Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-24М
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Марченко Б.С.
«____» _____ 2025 р.

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Петренюк В.І.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

АНОТАЦІЯ

Марченко Б.С. Дослідження та програмна реалізація системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Метою розробки є дослідження та програмна реалізація системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Об'єктом дослідження є процес забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Предметом дослідження є методи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Методи дослідження базуються на методах теорії захисту інформації та теорії побудови комп'ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Builder C++.

Ключові слова: комп'ютерна інженерія, захисту доступу, WEB ресурси

ABSTRACT

Marchenko B.S. Research and software implementation of a security system for access to a WEB server through opportunistic encryption. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the second (master's) level of higher education, software has been developed that is intended for a security system for access to a WEB server through opportunistic encryption.

The purpose of the development is the research and software implementation of a security system for access to a WEB server through opportunistic encryption.

The object of the research is the process of ensuring security when accessing a WEB server through opportunistic encryption.

The subject of the research is methods for ensuring security when accessing a WEB server through opportunistic encryption.

The research methods are based on the methods of information protection theory and the theory of building computer networks, methods of mathematical statistics, methods of software development.

The result of the work is a software implementation of a security system for accessing the WEB server using opportunistic encryption.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with the software are provided.

The program can be used on a PC with OS Windows 10/11.

The program was developed in the Builder C++ environment.

Keywords: computer engineering, access protection, WEB resources

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

AH	–	автентифікуючий заголовок
CA	–	сертифікаційне співтовариство
DES	–	Data Encription Standard
DoS	–	атака "Відмова в обслуговуванні"
DOI	–	область інтерпретації
ESP	–	Інкапсуляція зашифрованих даних
HTTPS	–	зашифрований http
IAB	–	координаційна рада мережі Internet
IDS	–	система, яка автоматизує процес перегляду подій
IETF	–	проблемна група проектування Internet
IKE	–	протокол обміну ключами за замовчуванням для ISAKMP
IKMP	–	протоколу керування ключами прикладного рівня
IPsec	–	комплект протоколів захисту інформації по IP
ISAKMP	–	механізми узгодження атрибутів використовуваних протоколів
ISP	–	постачальник послуг Internet
MAC	–	коди на перевірку цілісності
MD5	–	дайджест повідомлення
Oakley	–	сесійні ключі на комп'ютери мережі Інтернет
PFS	–	ідеальна пряма безпека
PRF	–	псевдовипадкова функція
SA	–	Security Association
SKIP	–	команда підготовки наступної команди
SPI	–	індекс параметрів безпеки
SPD	–	база даних політики безпеки
TLS	–	протокол захищених сокетів
TCP	–	транспортний протокол

ВСТУП

Актуальність теми. Опортуністичний TLS схожий на згоду зустрітися з кимось для приватної розмови в людному громадському місці. Якщо є вільний куточок для приватної розмови, ви можете безпечно провести делікатну розмову; однак, якщо тихе місце вільне, ви будете вести розмову відкрито та ризикуватимете, що вас можуть підслухати. Спілкування відбувається, але його конфіденційність не гарантується. У світі електронної пошти, коли один поштовий сервер (наприклад, сервер вашої компанії) надсилає повідомлення на інший сервер (наприклад, сервер Gmail одержувача), він намагається встановити з'єднання, зашифроване за допомогою TLS. Зазвичай це робиться за допомогою команди STARTTLS у протоколі SMTP. Якщо сервер-отримувач підтримує STARTTLS і узгодження пройшло успішно, вміст електронного листа шифрується під час передачі. Однак, якщо сервер-отримувач не підтримує TLS або якщо проблема перешкоджає підтвердженню шифрування, електронний лист буде просто надіслано у вигляді звичайного тексту, тобто він не зашифрований і потенційно може бути прочитаний будь-ким, хто його перехопить.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

– Дослідження системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

– Програмна реалізація системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Об'єктом дослідження є процес забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Предметом дослідження є методи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Методи дослідження базуються на методах теорії захисту інформації та теорії побудови комп'ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

– Розроблено вітчизняний продукт забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти LV науково-технічної конференції «Наука в ЦНТУ: основні досягнення та перспективи розвитку» (2025 р.), основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №15.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Переваги та ризики опортуністичного TLS:

Переваги:

– Висока сумісність: TLS гарантує доставку електронних листів практично на будь-який сервер, незалежно від того, чи підтримує цей сервер TLS чи ні. Це запобігає збоям доставки електронної пошти.

– Просте налаштування: Зазвичай його легко ввімкнути в більшості поштових систем.

– Краще, ніж нічого: хоча й не ідеально, але пропонує базовий рівень безпеки, де це можливо, що є покращенням порівняно з надсиланням усіх електронних листів у незашифрованому вигляді.

Ризики:

– Вразливість до підслуховування: якщо зловмисник може втрутитися в рукостискання TLS (наприклад, через атаку зниження рівня захисту), він може змусити з'єднання стати розшифрованим без явного відома ні відправника, ні одержувача.

– Відсутність гарантії конфіденційності: Ви не можете бути впевнені, що ваш електронний лист був зашифрований під час передачі, якщо спеціально не перевірите заголовки листа після доставки.

– Безпека «найкращих зусиль»: забезпечує «найкраще зусилля» при шифруванні, але не гарантує цього.

Випадки використання опортуністичного TLS:

– Загальне електронне спілкування між непов’язаними доменами: це режим за замовчуванням для більшості стандартного електронного трафіку в Інтернеті. Цілком ймовірно, що ваші особисті електронні листи часто

використовують опортуністичний TLS.

– Публічні або відкриті веб-сервіси: де універсальна доступність має пріоритет над гарантованим шифруванням для кожного окремого з’єднання.

– Зворотна сумісність між застарілими середовищами: необхідна для забезпечення зв'язку зі старими системами, які можуть не повністю підтримувати сучасні стандарти TLS.

Приклади опортуністичного TLS:

– Надсилання звичайного електронного листа з вашого особистого облікового запису Gmail до облікового запису Outlook.com друга: сервери спробують використовувати TLS, але якщо це не вдасться, електронний лист все одно пройде незашифрованим.

– Більшість стандартних міждоменних розсилок електронної пошти в Інтернеті спираються на опортуністичний TLS через команду STARTTLS у протоколі SMTP.

1.2 Область застосування

За замовчуванням Office 365 використовує опортуністичний TLS, який намагається зашифрувати повідомлення за допомогою TLS, якщо сервер-отримувач підтримує його, але якщо ні, повернеться до незашифрованої доставки. Однак адміністратори можуть налаштувати примусовий TLS (також відомий як TLS з вимогами до шифрування) за допомогою правил потоку пошти або параметрів конектора, щоб гарантувати, що електронна пошта доставлятиметься лише за умови встановлення безпечного з’єднання TLS, інакше це не вдасться. Це робить примусовий TLS у M365 налаштовуваним, а не стандартним варіантом, з великим недоліком – неможливостю доставки, коли TLS не підтримується сервером-отримувачем.

Gmail за замовчуванням використовує опортуністичний TLS. Під час надсилання або отримання електронної пошти Gmail намагатиметься встановити

безпечне TLS-з’єднання з іншим поштовим сервером. Якщо сервер одержувача підтримує TLS, повідомлення шифрується під час передачі. Однак, якщо TLS не підтримується, Gmail доставить повідомлення незашифрованим. Цей підхід покращує безпеку електронної пошти, не вимагаючи суворого налаштування, але не гарантує шифрування, що робить його менш безпечним, ніж примусовий TLS, у сценаріях, де шифрування є обов’язковим.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Основні методи забезпечення безпеки WEB-додатків

Проблема створення безпечного WEB-застосунку дуже велика. Для її розгляду необхідно повне розуміння слабких місць у системі безпеки. Також необхідно ознайомитися з функціями забезпечення безпеки Windows,.NET Framework і ASP.NET. І, нарешті, дуже важливо розуміти способи використання цих функцій безпеки для боротьби з погрозами.

Навіть при відсутності досвіду забезпечення безпеки існують основні міри забезпечення безпеки WEB-застосунку, які варто почати.

Загальні рекомендації із забезпечення безпеки WEB-застосунку

Навіть самі ретельно продумані системи безпеки додатки можуть бути зламані, якщо зловмисний користувач може скористатися простими способами доступу до комп'ютерів. Необхідно дотримувати наступних правил:

– Частіше створюйте резервну копію й зберігаєте її у фізично безпечному місці.

– Розміщайте комп'ютер WEB-серверу у фізично безпечному місці, куди не можуть потрапити зловмисники, виключайте його або забирайте із собою.

– Використовуйте файлову систему Windows NTFS, а не FAT32. NTFS забезпечує значно більше високий рівень безпеки, чим FAT32. Докладні відомості див. у документації Microsoft Windows.

– Комп'ютер WEB-серверу й всі комп'ютери однієї мережі необхідно захистити за допомогою надійних паролів.

– Забезпечте безпеку служб IIS. Докладніше див. відомості на WEB-вузлі Центра безпеки TechNet.

– Закрийте невикористовувані порти й відключите невикористовувані служби.

– Установите антивірусне програмне забезпечення, що відслідковує вхідний і вихідний трафік.

– Створіть і впровадите політикові, що забороняє користувачам записувати свої паролі в місцях, де їх легко знайти.

– Використовуйте брандмауера.

– Установите останні пакети виправлень від корпорації Microsoft і інших розроблювачів. Наприклад, на WEB-вузлі Центр безпеки TechNet приводиться список останніх бюлетенів безпеки для всіх продуктів Microsoft. Інші розроблювачі мають такі ж WEB-вузли.

– Використовуйте журнали подій Microsoft Windows і регулярно перевіряйте їх на предмет підозрілих операцій. Варто звертати увагу на часті спроби входу в систему або на занадто велике число запитів до WEB-серверу.

Запускайте застосунку з найменшими привілеями

Кожний застосунок запускається в деякому контексті, що володіє певними правами доступу на локальному комп'ютері, а також, можливо, на вилучених комп'ютерах. Щоб запустити застосунок з мінімальними необхідними привілеями, виконуйте наступні рекомендації.

– Не запускайте застосунок із правами системного користувача (адміністратора).

– Запускайте застосунок у контексті користувача з мінімальними необхідними привілеями.

– Задайте дозволу доступу (списки керування доступом) до всіх ресурсів, необхідним додатку. Використовуйте налаштування з мінімальним рівнем дозволів. Наприклад, зробіть файли доступними тільки для читання, якщо це прийнятно для застосунку.

– Зберігаєте файли WEB-застосунку в папці під коренем застосунку. Не дозволяйте користувачам задавати шляхи доступу до файлів застосунку. Це дозволить уникнути одержання користувачами доступу до кореня сервера.

Виконуйте ідентифікацію й перевірку дійсності користувачів

У багатьох додатках користувачі можуть одержати анонімний доступ до вузла (не вводячи облікові дані). При цьому застосунок одержує доступ до ресурсів, запускаючись у контексті стандартного користувача. За замовчуванням використовується контекст локального користувача ASPNET (для Windows) або користувача NETWORK SERVICE (для Windows Server) на комп'ютері WEB-серверу. Щоб дозволити доступ до ресурсів тільки тим користувачам, які пройшли перевірку дійсності, Необхідно дотримувати наступних правил:

– Якщо застосунок – це внутрішній мережний застосунок, налаштуйте його так, щоб воно використовувало убудовані засоби забезпечення безпеки Windows. Таким чином, облікові дані користувача для входу в систему можна використовувати для доступу до ресурсів.

– Щоб зажадати від користувача уведення своїх облікових даних, використовуйте одну зі стратегій перевірки дійсності ASP.NET.

Захист від уведення шкідливих даних

Як правило, що вводяться користувачем дані не слід уважати безпечними. Зловмисники можуть легко відправити потенційно небезпечну інформацію із клієнта в застосунок. Щоб захиститися від даних, що вводяться зловмисником, дотримуйте наступних правил:

– На сторінках ASP.NET фільтруйте дані, що вводяться користувачем, і перевіряєте їх на наявність тегів HTML, які можуть містити сценарій.

– Не відображайте уведені користувачем дані без їхньої попередньої фільтрації. До відображення неперевірених даних виконаєте HTML-кодування, для перетворення потенційно шкідливого сценарію у відображувані рядки.

– Ніколи не зберігаєте нефільтровані уведені користувачем дані в базі даних.

– Щоб прийняти від користувача HTML-дані, фільтруйте їх вручну. У фільтрі треба явно визначити прийняті дані. Не намагайтеся створити фільтр, що намагається відфільтрувати шкідливі дані, оскільки важко передбачити всі можливі типи таких даних.

– Не припускайте, що інформація, одержувана із заголовка HTTP-запиту (в об'єкті HttpRequest), безпечна. Вживайте заходів безпеки відносно рядків запитів, файлів Cookie і інших елементів. Якщо інформація, передана оглядачем серверу (інформація агента користувача), важлива для застосунку, варто пам'ятати, що вона може бути підроблена.

– По можливості не зберігаєте конфіденційні відомості в місцях, доступних з оглядача (наприклад у схованих полях або файлах Cookie). Приміром, не слід зберігати ім'я користувача або пароль у файлі Cookie.

Безпечний доступ до баз даних

Як правило, бази даних мають власні засоби забезпечення безпеки. Важливим аспектом при створенні безпечного WEB-застосунку є розробка способу доступу до бази даних з застосунку. Необхідно дотримувати наступних правил:

– Для обмеження доступу до ресурсів бази даних використовуйте її убудовані засоби безпеки. Конкретна стратегія залежить від бази даних і застосунку:

– По можливості використовуйте убудовані засоби безпеки, щоб доступ до бази даних був можливий тільки для користувачів, що пройшли перевірку дійсності Windows. Убудовані функції забезпечення безпеки більше надійна, чим передача явно зазначених облікових даних у базу даних.

– Якщо застосунок дозволяє анонімний вхід у систему, створіть єдиного користувача з дуже обмеженим числом дозволів і виконуйте запити, підключаючись як цей користувач.

– Не створюйте інструкції SQL шляхом об'єднання рядків, що містять уведені користувачем дані. Замість цього краще створити параметризований запит і використовувати уведені користувачем дані як значення цих параметрів.

– Якщо необхідно зберігати ім'я користувача й пароля, щоб використовувати їх у якості облікових даних для входу в базу даних, зберігаєте з у файлі Web.config і забезпечте безпеку цього файлі з використанням захищеної конфігурації.

Створення безпечних повідомлень про помилки

Якщо не вжити необхідних заходів обережності, зловмисний користувач може почерпнути важливі відомості про застосунок з відображуваних їм повідомлень про помилки. Необхідно дотримувати наступних правил:

– Не включайте в повідомлення про помилки інформацію, що може бути використана зловмисними користувачами (наприклад, ім'я користувача).

– Налаштуйте застосунок таким чином, щоб воно не виводило докладних відомостей про помилки для користувачів. Для відображення докладних повідомлень про помилки з метою налагодження спочатку визначите, чи є користувач локальним користувачем WEB-серверу.

– За допомогою елемента конфігурації customErrors визначите, хто може переглядати відомості про виключення із сервера.

– Для операцій, при виконанні яких можуть відбуватися помилки, таких як доступ до бази даних, напишіть код обробки помилок.

Захист конфіденційної інформації

Конфіденційна інформація – це будь-які відомості, розголошення яких небажано. Типовими прикладами важливої інформації є пароль і ключ шифрування. Якщо зловмисний користувач одержує доступ до важливої інформації, то ці дані піддаються небезпеки. Необхідно дотримувати наступних правил:

– Якщо застосунок передає конфіденційну інформацію між оглядачем і сервером, використовуйте протокол SSL.

– Для захисту важливої інформації у файлах конфігурації, таких як Web.config або Machine.config, використовуйте захищену конфігурацію.

– Щоб зберігати конфіденційну інформацію, не зберігайте її на WEB-сторінці, навіть у такій формі, у якій, на вашу думку, користувачі не зможуть неї побачити (наприклад, у серверному коді).

– Використовуйте алгоритми строгого шифрування, реалізовані в просторі імен System.Security.Cryptography.

Безпечне використання файлів Cookie

Файли Cookie – це простий і зручний спосіб надання доступу до специфічного для користувача інформації. Однак, оскільки файли Cookie відправляються на комп'ютер оглядача, вони уразливі для підміни й інших зловмисних дій. Необхідно дотримувати наступних правил:

– Не зберігаєте у файлах Cookie важливу інформацію. Наприклад, не треба, навіть тимчасово, зберігати у файлі Cookie пароль користувача. Як правило, не слід зберігати у файлах Cookie такі відомості, які при їхній крадіжці можуть порушити безпека застосунку. Замість цього зберігаєте у файлах Cookie посилання на розташування інформації на сервері.

– Установлюйте для файлів Cookie мінімальний термін дії. По можливості уникайте використання постійних файлів Cookie.

– Шифруйте інформацію у файлах Cookie.

– Властивостям Secure і HttpOnly файлу Cookie привласніть значення true.

Захист від атак типу "відмова в обслуговуванні"

Зловмисний користувач може заподіяти непряму шкоду додатку, зробивши його недоступним. Зловмисник може настільки навантажити застосунок, що воно не може обслуговувати інших користувачів або відбувається збій застосунку. Дотримуйте наступних правил.

– Використовуйте обробку помилок (наприклад, try-catch). Включите блок Finally, у який звільняються ресурси у випадку збоїти.

– Налаштуйте служби IIS для регулювання процесу, що дозволяє уникнути використання необмеженого обсягу часу ЦПУ додатком.

– Перевіряйте дані, що вводяться користувачем, на відповідність обмеженню розміру до їхнього використання або збереження.

– Установите обмеження розміру для запитів до бази даних. Наприклад, до відображення результатів запиту на WEB-сторінці ASP.NET переконаєтеся, що припустиме число записів не перевищено.

– Установите обмеження на розмір для завантаження файлів, що входять до складу застосунку. Можна задати обмеження у файлі Web.config за допомогою синтаксису, аналогічного наведеному нижче, де значення maxRequestLength дано в кілобайтах.

```
<configuration>
   <system.web>
        <httpRuntime maxRequestLength="4096" />
   </system.web>
</configuration>
```

Можна також використовувати властивість RequestLengthDiskThreshold для зниження обсягу використовуваної пам'яті при завантаженні великих обсягів і передачі форм.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Оскільки потрібно розробити просту та легку у користуванні програму, яка б виконувалась під операційною системою Windows, то для її реалізації я обрав Builder C++. Існує велике число бібліотек написаних під Builder C++ , тому це одна з важливих причин вибору мови програмування. Середовище Builder C++ досить просте в користуванні, його вихідний код значно менше по об’єму в порівнянні з Delphi чи деякими іншими програмами такого типу. Досить легко організувати взаємодію між модулями програм, об’єктно-орієнтований підхід дає можливість значно скоротити код програми, а отже і час його виконання.

На заміну старого розробленого набору елементів управління у Builder C++ інтегрована бібліотека візуальних компонентів VCL, представлених на

палітрі компонентів. Після переносу на форму методом перетягування (drag-and-drop) компоненти відразу становляться діючими об’єктами вашої програми. Окрім типізованих інтерфейсних елементів Windows (кнопки, смуги прокручування, редагуємі текстові області, прості та комбіновані списки, та інше) у бібліотеку включені елементи підтримки діалогових вікон, обслуговування баз даних та багато іншого. Можливо не тільки модифікувати поведінку існуючих компонентів, але і будувати нові.

Builder C++ підтримує останні розширення стандарту мови C++ та забезпечує швидку компіляцію та складання 32-розрядних програм для Windows. Результуючі програми оптимізовані з точки зору швидкості виконання програм та затрат пам’яті. Зручний відладгоджувальник (з асемблерним вікном, можливістю крокового виконання, завдання точок зупинки, трасування та інше) повністю інтегрований у систему проектування. Дизайнер форм, редактор коду, інспектор об’єктів та інші інструменти зостаються доступними під час виконання програми, саме через це вносити зміни до коду можна прямо у процесі відлагодження.

Дизайнер форм, Інспектор об'єктів і інші засоби залишаються доступними під час роботи програми, тому вносити зміни можна в процесі відлагодження.

Builder C++ поставляється в трьох варіантах: Standard (стандартний), Professional (для професіоналів розробників, орієнтованих на мережеву архітектуру) і Client/Server Suite (для розробки систем в архітектурі клієнт/сервер). Останні два варіанти доповнюють стандартний початковими текстами візуальних компонентів, різномасштабним словником даних, новими функціями мови запитів SQL для бази даних, пакетом підтримки систем Internet, службою моніторингу програм, а також рядом інших засобів.

Builder C++ підтримує зв'язок з різними базами даних 3-х видів: dBASE і Paradox: Sybase, Oracle, InterBase і Informix; Excel, Access, FoxPro і Btrieve. Механізм BDE (Borland Database Engine) додає обслуговуванню зв'язків з базами даних дивовижну простоту і прозорість. Провідник Database Explorer дозволяє

зображати зв'язки і об'єкти баз даних графічно. Використовуючи компоненти баз даних, я побудував електронний записник згідно таблиці dBASE за півгодини роботи на комп'ютері. Спадкоємство готових форм і їх "підгонка" під специфічні вимоги помітно скорочують тимчасові витрати на вирішення подібних завдань.

Довідкова служба Builder C++ надавала мені допомогу в цій і багатьох інших подібних ситуаціях. Є повний опис кожного управляємого компонента, включаючи списки властивостей і методів, а також численні приклади. Виклад матеріалу в книзі був значно покращуваний і систематизований завдяки відомостям, почерпнутим мною з довідкової служби.

Завдяки засобам управління проектами, двосторонній інтеграції додатку і синхронізації між засобами візуального і текстового редагування, а також вбудованому відладнику (з асемблерним вікном прокрутки, покрокового виконання, точок останову, трасуванням і тому подібне) – Builder C++ корпорації Borland надає собою вражаюче середовище розробки, яка, мабуть, витримає конкурентну боротьбу з такими модними продуктами як Developer Studio фірми Microsoft.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі.

Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Примусовий TLS – це набагато суворіший підхід до шифрування. Примусовий TLS наполягає на тому, щоб ваша розмова проводилася в приватному місці; якщо немає вільного місця, розмова просто не відбудеться.

Примусовий TLS (також відомий як «Суворий TLS» або «Обов'язковий TLS») вимагає від сервера електронної пошти зашифрованого з'єднання із сервером-отримувачем. Якщо рукостискання TLS не вдається або якщо сервер-отримувач взагалі не підтримує TLS, електронний лист не буде доставлено. З'єднання просто розривається, і відправник зазвичай отримує повідомлення про повернення, що вказує на помилку доставки. Це гарантує, що електронний лист ніколи не буде передано у вигляді звичайного тексту.

Примусовий TLS часто передбачає попередньо налаштовані політики, які іноді називають «політиками TLS» або «конекторами» в системах електронної пошти, де ви явно вказуєте, що зв'язок із певними доменами має використовувати TLS. Це поширена функція в таких службах, як Opportunistic TLS у конфігураціях Office 365, під час налаштування суворих правил потоку пошти.

Переваги та ризики примусового TLS

Переваги:

– Максимальна безпека: Гарантує, що конфіденційна інформація завжди шифрується під час передачі, усуваючи ризик її незашифрованої передачі.

– Запобігає атакам зниження версії: Зловмисник не може змусити з'єднання стати розшифрованим.

– Необхідно для дотримання вимог: Вирішально важливо для виконання суворих нормативних вимог у різних галузях, таких як HIPAA та GDPR.

Ризики:

– Можливі збої доставки: якщо сервер одержувача не підтримує потрібну версію TLS або має проблеми з конфігурацією, електронний лист не буде доставлено.

– Накладні витрати на конфігурацію: Вимагає ретельного налаштування та постійного управління, щоб забезпечити підтримку та налаштування примусового TLS усіма цільовими доменами.

Випадки використання примусового TLS:

– Охорона здоров'я (комунікації, що відповідають HIPAA): Абсолютно необхідна для передачі захищеної медичної інформації (PHI) відповідно до таких правил, як HIPAA, які вимагають безпеки даних пацієнтів.

– Фінансові установи: використовуються для безпечного зв'язку, що включає конфіденційні фінансові дані, реквізити рахунків та транзакції, щоб відповідати стандартам відповідності, таким як PCI DSS.

– Урядові установи: Для обміну секретною або конфіденційною урядовою інформацією, що вимагає суворої конфіденційності.

– Внутрішня корпоративна електронна пошта: Багато організацій застосовують примусове використання TLS між власними внутрішніми поштовими серверами або з довіреними партнерами, щоб забезпечити безпеку всіх внутрішніх комунікацій.

– Висококонфіденційна комунікація B2B: обмін конфіденційною діловою інформацією між двома компаніями, які мають взаємну угоду про забезпечення максимальної безпеки.

Приклади примусового TLS:

– Лікарня надсилає записи пацієнтів до страхової компанії, де обидві сторони домовилися використовувати примусове TLS для всіх комунікацій між своїми доменами. Якщо TLS-з’єднання не вдається встановити, запис не надсилається.

– Система електронної пошти банку часто використовує примусовий TLS

під час надсилання виписок з рахунку або конфіденційних сповіщень, гарантуючи шифрування цих листів під час передачі до вашого постачальника послуг електронної пошти (за умови, що ваш постачальник також підтримує необхідний TLS).

– Компанії, що використовують Microsoft 365, можуть налаштувати примусові правила TLS (часто їх називають «конекторами»), щоб електронні листи, надіслані до певних доменів партнерів або з них, повинні використовувати TLS, інакше вони відхиляються. Це допомагає забезпечити відповідність вимогам та надійний захист електронної пошти.

За замовчуванням Microsoft 365 (M365) не використовує примусовий TLS. Натомість він використовує опортуністичний TLS, який намагається зашифрувати електронну пошту під час передачі, якщо сервер-отримувач підтримує його. Якщо TLS недоступний, повідомлення надсилаються незашифрованими.

Однак, M365 можна налаштувати на використання примусового TLS через конектори або правила потоку пошти. За належного налаштування примусовий TLS у M365 гарантує, що електронна пошта надсилатиметься лише за умови встановлення безпечного з'єднання TLS, інакше доставка не вдасться. Коротше кажучи, примусовий TLS – це налаштовуваний параметр, а не поведінка за замовчуванням.

Gmail не використовує примусовий TLS за замовчуванням. Gmail покладається на опортуністичний TLS, тобто намагатиметься зашифрувати електронну пошту під час передачі, якщо сервер-отримувач підтримує TLS; якщо TLS недоступний, повідомлення доставляється незашифрованим.

Хоча Google підтримує примусове використання TLS для певних доменів за допомогою розширених налаштувань Gmail (переважно в Google Workspace), примусове використання TLS не є поведінкою за замовчуванням і має бути налаштоване явно. Крім того, коли TLS не підтримується, а примусове використання TLS налаштоване, альтернативою є те, що повідомлення не

надсилатимуться, і цю проблему може вирішити безпечне рішення для електронної пошти Zivver.

Як опортуністичний TLS, так і примусовий TLS відіграють певну роль у безпеці електронної пошти. Однак для організацій, які мають справу з конфіденційними даними та дотримуються вимог до дотримання вимог, примусовий TLS є явним переможцем, оскільки він гарантує шифрування під час передачі. Хоча опортуністичний TLS пропонує підхід «найкращих зусиль», він не усуває ризик незашифрованої передачі, якщо умови не виконуються; у традиційних рішеннях електронної пошти, якщо сервер одержувача не підтримує TLS, ваше повідомлення не може бути надіслано.

Для справді надійної безпеки електронної пошти, яка виходить за рамки базових можливостей TLS, спеціалізоване рішення для шифрування електронної пошти, розширює поштові клієнти, такі як Microsoft 365 та Gmail, за допомогою розширених протоколів шифрування, гарантуючи захист конфіденційних листів та їх можливість читання на безпечному веб-порталі, захищеному потужною двофакторною автентифікацією (2FA). Окрім розширеного шифрування, додатково захищає конфіденційні листи за допомогою зручних функцій запобігання втраті даних, допомагаючи запобігти людським помилкам – усі ці важливі елементи захисту вашої інформації.

3.2 Розробка структурної схеми

Опортуністичний TLS – це стандартна конфігурація безпеки транспортування пошти на всіх агентах передачі повідомлень (MTA) та поштових серверах. Сервери відправника та одержувача пошти домовлятимуться один з одним про те, яке шифрування вони можуть використовувати. Буде використано найбезпечніше з’єднання, яке можуть підтримувати обидві сторони. Це означає, що пошту можна надсилати взагалі без шифрування. Але ви можете це змінити.

Дивно, що ми посилюємо безпеку наших операційних систем і

хвилюємося про те, як захищений Microsoft Teams. Іноді ми також звинувачуємо у будь-якому витоку даних прислів'я-стажера (що, до речі, неправильно). Але коли справа доходить до пошти, організації досить погано налаштовують належну автентифікацію пошти. Це все про SPF, DKIM, DMARC, SRS та ARC, я продовжую працювати над своїми соціальними мережами. Особливо є простір для вдосконалення, коли домен організації використовується в сторонніх поштових продуктах.

Шифрування повідомлень

Звичайно, ви можете підписати або зашифрувати свою електронну пошту за допомогою S/MIME, PGP або Microsoft Purview Mail Encryption. Це означає, що окреме поштове повідомлення було зашифровано, незалежно від того, як воно передається. Але, на мою скромну думку, це не дуже зручно для користувача. Вам потрібно попереднє налаштування, таке як отримання сертифікатів та обмін ключами. Шифрування повідомлень забезпечує безпеку, але часто конфліктує з процесами відповідності (наприклад, збереження/архівування). Purview забезпечує захист лише на стороні відправника (для нових листів).

Так, існують шлюзові рішення, але вони часто суттєво збільшують вартість (ліцензії на робоче місце, обслуговування тощо). Видима та прихована вартість електронної пошти зростатиме з кожним додаванням. Кілька причин, чому пошта є привабливим рішенням, – це високий рівень впровадження та сумісності. Вона також відносно проста у використанні та має відносно низьку вартість з майже миттєвою доставкою. (Справедливості заради, економічність пошти – це обґрунтоване припущення. Вона достатньо хороша та була першою, ймовірно, також є вагомими причинами).

Шифрування транспорту

Не зрозумійте мене неправильно. Шифрування повідомлень, безумовно, має свої варіанти використання. Мені здається, що воно іноді заважає думати про інші рішення чи практики. Наприклад, чи варто мені взагалі надсилати цю цінну інформацію, якщо вона потребуватиме шифрування повідомлень? Чи справді

пошта є найкращим способом безпечного спілкування?

Хоча я починав із шифрування повідомлень, транспортне шифрування насправді є основною та неявною формою шифрування. Це означає, що наразі майже завжди використовується транспортне шифрування завдяки опортуністичному TLS. У більшості перевірених мною орендарів діапазон «Без TLS» знаходиться в межах 1%. І цей 1% «імовірність» може бути причиною для організації використовувати шифрування повідомлень, зменшуючи потенційні прогалини за допомогою транспортного шифрування та запобігаючи перехопленню повідомлень.

Як варіант, організації-партнери використовують обов'язкові TLS - з'єднання. Вони налаштовуються для забезпечення наявності шифрування транспорту перед надсиланням будь-якої пошти. Часто вони також мають форму автентифікації за IP-адресою або сертифікатом. Проблема, яку я бачу, полягає в тому, що вони вимагають спеціальної конфігурації та налаштування для кожного поштового домену, що збільшує витрати, а іноді вимагає контакту з вузлом для повідомлення правильної кінцевої точки SMTP. MTA-STS є покращенням, оскільки він певною мірою запобігає атакам Advisary-in-the-Middle, оскільки публікує кінцеву точку MX у файлі конфігурації на захищеному веб-сайті з тим самим доменом організації.

Найкращим рішенням для безпеки транспорту на сьогодні є DANE. Він забезпечує навіть більший рівень безпеки, ніж обов'язковий TLS, оскільки значною мірою залежить від DNSSEC. На жаль, не всі організації можуть використовувати DNSSEC. (Azure DNS мав його в попередньому перегляді, але цього ярлика більше немає?). Особливо, коли Exchange Online підтримуватиме обов'язковий DANE пізніше у 2025 році. Це примусово використовуватиме DANE на основі клієнта або домену та не дозволить переходу на MTA-STS або опортуністичний TLS. Немає потреби в обов'язковому TLS або навіть маршрутизації через захищені приватні мережі.

Але і MTA-STS, і DANE страждають від тих самих проблем, що й інші

акроніми, а саме від їхнього впровадження. Іноді це пов'язано з технічними причинами, але частіше з усвідомленням або розумінням їхніх переваг порівняно з тим, що використовується за замовчуванням, тобто опортуністичним TLS.

Примусове використання TLS в Exchange Online

Щоб примусово використовувати TLS для підключення SMTP за замовчуванням, потрібно налаштувати як вхідний, так і вихідний конектор. Базові однорядкові інструкції Exchange Online PowerShell у найпростішій формі (без виконання перевірки):

```
New-OutboundConnector -ConnectorType Partner -RecipientDomains * -TlsSettings EncryptionOnly -UseMXRecord $True -Name "Outbound Forced TLS"
New-InboundConnector -ConnectorType Partner -SenderDomains * -TlsSettings EncryptionOnly
```

Є три основні висновки: тип конектора – Партнер. Ви налаштовуєте домени одержувача та відправника за допомогою символу підстановки *. І останній етап – лише шифрування, а не перевірка сертифіката. Більш специфічні конектори (наприклад, ті, що мають IP-адреси та/або повні доменні імена) замінять ці конектори. Ви можете мати певні конектори, які не потребують TLS, якщо вам це потрібно для прямого надсилання або ретрансляції SMTP.

Якщо ваша організація надає пріоритет отриманню пошти перед шифруванням транспортування, ви можете розглянути можливість початку роботи з вихідного конектора. У поєднанні з оповіщенням про чергу пошти або використанням вбудованих звітів про повідомлення ви можете відстежувати ситуацію та діяти відповідно.

Більш цілісний підхід до безпеки (пошти)

Організації повинні регулярно перевіряти весь свій підхід до обміну інформацією з іншими сторонами. Пошта – це просто, і майже завжди вона вже доступна. Але за останні кілька років з'явилися інші продукти або їх удосконалили до такого стану, що це може бути найкращим рішенням для цієї роботи. Вона також може бути дешевшою, простішою в управлінні та навіть безпечнішою.

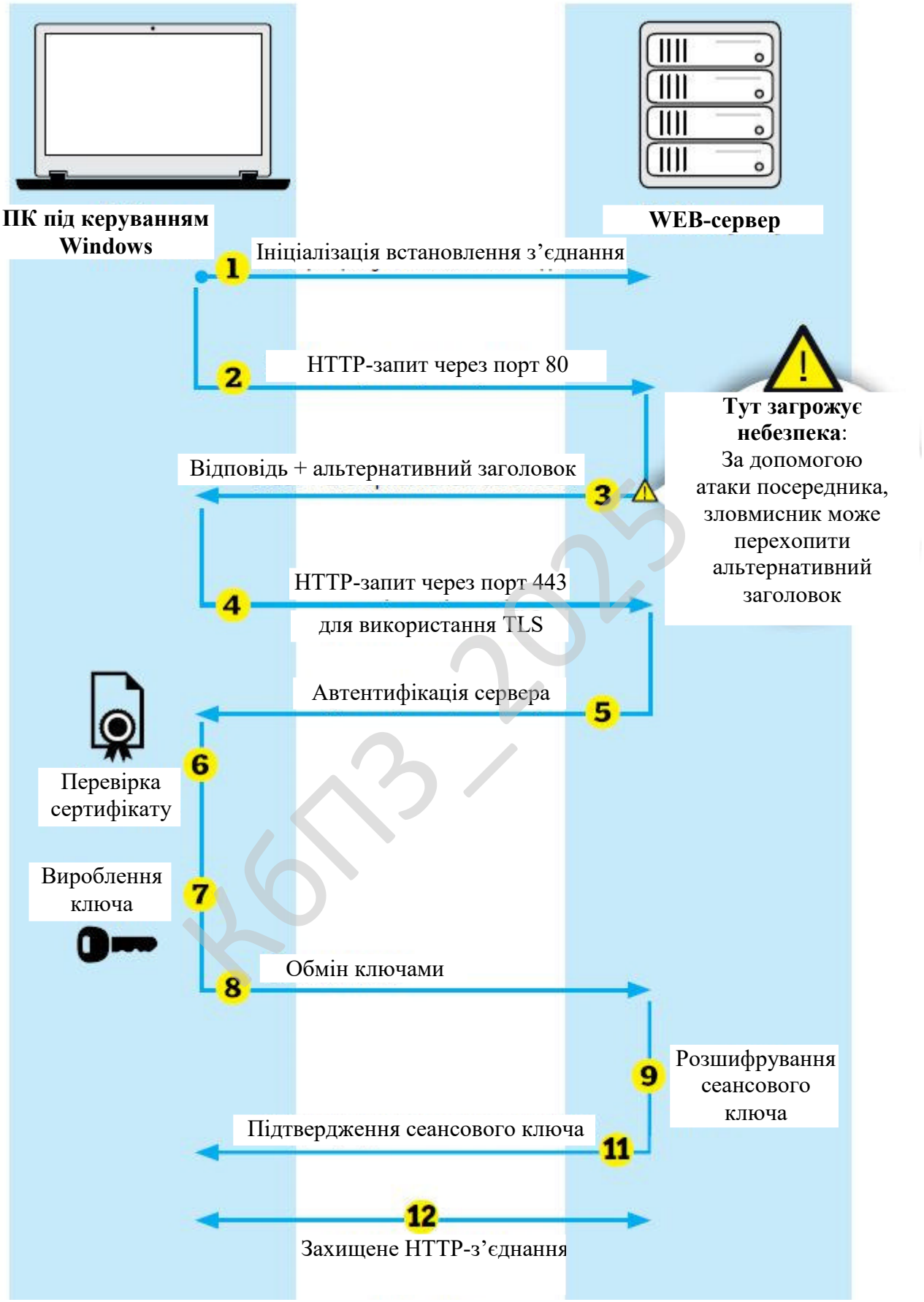

Рисунок 3.1 – Структурна схема системи

Я б зосередився на транспортному шифруванні, такому як примусове використання TLS та впровадження як MTA-STS, так і DANE. Особливо (обов'язкове) DANE є значним покращенням і може бути причиною для переоцінки використання шифрування повідомлень у певних сценаріях.

Опортуністичне шифрування

Опортуністичне шифрування починається з незашифрованого запиту. Шифрування встановлюється тільки тоді, коли сервер відправляє альтернативний заголовок із пропозицією зашифрувати канал.

Робота системи проілюстрована структурною схемою, відображеної на рисунку 3.1.

Тут загрожує небезпека: за допомогою атаки посередника зловмисник може перехопити альтернативний заголовок.

Порт 80 – **WWW-сервер**. Показує присутність http-сервера в системі. За допомогою WWW-сервісу можна довідатися назву й версію програмного забезпечення встановленого на WEB-сервері.

Порт 443 – **Протокол HTTPS** (**SSL**). Варіант безпечного протоколу **HTTP**.

3.3 Розробка функціональної схеми

Три методи звертання до WEB-сайтів

Браузер і WEB-сайт обмінюються даними або по незашифрованому каналу за протоколом HTTP, або по зашифрованому каналу по HTTPS, або за допомогою нового методу шифрування Opportunistic Encryption по HTTP.

HTTP

HTTP – широко розповсюджений протокол передачі даних, споконвічно призначений для передачі гіпертекстових документів (тобто документів, які можуть містити посилання, що дозволяють організувати перехід до інших документів).

Абревіатура HTTP розшифровується як HyperText Transfer Protocol, «протокол передачі гіпертексту». У відповідності зі специфікацією OSI, HTTP є протоколом прикладного (верхнього, 7-го) рівня. Актуальна на даний момент версія протоколу, HTTP 1.1, описана в специфікації RFC 2616.

Протокол HTTP припускає використання клієнт-серверної структури передачі даних. Клієнтський застосунок формує запит і відправляє його на сервер, після чого серверне програмне забезпечення обробляє даний запит, формує відповідь і передає його назад клієнтові. Після цього клієнтський застосунок може продовжити відправляти інші запити, які будуть оброблені аналогічним образом.

Завдання, що традиційно вирішується за допомогою протоколу HTTP – обмін даними між користувальницьким додатком, що здійснює доступ до WEB-ресурсів (звичайно це WEB-браузер) і WEB-сервером. На даний момент саме завдяки протоколу HTTP забезпечуються робота Всесвітньої павутини.

Також HTTP часто використовується як протокол передачі інформації для інших протоколів прикладного рівня, таких як SOAP, XML-RPC і WebDAV. У такому випадку говорять, що протокол HTTP використовується як «транспорт».

API багатьох програмних продуктів також має на увазі використання HTTP для передачі даних – самі дані при цьому можуть мати будь-який формат, наприклад, XML або JSON.

Як правило, передача даних за протоколом HTTP здійснюється через TCP/IP-з'єднання. Серверне програмне забезпечення при цьому звичайно використовує TCP-порт 80 (і, якщо порт не зазначений явно, те звичайно клієнтське програмне забезпечення за замовчуванням використовує саме 80-й порт для що відкриваються HTTP-з'єднань), хоча може використовувати й будь-який іншої.

Як відправити HTTP-запит?

Найпростіший спосіб розібратися із протоколом HTTP – це спробувати звернутися до якому-небудь WEB-ресурсу вручну.

Припустимо, що він увів в адресному рядку наступне:

http://it-kntu.kr.ua/

Відповідно вам, як WEB-браузеру, тепер необхідно підключитися до WEB-серверу за адресою it-kntu.kr.ua.

Для цього ви можете скористатися будь-якою підходящою утилітою командного рядка. Наприклад, telnet:

```
telnet it-kntu.kr.ua 80
```

Відразу уточню, що якщо ви раптом передумаєте, то натисніть Ctrl + «]», і потім уведення – це дозволить вам закрити HTTP-з'єднання. Крім telnet можете спробувати nc (або ncat).

Після того, як ви підключитеся до сервера, потрібно відправити HTTP-запит. Це, до речі, дуже легко – HTTP-запити можуть складатися всього із двох рядків.

Для того, щоб сформувати HTTP-запит, необхідно скласти стартовий рядок, а також задати принаймні один заголовок – це заголовок Host, що є обов'язковим, і повинен бути присутнім у кожному запиту. Справа в тому, що перетворення доменного ім'я в IP-адресу здійснюється на стороні клієнта, і, відповідно, коли ви відкриваєте TCP-з'єднання, те вилучений сервер не має ніяку інформацію про те, який саме адреса використовувалася для з'єднання: це міг бути, наприклад, адреса it-kntu.kr.ua або m.it-kntu.kr.ua – і у всіх цих випадках відповідь може відрізнятися. Однак фактично мережне з'єднання у всіх випадках відкривається з вузлом 212.24.43.44, і навіть якщо спочатку при відкритті з'єднання був задана не ця IP-адреса, а яке-небудь доменне ім'я, то сервер про цьому ніяк не інформується – і саме тому ця адреса необхідно передати в заголовку Host.

Стартова (початкова) рядок запиту для HTTP 1.1 складається за наступною схемою:

Метод URI HTTP/Версія

Наприклад (такий стартовий рядок може вказувати на те, що запитується головна сторінка сайту):

```
GET / HTTP/1.1
```

Метод (в англомовній тематичній літературі використовується слово method, а також іноді слово verb – «дієслово») являє собою послідовність із будь-яких символів, крім керуючих і роздільників, і визначає операцію, яку потрібно здійснити із зазначеним ресурсом. Специфікація HTTP 1.1 не обмежує кількість різних методів, які можуть бути використані, однак з метою відповідності загальним стандартам і збереження сумісності з максимально широким спектром програмного забезпечення як правило використовуються лише деякі, найбільш стандартні методи, зміст яких однозначно розкритий у специфікації протоколу.

URI (Uniform Resource Identifier, уніфікований ідентифікатор ресурсу) – шлях до конкретного ресурсу (наприклад, документа), над яким необхідно здійснити операцію (наприклад, у випадку використання методу GET мається на увазі одержання ресурсу). Деякі запити можуть не ставитися до якого-небудь ресурсу, у цьому випадку замість URI у стартовий рядок може бути додана зірочка (астериск, символ «*»). Наприклад, це може бути запит, що ставиться до самому WEB-серверу, а не якому-небудь конкретному ресурсу. У цьому випадку стартовий рядок може виглядати так:

```
OPTIONS * HTTP/1.1
```

Версія визначає, відповідно до якої версії стандарту HTTP складений запит. Вказується як два числа, розділених точкою (наприклад **1.1**).

Для того, щоб звернутися до WEB-сторінки по певній адресі (у цьому випадку шлях до ресурсу – це «/»), нам варто відправити наступний запит:

```
GET / HTTP/1.1
Host: it-kntu.kr.ua
```

При цьому враховуйте, що для переносу рядка варто використовувати символ повернення каретки (Carriage Return), за яким треба символ перекладу рядка (Line Feed). Після оголошення останнього заголовка послідовність символів для переносу рядка додається двічі.

Втім, у специфікації HTTP рекомендується програмувати HTTP-сервер таким чином, щоб при обробці запитів у якості міжрядкового роздільника сприймався символ LF, а попередній символ CR, при наявності такого,

ігнорувався. Відповідно, на практиці більша частина серверів коректно обробить і такий запит, де заголовки відділені символом LF, і він же двічі доданий після оголошення останнього заголовка.

Якщо ви хочете відправити запит у точній відповідності зі специфікацією, можете скористатися керуючими послідовностями \r і \n:

```
echo -en "GET / HTTP/1.1\r\nHost: it-kntu.kr.ua\r\n\r\n" | ncat it-kntu.kr.ua 80
```

Як прочитати відповідь?

Стартовий рядок відповіді має наступну структуру:

```
HTTP/Версія Код стану Пояснення
```

Версія протоколу тут задається так само, як у запиту.

Код стану (Status Code) – три цифри (перша з яких указує на клас стану), які визначають результат здійснення запиту. Наприклад, у випадку, якщо був використаний метод GET, і сервер надає ресурс із зазначеним ідентифікатором, то такий стан задається за допомогою коду 200. Якщо сервер повідомляє про те, що такого ресурсу не існує – 404. Якщо сервер повідомляє про те, що не може надати доступ до даного ресурсу через відсутність необхідних привілеїв у клієнта, то використовується код 403. Специфікація HTTP 1.1 визначає 40 різних кодів HTTP, а також допускається розширення протоколу й використання додаткових кодів станів.

Пояснення до коду стану (Reason Phrase) – текстове (але не включає символи CR і LF) пояснення до коду відповіді, призначено для спрощення читання відповіді людиною. Пояснення може не враховуватися клієнтським програмним забезпеченням, а також може відрізнятися від стандартного в деяких реалізаціях серверного ПО.

Після стартового рядка випливають заголовки, а також тіло відповіді. Наприклад:

```
HTTP/1.1 200 OK
Server: nginx/1.2.1
Date: Sat, 08 Mar 2014 22:53:46 GMT
Content-Type: application/ octet-stream
```

```
Content-Length: 7
Last-Modified: Sat, 08 Mar 2014 22:53:30 GMT
Connection: keep-alive
Accept-Ranges: bytes
Wisdom
```

Тіло відповіді треба через два переноси рядка після останнього заголовка. Для визначення закінчення тіла відповіді використовується значення заголовка Content-**Length** (у цьому випадку відповідь містить 7 восьмеричних байтів: слово «Wisdom» і символ переносу рядка).

Але от по тому запиту, що ми склали раніше, WEB-сервер поверне відповідь не з кодом 200, а з кодом 302. У такий спосіб він повідомляє клієнта про те, що звертатися до даного ресурсу на даний момент потрібно по іншій адресі.

Дивитеся самі:

```
HTTP/1.1 302 Moved Temporarily
Server: nginx
Date: Sat, 08 Mar 2014 22:29:53 GMT
Content-Type: text/html
Content-Length: 154
Connection: keep-alive
Keep-Alive: timeout=25
Location: http:// it-kntu.kr.ua/users/kovalenko/
<html>
<head><title>302 Found</title></head>
<body bgcolor="white">
<center><h1>302 Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

У заголовку Location передана нова адреса. Тепер URI (ідентифікатор ресурсу) змінився на /users/kovalenko/, а звертатися потрібно цього разу до сервера за адресою it-kntu.kr.ua (втім, у цьому випадку це той же самий сервер), і його ж указувати в заголовку Host.

Тобто:

```
GET /users/kovalenko/ HTTP/1.1
Host: it-kntu.kr.ua
```

У відповідь на цей запит WEB-сервер It-kntu.kr.ua уже видасть відповідь із кодом 200 і досить великий документ у форматі HTML.

Якщо ви вже встигли вжитися в роль, то можете тепер прочитати отриманий від сервера HTML-код, взяти олівець і блокнот, і намалювати профайл Kovalenko – у принципі, саме цим би на вашім місці браузер зараз і зайнявся.

Безпека

Сам по собі протокол HTTP не припускає використання шифрування для передачі інформації. Проте, для HTTP є розповсюджене розширення, що реалізує впакування переданих даних до криптографічного протоколу **SSL** або **TLS**.

Назва цього розширення – **HTTPS** (HyperText Transfer Protocol Secure). Для HTTPS-з'єднань звичайно використовується TCP-порт 443. HTTPS широко використовується для захисту інформації від перехоплення, а також, як правило, забезпечує захист від атак виду in-the-middle – у тому випадку, якщо сертифікат перевіряється на клієнті, і при цьому приватний ключ сертифіката не був скомпрометований, користувач не підтверджував використання непідписаного сертифіката, і на комп'ютері користувача не були впроваджені сертифікати центра сертифікації зловмисника.

На даний момент HTTPS підтримується всіма популярними WEB-браузерами.

Протокол HTTP припускає досить велика кількість можливостей для розширення. Зокрема, специфікація HTTP 1.1 припускає можливість використання заголовка Upgrade для перемикання на обмін даними за іншим протоколом. Запит з таким заголовком відправляється клієнтом. Якщо серверу потрібно зробити перехід на обмін даними за іншим протоколом, то він може повернути клієнтові відповідь зі статусом «426 Upgrade Required», і в цьому випадку клієнт може відправити новий запит, уже із заголовком Upgrade.

Така можливість використовується, зокрема, для організації обміну даними за протоколом WebSocket (протокол, описаний у специфікації RFC 6455, що дозволяє обом сторонам передавати дані в потрібний момент, без

відправлення додаткових HTTP-запитів): стандартне «рукостискання» (handshake) зводиться до відправлення HTTP-запиту із заголовком Upgrade, що має значення «websocket», на який сервер повертає відповідь зі станом «101 Switching Protocols», і далі будь-яка сторона може почати передавати дані уже за протоколом WebSocket.

На даний момент існують і інші протоколи, призначені для передачі WEB-змісту. Зокрема, протокол **SPDY** (вимовляється як англійське слово speedy, не є абревіатурою) є модифікацією протоколу HTTP, ціль якої – зменшити затримки при завантаженні WEB-сторінок, а також забезпечити додаткову безпеку.

Збільшення швидкості забезпечується за допомогою стиску, пріоритизації й мультиплексування додаткових ресурсів, необхідних для WEB-сторінки, щоб всі дані можна було передати в рамках одного з'єднання.

Опублікована у листопаді 2012 року чернетка специфікації протоколу HTTP 2.0 (наступна версія протоколу HTTP після версії 1.1, остаточна специфікація для якої була опублікована в 1999) базується на специфікації протоколу SPDY.

Багато архітектурних рішень, використовувані в протоколі SPDY, а також в інших запропонованих реалізаціях, які робоча група httpbis розглядала в ході підготовки чернетки специфікації HTTP 2.0, уже раніше були отримані в ході розробки протоколу HTTP-NG, однак роботи над протоколом HTTP-NG були припинені в 1998.

На даний момент підтримка протоколу SPDY є в браузерах Firefox, Chromium/Chrome, Opera, Internet Exporer і Amazon Silk.

URL

Серцевиною WEB-спілкування є запит, що відправляється через Єдиний покажчик ресурсів (URL). Я впевнений, що ви вже знаєте, що таке URL адреса, однак для повноти картини, вирішив все-таки сказати пари слів. Структура URL дуже проста й складається з наступних компонентів:

Протокол може бути як http для звичайних з'єднань, так і https для більше безпечного обміну даними. Порт за замовчуванням – 80. Далі треба шлях до ресурсу на сервері й ланцюжок параметрів.

Методи

За допомогою URL, ми визначаємо точну назву хосту, з яким хочемо спілкуватися, однак яка дія нам потрібно зробити, можна повідомити тільки за допомогою HTTP методу. Звичайно ж існує кілька видів дій, які ми можемо зробити. В HTTP реалізовані самі потрібні, підходящі під потреби більшості додатків.

Існуючі методи:

GET: одержати доступ до існуючого ресурсу. В URL перерахована вся необхідна інформація, щоб сервер зміг знайти й повернути як відповідь шуканий ресурс.

POST: використовується для створення нового ресурсу. POST запит звичайно містить у собі всю потрібну інформацію для створення нового ресурсу.

PUT: обновити поточний ресурс. PUT запит містить обновлювані дані.

DELETE: служить для видалення існуючого ресурсу.

Дані методи самі популярні й найчастіше використовуються різними інструментами й фреймворками. У деяких випадках, PUT і DELETE запити відправляються за допомогою відправлення POST, у змісті якого зазначене дія, яку потрібно зробити з ресурсом: створити, обновити або видалити.

Також HTTP підтримує й інші методи:

HEAD: аналогічний GET. Різниця в тому, що при даному виді запиту не передається повідомлення. Сервер одержує тільки заголовки. Використовується, приміром, для того щоб визначити, чи був змінений ресурс.

TRACE: під час передачі запит проходить через безліч точок доступу й проксі серверів, кожний з яких вносить свою інформацію: IP, DNS. За допомогою даного методу, можна побачити всю проміжну інформацію.

OPTIONS: використовується для визначення можливостей сервера, його параметрів і конфігурації для конкретного ресурсу.

Коди стану

У відповідь на запит від клієнта, сервер відправляє відповідь, що містить, у тому числі, і код стану. Даний код несе в собі особливий зміст для того, щоб клієнт міг більш ясно зрозуміти, як інтерпретувати відповідь:

1xx: Інформаційні повідомлення

Набір цих кодів був уведений в HTTP/1.1. Сервер може відправити запит виду: Expect: 100-continue, що означає, що клієнт ще відправляє частину, що залишилася, запиту. Клієнти, що працюють із HTTP/1.0 ігнорують дані заголовки.

2xx: Повідомлення про успіх

Якщо клієнт одержав код із серії 2xx, то запит пішов успішно. Найпоширеніший варіант – це 200 OK. При GET запиту, сервер відправляє відповідь у тілі повідомлення. Також існують і інші можливі відповіді:

– **202 Accepted**: запит прийнятий, але може не містити ресурс у відповіді. Це корисно для асинхронних запитів на стороні сервера. Сервер визначає, відправити чи ресурс ні.

– **204 No Content**: у тілі відповіді немає повідомлення.

– **205 Reset Content**: вказівка серверу про скидання подання документа.

– **206 Partial Content**: відповідь містить тільки частина контенту. У додаткових заголовках визначається загальна довжина контенту й інша інформація.

3xx: Перенапрямок

Своєрідне повідомлення клієнтові про необхідність зробити ще одна дія. Найпоширеніший варіант застосування: перенаправляти клієнт на іншу адресу.

– **301 Moved Permanently**: ресурс тепер можна знайти по іншому URL адресі.

– **303 See Other**: ресурс тимчасово можна знайти по іншому URL адресі. Заголовок Location містить тимчасовий URL.

– **304 Not Modified**: сервер визначає, що ресурс не був змінений і клієнтові потрібно задіяти закешовану версію відповіді. Для перевірки ідентичності інформації використовується ETag (геш Сутності – Enttity Tag);

4xx: Клієнтські помилки

Даний клас повідомлень використовується сервером, якщо він вирішив, що запит був відправлений з помилкою. Найпоширеніший код: 404 Not Found. Це означає, що ресурс не знайдений на сервері. Інші можливі коди:

– **400 Bad Request**: питання було сформовано невірно.

– **401 Unauthorized**: для здійснення запиту потрібна автентифікація. Інформація передається через заголовок Authorization.

– **403 Forbidden**: сервер не відкрив доступ до ресурсу.

– **405 Method Not Allowed**: невірний HTTP метод був задіяний для того, щоб одержати доступ до ресурсу.

– **409 Conflict**: сервер не може до кінця обробити запит, тому що намагається змінити більше нову версію ресурсу. Це часто відбувається при PUT запитах.

5xx: Помилки сервера

Ряд кодів, які використовуються для визначення помилки сервера при обробці запиту. Найпоширеніший: 500 Internal Server Error. Інші варіанти:

– **501 Not Implemented**: сервер не підтримує запитувану функціональність.

– **503 Service Unavailable**: це може трапитися, якщо на сервері відбулася помилка або він перевантажений. Звичайно в цьому випадку, сервер не відповідає, а час, даний на відповідь, минає.

Формати повідомлень запиту/відповіді

На наступному зображенні ви можете побачити схематично оформлений процес відправлення запиту клієнтом, обробка й відправлення відповіді сервером.

Давайте подивимося на структуру переданого повідомлення через HTTP:

```
1  message = < start-line>
2            *(< message-header>)
```

```
3            CRLF
4             [< message-body>]
5
6  < start-line> = Request-Line | Status-Line
7  < message-header> = Field-Name ':' Field-Value
```

Між заголовком і тілом повідомлення повинна обов'язково бути присутнім порожній рядок. Заголовків може бути декілька:

– Загальні заголовки.

– Заголовки запиту.

– Заголовки відповіді.

– Заголовки сутностей.

Тіло відповіді може містити повну інформацію або її частину, якщо активовано відповідну можливість (Transfer-Encoding: chunked). HTTP/1.1 також підтримує заголовок Transfer-Encoding.

Загальні заголовки

От кілька видів заголовків, які використовуються як у запитах, так і у відповідях:

```
1  general-header = Cache-Control
2                 | Connection
3                 | Date
4                 | Pragma
5                 | Trailer
6                 | Transfer-Encoding
7                 | Upgrade
8                 | Via
9                 | Warning
```

Заголовок via використовується в запиту типу TRACE, і обновляється всіма проксі-серверами.

Заголовок Pragma використовується для перерахування власних заголовків. Приміром, Pragma: no-cache – це те ж саме, що Cache-Control: no-cache..

Заголовок Date використовується для зберігання дати й часу запиту/відповіді.

Заголовок Upgrade використовується для зміни протоколу.

Transfer-Encoding призначається для поділу відповіді на кілька фрагментів за допомогою Transfer-Encoding: chunked. Це нововведення версії HTTP/1.1.

Заголовки сутностей

У заголовках сутностей передається позначка-інформація контенту:

```
entity-header  = Allow
               | Content-Encoding
               | Content-Language
               | Content-Length
               | Content-Location
               | Content-MD5
               | Content-Range
               | Content-Type
               | Expires
               |         Last-
Modified
```

Всі заголовки із префіксом Content надають інформацію про структуру, кодування й розмір тіла повідомлення.

Заголовок Expires містить час і дату витікання сутності. Значення “never expires” означає час + 1 код із сучасний момент. Last-Modified містить час і дату останньої зміни сутності.

За допомогою даних заголовків, можна задати потрібну для ваших завдань інформацію.

Формат запиту

Запит виглядає приблизно так:

```
Request-Line = Method SP URI SP HTTP-Version CRLF
Method = "OPTIONS"
       | "HEAD"
       | "GET"
       | "POST"
```

```
6         | "PUT"
7         | "DELETE"
8         | "TRACE"
```

SP – це роздільник між токенами. Версія HTTP вказується в HTTP-Version. Реальний запит виглядає так:

```
1  GET /articles/ http-basics HTTP/1.1
2  Host: www.articles.com
3  Connection: keep-alive
4  Cache-Control: no-cache
5  Pragma: no-cache
6  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

Список можливих заголовків запиту:

```
01  request-header = Accept
02                 | Accept-Charset
03                 | Accept-Encoding
04                 | Accept-Language
05                 | Authorization
06                 | Expect
07                 | From
08                 | Host
09                 | If-Match
10                 | If-Modified-Since
11                 | If-None-Match
12                 | If-Range
13                 | If-Unmodified-Since
14                 | Max-Forwards
15                 | Proxy-Authorization
16                 | Range
17                 | Referer
18                 | TE
19                 | User-Agent
```

У заголовку Accept визначаються підтримувані mime типи, мову, кодування символів. Заголовки From, Host, Referer і User-Agent містять

інформацію про клієнта. Префікси If- призначені для створення умов. Якщо умова не пройшла, то виникне помилка 304 Not Modified.

Формат відповіді

Формат відповіді відрізняється тільки статусом і рядом заголовків. Статус виглядає так:

```
1  Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

– HTTP версія.

– Код статусу.

– Повідомлення статусу, зрозуміле для людини.

Звичайний статус виглядає приблизно так:

```
1  HTTP/1.1 200 OK
```

Заголовки відповіді можуть бути наступними:

```
1  response-header = Accept-Ranges
2                  | Age
3                  | ETag
4                  | Location
5                  | Proxy-Authenticate
6                  | Retry-After
7                  | Server
8                  | Vary
                   |          WWW-
9  Authenticate
```

– Age час у секундах, коли повідомлення було створено на сервері.

– ETag MD5 сутності для перевірки змін і модифікацій відповіді.

– Location використовується для перенапрямку й містить нову URL адреса.

– Server визначає сервер, де була сформовано відповідь.

При стандартному запиту по HTTP браузер звертається до сервера й відправляє запит по незашифрованому каналу, на який сервер відповідає без автентифікації також у відкритому виді.

HTTPS

HTTPS (англ. HyperText Transfer Protocol Secure) – розширення протоколу HTTP, що підтримує шифрування. Дані, передані за протоколом HTTPS, «упаковуються» до криптографічного протоколу SSL або TLS. На відміну від HTTP, для HTTPS за замовчуванням використовується TCP-порт 443 [1].

Netscape Communications створили HTTPS в 1994 році для свого браузера Netscape Navigator . Спочатку, **HTTPS** був використаний з SSL-протоколом. Оскільки SSL перетворилася в TLS (Transport Layer Security), поточна версія **HTTPS** була формально визначена стандартному RFC 2818 у травні 2000 року [2].

Принцип роботи

Власне кажучи, HTTPS це не окремий протокол, а звичайний HTTP працюючий через SSL [3] або TLS. Щоб підготувати WEB-сервер для прийняття https-транзакцій адміністратор повинен створити сертифікат з відкритим ключем для WEB-серверу. Ці сертифікати можуть бути створені на UNIX-сервері такими програмами, як наприклад OpenSSL.

Про переваги й технології застосовуваних в HTTPS

Перед тим як ми поринемо в те, як це працює, давайте коротко поговоримо про те, чому так важливо захищати Інтернет-з'єднання й від чого захищає HTTPS.

Коли браузер робить запит до Вашого улюбленого WEB-сайту, цей запит повинен пройти через безліч різних мереж, кожна з яких може бути потенційно використана для прослуховування або для втручання у встановлене з'єднання.

З вашого власного комп'ютера на інші комп'ютери вашої локальної мережі, через роутери й світчи, через вашого провайдеру й через безліч інших проміжних провайдерів – величезна кількість організацій ретранслює ваші дані. Якщо зловмисник виявиться хоча б в одній з них – у нього їсти можливість подивитися, які дані передаються.

Як правило, запити передаються за допомогою звичайного HTTP, у якому й запит клієнта, і відповідь сервера передаються у відкритому виді. І є безліч вагомих аргументів, чому HTTP не використовує шифрування за замовчуванням:

– Для цього потрібно більше обчислювальних потужностей.

– Передається більше даних.

– Не можна використовувати кешування.

Але в деяких випадках, коли по каналі зв'язку передається винятково важлива інформація (така як, паролі або дані кредитних карт), необхідно забезпечити додаткові заходи, що запобігають прослуховування таких з'єднань.

Secure Socket Layer

SSL – це скорочення від Secure Socket Layer – це стандартна інтернет технологія безпеки, що використовується, щоб забезпечити зашифроване з'єднання між WEB-сервером (сайтом) і браузером. SSL сертифікат дозволяє нам використовувати https протокол. Це безпечне з'єднання, що гарантує, що інформація яка передається від вашого браузера на сервер залишається приватної; тобто захищеної від хакерів або кожного, хто хоче украсти інформацію. Один з найпоширеніших прикладів використання SSL – це захист клієнта під час онлайн транзакції (покупки товару, оплати). TLS – гібридна криптографічна система. Це означає, що вона використовує кілька криптографічних підходів:

– Асиметричне шифрування (криптосистема з відкритим ключем) для генерації загального секретного ключа й автентифікації (тобто посвідчення в тому, що ви – той за кого себе видаєте).

– Симетричне шифрування, що використовує секретний ключ для подальшого шифрування запитів і відповідей.

Симетричне шифрування

Обмін ключами відбувається всього один раз за сесію, під час установлення з'єднання. Коли ж сторони вже домовилися про секретний ключ, клієнт-серверна взаємодія відбувається за допомогою симетричного шифрування,

що набагато ефективніше для передачі інформації, оскільки не потрібно додаткові витрати на підтвердження [4].

Криптосистема з відкритим ключем

Криптосистема з відкритим ключем – це різновид криптографічної системи, коли в кожної сторони є й відкритий, і закритий ключ, математично зв'язані між собою. Відкритий ключ використовується для шифрування тексту повідомлення в “тарабарщину”, у той час як закритий ключ використовується для дешифрування й одержання вихідного тексту.

З тих пор як повідомлення було зашифровано за допомогою відкритого ключа, воно може бути розшифровано тільки відповідним йому закритим ключем. Жоден із ключів не може виконувати обидві функції. Відкритий ключ публікується у відкритому доступі без ризику піддати систему погрозам, але закритий ключ не повинен потрапити до кого-небудь, що не має правий на дешифрування даних. Отже, ми маємо ключі – відкритий і закритий. Одним з найбільш вражаючих достоїнств асиметричного шифрування є те, що дві сторони, раніше зовсім не знаючі один одного, можуть установити захищене з'єднання, споконвічно обмінюючись даними по відкритому, незахищеному з'єднанню.

Клієнт і сервер використовують свої власні закриті ключі (кожний – свій) і опублікований відкритий ключ для створення загального секретного ключа на сесію.

Це означає, що якщо хто-небудь перебуває між клієнтом і сервером і спостерігає за з'єднанням – він однаково не зможе довідатися ні закритий ключ клієнта, ні закритий ключ сервера, ні секретний ключ сесії.

Алгоритм Діффі-Хеллмана

Одним з найпоширеніших підходів є алгоритм обміну ключами Діффі-Хеллмана (DH). Цей алгоритм дозволяє клієнтові й серверу домовитися із приводу загального секретного ключа, без необхідності передачі секретного ключа по з'єднанню. Таким чином, зловмисники, що прослуховують канал, не

зможу визначити секретний ключ, навіть якщо вони будуть перехоплювати всі пакети даних без винятку.

Як тільки відбувся обмін ключами по DH-алгоритму, отриманий секретний ключ може використовуватися для шифрування подальшого з'єднання в рамках даної сесії, використовуючи набагато більше просте симетричне шифрування.

Використовуючи секретний ключ, отриманий раніше, а також домовившись із приводу режиму шифрування, клієнт і сервер можуть безпечно обмінюватися даними, шифруючи й дешифруючи повідомлення, отримані друг від друга з використанням секретного ключа. Зловмисник, що підключився каналу, буде бачити лише “сміття”, що гуляє по мережі взад-уперед.

Автентифікація

Алгоритм Діффі-Хеллмана дозволяє двом сторонам одержати закритий секретний ключ. Але звідки обидві сторони можуть упевнені, що розмовляють дійсно один з одним? Ми ще не говорили про автентифікації. Що якщо я подзвоню своєму приятелеві, ми здійснимо DH-Обмін ключами, але раптом виявиться, що мій дзвінок був перехоплений і насправді я спілкувався з кимсь іншим? Я по колишньому смозі безпечно спілкуватися із цією людиною – ніхто більше не зможе нас прослухати – але це буде зовсім не той, з ким я думаю, що спілкуюся. Це не занадто безпечно.

Для рішення проблеми автентифікації, нам потрібна Інфраструктура відкритих ключів, що дозволяє бути впевненим, що суб'єкти є тими за кого себе видають. Ця інфраструктура створена для створення, керування, поширення й відкликання цифрових сертифікатів. Сертифікати – це те, за що потрібно платити, щоб сайт працював по HTTPS. Але, насправді, що це за сертифікат, і як він надає нам безпека?

Сертифікати

У самому грубому наближенні, цифровий сертифікат – це файл, що використовує електронної-цифровий підпис (докладніше про це через мінуту) і єднальний відкритий (публічний) ключ комп'ютера з його приналежністю.

Цифровий підпис на сертифікаті означає, що хтось засвідчує той факт, що даний відкритий ключ належить певній особі або організації [5].

По суті, сертифікати зв'язують доменні імена з певним публічним ключем. Це запобігає можливість того, що зловмисник надасть свій публічний ключ, видаючи себе за сервер, до якого звертається клієнт.

У прикладі з телефоном, наведеному вище, хакер може спробувати пред'явити мені свій публічний ключ, видаючи себе за мого друга – але підпис на його сертифікаті не буде належати тому, кому я довіряю.

Щоб сертифікату довіряв будь-який WEB-браузер, він повинен бути підписаний акредитованим центром, що засвідчує (центром сертифікації, Certificate Authority, CA). CA – це компанії, що виконують ручну перевірку, того що особа, що намагається одержати сертифікат, задовольняє наступним двом умовам:

– є реально існуючим;

– має доступ до домену, сертифікат для якого воно намагається одержати.

Як тільки CA засвідчує в тому, що заявник – реальний і він реально контролює домен, CA підписує сертифікат для цього сайту, по суті, установлюючи штамп підтвердження на тім факті, що публічний ключ сайту дійсно належить йому і йому можна довіряти.

У ваш браузер уже споконвічно передзавантажен список акредитованих CA. Якщо сервер повертає сертифікат, не підписаний акредитованим CA, то з'явиться велике червоне попередження. У противному випадку, кожний міг би підписувати фіктивні сертифікати. Так що навіть якщо хакер взяв відкритий ключ свого сервера й згенерував цифровий сертифікат, що підтверджує що цей публічний ключ, асоційований із сайтом, приміром, з facebook.com, браузер не повірить у це, оскільки сертифікат не підписаний акредитованим CA.

Фактичний захист

Рівень захисту залежить від коректності введення браузерного й серверного програмного забезпечення й підтримки криптографічних алгоритмів. Серед користувачів кредитних карт в Інтернеті існує помилкова думка, що

HTTPS повністю захищає номер їхньої карти від злодіїв. Фактично шифроване підключення до WEB-серверу тільки захищає номер кредитної карти при передачі між комп'ютером користувача й сервером прямо. Це не гарантує що сервер безпосередньо захищений – він навіть може бути зламаним [6].

Нападу на WEB-сервери, які зберігають дані клієнта, здійснити простіше, ніж намагатися перехопити дані при передачі. Вважається, що комерційні сайти негайно пересилають операції вступників до шлюзу оплати й зберігають тільки операційний номер, але вони часто зберігають номери карт у базі даних. Звичайно сервер і база даних є метою для нападу номер один для зловмисників.

Перед тим, як відправити хоча б біт корисних даних, за протоколом HTTPS між браузером і сервером установлюється зашифроване з'єднання. Цей метод має на увазі в тому числі автентифікацію.

TLS

SSL і TLS являють собою розвиток однієї й тої ж технології. Абревіатура TLS з'явилася як заміна позначення SSL після того, як протокол остаточно став інтернет-стандартом. Така заміна викликана юридичними аспектами, тому що специфікація SSL споконвічно належала компанії Netscape. У багатьох випадках назви SSL і TLS продовжують використовувати як синоніми, хоча канонічним ім'ям зараз є TLS. При цьому TLS має окрему нумерацію версій і є більше сучасним протоколом.

TLS/SSL споконвічно розробили для захисту комерційних транзакцій, проведених через Інтернет. Тобто основною метою було одержання щодо безпечного каналу для здійснення покупок або керування банківським рахунком – хоча, ні перше, ні друге ще не користувалися якоюсь популярністю в рядових користувачів у часи становлення SSL. Зате в сучасному Інтернеті на TLS/SSL покладаються не тільки в комерційній діяльності, але й при рішенні набагато більше загального завдання збереження "приватності" і конфіденційності важливої інформації. Одним з найпоширеніших застосувань TLS є HTTPS. HTTPS стрімко витісняє незахищену версію (HTTP): частка

зашифрованого WEB-трафіку росте, швидше за все, у швидкому майбутньому (кілька років, 2020) практично весь WEB-трафік буде зашифрований. У новій версії HTTP/2 захист інформації засобами TLS повинна використовуватися за замовчуванням. Завдяки такому положенню справ, SSL/TLS – один із самих вивчених, досліджених протоколів сучасного Інтернету. Разом з тим, історія цього протоколу показує, наскільки мало буває практичної користі від досліджень: наприклад, в 2010 році корпорація Microsoft виправила дефект у реалізації SSL свого WEB-серверу IIS – CVE-2010-3332; а відповідна уразливість була продемонстрована в 2003 році. Тобто, сім років результати дослідження вважалися чистою теорією, що не заслуговує практичного виправлення. Втім, починаючи, приблизно, з 2014 року, інтерес до пошуку уразливостей в TLS сильно зріс, а виявлені уразливості сталі привертати увагу "самих широких шарів" інтернет-користувачів. Завдяки цьому, процес виправлення дефектів придбав деяку оперативність і навіть безперервність.

Уперше SSL впровадили в якості пропрієтарної технології, реалізованої в браузері Netscape Navigator, одному з перших WEB-браузерів. Версія 1 протоколу не була опублікована, а так і залишилася внутрішньою розробкою Netscape, що розвивалася в 1994-95 роках. SSLv2 – наступну, другу версію протоколу – опублікували, однак специфікація так і не вийшла зі стану чернетки (draft). Більше того, хоч в SSLv2 і скорегували окремі дефекти й уразливості v1, протокол виявився ненадійні, утримуючі серйозні архітектурні огріхи. SSLv2 давно (більше 15 років тому) і без застережень визнаний небезпечним, тому зараз не повинен використовуватися. Хоча на просторах WEB усе ще можна зустріти архаїчні сервери, які підтримують SSLv2, більше того, багато сучасних криптографічних бібліотек усе ще підтримують SSLv2 по історичних причинах. Незважаючи на ріст популярності TLS, витиснення дефектних технологій іде повільно. Однак якщо ваш сервер або клієнтське ПО підтримує SSLv2, можна сміло сказати, що у вас просто немає безпечного каналу, ви не підтримуєте TLS/SSL зовсім. Більше того, на практиці було продемонстровано, що наявність

сервера з SSLv2, що використовує ті ж ключі, що й сучасний TLS-сервер, ставить під погрозу сесії TLS (навіть якщо це різні фізичні сервери, а трафік до них доставляється незалежно).

SSLv3 – це розвиток SSLv2, але з досить істотними доробками. SSLv3 представлений в 1996 році. Цей протокол одержав повноцінну специфікацію RFC, щоправда, значно пізніше своєї появи, і в статусі історичного документа: RFC 6101. На час SSLv3 довелося становлення TLS як інтернет-технології. Саме на базі SSLv3 і з'явився протокол TLS.

Зараз існує три версії TLS, всі вони описані в RFC: TLS 1.0, TLS 1.1 і TLS 1.2. Розробляється версія TLS 1.3, очікується, що в ній з'являться значні поліпшення. TLS 1.0 багато в чому повторює SSLv3, за винятком ряду деталей, які, втім, є досить важливими. (У криптографічних протоколах деталі важливі як ні в яких інших протоколах Інтернету.) Ключовою відмінністю TLS від SSL є наявність підтримки цілого ряду розширень протоколу, що дозволяють реалізувати сучасні методи захисту інформації. TLS 1.1 і 1.2 досить близькі до 1.0, але у версії 1.2 з'явилися помітні зміни, наприклад, використовується інша основа псеводвипадкової функції й з'явилася підтримка шифрів у режимі автентифікованого шифрування. Розробляється (вересень 2018) версія 1.3, що архітектурно дуже істотно відрізняється й від 1.2, і, тим більше, від більше ранніх версій протоколу. Так, в TLS 1.3 повністю змінений алгоритм установлення з'єднання; вилучені зі специфікації або перероблені багато повідомлень (елементи протоколу). Однак TLS 1.3 поки перебуває в статусі Draft, специфікація активно міняється. Даний текст ставиться до сучасним затверджених відповідно до процесу IETF версіям TLS – 1.0, 1.1, 1.2. Там, де це потрібно, по тексту дані пояснення, що стосуються версії 1.3, однак використовувані назви полів і схеми повідомлень – ставляться до версій до 1.2 (включно). Для детального розуміння роботи протоколів TLS можливі нововведення 1.3 некритичні. Докладний опис TLS 1.3 у версії одного із чернеток (draft) дано в Додатку 1.

Рекомендованої для застосування версією є TLS 1.2, допускається використання TLS 1.1. TLS 1.0 повсюдно підтримується багато років – даний протокол сполучимо навіть із древнім браузером IE 6. TLS 1.1 підтримується практично всіма скільки-небудь розповсюдженими програмними сервісами WEB, а також усіляких інших інтернет-додатків. Підтримка TLS 1.2 в 2018 році також є повсюдною: всі сучасні браузери підтримують цю версію, а проблеми можуть виникнути тільки з аномально старими складаннями й, іноді, на стороні сервера, при використанні дійсно древніх додатків.

TLS 1.0 можна назвати версією 3.1 SSL. Властиво, саме так і зроблено в специфікаціях при описі способу найменування версій: SSL 3.1 – це TLS 1.0; 3.2 – TLS 1.1; 3.3 – TLS 1.2.

З 2015 року SSLv3, слідом за публікацією чергових уразливостей, перейшов у статус нерекомендованих прямо. Цей статус цілком офіційний, якої тільки офіційним він може бути в рамках технологічних традицій Інтернету: у червні 2015 року випущений документ RFC 7568, що вимагає виключити SSLv3 з розряду підтримуваних клієнтами й серверами протоколів. Тепер залишилися тільки версії TLS.

Призначення

TLS ставить своєю метою створення між двома вузлами мережі захищеного від прослуховування й підміни інформації каналу зв'язку, придатного для передачі довільних даних в обох напрямках, а також перевірку того, що обмін даними відбувається між саме тими вузлами, для яких канал і планувався. Ці завдання називаються, відповідно, забезпеченням конфіденційності, цілісності й дійсності з'єднання (автентифікації). З фундаментальних завдань захисту інформації, TLS не охоплює тільки одну: забезпечення доступності інформації – воно перебуває далеко за рамками даного протоколу.

Передбачається, що TLS працює поверх існуючі між вузлами "потокового" з'єднання (з яким звичайно зв'язаний "сокет" – звідки назва). Зараз, у переважній більшості випадків, TLS буде працювати поверх TCP. Саме TCP

відповідає за встановлення з'єднання, розбивку даних на пакети, гарантовану доставку цих пакетів (при наявності з'єднання, природно) і за різні інші телекомунікаційні моменти. TLS уважає, що між вузлами встановлене надійне з'єднання, тому, наприклад, протокол не охоплює повторне відправлення загублених пакетів даних. (Для роботи в умовах негарантованої доставки й нестійкого зв'язку існує особливий, родинний протокол – DTLS, що ми тут не розглядаємо.) TLS використовує встановлений канал зв'язку для передачі повідомлень. Ці повідомлення кардинальним образом відрізняються від пакетів даних (IP-пакетів в TCP/IP) і являють собою структуру більше високого рівня. TCP дозволяє надійно обмінюватися даними, але вміст пакетів (за винятком спеціальних випадків) може бути легко прочитане кимсь, хто має доступ до каналу зв'язку. Гірше того, цей хтось може замінити пакети або змінити передані в них дані. Саме для захисту від цих погроз і використовується TLS.

Інакше кажучи, модель погроз TLS припускає, що атакуючий може як завгодно втручатися в канал зв'язку, у тому числі активно підмінювати пакети й взагалі – переривати зв'язок. Ключові завдання TLS: 1) забезпечити конфіденційність, тобто реалізувати захист від витоків переданої інформації; 2) забезпечити виявлення підміни, тобто реалізувати збереження цілісності переданої інформації; 3) забезпечити автентифікацію вузлів, тобто дати механізм перевірки дійсності джерела повідомлень. TLS якось справляється із цими завданнями. Але не слід думати, що TLS вирішує ці завдання повністю. Така думка є великою помилкою, на жаль, досить розповсюдженої. Доказом того, що TLS не вирішує дані завдання повністю, є уразливості, виявлені в самому протоколі (а не тільки в його реалізаціях). Протокол і його реалізації намагаються вирішити описані завдання на максимально доступному рівні надійності. Однак TLS у цілому не має доведену стійкість, як не володіють їй і багато найважливіших складових частин протоколу. Звичайно в якості екстремального прикладу даного спостереження приводять таку рекомендацію: не слід довіряти TLS і зв'язаним технологіям своє життя або життя інших людей.

TLS використовує концепцію клієнт-сервер, що, у логіку даного протоколу, багато в чому збігається з логікою клієнт-сервер TCP: наприклад, і там і там з'єднання ініціює клієнт.

Логіка TLS

TLS працює із записами (records). Записи перебувають у фундаменті протоколу. Це нижній транспортний рівень TLS. Якщо розглядати сеанс TLS на рівні умовного сокета (TCP), то кожний переданий запис являє собою блок, що складається з короткого заголовка й, властиво, самих даних. Повідомлення TLS, що ставляться до верхніх рівнів, можуть бути розбиті на кілька записів. Це досить важливий момент: у деяких випадках повідомлення вимагають складання з декількох записів, що істотно впливає на логіку обробки станів протоколу.

Зверніть увагу, що в TLS використовується мережний порядок байтів (старший байт іде першим, ліворуч праворуч – "тупокінцеве" подання). Розберемо заголовок запису. Заголовок має довжину 5 байтів, і наступний формат:

```
00   Тип (1 байт)
01   Версія протоколу (2 байти)
03   Довжина даних (2 байти)
```

Тип – це тип запису. Визначено чотири типи: 20 (0x14) – повідомлення Change Cipher Spec (CCS); 21 (0x15) – повідомлення Alert (це не обов'язково попередження, є цілком "фатальні alert-и"); 22 (0x16) – повідомлення Handshake (установлення з'єднання); 23 (0x17) – Application Data (запис містить дані додатки – тобто, корисне навантаження). При передачі інформації, основні перетворення із шифрами й кодами автентифікації саме відбуваються в блоці дані записи з типом 23 (0x17) Application Data. Інші записи звичайно передаються у відкритому виді, але можуть бути й зашифровані, залежно від типу запису й поточного стану TLS-з'єднання;

Версія протоколу – це версія, записана у двох байтах, як зазначено вище: 03 00 – це SSLv3, 03 01 – TLS 1.0; 03 02 – TLS 1.1; 03 03 – TLS 1.2;

Довжина даних – кількість байтів, що містять дані (самі байти випливають після заголовка). Тому що використовується мережний порядок байтів, для визначення довжини потрібно значення першого байта (h) помножити на 28 і додати значення другого (молодшого, l) байта, от так: h * 256 + l. Порожній блок даних (довжина нуль) – допускається протоколом, але іноді викликає збої в клієнтах, написаних з помилками.

Приклад заголовка запису (значення байтів у шістнадцяткового запису):

```
16 03 02 03 FE
```

Розкодуємо: це повідомлення Handshake (0x16); версія 3.2 (0x03,0x02)- тобто TLS 1.1; довжина даних – 0x03FE байтів (1022 байта).

Заголовок завжди передається у відкритому виді.

За заголовком повинне випливати певне число байтів (>=0), які являють собою вміст даного запису. Максимальне значення – 18432 байта. 18432 = 16384 + 2048. Тобто, 16 кілобайт + два рази по кілобайті. Таке значення може здатися дивним. Дійсно, "базова" версія запису, описана в специфікації, припускає блок даних в 214 байтів, тобто 16 кілобайт. Але такий розмір ставиться тільки до записів, що містять відкритий текст (TLSPlaintext). Наприклад, до повідомлень, що управляють установленням з'єднання (Handshake). Тут, дійсно, максимальна довжина – 16 К.

Для захищених записів ситуація міняється. По-перше, уводиться стиск даних. В TLS всі дані захищеного запису повинні стискуватися, так запропоновано специфікацією. Але на практиці це означає не зовсім те, про що можна подумати (для TLS це нерідка ситуація): на практиці, стиск зараз не використовується. Справа в тому, що TLS серед алгоритмів стиску містить алгоритм null, що означає, що ніякого стиску в реальності не виробляється. Цей алгоритм служить варіантом за замовчуванням (якийсь час можна було зустріти алгоритм DEFLATE, але зараз він ставиться до нерекомендованого через виявлений уразливостей). Проте, реально стислі дані можуть, у ряді випадків, виявитися більш довгими, через особливості алгоритму стиску (цей факт може здатися контрінтуітивним, але для будь-якого алгоритму стиску можна підібрати

вхідні дані таким чином, що результат кодування "зі стиском" виявиться більш довгими). TLS відводить на таке збільшення довжини 1024 байта (добре реалізовані алгоритми використовують значно менше).

По-друге, для перевірки цілісності зашифрованих даних до них приписується код автентифікації повідомлення (MAC) і додаткова інформація (начебто векторів ініціалізації шифрів) – на це приділяється ще 1024 байта. Разом – 2048, хоча даний момент і вносить деяке сум'яття в організацію протоколу. Природно, універсальні й, тому, фіксовані 16K як максимальна межа дивилась би краще. У реальності, рідкий запис TLS дотягає й до 16K – звичайно довжина помітно менше.

Код автентифікації – найважливіший елемент захисту інформації в TLS. Звичайно використовується HMAC – версія з тої або іншою геш-функцією. Типовий вибір: SHA-1 або SHA-256. Код автентифікації приписується до блоку даних. Один з архітектурних дефектів всіх використовуваних зараз версій TLS полягає в тому, що специфікація пропонує спершу обчислювати MAC, а потім шифрувати повідомлення. Тобто, обчислений код автентифікації приєднується до відкритого тексту повідомлення, а потім усе разом шифрується. При цьому приймаюча сторона повинна спершу розшифрувати отримані дані, а потім – перевірити MAC. Добре відомо, що такий метод легко приводить до виникнення "криптографічних оракулів", на використанні яких засноване кілька ефективних атак проти TLS (ці атаки докладно розглянуті в спеціальному розділі). Сучасний підхід: код автентифікації додається після шифрування відкритого тексту, тобто MAC обчислюється для вже зашифрованого повідомлення й прикріплюється до нього ("спершу шифр, потім – MAC"). Для впровадження сучасного підходу в специфікацію додане спеціальне розширення, що дозволяє клієнтові й серверу домовитися про те, у якій послідовності вони генерують коди автентифікації – RFC 7366. Більше того, сучасні шифри в TLS використовуються в режимі автентифікованого шифрування (а саме – AEAD). Цей режим не має потреби в окремому MAC, тому що цілісність даних гарантує сам алгоритм шифрування

(докладний опис даний нижче, у розділі, присвяченому шифрам). Приклад: AES у режимі GCM, цей варіант є кращим, підтримується всім сучасним ПО, і в 2016 році одержав дуже велике поширення.

Використовувані криптосистеми в TLS поєднуються в типові шифронабори (Cipher Suites). Щоб почати обмін інформацією із захищеного каналу, клієнт і сервер повинні погодити використовуваний шифронабор. Очевидно, що параметри шифрів і обміну супутньою інформацією повинні бути сумісні між собою. Узгодження проводиться на етапі встановлення з'єднання (Handshake). У шифронабор входять:

– криптосистема, використовувана для автентифікації сервера й сеансового секрету;

– шифр, що послужить для захисту переданих даних;

– геш-функція, що є основою для HMAC.

– Існують стандартні позначення для шифронаборів. Наприклад, вибір типового шифронабора TLS_RSA_WITH_AES_128_CBC_SHA означає, що буде використана криптосистема RSA для передачі сеансового ключа, AES з 128-бітним ключем у режимі CBC як симетричний шифр (тобто, що захищаються дані додатки будуть зашифровані симетричним шифром AES), SHA-1 у якості геш-функції HMAC. Шифронабори строго фіксовані, состав закріплений в RFC, кожному приписаний свій індекс, а реєстр веде IANA.

Шифронабори мають найважливіше значення для безпеки TLS. Вибір нестійкої комбінації означає, що достатнього захисту конкретна реалізація TLS не забезпечує. Стосовно до вебу: браузери використовують убудований комплект шифронаборів, звичайно це 10-15 варіантів; на сервері підтримувані шифронабори настроюються адміністратором. Реєстр IANA у даний момент містить понад 300 (три сотні, так) шифронаборів (у число яких входять і так звані псевдошифри, які використовуються як сигнали). Серед шифронаборів зустрічається екзотика начебто TLS_KRB5_WITH_RC4_128_SHA і безнадійно застарілі варіанти, наприклад – TLS_RSA_EXPORT_WITH_RC4_40_MD5.

Сучасні сервери й клієнти не повинні використовувати подібного. Наявність того або іншого шифронабора в реєстрі IANA означає, що цьому шифронабору привласнено індекс (номер) і позначення, не більше того: потрібно розуміти, що даний реєстр не має ніякого відношення до тому, які шифри й криптосистеми рекомендовані або не рекомендовані для використання. В 2018 році добротним варіантом уважається TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA256: тобто, генерація загального секрету проводиться за протоколом Діффі-Хеллмана на еліптичних кривих, для автентифікації даних на етапі встановлення з'єднання використовується криптосистема електронного підпису ECDSA (теж працююча на еліптичних кривих), корисне навантаження шифрується AES з 256-бітним ключем у режимі GCM, а в якості геш-функції служить SHA-256 (використовується при генерації сеансових ключів).

Установлення з'єднання (Handshake)

TLS Handshake іноді в сучасних статтях називають "рукостисканням", що є прямим, і не дуже вдалим, перекладом англійського терміна; цей варіант ми не станемо використовувати. Клієнт і сервер повинні домовитися про використовувані шифри й методи автентифікації, погодити ключі й інші параметри сеансу зв'язку. Набір погоджених параметрів називається криптографічним контекстом. Узгодження відбувається при встановленні з'єднання, шляхом обміну спеціальними повідомленнями – handshake-повідомленнями. Такий обмін в TLS здійснюється поверх обміну записами. Кожне handshake-повідомлення містить спеціальний заголовок, що складається із чотирьох байтів. Перший байт позначає код типу повідомлення, три наступні байти – довжину повідомлення.

Повідомлення Handshake

Handshake-повідомлення відправляються у вигляді записів, які ми обговорили вище. Це означає, що заголовку повідомлення може передувати заголовок запису, де тип повідомлення визначений кодом 22 (0x16) – див. вище. Трохи handshake-повідомлень можуть бути передані в одному записі – це

розповсюджений випадок, наприклад, так нерідко надходить Miscrosoft IIS. А саме цікаве, що одне повідомлення може бути розбите на кілька записів, переданих послідовно – специфікація допускає й це. На щастя, останній варіант скоріше теоретичний, на практиці зустрічається яке щезає рідко. (А екстремальний метод "заплутування" складається в передачі перед розрізаним на кілька записів повідомленням порожніх записів, тобто TLS-записів, що складаються з одного заголовка, де як довжина зазначена нуль; такий варіант також допускається специфікацією.) Всі ці випадки повинні коректно підтримуватися добротною реалізацією TLS. Таким чином, що перебуває нижче протоколу встановлення з'єднання рівень TLS-записів потрібно розглядати як якийсь потоковий псевдосокет, куди повідомлення записуються як є, і де вони можуть бути розрізані на блоки для передачі. Єдине, що забороняє специфікація TLS – це порушення порядку при доставці записів (на відміну від IPv4).

Специфікація припускає, що будь-який клієнт і сервер TLS за замовчуванням уже погодили шифронабор null, без MAC і з "нульовим" стиском. Тобто клієнт і сервер можуть обмінюватися повідомленнями у вигляді відкритого тексту без автентифікації. Установлення з'єднання завжди починає клієнт. У типовій конфігурації, TLS тут виявляється схожий з TCP, тому що, наприклад, при роботі із сервером по HTTPS – TCP-з'єднання також ініціює клієнт. Завдяки історичному збігу принципів побудови технологій, клієнт HTTPS, TLS, TCP – той самий вузол. Можна запропонувати конфігурації протоколів, коли це не так, але в найпоширенішій ситуації WEB – подібні конфігурації навряд чи можливі.

```
Client                                               Server
ClientHello ------->
                                                   ServerHello
                                                  Certificate*
                                            ServerKeyExchange*
                                           CertificateRequest*
                                 < ----    ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
 [ChangeCipherSpec]
```

```
  Finished ------->
                                          [ChangeCipherSpec]
                               < ----            Finished
  Application Data             < ------>     Application Data
```

Схема обміну Handshake-повідомленнями TLS 1.2
(ChangeCipherSpec – не є повідомленням Handshake).
Джерело: RFC 5246

Першим повідомленням у протоколі встановлення TLS-з'єднання завжди є повідомлення ClientHello (тип 01). Повідомлення містить наступні дані:

– версію протоколу – максимальну версію, що готовий підтримувати клієнт;

– 32 байта випадкових значень – ClientRandom. Споконвічно, у специфікації рекомендувалося використовувати перші 4 байти для передачі UNIX-таймстемпа, а що залишилися 28 – заповнювати результатом роботи криптографічного генератора псевдовипадкових чисел. Однак зараз багато браузерів і WEB-сервери генерують всі 32-байта випадковим образом. Забігаючи ледве вперед: передбачалося, що наявність таймстемпа в handshake-повідомленнях може допомогти при виявленні проблем з підміною часу на тім або іншому вузлі. Однак даний метод зараз ніяк не використовується, тому байти ClientRandom часто просто випадкові;

– ідентифікатор TLS-сесії – SessionID: TLS дозволяє відновляти раніше встановлені сесії, використовуючи скорочений варіант протоколу встановлення з'єднання. Ідентифікатор сесії саме містить номер такої сесії, параметри якої (можливо) збережені на сервері (докладніше ми розглянемо цей варіант нижче);

– список шифронаборів, які підтримує клієнт – Cipher Suites. Порядок шифронаборів у списку відбиває їхній ступінь переваги клієнтом (кращі передаються першими), цей порядок – усього лише рекомендація, і сервер далеко не завжди їй треба;

– список підтримуваних методів стиску – Compression Methods, порядок, знову ж, відповідає ступеню переваги, але звичайно в цьому полі лише одне значення – null, тому що стиск не рекомендується використовувати;

– дані декількох розширень протоколу.

Кожне з полів у повідомленні ClientHello також має свій формат, а полю передують дані про його довжину й, у випадку розширень, додатковий заголовок. Заголовок звичайно містить тип розширення й довжину його даних. Така структура дозволяє реалізаціям коректно розбирати повідомлення. Наприклад, список шифронаборів може бути представлений у такому виді (шістнадцятковий запис):

```
 [...] 00 06 c0 2b c0 2f з0 29
00 06 – шість байтів займають ідентифікатори шифронаборів;
c0 2b – шифронабор 0xc02b – TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256;
c0 2f – шифронабор 0xc02f – TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256;
c0 29 – шифронабор 0xc029 – TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256.
```

Розширення ClientHello дозволяють використовувати різні нововведення, які з'явилися пізніше, ніж структура даного повідомлення. Так, наприклад, у розширеннях можуть бути перераховані еліптичні криві, які готова використовувати клієнт. Найпоширенішим розширенням є SNI (Server Name Indication), що дозволяє браузеру вказати ім'я WEB-сервера, з яким він намагається встановити з'єднання. SNI необхідно для того, щоб на одному IP-адресі могли коректно відгукуватися по HTTPS WEB-сервери, розміщені під різними доменами.

Розберемо ClientHello у деталях. Як приклад візьмемо спеціальний зразок ClientHello, згенерований утилітою опитування TLS-серверів у дослідницьких цілях. Як сервер використовується kbpz.kntu.kr.ua.

```
Зсув Байти                          Коментар
(десяткове)    (шістнадцяткове подання)
0000 16        ; Перший байт заголовка TLS-запису – тип 22, повідомлення Handshake.
0001 03 02 00 65    ; Версія TLS 1.1 (03 02), довжина пакета даних – 0065 (101 байт).
0005 01        ; Перший байт повідомлення TLS: 01 – тип ClientHello.
0006 00 00 61  ; Довжина блоку даних – 000061 (97 байтів).
0009 03 02          ; Версія TLS 1.1 (03 02) клієнта.
0011 53 53 4c 20 53  ; 32 байти випадкових значень, ClientRandom
 [...]
0043 00        ; Довжина поля SessionID – 0 байтів, тому що SessionID у даному
повідомленні
                 не використовується (у випадку браузерного ClientHello буде зазначена
довжина
```

```
                запису SessionID, за якої піде значення SessionID).
0044      00 16         ; Довжина поля зі списком шифронаборів (Cipher Suites) – 22
байта: у даному
                ServerHello зазначено 11 шифронаборів, по двох байта на кожний з них
0046      c0 2b         ; Шифронабори.
 [...]
0068      01 00         ; Довжина поля CompressionMethods (1 байт) і значення цього
поля (00 – null,
                без стиску: клієнт не підтримує стиск).
0070 00 22          ; Довжина поля Extensions (0022 – 34 байта), поле, у нашім
випадку, містить
                розширення: SNI, Elliptic Curves, Elliptic Curves Point Formats.
0072 00 00          ; Розширення SNI (Server Name Indication, тип 0000). Дане
розширення
                дозволяє вказати ім'я сервера, з яким буде відбуватися з'єднання
                за протоколом HTTPS (у цьому випадку – HTTPS, але SNI може
використовуватися й для
                інших протоколів).
0073 00 0c          ; Довжина даних у розширенні SNI.
0075 00 0a 00 00 07; Заголовок розширення SNI – довжина списку імен (000a – 10
байтів), тип
                ім'я (hostname – 00), довжина рядка з ім'ям (7 байтів).
0080 73 62 72 66 2e 72 75    ; Значення ім'я SNI: kbpz.kntu.kr.ua
0086 00 0a          ; Тип розширення – Elliptic Curves (000a), дане розширення
дозволяє
                клієнтові вказати список еліптичних кривих, з якими цей клієнт
                уміє працювати.
0088 00 08          ; Довжина дані розширення (8 байтів).
0090 00 06          ; Довжина списку кривих (6 байтів – три криві, по двох байта на
                ідентифікатор).
0092 00 17 00 18 00 19  ; Ідентифікатори кривих.
0098 00 0b 00 02 01 00       ; Розширення Elliptic Curves Point Format, тип (000b),
довжина
                     (0002), довжина списку (01), формат запису точок – повний
                     (тобто, повністю передаються дві координати x,y в аффинном
                     поданні; є формати зі стиском, де передається тільки
                     одна координата, x, а для другої вказується знак).
```

Відповідно до специфікації, після відправлення ClientHello, клієнт очікує відповіді сервера. У відповідь може прийти або повідомлення про помилку, у вигляді Alert, або повідомлення ServerHello.

Alert – короткі повідомлення, що містять інформацію про рівень помилки й про її тип. Повідомлення Alert, наприклад, може сигналізувати про збій установлення з'єднання. Розглянемо такий варіант докладніше:

```
Зсув Байти                    Коментар
(десяткове)    (шістнадцяткове подання)
0000 15   ; Тип повідомлення – 0x15 – Alert.
0001 03 01     ; Версія протоколу – 0x0301 – TLS 1.0.
0003 00 02     ; Довжина повідомлення (два байти).
0005 02   ; Рівень помилки – 0x02 – Fatal (фатальна, з'єднання неможливо).
0006 28   ; Состав помилки – 0x28 – Handshake Failure (збій установлення з'єднання).
```

Якщо ж сервер зміг успішно обробити ClientHello, то він відповідає повідомленням ServerHello. Це повідомлення так само має заголовок із чотирьох байтів: тип повідомлення (один байт) і довжина (три байти). ServerHello містить наступні поля:

1) версію протоколу, що будуть використовувати клієнт і сервер;

2) 32 байта випадкових значень – ServerRandom. Із цим рядком ситуація така ж, як і з Client Random: перші чотири байти можуть бути таймстемпом, а можуть і не бути. Цікавою властивістю даної мітки часу є те, що вона може послужити деяким довідковим матеріалом при наступному аналізі записаного трафіку: якщо сервер відповідає достовірним значенням часу, те, з огляду на, що повний набір повідомлень Handshake містить повідомлення, криптографічно засвідчуюче цілісність даних одержуємо підписану сервером квитанцію про час з'єднання по його годинниках, з точністю до секунди – іноді ці дані допомагають правильно скласти запит на адресу адміністрації сервера про добування записів з логів;

3) ідентифікатор сесії – SessionID, привласнений нової сесії сервером;

4) обраний сервером шифронабор – Cipher Suite, цей шифронабор буде використовуватися надалі й клієнтом, і сервером. Сервер вибирає шифронабор із запропонованих клієнтом в ClientHello, але не обов'язково треба пріоритету клієнта: поширена саме зворотна ситуація, коли сервер винятково сам визначає кращий шифронабор;

5) обраний сервером метод стиску – швидше за все, це null;

6) деякий набір розширень.

У сучасних реалізаціях TLS розділ "Розширення" (Extensions) має дуже велике значення, тому що в ньому передаються параметри, що визначають схему роботи й клієнта, і сервера. Призначення ServerHello – погодити параметри з'єднання. TLS використовує добре відому схему " запит-відповідь-підтвердження", а повідомлення ClientHello і ServerHello запускають схему. Ці повідомлення не містять ніякої секретної інформації, відповідно – передаються у відкритому виді (в TLS 1.3 ситуація помінялася – практично відразу ж повідомлення шифруються). Дані повідомлення, особливо – ClientHello, часто використовуються в системах DPI для виявлення факту встановлення (або спроби встановлення) TLS-з'єднання.

У рамках кожної сесії TLS, клієнт і сервер погоджують наступні параметри (RFC 5246):

1) ролі вузлів: який вузол є клієнтом, а який – сервером. Розподіл цих ролей відбувається "природно", при встановленні з'єднання – клієнт завжди починає сесію;

2) алгоритм PRF (псевдовипадкова функція, PseudoRandom Function) – використовується для генерації сеансового ключа, на основі даних, переданих в ClientHello і ServerHello, тому клієнт і сервер повинні використовувати погоджений алгоритм;

3) алгоритм шифрування – для успішного шифрування потоку даних усередині сесії обидва вузли повинні використовувати той самий алгоритм, у погодженому режимі. Вузли погодять тип шифру (потоковий, блоковий), сам шифр, режим використання шифру, розмір ключа, розмір блоку (для блокових шифрів), ініціалізуючі вектори, а також додаткові параметри (якщо вони потрібні, наприклад, у випадку з AEAD-режимами);

4) алгоритм обчислення коду автентифікації повідомлення (MAC) – аналогічно шифруванню, у рамках сесії необхідно використовувати погоджений

алгоритм автентифікації. Для AEAD-режиму шифрування MAC не використовується;

5) алгоритм стиску – також повинен бути загальним, однак у сучасній реалізації стиск, фактично, не використовується;

6) основний секрет (MasterSecret) – масив з 48 секретних байтів, відомий серверу й клієнтові. Основне призначення даного масиву – генерація сеансового ключа, у тих шифронаборах, де застосовне. Це 384-бітне значення MasterSecret дозволяє обчислити секретні ключі сесії, тобто, якщо це значення стало відомо третій стороні, то вона зможе розшифрувати трафік сесії;

7) випадкові дані клієнта (ClientRandom) – 32 байта випадкових значень, передаються в ClientHello;

8) випадкові дані сервера (ServerRandom) – 32 байта випадкових значень, передаються в ServerHello.

Ці параметри дозволяють побудувати контекст для роботи криптосистеми, що буде обробляти захищені TLS-запису на стороні сервера й клієнта. Крім ClientHello і ServerHello, установлення з'єднання має на увазі обмін декількома іншими повідомленнями.

За відправленням ServerHello, з боку сервера треба набір інших повідомлень, состав і зміст яких залежать від обраного сервером режиму роботи.

Certificate. Ключовим аспектом для сучасних реалізацій TLS є використання TLS-сертифікатів (раніше вони називалися SSL-сертифікатами, зараз рекомендується варіант із TLS), ми розглянемо їх докладніше нижче. Сертифікати передаються сервером у повідомленні Certificate. Це повідомлення присутнє практично завжди. Серверний сертифікат містить відкритий ключ сервера. У теорії, специфікація дозволяє встановити з'єднання без відправлення серверних сертифікатів. Це так званий "анонімний" режим, однак він практично не використовується, як і режими TLS без шифрування. Так, приблизно 100% WEB-серверів, що підтримують TLS, передають TLS-сертифікати. Для типової конфігурації, повідомлення Certificate буде включати трохи TLS-сертифікатів,

серед яких один серверний сертифікат і так звані "проміжні сертифікати", що дозволяють клієнтові вибудувати ланцюжок валідації. Серверний сертифікат – це сертифікат, по ім'ю відповідному серверу й утримуючий серверний відкритий ключ електронного підпису (RSA або ECDSA). Специфікація пропонує строгий порядок проходження сертифікатів у повідомленні: першим повинен іти серверний сертифікат, а наступні – у порядку посвідчення попереднього. Однак на практиці сервери нерідко повертають сертифікати в довільному порядку. На клієнтській стороні браузери теж досить вільно ставляться до порядку сертифікатів, намагаючись вибудувати з них коректний ланцюжок шляхом різних перестановок. Таке положення справ прямо порушує специфікацію TLS версій 1.0, 1.1, 1.2, але така реальність: незважаючи на все строгості криптографії – реалізації допускають вільності (останні нерідко ведуть до уразливостей). Хоча, досить складно знайти вагомі причини для строго порядку проходження сертифікатів в TLS-повідомленні. У новій версії протоколу (TLS 1.3) дана вимога пропонується зм'якшити, зробивши припустимим довільний порядок проміжних сертифікатів (серверний однаково очікується першим).

ServerKeyExchange. Це повідомлення, що містить серверну частину даних, необхідних для генерації загального сеансового ключа. Повідомлення може буде відсутнім. Залежно від обраного шифронабора, даних, переданих в SSL-сертифікаті, може бути недостатньо для виробітку загального ключа. Відсутні дані саме й передаються в ServerKeyExchange. Звичайно це параметри протоколу Діффі-Хеллмана (DH), однак історичний варіант передбачає й передачу тимчасового ключа RSA. Якщо використовується класичний варіант DH, то в повідомленні ServerKeyExchange передається значення модуля й серверний відкритий ключ. У варіанті на еліптичних кривих (ECDH) – ідентифікатор самій кривій і, аналогічно DH, відкритий ключ сервера. Параметри підписуються сервером (за винятком екзотичних варіантів обміну, начебто анонімного DH), клієнт може перевірити підпис, використовуючи відкритий ключ сервера з SSL-сертифіката. Залежно від використовуваної криптосистеми,

підпис може бути DSA (зараз практично не зустрічається), RSA або ECDSA. Підпис на параметрах DH дуже важлива, тому що дозволяє захистити з'єднання від атаки типу "Людина посередині". У різних версіях TLS підпис обчислюється для різних наборів даних. В TLS 1.0, 1.1 (при використанні RSA, що є основним варіантом) – від об'єднання значень двох геш-функцій – MD5 і SHA-1, узятих від параметрів обміну DH. В TLS 1.3 – від значення геш-функції, заданої у використовуваному шифронаборе, обчисленого для об'єднання ServerRandom, ClientRandom і параметрів обміну DH (ECDH). У сучасній ситуації повідомлення ServerKeyExchange можна побачити в більшості TLS-сесій.

CertificateRequest. В TLS можлива обопільна (двостороння) автентифікація вузлів, що використовує TLS-сертифікати (SSL-сертифікати). Сертифікати можуть бути випущені для самих різних імен і, відповідно, придатні для автентифікації клієнта. Звичайно, клієнтські сертифікати використовуються при доступі до банківських, платіжних систем, до корпоративних WEB-шлюзам різного призначення, а також до державних інформаційних систем через WEB-інтерфейс. Сервер може запросити клієнтський сертифікат за допомогою повідомлення CertificateRequest. Повідомлення містить список підтримуваних сервером типів сертифікатів і типів криптосистем – тут вказуються криптосистеми, що ставляться до процедури валідації клієнтського сертифіката: алгоритми підпису, геш-функції. Також у складі CertificateRequest можуть бути передані імена центрів, що засвідчують, ключі яких сервер буде використовувати для перевірки клієнтського сертифіката.

ServerHelloDone – позначає закінчення пакета повідомлень, очолюваного ServerHello. Це повідомлення має нульову довжину (однак заголовок ніхто не скасовував, тому в TLS-записі ServerHelloDone відповідає 4 байти) і служить простим прапором, що позначає, що сервер передав свою частину початкових даних і тепер очікує відповіді від клієнта.

Отже, сервер відповідає на ClientHello послідовністю повідомлень TLS Handshake, максимум – п'ятьома повідомленнями. Типовий для сучасного стану

TLS випадок: передача сервером чотирьох повідомлень – ServerHello, Certificate, ServerKeyExchange (з параметрами алгоритму Діффі-Хеллмана), ServerHelloDone. Після передачі ServerHelloDone сервер очікує відповіді клієнта. Клієнт повинен відповісти своїм набором повідомлень:

Certificate – це повідомлення містить клієнтський сертифікат, якщо він був запитаний сервером. Якщо в клієнта сертифіката ні, а сервер його запитує, то клієнт або пропускає дане повідомлення (SSLv3), або відповідає порожнім повідомленням з типом Certificate (TLS). Клієнтський сертифікат потрібно для двосторонньої автентифікації. Всі сучасні браузери, що працюють на десктопі, підтримують клієнтські сертифікати. Клієнтський сертифікат, у ряді випадків, може замінити пари "логін/пароль" для автентифікації на тім або іншому онлайн-ресурсі (для автентифікації потрібен секретний ключ, що відповідає сертифікату).

ClientKeyExchange – клієнтська частина обміну даними, що дозволяють вузлам одержати загальний сеансовий ключ. Зміст цього повідомлення залежить від того, який шифронабор обраний. Є два основних типи – RSA і кілька варіантів протоколу Діффі-Хеллмана. У випадку використання RSA, клієнт генерує 48-байтовий випадковий секрет (при цьому перші два байти містять версію використовуваного протоколу), шифрує його відкритим ключем сервера (цей ключ передається в складі серверного SSL-сертифіката або в повідомленні ServerKeyExchange) і передає зашифровані дані на сервер. Сервер може розшифрувати значення, використовуючи відповідний секретний ключ. Дана схема є історичної, тому що володіє цілим рядом недоліків. Наприклад, якщо секретний серверний ключ стане відомий третій стороні, то вона зможе розшифрувати раніше записаний TLS-трафік. Проте, обмін сеансовим ключем за допомогою RSA продовжує використовуватися в деяких реалізаціях TLS. Сучасний метод – використання протоколу Діффі-Хеллмана. У цьому випадку, ClientKeyExchange містить відкритий ключ DH. Цей ключ генерується клієнтом або відповідно до параметрів, переданими сервером в ServerKeyExchange, або відповідно до параметрів, зазначеними в серверному SSL-сертифікаті, якщо

останній підтримує DH. Випадок з передачею параметрів класичного DH у складі сертифіката – зараз є екзотичним, таких сертифікатів в "дикій природі" не зустрічається. Серверні параметри DH (або ECDH) підписуються серверним ключем, клієнт перевіряє підпис, використовуючи відкритий ключ сервера.

CertificateVerify – якщо клієнт передав у відповідь на запит сервера свій сертифікат, то серверу потрібно деякий механізм, що дозволяє перевірити, що клієнт дійсно має секретний ключ, пов'язаним із сертифікатом. Для цього клієнт підписує масив переданих і прийнятих раніше повідомлень Handshake. Такий підпис, якщо її значення вдасться успішно перевірити відкритим ключем із сертифіката, засвідчить факт наявності секретного ключа в клієнта. Повідомлення передається тільки якщо був переданий клієнтський сертифікат у повідомленні Certificate.

У сучасній конфігурації клієнтські сертифікати використовуються рідко. Тому в типовому випадку клієнт, на даному етапі, передає одне повідомлення – ClientKeyExchange. Це повідомлення є, відповідно до специфікації, обов'язковим.

Слідом за повідомленням ClientKeyExchange, якщо воно було єдиним, або за CertificateVerify, клієнт повинен передати повідомлення ChangeCipherSpec. Важливий момент: це повідомлення не є повідомленням Handshake. Так, у специфікації TLS зустрічаються такі, не зовсім прозорі, моменти, коли начебто б логічний плин протоколу переривається спеціальними "вставками". При розробці TLS 1.3 від цієї сумнівної практики відмовилися. Зокрема, ChangeCipherSpec – запропоновано видалити.

ChangeCipherSpec – це спеціальне повідомлення-сигнал, що позначає, що з даного моменту клієнт переходить на обраний шифр, а наступні TLS-записи будуть зашифровані. ChangeCipherSpec (CCS) має власний тип і, відповідно, передається в окремої TLS-записі. Таким чином, CCS має досить важливе значення в TLS (раніше версії 1.3), відокремлюючи відкриту частину сеансу зв'язку від закритої.

З боку клієнта встановлення з'єднання завершується відправленням повідомлення Finished.

Finished. Це повідомлення передається в зашифрованої TLS-записі, тому що треба за сигналом ChangeCipherSpec, що позначає момент переходу до захищеного обміну інформацією. Finished є першим захищеним повідомленням, у рамках нового сеансу TLS. Згадаєте, що до моменту його відправлення, якщо все пройшло успішно, сервер і клієнт уже погодили всі необхідні параметри сесії: шифронабор, сеансовий ключ, алгоритм коду автентифікації повідомлення. Призначення повідомлень Finished – одержати криптографічно стійке підтвердження, що сервер погодив ці параметри саме з тим вузлом, з яким передбачалося, то ж саме – і для клієнта, що одержить повідомлення Finished від сервера.

Клієнтське Finished містить відбиток – значення геш-функції, – від всіх попередніх повідомлень Handshake, відправлених, на даний момент, і клієнтом, і сервером. Одержавши це повідомлення в захищеної TLS-записи, сервер може обчислити значення геш-функції від відомих йому попередніх повідомлень і зрівняти результат. У такий спосіб підтверджується дійсність повідомлень і, відповідно, виявляються криптографічно захищені обраний шифронабор і інші параметри з'єднання. Втім, дана захист не позбавлена недоліків: наприклад, тому що повідомлення ClientHello і ServerHello не містять підписів, Finished ніяк не захищає сесію від атак, заснованих на підміні параметрів до відправлення ChangeCipherSpec. Цей дефект протоколу використаний у нашумілій в 2015 році атаці Logjam (про атаки на TLS/SSL ми докладно поговоримо нижче).

У відповідь на Finished з боку клієнта, сервер, у випадку успішного встановлення з'єднання, відправляє свою пару ChangeCipherSpec і Finished. На стороні клієнта отримане серверне повідомлення Finished також використовується для перевірки, що повідомлення Handshake не піддалися модифікації активним атакуючої, котрий перехопив канал. Состав Finished

сервера відрізняється від клієнтського варіанта, тому що включає у свій состав клієнтське повідомлення (і відмінну від клієнтської додатковий рядок).

Третій стороні, що активно перехоплює канал зв'язку, для того, щоб успішно зробити підміну повідомлень Handshake, буде потрібно обчислити сеансовий ключ і інші секретні параметри (якщо вони використовувалися) для того, щоб сфабрикувати коректне повідомлення Finished. Це розрахунково важке завдання, звичайно нерозв'язна на практиці, якщо обрані правильні налаштування протоколу й коректна реалізація. Однак, у випадку використання нестійких шифронаборів, що атакує може на лету обчислити секретний сеансовий ключ і – успішно підробити повідомлення Finished.

У пропонованому варіанті TLS 1.3 порядок повідомлень помітно міняється. Це стосується й Finished. Серверне повідомлення Finished відправляється відразу після повідомлень, які відносяться до генерації загальних ключів, тобто, раніше, ніж клієнт відправить своє Finished. Після того як клієнт і сервер обмінялися парами ChangeCipherSpec і Finished, захищене з'єднання успішно встановлене й дані можуть передаватися в закритому виді.

Подивимося на ServerHello і наступні повідомлення сервера в деталях, прикладом послужить відповідь сервера на повідомлення ClientHello, що розбиралося вище. У цьому прикладі кілька повідомлень TLS передаються в одній TLS-записі.

```
Зсув Байти                    Коментар
(десяткове)    (шістнадцяткове подання)
0000 16 03 02       ; Тип запису – Handshake (0x16) – версія TLS 1.1 (0x0302).
0003 07 02              ; Довжина пакета даних (0x0702 = 1794 байта – у цьому записі
                      утримуються всі відправлені сервером повідомлення –
                      ServerHello, Certificate, ServerKeyExchange,
                      ServerHelloDone – звідси й така довжина).
ServerHello. Початок повідомлення
0005 02             ; Тип повідомлення: ServerHello (02).
0006 00 00 46       ; Довжина даних: (0x000046 = 70 байтів – це дані
                      ServerHello).
0009 03 02          ; Версія протоколу: 0x0302 – TLS 1.1.
0011 55 dc              ; 32 байта ServerRandom, на початку поля сервер указує
```

```
                    timestamp.
 [...]
0043 20            ; Довжина SessionID – (0x20 = 32 байта).
0044 81 19            ; 32 байта, що містять переданий сервером ідентифікатор
                    сесії (SessionID).
 [...]
0076 c0 14            ; Обраний сервером шифронабор – 0xc014 -
                    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (протокол Діффі-Хеллмана
                    на еліптичних кривих для генерації сеансового ключа,
                    RSA – для підпису, AES з 256-бітним ключем у режимі CBC
                    як потоковий шифр, SHA-1 для генерації MAC).
0078 00            ; Обраний сервером метод стиску – 00 – null.
Certificate. Початок повідомлення
0079 0b            ; Повідомлення Certificate (тип – 0x0b = 11).
0080 00 05 21      ; Довжина повідомлення (0x000521 = 1313 байтів).
0083 00 05 1e      ; Довжина даних сертифікатів (0x00051e = 1310 байтів).
0086 00 05 1b      ; Довжина поля із сертифікатом. Кожний із сертифікатів,
зазначених
                    у повідомленні Certificate, випереджається трьома байтами,
                    утримуючу довжину запису даного сертифіката. Також передається
                    загальна довжина набору сертифікатів – звідси й ці "вкладені"
                    поля із числом байтів.
0089 30 82 05 17 30         ; Байти, що становлять сертифікат (сам сертифікат тут не
                    приводиться). Сертифікат передається сервером у бінарному
                    поданні, з використанням певного кодування.
                    На практиці це означає, що передаються сертифікати в
                    загальноприйнятому форматі DER, що підтримується більшістю
                    утиліт, наприклад з пакета OpenSSL і ін.
 [...]
ServerKeyExchange. Початок повідомлення
1396 0c 00 01 8b        ; Повідомлення ServerKeyExchange (тип – 0x0c = 12), довжина
даних
                    (0x00018b = 395 байтів). Тому що обрано шифронабор з ECDHE,
                    те повідомлення ServerKeyExchange містить параметри саме для
                    алгоритму ECDHE. Якихось спеціальних прапорів, що позначають
                    ServerKeyExchange як ECDHE, – не передбачено, всі
                    визначається шифронабором. Використання DH на еліптичних
                    кривих описане в RFC 4492
(https://tools.ietf.org/html/rfc4492).
1400 03            ; Тип кривій (0x03 – типова іменована крива).
1401 00 19            ; Ідентифікатор обраної сервером кривій (0x0019 – secp521r1,
                    реєстр кривих веде IANA, кожна з типових кривих містить
```

```
                    у своєму визначенні необхідні для DH параметри, які строго
                    зафіксовані: це генератор і розрядність групи точок).
1403 85           ; Довжина поля публічного ключа, що йде слідом
                    (0x85 = 133 байта).
1404 04 00 8f 96 86     ; 133 байта, що представляють собою публічний ключ ECDHE
                    сервера. Публічний ключ – це точка на кривій, що сервер
обчислив,
                    використовуючи параметри кривої. Відповідно до формату запису
                    точок, зазначеним клієнтом у розширенні ClientHello,
передаються
                    аффинні координати точки (x,y) (запис випереджається
заголовком).
                    Клієнтові параметри кривій відомі, тому що використовується
                    типова крива.
 [...]
1537      01 00             ; Довжина підпису, що відповідає параметрам DH
                    (0x0100 – 256 байтів). За значенням відкритого ключа DH треба
                    серверний підпис, виконаний по алгоритму й із ключем,
                    відповідної зазначеним у серверному сертифікаті (у нашім
                    випадку це RSA). В TLS 1.2 формат подання підпису трохи
                    іншої: у складі ServerKeyExchange додані поля, що позначають
                    використовуваний для генерації підпису на параметрах DH набір з
                    геш-функції й криптосистеми електронного підпису (наприклад,
RSA+ SHA-256).
1539      25 23 f5      ; Значення підпису (повністю не приводиться).
 [...]
ServerHelloDone. Початок повідомлення
1795 0e 00 00 00    ; Повідомлення ServerHelloDone (тип – 0x0e = 14). Це
повідомлення має
                    нульову довжину, тому слідом за типом ідуть три байти з
нульовими
                    значеннями (це байти, що кодують довжину,).
```

Скорочений формат Handshake

Установлення TLS-з'єднання – багатоступінчастий процес, досить складного й потребуючого проведення помітної кількості обчислювальних операцій, особливо жадібними в плані процесорного часу виявляються операції перевірки підписів і інших дій з асиметричними криптосистемами. Установлення з'єднання TLS – затратно, особливо якщо ваш онлайн-сервіс обслуговує велика кількість клієнтів. Для економії ресурсів існує скорочена версія Handshake.

У складі ServerHello сервер передає SessionID – ідентифікатор нової сесії. Цей ідентифікатор може бути використаний клієнтом пізніше, у складі ClientHello (див. вище). Передбачається, що на стороні сервера (і клієнта) збережені необхідні параметри, які можуть бути швидко відновлені, відновивши тим самим сесію.

```
  Client                                               Server
  ClientHello ------->
                                                  ServerHello
                                                [ChangeCipherSpec]
                                  < ----            Finished
   [ChangeCipherSpec]
  Finished ------->
  Application Data              < ------->     Application Data
```

Установлення з'єднання за скороченою схемою

Джерело: RFC 5246

У випадку використання скороченої схеми, відразу після одержання ClientHello, що містить валідний ідентифікатор сеccії SessionID, сервер відповідає повідомленнями ServerHello, ChangeCipherSpec і Finished (цей варіант, до речі, нагадує пропоновану в TLS 1.3 послідовність). Після того як клієнт надішле свої CCS і Finished, сесія відновляється й вузли можуть почати обмін даними в захищеному режимі.

Скорочений сценарій містить істотно менше повідомлень, дозволяє не використовувати розрахунково витратні операції перевірки підписів і генерації загального секрету, крім того, через меншу кількість повідомлень помітно знижується затримка при встановленні з'єднання (заощаджується час, необхідне на відправлення пакетів від клієнта до сервера й назад). Сучасні браузери широко використовують скорочений Handshake. Сесії можуть зберігатися на серверах протягом декількох мінут і навіть годин.

Зараз використовуються й інші способи поновлення сесій. Наприклад, RFC 5077 уводить поняття тікета сесії (TLS Session Ticket) – це розширення, у рамках якого сервер передає клієнтові зашифроване подання серверного

контексту TLS. Клієнт може відновити сесію, передавши в складі ClientHello збережений тікет.

Після того як клієнт і сервер установили з'єднання (по повній або скороченій схемі), вони обмінюються TLS-записами з типом Application Data, кожний такий запис містить блок даних, зашифрований симетричним ключем, а також код автентифікації повідомлення. Код автентифікації захищає повідомлення від зміни. Клієнт і сервер самостійно ведуть облік отриманих і переданих блоків.

TLS має ряд важливих, і досить загальних, криптографічних особливостей: по-перше, цей протокол ніяк не приховує факт установлення з'єднання (як зазначено вище, системи DPI можуть відносно впевнено детектувати початок сеансу TLS за допомогою простих правил-фільтрів); по-друге, в TLS можливі витоки позначка-інформації: число переданих блоків, різні попередження, передані у відкритому виді – все це дозволяє пасивному спостерігачеві зробити деякі виводи про характер переданої інформації й, у випадку WEB, про тип чинених користувачем на WEB-ресурсі дій. TLS лише більш-менш надійно захищає від перехоплення й підміни саму передану інформацію.

Повторне проведення Handshake

Під час роботи в рамках TLS-сесії клієнт і сервер можуть зштовхнутися з необхідністю повторно провести Handshake. Такий випадок передбачений протоколом. Хрестоматійна ситуація, коли це може знадобитися, виглядає в такий спосіб: після того як установлене TLS-з'єднання, клієнт, що використовує HTTP, запитує документ по деякому URL, однак цей URL вимагає додаткової авторизації з використанням клієнтського сертифіката. Клієнтський сертифікат може бути переданий тільки в складі повідомлення Handshake, тому сервер відправляє клієнтові повідомлення HelloRequest, що вимагає повторного обміну повідомленнями Handshake (уже із клієнтським сертифікатом). Клієнт може ініціювати нову сесію в будь-який момент, передавши повідомлення ClientHello

(в TLS 1.3 цю можливість планується виключити: сервер повинен буде закривати з'єднання з фатальною помилкою, одержавши ClientHello після встановлення сесії). Тому що вузли вже погодили TLS-з'єднання, обмін повідомленнями повторного Handshake буде проводитися в захищеному виді. У прикладі із клієнтським сертифікатом це означає, що сертифікат не буде видний стороні, що прослуховує канал.

Попередження (Alert)

Специфікація TLS передбачає обмін повідомленнями з типом Alert. Це попередження й повідомлення про помилки. Наприклад, у випадку, якщо сервер не зміг коректно розібрати повідомлення ClientHello, він відповідає повідомленням Alert, що містить код помилки Parse Error. Повідомлення Alert іноді можуть бути використані в складі атак на TLS-Вузли: так, за допомогою відправлення цих повідомлень можна "підвішувати" сесію, що коли перехоплює пристрою потрібен час на, наприклад, обчислення ключа.

Після того, як вузли погодили криптографічний контекст і обмінялися повідомленнями ChangeCipherSpec – повідомлення Alert передаються в зашифрованому виді. Тобто, із цього моменту зміст попередження не може бути прочитано що прослуховує трафік стороною, але сам тип запису – Alert – однаково передається у відкритому виді, що, відповідно, може приводити до витоку інформації про стан вузлів і з'єднання.

Сеансові ключі

Асиметричні криптосистеми (з відкритим ключем) не підходять для швидкої передачі потоків даних. Тому для шифрування даних усередині сесії TLS використовує симетричні шифри. Звичайно це блокові шифри, використовувані в режимі, подібному з потоковим (наприклад, GCM). Є кілька основних способів вироблення загальних даних для сеансового ключа. Сеансовий ключ, при цьому, повинен по зрозумілих причинах залишатися секретним. Про симетричний сеансових ключі вузли домовляються в процесі встановлення з'єднання (Handshake). Основні методи генерації загального секрету розглянуті вище,

залишилося обговорити деякі подробиці. Насамперед: симетричних ключів використовується пара – один для переданих повідомлень, другий – для прийнятих. Серверному ключу для переданих повідомлень (Write) відповідає клієнтський ключ для прийнятих (Read), і навпаки.

Основа ключів – загальний секрет Master Secret – генерується з декількох змінних складових: так званий Premaster Secret, ClientRandom і ServerRandom. Premaster Secret – погодиться в рамках обміну ключами, це або послідовність випадкових байтів, зашифрована відкритим ключем сервера, або значення, отримане в результаті обміну за протоколом Діффі-Хеллмана. Master Secret – це масив з 48 байтів, одержуваний у результаті застосування клієнтом і сервером до цих складових псевдовипадкової функції, певною специфікацією. Псевдовипадкова функція (PRF) TLS 1.2 побудована на базі геш-функції SHA-256 (або може бути використана більше "потужна" геш-функція, зазначена в складі шифронабора), що попередні версії використовують конструкцію на базі сполучення MD5 (яка давно не вважається криптографічно стійкою) і SHA-1 (яку перестали вважати досить стійкою в 2015 році). При цьому, спосіб використання MD5 при генерації сеансового ключа не приводить до виникнення уразливостей, пов'язаних з недостатньою якістю даної функції. Те ж саме можна сказати й про SHA-1. RFC 5246 визначає Master Secret для TLS 1.2 у такий спосіб:

```
master_secret = PRF(pre_master_secret, "master secret",
                ClientHello.random + ServerHello.random) [0..47];
```

Master Secret – це ще не сеансовий ключ. Справа в тому, що різні шифри вимагають різних ключів і додаткових даних (наприклад, ініціалізуючих векторів). Ці ключі й дані обчислюються на основі Master Secret, з використанням тої ж псевдовипадкової функції. Робиться це в такий спосіб (TLS 1.2):

– для обраного шифронабора визначається кількість байтів, необхідне для одержання ключа й ініціалізуючої інформації;

– потрібне число байтів (Key Block) виходить на основі Master Secret, ServerRandom, ClientRandom за допомогою послідовних викликів усе тої ж псевдовипадкової функції;

– отриманий Key Block розбивається на підмножини, потрібні для ініціалізації й роботи шифру, а також для обчислення коду автентифікації повідомлень (MAC). При цьому можуть розрізнятися ключі й вектори ініціалізації сервера й клієнта (але повний набір криптографічних параметрів відомий обом вузлам).

Саме шифрування здійснюється на рівні TLS-записів.

Більшість методів одержання вихідних даних сеансового ключа так чи інакше оперують публічним ключем сервера. На практиці цей ключ використовує майже 100% TLS-сесій. Публічний ключ передається сервером у сертифікаті. Крім розглянутих вище варіантів RSA і Діффі-Хеллмана, можливі екзотичні схеми одержання загального секрету:

1. **PSK – pre-shared key.** Схема заснована на використанні загального секретного ключа й симетричної криптосистеми, за умови, що ключ був погоджений заздалегідь;

2. **Тимчасовий ключ RSA** – історичний метод, що припускав створення сеансового ключа RSA. Зараз даний метод не використовується, однак його підтримка послужила основою для атаки FREAK, опублікованої в 2014 році;

3. **SRP** – протокол SRP (Secure Remote Password), RFC 5054. Протокол, що дозволяє згенерувати загальний симетричний секретний ключ достатньої стійкості на основі відомого клієнтові й серверу пароля, без розкриття цього пароля через незахищений канал;

4. **Анонімний DH** – анонімний варіант протоколу Діффі-Хеллмана, у якому не використовується підпис на серверних параметрах. Така схема піддається атаці типу "людина посередині", практично не зустрічається;

5. **DH із сертифікатом** – варіант DH, у якому параметри (модуль, генератор і відкритий ключ сервера) визначені в серверному сертифікаті. Цей метод є історичним, вимагає спеціального сертифіката й на практиці не зустрічається. Основна його відмінність від використовуваних зараз варіантів (нерідко називаних також "ефемерним Діффі-Хеллманом", хоча подібного

терміна варто уникати) полягає в тому, що серверний відкритий ключ протоколу Діффі-Хеллмана виявляється зафіксований: саме він входить у сертифікат і підписується центром, що засвідчує. Це означає, що схема не має прогресивну таємність. Сервер щораз використовує той самий секретний ключ Діффі-Хеллмана, а розкриття цього ключа дозволяє розшифрувати раніше записаний трафік TLS.

У розроблювальній специфікації TLS 1.3 підхід до генерації секретних ключів перероблений. Так, ключі розділені на класи, що відповідають фазі протоколу. Для захисту трафіку використовуються ключі, одержувані після встановлення з'єднання. Також пропонується (і, швидше за все, увійде у фінальну версію специфікації) інший алгоритм генерації ключів на основі інформації з контексту сесії. Зміни досить істотні, однак основна логіка залишається незмінної: сеансові ключі генеруються з параметрів, переданих у складі повідомлень Handshake, загального секрету, наявного на момент генерації у вузлів, і деяких констант, які визначені протоколом.

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

Сертифікати, які застосовуються в розробленій системі

1. TLS сертифікати RapidTLS:

– RapidTLS®.

– RapidTLS® WildCard.

2. TLS сертифікати Geotrust:

– GeoTrust QuickTLS® Basic.

– GeoTrust QuickTLS® Premium.

– GeoTrust True BusinessID Wildcard.

– GeoTrust TrueBusinessID Multi-Domain.

– GeoTrust TrueBusinessID EV Multi-Domain.

– GeoTrust True BusinessID with EV Certificate.

Сертифікати:

1. TLS сертифікати RapidTLS.
2. TLS сертифікати Geotrust.
3. TLS сертифікати Thawte.
4. TLS сертифікати VeriSign.
5. TLS сертифікати Comodo.
6. TLS сертифікати GlobalSign.
7. TLS сертифікати DigiCert.

Алгоритми перетворення інформації з метою збереження конфіденційності:

- Обмін ключами й перевірка їх дійсності: RSA, Diffie-Hellman, ECDH, SRP, PSK.
- Автентифікація: RSA, DSA, ECDSA.
- Симетричне шифрування: RC2, RC4, IDEA, DES, Triple DES, AES, Camellia.
- Геш-функція: SHA, MD5, MD4, MD2.

Інтерфейс користувача системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування

Блок реалізації алгоритму системи передачі даних у мережі за протоколом TLS

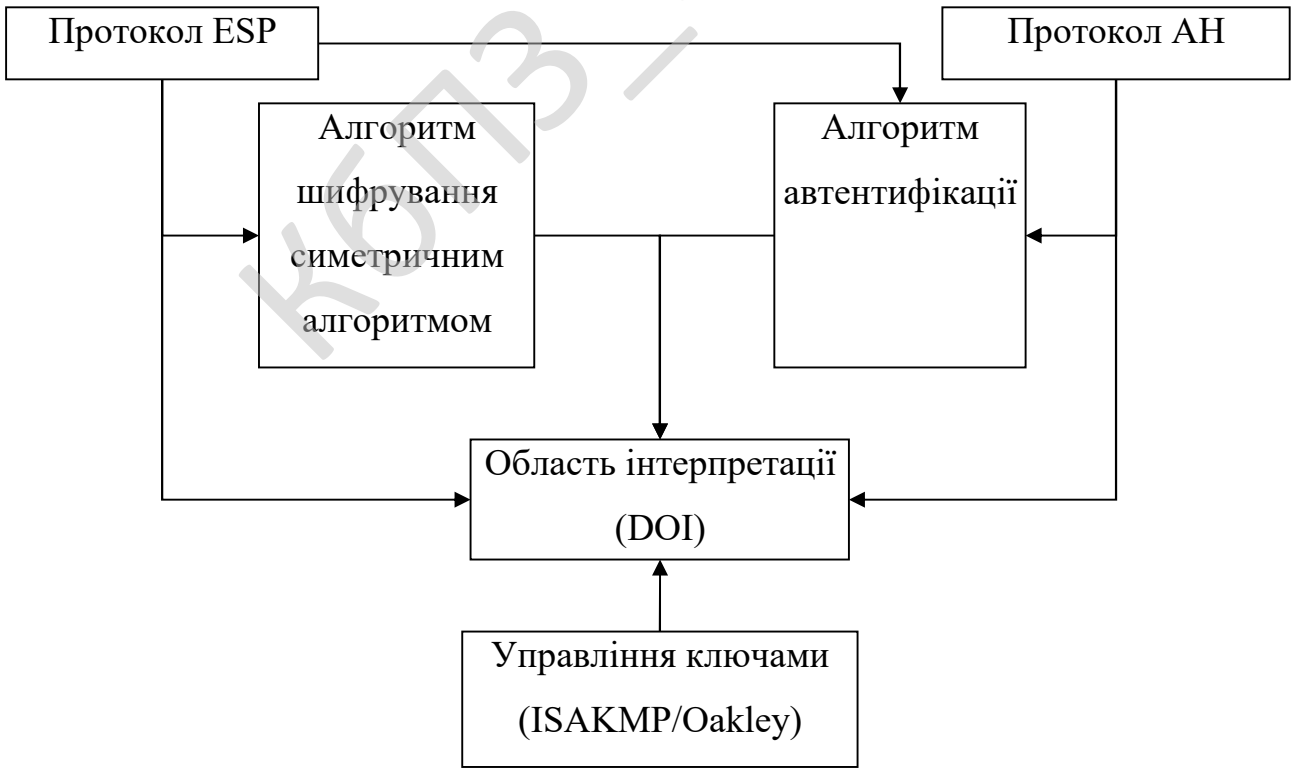

Рисунок 3.2 – Функціональна схема системи

3. TLS сертифікати Thawte:

– Thawte TLS123.

– Thawte TLS Web Server.

– Thawte TLS Web Server with EV.

– Thawte Code Signing.

– Thawte SGC SuperCerts.

– Thawte TLS Webserver Wildcard.

4. TLS сертифікати VeriSign:

– VeriSign Secure Site.

– VeriSign Secure Site EV.

– VeriSign Secure Site Pro.

– VeriSign Secure Site Pro EV.

– VeriSign Code Signing.

5. TLS сертифікати Comodo:

– Comodo Positive TLS.

– Comodo InstantTLS.

– Comodo InstantTLS Premium.

– Comodo InstantTLS Pro.

– Comodo Multi-Domain Certificate.

– Comodo EV Multi-Domain.

– Comodo EV SGC Certificate.

– Comodo EV TLS Certificate.

– Comodo Positive TLS Wildcard.

– Comodo Premium Wildcard.

– Comodo SGC TLS Wildcard.

– Comodo SGC TLS.

– Comodo Unified Communications.

– Comodo Secure Email Certificate.

6. TLS сертифікати GlobalSign:

– GlobalSign DomainTLS Certificate.

– GlobalSign OrganizationTLS.

– GlobalSign EV TLS Certificate

7. TLS сертифікати DigiCert:

– DigiCert TLS Plus.

– DigiCert EV Certificate.

– DigiCert WildCard Certificate.

Алгоритми, що використовуються в TLS

– Для обміну ключами й перевірки їхньої дійсності застосовуються: RSA, Diffie-Hellman, ECDH, SRP, PSK.

– Для автентифікації: RSA, DSA, ECDSA.

– Для симетричного перетворення інформації з метою збереження конфіденційності: RC2, RC4, IDEA, DES, Triple DES або AES, Camellia.

– Для хеш-функцій: SHA, MD5, MD4 і MD2.

Заголовок ESP – інкапсуляція зашифрованих даних

У випадку використання інкапсуляції зашифрованих даних заголовок ESP є останнім у ряді опціональних заголовків, "видимих" у пакеті. Оскільки основною метою ESP є забезпечення конфіденційності даних, різні види інформації можуть вимагати застосування істотно різних алгоритмів перетворення інформації з метою збереження конфіденційності. Отже, формат ESP може перетерплювати значні зміни залежно від використовуваних криптографічних алгоритмів. Проте, можна виділити наступні обов'язкові поля: SPI (SPI – Secutity Parameter Index – індекс параметра безпеки), що вказує на контекст безпеки, поле порядкового номера, що містить послідовний номер пакета, і контрольна сума, призначена для захисту від атак на цілісність зашифрованих даних. Крім цього, як правило, у тілі ESP присутні параметри (наприклад, режим використання) і дані (наприклад, вектор ініціалізації) застосовуваного алгоритму перетворення інформації з метою збереження

конфіденційності. Частина ESP заголовка може бути зашифрована на відкритому ключі одержувача або на спільному ключі пари відправник-одержувач. Одержувач пакета ESP розшифровує ESP заголовок і використовує параметри й дані застосовуваного алгоритму перетворення інформації з метою збереження конфіденційності для декодування інформації транспортного рівня.

Заголовок AH

Автентифікуючий заголовок (AH) є звичайним опціональним заголовком і, як правило, розташовується між основним заголовком пакета IP і полем даних. Наявність AH ніяк не впливає на процес передачі інформації транспортного й більш високого рівнів. Основним і єдиним призначенням AH є забезпечення захисту від атак, пов'язаних з несанкціонованою зміною вмісту пакета, і в тому числі від підміни вихідної адреси мережного рівня. Протоколи більш високого рівня повинні бути модифіковані з метою здійснення перевірки автентичності отриманих даних.

Формат AH досить простий і складається з 96-бітового заголовка й даних змінної довжини, що складаються з 32-бітових слів. Назви полів досить ясно відбивають їхній зміст: Next Header указує на наступний заголовок, Payload Len представляє довжину пакета, SPI є покажчиком на контекст безпеки й Sequence Number Field містить послідовний номер пакета.

На відміну від алгоритмів обчислення контрольної суми, застосовуваних у протоколах передачі інформації з линиям зв'язку, що комутуються або по каналах локальних мереж і орієнтованих на виправлення випадкових помилок середовища передачі, механізми забезпечення цілісності даних у відкритих телекомунікаційних мережах повинні мати засоби захисту від внесення цілеспрямованих змін. Одним з таких механізмів є спеціальне застосування алгоритму MD5: у процесі формування AH послідовно обчислюється хеш-функція від об'єднання самого пакета й деякого попередньо погодженого ключа, а потім від об'єднання отриманого результату й перетвореного ключа. Даний механізм застосовується за замовчуванням з метою забезпечення всіх реалізацій

IPv6, принаймні, одним загальним алгоритмом, не підданим експортним обмеженням.

Протокол ISAKMP/Oakley

Завдання алгоритмів IPsec – справа непроста, для цього потрібен протокол керування сеансом. Протокол ISAKMP (Internet Security Association Key Management Protocol) є рамковою основою для такого протоколу, а протокол Oakley – це вже конкретна реалізація його на цій основі, призначена для спільного використання з IPsec.

Протокол Oakley має більш широкий набір функціональних можливостей, ніж необхідно для керування IPsec-сеансами. Реалізація ISAKMP/Oakley являє собою функціональну підмножину, достатню, щоб забезпечити безпечний спосіб повідомлення автентифікованих даних для генерації ключів і SA-параметрів. Обмін по протоколу ISAKMP/Oakley відбувається у двох режимах (фазах): основному й швидкому. Відповідно до протоколу Oakley, обмін починається в основному й триває у швидкому режимі. У першому режимі встановлюються угоди SA для обміну даними по протоколу Oakley, а в другому – по протоколу IPsec.

На один обмін в основному режимі може доводитися кілька обмінів у швидкому, так як час існування SA-угоди для протоколу Oakley може бути більш тривалим, ніж для протоколу IPsec. Завдяки обмеженому строку існування SA-угод комбінування в сеансі основного й швидкого режимів забезпечує дуже потужний захисний механізм обміну ключами.

Обмін ключами в основному режимі здійснюється по методу Діффі-Хелмана (DH), що вимагає інтенсивного використання обчислювальних ресурсів. Цей метод є механізмом розподілу відкритих ключів для безпечного обміну секретною інформацією без застосування якої-небудь інформації, заздалегідь відомим обом сторонам. Тому ним активно користуються для встановлення безпечних сеансів зв'язку в тих випадках, коли необхідний динамічний захист і коли кіцеві системи не належать однієй й тій же системі адміністративного

керування. Наприклад, метод DH можна використовувати в електронній комерції при встановленні з'єднання для передачі транзакцій між двома компаніями.

Хоча цей метод і вимагає більших обчислювальних ресурсів, при його застосуванні можливий компроміс між криптостійкістю алгоритму (при використанні менш довгих відкритих ключів) і необхідним об'ємом обчислень. Обмін ключами у швидкому режимі не вимагає великого об'єму обчислень, так як тут використовується набір простих математичних операцій. Існує обмеження припустимого числа швидких фаз, перевищення якого веде до того, що ключі, згенеровані в основній фазі, а потім використовувані у швидких фазах, виявляться під погрозою розкриття. На сьогоднішній день немає твердого правила, що визначає число швидких фаз на одну основну фазу; криптографи діють, керуючись загальними міркуваннями й з огляду на оперативну обстановку.

В основному режимі обоє учасника обміну встановлюють SA-угоди для безпечного спілкування один з одним по протоколу Oakley. У швидкому режимі SA-угоди встановлюються вже "від імені" протоколу IPsec або будь-якої іншої служби, який необхідні дані для генерації ключів або узгодження параметрів. Протокол Oakley розроблений таким чином, що він ніяк не пов'язаний з IPsec. Наприклад, для підвищення безпеки процесу встановлення сеансів його цілком можна використовувати разом із протоколом TLS (Secure Sockets Layer) версії 4.0 замість механізму обміну ключами TLS 3.0.

DOI – область інтерпретації

Протокол ISAKMP/Oakley не був спеціально розроблений для спільного використання із протоколом IPsec, тому виникає необхідність у так званій області інтерпретації (Domain Of Interpretation – DOI), що забезпечила б спільну роботу протоколів IPsec і ISAKMP/Oakley. Щоб інші протоколи також могли використовувати ISAKMP/Oakley, вони повинні мати власні DOI-області. У даний момент таких областей для інших протоколів не існує, але ситуація може змінитися на черговій конференції групи IETF або в тому випадку, якщо приватний розроблювач, наприклад фірма Netscape, вирішить використовувати

цей механізм. Більш докладно про це можна прочитати в документі "The Internet Key Exchange (IKE)", розробленому робочою групою IP Security Protocol Working Group (ftp://ftp.ietf.org/ internet-draft/draft-ietf-IPsec-isakmp-oakley-06.txt).

В основному режимі між сторонами погоджуються методи перетворення інформації з метою збереження конфіденційності, хешування, автентифікації й так звана група DH (їх усього чотири), що визначає криптографічну стійкість алгоритму відкритого розподілу ключів. Перша група DH характеризується високою стійкістю й дозволяє використовувати стандарт DES, у той час як для другої й третьої груп варто застосовувати Triple DES. Оскільки в основному режимі іноді потрібно передавати до шести пакетів, то, наприклад, при використанні космічного сегмента з великою тимчасовою затримкою, DES краще застосовувати з більш сильною групою DH. Тоді перед виконанням чергового основного режиму, сполученого з інтенсивними обчисленнями й обміном пакетами, вам удасться виконати більше обмінів у швидкому режимі.

Коли SA-угода для обміну по протоколу Oakley установлюється в основному режимі, створюється ланцюжок випадкових біт, що використовують для генерації ключів. Також визначається тривалість (за часом або кількістю переданих даних) "життя" SA-угоди Oakley і дані для генерації ключів до того, як буде потрібно наступний обмін в основному режимі.

Швидкий режим простіше основного, і узгодження SA для IPsec здійснюється за допомогою трьох пакетів. IPsec-ключі створюються за допомогою простих операцій піднесення в ступінь переданих в основному режимі даних. У швидкому режимі погодяться також алгоритми перетворення інформації з метою збереження конфіденційності й строки існування SA для IPsec-сеансів.

Згідно із цими строками визначається, як незабаром, залежно від часу або об'єму переданих даних, буде потрібно нове узгодження у швидкому режимі. Помітьте, є два різних строки існування SA-угоди. Основний режим задає його для протоколу Oakley, а швидкий – для обміну по протоколу IPsec. Як приклад

пропонуємо значення цих параметрів для перетворення інформації з метою збереження конфіденційності IPsec-сеансів за допомогою алгоритму DES: 15 хв або 10 Мбайт для швидкого режиму, і 60 хв або 40 Мбайт для основного. Ці числа варто збільшити для Triple DES і зменшити для ARCFour (в ARCFour застосовується 40-бітний, а в TripleDES – 112-бітний ключ). Такий підхід дозволяє збалансувати криптографічну стійкість сервісів IPsec і вартість накладних витрат на передачу пакетів ISAKMP/Oakley.

При генерації ключів в основному режимі сеанс можна примусово перервати на підставі відкликання сертифіката. Сертифікати кінцевих вузлів використовуються тільки під час основного режиму. Таким чином, при анулюванні одного із сертифікатів обмін перерветься тільки в основному режимі. Тимчасові обмеження, погоджені в основному й швидкому режимах, значно відрізняються друг від друга й залежать від типу даних і транзакцій, що використовують IPsec-з'єднання. Для правильного визначення цих обмежень із обліком, з одного боку, об'єму обчислень і навантаження на мережу, а з іншого боку – імовірності порушення захисту даних, потрібно деякий аналіз.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає

уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

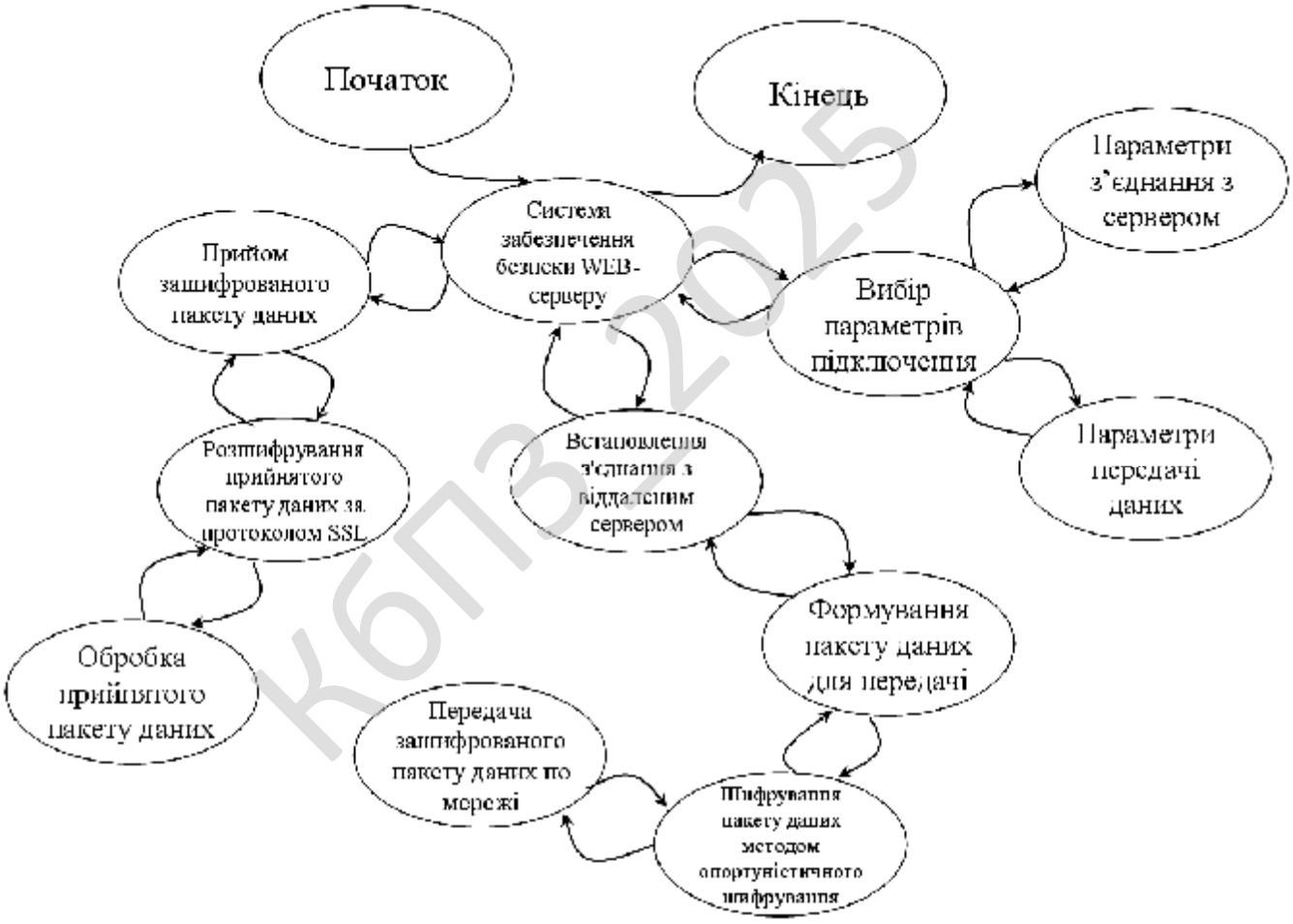

Рисунок 3.3 – Діаграма взаємодії процесів

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Під час роботи над бакалаврською дипломною роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Redmine – вільне серверне ПЗ для управління проектами та відстежування помилок. До системи входить календар-планувальник та діаграми Ганта для візуального представлення ходу робіт за проектом та строків виконання. Redmine написано на мові Ruby і є ПЗ розробленим з використанням відомого веб-фреймворку Ruby on Rails, що означає легкість в розгортанні системи та її адаптації під конкретні вимоги. Для кожного проекту можна вести свої вікі та форуми.

Функціональні можливості:

– Ведення декількох проектів.

– Гнучка система доступу з використанням ролей.

– Система відстеження помилок.

– Діаграми Ганта та календар.

– Ведення новин проекту, документів та управління файлами.

– Сповіщення про зміни за допомогою RSS-потоків та електронної пошти.

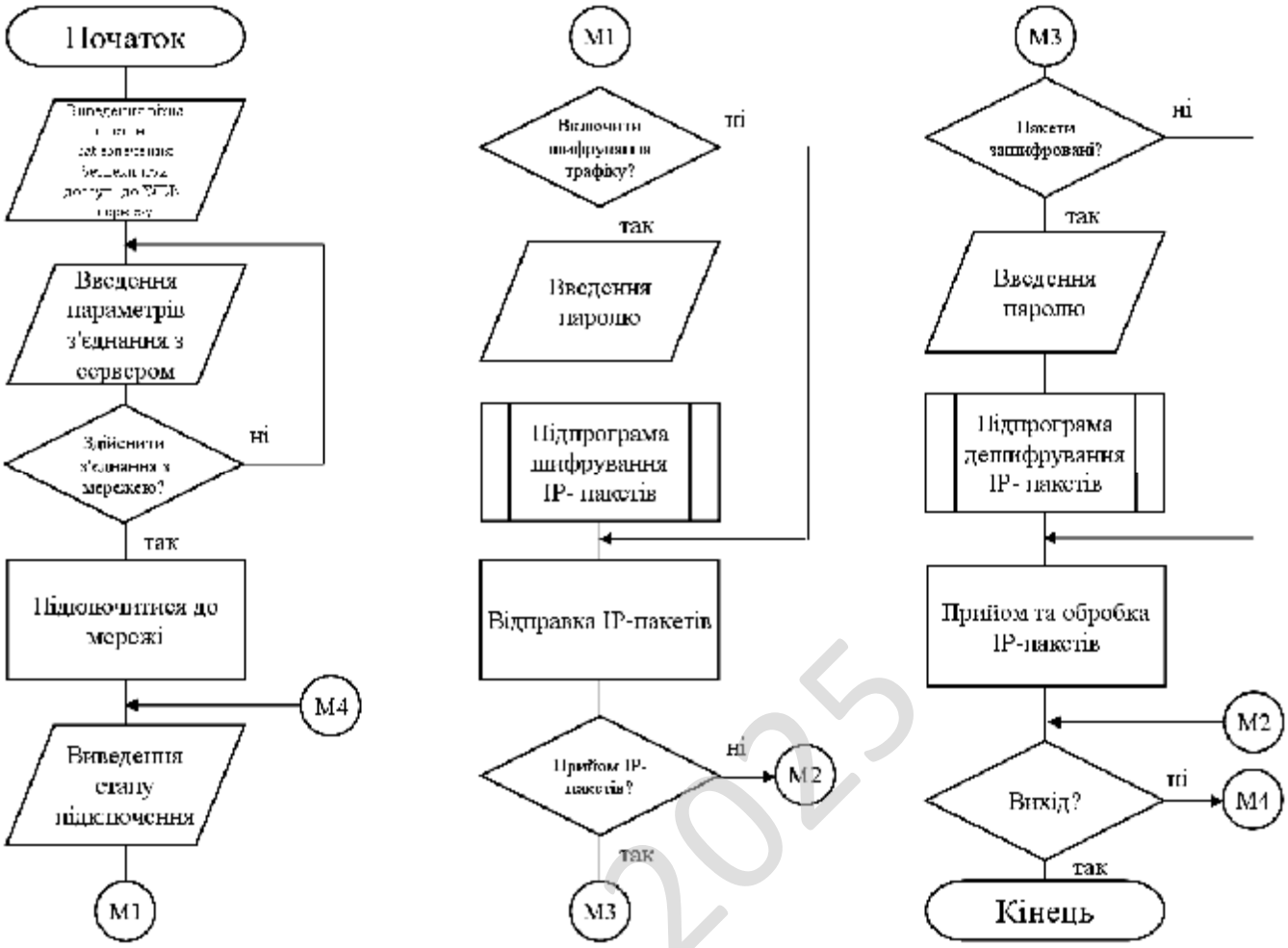

Рисунок 4.1 – Блок-схема основної програми

– Власна Wiki для кожного проекту.

– Форуми для кожного проекту.

– Облік часових витрат.

– Налаштування власних (custom) полів для задач, затрат часу, проектів та користувачів.

– Легка інтеграція із системами керування версіями (SVN, CVS, Git, Mercurial, Bazaar и Darcs).

– Створення записів про помилки на основі отриманих листів

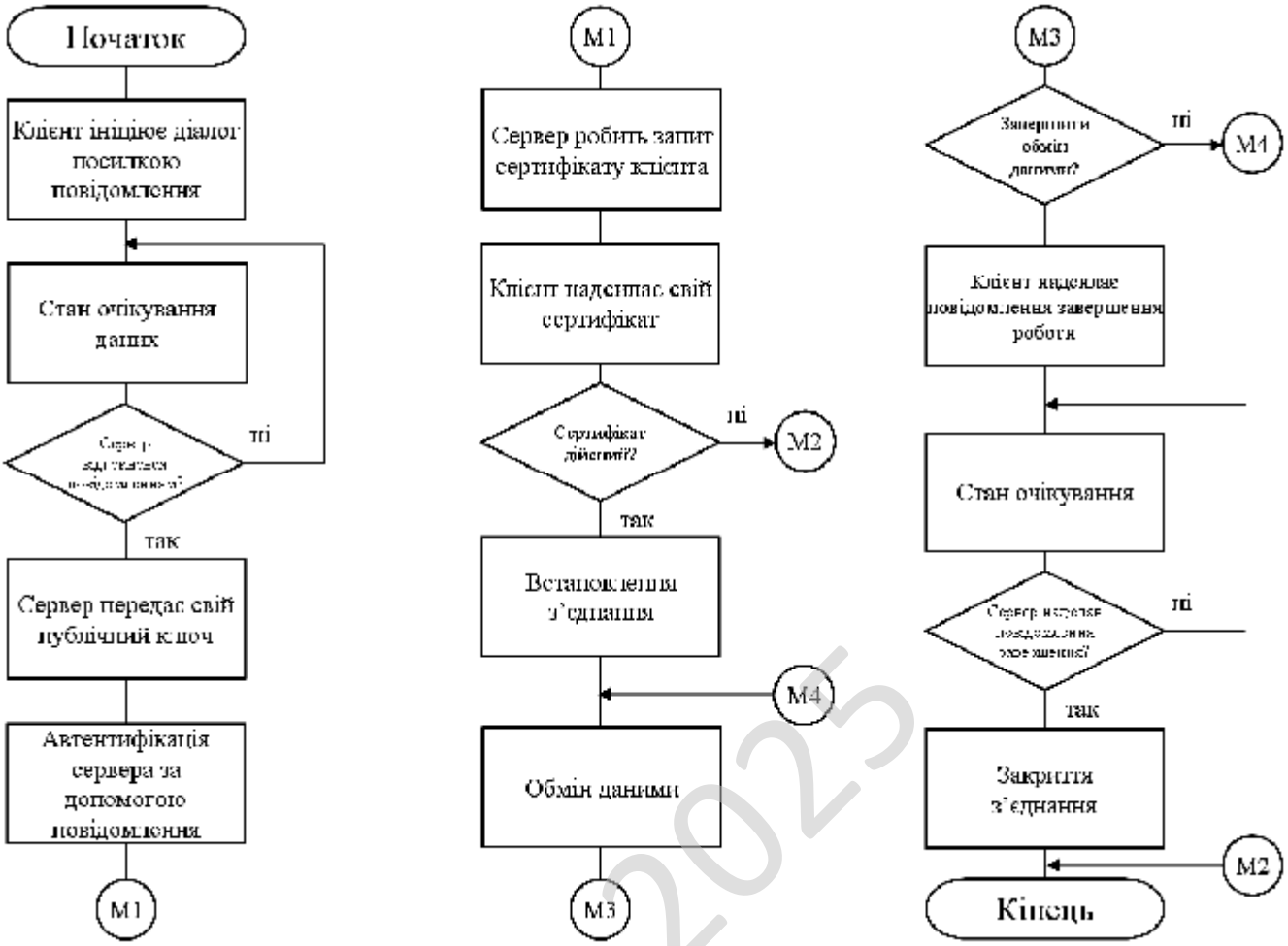

Рисунок 4.2 – Блок-схема роботи підпрограми

– Підтримка LDAP автентифікації.

– Можливість самореєстрації нових користувачів.

– Багатомовний інтерфейс (у тому числі українська мова).

– Підтримка СКБД: MySQL, PostgreSQL, SQLite.

Діаграма Ганта (Gantt chart, також стрічкова діаграма, графік Ганта) – це популярний тип діаграм, який використовується для ілюстрації плану, графіка робіт за будь-яким проектом. Є одним з методів планування та управління проектами.

Діаграма Ганта являє собою відрізки (графічні плашки), розміщені на горизонтальній шкалі часу. Кожен відрізок відповідає окремому завданню або підзадачі. Завдання і підзадачі, складові плану, розміщуються по вертикалі.

Початок, кінець і довжина відрізка на шкалі часу відповідають початку, кінцю і тривалості завдання. На деяких діаграмах Ганта також показується залежність між завданнями.

Діаграма може використовуватися для представлення поточного стану виконання робіт: частина прямокутника, що відповідає завданню, заштриховується, відзначаючи відсоток виконання завдання; показується вертикальна лінія, що відповідає моменту «сьогодні».

Часто діаграма Ганта використовується спільно з таблицею зі списком робіт, рядки якої відповідають окремо взятій задачі, зображеній на діаграмі, а стовпці містять додаткову інформацію про задачу.

Система відстеження помилок Багтрекер – прикладна програма для допомоги розробникам програмного забезпечення (програмістам, тестувальникам тощо) враховувати і контролювати помилки, знайдені у програмах, питання щодо функціональності, рішення та оновлення, побажання користувачів, а також стежити за процесом їх виконання.

Кожному, хто розробляв програмні продукти, добре знайоме співвідношення «20/80» – останні 20 % роботи тривають 80 % часу.

Як це не парадоксально, але нічого дивного в цій пропорції немає, адже саме на завершальній стадії починається тестування проекту, коли виявляються помилки, і що більший проект, то більше буде знайдено помилок.

Водночас досить часто виявляється, що більшість цих помилок були відомі та могли бути виправлені з меншими витратами на попередніх стадіях роботи, але не були вчасно описані, а потім загубилися серед інших важливих завдань.

Отже, система відстеження помилок у найпростішому варіанті – це процес, що включає в себе виявлення помилки, її опис, виправлення і перевірку цього виправлення, тобто процес «стеження» за багом протягом всього як його життєвого циклу, так і життєвого циклу розробки в цілому.

Сукупність інформації про дефект. Головний компонент такої системи – база даних, що містить відомості про виявлені дефекти. Ці відомості можуть включати в себе:

– номер (ідентифікатор) дефекту;

– хто повідомив про дефект;

– дата і час виявлення дефекту;

– версія продукту, в якій виявлено дефект;

– серйозність (критичність) дефекту та пріоритет рішення;

– опис кроків для відтворення дефекту (неправильної поведінки програми);

– відповідальний за усунення дефекту;

– обговорення можливих рішень та їх наслідків;

– поточний стан виправлення дефекту;

– версії продукту, в якій дефект виправлений.

Крім того, розвинені системи надають можливість прикріплювати файли, які допомагають описати проблему, наприклад, дамп пам'яті або скріншот.

Використання. Основна перевага систем відстеження помилок полягає в забезпеченні чітких централізованих оглядів, запитів на розробку (включаючи помилки і виправлення) та їх стан. У корпоративному середовищі, системи відстеження помилок можуть бути використані для генерації звітів по продуктивності програмістів виправлення помилок. Однак, це може іноді приводити до неточних результатів, тому що різні помилки можуть мати різні ступені пріоритету та серйозності, що пов'язано з складністю їх фіксації.

Життєвий цикл дефекту. Як правило, система відстеження помилок використовує той чи інший варіант «життєвого циклу» помилки, стадія якого визначається поточним станом помилки.

Типовий життєвий цикл дефекту:

1. Новий – дефект зареєстрований тестувальником.

2. Призначений – призначений відповідальний за виправлення дефекту.

3. Дозволений – дефект переходить назад у сферу відповідальності тестувальника. Як правило, супроводжується резолюцією, наприклад:

– Виправлено (виправлення включені у версію таку-то).

– Дубль (повторює дефект, що вже знаходиться в роботі).

– Не виправлено (працює відповідно до специфікації, має занадто низький пріоритет, виправлення відкладено до наступної версії тощо).

– «В мене все працює» (запит додаткової інформації про умови, в яких дефект проявляється).

4. Далі тестувальник проводить перевірку виправлення, залежно від чого дефект або знову переходить у стан «Призначений» (якщо він описаний як виправлений, але не виправлений), або у стан «Закрито».

5. Відкрито повторно – дефект знайдено знову в іншій версії.

Система може надавати адміністраторові можливість налаштування користувачі, які можуть переглядати і редагувати помилки залежно від їх стану, переводити їх в інший стан або видаляти.

У корпоративному середовищі, система відстеження помилок може використовуватися для отримання звітів, що показують продуктивність програмістів при виправленні помилок. Однак, часто такий підхід не дає достатньо точних результатів через те, що різні помилки мають різну ступінь серйозності та складності. При цьому серйозність проблеми прямо не стосується складності її усунення.

Хоча я реалізовував програму сам, було використано підходи Scrum для саморозвитку та пришвидшенню розробки, розглянемо цей метод. Scrum – підхід управління проектами для гнучкої розробки програмного забезпечення. Скрам чітко робить акцент на якісному контролі процесу розробки.

Підхід вперше описали Гіротака Такеучі та Ікуджіро Нонака в статті The New New Product Development Game (Гарвардський Діловий Огляд, січ–лют 1986). Вони відзначили, що проекти, над якими працюють невеликі, крос–функціональні команди, зазвичай систематично продукують кращі результати, і

пояснили це, як «підхід регбі». У 1991 році ДеҐрейс та Шталь у книжці Злі проблеми, справедливі рішення послалися на цей підхід, як на Scrum (штовханина; сутичка навколо м'яча (у регбі)), спортивний термін, згаданий в статті Такеучі і Нонака. Кен Швабер на початку 1990–х використовував підхід який привів Scrum в його компанію.

Вперше метод Scrum було представлено на загальний огляд задокументованим, чітко сформульованим та описаним спільно Сазерлендом та Швабером на OOPSLA'96 в Остіні. Швабер та Сазерленд протягом наступних років працювали разом щоб обробити та описати весь їхній досвід та найкращі практичні зразки для індустрії в одне ціле, в ту методологію, що відома сьогодні як Scrum. Швабер об'єднав зусилля з Майком Бідлом в 2001, щоб детально описати метод в книжці Agile Software Development with SCRUM. Не зважаючи на те, що для Scrum нарікли долю управління проектами з розробки ПЗ, він може також використовуватися в роботі команд обслуговувань програмного забезпечення (software maintenance teams), або як підхід управління розробкою і супроводом програм: Scrum of Scrums.

Scrum – це кістяк процесу, який включає набір методів і попередньо визначених ролей. Головні дійові особи – ScrumMaster, той хто опікується процесами, веде їх і працює як керівник проекту, Власник Продукту, людина, що представляє інтереси кінцевих користувачів та інших зацікавлених в продукті сторін, та Команду, яка включає розробників.

Протягом кожного спринту, 15-30 денного періоду (тривалість визначається командою), працівники створюють функціональний ріст програмного забезпечення.

Набір можливостей, які імплементуються кожного спринту, приходять з етапу, що має назву product backlog (документація запитів на виконання робіт), який має найвищу пріоритетність за рівнем вимог до роботи, що повинна бути виконана.

Запити на виконання робіт (backlog items), що визначені протягом наради з планування спринту (sprint planning meeting), переміщуються в етап спринту. Протягом цієї наради Власник Продукту інформує про завдання, які він хоче, аби були виконані. Тоді Команда визначає, скільки з бажаного вони можуть зробити, щоб завершити необхідні частини протягом наступного спринту. Протягом спринту команда виконує визначений фіксований список завдань (т.з. backlog items). Впродовж цього періоду ніхто не має права змінювати перелік запитів на виконання робіт, що слід розуміти, як заморожування вимог (requirements) протягом спринту.

Product backlog – це документ, який має список вимог до функціональності, які упорядковані згідно зі ступенем важливості. Product backlog представляє список того, що повинно бути реалізовано. Елементи цього списку називається «історіями» (user story) або елементами backlog–у (backlog items). Product backlog відкритий для редагування усім учасникам Scrum–процесу.

Обов'язкові поля:

1. ID – унікальний ідентифікатор, порядковий номер, який використовується для ідентифікації історій у разі їх перейменування.

2. Назва (Name) – стислий опис історії. Він повинен бути однозначним, щоб і розробники і product owner могли зрозуміти, про що йдеться і відрізнити одну історію від іншої.

3. Важливість (Importance) – ступінь важливості даної історії на погляд product owner ’а. Зазвичай являє собою натуральне число, іноді для цієї цілі використовуються числа Фібоначчі. Чим більше значення, тим більше пріоритет.

4. Попередня оцінка (initial estimate) – початкова оцінка об'єму робіт, необхідного для реалізації історії порівняно з іншими історіями. Вимірюється у story point'ах. Приблизно відповідає числу «ідеальних людино–днів».

5. Як продемонструвати (how to demo) – стисле пояснення того, як завершена задача буде продемонстрована у кінці спринта. Дане поле може являти собою код автоматизованого приймального тесту.

Додаткові поля. Іноді, також, використовуються додаткові поля у product backlog, в основному для того, щоб допомогти product owner'у визначитися з його пріоритетами.

Категорія (track). Наприклад, «панель управління» чи «оптимізація». За допомогою цього поля product owner може легко вибрати усі пункти категорії «оптимізація» і задати їм низький пріоритет.

Компоненти (components) – указує, які компоненти (наприклад, база даних, сервер, клієнт) будуть зачеплені при реалізації історії. Дане поле складається з групи checkbox'ів, які відмічаються, якщо відповідні компоненти потребують змін.

Ініціатор запиту (requestor). Product owner може захотіти зберігати інформацію про усіх замовників, зацікавлених у даній задачі. Це потрібно для того, щоб тримати їх у курсі діла про хід виконання робіт.

ID у системі обліку помилок (bug tracking ID) – якщо ви використовуєте окрему систему обліку помилок, тоді у описі історії корисно зберігати посилання на всі дефекти, які до неї відносяться.

Sprint backlog – містить функціональність, обрану Product Owner із Product Backlog. Всі функції розбиті по задачах, кожна з яких оцінюється командою. Кожен день команда оцінює об'єм роботи, який необхідно провести для завершення задачі.

Burndown chart – показує, скільки вже виконано і скільки ще залишається зробити.

Планування спринта (Sprint Planning Meeting)

Проходить на початку нової ітерації Спринта:

– Із Product Backlog обираються задачі, зобов'язання по виконанню яких за спринт приймає на себе команда;

– На основі обраних задач створюється Sprint Backlog. Кожна задача оцінюється у ідеальних людино–годинах;

– Рішення задачі не повинно займати більше 12 годин або одного дня. При необхідності задача розбивається на підзадачі;

– Обговорюється та визначається, яким чином буде реалізовано цей об'єм робіт;

– Тривалість наради обмежена зверху 4–8 годинами в залежності від тривалості ітерації, досвіду команди тощо;

– (перша частина наради) Беруть участь Product Owner + Команда: обирають задачі із Product Backlog;

– (друга частина наради) Бере участь лише команда: обговорюють технічні деталі реалізації, наповнюють Sprint Backlog.

Щоденна нарада (Daily Scrum Meeting)

Відбувається кожен день протягом спринта. Є «пульсом» ходу спринта. Нараді властиві наступні обмеження:

– починається точно вчасно;

– всі можуть спостерігати, але говорять тільки обрані;

– триває не більш ніж 15 хвилин;

– проводиться в одному і тому ж місці протягом одного спринта.

Протягом наради кожен член команди відповідає на 3 запитання:

– Що зроблено з моменту попередньої щоденної наради?;

– Що буде зроблено з моменту поточної наради до наступної?;

– Які проблеми заважають досягненню цілей спринта? (Над рішенням цих проблем працює ScrumMaster. Зазвичай це рішення проходить за рамками щоденної наради і у складі осіб, що безпосередньо займаються даною перешкодою.)

Демонстрація (Sprint Review Meeting):

– Проходить у кінці ітерації (спринта).

– Команда демонструє внесок функціональності до продукту всім зацікавленим особам.

– Залучається максимальна кількість глядачів.

– Усі члени команди беруть участь у демонстрації (одна людина на демонстрацію або кожен показує, що зробив за спринт).

– Обмежена 4–ма годинами в залежності від тривалості ітерації і змін у продукті.

Ретроспектива (Sprint Retrospective):

– Члени команди висловлюють свою думку про минулий спринт.

– Відповідають на два основних запитання: Що було зроблено добре у минулому спринті?; Що потрібно покращити в наступному?.

– Виконують покращення процесу розробки (вирішують питання та фіксують вдалі рішення).

– Обмежена 1-3ма годинами.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм MMB, в основі якого лежить змішування операцій різних алгебраїчних груп. MMB – ітеративний алгоритм, що складається з лінійних дій (XOR і використання ключа) і паралельного застосування чотирьох великих оборотних нелінійних підстановок. Ці підстановки визначаються за допомогою множення по модулю 2^{32}-1 з постійними множниками. У підсумку з'являється алгоритм, що використовує 128-бітовий ключ і 128-бітовий блок.

Алгоритм MMB оперує 32-бітовими підблоками тексту (x_0, x_1, x_2, x_3) і 32-бітовими підблоками ключу (k_0, k_1, k_2, k_3). Це спрощує реалізацію алгоритму на сучасних 64-бітових процесорах. Чергуючись із операцією XOR, шість разів використовується нелінійна функція *f*. Запишемо операції алгоритму (всі операції з індексами виконуються по модулю 4):

$$x_i = x_i \oplus k_i \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+1} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+2} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_i \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+1} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+2} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

Функція f виконується в три кроки:

1. $x_i = c_i * x_i$ для $i = 0..3$ (Якщо на вході множення одні одиниці, то на виході – теж одні одиниці).

2. Якщо молодший значущий біт $x_0 = 1$, то $x_0 = x_0 \oplus C$. Якщо молодший значущий байт $x_3 = 0$, то $x_3 = x_3 \oplus C$.

3. $x_i = x_{i-1} \oplus x_i \oplus x_{i+1}$ для $i = 0..3$.

Всі операції з індексами виконуються по модулю 4. Операція множення на кроці 1 виконується по модулі 2^{32}-1. Спеціальний випадок для даного алгоритму: якщо другий операнд дорівнює 2^{32}-1, результат теж дорівнює 2^{32}-1. В алгоритмі використовуються наступні константи:

C = 2aaaaaaa, c_0 = 025f1cdb, $c_1 = 2*c_0$, $c_2 = 2^3 * c_0$, $c_3 = 2^7 * c_0$.

Константа C – «найпростіша» константа без кругової симетрії, високою трійковою вагою й нульовим молодшим значущим бітом. У константи c_0 є інші особливі характеристики. Константи c_1, c_2 і c_3 – зрушені версії c_0, і служать для запобігання атак, заснованих на симетрії.

Розшифрування виконується у зворотному порядку, Етапи 2 і 3 інверсні їм самим. На етапі 1 замість c_i використовується c_i^{-1}. Значення c_0^{-1} = 0dad4694.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Навігаційне меню: Файл; Дії; Сертифікати та ключі; Імпорт; Параметри; Довідка.

– Розділу обрання групи.

– Розділу виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

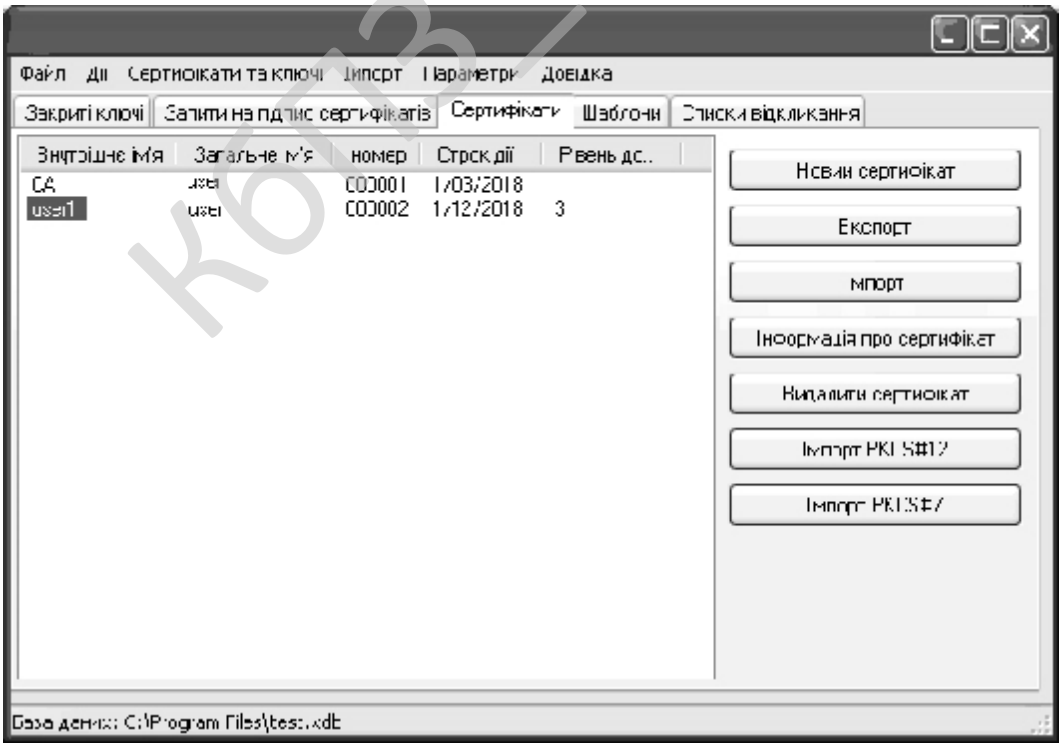

Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і інтуїтивно зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий.

Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

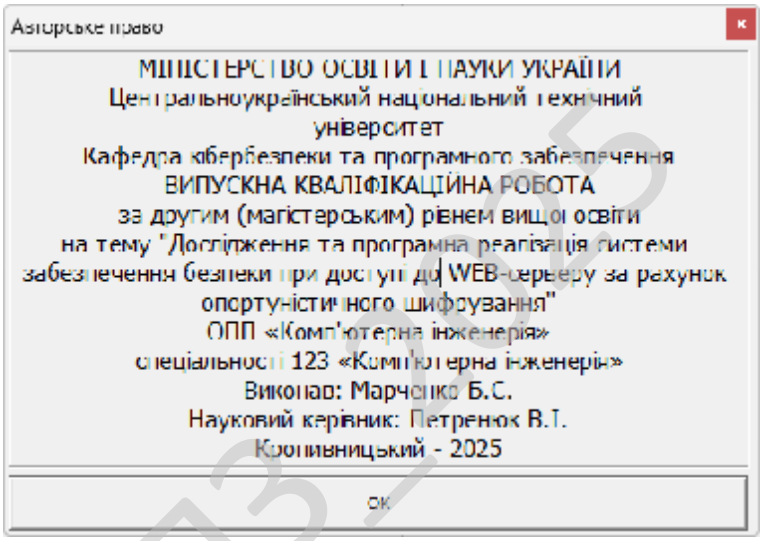

Рисунок 5.2 – Авторське право

Розглянемо процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом. Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення.

Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника так і з боку споживача. Оскільки кожна програмна система є

унікальною, то усі процеси та процедури під час розгортання важко передбачити. Тому, "розгортання" можна трактувати як загальний процес відповідно до певних вимог та характеристик. Розгортання може здійснюватись програмістом і в процесі розробки програмного забезпечення.

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

– Випуск.

– Встановлення та активація.

– Деактивація.

– Адаптація.

– Обновлення.

– Вмонтування.

– Відстежування версій.

– Видалення.

– Вилучення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

– Виділення критичних, з точки зору загального результату, процедур в діяльності організації. Коли набір таких процедур визначений, необхідно в першу чергу використовувати IT рішення для автоматизації операцій усередині саме цих процедур. Таким чином, розроблене IT рішення автоматично стає життєво важливим і затребуваним для організації, а також буде забезпечена публічність процесу впровадження;

– Розширення нормативної бази організації шляхом включення до неї регламентів, що описують порядок виконання процедур автоматизованих процесів. В іншому випадку є небезпека виникнення неузгодженості між автоматизованими процедурами та іншими процесами організації.

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються

в ІТ рішення за принципом найбільшої корисності для більшості учасників. Відсоток таких процедур щодо загального обсягу автоматизації може бути невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування форматом білої скриньки засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

– Виконуються всі цикли (у межах їхніх кордонів та діапазонів).

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

– Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

– У програмі можуть бути пропущені деякі маршрути.

– Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

– Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

– Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

– При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

– Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Проводилось тестування чорної скриньки.

Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ITc).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його OT).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITc.

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору OT.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій.

– Помилок інтерфейсу.

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних.

– Помилок характеристик (необхідна ємність пам'яті і т.д.).

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – commercial software. Програмне забезпечення, створене комерційною організацією з метою отримання прибутку від його використання іншими, наприклад, шляхом продажу копій.

Найважливішою особливістю комерційний програмних продуктів є підтримка великих компаній, прямо зацікавлених у поширенні програм. Багато організацій надають виключно платну підтримку своїх продуктів, такий підхід, як правило, використовують організації надають відкриті вихідні коди. Для продуктів, що розповсюджуються на комерційній основі діють зазвичай безкоштовні служби підтримки, покликані збільшити рівень довіри у клієнтів і потенційних покупців.

Далеко не завжди, але як правило терміни критично важливих змін в комерційних продуктах значно менше, ніж у некомерційних проектів. Це пов'язано з тим, що над комерційним продуктом працюють цілі групи розробників і ця робота є їх основним заняттям. Розробникам-початківцям як правило доводиться шукати додаткові способи заробітку, і це збільшує час, що витрачається на доповнення і зміни програм. Так як основним рушійним фактором створення комерційного ПЗ є одержання прибутку, то комерційні програмні продукти першими заповнюють вільні ніші та пропонують варіанти вирішення завдань відразу по мірі виявлення вакууму в будь-якому секторі ринку.

Окремий вид комерційних програм, коли їх розробка оплачується безпосередньо замовником. Такі програми найчастіше позбавлені всіх переваг комерційних продуктів, оскільки мають обмежений бюджет, але більш адаптовані до вимог замовника, ніж аналоги.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Метою розробки є дослідження та програмна реалізація системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Об'єктом дослідження є процес забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Предметом дослідження є методи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Методи дослідження базуються на методах теорії захисту інформації та теорії побудови комп'ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

– Розроблено вітчизняний продукт забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати дослідження і програмної реалізації системи опортуністичного шифрування можуть бути цікавими широкому колу користувачів – від власників невеликих сайтів до великих корпоративних структур. У першу чергу це актуально для компаній, які працюють із персональними або фінансовими даними клієнтів, адже навіть невеликий витік інформації може завдати серйозних репутаційних і фінансових збитків. Для таких підприємств важливо мати ефективну, але водночас економічно доцільну систему безпеки, яку не складно впровадити.

Також ця система може зацікавити державні установи, які зобов'язані забезпечувати безпечну передачу інформації між користувачами та своїми онлайн-сервісами. Наприклад, при роботі з електронними чергами, цифровими документами або персональними кабінетами громадян. Опортуністичне шифрування забезпечує додатковий рівень безпеки без необхідності кардинальних змін у поточній ІТ-інфраструктурі.

Окрему зацікавленість можуть проявити освітні та наукові організації, які займаються розробкою інноваційних технологій захисту інформації. Вони можуть використати результати реалізації як приклад ефективного впровадження сучасних методів криптографічного захисту у реальні веб-середовища.

Крім того, для розробників веб-платформ, хостинг-провайдерів та ІТ-компаній цей проєкт може стати конкурентною перевагою. Вони зможуть пропонувати клієнтам рішення з підвищеним рівнем безпеки, що підвищить їхню довіру до сервісу та допоможе залучити нову аудиторію.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Для оцінки привабливості системи опортуністичного шифрування було проведено опитування серед експертів у сфері веб-розробки та кібербезпеки. Фахівці оцінювали рішення за кількома критеріями: рівень захисту, простота впровадження, вартість реалізації, вплив на продуктивність сервера та користувацький досвід. За результатами опитування система отримала середній бал 8,6 із 10, що свідчить про її високу практичну цінність і потенціал для широкого використання.

Більшість експертів наголосили, що головна перевага опортуністичного шифрування полягає у його універсальності. Воно не потребує від користувача спеціальних дій для захисту з'єднання, що робить технологію доступною навіть для невеликих компаній або індивідуальних розробників. При цьому воно не вимагає значних фінансових витрат на сертифікацію або налаштування.

Деякі експерти зазначили, що OE може бути особливо корисним для компаній, які прагнуть поступово перейти на повноцінне HTTPS-з'єднання. Це проміжне рішення дозволяє підвищити рівень безпеки вже зараз, не витрачаючи ресурси на складну інтеграцію.

У цілому результати експертної оцінки підтверджують високу привабливість системи, особливо для малого та середнього бізнесу, який не має великих IT-бюджетів, але прагне гарантувати безпечний доступ своїх клієнтів до веб-ресурсів.

7.3 Вибір методу оцінки вартості ПЗ

Для визначення вартості впровадження системи опортуністичного шифрування найдоцільніше застосувати витратний метод. Цей підхід базується на аналізі фактичних витрат, пов'язаних із закупівлею програмного забезпечення,

налаштуванням серверів, тестуванням та подальшим обслуговуванням системи. Оскільки опортуністичне шифрування не потребує значних капіталовкладень, цей метод дозволяє максимально точно оцінити реальну собівартість впровадження.

У випадку з веб-серверами цей метод також враховує витрати на оновлення сертифікатів, навчання системних адміністраторів і адаптацію існуючих додатків до нової системи безпеки. Оцінка витрат допомагає визначити, наскільки впровадження рішення є економічно обґрунтованим для конкретного підприємства.

Разом із витратним методом варто застосувати порівняльний підхід – оцінити альтернативні рішення, такі як повне SSL/TLS-шифрування або використання комерційних сервісів безпеки. Це дозволяє показати переваги ОЕ у контексті співвідношення ціни та якості.

Завдяки цьому комбінованому підходу компанія може прийняти зважене рішення, яке базується не лише на вартості, а й на ефективності та доцільності впровадження з точки зору реальних бізнес-потреб.

7.4 Розрахунок економічної ефективності від впровадження реалізованого ПЗ як фактору його привабливості

Компанія має корпоративний веб-сайт та клієнтський портал, через який користувачі передають персональні дані, здійснюють замовлення, авторизацію або завантаження файлів. До впровадження опортуністичного шифрування передача даних між клієнтом і сервером здійснювалася переважно через звичайні HTTP-з’єднання або базові HTTPS-канали з фіксованими сертифікатами, що створювало ризики перехоплення інформації (MITM-атаки), компрометації паролів та зниження довіри користувачів.

Для зменшення цих ризиків компанія впровадила систему опортуністичного шифрування, яка дозволяє автоматично переходити до

захищеного з’єднання (TLS) під час будь-якої взаємодії клієнта з сервером. Це рішення не потребує попереднього налаштування користувача, забезпечує гнучку адаптацію до різних браузерів і серверів, а також мінімізує ризики незашифрованого трафіку. Вхідні дані зафіксовано в таблиці 7.1.

Таблиця 7.1 – Вихідні дані для розрахунку

Показник	**До впровадження**	**Після впровадження**	**Економічний ефект**
Кількість інцидентів безпеки на рік	12	2	–10
Середня вартість ліквідації одного інциденту (аналіз, відновлення, втрати клієнтів)	75 000 грн	20 000 грн	–55 000 грн
Загальні витрати на інциденти на рік	900 000 грн	40 000 грн	–860 000 грн
Витрати на впровадження системи ОЕ (серверні сертифікати, налаштування, аудит)	—	300 000 грн	—
Щорічне обслуговування (оновлення сертифікатів, підтримка)	—	60 000 грн	—
Зростання кількості користувачів (довіра клієнтів, SEO-рейтинг HTTPS)	—	+15 %	+ прибуток

Розрахунок економічного ефекту демонструє наступне: економія від зниження кількості інцидентів безпеки – 750 000 грн/рік, сукупний річний економічний ефект – 1 500 000 грн/рік, чистий економічний ефект – 1 440 000

грн/рік, термін окупності (Payback Period) – 0,21 року (~2,5 місяці), коефіцієнт рентабельності (ROI) -480 %.

Додаткові переваги (немонетарні): зростання довіри клієнтів – наявність HTTPS із автоматичним шифруванням позитивно впливає на сприйняття безпеки користувачем, покращення позицій у пошукових системах – Google віддає перевагу сайтам із HTTPS, що забезпечує стабільний приплив нових користувачів, захист репутації бренду – уникнення витоків даних запобігає негативному PR та штрафам за порушення GDPR, зниження навантаження на службу підтримки – менше звернень від користувачів, які стикаються з проблемами авторизації чи безпеки, покращення показників SLA – стабільність з'єднання й автоматичний перехід до захищених каналів підвищують якість обслуговування. Таким чином, опортуністичне шифрування довело свою ефективність не лише як технічне рішення, а й як економічно вигідний елемент цифрової безпеки бізнесу.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Просування системи опортуністичного шифрування доцільно почати з популяризації серед розробників і власників веб-ресурсів. Для цього варто створити інформаційну кампанію у вигляді статей, відеоуроків і технічних гайдів, які пояснюватимуть, як просто інтегрувати технологію OE у вже існуючі сайти. Демонстрація реальних кейсів успішного впровадження допоможе зняти сумніви та продемонструвати переваги на практиці.

Далі важливо налагодити співпрацю з провайдерами хостинг-послуг, які можуть пропонувати OE як стандартну функцію в пакетах обслуговування клієнтів. Це дозволить масштабувати технологію та зробити її доступною широкому колу користувачів.

Крім того, варто брати участь у галузевих конференціях і виставках з тематики кібербезпеки, де можна презентувати рішення потенційним партнерам.

Особливу увагу слід приділити створенню демонстраційного середовища, де відвідувачі можуть протестувати систему в дії.

На фінальному етапі просування доцільно зосередитися на побудові спільноти користувачів ОЕ – розробників, адміністраторів і технічних спеціалістів, які обмінюватимуться досвідом і поширюватимуть позитивний імідж технології.

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для ефективної реалізації проєкту оптимізація каналів збуту має бути орієнтована на простоту та доступність технології. Найкраще запропонувати ОЕ як частину готових веб-пакетів або модулів безпеки для популярних CMS – WordPress, Joomla, Drupal. Це дозволить користувачам впровадити рішення буквально за кілька кліків, без потреби у складній технічній підготовці.

Важливо також налагодити партнерські зв’язки з компаніями, що займаються хмарними сервісами, оскільки саме вони можуть стати ключовим каналом розповсюдження продукту. У таких партнерствах ОЕ може бути включене до базового набору інструментів безпеки для клієнтів.

Ще один ефективний шлях – створення веб-платформи, де користувачі можуть отримати консультацію, технічну підтримку або автоматичне налаштування системи. Це сприятиме формуванню довіри та полегшить впровадження.

Крім того, оптимізація може передбачати розробку безкоштовної базової версії продукту з можливістю переходу на розширений функціонал. Такий підхід стимулюватиме користувачів спробувати технологію, а згодом перейти до комерційного використання.

7.7 Визначення ключових факторів успіху конкретного проєкту

Ключовими факторами успіху впровадження опортуністичного шифрування є поєднання простоти інтеграції, високого рівня безпеки та мінімальних витрат. У сучасних умовах бізнес цінує рішення, які не лише підвищують захист, але й не ускладнюють роботу користувачів і розробників.

Важливим чинником є сумісність ОЕ з різними типами серверів і браузерів, що дозволяє забезпечити універсальність рішення. Якщо система працює стабільно незалежно від платформи, це значно підвищує її привабливість для впровадження у різних галузях.

Ще одним фактором успіху є підтримка з боку ІТ-спільноти. Відкрита документація, активна комунікація з розробниками та швидке реагування на виявлені вразливості підвищують довіру користувачів і гарантують сталість проєкту.

У підсумку, успіх ОЕ визначається не лише технологічною досконалістю, а й тим, наскільки ефективно вона вирішує реальні проблеми безпеки, з якими стикаються підприємства щодня. Саме це робить технологію привабливою, корисною та перспективною для подальшого розвитку.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Сучасний розвиток технічного та технологічного стану виробництва передбачає постійну автоматизацію та оптимізацію виробничих процесів. Комп'ютер – невід'ємна складова сучасного життя. За допомогою обчислювальної техніки вирішують складні робочі задачі, ведуться наукові дослідження, створюються архітектурні креслення і твори мистецтва. Сьогодні, напевно, важко уявити компанію, господарська діяльність в якій здійснювалась би без використання комп'ютерної техніки.

Незважаючи на видиму безпеку та розвитку сучасних технологій, при роботі за комп'ютером є ряд чинників, які можу вплинути на здоров'я людини. Через масовий характер робіт, що виконуються працівниками за допомогою комп'ютера, законодавством України чітко врегульовано норми та вимоги до використання комп'ютерної техніки на підприємстві, безпосередньо й охорона праці на підприємстві при роботі за комп'ютером.

Законом України "Про охорону праці" [1] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями, затверджені наказом Мінсоцполітики від 14.02.2018р. № 207, зареєстровані в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 [2].

Робота з комп'ютером характеризується значною розумовою напругою і нервово-емоційним навантаженням операторів, високою напруженістю зорової роботи і достатньо великим навантаженням на м'язи рук при роботі з клавіатурою ЕОМ. Шкідливими факторами при роботі з персональним комп'ютером є неіонізуюче випромінювання промислової частоти, збільшене нервово-емоційне

навантаження на оператора, збільшення навантаження на органи зору та дрібні стереостатичні рухи кінцівок.

У розділі даної магістерської роботи висвітлюються основні питання охорони праці працівників, робота яких пов'язана з роботою за комп'ютером, планування робочого приміщення, де працюють користувачі ПК; параметри мікроклімату, освітленість робочих місць та виробничих приміщень; шумові завади.

Правильна організація і раціональне устаткування робочого місця можливість ефективно і з якнайменшими витратами праці виконувати свої функції, плідно спілкуватися співробітниками і підлеглими, підтримувати високу працездатність і робочий настрій.

Велике значення має раціональна конструкція і розташовує елементів робочого місця, що важливе для підтримки оптимальної робочої пози людини-оператора, а також необхідно дотримувати правильний режим праці і відпочинку.

Що стосується питання охорони праці людини необхідно вирішувати на всіх стадіях трудового процесу незалежно від виду професійної діяльності.

Забезпечення безпечних і здорових умов праці в значній мірі залежить від правильної оцінки небезпечних, шкідливих виробничих факторів. Однакові по складності зміни в організмі людини можуть бути викликані різними причинами. Це можуть бути фактори виробничого середовища, надмірне фізичне і розумове навантаження, нервово-емоційна напруга, а також різне сполучення цих причин. Робота працівників пов'язана з роботою за комп'ютером, тому актуальною є розгляд саме умов праці та стану охорони праці працівників які постійно працюють з комп'ютерною технікою.

Завдання даного розділу полягає у тому, щоб розробити якісний програмний продукт необхідно організувати безпеку на робочому місці програміста. Під час проектування безпеки робочому місці з ПК необхідно домагатися високої якості та надійності технічного забезпечення, але й створювати комфортні параметри довкілля для розробників.

8.2 Характеристика умов праці програміста

Приміщення в якому проводиться розробка і дослідження програмного продукту за завданням наведені в таблиці 8.1

Таблиця 8.1 – Характеристика умов праці

Шкідливі та небезпечні фактори на робочому місці	Джерела утворення небезпек	Примітка (дані наведені для приміщення)
Електрична напруга вище 127В; Шум; Електромагнітні випромінювання; Статична електрика; Іонізація повітря; Пожежна небезпека у приміщені; Не якісне освітлення;	Кондиціонер 6 ПЕОМ Принтер Папір Світильники	Розміри приміщення (м) Довжина – 9 Ширина – 6 Висота – 3 Кількість працюючих – 7

Згідно Державними санітарними правилами і нормами ДСанПіН 3.3.2.007-98 [3] може перебувати 7 працівників. Мінімальна площа приміщення на 1 людину повинна складати не менше 6 м^2. Висновок – за умовами завдання це виконується. В приміщенні відсутні умови, які можуть створювати підвищену або особливо підвищену небезпеку, тому воно відноситься до класу звичайних приміщень згідно ПУЕ. Джерелом живлення є трифазна мережа напруги 380/220 В з глухо заземленою нейтралі, з частотою 50 Гц. За пожежо-вибухонебезпекою приміщення відноситься до класу В.

В таблиці 8.2 наведена загальна характеристика приміщення щодо вибухопожежо-небезпеки та важкістю робіт. Температура повітря в приміщенні визначається температурою зовнішнього повітря і тепловою енергією, що виділяється всередині приміщення.

Таблиця 8.2 – Загальна характеристика приміщення щодо вибухопожежонебезпеки та важкістю робіт

Характеристика приміщень за вибухопожежною категорією та класом зони	Загальна характеристика приміщення	Категорія за важкістю робіт згідно ГН 3.3.5-8.6.6.1 -2002
В – пожежонебезпечне клас П – II	Звичайне без ознак хімічного забруднення та нормальної вологості	1а – до 139 Вт/м^2 1б – до 140-174 Вт/м^2 Клас умов праці – оптимальний

Джерелами тепла в даному приміщенні є люди, електроустаткування, а також освітлювальні прилади в темний час доби. Зовнішнім джерелом надлишкового тепла є сонячна радіація у світлий час доби. Робота, що виконуванується в даному приміщенні, відноситься до категорії І-а. Людиною в цьому випадку виділяється до 120 ккал теплової енергії в годину. Вологість повітря в приміщенні визначається вологістю атмосферного і видихуваного людьми повітря, а також випарами з поверхні шкіри.

У таблиці 8.3 приведені оптимальні значення параметрів мікроклімату для категорії ваги робіт І-а, а також фактичні значення цих параметрів у розглянутому приміщенні. У приміщеннях з використанням обчислювальної техніки рекомендується застосування тільки оптимальних значень показників мікроклімату, тобто таких, за яких людина почуває себе комфортно.

Таким чином, показники мікроклімату в приміщенні, загалом, відповідають установленим нормам. В холодний період року використовується індивідуальне опалення, завдяки якому підтримується температурний режим у приміщенні в залежності від температури повітря навколишнього середовища.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Воло-гість,%	Швидкість повітря, м/с	Температура,°С	Воло-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-24	40-55	0,12
Тепла	23-25	50-70	0,1	24-25	50-65	0,9

Для підтримки температури в літню пору встановлений кондиціонер, який має достатню потужність по холоду, а для підтримки вологості є зволожувач повітря.

Джерелами запиленості повітря в приміщенні є одяг людей і пил, що проникає з вулиці. З метою боротьби з пилом робляться регулярні вологі прибирання і провітрювання.

У приміщенні немає виділення шкідливих газів, тому що в ньому не проводиться монтажних робіт, пайки чи інших робіт, при яких виділяються шкідливі гази.

Для нормалізації параметрів повітряного середовища також періодично здійснюється провітрювання приміщення і вологе прибирання. У всьому будинку діє встановлена загально обмінна витяжна вентиляція.

Особливістю роботи за дисплеєм ЕОМ є постійна й значна напруга функцій зорового аналізатора, обумовленого необхідністю розходження самосвітних об'єктів (символів, знаків тощо) при наявності відблисків на екрані, рядковій структурі екрана, мерехтіння зображення, недостатньою чіткістю об'єктів розходження. По характеру зорової роботи, робота відноситься до високої точності, розряд зорової роботи III, підрозряд г.

Раціональне освітлення приміщення сприяє кращому виконанню виробничого завдання і забезпеченню комфорту при роботі. Для забезпечення нормального освітлення застосовуються природне, однобічне, бічне і штучне

освітлення, а також сполучене, які нормуються згідно ДБН В.2.5-28-2006 Природне і штучне освітлення [4]. Дані по нормах освітлення наведені в таблиці 8.4

Таблиця 8.4 – Норми освітлення

Мінімальний розмір об'єкта розрізнювання, мм	Фон	Контраст	Розряд, під розряд зорової роботи	Нормоване значення		
				Природне освітлення КПО,%	Штучне освітлення	
					$E_{мін.}$ лк	Тип ламп
Від 0,3 до 0,5	Світлий	Середній	IIIг	1,5	300	Газороз-рядні

За результатами виміру освітленості величина освітленості від системи загального штучного освітлення лежить у межах – 320 лк, що відповідає вимогам, які пред'являються до даного приміщення. Основними джерелами шуму на робочих місцях, обладнаних відео дисплейними терміналами, є принтер, сканер факс і обладнання для кондиціювання повітря, в самих відео дисплейних терміналів – вентилятори систем охолодження і трансформатори. Згідно ДСанПіН 3.3.2.007-98 [3] допустимий еквівалентний рівень шуму для робочого місця програміста складає 50 дБА, Допустимі параметри рівнів звуку та звукового тиску представлені в таблиці 8.5.

Таблиця 8.5 – Рівні звукового тиску від різних джерел.

Джерело шуму	Рівень шуму, дБА
Жорсткий диск	45
Вентилятор	45
Принтер	55
Сканер	50

8.3 Розробка заходів з умов поліпшення охорони праці

У комплексі заходів, що сприяють удосконаленню організації праці програміста, збереженню здоров'я і підвищення працездатності, його велике значення має організація робочих місць. Проводячи аналіз умов праці в розглянутому приміщенні, ми одержали наступні результати:

– обсяг приміщення, що приходиться на одному працюючого, відповідає нормативному значенню;

– показники мікроклімату відповідають нормативному значенню;

– акустичні умови роботи в нормі.

Важлива роль в ефективному забезпеченні праці належить моральному мікроклімату. Відносини працівників повинні ґрунтуватися на об'єктивності, доброзичливості, взаємодопомозі, глибокій повазі до кожного члена колективу, турботі про молодих співробітників.

Для забезпечення найбільш ефективного виконання обов'язків, плануючи розпорядок, слід дотримуватися таких принципів:

– для зняття втоми через кожні 1,5-2 год. робити перерви для відпочинку на 5-10 хв.;

– для усунення монотонності виконуваних робіт чергувати характер праці.

За умови неправильної організації праці та відпочинку, втома може нагромаджуватися щоденно й переходити в перевтому або захворювання. У зв'язку з цим режим праці та відпочинку користувачів ПК необхідно будувати з урахуванням працездатності, яка змінюється протягом доби.

Виходячи із наступного можна зробити висновок, що основною причиною втомлюваності та зниження працездатності працівника, який постійно працює за комп’ютером є психофізіологічний фактор, тому основною пропозицією правильна організація робочого місця з урахуванням ергономічних вимог, а також дотримання регламентованого режиму праці та відпочинку.

8.4 Розрахункова частина

Виконаємо розрахунок ризику травмування людей, зайнятих певним видом діяльності (в розрахунку за рік), якщо середньорічна кількість осіб, що займаються цією діяльністю – 600 , а за останні 4 років травми одержали 9 осіб. Порівняємо обчислений рівень ризику з нормованим (прийнятим на сьогоднішній день) у світовій практиці.

1. Визначаємо середньорічну кількість травмованих осіб, для цього ділимо кількість постраждалих на кількість років, за які сталися ці трагічні події:

$$N = К/Т \quad (8.1)$$

де, К – кількість постраждалих; Т – період часу; підставляємо значення і отримуємо:

$$N = 9 / 4 = 2.25 \text{ осіб на рік.}$$

2. Знаходимо величину ризику за формулою

$$R = n / N , \quad (8.2)$$

де n – кількість подій, які відбулись з небажаними наслідками, N – загальна кількість подій, що може відбутися.

$$R = 2.25 / 600 = 0.00375 = 3{,}75 \cdot 10^{-3}.$$

Далі порівняємо отриманий результат з величиною прийнятного ризику, який визначений у світовій практиці і дорівнює $1 \cdot 10^{-6}$.

Для цього отриманий результат ділимо на прийнятий ризик:

$$П = 0.003.75 / 0.000001 = 3750 \text{ разів.}$$

Відповідь: величина ризику становить $3.75 \cdot 10^{-3}$, а ризик травмування людей більший за прийнятий у 3750 разів. Крім того, можна розрахувати коефіцієнт частоти виникнення небезпечних ситуацій на 1000 чоловік, а для цього необхідно:

$$К = R \cdot 1000 \quad (8.3)$$

$$К = 0.00375 \cdot 1000 = 3.75.$$

Таким чином, коефіцієнт частоти дорівнює 3.75 людини на 1000 осіб.

8.5 Висновки до розділу

У даному розділі магістерської роботи розглянуті умови праці програміста, проведено аналіз санітарно-гігієнічних умов праці працівників, які зайняті роботою з комп'ютерною технікою. Наведено приклад розрахунку ризику травмування людей, зайнятих певним видом діяльності. Можна зробити наступний висновок, що шкідливі та небезпечні виробничі фактори існують практично на будь якому робочому місці. Тільки повна усвідомленість працівника про можливі небезпеки, що можуть підстерігати його на робочому місці та дотримання вимог нормативних актів о питань охорони праці та відповідних рекомендацій фахівців, дозволять значною мірою знизити негативний вплив шкідливих та небезпечних факторів при роботі з комп'ютером на організм людини.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

– Досліджена система забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

– На основі отриманих результатів досліджень створена програмна реалізація системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Builder C++. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм MMB.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Проведено маркетингове та економічне обґрунтування ІТ-проєкту, що дозволило визначити ключові фактори успіху даного проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Марченко Б.С. Дослідження та програмна реалізація системи забезпечення безпеки при доступі до WEB-серверу за рахунок опортуністичного шифрування // Збірник праць молодих науковців ЦНТУ. – Вип. 15. – Кропивницький: ЦНТУ, 2025.

2. Loren Kohnfelder. Designing Secure Software. No Starch Press. 2022. 332 p.

3. Samir Kumar Rakshit. Ethical Hacker's Penetration Testing Guide. BPB Online. 2022. 509 p.

4. Corey J. Ball. Hacking APIs. No Starch Press. 2022. 353 p.

5. Kevin Beaver. Hacking for Dummies. John Wiley & Sons. 2022. 419 p.

6. Mark S. Merkow. Practical Security for Agile and DevOps. CRC Press. 2022. 236 p

7. Derek Fisher. Application Security Program Handbook. Manning Publications. 2021. 155 p.

8. Cameron Wyatt PH.D. Kali Linux Tutorial. Independently published. 2021. 60 p.

9. Alex Matrosov, Eugene Rodionov, Sergey Bratus. Rootkits and Bootkits. No Starch Press. 2019. 450 p.

10. Петрик В.М., Присяжнюк М.М., Аль-Файюмі Халед та ін. «Системи інформаційної зброї та технології інформаційної війни»: підручник / Петрик В.М., Присяжнюк М.М., Аль-Файюмі Халед, Жарков Я.М., Смірнов О.А, Буравченко К.О., Давидюк А.В., Кононович В.Г., Корчинский В.В., Кудирко В.М., Фесенко А.О.; за заг. ред. В.М. Петрика, М.М. Присяжнюка.– К.: Видавничий центр “Кафедра”, 2025.– 320 с.

11. Усік, П.С., Смірнова, Т.В., Буравченко, К.О., Смірнов, О.А., Улічев, О.С., Смірнов, С.А. «Дослідження технологій забезпечення кібербезпеки

банківських систем з використанням штучного інтелекту». *Кібербезпека: освіта, наука, техніка.* 2025. Том 1 № 29. С.704–716, 2025

12. Kuznetsov, O., Frontoni, E., Kuznetsova, K., Arnesano, M., Smirnov, O. «A secure biometric authentication architecture for blockchain-driven cyber-physical systems». *Security and Privacy of Cyber Physical Systems Emerging Trends Technologies and Applications*, 2025, pp. 193–224.

13. Kuznetsov, O., Atzeni, G., Arnesano, M., Randieri, C., Smirnov, O. «Secure IoT-based smart wheelchair system: From implementation to security enhancement strategy». *Security and Privacy of Cyber Physical Systems Emerging Trends Technologies and Applications*, 2025, pp. 225–257.

14. Kuznetsov, O., Smirnov, O., Kuznetsova, T., Shaikhanova, A., Svatowsky, I. «Privacy-utility trade-offs in IoT networks: A comparative analysis of differential privacy mechanisms for sensor data aggregation». *Security and Privacy of Cyber Physical Systems Emerging Trends Technologies and Applications*, 2025, pp. 589–622.

15. Lakhno, V., Malyukov, V., Smirnov, O., Bebeshko, B., Chubaievskiy, V., Zhumadilova, M., Malyukova, I., Smirnov, S. «Multifactorial Model for Targeted Attacks Counteracting Within the Framework of a Multi-Step Quality Game with Fuzzy Information». *8th International Symposium on Intelligent Informatics, ISI 2023*, 2025. vol 389. pp 377-389. Springer, Singapore.

16. Kuznetsov O., Frontoni E., Kuznetsova Y., Smirnov O., Moskovchenko I. «Trust-Based Security Architecture for Edge Computing: A Simulation Study of Dynamic Trust Evolution and Attack Detection». *CEUR Workshop Proceedings,* 2024, 3909, pp. 227–241.

17. Kuznetsov, O., Frontoni, E., Kryvinska, N., Chevardin, V., Smirnov, O. «Wireless Network Encryption Stream Ciphers, Computational Modeling, and Security Analysis». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 379–402.

18. Kuznetsov, O., Frontoni, E., Kryvinska, N., Smirnov, O., Imoize, G.L. «Computational Modeling of Enhanced Spread Spectrum Codes for Asynchronous Wireless Communication». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 403–447.

19. Ткаченко, О., Ільєнко, А., Улічев, О., Мелешко, Є., Смірнов, О. «Правові засади поширення інформаційних впливів в соціальних мережах». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»,* 2024. № 2(26), С. 170–188.

20. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

21. Вінтенко, Б., Миронець, І., Смірнов, О., Кравчук, О., Козірова, Н., Савеленко, Г., Коваленко, А. «Дослідження вимог та аналіз кібербезпеки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Кібербезпека: освіта, наука, техніка.* 2024. №3(23), С. 111-131.

22. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

23. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

24. Akhalaia, G., Iavich, M., Iashvili, G., Prysiazhnyy, D., Smirnova, T. «Secure Encrypted Connection on Georgian Website». *CEUR Workshop Proceedings,* 2023, 3550, pp. 313-320.

25. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56

26. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

27. Kuznetsov, O., Kandiy, S., Frontoni, E., Smirnov, O. «Trade-offs in Post-Quantum Cryptography: A Comparative Assessment of BIKE, HQC, and Classic McEliece». *CEUR Workshop Proceedings,* Volume 3504, 2023, pp. 1-11.

28. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». *In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies*, vol 131. 2023. Springer, Singapore. pp. 21-34.

29. Смірнова Т.В., Гнатюк С.О., Бердибаєв Р.Ш., Сидоренко В.М., Жигаревич О.К., «Система корелювання подій та управління інцидентами кібербезпеки на об'єктах критичної інфраструктури». *Кібербезпека: освіта, наука, техніка*, №*3*(19), 2023, С. 176-196.

30. Смірнов О.А. Козлов Я.О., Смірнова Т.В. «Дослідження застосування SIEM-систем для забезпечення кібербезпеки та захисту інформації». *ІІ Міжнародна науково-практична Інтернет-конференція «Інновації та перспективні шляхи розвитку інформаційних технологій (ІПШРІТ-2023)»* м.Черкаси 6 грудня 2023 року – Черкаси: ЧДТУ.– 2023. – С.251-252.

31. Козлов Я.О., Смірнова Т.В., Смірнов О.А. «Дослідження SIEM-систем для забезпечення кібербезпеки». *VII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення,* м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 26.

32. Козлов Я.О., Козірова Н.Л., Смірнов О.А. «Дослідження структури та принципу роботи SIEM-системи». *VII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя*

кафедри кібербезпеки та програмного забезпечення, м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 59.

33. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А., Коваленко А.С. «Дослідження нормативних документів та галузевих стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 2(72), С. 170-178.

34. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022,

35. Smirnov O.A., Al-Oraiqat A.M., Ulichev O.S., Meleshko Ye.V., Al-Rawashdeh H.S., Polishchuk L.I. «Modeling strategies for information influence dissemination in social networks». *Journal of Ambient Intelligence and Humanized Computing* Volume 13, Issue 5. Springer, Cham. 2022, pp. 2463-2477.

36. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-телекомунікаційних систем». *Проблеми інформатизації та управління,* № 2(70). 2022. С. 28-37.

37. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

38. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

39. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

40. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021,* Cracow, Poland, 22-25 September 2021. P. 414-418

41. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». *4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) – 2021*, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

42. Smirnov O., Kuznetsov A., Girzheva O., Kiian A., Nakisko O., Kuznetsova T. «Advanced Code-Based Electronic Digital Signature Scheme». *2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020,* Kharkiv, 6 October 2020-9 October 2020, P. 358-362.

43. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova K. «Data hiding scheme based on spread sequence addressing». *CEUR Workshop Proceedings* Volume 2805, 2020, Pages *44-58.*

44. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». *International Journal of Computing*; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

45. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

46. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

47. Smirnov O., Kuznetsov A., Arischenko A., Chepurko I., Onikiychuk A., Kuznetsova T. «Pseudorandom sequences for spread spectrum image steganography». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 122-131.

48. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 1-14.

49. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». *Lecture Notes in Networks and Systems*, vol 152. Springer, Cham. 2021, pp 66-84.

50. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

51. Smirnov O., Kuznetsov A., Kiian A., Babenko V., Perevozova I.,Chepurko I. «New Approach to the Implementation of Post-Quantum Digital Signature Scheme». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 166-171.

52. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.