

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ МЕТОДІВ
КРИПТОГРАФІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ.
RESEARCH AND IMPLEMENTATION OF METHODS FOR
CRYPTOGRAPHIC PROTECTION OF INFORMATION

Виконав(ла): студент(ка) 2 курсу, групи КНТ-122м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

МАСЛОВ М.Г.

(ПРИЗВИЩЕ та ініціали)

Керівник

ЗАЙКО Т.А

(ПРИЗВИЩЕ та ініціали)

Рецензент

КАЗУРОВА А.Є

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

МАСЛОВА Максима Геннадійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та реалізація методів криптографічного захисту інформації.

Research and Implementation of Methods for Cryptographic Protection of Information

керівник проєкту (роботи) к.т.н., доцент, ЗАЙКО Тетяна Анатоліївна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “14” листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року

3. Вихідні дані до проєкту (роботи) ОС Windows 10 та Ubuntu server,

Криптографічні методи шифрування та криптографічні алгоритми,

Мережові протокол SSH та TSP/IP, Криптографічний протоколи SSL/TLS.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Теоретичні аспекти криптографічного шифрування інформації.

2. Методологічні основи та способи шифрування інформації. 3. Розробка архітектури програми та налаштування систем криптографії. 4. Рішення при розробці криптографічного захисту сайту, експлуатація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЗАЙКО Т.А., доцент		
Нормоконтроль	ЛИПОВЕЦЬ М.В., асистент		

7. Дата видачі завдання “08” вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Аналіз, розробка й вдосконалення алгоритмів, методів та модулів криптографічного шифрування даних.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми, тестування та експериментальне дослідження програмного забезпечення.	7-9 тижні	Розділ 4
6	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
7	Нормоконтроль та рецензування.	12 тиждень	
8	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Максим МАСЛОВ
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна ЗАЙКО
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
138 с., 7 табл., 55 рис., 4 дод., 38 джерел.

КРИПТОГРАФІЧНИЙ ЗАХИСТ ДАНИХ, ЗАХИСТ ДАНИХ, КОМП'ЮТЕРНА ПРОГРАМА, СЕРВЕР, ЗАХИСТ ДАНИХ САЙТУ, ВЕБСЕРВЕР, КРИПТОГРАФІЯ, ШИФРУВАННЯ, ВИКОРИСТАННЯ КРИПТОГРАФІЇ У СУЧАСНИХ САЙТАХ.

Об'єкт дослідження – реалізація систем криптографічного захисту та методів авторизації на комерційному вебсайті в сучасних умовах.

Предмет дослідження – криптографічне шифрування та розшифровування даних, впровадження інноваційного методу авторизації на комерційному вебсайті за допомогою одноразових паролів (ОТР) в сучасних умовах використання інтернет-ресурсів.

Мета роботи – дослідження та реалізація на практиці методів захисту даних та кондиційної інформації, в сучасних умовах, за допомогою криптографічного шифрування інформації.

Матеріали, методи та технічні засоби: мови програмування JavaScript, Python, середовище розробки та редагування програмного забезпечення Microsoft Visual Studio 2016 та середовище розробки PyCharm community Edition, методи структурного та об'єктно-орієнтоване програмування, а також програмне забезпечення багатофункціональний графічний редактор Adobe Photoshop та графічний редактор Affinity Designer. Персональний комп'ютер, що функціонує на базі операційної системи Microsoft Windows 10 та з додатковим окремим використанням Ubuntu Server для розгортання та базування сайту на цій платформі. З використанням таких технічних комплектуючих персонального комп'ютера як, процесором Intel Core i3 10100, оперативною пам'яттю DDR4 з об'ємом 16 Гігабайтів, записним

пристроєм тип SSD NVMe та периферійними пристроями: монітор з діагоналлю 19'', клавіатурою, координаційний пристрій типу «миша», вебкамерою та гарнітуром.

Результат: Було створено вебсайт, керований операційною системою Ubuntu Server, який дозволяє безпечно робити замовлення товарів та передавати ці дані між сервером та клієнтським веббраузером.

Висновки. Розроблено систему, яка за допомогою криптографії дозволяє забезпечити необхідний рівень захисту даних та інформації у відповідності до сучасних вимог захисту інформації, а також створено робочий комерційний сайт для розширення діяльності фірми на рівень електронної комерції.

Галузь використання – інтернет-магазини, зорієнтовані на забезпечення високого рівня безпеки та оптимізації функціоналу для ефективної роботи в галузі електронної комерції (E-commerce).

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 138 pages, 7 tables, 55 figures, 4 appendixes, 38 sources.

CRYPTOGRAPHIC DATA PROTECTION, DATA PROTECTION, COMPUTER PROGRAM, SERVER, WEBSITE DATA PROTECTION, WEB SERVER, CRYPTOGRAPHY, ENCRYPTION, USE OF CRYPTOGRAPHY IN MODERN WEBSITES.

The object of research is the implementation of cryptographic protection systems and authorization methods on a commercial website in modern conditions.

The subject of the research is cryptographic encryption and decryption of data, implementation of an innovative method of authorization on a commercial website using one-time passwords (OTP) in the current conditions of using Internet resources.

Purpose of the work: Research and implementation in practice the methods of data protection and conditioned information, in modern conditions, using cryptographic encryption of information

Materials, methods and tools: JavaScript, Python programming languages, Microsoft Visual Studio 2016 software development and editing environment and PyCharm community Edition development environment, structural and object-oriented programming methods, as well as Adobe Photoshop multifunctional graphic editor and Affinity Designer graphic editor. A personal computer running on the Microsoft Windows 10 operating system with the additional separate use of Ubuntu Server for deploying and basing the website on this platform. Using such technical components of a personal computer as an Intel Core i3 10100 processor, 16 Gigabytes of DDR4 RAM, an NVMe SSD storage device and peripherals: a 19" monitor, a keyboard, a mouse, a webcam and a headset.

Result: A website running on the Ubuntu Server operating system was created that allows you to securely place product orders and transfer this data between the server and the client web browser.

Conclusions. A system has been developed that, using cryptography, allows to ensure the necessary level of data and information protection in accordance with modern information security requirements, and a working commercial website has been created to expand the company's activities to the level of e-commerce.

The field of application is online stores focused on ensuring a high level of security and optimizing functionality for effective work in the field of electronic commerce (E-commerce).

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	10
Вступ.....	11
1 Теоретичні аспекти криптографічного шифрування інформації	12
1.1 Сутність та властивості криптології та криптографічного шифрування інформації	12
1.2 Основні етапи розвитку криптографії	19
1.3 Розвиток історичних способів шифрування інформації.....	23
1.4 Класифікація криптографічний шифрів	34
Висновки за розділом 1	37
2 Методологічні основи та способи шифрування інформації.....	38
2.1 Проблематика розвитку сучасної криптографії.....	38
2.2 Основні методи та вимоги до захисту інформації.....	40
2.3 Основні напрямки та способи криптографічного шифрування даних в сучасному світі	48
2.4 Види алгоритмів та сучасні напрямки криптографічного шифрування даних.....	51
Висновки за розділом 2	56
3 Розробка архітектури програми та налаштування систем криптографії.....	57
3.1 Стратегія та методи вибору розташування сервера	57
3.2 SSH як спосіб автентифікації.....	63
3.3 Вебсервер nginx та його роль у захисті сервера	64
3.4 Мова програмування для вебсайту	70
3.5 Опис та методи вибору WSGI-сервера	73
Висновки за розділом 3	76

4 Рішення при розробці криптографічного захисту сайту, експлуатація та тестування	77
4.1 Налаштування ОС сервера та загальної архітектури	77
4.1.1 Налаштування Firewall UFW	77
4.1.2 Завантаження сертифікатів SSL	78
4.2 Конфігурація SSH та розбір етапів роботи протоколу.....	79
4.3 Засоби захисту через вебсервер Nginx.....	84
4.4 Фреймворк Django – структура проєкту та методи захисту аутентифікації.....	89
4.4.1 Огляд функціоналу вебсайта	90
4.4.2 Розробка та впровадження метода двоетапної аутентифікації OTP	94
4.4.3 Структура PostgreSQL та захист від SQL-ін'єкцій.....	99
4.4.4 Сервер-додатків Gunicorn	102
4.5 Тестування систем безпеки вебсайту.....	104
Висновки за розділом 4	107
Висновки	109
Перелік джерел посилань	110
Додаток А Налаштування NGINX.....	114
Додаток Б Текст програми	117
Додаток В Слайди презентації.....	130

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

2FA	– Two-Factor Authentication;
AES	– Advanced Encryption Standard;
DES	– Data Encryption Standard;
DDoS	– Distributed Denial-of-Service;
DNS	– Domain Name System;
DoS	– Denial-of-Service;
ECC	– Elliptic Curve Cryptography;
FDE	– Full Disk Encryption;
GUI	– Graphical User Interface;
IPsec	– IP Security;
MVT	– Model-View-Template;
NIST	– The National Institute of Standards and Technology;
OCSP	– Online Certificate Status Protocol;
OTP	– One Time Password;
PFS	– Perfect Forward Secrecy;
RSA	– R – Rivest, S – Shamir, A – Adieman;
SHA	– Secure Hash Algorithm;
SNI	– Server Name Indication;
SSH	– Secure Shell;
SSL	– Secure Sockets Layer;
TLS	– Transport Layer Security;
TDEA	– Triple Data Encryption Algorithm;
UFW	– Uncomplicated Firewall;
ЕЦП	– електронний цифровий підпис;
ОС	– операційна Система.

ВСТУП

Актуальність обраної теми обумовлена поточною політичною світовою нестабільністю та вступом світу у еру простистояння економік та технологій захист інформації, як ніколи має першочерговий пріоритет й той хто буде робити це краще вийде з цієї ери переможцем. Щодо нашої країни, то для нас дослідження цієї теми є однією з найпріоритетніших так як захист свого інформаційного поля та даних для нас є одним напрямків, які будуть вирішувати нашу подальшу долю.

Разом з цим до актуальності подальшого дослідження теми також можна додати те, що методи та способи шифрування даних, якими зараз користується світ, були винайдені та розроблені здебільшого починаючи з вісімдесятих років двадцятого століття і закінчуючи нуловими роками двадцятьпершого. З того часу вони лише незначно вдосконалювалися, проте технічний прогрес з того часу дуже сильно зріс, а разом з ним і можливості техніки. Тому на сьогодні вже провідні країни світу замислюються над винайденням нових способів та методів захисту своїх даних, що ще раз підтверджує актуальність та правильність подальшого розвитку, дослідження та вдосконалення обраної теми.

Тому у дипломній роботі розглядаються теоритичні аспекти та історія розвитку та вдосконалення криптографічного шифрування інформації, оскільки тальки досить глибоке дослідження теоретичного матеріалу дасть можливість ефективно використовувати наявні інструменти захисту інформації. Разом з цим без дослідження історії розвитку, розробити та впровадити дієву систему захисту буде не можливо.

Разом з дослідженням методів та способів захисту інформації шляхом криптографічного шифрування інформації у даній роботі проводиться розробка та опис додатка, який дасть змогу безпечно обмінюватись кондиційною інформацією, а разом з цим розглядається питання та способи керування ключами криптосистем, які є невдумливою їх частиною.

1 ТЕОРЕТИЧНІ АСПЕКТИ КРИПТОГРАФІЧНОГО ШИФРУВАННЯ ІНФОРМАЦІЇ

1.1 Сутність та властивості криптології та криптографічного шифрування інформації

Застосування комп'ютерної техніки та електронних засобів передачі інформації, хоча і є потужним, проте не позбавлене ризиків втрати або незаконного розголошення конфіденційної інформації. Це ставить під сумнів ефективність і безпеку електронних засобів передачі даних. Зокрема, існує необхідність вдосконалення методів, алгоритмів та систем, що гарантуватимуть безпечність та швидкість передавання інформації, мінімізуючи ризики її втрати чи неправомірного доступу.

Для більшої обізнаності та поглиблення до теми роботи розбиремо наступні теоретичні аспекти такі як: криптологія, криптографія, криптоаналіз, шифрування та розшифрування інформації, а також що з себе представляє криптосистема та деякі інші терміни, які знадобляться у процесі дослідження обраної теми.

В загальному розумінні сучасна криптологія являє собою змагання між різними методами шифрування інформації, тобто криптографією та її способами її розшифровки без використання спеціально призначеного для цього ключа, тобто криптоаналізом. У більшості випадків дослідники розглядають криптологію саме як цю боротьбу між криптографією та криптоаналізом.

Початковим етапом є визначення та розгляд основ криптографії. Розглянемо сутність цього поняття. У сучасній науковій галузі, криптографія та криптоаналіз є складовою частиною дисципліни, відомої як криптологія. Тому наступним кроком для розкриття теми буде логічно більш детально розібратись з цими термінами.

Визначення терміну «криптографія», запропоновані різними науковцями, наведено в табл. 1.1.

Таблиця 1.1 – Основні визначення терміну криптографія

Автор	Визначення терміну
Великий тлумачний словник сучасної української мови	Криптографія (в перекладі з грецької, це тайнопис) – у сучасному розумінні цей термін можна визначити як галузь науки, що досліджує математичні методи, способи та технології забезпечення конфіденційності інформації, цілісності, а також її автентичності. Потреба у її вивченні була сформована практичною потребою у віднайденні способів та методів передачі важливих та конференційній даних більш надійними способами [1].
Американський Національний інститут стандартів та технологій (NIST)	Криптографія – це наука, яка ставить собі на меті дослідження та вдосконалення принципів, методів, а також засобів, за допомогою яких можна перетворювати інформацію в інший вигляд для приховання її змісту, таким чином унеможливаючи несанкційований доступ до неї чи її підробку [2].
Горбенко І. Д. (український науковець)	Криптографія – це напрям криптології який займається дослідженням методів, противоріч та основних закономірностей за допомогою яким можна забезпечити конфіденційність, цілісність та доступності інформації за допомогою криптографічних перетворень [3].
Брюс Шнайєр (американський криптограф та вчений)	Визначає криптографію, як науку та мистецтво, яке дає змогу забезпечити секретність даних та захищеність інформації [4].

З розглянутих визначень, які були сформовані вченими та криптографами різних країн, можна побачити, що хоч кожний і бачить цей термін по-своєму, але серед них усіх є певні сходження. Спираючись на усе вищезазначено, можна зробити визначення, що криптографія являє собою змагання між різними методами шифрування інформації та способами її розшифрування.

З іншої сторони криптографія використовує математичні дисципліни, такі як абстрактна алгебра та теорія ймовірностей, у своїй діяльності. Ці області є необхідними інструментами для процесу шифрування інформації, оскільки вони теоретично дозволяють оцінити надійність шифру та передбачити складність його криптоаналізу. Використання математичних методів у цих областях науки сприяє утворенню шифрів та даних, які є важкорозкриваними та більш стійкими до криптоаналізу. Це, в свою чергу, підвищує рівень надійності та впевненість у захищеності інформації.

Наступним кроком, після надання визначення терміну криптографія, є логічно визначити поняття криптоаналіз як протилежність криптографії. Сам термін криптоаналіз був введений громадянином США молдавського походження, криптографом Вільямом Фрідманом у 1920 році. Однак більш доцільно цей термін було пояснено ще одним американським науковцем, Брюсом Шнайєром, який розглядає цей термін як мистецтво розшифрування шифротексту [4].

Проте дане визначення є загальним і було сформульоване більше 40 років тому. Для більш детального розкриття цього поняття приймемо за основу наступне визначення криптоаналізу.

Криптоаналіз – область криптології, що ставить своєю метою протидію криптографії шляхом виявлення методів та способів розшифрування зашифрованих даних без використання відповідного ключа. Це включає в себе пошук методів генерації необхідного ключа для розшифрування даних. Криптоаналіз спрямований на виявлення вразливостей у криптосистемі для подальших атак, що в результаті надає

доступ до зашифрованої інформації. У відомих термінах, криптоаналіз є синонімом терміну «дешифрування».

Після дослідження термінів криптографія та криптоаналіз, наступним кроком розглянемо термін, який об'єднує ці два поняття у загальну науку - криптологія. Оскільки цей термін має декілька широких визначень, розглянемо декілька із них, представлених у табл. 1.2.

Таблиця 1.2 – Визначення терміну криптологія

Автор	Визначення
О.Г. Корченко, В.П. Сіденко, Ю.О. Дрейс	Криптологія – це наука, яка займається дослідженням тайнописом (тобто криптографією), а також методами розкриття цього тайнописа (тобто криптоаналізом) [5]
В.М. Богуш, В.А. Мухачов	Криптологія (грецьк. «таємний» та «слово, вчення») в загальному значенні це - наука, що вивчає способи побудови, з наступним їх аналізом, збудованих систем та методів захисту даних, в основному шляхом шифрування інформації, що спирається у цьому процесі на математичні галузі науки при використанні «таємних» параметрів [6]
О.М. Гапак	Криптологія в сучасному розумінні – це змагання між методами шифрування та способа взому цього шифру [7]
В.В. Гребенніков	Криптологія – це математична наука, призначенням якої є шифрування та розшифровка інформації [8]

Закладенням фундаменту цієї науки заведено вважати роботу «Теорія зв'язку в секретних системах», яка була опублікована у 1949 р., американським вченим Клодом Шенноном, в якій фактично вперше було

зібрано, сформовано та представлено научній спільноті математичну модель шифрів.

У наступному етапі дослідження даної теми, обґрунтованим є перехід до дослідження більш конкретних термінів, таких як: зашифрування, розшифрування, криптосистема, криптостійкість, криптографічний алгоритм та криптографічний ключ.

Суть криптографічного алгоритму полягає у наборі математичних правил та процедур, якими можна описати такі види перетворення інформації та даних, як формування електронного цифрового підпису, а також перевірка такого підпису на достовірність, обчислення хеш-значень, шифрування та формування ключів для розшифровки, а також обчислення спеціальних контрольних, так званих «криптосум» тощо. У поєднанні та за допомогою декількох таких криптографічних алгоритмів, які використовуються для шифрування інформації, заведено називати – шифром.

Наряду з криптографічним алгоритмом використовується таке поняття, як «ключ», що по своїй суті який є секретним набором значень, які у поєднанні з криптографічними алгоритмами дають знати системі, що доступ до інформації запрошує особа, яка так чи інакше повинна мати до неї доступ оскільки володіє усім необхідним для цього.

Дослідивши основні концепції дослідження, важливим залишається розгляд терміну «шифрування», який був узгоджений вище.

У загальному розумінні, шифрування інформації – це процес захисту інформації, що відбувається у спеціальній криптосистемі, шляхом трансформації вхідної інформації в криптограму, непрості символи та знаки, нерозбірливі для будь-кого, хто не має відповідного ключа для розшифрування. Ступінь захисту зашифрованої інформації визначається рівнем її криптостійкості.

Для більш глибокого розуміння теми, доцільним буде розглянути наведене вище визначення терміну шифрування. Першочерговою задачею буде розгляд основних компонентів шифрування, зосереджуючись на

процесах «зашифрування» інформації та «розшифрування» її зашифрованого вигляду.

Зашифрування інформації – це процес перетворення доступної усім зрозумілої інформації та даних (інформації типу відкритого тексту), до стану незрозумілого набору символів, літер, цифр або в деяких випадках навіть малюнків та створення відповідного «ключа» за допомогою якого потім буде можливість отримати можливість доступу до цієї інформації [9].

У ролі ілюстрації, як способу шифрування, можна привести QR-код, що відомий своєю здатністю генерувати машинозчитувальну інформацію та перетворювати її на послідовність нерозпізнаваних малюнків, точок та проміжних символів. Важливо відзначити, що QR-код використовується лише як наочний приклад терміну шифрування і не має претензій на захищеність даних, оскільки його розшифрування можливе з будь-якого сучасного смартфона.

Після шифрована інформація, шифр може бути переданий на інший пристрій, і в процесі передачі існує ризик перехоплення даних. Однак зашифрований текст не дозволить несанкціонованому користувачеві отримати доступ до інформації користувача. Замість цього він отримає лише незрозумілу послідовність символів. Наступним етапом буде визначення терміну «розшифрування».

Розшифрування представляє собою процес перетворення зашифрованого тексту на його вихідний вигляд або відкритий текст для подальшого використання. Цей процес здійснюється за допомогою визначених логічних або математичних правил, які в більш широкому контексті відомі як «ключ». Далі слід розглянути та проаналізувати визначення термінів «криптосистема» та «криптостійкість» [9].

Криптосистема – це система криптографічних перетворень вхідної інформації, яка в загальному вигляді складається з п'яти складових: множин відкритого вхідного тексту, сімейства зашифровуючих та розшифровуючих перетворень даних (або як їх ще називають - криптоалгоритмів), складових

шифротекстів та складових ключів. Спеціаліста, що здійснює розробку подібних криптосистем, ідентифікують як криптографа [10].

Кожна така система характеризується своєю криптостійкістю, тобто можливостями до протидії направлених на неї атак для отримання доступу до зашифрованих даних. Якщо ж криптостійкості недостатньо для протидії атаці, то таку успішну атаку заведено називати – взломом.

Здійснивши аналіз вище згаданих аспектів та враховуючи різні визначення та підходи вчених, для кращого розуміння можна узагальнити всі обговорені терміни у єдину систему, наведену на рис. 1.1.

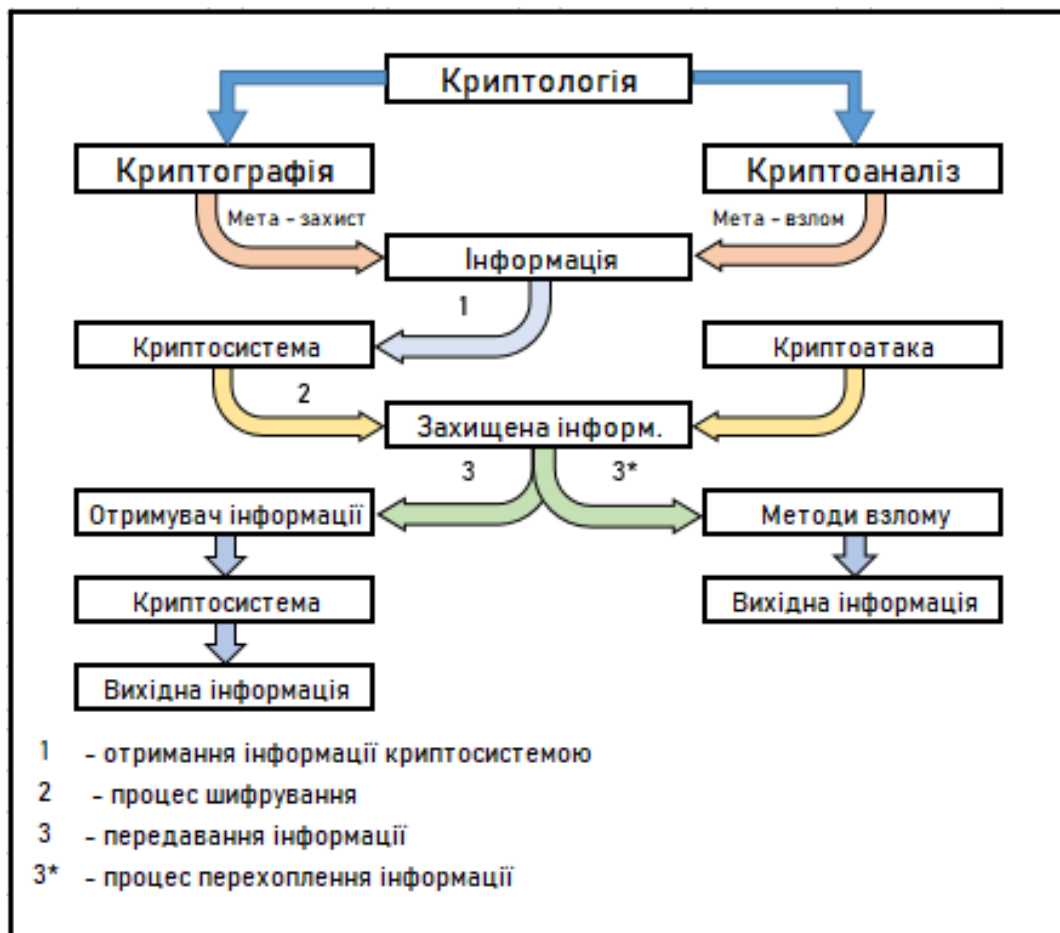


Рисунок 1.1 – Схема термінології криптології

На поданій вище схемі видно, що вдалося узагальнити основні концепції та принципи даної науки, розглядаючи етапи її еволюції. З перших примітивних методів перехоплення листів до складних математичних

алгоритмів, що лягли в основу сучасних систем шифрування, криптографія перетворилася в ключовий інструмент для захисту конфіденційності та безпеки інформації.

1.2 Основні етапи розвитку криптографії

Досліджуючи історію даної науки, можна визначити її витoki ще з періоду виникнення писемності. Перші текстові матеріали, створені людством, можна вважати криптологією, оскільки доступ до цих інформаційних ресурсів мали лише особи, ознайомлені з відповідною системою письма. З історичної перспективи абетка, фактично, виступала в ролі ключа, що надавала можливість отримання інформації обраним особам.

Однак, з іншого погляду, початок цієї науки можна розглядати лише як один із можливих варіантів, оскільки в більш широкому контексті цей підхід не витримує критики через відсутність переконливих доказів. Так, зокрема якщо взяти ранні уявлення та систематизацію знань людства про природу, фізику, інженерію та розглядати це як зародження таких сучасних галузей науки, як фізика, математика, інженерія, природознавство чи медицина, є вельми суб'єктивним та підлягає сумнівам через відсутність достатніх доказів.

У зв'язку з цим, перед розглядом більш докладної інформації щодо даної теми, важливо зазначити внесок вчених і науковців, які проводили дослідження у даній області та робили вагомий внесок у її розвиток. Серед таких дослідників, які внесли свій вклад у теоретичний прогрес у галузі світової криптографії, можна визначити С. А. Доріченко та В. В. Ященко [11] А. В. Бабаш, С. Г. Барічев [12], М. В. Адаменко [13], А. А. Молдовян [14], А. П. Алферов [15], А. С. Кузьмін, А. Ю. Зубов [15] та А. В. Черемушкін [15], О. В. Вербіцький [16].

Крім цього, такі вчені, як В. К. Задірака, Й. У. Мастяниця, І. М. Коваленко, Т. О. Гріненко, А. В. Аграновський [17] внесли свій вклад у

формування основних понять у цій науці. Наступним кроком буде огляд етапів розвитку цієї науки, що включає в себе історію стародавніх методів шифрування, виникнення сучасних криптографічних концепцій, таких як асиметричні алгоритми та електронні підписи, а також вивчення ключових технологічних вдосконалень, які дозволили криптографії здійснювати захист інформації в електронному віці.

Періодику криптографії заведено поділяти на 4-ри основні етапи розвитку цієї науки:

- перший етап розвитку – це так звана «наївна» криптологія (вся історія до XV сторіччя, навіть приблизної дати встановити не має можливості, а погляди науковців на цю тему досить сильно різняться);
- другим етапом розвитку цієї науки заведено вважати «формальну» криптологію (цей етап характеризується більш чіткими термінами: XV – XX сторіччя);
- третім етапом розвитку науковці виділяють «наукову» криптологію (цей етап характеризується більш стрімким розвитком ніж всі попередні і тримає з 20 по 70 рр XX сторіччя);
- останнім періодом, який триває по сьогодні, вчені вважають «комп'ютерну» криптологію (яка починається з закінченням «наукової» криптології, тобто з 70 рр. XX сторіччя і триває по сьогодні) [18].

Розгляд цих етапів розвитку надає можливість відслідковувати еволюцію криптографії від її витоків в давнині до вирішення сучасних завдань та впровадження передових технологічних рішень. Замість простого розуміння визначальних моментів, цей огляд вкладається в більш широкий контекст, розширюючи наше бачення та усвідомлення динаміки розвитку цієї важливої галузі в глобальному інформаційному суспільстві.

Підсумовуючи вищезазначене, можна сформулювати та систематизувати цю інформацію для досягнення більшої наочності та відповідної структури, що відображена на рис. 1.2. Такий підхід дозволяє лаконічно узагальнити криптографічний прогрес через етапи його історії.



Рисунок 1.2 – Схема етапів розвитку криптографічної науки

Враховуючи послідовність етапів розвитку криптології та криптографії, розглянемо вищезазначені фази більш докладно для забезпечення більшого розуміння та глибшого поглиблення у тему дослідження, розгляд даних етапів більш детально наведено у табл. 1.3.

Таблиця 1.3 – Опис етапів та характеристик розвитку криптографії

Етап розвитку	Основні характеристики
Наївна криптографія	Період наївної криптографії, коли ця наука тільки формувалась, важко визначити через відсутність письмових джерел. Цей етап характеризується використанням простих методів шифрування, таких як прості перестановки літер та моноалфавітні підстановки. Шифри Цезаря, квадрат Полібія та "Считала" є прикладами цього періоду.
Формальна криптографія (XV – поч. XX ст.)	На цьому етапі еволюції покращили попередні розробки та реалізували більш стійкі до криптоаналізу шифри. Цей період визначає початок систематизації та поліпшення наукових досліджень у галузі криптології.

Продовження таблиці 2.2

Етап розвитку	Основні характеристики
Наукова криптологія (20-70 роки XX ст.)	Цей золотий вік криптології відзначався інтенсивним розвитком, спричиненим Першою та Другою світовими війнами. Також відзначився формуванням основних розділів математики, включаючи теорію інформації, теорію чисел та кібернетику.
Комп'ютерна криптографія (з 70-х років XX ст. до сьогодення)	Цей етап відзначається широким використанням обчислювальної та комп'ютерної техніки в криптографії. Також спостерігається розширення застосування криптографії в повсякденному житті, такому як QR-коди, банківські картки, криптовалюта та інше.
Сучасний стан	Криптографія охоплює як цивільні, так і військові аспекти, конкуруючи в складності коду та технологіях шифрування. Теорія інформації та теорія зв'язку в таємних системах внесли значний внесок у її вивчення. Тенденції включають застосування криптографії в комп'ютерних мережах, електронному голосуванні та електронно-цифрових підписах.

Визначення початку періоду наївної криптографії є ключовим, оскільки ця фаза представляє собою етап начаткового розвитку криптографії. Основною визитівкою цього періоду є використання базових методів шифрування, таких як прості перестановки літер і моноалфавітні підстановки. Такі шифри, як Цезаря, Полібія та Сцитала, служать ілюстрацією методів цього періоду, в якому використання логічних перестановок літер і інших простих методів є основним [19].

Етап формальної криптографії, який відзначається удосконаленням шифрів та врахуванням помилок, є важливим кроком у формуванні фундаменту для розвитку криптології. Цей перехід від роздільних знань до їх систематизації та поліпшення є ключовим елементом цього етапу.

Етап наукової криптології характеризується активним дослідженням і розвитком науки. Визначений Першою та Другою світовими війнами, він призвів до постійної потреби у найефективніших методах захисту інформації. Впровадження концепції маневрової війни та широке використання радіозв'язку визначили його розвиток.

Етап комп'ютерної криптографії характеризується широким застосуванням обчислювальної та комп'ютерної техніки. Це призвело до поширення криптографії серед широкого загалу, включаючи технології, такі як QR коди, безконтактні платіжні картки та криптовалюта. Роль цивільної та військово-політичної криптографії залишається важливою, не зважаючи на різний рівень складності та застосування технологій шифрування.

Усі ці етапи сприяли розвитку математичних галузей, таких як теорія ймовірності та математична статистика, що стали основою для подальших досягнень в криптології. Сучасні технології та методи шифрування, враховуючи потреби сучасного суспільства, підтримують постійний розвиток цієї науки.

1.3 Розвиток історичних способів шифрування інформації

Наукове співтовариство вважає, що криптологія почала свій розвиток близько 4000 років до нашої ери, в період стародавнього Єгипту. Ця періодизація визначена науковими дослідниками на основі наявних письмових доказів застосування криптології на стародавніх папірусах. Беручи до уваги, що більш ранніх доказів не збереглося або їх ще не виявлено, ця дата вважається початком розвитку криптологічної науки.

Доказом цьому служить наявність давнього папірусу, де невідомий автор змінив ієрогліфи, проте єдиної теорії про те для чого це було зроблено, а також єдиного думки на цей рахунок серед вчених немає досі. Більшість дослідників вважає, що це було зроблено з метою викликати сильніше враження на читача, хоча точні мотиви залишаються предметом обговорення.

Важливо відзначити, що неможливо виключити можливість неточностей або простих помилок.

Насупроти твердження є те, що зміни були внесені для приховання інформації, проте існують аргументи, які відкидають цю гіпотезу. Розшифрування тексту вказує на те, що папірус містив перелік назв давньоєгипетських монументів та архітектурних об'єктів, що виключає можливість приховування інформації.

Не зважаючи на це, сам факт маніпуляції текстом з метою отримання нового значення вже є достатнім приводом вважати цю дату зародженням такої науки як криптологія та зокрема і криптографії.

У контексті еволюції шифрування інформації, наступним кроком є період від 600 до 200 років до н.е. Під час цього періоду застосовувалися методи переміщення літер або їх перемішування для отримання певного ступеню конфіденційності повідомлень. Один з найдавніших прикладів такого «заплутування» - давньоєврейський шифр атбаш, що має витоки близько 600 років до н.е. Суть цього шифру полягає в тому, що абетка зчитується в зворотньому порядку, і текст формується стандартним способом. На рис. 1.3 наведено візуальний приклад застосування цього методу до латинської абетки.

Тому, щоб відобразити такий процес шифрування, розглянемо власний етап шифрування для тексту "Some text" і отримаємо зашифрований текст: "Hlnv Gvcg". Цей приклад відображає, як застосовується конкретний алгоритм або метод шифрування до вхідного тексту, щоб отримати вихідний зашифрований варіант.

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

Рисунок 1.3 – Принцип шифрування давніх євреїв на прикладі латинської абетки

З погляду хронології наступним етапом можна визнати метод шифрування, що використовувався давньою Спартою під час конфліктів між Афінами та Спартою. Під час цих воєн спартанці використовували спеціальний метод шифрування, що називався «Скитала», для захисту конфіденційної інформації в разі потрапляння листів із повідомленнями до ворога. Хоча цей метод шифрування не представляє значної складності для сучасної криптології, він має досить оригінальну структуру. У процесі шифрування та розшифрування спартанці використовували спеціальний пристрій, відомий як «Скитала» рис. 1.4, звідки і походить назва цього методу шифрування. В сучасному світі аналогічний підхід використовується у кодових замках, хоча замість літер використовуються цифри.



Рисунок 1.4 – Вигляд пристрою Скитала

Для процесу шифрування використовувався довгий вертикальний шматок паперу, який намотувався на Скиталу. Текст повідомлення писався стандартним методом на цьому шматку, і після завершення запису папіру розматували, перетворюючи його на шифр. Після цього зашифроване повідомлення передавалося гонцю, який доставляв його до місця призначення. Отримувач використовував аналогічну Скиталу для розшифрування, намотуючи шифрований текст на неї.

У випадку перехоплення противником отримувався нерозбірливий набір літер. Наприклад, текст «текст, що потрібно зашифрувати» зашифровувався за цим методом і перетворювався у «ТЙОВЕ АКПЗТСОАИТТШ РИ ЯІФ КБР ИНУ».

Деталі цього шифрування наведено на рис. 1.5. Даний процес використання методу «Скитала» на рисунку нижче. Важливо відзначити, що ефективність цього методу залежить від того, чи знає противник ключ для розшифрування, яким є довгий циліндричний або шестикутний предмет, зображений на вищезазначеному малюнку.

1	2	3	4	5	6	7	8	9
Т	Е	К	С	Т	—	Я	К	И
Й	—	П	О	Т	Р	І	Б	Н
О	—	З	А	Ш	И	Ф	Р	У
В	А	Т	И	—	—	—	—	—

Рисунок 1.5 – Приклад шифрування способом Скитала

Правозаступником та наступним етапом розвитку шифрування тієї епохи можна вважати шифр під назвою «шифр зсуву» або його більш відома назва «шифр Цезаря». Це шифр часів давнього Риму, названий на честь давньоримського полководця Гая Юлія Цезаря, який активно його використовував для листування та надання наказів своїм полководцям. Хоча цей шифр і використовувався пізніше ніж попередній, але його модернізація торкнулася лише послідовності літер.

В цьому шифрі замість простого обертання абетки використовується зсув останніх літер абетки наперед, при цьому зсув може бути від 1 літери до половини абетки від чого і змінюється шифр, оскільки при різній кількості зсунутих літер змінюється і ключ для розшифрування. Так, наприклад при

здвигнутих перших 5 літер при використанні ключа на зсув перших 3 літер шифр не вдасться розшифрувати, тому основним критерієм криптоаналізу є знання точної кількості літер зсуву. Для більшої наочності суть такого шифру представлений на рис. 1.6.

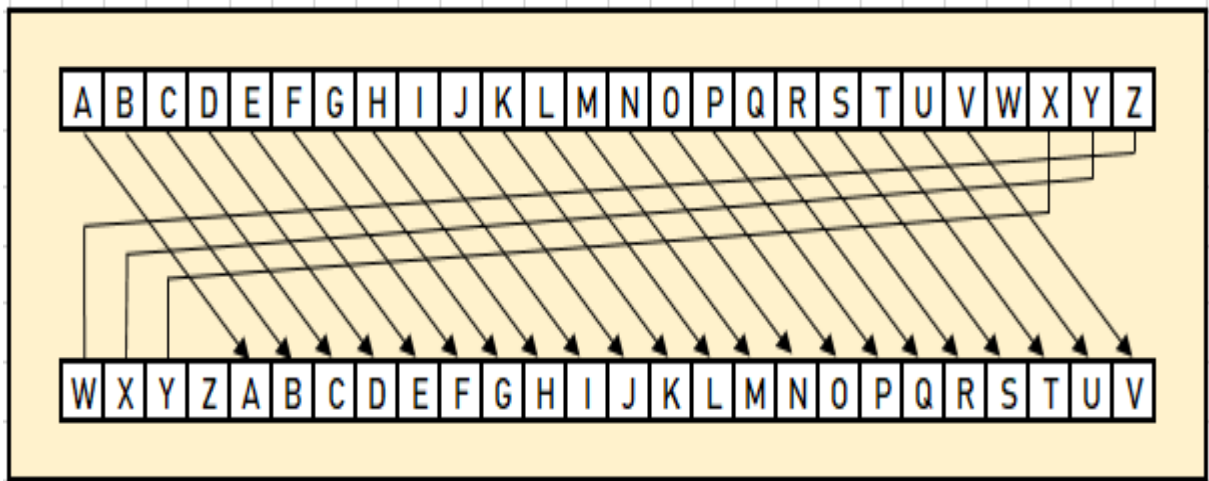


Рисунок 1.6 – Приклад Шифру Цезаря

Удосконаленою модифікацією двох попередніх шифрів є квадрат Полібія, який застосовує принцип шифрування інформації, аналогічний до попередніх, але має свої особливості.

У цьому методі шифрування літери розміщуються у квадраті розміром 5 на 5, при цьому існує одна особливість: оскільки в латинському алфавіті є 26 літер, а у квадраті поміщається лише 25 (оскільки $5 \cdot 5 = 25$), літери I та J ідентифікуються як одна літера.

Крім того, існує метод, коли при шифруванні використовується літера, яка знаходиться на один рівень нижче порівняно з тією, яку потрібно зашифрувати [20].

Для кращого розуміння цього методу суть такого шифрування надано на рис. 1.7, на якому представлено Квадрат Полібія в латинському та українському варіантах алфавітів. Завдяки цьому методу шифрування можна зашифрувати текст як у числовому виразі, так і у літерному, згідно представленому зразку.

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I/J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

а

	1	2	3	4	5	6
1	А	Б	В	Г/Ґ	Д	Е
2	Є	Ж	З	И/Й	І/Ї	К
3	Л	М	Н	О	П	Р
4	С	Т	У	Ф	Х	Ц
5	Ч	Ш	Щ	Ю	Я	Ь

б

а - варіація латинцею; б - варіація кирилицею.

Рисунок 1.7 – Варіації Квадрата Полібія різними абетками

На українських землях за часів Київської Русі, також існувала система шифрування інформації під назвою «Літероя». Принцип цієї системи був схожий на шифр Цезаря, де одні літери заміщувались іншими, але існували власні особливості. Літероя поділялась на два види: просту і мудру.

Принцип простої Літерої майже нічим не відрізнявся від раніше описаних методів. У мудрій Літерої деякі літери замінялись комбінаціями крапок і рисків. Цей метод схожий на шифрування «Морзе», де перша буква відповідає одній крапці, друга - двом крапкам, і так далі до певної кількості крапок. Після цього застосовувався аналогічний метод, але замість крапок для наступних літер використовувались риси і кола. Стуність такого "морзянського" шифру Київської Русі можна побачити на рис. 1.8.

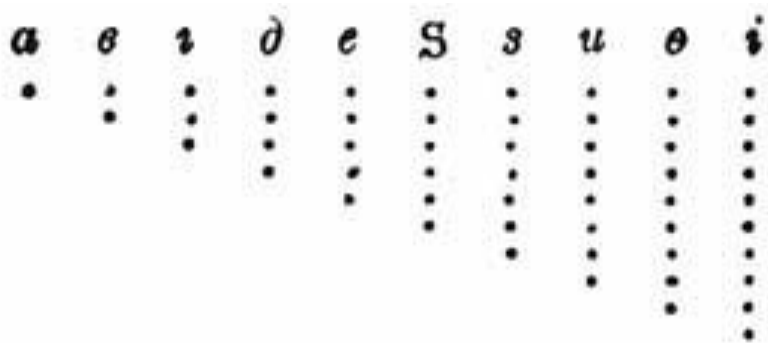


Рисунок 1.8 – Мудра Літороя

Проте справжній розвиток криптографії відбувається в епоху відродження, яку розглядають як перехід від наївного до формального етапу в розвитку цієї науки. У цей період відзначається появою новаторських шифрів та значним покращенням вже існуючих. Наприклад, італійський вчений Леон Альберті в 1466 році винайшов шифрувальний диск, який, в сутності, є удосконаленою версією шифру Цезаря. Цей диск складається з двох кіл різного діаметру, одне з яких використовується як стандартна абетка, а інше - як засіб для шифрування.

Для шифрування потрібно було перемістити внутрішнє коло на необхідну кількість обертів і при написанні речення знаходити необхідну букву на зовнішньому колі, а брати букву ідентичну потрібній, але з внутрішнього. Для більшої наочності цей «шифрувальний пристрій», періоду Ренесансу, можна побачити на рис. 1.9. Проте це не всі досягнення даного науковця, він також є автором одного з перших трактатів про криптографію під назвою «Трактат про шифри».



Рисунок 1.9 – Шифрувальний диск Леона Альберті

У XVI столітті криптологія отримала новий імпульс розвитку, а визначеним вченим цього періоду є француз Блез де Віженер. Він вперше винайшов абсолютно стійкий шифр, що використовував 26 алфавітів для шифрування тексту. Незважаючи на схожість із шифром Цезаря, основна складність полягала в тому, що для криптоаналізу зашифрованого тексту потрібно було успішно пройти дві ступені захисту. Перша ступінь передбачала знання відповідних алфавітів, а друга - ключа для зсуву букв. Це ускладнювало завдання криптоаналізу в декілька разів, зробивши цей шифр практично непроникним на певний час.

Кажучі більш узагальнено, цей період є значущим для криптології, оскільки деяким науковцям знадобилося від десятків років до кількох століть на криптоаналіз певних шифрів.

Наступний етап у розвитку криптографії припадає на період промислової революції, а видатною постаттю цього часу є один з першовідкривачів Сполучених Штатів Америки – Томас Джефферсон. У 1790 році він створює криптографічний пристрій, відомий під назвою "циліндр Джефферсона". Цей пристрій складається з кількох дисків, кожен з яких містить усі літери алфавіту, і розташування літер на кожному диску відрізняється від одного до іншого.

Ці диски послідовно насаджуються на вісь, після чого фіксувались і після цього за допомогою спеціального важеля вся конструкція могла рухатись без прокрутки окремих дисків. Отримувачу залишалось тільки прокрутити диски до того стану поки не буде отримано потрібно повідомлення, отримання двох читабельних повідомлень вкрай мала, до того ж той хто зашифровує інформацію має змогу перевірити наявність такого варіанту.

Більш наочно циліндр Джефферсона представлений на рис. 1.10 [20]. Цей пристрій також можна вважати першим механічним криптопристроєм нового часу, хоча принцип по своїй суті це вдосконалений та покращений принцип шифрування за допомогою Скитали.



Рисунок 1.10 – Циліндр Джефферсона [20]

Наступним прискорюючим фактором у розвитку криптографії було винайдення електронного телеграфу в 1837 році. З цього моменту можна почати спостерігати початок ери формальної криптографії та перехід до більш наукового підходу.

Винахід електронного телеграфу розширив можливості досліджень і вдосконалення криптографії, що суттєво підштовхнуло розвиток цієї науки. Цей винахід виявився додатковим каталізатором для росту, оскільки попередні методи стали морально застарілими, а потреба у високоякісному забезпеченні безпеки під час передачі конфіденційної інформації значно зросла через великі військові конфлікти та революції, що охопили весь світ протягом XIX століття та на початку XX.

Візитівкою цієї епохи можна вважати шифр Плейфера. Цей шифр вже виявляється складним та, до певної міри, новаторським. Його основу складає принцип квадрата Полібія, який використовується для створення ключа, за допомогою якого здійснюється шифрування та розшифрування інформації.

Для формування ключа обирається слово-ключ, для розкриття суті цього шифрування, візьмемо англійське слово "Commander". Потім це слово-ключ записується у квадрат, виключаючи букви, які повторюються у слові більше 1 разу.

Пусті клітинки заповнюються невикористаними у слові-ключі літерами абетки, створюючи таким чином алгоритм для шифрування та розшифрування.

Шифрування виконується за допомогою біграм (1 біграма – це пара літер у слові, з додаванням літери "пустишки" для слів з непарною кількістю літер). Процес шифрування базується на низці правил, додатково може включати вимір літер як чисел, що призводить до отримання надійно зашифрованої інформації на багато рівнів. Цей метод шифрування широко використовувався англійською армією від часу його винайдення до початку Другої світової війни, коли його замінили більш сучасні та ефективні методи шифрування, засновані на обчислювальних технологіях.

Протягом Другої світової війни сторони використовували передові на той час шифрувальні машини, коди яких вважались практично непроникними. Зазвичай такі пристрої мали роторний механізм, хоча існували і модифікації на основі цеволочних дисків. Один із визначних прикладів цього періоду - німецька шифрувальна машина «Енігма», коди якої було розкрито завдяки математичним розрахункам Алана Тюрінга та його спеціально створеному пристрою під назвою «Бомба». Зображення цього легендарного шифрувального пристрою наведено на рис. 1.11 [21].



Рисунок 1.11 – Шифрувальний пристрій Енігма [21]

Американський еквівалент «Енігми» включає в себе шифрувальний пристрій, відомий як M-209, що володіє функціональністю подібною до

німецької шифрувальної машини. Зовнішній вигляд цього шифрувального пристрою представлений на рис. 1.12 [22].



Рисунок 1.12 – Шифрувальний пристрій М-209 [22]

Після завершення Другої світової війни, роторні шифрувальні машини протягом двадцяти років уступають місце новітнім та більш вдосконаленим блоковим шифрам, що можна вважати останніми досягненнями епохи наукової криптології та криптографії.

З появою та розповсюдженням комп'ютерної техніки криптологія переходить на новий якісний рівень, відомий як «комп'ютерна криптографія». Нові обчислювальні машини, що тепер називаються комп'ютерами, дозволяють створювати значно більш складні шифри, виводячи криптографію на абсолютно новий рівень.

Наприклад, з використанням обчислювальної техніки тепер можна ламати коди, подібні до Енігми, за декілька годин чи навіть хвилин, у порівнянні з місяцями, які це займало у союзників під час Світової війни, коли вони мали використовувати спеціалізовані машини для взлому таких кодів.

У 1978 році був розроблений стандарт DES, який послужив основою для багатьох сучасних криптографічних алгоритмів. З цього моменту розпочинається ера новітньої криптології, яку ми зараз спостерігаємо. DES

став першим широко використовуваним стандартом для шифрування даних і грав ключову роль у встановленні стандартів безпеки. Його вплив на розвиток криптографії надалі підсилюється введенням сучасних алгоритмів, таких як AES (Advanced Encryption Standard) у 2001 році, який став наступником DES і дотримується високих стандартів стійкості та безпеки.

1.4 Класифікація криптографічних шифрів

Після проведеного аналізу та врахування вищезазначеного, давайте розглянемо класифікацію криптографічних шифрів. Для побудови вірної класифікації необхідно спочатку визначити одну з ключових характеристик – обсяг інформації, яку потрібно захистити від несанкціонованого доступу третіх сторін. Крім того, важливо мати інформацію про інші ключові параметри, такі як рівень захисту, необхідний для забезпечення безпеки, а також часові параметри для шифрування, передачі та розшифрування. Ці параметри впливають на розробку та створення системи класифікації.

До першої та найпростішої класифікації можна віднести таку, яка може розділити всі шифри на основі доступності інформації про процес шифрування, визначеної у відкритих джерелах, та їх приналежність до певного класу. Зокрема, можна виділити дві основні групи: шифри з відкритою інформацією та шифри з закритою інформацією. До першої групи відносяться ранні історичні шифри, де сам процес шифрування був недосконалим, і основний рівень криптостійкості залежав від знань отримувача щодо особливостей шифрування та розшифрування. Також до цієї групи входять специфічні шифри, інформація про процес кодування яких практично недоступна відкритому доступу. Такі шифри зазвичай створюються з метою додаткового ускладнення криптоаналізу при перехопленні інформації і мають високий рівень конфіденційності.

До другої категорії відносяться сучасні шифри, інформацію про суть шифрування яких можна знайти у відкритому доступі. Вони широко

використовуються, і ступінь захисту цих шифрів забезпечується створенням достатньої кількості варіантів розшифрування, щоб сам процес, навіть при великих обчислювальних можливостях, займав такий час, щоб доцільність розшифровка стала неприйнятною. Ці методи мають перевагу в тому, що їх можна вивчити кожному за короткий період часу, при цьому вони забезпечують достатній рівень захисту. Такі шифри можна характеризувати як широко поширені. У сучасній криптографії ключі до таких стандартних шифрів, у більшості випадків, мають довжину від 56 до 4096 біт [23].

Для подальшої класифікації шифрів, необхідно визначити основні цілі такого шифрування. З метою більшої наочності ми визначили та групували основні завдання на рис. 1.13.

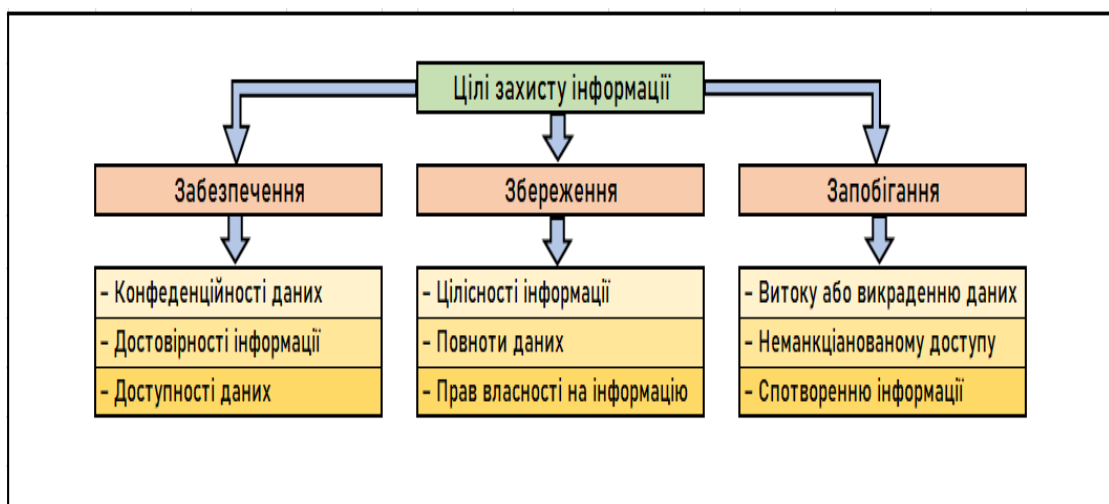


Рисунок 1.13 – Основні цілі захисту інформації

Відповідно до рисунку, всі цілі можна розділити на три основні категорії: збереження, забезпечення та запобігання, кожна з яких в подальшому розглядається як набір трьох підкатегорій.

З метою більшої наочності ознайомимося із зазначеними завданнями більш докладно. На сучасному етапі розвитку шифрування інформації ключовими завданнями є:

— забезпечення достатнього рівня конфіденційності інформації (тобто зашифрованого повідомлення);

- забезпечення цілісності та повноти інформації (мається на увазі здатність забезпечити достовірність та відповідність інформації до аналогічного варіанту до процесу криптографічного шифрування);
- забезпечення стану безслідності (не відстежуваності) дій користувача з інформацією;
- забезпечення доступності інформації без втрати рівня її захищеності (мається на увазі стан при якому для отримання доступу до інформації не потрібно проводити велику кількість маніпуляцій, а також це не буде займати велику кількість часу для отримання доступу до інформації для її використання чи редагування).

До всього вищезазначеного можна також додати, що кожен сучасний шифр та криптографічний алгоритм має свій унікальний набір характеристик, які переважно визначають області його використання та застосування. Отже, для побудови класифікації необхідно додатково розглянути ці характеристики.

До таких характеристик сучасних криптоалгоритмів та шифрів можна віднести:

- спосіб роботи з ключем;
- розміри даних з якими працює та може працювати криптоалгоритм та шифр;
- складність шифру та затрачувана кількість часу, яка потрібна для роботи з криптоалгоритмом (зокрема кількість часу, яка витрачається на шифрування та розшифровку шифру);
- ступінь надійності та рівень, який потрібен для криптоаналізу шифру.

Враховуючи все вищезазначене, можна провести класифікацію, за якою шифри можна розподілити відповідно до:

- за особливостями ключа шифру (симетричні та асиметричні, електронні ключі типу електронно-цифрового підпису, типу хешування);
- за кількістю ключів (з одним ключем, з декількома ключами);

- за методом шифрування даних (шифри перестановки (мішання інформації) та шифри заміни та комбіновані;
- за технологією шифрування інформації (блокові або потокові).

Висновки за розділом 1

У першому розділі було досліджено фундаментальні теоретичні аспекти криптології, включаючи широкий спектр термінів у сфері криптографії та криптоалгоритмів. Аналіз цих термінів надав докладну основу для подальшого розгляду теми дослідження. В другому підрозділі була представлена загальна історія криптології та криптографії з метою поглибленого розуміння контексту. Розкриття історичних аспектів дозволило глибше вивчити історичні методи шифрування, які історично відомі також як тайнопис.

Наступним етапом в дослідженні обраної теми було вивчення історичних методів тайнопису. Цей аналіз надав більше загального розуміння та експертизи у вибраній темі, а також забезпечив необхідні знання для подальшого дослідження. Крім того, цей розділ відкрив можливість далекоочікуваного погляду на досліджувану тему. Під час аналізу стало зрозуміло, що всі історичні методи шифрування та тайнопису, без винятку, базуються на раніше винайдених принципах та в різних аспектах їх поєднують та удосконалюють.

Після ретельного дослідження зазначених аспектів було розроблено класифікацію методів шифрування, щоб систематизувати отримані знання. Результати цієї роботи відкривають можливості та надають необхідні знання для подальших досліджень у вибраній темі.

2 МЕТОДОЛОГІЧНІ ОСНОВИ ТА СПОСОБИ ШИФРУВАННЯ ІНФОРМАЦІЇ

2.1 Проблематика розвитку сучасної криптографії

Розглядаючи аспекти проблематики сучасної криптографії та криптології, слід відзначити, що сучасна наука вже має значну кількість наукових досліджень та відкриттів у цій галузі. З цим пов'язані нові можливості для розвитку, що виникають завдяки постійному вдосконаленню та технічному розвитку обчислювальної техніки.

Проте, цей розвиток також викликає певні проблеми, такі як збільшення потенційних можливостей криптоаналізу внаслідок росту обчислювальної потужності.

Незважаючи на це, сучасні досягнення в галузі криптографії та криптології, а також безперервний прогрес обчислювальної техніки, відкривають нові перспективи для створення більш надійних та ефективних методів шифрування.

Згідно дослідження компанії «Juniper Reseach» та прогнозами на його основі, втрати компаній та банків по всьому світу у 2019 році складуть більше 2.1 мільярдів доларів тільки від витоку інформації [24]-[25].

Паралельно із зазначеним, слід відзначити, що із поширенням інформаційно-комунікаційних технологій серед широких мас населення виникла проблема різного рівня важливості інформації.

Оскільки очевидно, що значення інформації в обміні листами між двома звичайними громадянами відрізняється від важливості та цінності комерційних або військових даних, це відзначається різним рівнем важливості, що, в свою чергу, впливає на визначення обсягу ресурсів, необхідних для захисту цієї інформації.

Після аналізу основних проблем можна визначити послідовність кроків, яка сприятиме вивченню вразливостей систем криптографічних алгоритмів.

Розглянемо ці проблеми у більш конкретному контексті:

- збільшення потенційних можливостей криптоаналізу внаслідок росту обчислювальної потужності;
- парадоксальний характер дослідженості та зростання потужності обчислювальної техніки: розвиток надійних методів, але з ризиком вразливості шифрів;
- високі втрати компаній та банків від витоку інформації, викликані сучасними досягненнями в галузі криптографії та криптології;
- різний ступінь важливості інформації та необхідність визначення рівня захисту для кожного виду та типу інформації;
- віковична природа науки та обмеженість у кількості робочих схем шифрування даних [26];
- відсутність схем, які б базувались на нових принципах та не спирались на невирішені проблеми сучасної математики;
- зростання розмірів блоків даних та ключів, викликане розвитком математики та інформатики;
- відсутність далеких перспектив та проблема пов'язаності криптосистем через базування на однакових основах;
- постійне зростання розмірів блоків даних та ключів, як відповідь на розвиток математики та інформатики;
- відсутність чітких перспектив у розвитку сучасної криптографії та потенційні ризики у випадку зламування невирішених математичних проблем;
- розробка квантового комп'ютера як потенційного розв'язання проблем сучасної криптографії та його поки що експериментальний характер [27].

Глибокий аналіз проблематики сучасної криптографії та криптології розкриває ряд труднощів та викликів, пов'язаних із стрімким розвитком обчислювальної потужності та наукових досягнень. Безперервний характер цієї галузі стимулює вдосконалення існуючих методів шифрування, оскільки

створення абсолютно нових методів важливіше є збором та вирішенням накопичених проблем.

Надійність сучасних криптографічних алгоритмів базується на математичних задачах, які можуть втратити ефективність через подальший розвиток математики. Алгоритми, такі як RSA, ґрунтуються на складних математичних проблемах, які, в свою чергу, можуть втратити свою стійкість у разі подальших математичних відкриттів.

Зростання розмірів блоків даних та ключів, спричинене розвитком математики та інформатики, породжує проблеми, оскільки вимоги до розмірів зростають, що може призвести до ускладнень та ризиків в розробці криптосистем.

Відсутність далекоглядності та проблема пов'язаності криптосистем виникають через стрімкий розвиток техніки та зростання вимог до криптосистем. Винайдення квантового комп'ютера розглядається як можливий вихід, але на сьогодні його експериментальний характер не гарантує масового використання у найближчому майбутньому.

У світлі цих викликів квантова криптографія виступає як потенційно ефективний метод розв'язання проблем, але її ефективність та масове використання залишаються предметами подальших досліджень. Вже зацікавленість деяких великих корпорацій свідчить про потенційне майбутнє цієї технології, але її широке впровадження залишається поки що науковою фантастикою [27].

2.2 Основні методи та вимоги до захисту інформації

Враховуючи весь попередній аналіз у роботі, логічно дослідити методи та алгоритми, які забезпечують конфіденційність та захист інформації. З загального висновку випливає, що всі існуючі на сьогодні схеми та алгоритми шифрування не рівнозначні за надійністю, швидкістю та вартістю. Отже, вибір відповідного методу вимагає попередньої класифікації

відповідно до потреб у захисті інформації, необхідного функціоналу алгоритму та фінансових можливостей замовника.

Найвищий рівень захисту, супроводжуваний високою швидкістю роботи, має відповідно високі вартість та витрати ресурсів. У той час як не для всієї інформації необхіден максимальний рівень захисту, оскільки важливість різної інформації різна, а зломисник вибере оптимальний спосіб для взлому, враховуючи ймовірний дохід та ризики.

Тому для успішного розв'язання цього завдання необхідно систематизувати методи захисту інформації та вибрати той, який відповідає конкретному завданню. Це надзвичайно важливо, оскільки правильний вибір методу так само важливий, як і процес створення криптосистеми для захисту інформації. Вибір вірного методу вимагає знаходження оптимальної рівноваги, яка забезпечить достатню криптостійкість, ефективність роботи та відповідність поставленим завданням.

Проте перед переходом до аналізу криптографічних методів захисту інформації слід ретельно розібратися у їхньому призначенні. Криптографічні методи захисту інформації можна визначити як специфічні техніки кодування та шифрування даних, або інші перетворення та операції над інформацією та даними. Результатом цих дій є отримання зашифрованої інформації, недоступної для будь-якої сторони без здійснення обернених процесів перетворення. Для здійснення цих процесів необхідно використовувати криптографічний ключ.

Перед розглядом самих методів слід також зауважити, що інформація захищена за допомогою криптографічних методів шифрування є більш стійкою до викрадення і на сьогодні є найбільш надійним методом захисту інформації та даних, бо охорона сфокусована на самій інформації, а не на доступі до неї.

Так, викрадений файл, зашифрований за допомогою криптографічних методів, є більш стійким до криптоаналізу, ніж файл, захищений тільки паролем. Оскільки зломисник навіть отримав носій інформації, де

знаходиться цей файл, і отримавши доступ до нього, все одно не зможе витягнути звідти первинну інформацію тому, що для отримання доступу в будь-якому варіанті потрібен буде ключ.

Розглянемо конкретні методи шифрування даних, розроблені з метою кращого вивчення матеріалу та для ілюстрації на рис. 2.1.

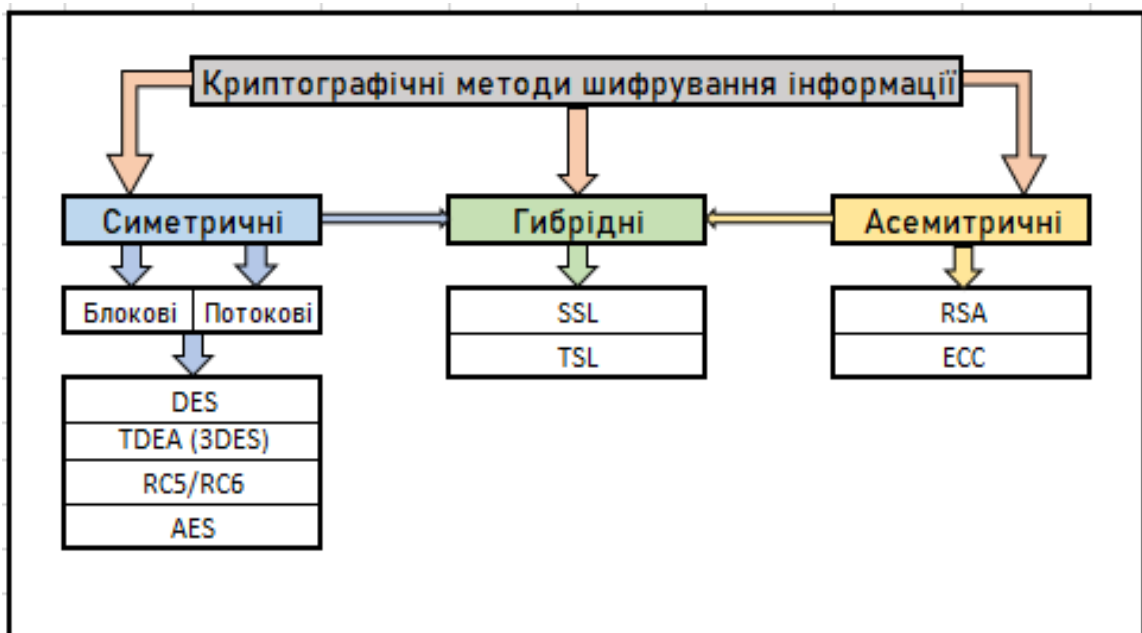


Рисунок 2.1 – Основні методи шифрування інформації

Більшість криптографічних методів захисту інформації реалізуються або у вигляді спеціальної програми, або у вигляді спеціальних пакетів для програм, за допомогою яких й проходить шифрування даних. Загальна логіка побудови цих програм чи пакетів схожа на ті історичні способи шифрування, які були розглянуті раніше.

Тобто для шифрування чи розшифрування даних потрібен ключ та програма або пакет, в якому запрограмована логіка шифрування та розшифрування. Подібно історичним матричним методам шифрування даних.

Однією із перших ознак за якими можна розділити сучасні методи криптографічного захисту інформації є вимоги, які виставляються до таких методів захисту даних.

Тому, основними вимогами до криптографічних методів шифрування є:

- першою вимогою, яка висувається до сучасних методів шифрування є те, що знання логіки алгоритмів не повинно жодним чином впливати на надійність захисту інформації, а також не повинно зменшувати криптостійкість алгоритму;

- другою, не менш важливою вимогою, є те, що алгоритм повинен мати такий рівень криптостійкості, який ускладнює взлом методами перебору та підбору ключів на рівні, який виходить за можливості сучасної обчислювальної техніки, доступної для вільного придбання, або хоча б був максимально близький до цього рівня (з урахуванням можливостей мережевого обчислення);

- третьою, не менш важливою вимогою, хоча і досить очевидною є, вимога до перегляду зашифрованої інформації, а саме, перегляд інформації повинен бути можливий тільки при умові наявності та надання ключа;

- четвертою вимогою є те, що при незначних змінах у ключі, такі дії повинні приводити до істотних змін у зашифрованому повідомленні;

- п'ятою вимогою до методів шифрування, а саме до їх ключів є те, щоб зміна довжини ключа не призводила до зменшення поріг криптостійкості шифру;

- разом з цим також висуваються вимоги способів реалізації, а саме шифр повинен мати можливість як до апаратної, так і до програмної реалізації без втрат при цьому своєї криптостійкості;

- останньою вимогою є те, що довжина вихідного тексту та зашифрованого тексту повинні бути не змінні протягом процесу шифрування та розшифровки, для уникнення неточностей та проблем під час шифрування та розшифровки, а також їх довжина не повинна дуже сильно різнитись, щоб уникати помилок при розшифровці.

Аналізуючи всі ці вимоги до алгоритмів і методів шифрування інформації, а разом з цим беручи до уваги те, що для різних

завдань, а також для кожної окремої групи даних потрібні різні методи, які б задовольняли виконання заявлених критеріїв не дивно, що на сьогодні на ринку представлений дуже великий спектр алгоритмів та методів шифрування інформації та даних.

Кожен окремий алгоритм чи метод відрізняється від іншого як своїми загальними характеристиками та стандартами щодо шифрування інформації, так і принципами шифрування. Слід зазначити, що не всі ці методи та алгоритми однаково ефективні та надають достатній рівень захисту. Зустрічаються навіть такі, які, хоч і оформлені як стандартні, не можуть забезпечити хоч якийсь захист інформації.

При поглибленому вивченні вони не витримують жодної критики і слугують більше як видимість захисту, ніж реальний захист. Здебільшого це застарілі алгоритми, оскільки в наш час при досить стрімкому розвитку науки та техніки, вчорашній надійний алгоритм або метод сьогодні або завтра вже буде викликати сумнівний рівень криптостійкості.

Також слід зауважити, що в сьогоднішніх умовах, створення методу шифрування, який би забезпечив достатній рівень захисту інформації, враховуючи всі вимоги, є досить складною задачею. При створенні нового методу або алгоритму, перш за все, потрібно брати до уваги тенденції розвитку науки та техніки, а також можливості, які вони зможуть забезпечити для криптоаналізу методу або алгоритму, що створюється. Проте також потрібно, по можливості, спробувати попередити можливі фактори, що зможуть полегшити взлом.

Щодо методів шифрування, то перша група таких методів захисту інформації та даних спирається на традиційне знання відправником інформації та її отримувачем спеціального секретного ключа. Суть даного методу криптографічного шифрування інформації, або як його ще інакше називають в наукових колах - симетричною криптографією, полягає у тому, що відправник при шифруванні інформації використовує ключ для шифрування інформації, а її отримувач використовує той самий

ключ для розшифрування повідомлення. В сучасних умовах, звісно, функції шифрування та розшифрування виконується комп'ютерами.

Основними ж перевагами даного методу шифрування є:

- даний метод шифрування є хорошою базою для створення нових криптографічних алгоритмів;
- володіє достатньо великою пропускнуою здатністю, що позитивно скасується при шифруванні великих об'ємів даних;
- може працювати з відносно невеликими ключами;
- має менші вимоги до обчислювальних можливостей техніки для роботи ніж інші методи;
- мають можливості для покращення своєї криптостійкості, завдяки своїй сумісності з іншими методами.

Серед основних недоліків можна виділити:

- для забезпечення стійкого рівня криптостійкості потрібна регулярна зміна ключів;
- має слабкості в області збереження конфіденційності ключа у великих мережах, а також важчий в адмініструванні великих мереж (через проблему таємного узгодження ключа, а також через необхідність регулярної, частой зміни цього ключа);
- має слабкі місця при масштабізації (використанню у великій системі), а також через велику кількість ключів, які потрібно розповсюдити на велику кількість користувачів.

Однією з слабкостей даного методу криптографічного шифрування проявляється в тому, що якщо комусь, будь-яким способом, вдасться викрасти ключ то він зможе отримати доступ до даних. Тому для підтримки криптостійкості та запобіганню таких ситуацій ключ потрібно змінювати через певні проміжки часу.

Разом з цим ще одним недоліком такого методу є проблематика регулярного конфіденційного узгодження ключа, оскільки при передачі ключа мережею існує вірогідність перехоплення ключа.

Проте ця проблема частково вирішується за допомогою вбудованого у деякі програми для такого шифрування генератора ключів, який при пов'язуванні між собою деяких програм дає змогу шифрувати сам ключ як певний перелік дій для генерації потрібних ключів.

Тобто для отримання доступу до інформації у користувача повинна бути програма та генератор ключів, який пов'язаний за допомогою авторизації користувача і дозволяє генерувати ключі, в яких у шифрі вже вказаний показник, який дозволить програмі зрозуміти, що вхід ініціює власник інформації або особа, якій дозволено вхід. Зазвичай такий захист називають двоетапною ідентифікацією.

Наступним методом для шифрування даних є так званий асиметричний метод. Даний метод виник як реакція на проблеми попереднього. Винахідником даного методу є американський криптограф Уїтфілд Діффі, який винайшов та опублікував даний метод у 1976 році.

Основна суть даного методу у тому, що він має два ключі. Один з яких персональний, тобто кожний користувач має свій персональний ключ, який відомий тільки йому, а інший – це публічний ключ, який надається усім користувачам, та який призначений для отримання доступу до інформації.

Таке поєднання публічного та особистого ключів дає можливість відкинути проблему конференційної передачі ключа між отримувачем та відправником, для шифрування та розшифрування інформації.

Проте також це дещо збільшує криптостійкість, оскільки для доступу до інформації потрібно надати обидва ключі. На сьогодні такий метод також дає можливість відстежувати дії користувачів, і у разі витоку інформації, первинно можна навіть виявити з якого аканту був витік цієї інформації.

Такий захист досягається завдяки певній математичній залежності між ключем, що використовується для шифрування, та процесом розшифрування.

Щодо основних переваг цього методу шифрування даних, то тут існує можливість виділити наступні переваги:

- відсутність необхідності частоті регулярної зміни ключів, при цьому рівень криптостійкості через це не зменшується;
- при використанні у великих мережах, легший в адміністрування через меншу кількість необхідних ключів;
- асиметричний метод шифрування позбавлений недоліку та проблеми обміну секретними ключами.

Серед основних недоліків цього методу можна виділити наступні:

- складніший при адмініструванні та корегування алгоритмів, які побудовані на цьому методі шифрування;
- для стабільної роботи вимагає та використовує великий об'єм ресурсу обчислювальної техніки;
- для досягнення стійкого рівня криптостійкості потрібно використовувати великий розмір ключа.

Наступною групою методів шифрування є так звані гібридні методи шифрування. Методи шифрування, які входять в цю групу методів шифрування використовують і симетричне і асиметричне шифрування для отримання більшої криптостійкості та надійності. Такий підхід дозволяє перекрити слабкі сторони кожного з попередньо представлених методів за рахунок сильних сторін кожного з них.

Основна думка цих методів полягає у тому, що симетричний метод шифрування даних використовується для шифрування великих об'ємів даних, тобто вихідного тексту, а для шифрування ключа, для його безпечного передавання отримувачу інформації зазвичай використовуються асиметричні методи шифрування, шляхом генерування ключа для симетричного методу шифрування подальшого його шифрування та передачі разом із зашифрованою інформацією.

Генерація ключа проходить таким чином, щоб доступ до цього зашифрованого ключа для розшифрування інформації, зашифрованої

симетричним методом шифрування, міг отримати тільки володар секретного ключа асиметричного методу. Таким чином, інформація і ключ для її розшифровки перебувають у захищеному стані, що звичайно підіймає надійність такого шифрування.

Наприклад, такі протоколи шифрування даних рівня захищених сокетів, як SSL та протокол захисту транспортного рівня TLS. Обидва ці протоколи є протоколами захисту даних в інтернеті, та обидва ці протоколи у своїй основі використовують для шифрування даних саме гібридні методи шифрування. Зокрема, більшість браузерів використовують транспортні протоколи TLS.

2.3 Основні напрямки та способи криптографічного шифрування даних в сучасному світі

Дослідивши методи та проблеми сучасної криптології та криптографії, наступним логічним кроком для більшого поглиблення у тему буде розгляд підходів та способів до сучасного шифрування інформації.

При цьому варто зауважити, що підходи до шифрування, це лише область застосування та логіка, завдяки якій шифруються дані. Звісно, деякі логіки можуть обмежувати варіацію алгоритмів, які можна використати при шифруванні, проте в багатьох випадках є вибір.

Окремими групами підходів до шифрування можна виділити три основні напрямки:

- перший – це симетричне шифрування, де використовується один і той же ключ для обох процесів – шифрування та розшифрування;
- другий напрямок – асиметричне шифрування, яке використовує два ключі: публічний та приватний;
- третій напрямок – використання хеш-функцій, що генерують фіксовані рядки з вхідних даних, забезпечуючи перевірку цілісності та унікальності інформації. До них зокрема можна віднести наступні у табл. 2.1.

Таблиця 2.1 – Опис підходів до шифрування інформації

Підхід до шифрування	Опис
Шифрування на рівні стовпців	Цей підхід базується на шифруванні стовпців даних, де кожній клітинці в стовпці привласнюється однаковий пароль. Використовується для шифрування інформації в базах даних
Шифрування на рівні полів	Цей підхід використовується для шифрування даних на рівні полів у вебсайтах. Кожне поле шифрується за допомогою унікального ключа
HTTPS	Використовується для шифрування всієї інформації на вебсайтах, застосовуючи протокол HTTPS, який накладається поверх алгоритму шифрування TLS. Забезпечує захищеність даних у мережі
Шифрування на каналному рівні	Шифрує потік даних перед їх відправленням для запобігання перехопленню під час передачі. Дані розшифровуються при отриманні на приймачі. Кожен канал може мати свій унікальний ключ для підвищення криптостійкості
Наскрізне шифрування	Забезпечує шифрування даних під час їх передачі з пристрою А до пристрою Б. Використовується, наприклад, у соціальних мережах
Заперечне шифрування	Використовує кілька засобів для шифрування, створюючи запутану мережу даних. Лише один або декілька пакетів містять правдиву інформацію, інші містять дезінформацію або зашифровані символи. Ускладнює виявлення правдивих даних

Продовження таблиці 2.1

Підхід до шифрування	Опис
Гомоморфне шифрування	Забезпечує шифрування даних так, щоб вони виглядали, начебто це вихідні дані. Важко розшифрувати такий шифр, і його можна повторно зашифрувати для підвищення криптостійкості
Повнодискове шифрування	Шифрує дані на рівні жорсткого диска, забезпечуючи їх непрочитуваність без відповідного ключа. Важко взламати без спеціальної підготовки
Шифрування у хмарі	Використовується для захисту даних, які зберігаються в хмарі (мережеве сховище за межами власності користувача)
Шифрування на мережевому рівні	Реалізується через спеціальний криптосервіс, створюючи приватну систему для безпечного обміну інформацією в мережі. Здебільшого використовується через мережевий протокол безпеки IPSec
Квантове криптографічне шифрування	Цей підхід є новим та на сьогодні не дуже поширеним у комерційному використанні. Забезпечує високий рівень безпеки через використання принципів квантової механіки. Хоча є дорогим та вимагає спеціалізованого обладнання, але складність криптоаналізу робить його одним із найнадійніших методів захисту даних. Спосіб та алгоритм захисту даних залежить від обраного методу квантової механіки

Представлені різноманітні методи шифрування використовуються у різних сценаріях, таких як бази даних, вебсайти, мережі та хмарові сервіси. Кожен метод має унікальну область застосування, наприклад, шифрування на рівні стовпців ефективно для баз даних, в той час як HTTPS шифрування застосовується для всього вебсайту.

Деякі підходи, такі як квантове криптографічне шифрування, вирізняються високою надійністю через складність криптоаналізу, що обумовлено труднощами вимірювання шифрованих даних без порушення процесу декодування. Інші методи можуть бути менш надійними в певних умовах.

Ураховуючи різноманітні загрози та особливості сценаріїв використання, ефективним може бути використання комбінації різних методів для максимального захисту даних.

Складність криптоаналізу ускладнюється тим, що точне вимірювання розташування та імпульсу шифрованих даних без порушення процесу декодування є практично неможливим. Невдалий доступ або спроба копіювання даних призводить до їх зміни та декодування, викликаючи попередження системи безпеки та авторизованих осіб. Такий спосіб шифрування можна порівняти з сучасною версією «Енігми».

2.4 Види алгоритмів та сучасні напрямки криптографічного шифрування даних

Після аналізу основних методів та підходів до криптографічного шифрування даних, наступним етапом для подальшого дослідження буде розгляд криптографічних алгоритмів, які використовуються у процесі шифрування даних. З метою систематизації досліджень криптографічних алгоритмів використовуватимемо основні методи, що раніше були розглянуті у цьому контексті. Разом з цим структуру криптографічних алгоритмів також подано на рис. 2.2.

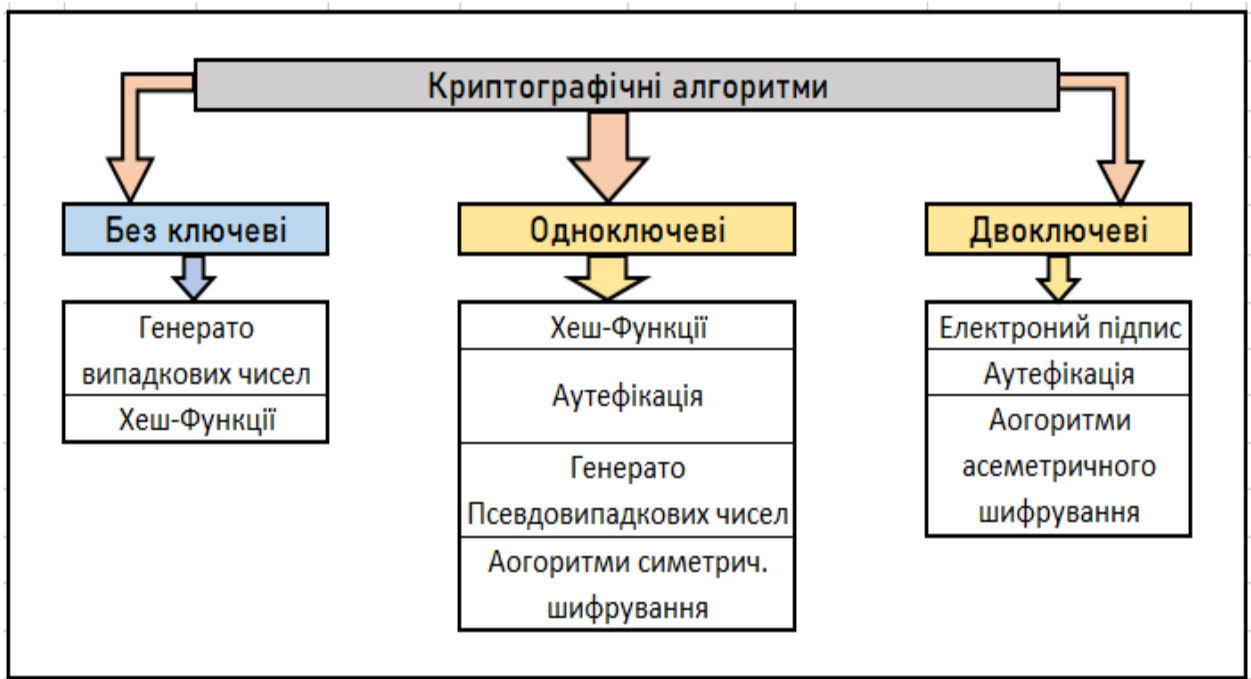


Рисунок 2.2 – Криптографічні алгоритми

Першої групою криптографічних алгоритмів для захисту інформації є: криптоалгоритми, які можна віднести до симетричного методу шифрування інформації. Слід також зауважити, що групу симетричних алгоритмів шифрування інформації також можна поділити на потокові та блокові. Перша група алгоритмів (потоків) оброблює вихідний текст послідовно в міру проступання інформації, звідси і їх назва.

Друга ж група алгоритмів (блокові) працюють з інформацією поділяючи її на рівні частини – блоки, довжина одного блоку зазвичай становить 64 біти. Проте з розвитком технологій та вдосконаленням деяких алгоритмів довжина одного блоку може бути більшою [28].

Серед найбільш поширених алгоритмів раніше зазначеного методу можна зокрема віднести наступні: DES; TDEA; RC5; RC6; AES. Розглянемо дані алгоритми криптографічного шифрування більш детально.

Перший криптографічний алгоритм, який ми дослідимо – це алгоритм «стандарт шифрування даних (або англ. data encryption standard), проте більш відома його скорочена назва DES. Цей алгоритм був розроблений у 1976 році і на сьогодні є найстарішим алгоритмом симетричного методу шифрування

даних. Даний алгоритм був розроблений для IBM і в ті роки використовувався здебільшого федеральним агентством США. Також цікавим фактом стосовно цього алгоритму є те, що саме цей алгоритм був одним із тих алгоритмів, які використовувались у ранніх версіях алгоритму TLS.

Вище зазначений алгоритм працює за наступною логікою – 64 бітні дані шифруються за 16 різними циклами шифрування, попередньо розділяючи 64 бітний блок на 2-ва 32 бітні і шифруючи кожен з них окремо. Серед циклів цього алгоритму зокрема можна виділити такі як: перестановка та заміна символів або інші подібні операції. В кінці цього процесу отримується 64 розрядний блок зашифрованих даних.

У 2005 році цей алгоритм шифрування був визнаний, як такий, що не відповідає сучасним потребам та стандартам. Серед основних причин, що послугувало для такого рішення став маленький розмір ключа, через що криптостійкість даного алгоритму дуже крихкою до криптоаналізу. Проте цей алгоритм на сьогодні дуже вдало замінює криптоалгоритм AES.

Логічним продовженням криптоалгоритму DES, став алгоритм AES, який був розроблений та почав застосовуватись, після затвердження у 2001 році в NIST (англ. The National Institute of Standards and Technology). Від самого початку даний алгоритм розроблювався як альтернатива DES, проте у 2005 році, як було сказано раніше, повністю його замінив.

Даний алгоритм працює по подібному до DES принципу, проте має деякі відмінності. Так алгоритм під час шифрування спирається на поділ даних по блоках та шифрування шляхом змішування та підстановки, проте на відміну від свого попередника в даному алгоритмі останній раунд перетворень відрізняється від усіх попередніх та являє собою процес мікшування вже зашифрованих даних.

Разом з цим даний алгоритм позбавлений недоліку свого попередника, а саме ключі, які використовує алгоритм значно більші ніж були у DES.

Наступний алгоритм симетричного шифрування є – TDEA (з англ. Triple Data Encryption Algorithm). По своїй суті це покращена версія DES, тому його ще називають 3DES. Вперше був опублікований у 1981 році, проте активно почав застосовуватися з кінці 90 років XX ст. У назві цього алгоритму дуже добре відображається суть його роботи, а саме даний алгоритм застосовує процеси кодування даних DES 3 рази, через що значно підвищується його криптостійкість.

Згідно з вимогами NIST, через виявлені у 2017 році значні вразливості у системі захисту цього алгоритму, розпочинаючи з 2023 року розробка програм на цьому алгоритмі не буде сертифікуватись NIST. До 2023 року даний алгоритм широко використовувався в електронних платіжних системах.

Серед основних алгоритмів асиметричного шифрування існує можливість виділити зокрема наступні: RSA та ECC. Розглянемо дані алгоритми більш детально.

Першим розглянемо алгоритм RSA. Назва даного алгоритму походить від прізвищ його розробників: R – Rivest (Ronald Linn Rivest – американець); S – Shamir (Adi Shamir – єврей); A – Adieman (Leonard Adieman - американець). Опис алгоритму був опублікований у 1977, а вже у 1983 році був виданий патент. Даний алгоритм можна поділити на 4 основні етапи за якими проходить процес шифрування:

- генерація ключа;
- шифрування;
- розшифрування;
- розповсюдження ключів для роботи з алгоритмом.

В основному даний алгоритм забезпечує безпеку даних спираючись у своєму шифруванні на складність принципу «факторизації цілих чисел».

Завдяки своїм характеристикам (безпеці та адаптивності) даний алгоритм є досить розповсюдженим, зокрема він використовується у різних додатках та таких сертифікатах, як SSL та TLS. Разом з цим алгоритм

застосовується і при шифруванні пошти і навіть використовується у тренді останнього десятиліття – криптовалюти.

Наступним алгоритмом асиметричного методу шифрування є раніше згаданий ЕСС (англ. Elliptic Curve Cryptography). Передумовою для створення даного криптографічного алгоритму стала запропонована у 1985 році ідея використання еліптичних кривих у криптографії. Ідея була запропонована двома американськими математиками (Victor S. Miller та Neal Koblitz). Проте їх ідеї використання еліптичних кривих отримали визнання, а також почали широко використовуватись починаючи з 2004 року.

Так шифрування проходить з використанням наступної формули (2.1) еліптичної кривої:

$$Y^2 = X^3 + AX + B, \quad (2.1)$$

де A і B – дійсні числа які задаються при шифруванні.

Алгоритм еліптичних кривих (ЕСС) працює по принципу незворотності. Тобто для шифрування даних береться випадкова точка на еліптичній кривій і за допомогою множення цієї точки на випадкове число в результаті отримується розташування іншої точки. Саме через цю особливість та математичну специфіку побудови даного методу знайти координати нової точки, навіть при умові, що є точні координати початкової точки, майже неможливо.

Таким чином даний метод криптографічного шифрування забезпечує один з найвищих рівнів захисту інформації. Цікавим фактом є те, що хоча в даному методі використовуються значно менші ключі для шифрування інформації, проте він забезпечує вищий рівень криптостійкості ніж RSA [29].

Разом з цим також є і гібридні алгоритми шифрування такі як SSL/TLS. Як і було сказано вище гібридний метод шифрування перевершує як симетричне, так і асиметричне, оскільки поєднує їх сильні сторони при цьому позбавляючись слабких. При даному методі шифрування, алгоритм

SSL/TLS встановлює зв'язок між клієнтами та сервером, шляхом перевірки ключів (відкритого та персонального (закритого)).

Після ідентифікації обох користувачів, шифрування відбувається за допомогою алгоритмів симетричного шифрування даних, що забезпечує велику швидкість передачі інформації. Сам процес шифрування відбувається за допомогою спеціально згенерованого сеансового ключа, який генерується при шифруванні інформації та передається з використанням асиметричних методів шифрування даних.

Висновки за розділом 2

У другому розділу було пророблено роботу з дослідження проблематики сучасної криптографії. Разом з цим були сформовані основні проблеми з якими стикнулася сучасна криптографія.

Наступним кроком були розглянуті основні вимоги до захисту інформації та її шифрування в сучасних умовах. Разом з цим було досліджено основні методи, за допомогою яких виконується шифрування інформації, а також були сформовані основні переваги та недоліки таких методів. Логічним продовженням дослідження, після дослідження проблематики та методів криптографічного шифрування, стало розгляд підходів та способів до криптографічного захисту інформації.

Після дослідження всього вище зазначеного логічним закінченням розділу, а також для більшого поглиблення у тему дипломної роботи, стало дослідження та розгляд криптографічних алгоритмів, за допомогою який проводиться шифрування даних.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ ТА НАЛАШТУВАННЯ СИСТЕМ КРИПТОГРАФІЇ

3.1 Стратегія та методи вибору розташування сервера

Обираючи операційну систему розробнику слід враховувати багато критеріїв для проведення досліджень та реалізації методів криптографії. Першочергово серверна операційна система відповідає за узгодження взаємодії програм, додатків, сервісів і мережевих рішень. Відмінною рисою серверної операційної системи є велика витрата обчислювальних ресурсів для виконання різноманітних операцій, порівняно з версією для звичайних користувачів. Її функціональність спрямована на обробку спеціалізованих завдань і забезпечення ефективної роботи в багатокористувацьких середовищах.

Для виконання серверних завдань використовуються три основні сімейства операційних систем: Microsoft Windows, Unix-подібні системи та Unix-сумісні операційні системи з відкритим вихідним кодом. З огляду на те, що кожна серверна операційна система має свої унікальні характеристики та функціональні особливості, детально розглянемо особливості кожного з цих сімейств.

Існує різноманіття типів та видів операційних систем, але зупинимося на трьох конкретних ОС: Ubuntu Server, FreeBSD та Microsoft Server. З цих варіацій ОС для сервера буде виступати за основу Ubuntu Server, обумовлене кількома ключовими факторами:

- Ubuntu Server є відкритою операційною системою, що дозволяє значні економії завдяки безкоштовній ліцензії;
- ця система відзначається високою стабільністю та отримує підтримку від активної та різнопланової спільноти користувачів;
- Ubuntu Server легко встановлюється, налаштовується та підтримується, що робить її зручним вибором для функціонування в якості вебсервера та хостингу вебсайтів.

При порівнянні Ubuntu Server з FreeBSD та Microsoft Server з погляду їх використання як вебсервера та хостингу вебсайтів можна виділити наступні технічні аспекти. FreeBSD, незважаючи на свою стабільність і надійність з розширеним функціоналом, обмежений у доступі до комерційного програмного забезпечення та має обмежену технічну підтримку. З іншого боку, Ubuntu Server, який є безкоштовним, високостабільним і користується широкою спільнотою з високоякісною документацією, є більш привабливим вибором для проєктів з обмеженими ресурсами. У порівнянні з Microsoft Server, Ubuntu Server відрізняється високою стабільністю, меншими вимогами до системних ресурсів і безкоштовністю, що робить його важливим вибором для проєктів з обмеженим бюджетом. В табл. 3.1 наведено зведені переваги між цими операційними системами.

Таблиця 3.1 – Порівняння основних аспектів ОС

Характеристики	Ubuntu Server	FreeBSD Server	Microsoft Windows Server
Тип ліцензії	поширюється під ліцензією GNU GPL (General Public License), вільний для комерційного використання	використовує ліцензію BSD (Berkeley Software Distribution), вільний для комерційного використання	поширюється під пропрієтарну (комерційну) ліцензію, що вимагає купівлі ліцензії для кожного сервера
Рівень безпеки	забезпечує високий рівень безпеки, особливо з регулярними оновленнями та детальною документацією з безпеки	славиться високим рівнем безпеки, надаючи цілісне ядро і можливості тонкого налаштування безпеки	вимагає додаткових зусиль із забезпечення безпеки та може бути схильний до атак, особливо в умовах активного використання

Продовження таблиці 3.1

Характеристики	Ubuntu Server	FreeBSD Server	Microsoft Windows Server
Підтримка спільноти	широка та активна спільнота, має велику документацію й активну технічну підтримку	активна, але відносно обмежена спільнота, що може позначитися на доступності ресурсів і підтримці	має велику спільноту, але можливий обмежений доступ до деяких ресурсів, платна підтримка
Управління ресурсами	пропонує ефективне використання ресурсів і простоту управління	забезпечує хорошу продуктивність, особливо за наявності певних навичок та тонкого налаштування ядра системи	вимагає великих системних ресурсів і може бути менш ефективним на малопотужних конфігураціях
Сумісність із вебтехнологіями	забезпечує ширшу сумісність із вебтехнологіями та має більше попередньо налаштованих варіантів для вебсерверів	добре підтримує вебтехнології, але може мати обмежену сумісність із деяким комерційним програмним забезпеченням	добре підтримує вебтехнології, особливо ті, які інтегровані з екосистемою Microsoft

З приведеної таблиці бачимо, що FreeBSD, Ubuntu Server та Microsoft Windows Server володіють власними унікальними перевагами та недоліками. FreeBSD визначається привабливістю завдяки безкоштовній ліцензії, високою продуктивністю та високим рівнем безпеки, але може стикається з обмеженістю ресурсів спільноти. Сервер Ubuntu вигідно відзначається безкоштовним використанням, ефективним управлінням ресурсами та активною спільнотою, що робить його зручним для різноманітних користувачів. З іншого боку, Microsoft Windows Server пропонує зручне

адміністрування та високий рівень сумісності з продуктами Microsoft, але вимагає значних фінансових витрат.

Після вибору операційної системи наступним кроком є визначення місця і методів розгортання серверної операційної системи. Існують два основних стратегічних підходи до забезпечення інфраструктури серверів: фізичне розташування і хостинг. Розглянемо деталі обох цих типів більш докладно.

Першим типом розглянемо фізичне розташування сервера, що включає встановлення апаратного забезпечення самостійно або в орендованому центрі обробки даних. Це може включати в себе власний корпоративний офіс або центр обробки даних із відповідним простором і інфраструктурою. Цей метод надає повний контроль над обладнанням, дозволяючи вибирати необхідне серверне обладнання, мережеві компоненти та системи зберігання. Однак він також супроводжується своїми особливостями, оскільки виникає зобов'язаність забезпечення приміщення електропостачанням, кондиціонуванням повітря та безпекою сервера. Це може забезпечувати високий рівень безпеки для сервера, але водночас супроводжується витратами на його утримання.

Другим типом розміщення, відомим як хмарне розміщення, ґрунтується на використанні інфраструктури хмарного постачальника, розташованої в їхньому дата-центрі. Ця інфраструктура включає в себе віртуалізовані ресурси та облікові записи, забезпечуючи тим самим гнучкість та масштабованість. Забезпечення доступу до цієї інфраструктури здійснюється через інтернет-з'єднання та мережеві сервіси. Недоліками можуть бути обмежений контроль над фізичним обладнанням та залежність від доступності інтернет-з'єднання.

Третім видом розміщення є віртуальний хостинг. Віртуальний хостинг представляє собою один із найбільш простих, доступних та розповсюджених методів отримання хостингу, при якому всі вебсайти клієнтів розміщуються на одному сервері. Його можна порівняти з гуртожитком чи хостелом, де

орендується ліжко-місце. Серед переваг цього підходу слід виділити доступні вартості та повну відповідальність провайдера за технічне адміністрування. Однак важливо врахувати обмежену потужність та відсутність можливості тонкого налаштування системи. В разі потреби у конкретному програмному забезпеченні чи модулях віртуальний хостинг може виявитися неефективним.

Є ще багато видів розміщення, але поки зупинимося на опису такої кількості варіацій. Проаналізувавши опис, наведений вище, зведемо всі переваги та недоліки в табл. 3.2.

Таблиця 3.2 – Порівняння основних аспектів ОС

Характеристика	Фізичне розміщення	Хмарне розміщення	Віртуальний хостинг
Контроль над обладнанням	Повний контроль	Обмежений контроль	Обмежений контроль
Обладнання та компоненти	Так	Обмежений	Немає
Вартість утримання	Високі витрати	Залежить від використання ресурсів	Низькі витрати
Відповідальність за інфраструктуру	Повна відповідальність	Ділена відповідальність	Повна відповідальність
Безпека	Високий рівень безпеки	Залежить від провайдера	Залежить від провайдера
Гнучкість та масштабованість	Обмежена	Велика	Обмежена
Залежність від доступності інтернету	Ні	Так	Так
Підтримка специфічних модулів	Так	Залежить від можливостей провайдера	Обмежена

Після проведення аналізу та вибору оптимального типу розміщення для серверів, приходимо до висновку, що найбільш підходящим варіантом є фізичне розміщення. Це рішення ґрунтується на технічних аспектах, а саме на високому рівні безпеки, який забезпечується повною контрольованістю над фізичним середовищем.

Важливим фактором вибору фізичного розміщення є його здатність забезпечити надійність роботи серверів та захист від різних зовнішніх загроз, таких як потенційний несанкціонований доступ. Маючи повний контроль над фізичною інфраструктурою, можна гарантувати безпеку та цілісність системи.

Незважаючи на те, що фізичне розміщення супроводжується витратами на утримання, його обрано через важливість максимального рівня безпеки для зберігання та обробки конфіденційної інформації. Це рішення враховує та балансує технічні та економічні аспекти, забезпечуючи оптимальну ефективність та високий рівень захисту даних.

Після ретельного аналізу вищезазначеної ситуації, прийнято рішення щодо розміщення сервера у віртуальному середовищі з метою емуляції фізичного розміщення сервера. Для реалізації цього вибору використовується програмний продукт VMware Workstation.

VMware Workstation представляє собою програмне забезпечення для віртуалізації, розроблене компанією VMware, і забезпечує розширені можливості в цій області. Основні функції включають можливість створення віртуальних машин з різними операційними системами, такими як Windows, Linux, macOS та інші. Це створює широкі можливості для тестування та розробки програмного забезпечення на різних платформах.

Крім того, VMware Workstation надає можливість спільного доступу до ресурсів, дозволяючи віртуальним машинам обмінюватися файлами та даними з фізичною машиною та між собою. Це сприяє полегшенню обміну інформацією між різними середовищами, роблячи процес розробки та тестування більш ефективним та зручним.

Додатково, VMware Workstation надає зручний інтерфейс для ініціації, запуску та керування віртуальними машинами. Це спрощує процес конфігурації та моніторингу віртуальних середовищ. Окрім того, ключовою можливістю є можливість створення точок відновлення (снапшотів) та клонування віртуальних машин. Ця функціональність дозволяє оперативно створювати копії та відновлювати систему до конкретних станів, що є особливо корисним для потреб розгортання та тестування різних конфігурацій.

3.2 SSH як спосіб автентифікації

Проаналізувавши процес встановлення операційної системи (ОС) сервера, який є фундаментом цього проекту, проаналізуємо методів захисту з'єднання та встановлення зв'язку з сервером через протокол SSH (Secure Shell). SSH – протокол захищеної оболонки. Це мережевий протокол, що забезпечує безпечний спосіб доступу до віддаленого комп'ютера незахищеною мережею. Він шифрує дані, надіслані мережею, і надає надійні механізми ідентифікації, гарантує, що тільки авторизовані користувачі можуть отримати доступ до віддаленого комп'ютера. SSH зазвичай використовується в додатках адміністрування сервера, передачі файлів [30].

На сервері було встановлено інструмент OpenSSH, який являє собою відкриту реалізацію протоколу SSH. Це забезпечує безпечний віддалений доступ та обмін файлами між комп'ютерами. Основні аспекти безпеки, які враховуються в OpenSSH, включають аутентифікацію на основі ключів та шифрування трафіку.

Користувач ініціює процес створення ключової пари, що включає приватний і публічний ключі. Приватний ключ залишається на локальному комп'ютері користувача, в той час як публічний ключ внесений до файлу `~/.ssh/authorized_keys` на сервері. Цей процес утворює безпечний механізм входу, при якому сервер виконує перевірку відповідності між приватним і

публічним ключами, а успішна аутентифікація забезпечує доступ до системи. Використання цього методу дозволяє здійснювати аутентифікацію без введення пароля, забезпечуючи високий рівень безпеки.

У контексті шифрування трафіку OpenSSH використовує криптографічні методи з метою захисту від перехоплення чи читання даних, які передаються між клієнтом і сервером. Застосування різноманітних алгоритмів для аутентифікації та шифрування, таких як RSA, DSA, ECDSA, AES, Blowfish, 3DES, гарантує високий рівень безпеки.

Іншими словами, SSH забезпечує безпеку віддаленого доступу, недопускаючи несанкціонованого використання та перехоплення інформації. Використання аутентифікації на основі ключів додає додатковий рівень безпеки, а шифрування трафіку гарантує конфіденційність під час передачі даних.

3.3 Вебсервер nginx та його роль у захисті сервера

При виборі вебсервера для розгортання нашого вебсайту, Apache та Nginx вибиваються у лідери як дві з найпопулярніших опцій. Обидва вони перебувають у стадії активного розвитку та мають великий користувацький базис, що використовується для реалізації різноманітних вебпроектів по всьому світу. Докладніше розглянемо кожний з цих вебсерверів та проведемо порівняльний аналіз їхніх переваг та недоліків.

Проаналізуємо кожний з цих вебсерверів, починаючи з Apache. З погляду технічних характеристик, Apache є класичним представником в світі вебсерверів і визначається як ключовий компонент стеку LAMP (Linux, Apache, MySQL, PHP). Його універсальність та висока надійність роблять його невід'ємною частиною багатьох вебпроектів.

Apache підтримує роботу на різних операційних системах, включаючи Linux та Windows, що робить його оптимальним вибором для широкого спектру проектів.

Однією з ключових характеристик Apache є його багатофункціональність. Вебсервер підтримує різні режими роботи, такі як процес, робітник та подія, що робить його високофункціональним та гнучким для різноманітних завдань. Завдяки зручному управлінню та широкому спектру функціональності, таких як балансування навантаження та кешування, Apache стає зручним інструментом для розробників та адміністраторів серверів.

Щодо аспектів безпеки, Apache використовує мову програмування C та різноманітні модулі, що забезпечують ефективний захист від різних видів атак, включаючи DDoS. Додаткові можливості, такі як використання модуля `mod_evasive`, розширюють спектр засобів захисту вебсервера.

У свою чергу, його прямиий конкурент – Nginx, хоч і з'явився на ринку пізніше, проте швидко здобув популярність завдяки винятковій продуктивності та сучасній архітектурі. Основною метою його розробки було вирішення проблеми C10k – здатність обслуговувати 10 000 одночасних підключень.

Вебсервер Nginx використовує асинхронну архітектуру, що дозволяє ефективно опрацьовувати велику кількість одночасних запитів. Його легкий та оптимізований код робить його ідеальним для сценаріїв, де продуктивність є ключовим чинником. При обслуговуванні статичного контенту Nginx демонструє швидкість у 2-3 рази вищу, ніж Apache, і при цьому використовує менше оперативної пам'яті. Крім того, він має чітко визначену систему підтримки, що надається розробницькою командою.

Nginx володіє потужними механізмами безпеки, що роблять його надійним вибором для розгортання вебсервера. Подібно до його конкурента Apache, у Nginx існує унікальний механізм захисту від DDoS-атак, відомий як Rate Limiting. Механізм Rate Limiting дозволяє обмежувати кількість запитів від одного клієнта за конкретний період часу, що сприяє запобіганню атак, спрямованих на перевантаження сервера, таких як DoS-атаки, DDoS-атаки тощо. Крім того, Nginx може використовуватися як вебфайер (WAF)

для фільтрації та блокування шкідливих запитів до вебзастосунків. Це є ключовим елементом з точки зору безпеки, оскільки забезпечує захист від різноманітних вразливостей.

Необхідно також враховувати технології SSL/TLS. Розгляд та порівняння реалізацій SSL/TLS у вебсерверах Nginx і Apache відкриває цікаві аспекти, пов'язані з безпекою обробки даних. Обидва сервери підтримують SSL/TLS, але вони відрізняються за реалізацією, продуктивністю та параметрами конфігурації.

Починаючи з версії 1.3.13, Nginx реалізувала підтримку останньої версії протоколу TLS – TLS 1.3, що забезпечує найшвидше і найбезпечніше встановлення захищеного з'єднання. Ця версія протоколу ефективно усуває уразливості попередніх версій і мінімізує затримки під час встановлення з'єднання.

В Apache підтримка TLS 1.3 була додана пізніше, починаючи з версії 2.4.36, і вимагає активації модуля `mod_ssl`. Обидва сервери дозволяють вибирати конкретні версії протоколу, шифри та хеш-функції, що надає адміністраторам гнучкість у налаштуванні рівнів безпеки.

Важливим аспектом є використання механізму Forward Secrecy (PFS), який гарантує, що навіть у випадку, якщо ключ шифрування було взято під контроль у минулому, це не становить загрози для безпеки поточних і майбутніх сеансів. Обидва вебсервери підтримують PFS, але, як правило, Nginx виявляє вищу продуктивність завдяки своїй асинхронній архітектурі.

Крім того, обидва вебсервери надають можливість конфігурування додаткових функцій, таких як зшивання OCSP (Online Certificate Status Protocol), яке підвищує безпеку перевірки статусу сертифіката, та SNI (Server Name Indication), що дозволяє віртуальним хостам використовувати різні SSL-сертифікати на одній IP-адресі.

Після завершення основного аналізу інформації, порівняннємо вебсервери Apache та Nginx, щоб визначити їхню поточну популярність та визначити переваги одного перед іншим. Розпочнемо з дослідження історії

популярності цих та інших вебсерверів. Нижче представлені графіки динаміки популярності по роках Як ілюстрованно на рис. 3.1 [31], Nginx вперше з'явився приблизно у 2007 році. На той момент Apache ще утримував лідерство з ринковою часткою 49%. Однак розробники Nginx активно працювали, постійно вдосконалюючи продукт та впроваджуючи інновації.

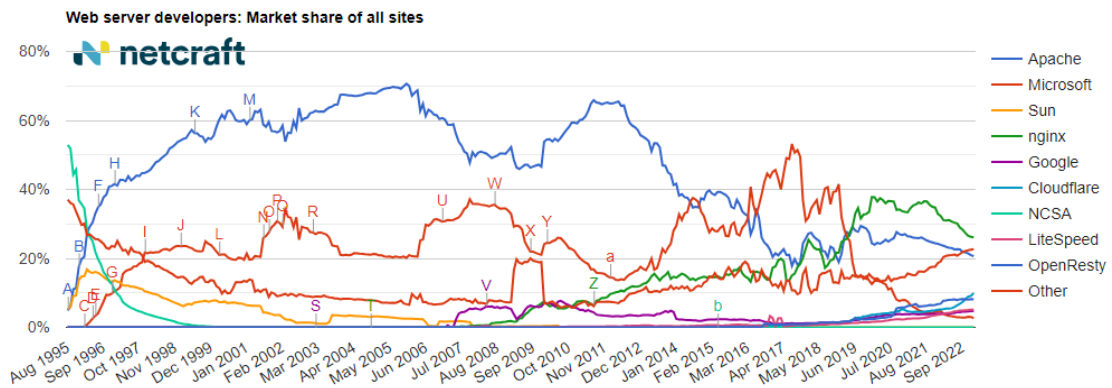


Рисунок 3.1 - Статистика від netcraft.com. Доля ринку всіх сайтів [31]

Ця стратегія привернула увагу численних користувачів, і невдовзі клієнтська база почала зростати. Компанії такі, як Sun, Google та Microsoft, які раніше володіли популярними вебсерверами, втратили свої позиції, і, нарешті, Nginx наздогнав їх.

Таблиця 3.2 - Статистика за грудень 2022 – січень 2023 [31]

Розробник	Грудень 2022	Відсотки	Січень 2023	Відсотки	Змінилося на
Nginx	295,366,783	26.25%	295,678,304	26.11%	-0.13
Apache	235,541,408	20.93%	233,636,177	20.63%	-0.30
Cloudeflare	102,829,746	9.14%	112,159,331	9.91%	0.77
OpenResty	92,711,293	8.24%	92,291,824	8.15%	-0.09

Лідерство Apache також втратило свою позицію до листопада 2018 року. Згідно з результатами інтернет-опитування, саме тоді Nginx вперше

посів перше місце. Таким чином, на сьогоднішній день обстановка в конкурентному боротьбі за популярність демонструє зміну динаміки:

- Nginx – 36%;
- Apache – 25%;
- Microsoft – 14%.

Але варто зазначити, що сервери Apache на даний момент ще містять більше активних вебсайтів, ніж Nginx. Проте, можна визначити тенденцію, яка свідчить про те, що це лише питання часу, коли все стане навпаки.

Проаналізуємо аспекти продуктивності вебсерверів Apache та Nginx у контексті завантаження статичного та динамічного контенту для отримання об'єктивної оцінки їхньої ефективності.

Уважно аналізуючи рейтинги та вивчаючи популярні вебсайти, стає видно, що існує принаймні одна загальна характеристика – швидкість завантаження. У сучасному світі швидкість вебсайту визначається як ключовий фактор, і кожна витрачена секунда на очікування призводить до втрати відвідувачів та зменшення обсягу продажів.

Nginx демонструє майже удвічі більшу швидкість у відношенні до статичного вмісту, порівняно з Apache при 512 одночасних підключеннях і у 2,4 рази ефективніше при збільшенні запитів. При цьому витрати на операційну пам'ять є меншими – Nginx використовує приблизно на 5-6% менше системної пам'яті для подолання навантаження. Це різниця головним чином обумовлена архітектурою сервера.

Хоча Nginx виходить переможцем у цьому порівнянні, слід відзначити, що високі показники продуктивності застосовуються в основному до статичних вебсайтів. У світі сучасних технологій більшість вебсайтів пропонує різноманітний динамічний вміст, і, отже, вебсервери повинні бути ефективно оптимізовані для обробки цього типу контенту.

У порівнянні зі статичним вмістом, Apache виявляється більш ефективним у роботі з динамічними вебсторінками. Проте, з належним налаштуванням Nginx може досягти аналогічної ефективності при обробці

динамічних вебсайтів. Результати тестів продуктивності не виявили значущої різниці між двома конкурентами, як у часі завантаження сторінок, так і в споживанні пам'яті.

Деякі ключовими показниками за своєю суттю є, архітектура, орієнтована на події, у Nginx виявляється більш продуктивною порівняно з архітектурою Apache, що базується на процесах, особливо в періоди максимального навантаження та при тривалих роботах. Це підтверджується декількома ключовими критеріями.

Наприклад, команда LiteSpeed використала реалізації протоколів HTTP/2 та HTTP/1.1, взяті від серверів Apache та Nginx, для проведення порівняльного аналізу їх продуктивності при завантаженні WordPress. У випадку використання HTTP/2, Nginx здатний обробляти понад 6000 запитів на секунду, тоді як Apache в змозі обслуговувати лише 826. Щодо швидкості передачі даних, Nginx демонструє показник 24,5 МБ/с, у порівнянні з 3,08 МБ/с у випадку Apache [32].

Результати тестування HTTP/1.1 продемонстрували аналогічну тенденцію. Nginx обслуговував 5730 запитів за секунду, тоді як Apache відповідав лише на 1000 запитів за секунду. Nginx завантажував дані зі швидкістю 24,15 МБ/с, тоді як Apache працював із тактовою частотою 4,09 МБ/с [32].

Підсумовуючи, можна стверджувати з впевненістю, що Nginx, як вебсервер, проявляє перевагу у 2-3 рази при обробці статичного контенту порівняно з Apache. Однак, при завантаженні динамічних вебсайтів, різниця в продуктивності між Nginx та Apache не є суттєвою. Це робить обидві платформи конкурентоспроможними при обробці динамічного контенту.

Навіть незважаючи на свою відносно новизну, Nginx вже зайняв ключові позиції на ринку вебсерверів завдяки своїй винятковій ефективності та оптимізованій структурі коду, що надає йому перевагу в обробці запитів та забезпечує вражаючий рівень продуктивності. З іншого боку, Apache успішно підтримує офіційні екземпляри операційних систем Linux та

Windows, що дозволяє йому охоплювати широкий спектр операційних систем. В той час як Apache бездоганно працює з Windows, Nginx надає лише часткову підтримку для цієї операційної системи, і це важливо враховувати при виборі вебсервера залежно від платформи.

Обидві платформи, Nginx та Apache, володіють обширним арсеналом корисних модулів, які дозволяють розробнику виконувати широкий спектр налаштувань і оптимізувати їхню роботу. В Інтернеті можна знайти безліч навчальних ресурсів, які присвячені використанню обох платформ, а також можна отримати підтримку від досвідчених фахівців, що забезпечує зручність та допомогу при роботі з вебсерверами. Таким чином, вибір між Nginx та Apache залежить від конкретних вимог проєкту, потреб у продуктивності та переваг операційної системи.

3.4 Мова програмування для вебсайту

Після аналізу та вибору вебсервера Nginx, ключовим чинником у розробці вебсайту залишається вибір мови програмування. З урахуванням вибору операційної системи Ubuntu Server, яка базується на ядрі Linux, мова програмування C# та його фреймворк ASP.NET виявляються непридатними за обраною конфігурацією. Навіть якщо є можливість розгорнути вебсайт за допомогою ASP.NET, це не є оптимальним рішенням, оскільки багато елементів і компонентів логічніше та ефективніше розгортати на сервері Microsoft Server, оскільки більшість функцій оптимізовані саме для цієї операційної системи. Таким чином, альтернативними варіантами розглядатимуться Python та PHP.

Почнемо аналіз з огляду на мову програмування PHP. PHP – це мова програмування, яка масово використовується у веброботці. Вона проста у вивченні, має зручний та зрозумілий синтаксис, спирається на підтримку величезного ком'юніті розробників [33]. Зазвичай PHP використовується для розробки вебпродуктів, і на його основі створюють вебсайти та програми.

RНР дозволяє ефективно розробляти професійні програми та вебсайти, зменшуючи необхідний час для цього. Ця мова постійно оновлюється і, незважаючи на наявність та популярність інших мов, продовжує займати провідне місце у списку найбільш важливих інструментів.

Більш детально проаналізуємо мову програмування Python. Python – це мова програмування, яка широко використовується в вебдодатках, розробці програмного забезпечення, науці даних і машинному навчанні (ML). Розробники використовують Python, тому що він ефективний, простий у вивченні та працює на різних платформах. Програми мовою Python можна завантажити безкоштовно, вони сумісні з усіма типами систем та підвищують швидкість розробки [34].

Python був вперше представлений в 1991 році голландським програмістом Гвідо Ван Россумом і зазнав значних змін упродовж свого розвитку. У 2000 році була випущена версія 2.0, а в 2008 році — версія 3.0. Незважаючи на суттєві відмінності між цими версіями, регулярно видають підтримуючі версії. На момент написання цих слів актуальною є версія 3.12, яка була випущена у жовтні 2023 року.

Основними характеристиками мови програмування Python є те, що вона є скриптовою мовою програмування, де код програм визначається у вигляді скриптів. Мова підтримує різні парадигми програмування, включаючи об'єктно-орієнтовану та функціональну парадигми. Особливість полягає в тому, що програми на Python інтерпретуються. Для виконання скриптів необхідний інтерпретатор, який запускає та виконує код програми.

Проаналізувавши основні особливості цих двох мов програмування, проаналізуємо переваг та недоліків цих мов. Важливо врахувати, що Python і RНР – це дві різні мови програмування з власними унікальними перевагами та недоліками.

RНР славиться своєю високою продуктивністю, широкою підтримкою для різних платформ та великою численністю ком'юніті. Розробницька

спільнота PHP активно використовує безліч інструментів і фреймворків, які забезпечують сумісність з різними операційними системами і системами управління базами даних. Однак PHP може зіткнутися з проблемами синтаксису, що може викликати труднощі під час розробки та ускладнювати виявлення помилок. Поважність програм, розроблених на PHP, також може бути піддана впливу через його репутацію.

З іншого боку, Python відзначається простотою, низькою трудомісткістю та обширним функціоналом завдяки різноманітним бібліотекам та фреймворкам. Його кросплатформенність та легко засвоюваний синтаксис роблять його особливо привабливим для розробників. Тим не менше, порівняно з PHP, Python може виявитися менш продуктивним і вимагати більше зусиль для досягнення високої продуктивності.

У обох випадках (PHP та Python) забезпечення ефективного захисту від вразливостей передбачає уважне ставлення розробників до безпеки, систематичне оновлення та використання оптимальних практик забезпечення.

Тим не менше, PHP може виявитися уразливим до атак, таких як атаки на введення та вилучення даних, XSS (Cross-Site Scripting) та SQL-ін'єкції, якщо розробник не дотримується належних заходів безпеки. У той час як у спільноті Python приділяється значна увага безпеці, розробники докладають великих зусиль для дотримання передових практик безпеки, таких як правильне управління ідентифікаторами сеансів та уникнення потенційно небезпечних функцій. Це ускладнює процес вторгнення в вебсайти, побудовані на Python.

Отже, провівши аналіз вищезазначеного з метою створення більш безпечного проєкту, рекомендованим вибором є мова програмування Python. Мову програмування Python можна використовувати - від веброзробки до наукових досліджень та штучного інтелекту. Python дозволяє ефективно розв'язувати широкий спектр завдань. Також простий та читабельний

синтаксис Python забезпечує легкість розробки та сприяє швидкому розвитку проєкту.

Завдяки великій кількості бібліотек та фреймворків, які додають багатофункціональності, Python полегшує роботу та дозволяє швидко впроваджувати нові функції. Важливим фактором є також активна та численна спільнота розробників Python, яка сприяє обміну досвідом та надає швидку підтримку. Це є критичним аспектом для вирішення проблем та забезпечення стабільності проєкту.

Обираючи між PHP та Python, важливо враховувати, що обидві мови програмування мають свої особливості та обмеження. Після ретельного аналізу та урахування вимог конкретного проєкту вибір між PHP та Python стає завданням, що вимагає уважності до кожної деталі. Кожна з цих мов програмування володіє своїми перевагами та обмеженнями, але на підставі детального вивчення специфікацій та характеристик цього проєкту було вирішено віддати перевагу мові програмування Python.

3.5 Опис та методи вибору WSGI-сервера

Проаналізувавши та обравши мову програмування Python, важливим фактором є аналіз стандарту WSGI. Проте перед цим варто ретельно розібратися, що представляє собою цей стандарт та як він може впливати на безпеку сервера та вебзастосунку.

WSGI (Web Server Gateway Interface) є стандартною специфікацією Python, яка визначає загальний інтерфейс для взаємодії програми або фреймворку з сервером програм або вебсервером. Основною метою WSGI є спрощення та стандартизація взаємодії між цими компонентами, що сприяє однорідності та ефективності у розробці вебзастосунків.

WSGI в собі не несе безпосереднього впливу на безпеку сервера або вебсайту, оскільки його головне призначення полягає в встановленні стандартного інтерфейсу для взаємодії між вебсерверами та додатками

Python. Проте використання WSGI може впливати на безпеку, залежно від того, як буде розгорнено та налаштовано вебсервер та додатки. Використання цього стандарту як додаткового інструмента може сприяти у забезпеченні безпеки нашого вебсайту.

Після детального розгляду основних аспектів стандарту WSGI, настав час обрати реалізацію розробника програмного забезпечення, яка оптимально реалізує цей стандарт і забезпечить найвищу продуктивність та підтримку безпеки для вебсервера. Для цієї ролі були визначені дві реалізації серверів додатків: uWSGI та Gunicorn.

uWSGI – це вебсервер і фреймворк для обробки вебзапитів та запуску вебдодатків у середовищі мови програмування Python. Як і очікується від назви, uWSGI відповідає стандарту WSGI, що дозволяє використовувати різноманітні вебфреймворки та додатки, які відповідають цьому стандарту. Крім того, uWSGI підтримує різні протоколи для обміну даними між вебсервером та додатком, такі як HTTP, uwsgi (власний протокол uWSGI), FastCGI, SCGI та інші.

У своєму функціоналі, uWSGI надає можливість використовувати асинхронні функції для оптимізації обробки багатьох одночасних з'єднань та запитів. Це дозволяє досягти ефективності в умовах високого навантаження та значної кількості одночасних клієнтів. Крім того, uWSGI вміло управляє процесами додатків, реалізуючи балансування навантаження та автоматичне масштабування, що підвищує його популярність в середовищі високозавантажених додатків.

Gunicorn (Green Unicorn) – це WSGI-сервер, призначений для запуску вебдодатків, розроблених на основі Python. Він вирізняється своєю простотою використання та високою ефективністю. Gunicorn відповідає стандарту WSGI, що робить його сумісним з різними вебфреймворками та додатками Python, такими як Flask, Django, Pyramid та інші.

Однією з вагомих переваг Gunicorn є висока продуктивність сервера, особливо у поєднанні з асинхронними фреймворками та додатками,

такими як FastAPI чи Tornado. Цей серверний додаток може працювати з параметрами за замовчуванням, але також надає низку опцій конфігурації для налаштування параметрів, таких як кількість робочих процесів, порт прослуховування, рівень журналізації тощо.

Зокрема, Gunicorn забезпечує підтримку SSL-шифрування, що гарантує безпечний обмін даними між сервером та клієнтом. Цей функціонал разом із іншими можливостями робить Gunicorn високо популярним серед розробників Python для розгортання вебдодатків на сервері.

Під час аналізу цих двох серверів-додатків проводилось тестування їх продуктивності, в результаті чого була виявлена перевага в швидкості на боці Gunicorn, яка ілюструється на графіках, представлених на рис. 3.2 [35].

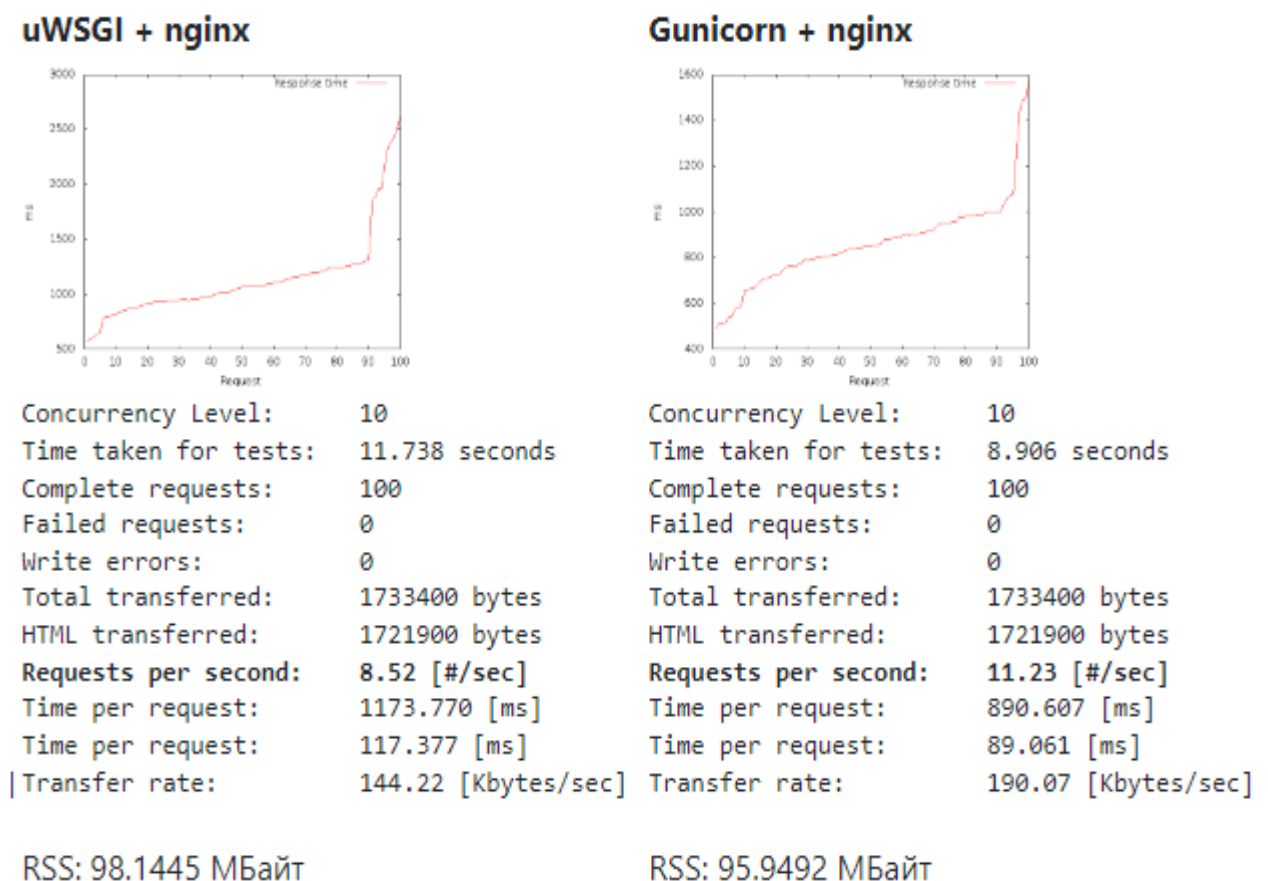


Рисунок 3.2 – Тест продуктивності uWSGI та Gunicorn у зв'язці з Nginx [35]

Підбиваючи підсумки, обидва сервери, Gunicorn і uWSGI, є потужними інструментами для розгортання вебдодатків, що базуються на

мові програмування Python. Проте при розгляді їхніх характеристик виявляються певні переваги Gunicorn. Висока ефективність, особливо при використанні асинхронних фреймворків, визначається як одна з ключових переваг Gunicorn.

Також важливою характеристикою є можливість безпечної передачі даних за допомогою SSL-шифрування, що робить його найбільш бажаним в контексті цього дослідження.

Висновки за розділом 3

Підводячи підсумки усього вище зазначеного, можна відзначити, що вибір компонентів системи може суттєво впливати на швидкодію та безпеку обробки даних. У цьому розділі було проведено аналіз та розглянуто характеристики кожного компонента системи.

Починаючи з вибору операційної системи для сервера, ми виявили, що кожна система має свої переваги та обмеження. Після порівняння FreeBSD, Microsoft Server та Ubuntu Server можна визначити, що для поставлених завдань найбільш оптимальним варіантом є Ubuntu Server. Ця операційна система дозволяє швидко та надійно налаштовувати сервер, ефективно опрацьовувати великі обсяги даних та легко адмініструвати систему.

Далі було досліджено метод з'єднання та шифрування запитів за допомогою SSH, який забезпечує максимальний рівень безпеки завдяки використанню пар ключів. Ми також проаналізували вебсервери та їхні можливості у забезпеченні безпеки системи та обрали Nginx як вебсервер для подальшого використання у проєкті.

В цьому розділі також було обрано оптимальну та популярну мову програмування Python для розгортання вебсайту та можливого розширення вебдодатків. Детально розглянули технологію WSGI та її роль у з'єднанні різноманітних вебдодатків.

4 РІШЕННЯ ПРИ РОЗРОБЦІ КРИПТОГРАФІЧНОГО ЗАХИСТУ САЙТУ, ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ

4.1 Налаштування ОС сервера та загальної архітектури

Ubuntu Server є варіантом операційної системи Linux, спеціально призначеним для використання на серверах. У цьому розділі детально розглянуто, як налаштування сервера може впливати на загальний рівень захисту системи. Особливу увагу приділено ролі брандмауера та методам захисту передачі даних через протоколи SSL/TLS.

Початковим етапом та основним типом конфігурації для забезпечення безпеки сервера є налаштування брандмауера UFW для Ubuntu Server.

4.1.1 Налаштування Firewall UFW

Почнемо з детального розгляду UFW. UFW (Uncomplicated Firewall або «простий брандмауер») є інтерфейс iptables, призначений для спрощення процесу налаштування брандмауера.

Хоча iptables – надійний і гнучкий інструмент, початківцям буває складно навчитися використовувати його для правильного налаштування брандмауера [36]-[37].

Спочатку виконаємо завантаження брандмауера для подальшого налаштування – `sudo apt-get install ufw`.

При переході до налаштувань, на початковому етапі необхідно завантажити політику за замовчуванням. Ці правила визначають обробку мережевого трафіку, який не відповідає жодним іншим попередньо встановленим правилам.

Це означає, що інші користувачі не матимуть можливості отримати доступ до нашого хмарного сервера, але всі програми, встановлені на сервері, зможуть взаємодіяти з зовнішніми програмами.

Для виправлення цього необхідно внести зміни у ці параметри:

- `sudo ufw default deny incoming;`
- `sudo ufw default allow outgoing.`

Заборона вхідного трафіку за замовчуванням забезпечує додатковий рівень безпеки, оскільки ми визначаємо, які підключення дозволені.

Також перш ніж увімкнути UFW, важливо надати дозвіл для SSH, щоб мати можливість підключення до сервера через цей протокол – `sudo ufw allow ssh.`

Необхідно також надати доступ до портів 22 або 2222, проте це буде розглянуто більш докладно в контексті SSH.

Основні конфігурації UFW вже здійснені. Будь-які інші налаштування слід додавати за необхідності для забезпечення надійного рівня захисту.

4.1.2 Завантаження сертифікатів SSL

Однією з ключових складових системного та вебзахисту є використання протоколів SSL/TLS, які забезпечують безпеку та шифрування передачі даних між сервером та клієнтом. В умовах сучасного інтернет-середовища, де акцент робиться на забезпеченні конфіденційності та цілісності інформації, важливість протоколів SSL/TLS є надзвичайно високою.

Отже, наступним кроком буде завантаження сертифікатів та аналіз деталей шифрування трафіку за допомогою протоколу SSL/TLS. Для цього необхідно встановити Certbot, введенням наступних команд:

- `sudo apt update;`
- `sudo apt install certbot.`

Після успішної інсталяції, наступним етапом є отримання та встановлення сертифіката та приватного ключа SSL. Для автоматизованого процесу отримання та встановлення сертифіката Certbot використовується за допомогою наступної команди – `sudo certbot --nginx -d sipsavorte.com.`

Після чого в конфігурацію Nginx буде додано шляхи до нашого сертифіката.

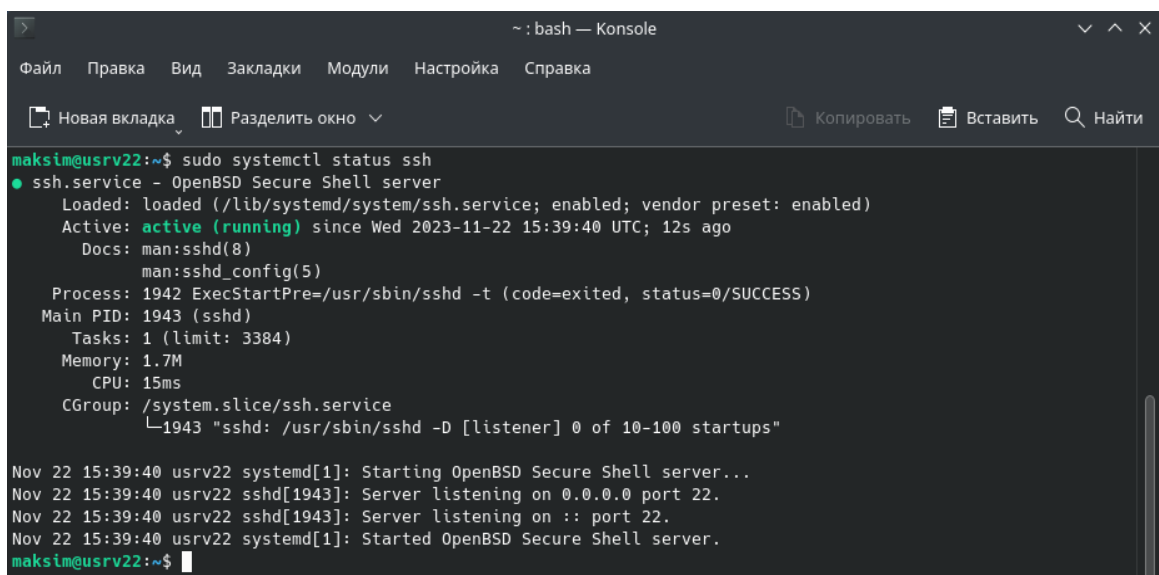
4.2 Конфігурація SSH та розбір етапів роботи протоколу

Приступаючи до розгляду параметрів налаштування протоколу SSH та його реалізації OpenSSH, починаємо із встановлення OpenSSH на сервер для подальшої конфігурації:

- `sudo apt update;`
- `sudo apt install openssh-server.`

Після успішного інсталювання OpenSSH автоматично запуститься. Перевіряємо, чи дійсно він запустився командою – `sudo systemctl status ssh`.

Якщо немає помилок, то у терміналі виведе інформацію стосовно цього процесу як зазначено на рис. 4.1.



```

~ : bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
[+] Новая вкладка  [ ] Разделить окно  [ ] Копировать  [ ] Вставить  [ ] Найти
maksim@usrv22:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2023-11-22 15:39:40 UTC; 12s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
  Process: 1942 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
 Main PID: 1943 (sshd)
    Tasks: 1 (limit: 3384)
   Memory: 1.7M
      CPU: 15ms
   CGroup: /system.slice/ssh.service
           └─1943 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Nov 22 15:39:40 usrv22 systemd[1]: Starting OpenBSD Secure Shell server...
Nov 22 15:39:40 usrv22 sshd[1943]: Server listening on 0.0.0.0 port 22.
Nov 22 15:39:40 usrv22 sshd[1943]: Server listening on :: port 22.
Nov 22 15:39:40 usrv22 systemd[1]: Started OpenBSD Secure Shell server.
maksim@usrv22:~$

```

Рисунок 4.1 – Інформація стосовно запущеного SSH-сервера

Після перевірки коректності роботи всіх компонентів переходимо до налаштування SSH-сервера. Основним файлом конфігурації є `~/.etc/ssh/sshd_config`, в якому розташовані ключові параметри SSH. Цей файл має велику кількість параметрів, які можна налаштовувати відповідно до

різних потреб сервера. Однак у цьому випадку буде проаналізована основні аспекти, які слід враховувати для підвищення безпеки.

Нам треба змінити у файлі `sshd_config` такі параметри, як:

а) модифікуємо номер порту з типового 22 на 2222 або 4444 відповідно до потреб проєкту. Змінюємо параметр `Port` наступним чином змінюємо на `Port 2222`.

Таким чином це сприяє покращенню безпеки, оскільки зломиснику стане складніше визначити, який саме порт ми встановили, ускладнюючи його поточну діяльність;

б) далі, важливим етапом є відключення можливості входу під користувачем "root", використання аутентифікації за допомогою пароля та, за необхідності, вимкнення методів виклику та відповіді для аутентифікації:

- 1) `PermitRootLogin no`;
- 2) `PasswordAuthentication yes`;
- 3) `ChallengeResponseAuthentication no`;
- 4) `AllowUsers username1 usernameN`;

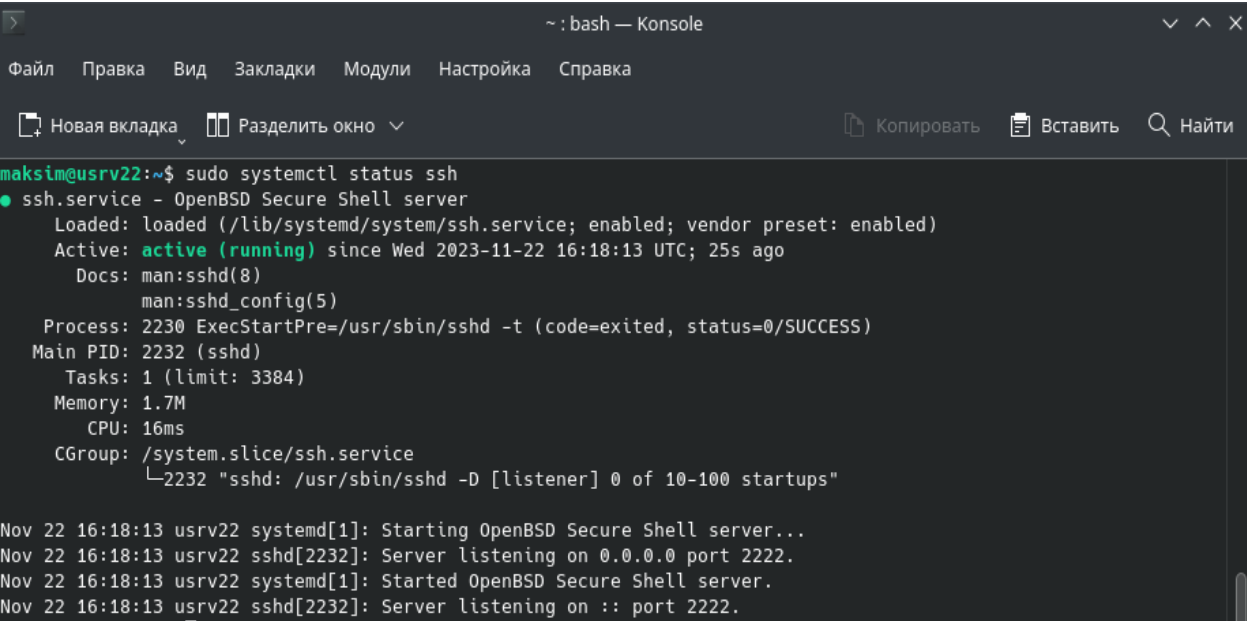
Таким чином було встановлюно заборону на вхід через обліковий запис "root", що забезпечить захист від несанкціонованого доступу по SSH під обліковим записом "root".

Крім того, за необхідності, можна встановити обмеження на підключення тільки під певним користувачем за допомогою параметра `AllowUsers`;

в) якщо на сервері використовується графічний інтерфейс користувача, активуємо опцію пересилання дисплеїв програми X клієнту, що дозволяє здійснювати підключення через X клієнти – `X11Forwarding yes`.

Після завершення усіх налаштувань, додаємо дозвіл на використання порту у фаєрволі за допомогою команди `sudo ufw allow 2222`.

Після цього перезапускаємо процес SSH-сервера та перевіряємо, чи застосовані всі зміни, використовуючи команду `sudo service ssh restart`, що представлено на рис. 4.2.



```

~ : bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти

maksim@usrv22:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2023-11-22 16:18:13 UTC; 25s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
  Process: 2230 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
 Main PID: 2232 (sshd)
    Tasks: 1 (limit: 3384)
   Memory: 1.7M
      CPU: 16ms
   CGroup: /system.slice/ssh.service
           └─2232 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Nov 22 16:18:13 usrv22 systemd[1]: Starting OpenBSD Secure Shell server...
Nov 22 16:18:13 usrv22 sshd[2232]: Server listening on 0.0.0.0 port 2222.
Nov 22 16:18:13 usrv22 systemd[1]: Started OpenBSD Secure Shell server.
Nov 22 16:18:13 usrv22 sshd[2232]: Server listening on :: port 2222.

```

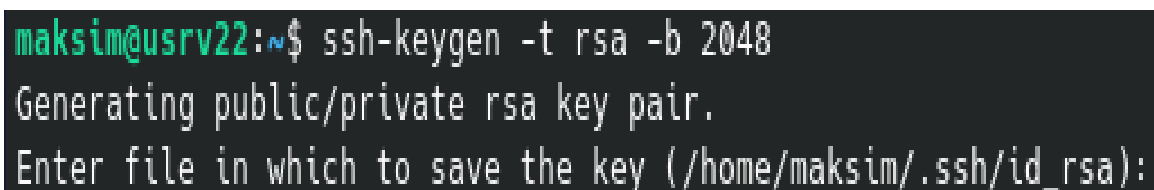
Рисунок 4.2 – Активність процесу та зміна порта на 2222

Процес успішно функціонує, і порт був змінений з 22 на 2222. Далі необхідно згенерувати зовнішній та приватний ключі для вказаного користувача – `ssh-keygen -t rsa -b 2048`.

Для параметра `-b` можна вказати значення 4096, що вказує на довжину ключа у бітах. Використання значення 4096 дозволяє отримати більш захищений ключ для системи.

Це важливо, особливо при генерації ключів для криптографічних застосувань, де велика довжина ключа сприяє високому рівню безпеки переданих даних.

Після виконання програми, розробника буде запитано вказати шлях до розташування ключів, проте залишаємо цей шлях без змін, використовуючи значення за замовчуванням (`/home/maksim/.ssh/id_rsa`), що можна побачити детально на рис. 4.3.



```

maksim@usrv22:~$ ssh-keygen -t rsa -b 2048
Generating public/private rsa key pair.
Enter file in which to save the key (/home/maksim/.ssh/id_rsa):

```

Рисунок 4.3 – Вибір директиви, де буде знаходитися ключі

Наступним етапом буде отримання повідомлення від системи, підтверджуючого, що процес генерації публічного та приватного ключів завершено успішно, що представлено на рис. 4.4.

```
Your identification has been saved in /home/maksim/.ssh/id_rsa
Your public key has been saved in /home/maksim/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:FZ0hXoipSNRp+r1J0w5K4MD7j68F7TM/45qUkP/ZYa4 maksim@usrv22
The key's randomart image is:
+---[RSA 2048]-----+
|    . . . . 00000    |
|    . + 0..+0       |
|    . . + . 0        |
|    0 +0. . .        |
|    ++0..S.          |
|    . . =0.= .       |
|    . .0o *o         |
|    . +.*==..        |
|    o+=oE+o          |
+-----[SHA256]-----+
```

Рисунок 4.4 – Сповіщення та надання інформації про згенеровані ключі

Крім того, обов'язково зберігаємо згенерований публічний ключ користувача на сервері за допомогою команди `ssh-copy-id -p 2222 maksim@usrv22`

Оскільки був змінений порт, обов'язково вказуємо його явно, оскільки без параметра `-p 2222` запит буде намагатися відправити через порт за замовченням – 22, що представлено на рис. 4.5.

```
maksim@usrv22:~$ ssh-copy-id -p 2222 maksim@usrv22
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/maksim/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys

Number of key(s) added: 1

Now try logging into the machine, with:  "ssh -p '2222' 'maksim@usrv22'"
and check to make sure that only the key(s) you wanted were added.
```

Рисунок 4.5 – Вдале завантаження публічного ключа користувача

На даний момент SSH-сервер налаштований та активний, проте розглянемо деталі з'єднання, що встановлюється між клієнтом та сервером на рівні протоколу SSH. Під час спроби підключення клієнта до сервера відбувається обмін ключами для забезпечення безпеки комунікації.

Початковий етап включає встановлення TCP-з'єднання між клієнтом SSH (користувачем, який намагається підключитися) та SSH-сервером. Зазвичай для SSH використовується порт 22, але в цьому випадку ми переназначили його на 2222.

Перед взаємодією даними, клієнт та сервер обмінюються параметрами з'єднання, такими як версія протоколу та підтримувані методи автентифікації. Цей етап є важливим для встановлення коректного та безпечного з'єднання між обома сторонами.

Процес аутентифікації є критичним етапом у взаємодії між клієнтом та сервером у контексті протоколу SSH. Стандартні методи автентифікації включають в себе використання паролів та відкритих ключів. При використанні ключів, клієнт надає свій відкритий ключ, який сервер перевіряє за вмістом файлу `authorized_keys`.

Після успішної аутентифікації, SSH налагоджує симетричні ключі для шифрування даних, що передаються між клієнтом та сервером. Ці ключі генеруються випадковим чином та використовуються лише протягом поточної сесії.

Обидва пристрої використовують єдиний ключ для кодування і декодування повідомлень під час їхнього пересилання. У цій стратегії для кожної зустрічі по SSH використовується особливий ключ. Цей вид безпеки використовується для кодування всієї асоціації, щоб убезпечити її від будь-яких атак в організації.

Нікому не вдасться перехопити це корисне навантаження, і в будь-якому випадку, коли вони це зроблять, розшифрування буде немислимим, оскільки для цього призначений звичайний ключ, дана схема представлена на рис. 4.6 [38].

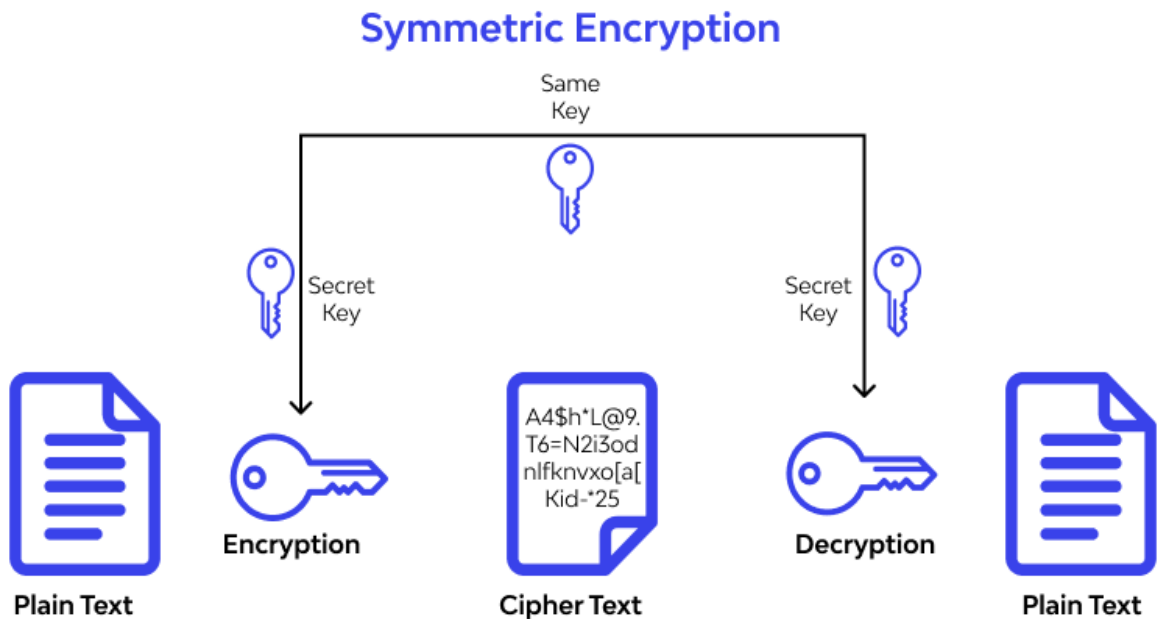


Рисунок 4.6 – Схема симетричного шифрування SSH [38]

Крім того, SSH створює зашифровані тунелі для безпечного передавання даних різних мережесих протоколів. У випадку переадресації портів пакети даних, які призначені для конкретної IP-адреси та порту на одному комп'ютері, автоматично переадресовуються на вказану IP-адресу та порт на іншому комп'ютері. Це механізм дозволяє забезпечити високий рівень безпеки при передачі даних різних мережесих протоколів, таких як протоколи для доступу до баз даних або віддаленого доступу до графічного інтерфейсу.

4.3 Засоби захисту через вебсервер Nginx

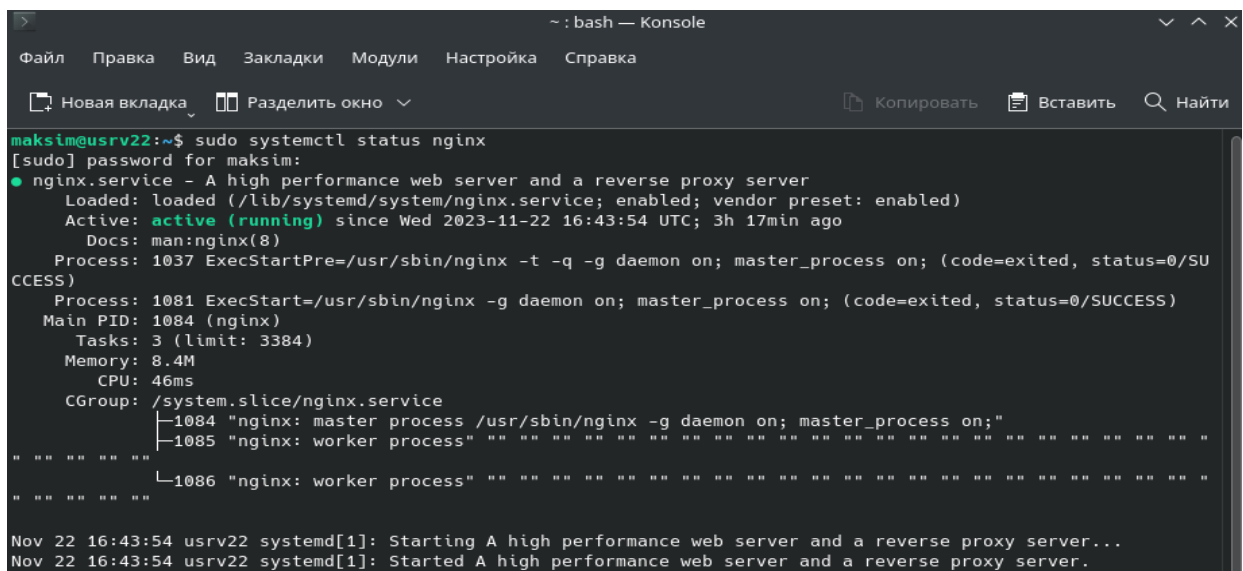
Nginx – потужний інструмент для обробки запитів в мережі, який використовується як вебсервер та балансувальник навантаження. Він відзначається високою ефективністю та надійністю, вирізняючись особливо швидким та легким ядром, що дозволяє оптимально обробляти трафік та забезпечувати стабільну роботу великої кількості одночасних підключень.

Безумовно, Nginx є ключовим елементом сучасної інфраструктури вебдодатків і надає широкий спектр можливостей для конфігурації та оптимізації серверних застосунків. З цієї причини розглянемо його заходи забезпечення від несанкціонованого доступу та докладні налаштування.

При інсталяції вебсервера Nginx використовується стандартна команда для оновлення пакетів у поточній сесії Linux, а також для безпосереднього завантаження Nginx:

- `sudo apt update;`
- `sudo apt install nginx.`

Після успішного завантаження вебсервера перевіряємо його стан для визначення його працездатності, що представлено на рис 4.7.



```

~ : bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти

maksim@usrv22:~$ sudo systemctl status nginx
[sudo] password for maksim:
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2023-11-22 16:43:54 UTC; 3h 17min ago
     Docs: man:nginx(8)
   Process: 1037 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
   Process: 1081 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
    Main PID: 1084 (nginx)
       Tasks: 3 (limit: 3384)
      Memory: 8.4M
         CPU: 46ms
    CGroup: /system.slice/nginx.service
            └─1084 "nginx: master process /usr/sbin/nginx -g daemon on; master_process on;"
               └─1085 "nginx: worker process"
                  └─1086 "nginx: worker process"

Nov 22 16:43:54 usrv22 systemd[1]: Starting A high performance web server and a reverse proxy server...
Nov 22 16:43:54 usrv22 systemd[1]: Started A high performance web server and a reverse proxy server.

```

Рисунок 4.7 – Первісний огляд працездатності процесу

Переконавшись в коректному функціонуванні процесу, переходимо до налаштування Nginx. Перше завдання - створити конфігураційний файл у директоріях `~/etc/nginx/sites-available/` та `~/etc/nginx/sites-enabled/`. У цьому випадку, ці файли отримають назву `sipsavorte`. Конфігуруємо їх відповідно до стандартів для протоколів HTTP та HTTPS:

```

server {
    listen 443 ssl http2;

```

```

listen 80;

server_name shop.sipsavortea.shp;

ssl_certificate /etc/nginx/ssl/cert.pem;
ssl_certificate_key /etc/nginx/ssl/key.pem;

ssl_session_cache shared:SSL:10m;
ssl_session_timeout 10m;
keepalive_timeout 70;
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers on;
ssl_stapling on;
location = /favicon.ico { access_log off; log_not_found off; }
location /static/ {
    root /home/maksim/AllSites/EcommerceSite/SipAndSavor/;
}

location / {
    include proxy_params;
    proxy_pass http://unix:/run/gunicorn.sock;
}
}

```

З даної конфігурації можна визначити, що в разі неможливості клієнта підключитися за протоколом HTTPS (за портом 443), його автоматично перенаправлять на протокол HTTP (за портом 80). Крім того, зауважимо, що доменне ім'я нашого сайту - shop.sipsavortea.shp, яке вже налаштоване на DNS-сервері. Додатково було сконфігуровано роботу сайту з попередньо завантаженими сертифікатами SSL для захисту передачі даних між клієнтом та сервером:

- `ssl_certificate /etc/nginx/ssl/cert.pem` та `ssl_certificate_key /etc/nginx/ssl/key.pem` - шляхи до SSL-сертифіката та його ключа, необхідних для забезпечення шифрування та безпечної передачі даних HTTPS;
- `ssl_session_cache shared:SSL:10m` - кеш сесій SSL для підвищення продуктивності;
- `ssl_session_timeout 10m` - час збереження SSL-сесії у кеші;
- `keepalive_timeout 70` – тайм-аут з'єднання `keepalive`;
- `ssl_protocols TLSv1.2 TLSv1.3` – вказує, які версії протоколу TLS підтримуватиме вебсервер;
- `ssl_prefer_server_ciphers on` - перевага серверних шифрів під час встановлення SSL-з'єднання;
- `ssl_stapling on` - включає SSL-запит до сервера для підтвердження статусу сертифіката.

Після внесення налаштувань, розміщуємо налаштований файл в директорію `~/etc/nginx/sites-enabled/`. Зміни, внесені в один файл, автоматично відображаються в іншому, забезпечуючи синхронізацію між ними. Це полегшує адміністрування та внесення змін у цих файлах.

Наступним етапом буду налаштування головного файлу `~/etc/nginx/nginx.conf`, де буде сконфігуровано заходи захисту від атак типу «відмова в обслуговуванні» (DoS) та «розподіленої атаки «відмова в обслуговуванні» (DDoS).

Основним елементом в захисті від атак DoS є обмеження на кількість запитів від кожного клієнта, що стає актуальним у випадку DDoS-атак, коли існують "зомбі-комп'ютери". Детальні налаштування наведено в додатку А:

- `limit_req_zone $binary_remote_addr zone=req_limit:10m rate=1r/s` – змінна `$binary_remote_addr` містить бінарне представлення віддаленої IP-адреси клієнта. `zone=req_limit:10m` створює область з ім'ям `req_limit` та розміром 10 мегабайтів для відстеження інформації про запити. `rate=1r/s` встановлюється максимальна швидкість запитів з однієї IP-адреси в 1 запит на секунду, при перевищенні цього значення буду затримувати запити

– `limit_conn_zone $binary_remote_addr zone=conn_limit:10m` –
`$binary_remote_addr` також використовується для ідентифікації унікальних IP-адрес клієнтів. `zone=conn_limit:10m` створює область з ім'ям `conn_limit` та розміром 10 мегабайтів, призначену для відстеження з'єднань від кожної унікальної IP-адреси.

Після внесення усіх модифікацій необхідно перезапустити Nginx за допомогою команди `sudo service nginx restart`. Далі слід перевірити конфігурацію на наявність помилок, використовуючи команду `sudo nginx -t`, що наочно представлено на рис. 4.8.

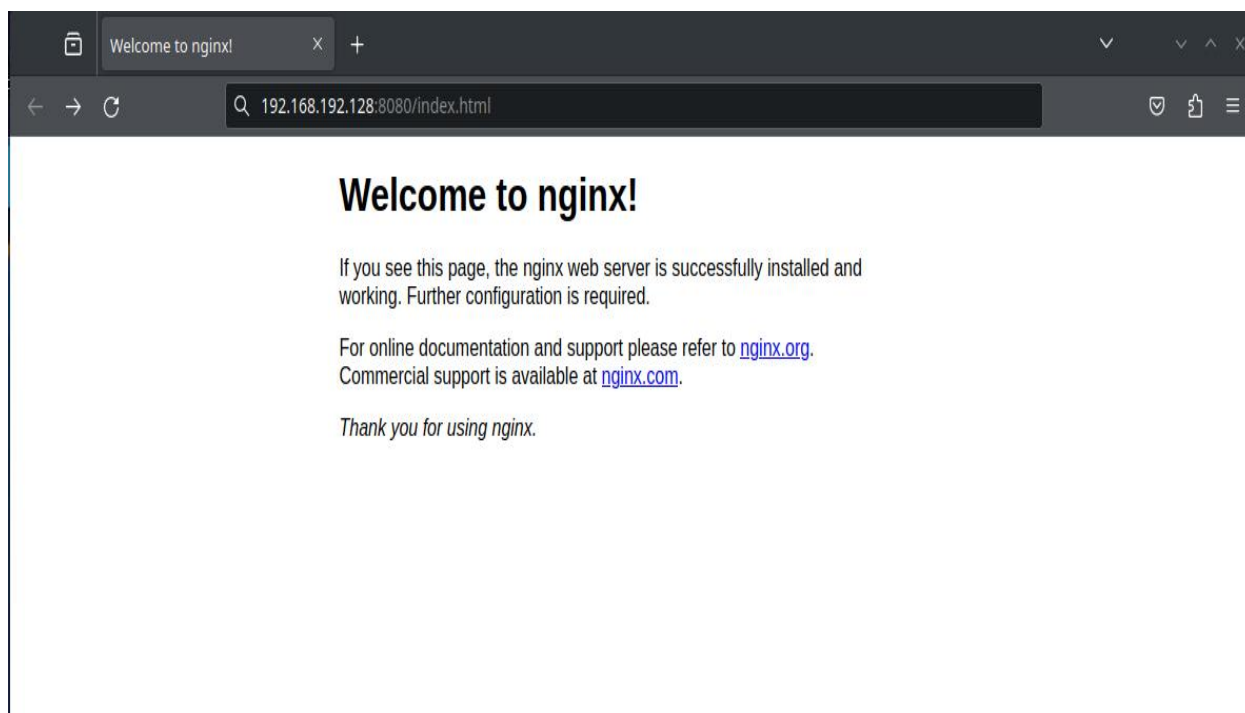


Рисунок 4.8 – Початкова сторінка вебсервера Nginx

Як бачимо, що система працює безперебійно і відображає основну сторінку Nginx рис. 4.8.

Ключовою частиною нашого проєкту є розробка вебдодатків, які будуть виступати у ролі інтерфейсу користувача.

Ці вебдодатки відіграють критичну роль у взаємодії між користувачем і системою, забезпечуючи зручний та ефективний доступ до функціоналу проєкту.

4.4 Фреймворк Django – структура проєкту та методи захисту аутентифікації

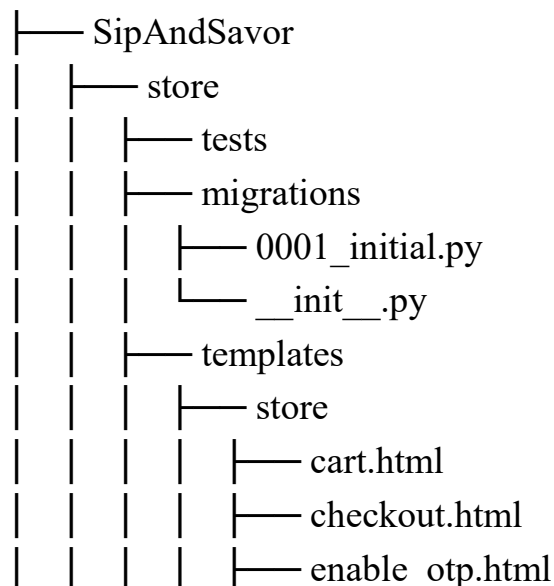
Головним компонентом системи буде вебдодаток, реалізований на базі фреймворку програмування Python, а саме Django. Django є високорівневим вебфреймворком, розробленим мовою програмування Python, який спрощує і прискорює процес створення вебзастосунків. Він надає зручний механізм для розробки вебзастосунків, забезпечуючи високий рівень безпеки та масштабованості.

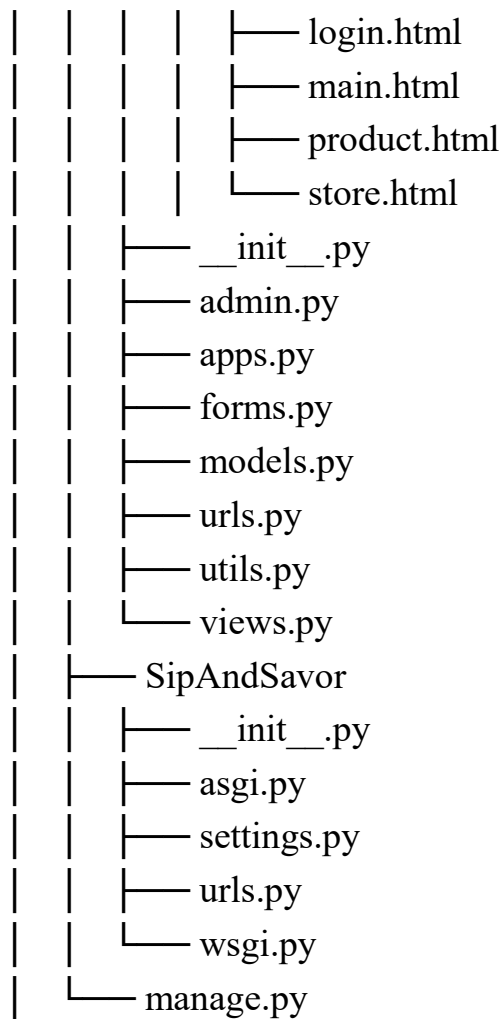
Однією з характерних особливостей фреймворку Django є модифікована реалізація шаблону проєктування Model-View-Controller, відома як Model-View-Template (MVT).

Моделі відповідають за взаємодію з базою даних, представлення відповідають за відображення інформації, а контролери (в Django вони відомі як "види" - views) виконують обробку логіки взаємодії.

У рамках цього проєкту був створений комерційний вебсайт, що включає головну сторінку з каталогом товарів, окремі сторінки для кожного товару, функціонал кошика, де можна переглянути загальну суму, можливість оформлення замовлення товарів, а також функціонал логування та реєстрації користувачів.

Детальніше розглянемо структуру цього проєкту:





В компоненті представлень (views) реалізована основна логіка навігації та взаємодії користувача з вебсайтом. Докладний код, який стосується компоненту представлень у моєму проєкті, надано в додатку Б.

У цьому контексті звернемо увагу на аспекти забезпечення безпеки вебсайту. Для цього використаємо двоетапну аутентифікацію, що дозволяє підвищити рівень безпеки вхідних даних та перешкоджає несанкціонованим доступам.

4.4.1 Огляд функціоналу вебсайта

Проаналізуємо функціонал вебсайту, що фактично є комерційною онлайн-платформою для замовлення різноманітних сортів листових чаїв. Специфікація функціоналу детально визначає можливості та сервіси, доступні користувачам.

Важливим елементом є інтуїтивно зрозумілий та зручний інтерфейс, який допомагає користувачам швидко зорієнтуватися на сайті та легко здійснювати замовлення. На рис. 4.9 наведено головну сторінку разом із списком товарів, які можна замовити на вказаному вебсайті.

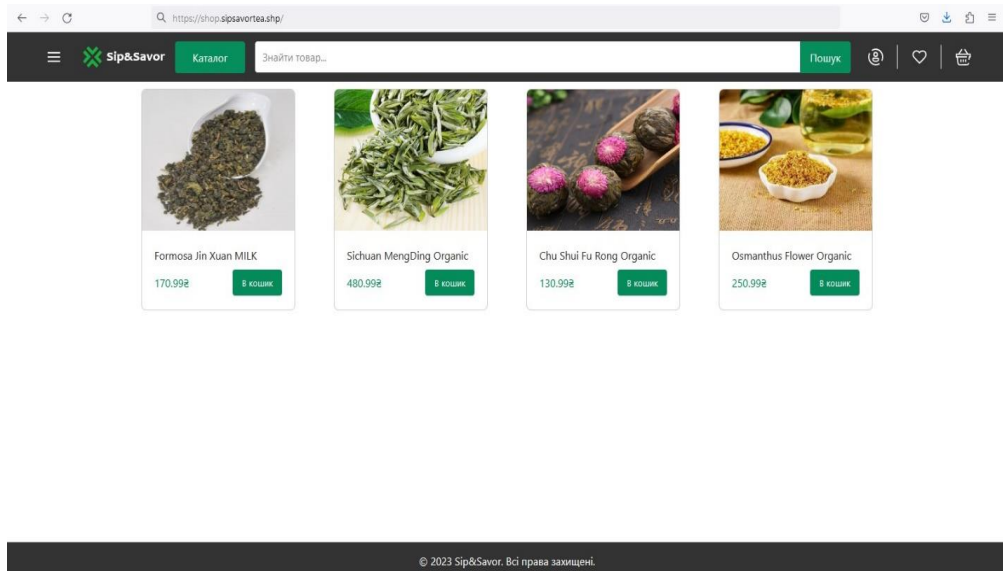


Рисунок 4.9 – Головна сторінка сайту та товари для замовлення

Після взаємодії з кнопкою «В кошик», обраний товар додається до кошика, і індикатор кількості товарів у кошику автоматично збільшується на одиницю. Це сприяє покупцеві в орієнтації та відображенні кількості товарів, що знаходяться у його кошику, більш наочно це можна побачити на рис. 4.10.

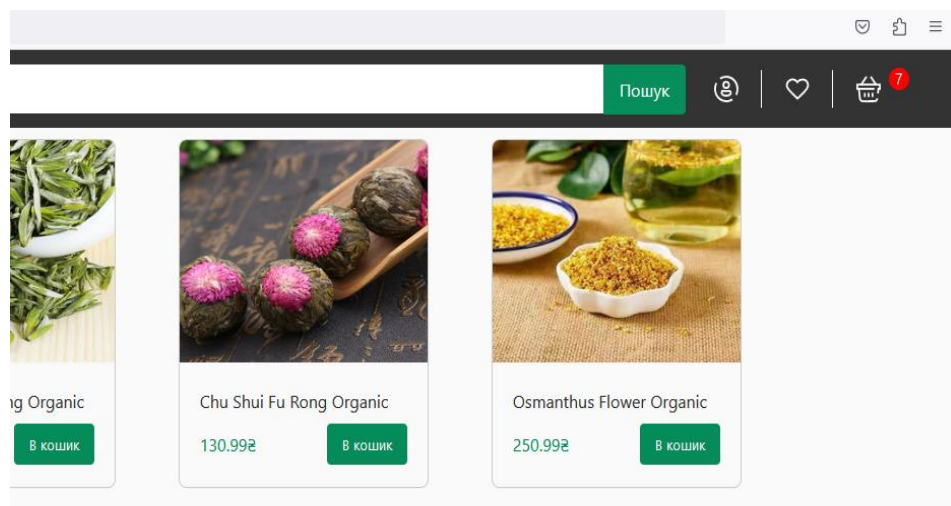


Рисунок 4.10 – Лічильник кількості товарів у кошику

Після додавання усіх цікавлячих користувача товарів важливо перейти до кошика для проведення докладного огляду та, за необхідності, редагування товарів, які вже знаходяться в кошику, що більш наочно представлено на рис. 4.11.

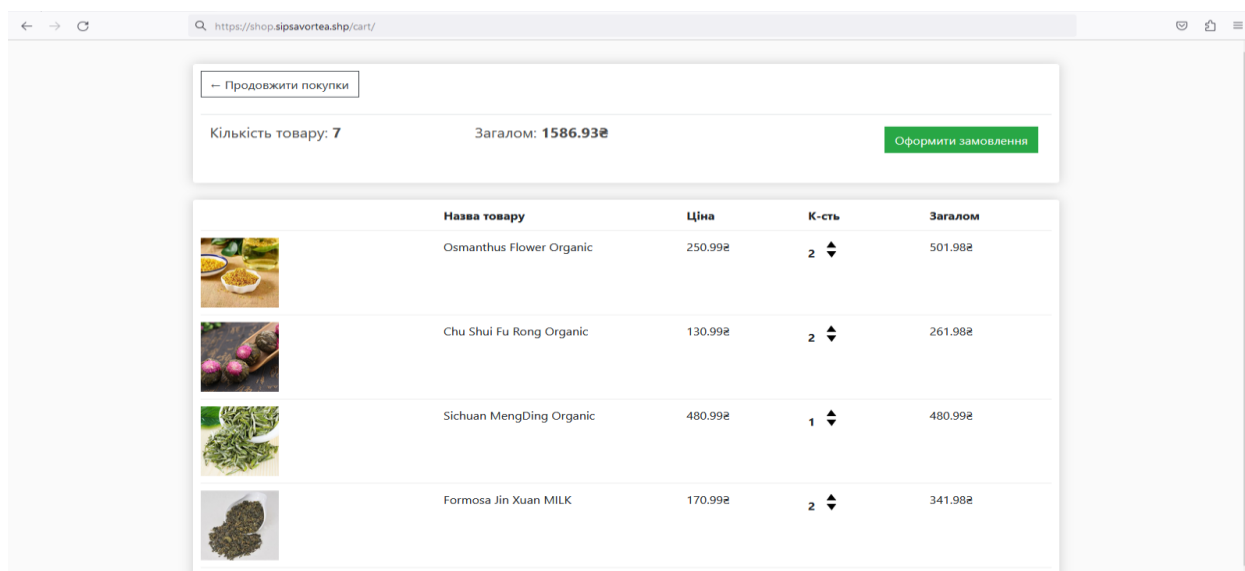


Рисунок 4.11 – Сторінка з товарами у кошику

На сторінці кошика користувач може отримати комплексну інформацію щодо замовлення, включаючи наступне:

- товари – усі товари представлені у вигляді списку з коротким описом кожного;
- кількість товару – користувач може редагувати кількість товарів за допомогою стрілочок;
- загальну суму до сплати – розраховує загальну суму, що полегшує користувачеві оцінку вартості замовлення.

Враховуючи потреби користувача, він має можливість повернутися на сторінку замовлення товарів, використовуючи кнопки «Оформити замовлення» та «Продовжити покупки».

У випадку вибору продовження оформлення замовлення, після натискання відповідної кнопки, користувач буде переадресований на сторінку заповнення форми замовлення, що представлено на рис. 4.12.

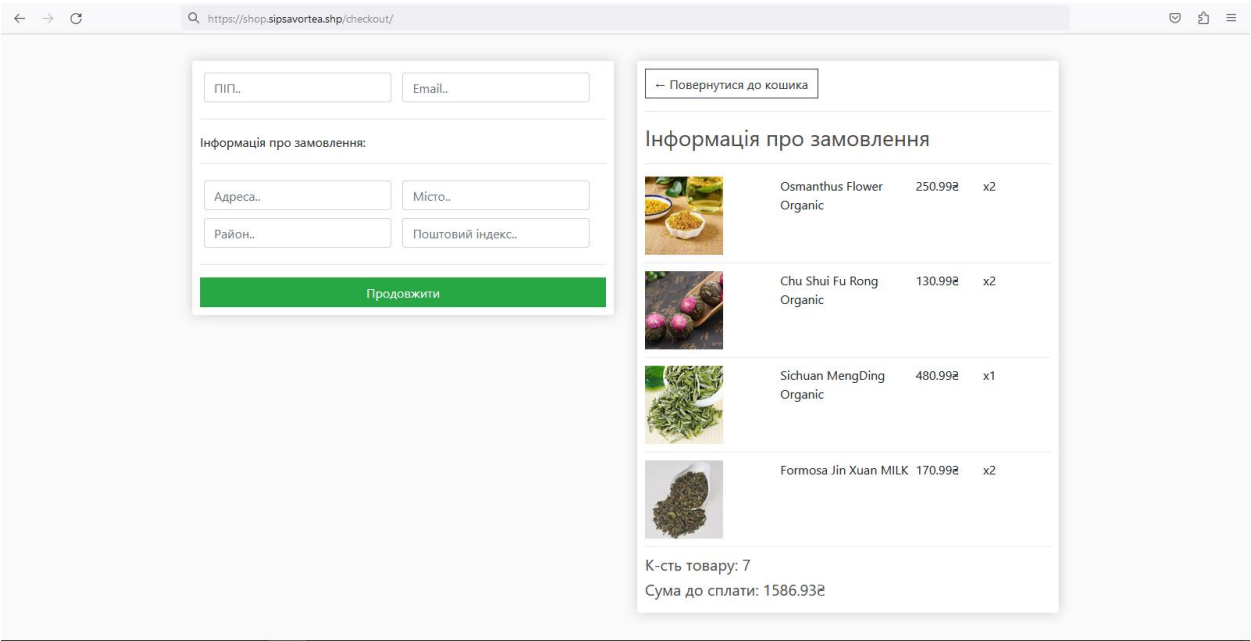


Рисунок 4.12 – Сторінка оформлення товарів та заповнення адреси доставки

На даній сторінці користувач буде представлений полями для введення адреси доставки. Блок із полями для ПІБ та електронної пошти відображатиметься лише у випадку, якщо користувач не авторизований у системі. У випадку авторизації користувача цей блок не буде відображено як на рис. 4.13.

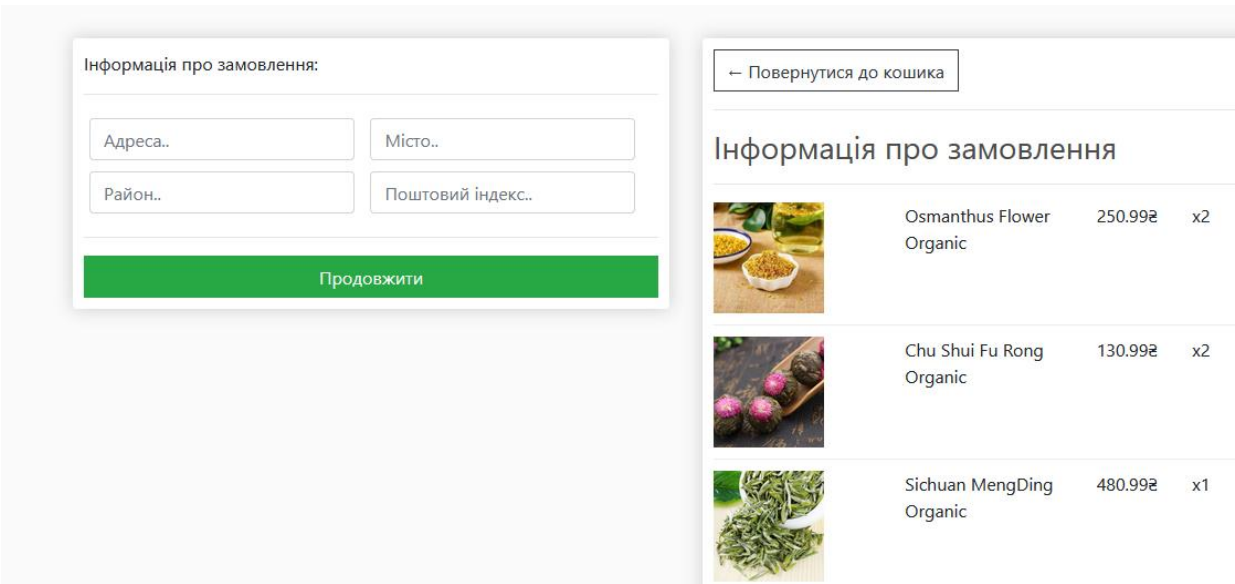


Рисунок 4.13 – Вигляд сторінки оформлення замовлення для авторизованого користувача

У розділі «Інформація про доставку» форми користувачу необхідно вказати наступну інформацію:

- місто – вказати місто, куди буде відправлено замовлення;
- район – вказати район, куди буде відправлено замовлення;
- адресу – вказати адресу, куди буде відправлено замовлення;
- поштовий індекс – вказати поштовий індекс, за яким буде відправлено замовлення.

Після введення вищезазначеної інформації користувач має натискати кнопку "Продовжити", щоб перейти до етапу оплати товарів. Наприклад, оплата може здійснюватися за допомогою платіжного сервісу PayPal.

Проаналізувавши основну частину функціоналу вебсайту, перейдемо до докладного опису методів аутентифікації користувача та засобів захисту облікового запису.

4.4.2 Розробка та впровадження метода двоетапної аутентифікації OTP

Один із ключових механізмів забезпечення безпеки користувачів – це система Двоетапної аутентифікації (2FA). Зазвичай користувачі використовують слабкі паролі, які можуть бути швидко взломані зловмисниками.

У цьому випадку надійність та захист облікового запису користувача забезпечує система двоетапної аутентифікація, особливо використовуючи OTP (One-Time Password) – одноразовий пароль.

Основний механізм передбачає, що користувач реєструється в системі та активує двофакторну аутентифікацію, що генерується через методи `enable_otp` та `otp_qr_code`. Метод `enable_otp` генерує код із самостійним записом його в базу даних. Після чого, користувач натискає кнопку "Згенерувати QR Code", яку він сканує у додатку Authenticator, що відображається на рис. 4.14.

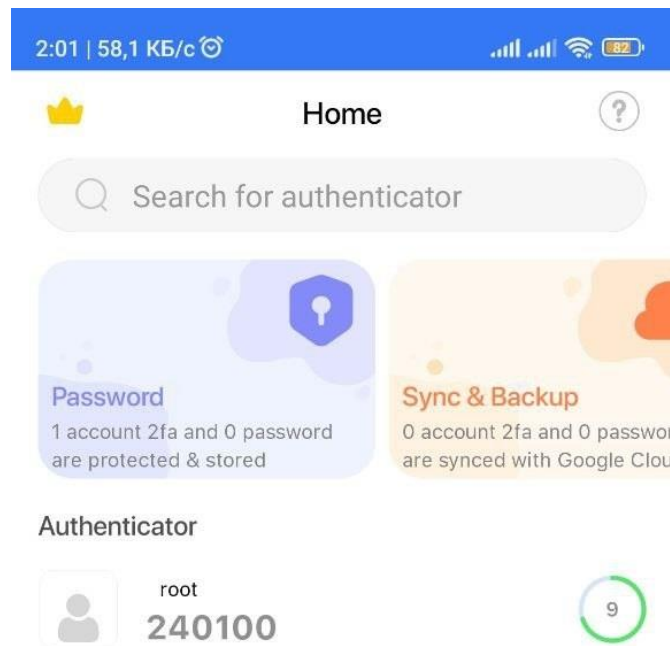


Рисунок 4.14 – Застосунок Authenticator у моб. додатку

У даному прикладі була реалізована двоетапна аутентифікація, використовуючи бібліотеки `django_otp`, `qrcode` та стандартні бібліотеки Django. Починаючи роботу, завантажуюмо необхідні бібліотеки для подальшого використання – `sudo pip install django_otp qrcode Pillow`.

Після завантаження бібліотек переходимо до конфігурації файлу `settings.py`. Вносимо зміни у наступні поля:

```
INSTALLED_APPS = [
    ...
    'django_otp',
    'django_otp.plugins.otp_totp',
    'django_otp.plugins.otp_hotp',
    'django_otp.plugins.otp_static',
    ...
]

...

MIDDLEWARE = [
    ...
```

```
'django_otp.middleware.OTPMiddleware',
...
]
```

Імпортуємо бібліотеки в `views.py` та `models.py`:

```
– from django.contrib.auth import authenticate;
– login from django.contrib.auth.decorators import login_required;
– from django_otp.plugins.otp_totp.models import TOTPDevice.
```

Після виконання основних налаштувань переходимо до ключової частини розробки методу OTP. Отже, потрібно створити класи-відображення для генерації та відображення QR-коду, який можна буде сканувати у додатку Authenticator.

Для цього розробимо представлення `otp_qr_code` та `enable_otp` з декораторами `@login_required`, що дозволяє відвідувати дану сторінку лише авторизованим користувачам.

Перший крок - активація та генерація TOTP-пристрою для параметрів генерації OTP-коду. Цей прилад генерує унікальний ключ-код, який ми подальше конвертуємо в QR-код за допомогою методу `otp_qr_code`. Розглянемо деталі самого методу `enable_otp`:

```
@login_required
def enable_otp(request):
    user_profile, created =
Customer.objects.get_or_create(user=request.user)
    data = cartData(request)
    if not user_profile.otp_device:
        otp_device = TOTPDevice.objects.create(user=user_profile.user,
confirmed=False)
        user_profile.otp_device = otp_device
        user_profile.save()
        cartItems = data['cartItems']
        order = data['order']
```

```

items = data['items']
products = Product.objects.all()
context = {'products': products, 'cartItems': cartItems, 'otp_device':
user_profile.otp_device}

return render(request, 'store/enable_otp.html', context).

```

З цього фрагменту коду бачимо, що спочатку ми намагаємося отримати існуючого користувача або створити нового. Після цього відбувається перевірка – якщо у користувача немає OTP-коду, то він генерується та записується у відповідну змінну `user_profile.otp_device`.

В кінцевому результаті, при реєстрації користувача в системі і заповненні всіх необхідних полів рис. 4.15, він автоматично переадресовується на сторінку генерації OTP-коду рис. 4.16. На цій сторінці, при натисканні кнопки «Відобразити QR-код», користувачу виводиться вікно з QR-кодом, який можна сканувати у додатку рис. 4.17.

Рисунок 4.15 – Форма реєстрації користувача у системі

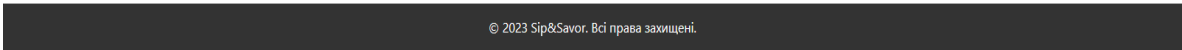
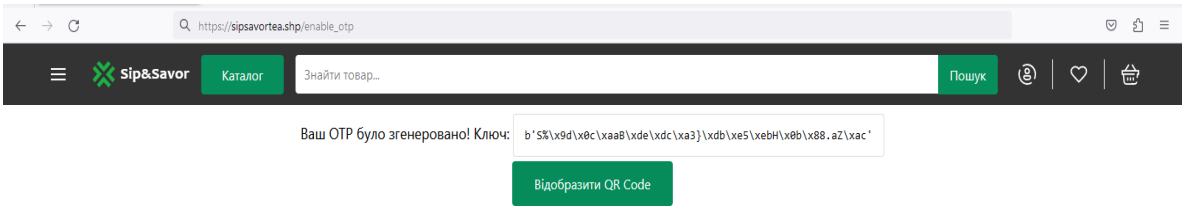


Рисунок 4.16 – Відображення згенерованого ключа та перехід до QR-коду

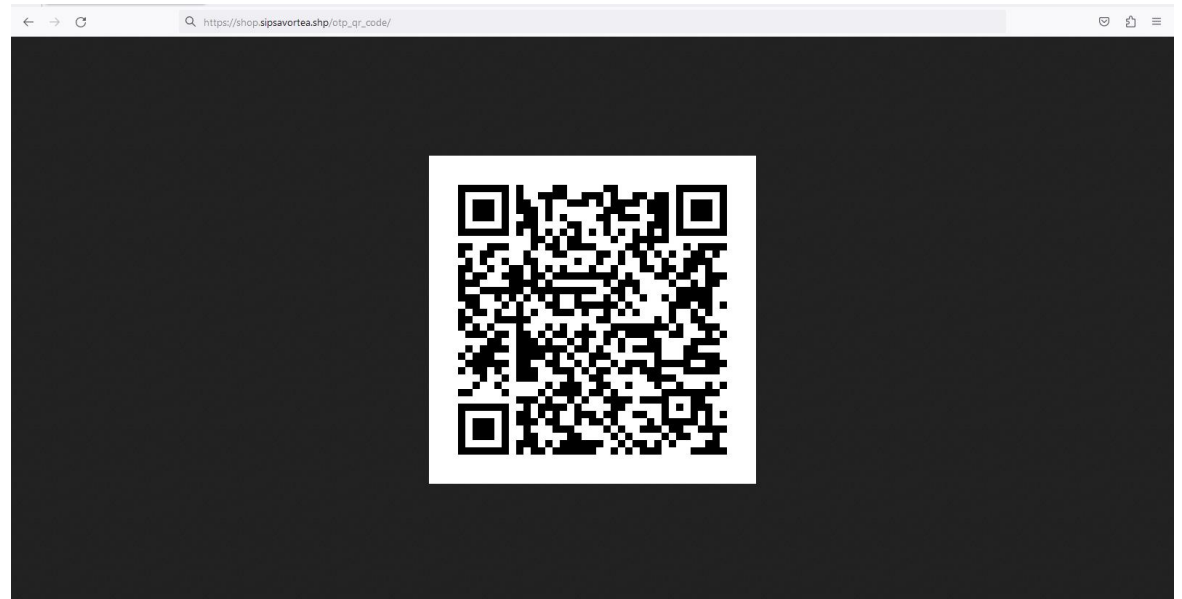


Рисунок 4.17 – Сторінка відображення згенерованого QR-коду

Після успішної активації OTP, згенерований одноразовий пароль може бути використаний для автентифікації та входу у облікову систему. Користувач вводить згенерований OTP-код, який тимчасово діє і надає доступ до облікового запису, забезпечуючи додатковий рівень безпеки та перевірки ідентичності рис. 4.18.

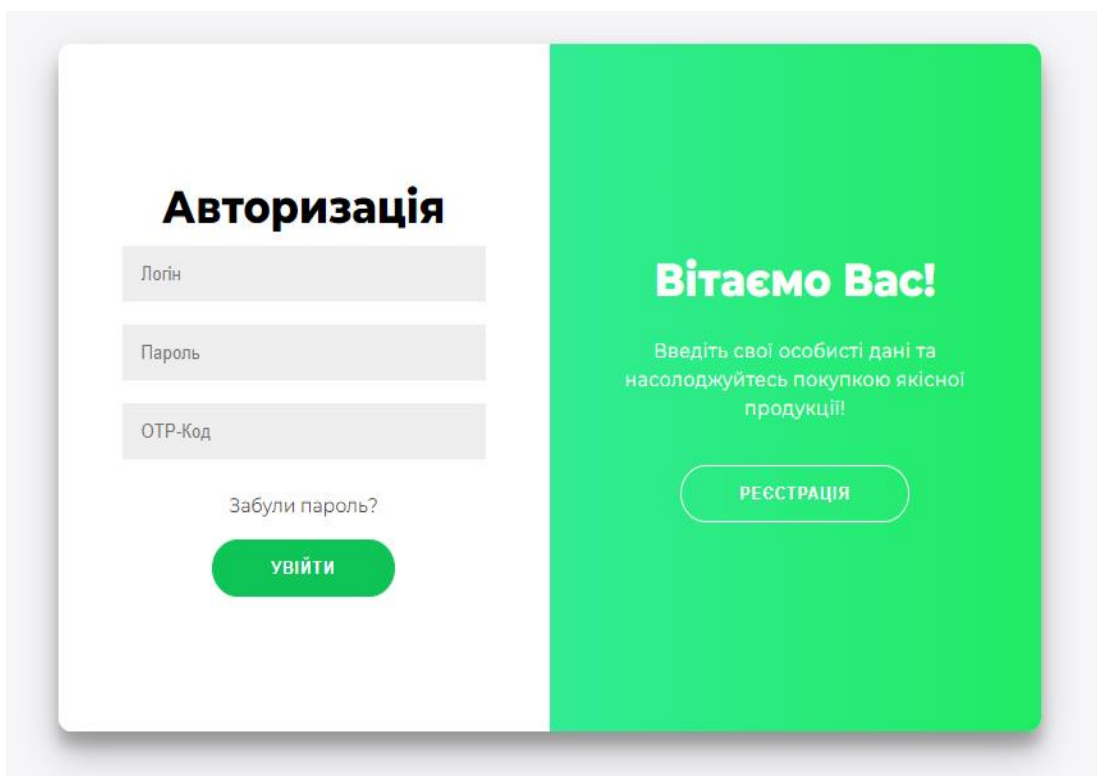


Рисунок 4.18 – Вікно авторизації та вводу ОTR-коду

Таким чином цей підхід надає користувачам додатковий рівень безпеки, при цьому використання його є малочасовитим та ефективним. Цей метод забезпечує надійний механізм захисту, який ускладнює можливість незаконного доступу та здобуття важливої інформації зловмисниками.

4.4.3 Структура PostgreSQL та захист від SQL-ін'єкцій

Центральним елементом управління інформацією користувачів та вебдодатком є система баз даних. База даних (Database) представляє собою набір даних, що відображає стан різних об'єктів у конкретній області та їх взаємозв'язки для відповіді на вимоги користувача.

Для цієї реалізації було обрано базу даних PostgreSQL, яка дозволяє ефективно зберігати та управляти інформацією, пов'язаною з різними аспектами діяльності, такими як облік клієнтів, замовлень та іншої важливої інформації. Вона надає зручний механізм для організації цих даних та ефективної взаємодії з ними.

Розглянемо структуру бази даних, яка була розроблена для вебсайту проєкту на рис. 4.19.

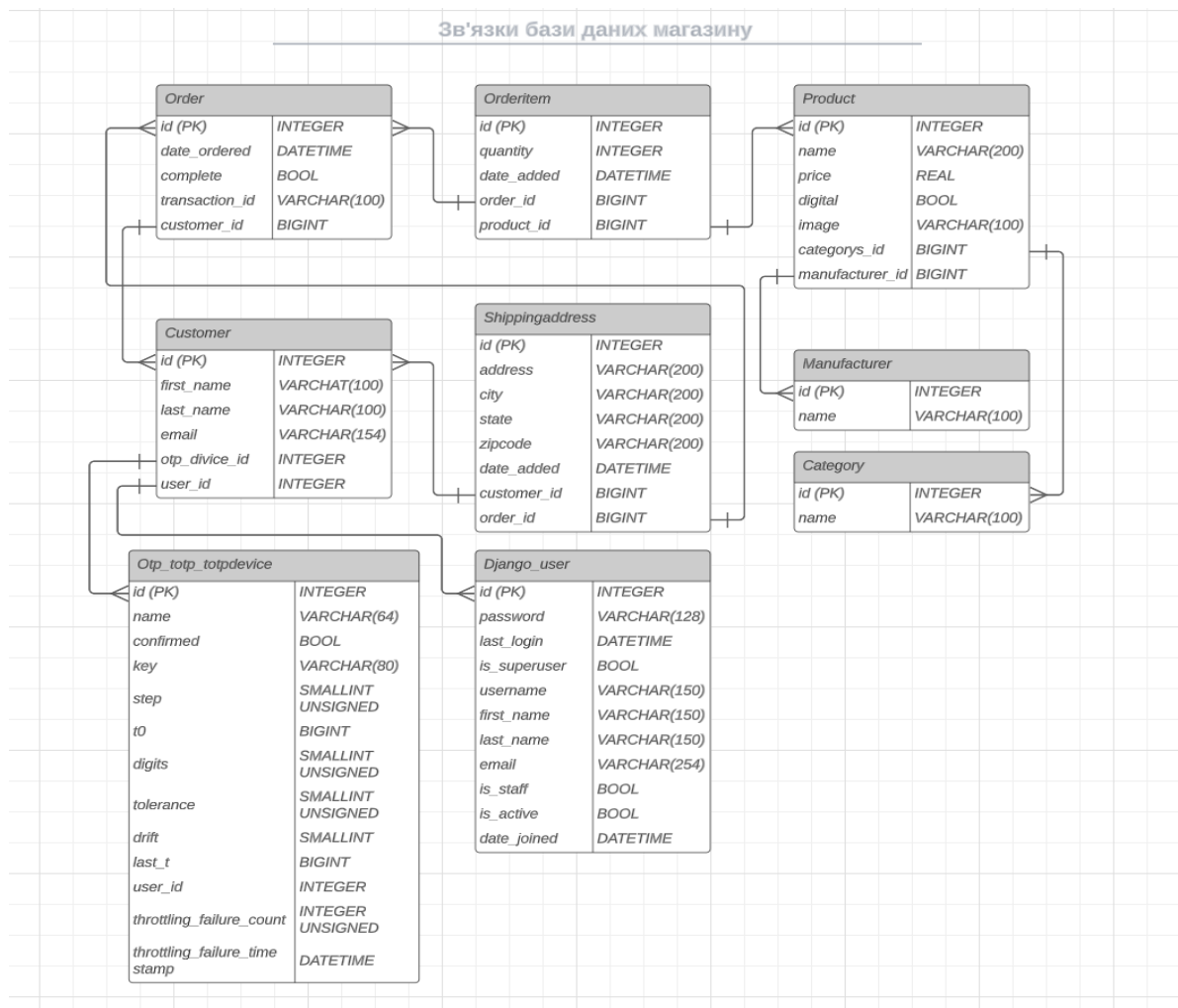


Рисунок 4.19 – Схема зв'язків бази даних вебсайту

З розгляду цих зв'язків можна визначити, що таблиці в базі даних мають чітку структуру та взаємозалежності, які визначаються логічною послідовністю. Детальний аналіз вказує на такі аспекти:

- центральною таблицею є Customer, яка зберігає otp-divice та виступає головним вузлом, на який посилаються інші таблиці;
- далі розташовані таблиці Product, Order, OrderItems та Shippingaddress, які ідентифікують записи замовлення користувача, елементи, включені у замовлення, опис товарів, їх ціна та адресу доставки замовлення;

- допоміжними таблицями є `Manufacturer` та `Category`, які використовуються для каталогізації товарів та позначення їх виробників;
- поміж системних таблиць важливими є `Django_user` та `Otp_totp_totpdivice`, які відповідають за авторизацію користувачів у системі.

Проаналізувавши основну структуру бази даних проєкту виникає необхідність вирішення питання щодо захисту бази даних від небажаних втручань та атак з боку злоумисників.

Необхідно налаштувати захист та шифрування через протоколи SSL/TLS. Це дозволить зашифрувати обмін даними між базою даних та запитами користувача.

Для активації цієї функціональності на нашому вебсайті, потрібно внести відповідні параметри у файл налаштувань Django, зокрема:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'sipsavordb',
        'USER': 'admin',
        'PASSWORD': '*****',
        'HOST': 'localhost',
        'PORT': '',
        'OPTIONS': {
            'sslmode': 'require',
        }
    }
}
```

У даній частині параметр `sslmode` визначає необхідність використання SSL для встановлення з'єднання з базою даних і вимагає використання захищеного з'єднання. Таким чином, всі взаємодії з базою даних будуть здійснюватися через захищене та зашифроване з'єднання, що забезпечить додатковий рівень безпеки.

4.4.4 Сервер-додатків Gunicorn

Після аналізу бази даних переходимо до налаштування сервера-додатків Gunicorn. Для встановлення Gunicorn використовуємо наступну команду в терміналі – `pip install gunicorn`.

Після успішного завантаження Gunicorn, проводимо перевірку роботи процесу, використовуючи наступну команду – `gunicorn --bind 0.0.0.0:8000 sipandsavor.wsgi`.

У нашому випадку установка пройшла успішно, тому розглянемо налаштування функціоналу gunicorn. Для того, щоб сервер функціонував у фоновому режимі, необхідно налаштувати сокет-процес. Почнемо з створення нового файлу сокету, в якому будуть зберігатися наші налаштування – `sudo nano /etc/systemd/system/gunicorn.socket`.

В новому файлі ми вказуємо такі параметри:

[Unit]

Description=gunicorn socket

[Socket]

ListenStream=/run/gunicorn.sock

[Install]

WantedBy=sockets.target.

У цьому файлі вказуються шляхи, на які слід посилатися нашому сокету. Після завершення налаштування сокету, розглянемо конфігураційний файл сервісу, де визначаються основні параметри нашого сервера-додатків. Для створення файлу сервісу використовуємо наступну команду – `sudo nano /etc/systemd/system/gunicorn.service`.

В ньому визначаємо наступні параметри:

[Unit]

Description=gunicorn daemon

Requires=gunicorn.socket

After=network.target

```
[Service]
User=maksim
Group=www-data
WorkingDirectory=/home/maksim/AllSites/EcommerceSite/SipAndSavor
ExecStart=/home/maksim/venv/ts/bin/gunicorn \
    --access-logfile - \
    --workers 3 \
    --bind unix:/run/gunicorn.sock \
    SipAndSavor.wsgi:application
[Install]
WantedBy=multi-user.target.
```

Проаналізуємо більш детально, які саме параметри було встановлено у файлі конфігурації. У секції [UNIT] визначаємо параметри для запуску нашого процесу, а саме в полі Requires ми вказуємо, що цей процес залежить від процесу gunicorn.socket. У полі After вказуємо, що цей процес має бути запущений після повного або успішного завантаження мережі.

В розділі Service вказуються дані користувача та групи, в межах яких процес буде ініційовано. Далі слід вказати шляхи до директив, де знаходиться наш вебдодаток, сам виконуваний файл gunicorn, а також параметри для нього, такі як файли журналів доступу, кількість робочих процесів, місцезнаходження сокету та модуль WSGI застосунку для запуску.

Після збереження файлу ми активуємо та запускаємо наш сервер-додаток:

- `sudo systemctl start gunicorn.socket;`
- `sudo systemctl enable gunicorn.socket.`

В конфігураційному файлі /etc/nginx/sites-available/sipsavortea також слід вказати шлях до нашого вебзастосунку:

```
location / {
    include proxy_params; proxy_pass
    http://unix:/run/gunicorn.sock;}
```

В кінцевому результаті ми перевіряємо, чи є помилки в журналі командою: `sudo systemctl status gunicorn.socket`, що представлено на рис. 4.20.

```

root@usrv22:/home/maksim# sudo systemctl status gunicorn
● gunicorn.service - gunicorn daemon
   Loaded: loaded (/etc/systemd/system/gunicorn.service; disabled; vendor preset: enabled)
   Active: active (running) since Tue 2023-11-28 21:55:48 UTC; 21s ago
 TriggeredBy: ● gunicorn.socket
   Main PID: 1903 (gunicorn)
     Tasks: 4 (limit: 3384)
    Memory: 103.5M
       CPU: 718ms
    CGroup: /system.slice/gunicorn.service
           └─1903 /home/maksim/venv/ts/bin/python3 /home/maksim/venv/ts/bin/gunicorn --access-logfile - --workers 3 --bind unix:/
           └─1904 /home/maksim/venv/ts/bin/python3 /home/maksim/venv/ts/bin/gunicorn --access-logfile - --workers 3 --bind unix:/
           └─1905 /home/maksim/venv/ts/bin/python3 /home/maksim/venv/ts/bin/gunicorn --access-logfile - --workers 3 --bind unix:/
           └─1906 /home/maksim/venv/ts/bin/python3 /home/maksim/venv/ts/bin/gunicorn --access-logfile - --workers 3 --bind unix:/

Nov 28 21:55:48 usrv22 systemd[1]: Started gunicorn daemon.
Nov 28 21:55:48 usrv22 gunicorn[1903]: [2023-11-28 21:55:48 +0000] [1903] [INFO] Starting gunicorn 21.2.0
Nov 28 21:55:48 usrv22 gunicorn[1903]: [2023-11-28 21:55:48 +0000] [1903] [INFO] Listening at: unix:/run/gunicorn.sock (1903)
Nov 28 21:55:48 usrv22 gunicorn[1903]: [2023-11-28 21:55:48 +0000] [1903] [INFO] Using worker: sync
Nov 28 21:55:48 usrv22 gunicorn[1904]: [2023-11-28 21:55:48 +0000] [1904] [INFO] Booting worker with pid: 1904
Nov 28 21:55:48 usrv22 gunicorn[1906]: [2023-11-28 21:55:48 +0000] [1906] [INFO] Booting worker with pid: 1906
Nov 28 21:55:48 usrv22 gunicorn[1905]: [2023-11-28 21:55:48 +0000] [1905] [INFO] Booting worker with pid: 1905
lines 1-21/21 (END)

```

Рисунок 4.20 – Кінцева перевірка працездатності процесу gunicorn

Розробивши функціонал, було отриманно повністю функціональний та готовий до роботи вебсайт. Наступним етапом буде процес тестування безпеки підключення та виявлення можливих фатальних дірок у системі безпеки.

4.5 Тестування систем безпеки вебсайту

Тестування буде здійснюватися із використанням програм виявлення вразливостей, таких як OWASP ZAP, який дозволить виявити потенційні вразливості та надати інформацію щодо їх місцезнаходження.

Процес сканування вразливостей з використанням OWASP ZAP. Ця програма вже завоювала популярність як засіб для тестування вебзастосунків на вразливості безпеки.

Для початку процесу сканування вебдодатку на наявність потенційних загроз, необхідно вибрати режим роботи (у цьому випадку, це буде режим ATTACK Mode). Цей режим дозволяє системі активно шукати можливі вразливості та потенційні атаки, використовуючи різноманітні методи і техніки. Більш детально про тестування представлено на рис. 4.21.

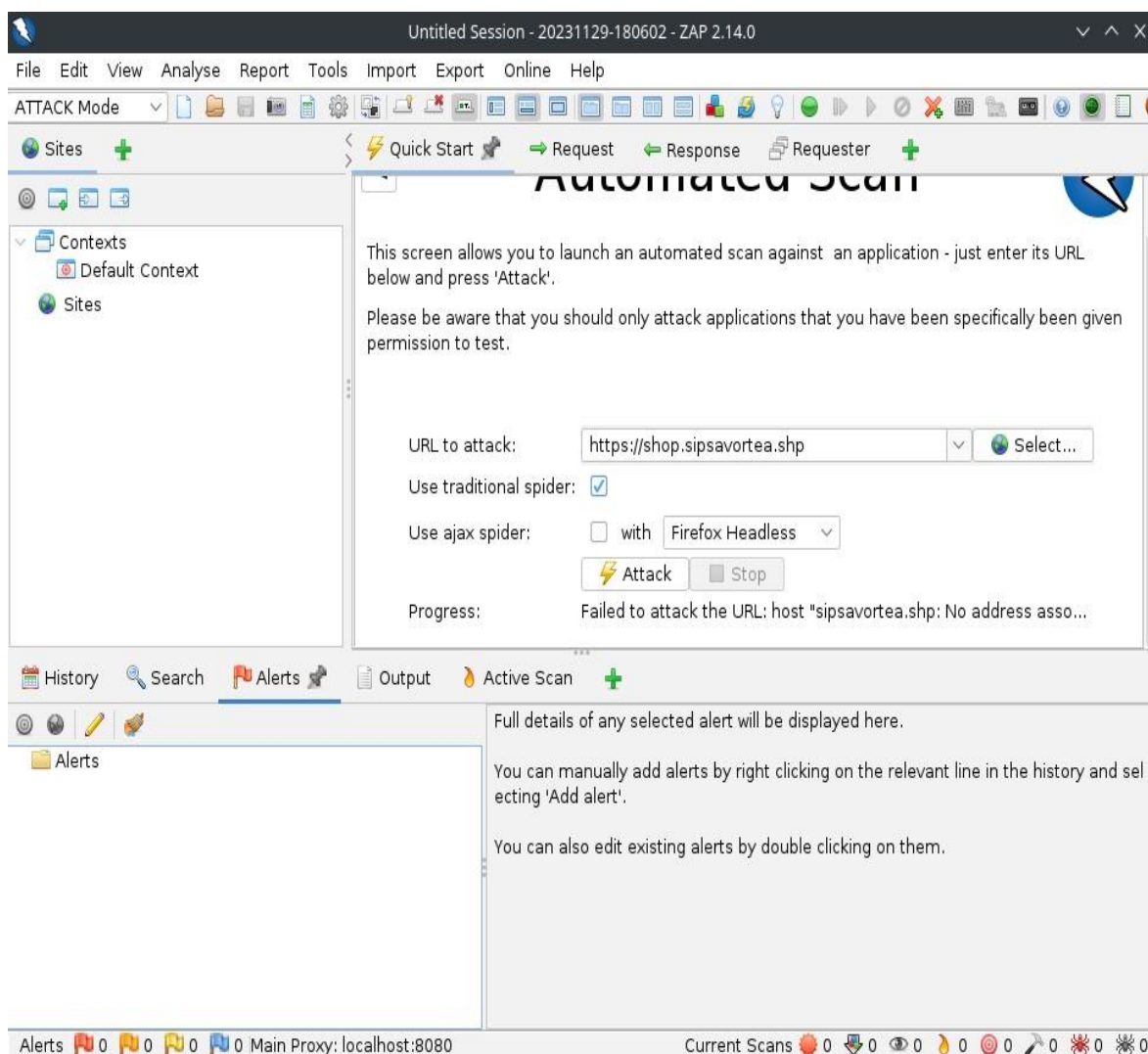


Рисунок 4.21 – Вікно вводу URL-адреси для «атаки» на сайт

Після завершення процесу сканування на вразливості, система надає нам журнали рис. 4.22, проаналізувавши, не було помічено критичних вразливостей у системі.

Є лише невеликі попередження, які можна вважати задовільним результатом, що можна побачити на рис. 4.23.

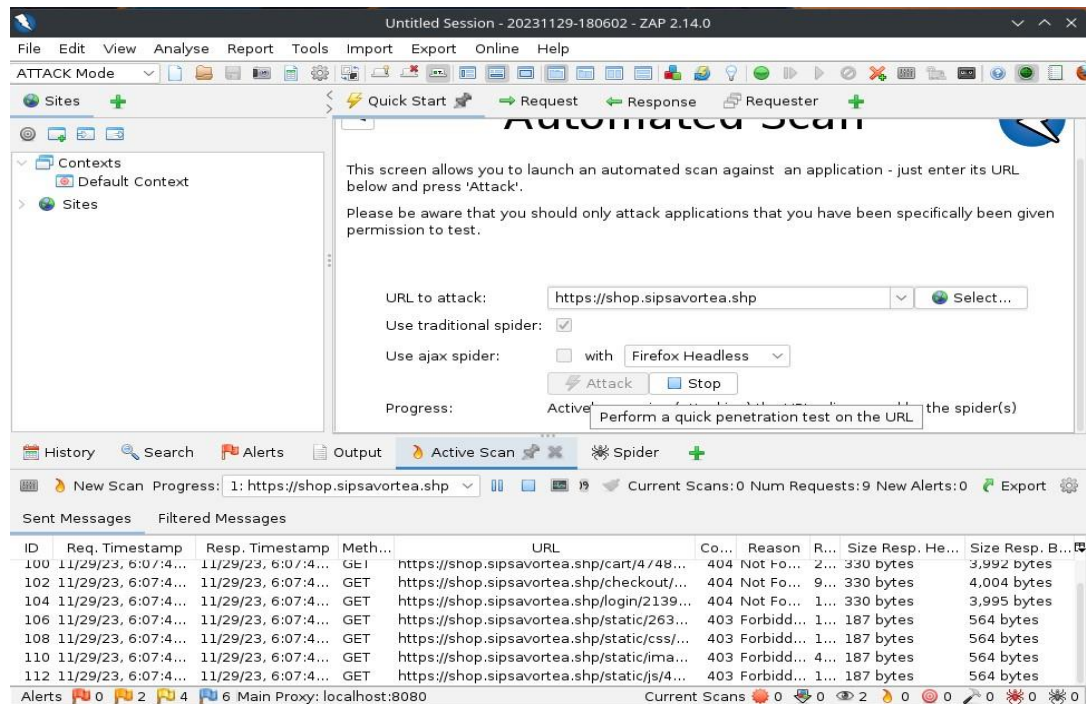


Рисунок 4.22 – Процес та запис логів «атаки» на сайт

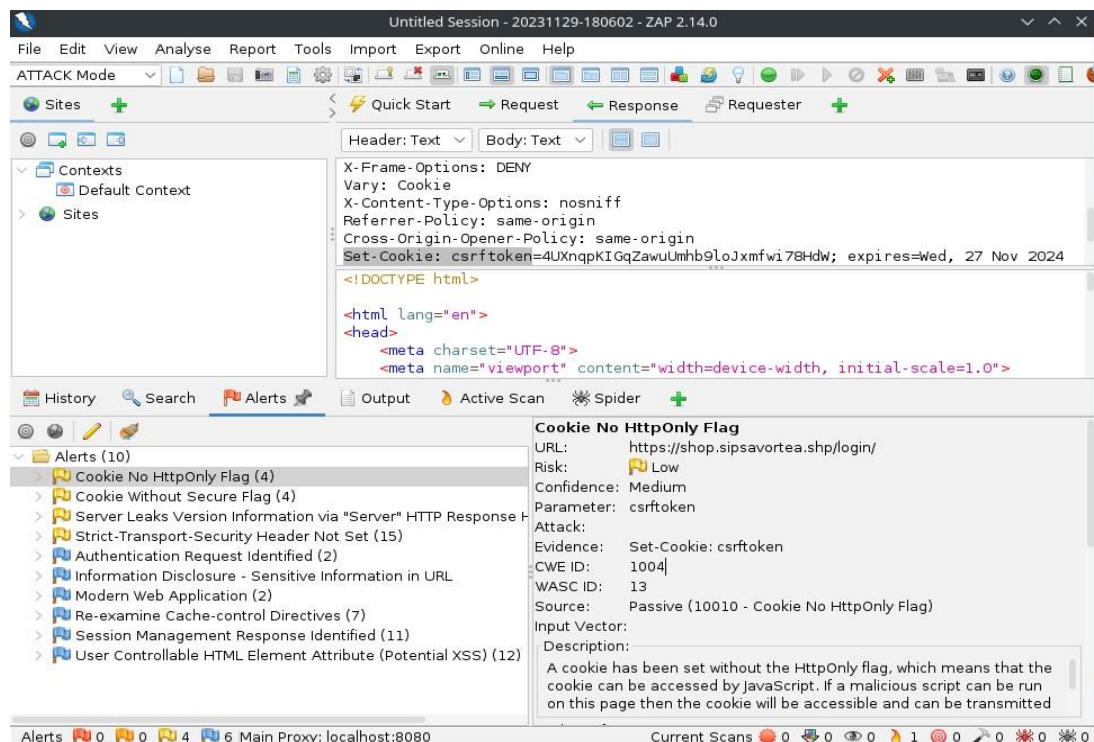


Рисунок 4.23 – Результати сканування безпеки вебдодатку

Завершивши тестування на вразливості за допомогою OWASP ZAP, можна зробити висновок, що система проявила ефективність у виявленні та локалізації потенційних загроз безпеці. За результатами сканування було

виявлено лише дрібні попередження, що свідчить про високий рівень безпеки системи та вдало підготовлений захист від типових атак.

Висновки за розділом 4

Підсумовуючи все вище зазначене можна однозначно сказати, що розробляючи та розгортаючи на сервері свої вебдодатки треба не забувати про їх безпеку. В цьому розділі було досліджено та розроблено багато компонентів системи Ubuntu Server.

Фундаментальним компонентом системи, без якого неможливо обійтись у сфері кібербезпеки, є брандмауер, відомий також як Firewall. Брандмауер виконує роль ефективного стража, який контролює потік інформації між внутрішньою мережею та зовнішнім середовищем. Він діє як невидимий щит, який визначає, які з'єднання та комунікації мають право пройти, а які мають бути заблоковані. І в цьому розділі було не тільки його налаштування, але й дослідження ролі, для якої він використовується у захисті системи.

Ключовими аспектами в області безпеки є протоколи мережевого захисту SSL/TLS, які забезпечують гарантію того, що всі дані, які передаються між сервером і користувачем, будуть зашифровані та залишаться конфіденційними перед третіми сторонами.

Зокрема, було розроблено захищений комерційний вебсайт, де користувачі можуть здійснити покупки та придбати необхідні товари. Зокрема, була впроваджена двоетапна аутентифікація, яка додає додатковий рівень захисту від несанкціонованого доступу з боку третіх осіб.

Додатково було проведено тестування системи на предмет наявності вразливостей за допомогою інструменту OWASP ZAP. За результатами аналізу виявлено, що система ефективно виявляє та ізолює потенційні загрози безпеки. Результати сканування показали лише дрібні попередження,

що свідчить про високий рівень безпеки системи та готовність до захисту від типових атак.

ВИСНОВКИ

Підбиваючи підсумки проведеної роботи, можна констатувати, що у даному дослідженні були ретельно розглянуті та вивчені основні теоретичні аспекти криптографічного шифрування даних. Це дозволило отримати глибше розуміння теми та встановити підґрунтя для подальшого поглибленого дослідження. Після розгляду фундаментальних концепцій теми, у магістерській роботі було здійснено аналіз історії криптографії та криптології як наукової дисципліни.

У ході дослідження були детально розглянуті історичні методи захисту інформації, а також сучасні методи шифрування, включаючи їхній розвиток та взаємозалежність. В результаті проведеного аналізу сформовано класифікацію криптографічних шифрів та виявлено ключові тенденції в розвитку сучасної криптографії та криптології.

Окрім теоретичних висновків, робота включає в себе практичний аспект, а саме розробку сучасного комерційного сайту. Під час його створення були використані передові методи та засоби шифрування даних, такі як протоколи SSL та TLS, що сприяють ефективному захисту інформації в онлайн-середовищі.

В заключенні, дослідження також охоплює питання перспективних напрямків у сфері захисту інформації, які ще перебувають на стадії розробки, але можуть мати значущий вплив на майбутнє шифрування даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Великий тлумачний словник сучасної української мови [Текст] / В. Т. Бусел. – Ірпінь: Перун, 2005. – 1728 с.
2. Richard, K. Glossary of Key Information Security Terms [Text] / Richard Kissel. – Gaithersburg : NIST, 2013. – 222 p.
3. Принципи побудування та аналізу інфраструктур відкритого ключа на основі застосування технології блокчейн [Електрон. ресурс] / [І. Д. Горбенко, О. В. Потій, Ю. І. Горбенко, А. І. Пушкарьов, М. В. Єсіна] / Радіотехніка. – Режим доступу: http://nbuv.gov.ua/UJRN/rvmnts_2019_198_14.
4. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си [Текст] / Брюс Шнайер. – М.: Триумф, 2002. – 816 с.
5. Корченко, О. Г. К34 Прикладна криптологія : системи шифрування : підручник [Текст] / О. Г. Корченко, В. П. Сіденко, Ю. О. Дрейс. – Житомир : ДУТ, 2014. – 448 с.
6. Богуш, В. М. Криптографічні застосування елементарної теорії чисел [Текст] / В. М. Богуш, В. А. Мухачов / Навчальний посібник – Київ : ДУІКТ, 2006. – 126 с.
7. Криптоаналіз. Криптографічні протоколи: Навчальний посібник / О. М. Гапак, П. П. Горват – Ужгород : УжНУ, 2021. – 93 с.
8. Гребенніков, В. В. Історія криптології & секретного зв'язку [Текст] / В. В. Гребенніков – Ужгород : Ліра, 2012. – 664 с.
9. Термінологія в галузі захисту інформації в комп'ютерних системах від несанкціонованого доступу : НД ТЗІ 1.1-003-99. – [Чинний від 1999-04-28]. – К.: ДСТСЗІ СБ України, 1999. – 30 с. – (Нормативний документ Департаменту спеціальних телекомунікаційних систем та захисту інформації Служби безпеки України).
10. Богуш, В. М. Інформаційна безпека: Термінологічний навчальний довідник [Текст] / В. Г. Кривуца, А. М. Кудін – К.: ООО Д.В.К., 2004. – 508 с.

11. Доріченко, С. А. 25 етюдів про шифри: Популярно про сучасну криптографію [Текст] / С. А. Доріченко, В. В. Ященко. – М.: Теїс, 1994. – 72 с.
12. Баричев, С. Г. Основи сучасної криптографії [Текст] / С. Г. Барич, Р. Е Сєров – М.: Гаряча лінія - Телеком, 2002. – 152 с.
13. Адаменко, М. В. Основи класичної криптології: секрети шифрів та кодів [Текст] / М. В. Адаменко – М.: ДМК Прес, 2012. – 256 с.
14. Молдовян, Н. А. Введення в криптосистеми з відкритим ключем. [Текст] / Н. А. Молдовян, А. А. Молдовян. – С.-Пб.: БХВ Петербург, 2005. – 288 с.
15. Основи криптографії: [навчальний посібник. 2-ге вид., Вик. і доп.] [Текст] / А. П. Алфьоров, А. Ю. Зубов, А. С. Кузьмін, А. В. Черемушкін. – М.: Геліос АРВ, 2002. – 480 с.
16. Вербіцький, О. В. Вступ до криптології [Текст] / О. В. Вербіцький – Л.: ВНТЛ, 1998. – 247 с.
17. Аграновський, А. В. Практична криптографія: алгоритми та їх програмування [Текст] / А. В. Аграновський, Р. А. Хаді. – М.: СОЛОН-Прес, 2009. – 256 с.
18. Історія розвитку криптографії [Електрон. ресурс]. – Режим доступу: <https://studfile.net/preview/7111623/>.
19. Криптографія, шифри їх види та властивості [Електрон. ресурс]. – Режим доступу: https://ua-referat.com/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D1%96%D1%8F_%D0%A8%D0%B8%D1%84%D1%80%D0%B8_%D1%97%D1%85_%D0%B2%D0%B8%D0%B4%D0%B8_%D1%96_%D0%B2%D0%BB%D0%B0%D1%81%D1%82%D0%B8%D0%B2%D0%BE%D1%81%D1%82%D1%96.
20. Jefferson Cipher Wheel | Taken at the National Cryptologic [Electronic resource]. – Access mode: <https://www.flickr.com/photos/ideonexus/5176773952>

21. The Enigma Machine – The National Museum of Computing [Electronic resource]. – Access mode: <https://www.tnmoc.org/bh-2-the-enigma-machine>.
22. Converter M-209-B – Mark's Tech Journal [Electronic resource]. – Access mode: <https://www.nf6x.net/2009/02/converter-m-209-b/>.
23. Захист інформації в телекомунікаційних системах: Навчальний посібник / [Г.Ф. Конахович, В.П. Климчук, С.М. Паук, В.Г. Потапов, В.М. Чуприн, О.О. Горбунов]. – К.: НАУ, 2009. – 380 с.
24. Juniper Research. The Future of Cybercrime & Security: Financial and Corporate Threats & Mitigation [Electronic resource]. – Access mode: <https://www-cdn.webroot.com/5415/0396/2242/The-Future-of-Cybercrime-and-Security-Juniper.pdf>.
25. McAfee. Net Losses: Estimating the Global Cost of Cybercrime [Electronic resource] / McAfee, Center for Strategic and International Studies. McAfee, 2018. – Access mode: <https://csis-prod.s3.amazonaws.com/s3fs-public/legac>.
26. Kahn, D. The Codebreakers – The Story of Secret Writing [Text]. / D. Kahn. – New York: The Macmillan Company, 1996. – 1200 p.
27. Cohen, F. A Short History of Cryptography [Text]. / Fred Cohen – Pittsburgh : ASP, PO Box, 1995. – 30 p.
28. Захист інформації. Криптографічні методи [Текст] : підруч. для вищ. навч. закл. / І. І. Маракова [та ін.] ; Одеський держ. політехнічний ун-т, Інститут радіоелектроніки і телекомунікацій. - Одеса : ОДПУ, 2001. – 164 с.
29. Технології захисту інформації: Навчальний посібник [Текст] / [С. Е. Остапов, С. П. Євсєєв, О. Г. Король] – Харків : Видавництво ХНЕУ, 2009. – 476 с.
30. Secure Shell (SSH) | CQR [Electronic resource]. – Access mode: <https://cqr.company/wiki/protocols/secure-shell-ssh/>.
31. January 2023 Web Server Survey | Netcraft [Electronic resource]. – Access mode: <https://www.netcraft.com/blog/january-2023-web-server-survey/>.

32. A detailed comparison between Apache and Nginx web server [Electronic resource]. – Access mode: <https://www.site24x7.com/learn/web-server/apache-vs-nginx-web-server.html>.

33. Що вибрати: PHP або Python - Компанія Wezom [Електрон. ресурс]. – Режим доступу: <https://wezom.com.ua/blog/chto-vybrat-php-ili-python>.

34. What is Python? - Python Programming Language Explained – AWS [Electronic resource]. – Access mode: <https://aws.amazon.com/what-is/python/>.

35. Налаштування gunicorn та uwsgi порівняння продуктивності [Електрон. ресурс]. – Режим доступу: <https://profit.me/2011/08/16/nastrojka-gunicorn-i-uwsgi-sravnenie-proizvoditeln/>.

36. Diffie, W., (1976). Multi-user cryptographic techniques. [Text] / Diffie, W., Hellman, M. – Montvale : AFIPS Press, 1976. – 1075 p.

37. How To Set Up a Firewall with UFW on Ubuntu 18.04 | DigitalOcean [Electronic resource]. – Access mode: <https://www.digitalocean.com/community/tutorials/how-to-set-up-a-firewall-with-ufw-on-ubuntu-18-04>.

38. What is SSH Protocol? How does it work? [Electronic resource]. – Access mode: <https://www.wallarm.com/what/what-is-ssh-protocol>.

ДОДАТОК А
Налаштування NGINX

```
user www-data;
worker_processes auto;

pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 768;
}

http {
    limit_req_zone $binary_remote_addr zone=req_limit:10m
rate=1r/s;
    limit_conn_zone $binary_remote_addr zone=conn_limit:10m;

    sendfile on;
    tcp_nopush on;
    types_hash_max_size 2048;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping
SSLv3, ref: POODLE
    ssl_prefer_server_ciphers on;

    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    gzip on;
    gzip_vary on;
```

```
gzip_proxied any;  
gzip_comp_level 6;  
gzip_buffers 16 8k;  
gzip_http_version 1.1;  
gzip_types text/plain text/css application/json application/javascript  
text/xml application/xml application/xml+rss text/javascript;  
  
include /etc/nginx/conf.d/*.conf;  
include /etc/nginx/sites-enabled/*;  
}
```

ДОДАТОК Б
Текст програми

Б.1 Текст програми з файлу views.py

```
import json
import datetime
import qrcode
import io

from django.shortcuts import render, redirect
from django.http import JsonResponse, HttpResponse
from django.contrib.auth import authenticate, login
from django.contrib.auth.decorators import login_required
from .forms import LoginForm, RegistrationForm
from .models import *
from django_otp.plugins.otp_totp.models import TOTPDevice
from .utils import cookieCart, cartData, guestOrder

def store(request):
    data = cartData(request)
    cartItems = data['cartItems']
    order = data['order']
    items = data['items']
    products = Product.objects.all()
    context = {'products': products, 'cartItems': cartItems}
    return render(request, 'store/store.html', context)

def cart(request):
    data = cartData(request)
    cartItems = data['cartItems']
    order = data['order']
    items = data['items']
    context = {'items': items, 'order': order, 'cartItems': cartItems}
    return render(request, 'store/cart.html', context)
```

```

def checkout(request):
    data = cartData(request)
    cartItems = data['cartItems']
    order = data['order']
    items = data['items']
    context = {'items': items, 'order': order, 'cartItems': cartItems}
    return render(request, 'store/checkout.html', context)

def updateItem(request):
    data = json.loads(request.body)
    productId = data['productId']
    action = data['action']
    print('Action:', action)
    print('Product:', productId)
    customer = request.user.customer
    product = Product.objects.get(id=productId)
    order, created = Order.objects.get_or_create(customer=customer,
complete=False)
    orderItem, created = OrderItem.objects.get_or_create(order=order,
product=product)
    if action == 'add':
        orderItem.quantity = (orderItem.quantity + 1)
    elif action == 'remove':
        orderItem.quantity = (orderItem.quantity - 1)
    orderItem.save()
    if orderItem.quantity <= 0:
        orderItem.delete()
    return JsonResponse('Item was added', safe=False)

def processOrder(request):

```

```

transaction_id = datetime.datetime.now().timestamp()
data = json.loads(request.body)
if request.user.is_authenticated:
    customer = request.user.customer
    order, created = Order.objects.get_or_create(customer=customer,
complete=False)
else:
    customer, order = guestOrder(request, data)
total = float(data['form']['total'])
order.transaction_id = transaction_id
if total == order.get_cart_total:
    order.complete = True
order.save()
if order.shipping == True:
    ShippingAddress.objects.create(
        customer=customer,
        order=order,
        address=data['shipping']['address'],
        city=data['shipping']['city'],
        state=data['shipping']['state'],
        zipcode=data['shipping']['zipcode'],
    )
return JsonResponse('Payment submitted..', safe=False)

```

#OTP

@login_required

def enable_otp(request):

user_profile, created =

Customer.objects.get_or_create(user=request.user)

if not user_profile.otp_device:

```

        otp_device = TOTPDevice.objects.create(user=user_profile.user,
confirmed=False)
        user_profile.otp_device = otp_device
        user_profile.save()
        return render(request, 'store/enable_otp.html', {'otp_device':
user_profile.otp_device})

```

```
@login_required
```

```

def otp_qr_code(request):
    user_profile = Customer.objects.get(user=request.user)
    otp_device = user_profile.otp_device
    if otp_device:
        otpauth_url = otp_device.config_url
        qr = qrcode.QRCode(
            version=1,
            error_correction=qrcode.constants.ERROR_CORRECT_L,
            box_size=10,
            border=4,
        )
        qr.add_data(otpauth_url)
        qr.make(fit=True)
        img = qr.make_image(fill_color="black", back_color="white")
        buffer = io.BytesIO()
        img.save(buffer, format="PNG")
        qr_code_img = buffer.getvalue()
        return HttpResponse(qr_code_img, content_type="image/png")
    else:
        return HttpResponse("OTP is not enabled for this user.")

```

```
#Login/Registration
```

```

def user_auth(request):
    if request.method == 'POST':
        loginform = LoginForm(request.POST)
        registerform = RegistrationForm(request.POST)
        if 'login_submit' in request.POST and loginform.is_valid():
            username = loginform.cleaned_data['username']
            password = loginform.cleaned_data['password']
            otp_code = loginform.cleaned_data['otp_code']
            user = authenticate(request, username=username,
password=password)
            if user is not None:
                user_profile = user.customer
                if user_profile.otp_device and
user_profile.otp_device.verify_token(otp_code):
                    login(request, user)
                    return redirect('store')
                else:
                    return render(request, 'store/login.html', {'loginform':
loginform, 'error': 'Invalid OTP'})
            else:
                return render(request, 'store/login.html', {'loginform': loginform,
'error': 'Invalid credentials'})
        elif 'register_submit' in request.POST and registerform.is_valid():
            user = registerform.save()
            login(request, user)
            return redirect('store')
        else:
            loginform = LoginForm()
            registerform = RegistrationForm()

```

```

    return render(request, 'store/login.html', {'loginform': loginform,
'registerform': registerform})

```

Б.2 Текст програми з файлу `models.py`

```

from django.db import models
from django.contrib.auth.models import User
from django_otp.plugins.otp_totp.models import TOTPDevice

class Customer(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    email = models.EmailField()
    otp_device = models.OneToOneField(TOTPDevice, null=True,
blank=True, on_delete=models.CASCADE)

    def __str__(self):
        return self.user.username

class Product(models.Model):
    name = models.CharField(max_length=200)
    categorys = models.ForeignKey('Category',
on_delete=models.PROTECT)
    manufacturer = models.ForeignKey('Manufacturer',
on_delete=models.PROTECT)
    price = models.FloatField()
    digital = models.BooleanField(default=False, null=True, blank=True)
    image = models.ImageField(null=True, blank=True)

```

```

def __str__(self):
    return self.name

@property
def imageURL(self):
    try:
        url = self.image.url
    except:
        url = "
    return url

```

```

class Category(models.Model):
    name = models.CharField(max_length=100, db_index=True)

    def __str__(self):
        return str(self.name)

```

```

class Manufacturer(models.Model):
    name = models.CharField(max_length=100, db_index=True)

    def __str__(self):
        return str(self.name)

```

```

class Order(models.Model):
    customer = models.ForeignKey(Customer,
on_delete=models.SET_NULL, null=True, blank=True)
    date_ordered = models.DateTimeField(auto_now_add=True)
    complete = models.BooleanField(default=False)
    transaction_id = models.CharField(max_length=100, null=True)

```

```

def __str__(self):
    return str(self.id)

@property
def shipping(self):
    shipping = False
    orderitems = self.orderitem_set.all()
    for i in orderitems:
        if i.product.digital == False:
            shipping = True
    return shipping

@property
def get_cart_total(self):
    orderitems = self.orderitem_set.all()
    total = sum([item.get_total for item in orderitems])
    return total

@property
def get_cart_items(self):
    orderitems = self.orderitem_set.all()
    total = sum([item.quantity for item in orderitems])
    return total

class OrderItem(models.Model):
    product = models.ForeignKey(Product, on_delete=models.SET_NULL,
null=True)
    order = models.ForeignKey(Order, on_delete=models.SET_NULL,
null=True)
    quantity = models.IntegerField(default=0, null=True, blank=True)
    date_added = models.DateTimeField(auto_now_add=True)

```

```

@property
def get_total(self):
    total = self.product.price * self.quantity
    return total

```

```

class ShippingAddress(models.Model):
    customer = models.ForeignKey(Customer,
on_delete=models.SET_NULL, null=True)
    order = models.ForeignKey(Order, on_delete=models.SET_NULL,
null=True)
    address = models.CharField(max_length=200, null=False)
    city = models.CharField(max_length=200, null=False)
    state = models.CharField(max_length=200, null=False)
    zipcode = models.CharField(max_length=200, null=False)
    date_added = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.address

```

Б.3 Текст програми з cart.js

```

import json
from .models import *

def cookieCart(request):
    # Create empty cart for now for non-logged in user
    try:
        cart = json.loads(request.COOKIES['cart'])
    except:
        cart = {}

```

```

print('CART:', cart)

items = []
order = {'get_cart_total': 0, 'get_cart_items': 0, 'shipping': False}
cartItems = order['get_cart_items']

for i in cart:
    # We use try block to prevent items in cart that may have been
    removed from causing error
    try:
        if (cart[i]['quantity'] > 0): # items with negative quantity = lot of
            freebies
            cartItems += cart[i]['quantity']

            product = Product.objects.get(id=i)
            total = (product.price * cart[i]['quantity'])

            order['get_cart_total'] += total
            order['get_cart_items'] += cart[i]['quantity']

            item = {
                'id': product.id,
                'product': {'id': product.id, 'name': product.name, 'price':
                    product.price,
                        'imageUrl': product.imageUrl}, 'quantity':
                cart[i]['quantity'],
                'digital': product.digital, 'get_total': total,
            }
            items.append(item)

```

```

        if product.digital == False:
            order['shipping'] = True
    except:
        pass

    return {'cartItems': cartItems, 'order': order, 'items': items}

def cartData(request):
    if request.user.is_authenticated:
        try:
            customer = request.user.customer
            order, created = Order.objects.get_or_create(customer=customer,
complete=False)
            items = order.orderitem_set.all()
            cartItems = order.get_cart_items
        except Customer.DoesNotExist:
            customer = Customer.objects.create(user=request.user,
first_name=request.user.first_name, last_name=request.user.last_name,
email=request.user.email)
            order = Order.objects.create(customer=customer, complete=False)
            items = order.orderitem_set.all()
            cartItems = order.get_cart_items
    else:
        cookieData = cookieCart(request)
        cartItems = cookieData['cartItems']
        order = cookieData['order']
        items = cookieData['items']

    return {'cartItems': cartItems, 'order': order, 'items': items}

```

```

def guestOrder(request, data):
    name = data['form']['name']
    email = data['form']['email']

    cookieData = cookieCart(request)
    items = cookieData['items']

    customer, created = Customer.objects.get_or_create(
        email=email,
    )
    customer.name = name
    customer.save()

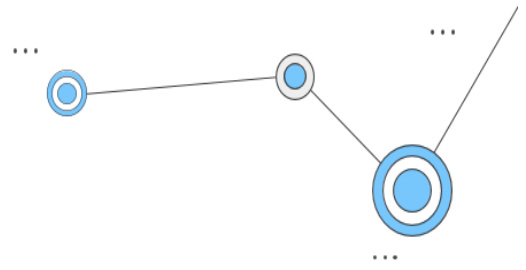
    order = Order.objects.create(
        customer=customer,
        complete=False,
    )

    for item in items:
        product = Product.objects.get(id=item['id'])
        orderItem = OrderItem.objects.create(
            product=product,
            order=order,
            quantity=(item['quantity'] if item['quantity'] > 0 else -1 *
item['quantity']),
            # negative quantity = freebies
        )
    return customer, order

```

ДОДАТОК В
Слайди презентації

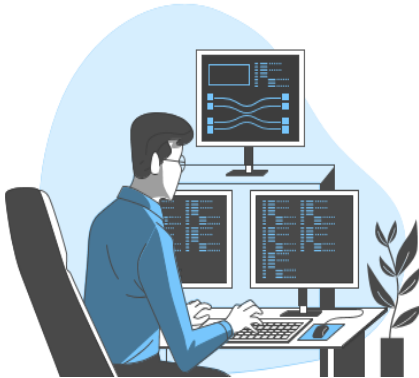
Національний університет «Запорізька політехніка»
Кафедра програмних засобів



Дипломна кваліфікаційна робота магістра

Дослідження та реалізація методів криптографічного
захисту інформації

Research and implementation of methods for
cryptographic protection of information



Виконав
студент групи КНТ-122м

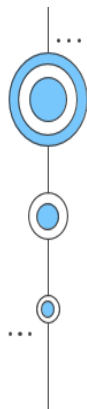
Максим МАСЛОВ

Керівник
к.т.н, доцент

Тетяна ЗАЙКО

1

Рисунок В.1 – Слайд 1



Об'єкт дослідження - реалізація систем криптографічного захисту та методів авторизації на комерційному вебсайті;

Предмет дослідження - впровадження інноваційного методу авторизації на комерційному вебсайті за допомогою одноразових паролів (OTP);

Мета роботи - дослідження та реалізація на практиці методів захисту даних та кондиційної інформації за допомогою криптографічного шифрування інформації.

...

2



Рисунок В.2 – Слайд 2



Вимоги до захисту

Незалежність від логіки алгоритмів - знання логіки алгоритмів не повинно впливати на надійність захисту інформації;

Рівень криптостійкості - рівень криптостійкості повинен перевищувати можливості сучасної обчислювальної техніки;

Перегляд зашифрованої інформації - можливість перегляду інформації тільки при наявності та наданні ключа;

Зміна ключа - незначні зміни у ключі повинні призводити до істотних змін у зашифрованому повідомленні;

Довжина ключа - зміна довжини ключа не повинна зменшувати криптостійкість шифру;

Реалізаційна гнучкість - можливість реалізації як апаратною, так і програмною шляхом без втрат криптостійкості;

Незмінність довжини текстів - довжина вихідного та зашифрованого текстів повинна залишатися незмінною протягом процесу шифрування та розшифрування.

3

Рисунок В.3 – Слайд 3

01
...

Аналіз

аналіз проблем криптографічного захисту інформації;

02
...

Проектування

проектування програмних засобів та методів шифрування;

03
...

Розробка

розробка програмних засобів та методів шифрування;

04
...

Тестування

тестування програмних засобів та методів шифрування.



Завдання

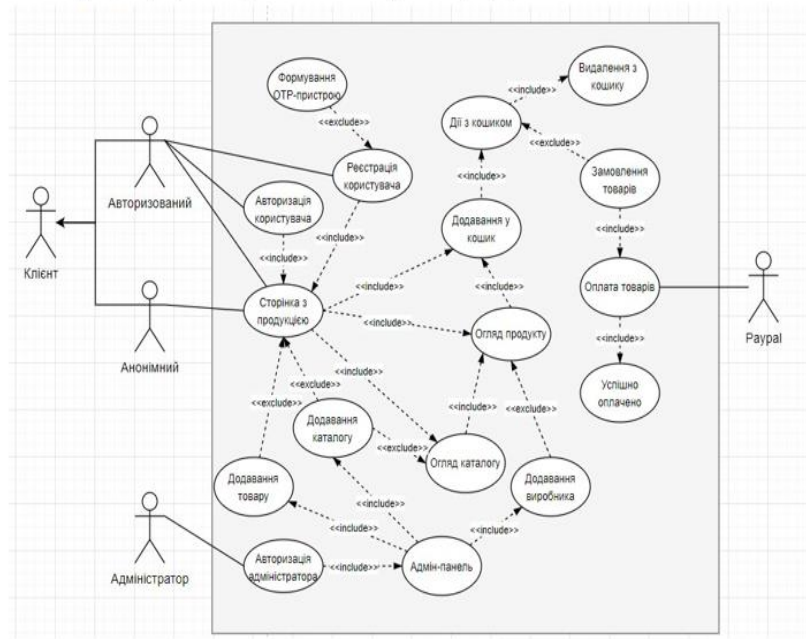


4

Рисунок В.4 – Слайд 4



Діаграма прецедентів



5

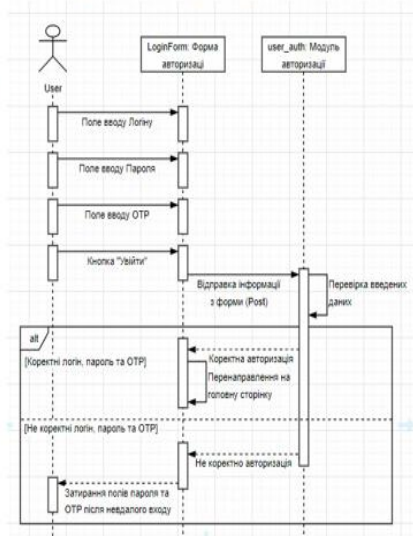
Рисунок В.5 – Слайд 5



Діаграми послідовності авторизації

01

Авторизація користувача



6

02

Реєстрація користувача

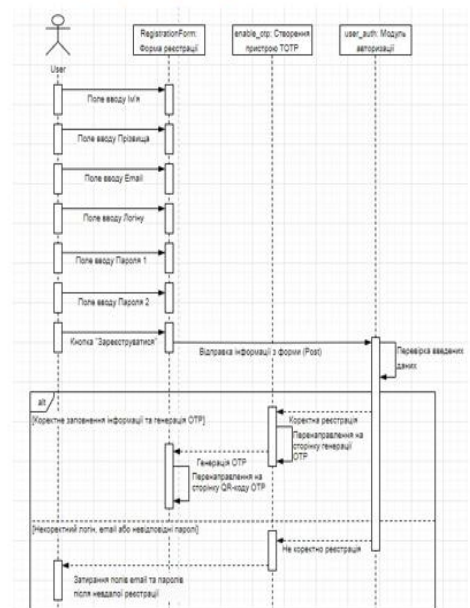


Рисунок В.6 – Слайд 6

Діаграма послідовності функціоналу

03

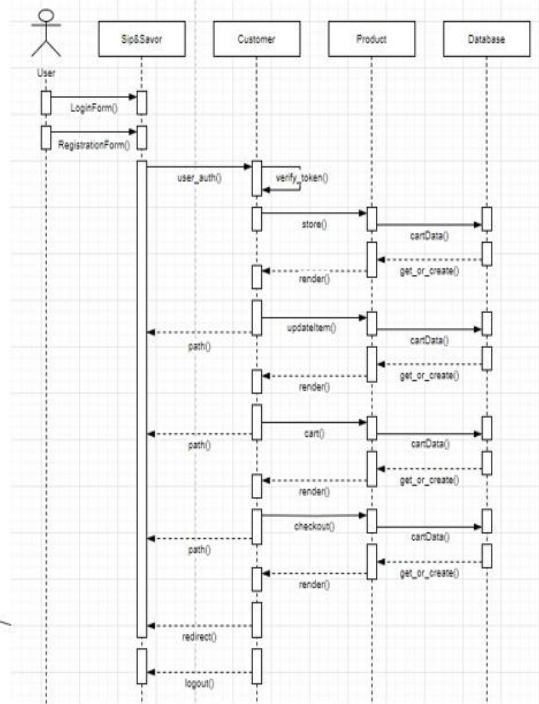


Рисунок В.7 – Слайд 7

Схема бази даних

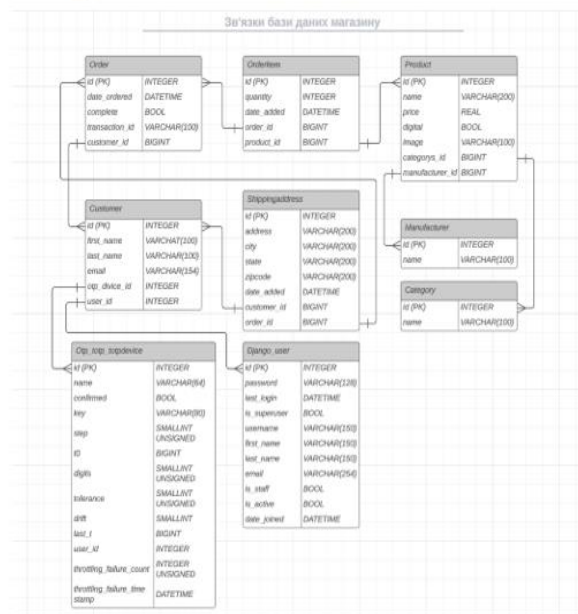
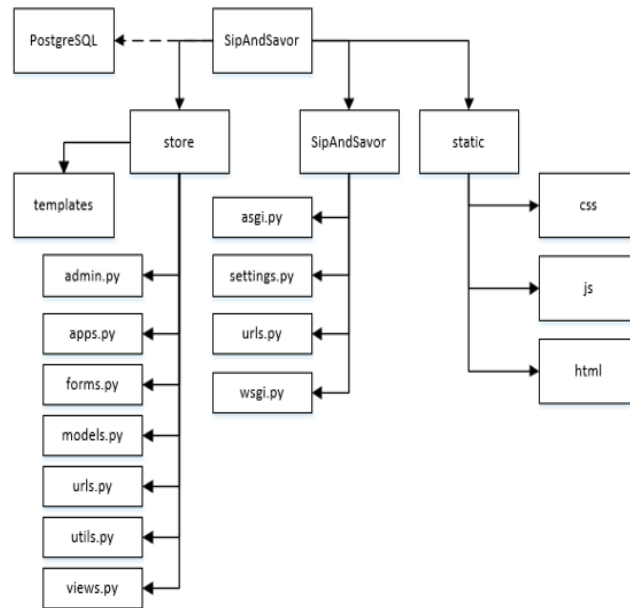


Рисунок В.8 – Слайд 8



Структурна схема програмного забезпечення



9

Рисунок В.9 – Слайд 9

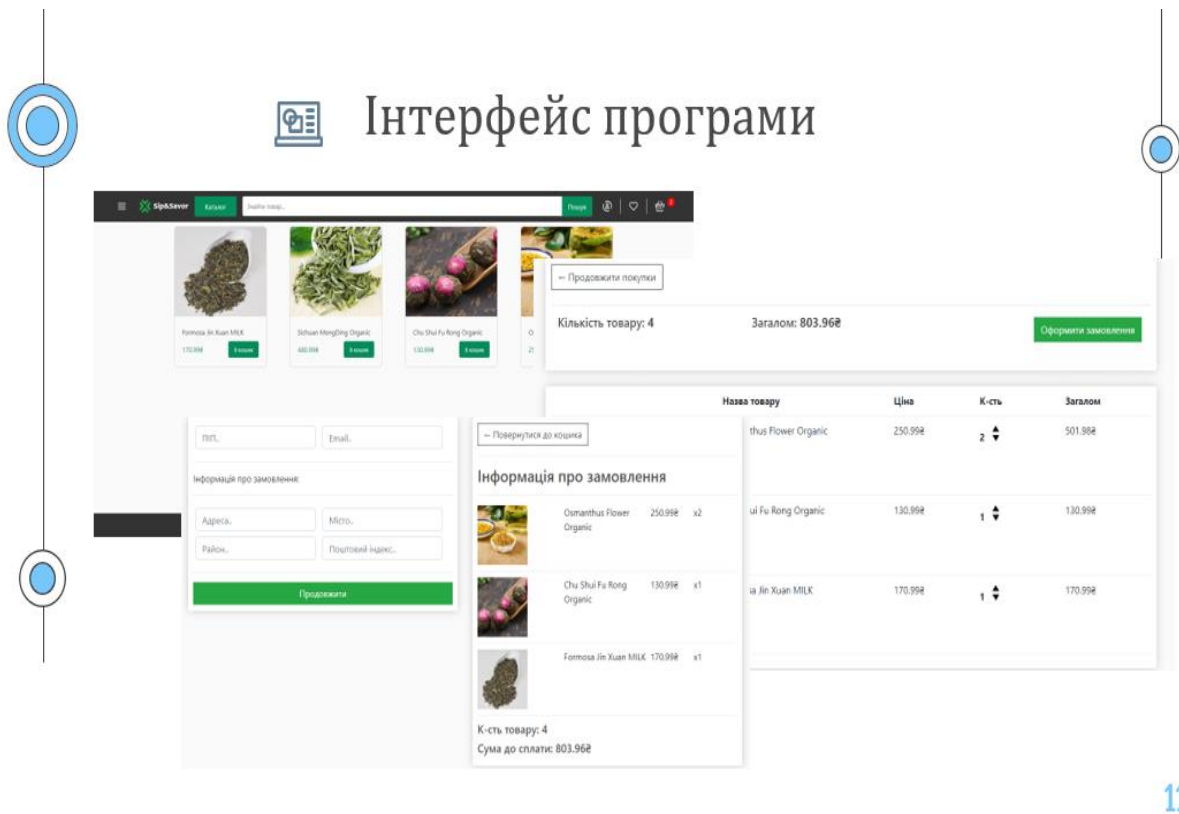


Вибір мови програмування

Властивості	PHP	Python (Django)
Тип мови	скриптова	високорівнева
Розширюваність	вимагає більше організації	багато засобів із можливістю розширення
Швидкість	висока	вище середньої
Безпека	вимагає визначення та виконання правил безпеки	має вбудовані механізми безпеки
Об'єктно-орієнтованість	так	так

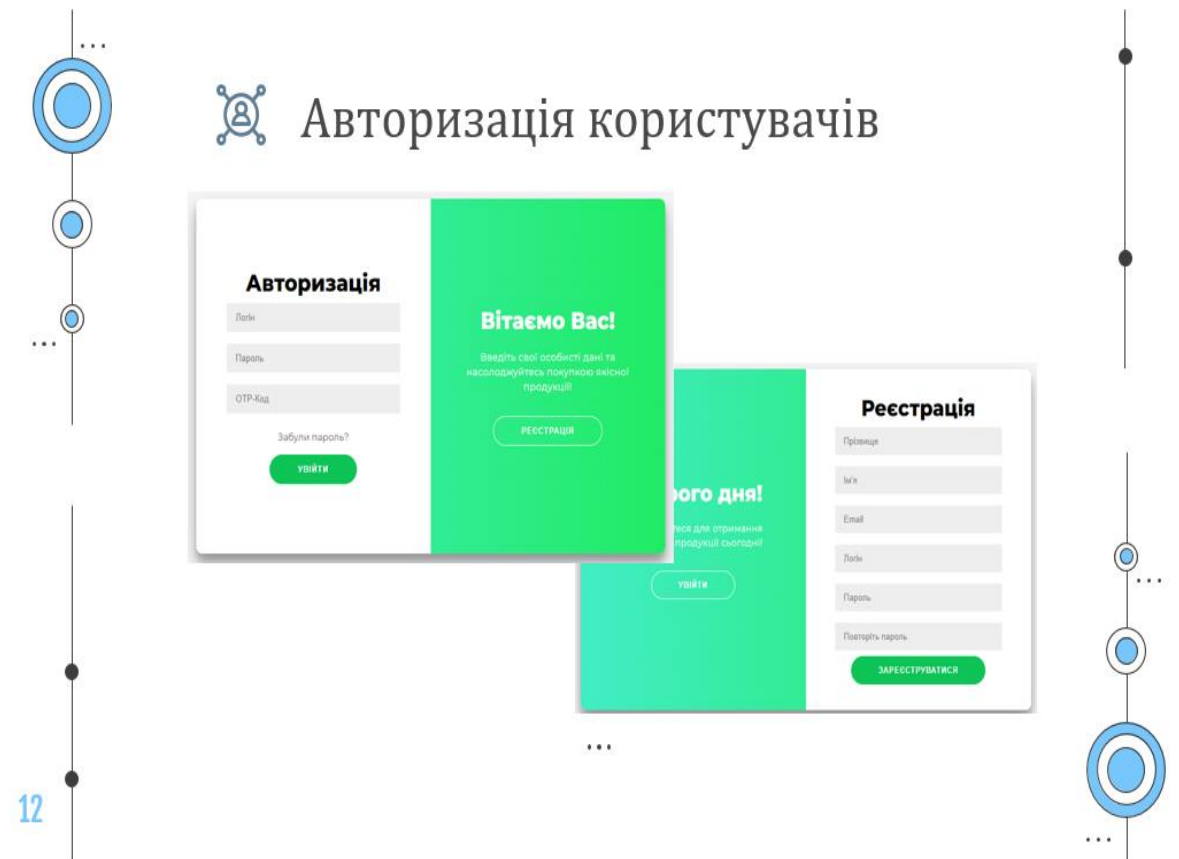
10

Рисунок В.10 – Слайд 10



11

Рисунок В.11 – Слайд 11



12

Рисунок В.12 – Слайд 12



Рисунок В.13 – Слайд 13

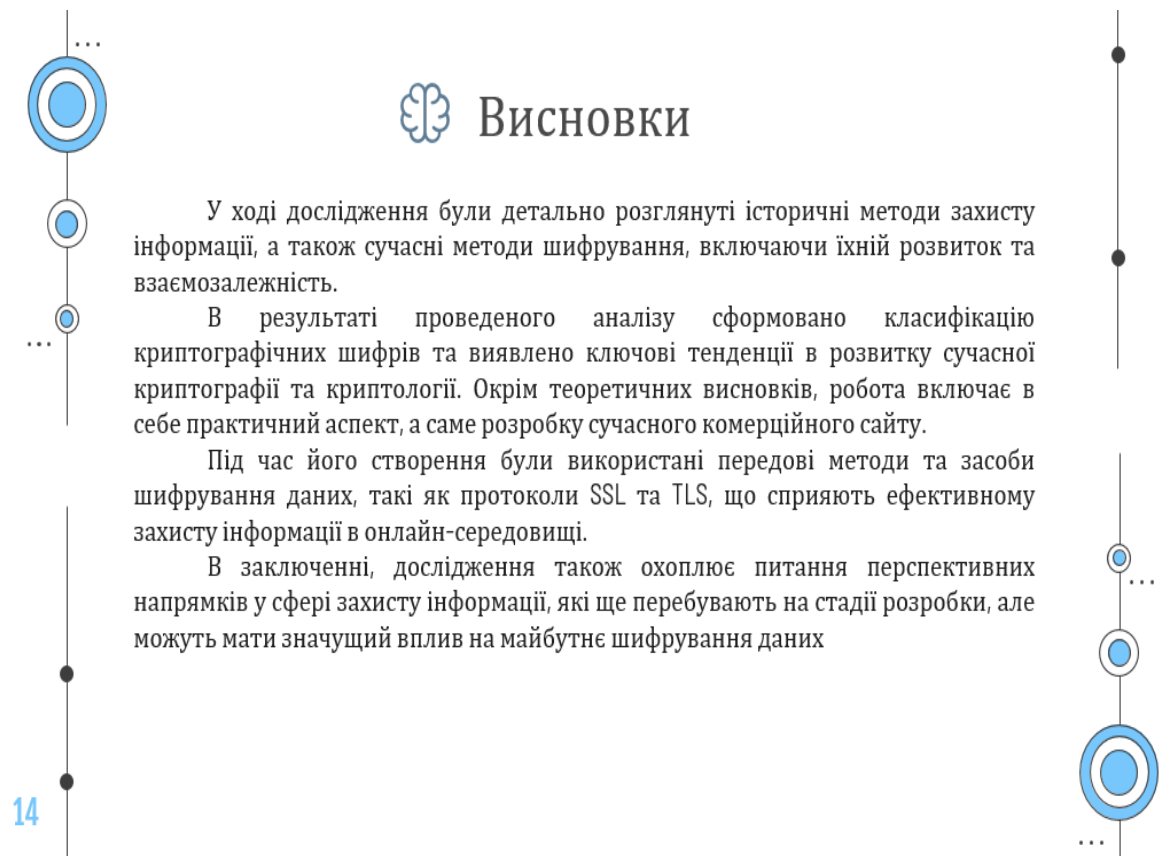


Рисунок В.14 – Слайд 14



Рисунок В.15 – Слайд 15