-security

11. Fullstack Foundations. (2024). Client Side vs. Server Side in Web Development: A Beginner's Guide. [Електронний ресурс] – https://www.fullstackfoundations.com/blog/client-side-vs-server-side
12. Google Sites. (2024). Криптографія з відкритим ключем, різновидності алгоритмів. [Електронний ресурс] – https://sites.google.com/view/blog-ua/основні-поняття-криптографії-та-захисту-інформації/криптографія-з-відкритим-ключем-різновидності-алгоритм
13. Cryptography.io. (2024). Fernet (symmetric encryption). [Електронний ресурс] – https://cryptography.io/en/latest/fernet/#
14. Tutorials Point. (2023). Fernet Symmetric Encryption using a Cryptography Module in Python. [Електронний ресурс] – https://www.tutorialspoint.com/fernet-symmetric-encryption-using-a-cryptography-module-in-python

ДОДАТКИ

Додаток А

Авторизація та завантаження файлу на Google Drive

```
from google.oauth2.credentials import Credentials

from google_auth_oauthlib.flow import InstalledAppFlow

from googleapiclient.discovery import build

from googleapiclient.http import MediaFileUpload

import os.path


CLIENT_SECRET_FILE = 'credentials.json'

API_NAME = 'drive'

API_VERSION = 'v3'

SCOPES = ['https://www.googleapis.com/auth/drive.file']


creds = None

    if os.path.exists('token.json'):

        reds = Credentials.from_authorized_user_file('token.json', SCOPES)


    if not creds or not creds.valid:
```

```
            if creds and creds.expired and creds.refresh_token:
                creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file
                (CLIENT_SECRET_FILE, SCOPES)
            creds = flow.run_local_server(port=0)
        with open('token.json', 'w') as token:
            token.write(creds.to_json())

service = build(API_NAME, API_VERSION, credentials=creds)

file_path = 'employees_encrypted.csv'
file_name = os.path.basename(file_path)

file_metadata = {'name': file_name}
media = MediaFileUpload(file_path, mimetype='text/csv')
file = service.files().create(body=file_metadata,
        media_body=media, fields='id').execute()

print(f"Файл '{file_name}' завантажено на Google Drive з ID: {file.get('id')}")
```

Додаток Б

Код для розшифрування даних

```
from cryptography.fernet import Fernet

with open('encryption_key.key', 'rb') as key_file:
    key = key_file.read()

cipher = Fernet(key)

with open('employees_encrypted_downloaded.csv', 'rb') as encrypted_file:
    encrypted_data = encrypted_file.read()

try:
    decrypted_data = cipher.decrypt(encrypted_data)
    print("Дані успішно розшифровано.")
except Exception as e:
    print(f"Помилка розшифрування: {e}")

with open('employees_decrypted.csv', 'wb') as decrypted_file:
    decrypted_file.write(decrypted_data)
```

```
print("Розшифровані дані збережено у файл 'employees_decrypted.csv'.")
```