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
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів”

Виконав здобувач вищої освіти
IV курсу, групи КІ-20-3СК
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Ясніцький Д.П.
«____» _____ 2023 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Дрєєв О.М.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2023 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Ясніцькому Дмитру Павловичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів*

2. Керівник роботи *Дрєєв Олександр Миколайович, канд. техн. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 8-02 від 5.01.2023 року

3. Строк подання студентом роботи до захисту *23.05.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2023 р.	
3.	Розробка моделі компонента	20.03.2023 р.	
4.	Розробка структур даних	25.03.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2023 р.	
6.	Програмування алгоритмів	10.04.2023 р.	
7.	Оформлення ПЗ	17.04.2023 р.	
8.	Попередній захист роботи	23.05.2023 р.	

Дата видачі завдання
« 17 » січня 2023 р.

Підпис керівника
_____ Дрєєв О.М.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2023 р.

Підпис здобувача
_____ Ясніцький Д.П.
(прізвище та ініціали)

АНОТАЦІЯ

Ясніцький Д.П. Програмне забезпечення системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

Метою розробки є програмне забезпечення системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

Результат роботи – програмна реалізація системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

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 архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі Builder C++.

Ключові слова: комп'ютерна інженерія, IP-мережі, Cisco, хмарні сервіси

ABSTRACT

Yasnitskyi D.P. Software of the system of protected IP networks based on Cisco solutions for cloud services. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the first (bachelor) level of higher education, software is developed, which is intended for a system of protected IP networks based on Cisco solutions for cloud services.

The purpose of the development is the software of the protected IP network system based on Cisco solutions for cloud services.

The result of the work is a software implementation of a system of protected IP networks based on Cisco solutions for cloud services.

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 software tools are provided.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

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

Keywords: computer engineering, IP networks, Cisco, cloud services

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
SSL	–	протокол захищених сокетів
TCP	–	транспортний протокол
VPN	–	віртуальні частні мережі

ВСТУП

Актуальність теми. У наш час головною цінністю на планеті вважається інформація, отже її, як і всяку іншу цінність, людина намагається зберегти від сторонніх рук і очей. А так як зараз уже 21 століття, і поняття інформації нерозривно пов'язане з комп'ютерними технологіями, системами й мережами зв'язку, то стає очевидною важливість питання захисту інформації в них. Винахід комп'ютера й подальший бурхливий розвиток інформаційних технологій у другій половині 20 століття зробили проблему захисту інформації настільки актуальною й гострою, наскільки актуальна сьогодні інформатизація для всього суспільства. Особливо актуально стоїть це питання в області секретної інформації держави й приватної комерційної інформації.

У бізнесі сумлінна конкуренція припускає суперництво, засноване на дотриманні законодавства й загальновизнаних норм моралі. Однак нерідко підприємці, конкуруючи між собою, прагнуть за допомогою протиправних дій одержати інформацію на шкоду інтересам іншої сторони й використовувати її для досягнення переваги на ринку. Криміналізація суспільства й недостатня ефективність державної системи охорони правопорядку змушує представників бізнесу самим вживати заходів для адекватного протистояння негативним процесам, що має місце, що наносить збиток конфіденційній інформації фірми.

Причин активізації комп'ютерних злочинів і пов'язаних з ними фінансових втрат досить багато, істотними з них є:

– перехід від традиційної "паперової" технології зберігання й передачі відомостей на електронну й недостатній при цьому розвиток технології захисту інформації в таких технологіях;

– об'єднання обчислювальних систем, створення глобальних мереж і розширення доступу до інформаційних ресурсів;

– збільшення складності програмних засобів.

Отже, головна тенденція, що характеризує розвиток сучасних інформаційних технологій – ріст числа комп'ютерних злочинів і пов'язаних з ними розкрадань конфіденційної й іншої інформації, а також матеріальних втрат.

Ми все частіше чуємо в ЗМІ слова "інформаційна безпека", "зроблена нова атака на комп'ютери", "створений новий вірус", "збиток від атаки склав ... " і далі приводяться усе більш приголомшуючі цифри.... Всі ці проблеми пов'язані з низьким і недостатнім рівнем забезпечення безпеки в інформаційних системах. Виходить, необхідно забезпечувати безпека тих даних, які зберігаються в наших комп'ютерах і передаються в мережах.

Так як більшість даних передається по мережах, а в основному мережі зараз будуються на основі протоколу TCP/IP, то забезпечення захисту цього протоколу є актуальною задачею. Існує множина протоколів захисту мереж. Однак у зв'язку з поширенням Інтернет і передачею інформації по протоколу HTTP, особливу актуальність здобувають протоколи, які забезпечують захист саме по цьому протоколу передачі даних. Протоколами захисту, що виконують перераховані вище умови є протокол SSL і IPsec.

Серед фірм, які поставляють мережне встаткування й різні мережні рішення виділяється Cisco. Тому, мережа, розглянута в даному бакалаврському проекті, буде побудована на встаткуванні саме цієї фірми.

Компанія Cisco Systems допомагає вирішити більшість виникаючих в області інформаційної безпеки задач, починаючи від проектування середовища передачі даних у захищеному виконанні й закінчуючи впровадженням засобів виявлення й запобігання несанкціонованих дій, що відбуваються як зсередини системи, так і спрямовані на неї зовні.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

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

– Огляд існуючих систем захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

– Дослідження системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

– Програмна реалізація системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Сьогодні в області захисту віддаленого доступу досить жорстко конкурують технології безпеки мережного (IPsec) і транспортного (SSL/TLS) рівня.

Щоб пояснити суть розходжень цих рішень, необхідно ввести деяку класифікацію. На рисунку 1.1 показані три принципово різні системні архітектури:

а) IPsec VPN;

б) "класичний" сценарій застосування захисту транспортного рівня на базі протоколів SSL або TLS;

в) SSL (TLS) VPN.

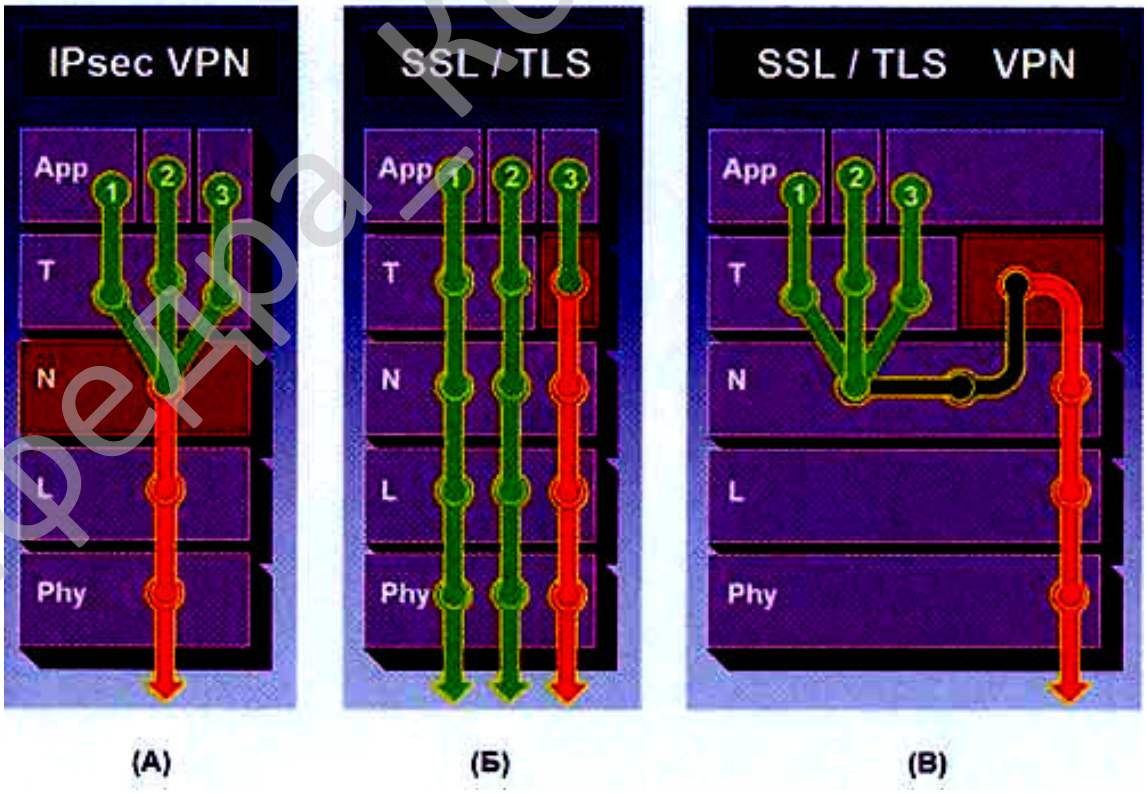

Рисунок 1.1 – Системні архітектури

У чому розходження? Насамперед варто сказати, що всі три архітектури використовують приблизно однаковий набір порівнянних по стійкості криптографічних алгоритмів. Таким чином, за критерієм криптографічної стійкості рішення можна вважати приблизно однаковими.

Що ж стосується комунікаційного протоколу – рішення IPsec одночасно й більш гнучке, і більш складне як по реалізації, так і в експлуатації. В архітектурі IPsec (рисунок 1.1, а): трафік додатків передається на мережний рівень у вигляді пакетів, пакети перехоплюються, шифруються й підписуються. Спеціальний протокол, Internet Key Exchange, IKE, піклується про керування ключами й узгодження політик безпеки.

Консалтингова компанія The Burton group в одному з аналітичних звітів стверджувала, що VPN-рішення на базі протоколу IPsec для систем з більш високими вимогами по безпеці кращі в порівнянні з SSL/TLS VPN.

Що стосується "класичної" схеми застосування протоколів SSL/TLS (рисунок 1.1, б), те це зовсім не VPN.

Як затверджує один з останніх по строку публікації глосаріїв IETF, VPN – це "спосіб використання відкритих або приватних мереж таким чином, щоб користувачі VPN були відділені від інших користувачів і могли взаємодіяти між собою, як якби вони перебували в єдиній закритій мережі".

Класичне SSL/TLS-рішення не відповідає цьому визначенню. Воно забезпечує цілісність і конфіденційність одного транспортного з'єднання (трафік додатка 3 на рисунку 1.1, б). Інші ж додатки (1 і 2 на рисунку 1.1, б) передають відкритий трафік, і доступ до їх "незашифрованого" портам ніхто не контролює. Така незамкнутість контуру захисту комп'ютера в SSL/TLS мене завжди дуже бентежила...

Але в цього рішення є своя, практично неконкурентна, "екологічна ніша". У системах захисту масового (не корпоративного) віддаленого доступу, де ми не можемо встановити кожному користувачеві "наш" VPN-клієнт, альтернативи Web-браузеру, що підтримує SSL– або TLS-з'єднання, просто немає. Тому

електронна комерція, системи доступу приватних осіб до банківських ресурсів та інші системи з масовим і непідконтрольним парком користувачів побудовані за схемою Web-браузер – захищене SSL/TLS з'єднання -портал.

IPsec, навпроти, являє собою архітектуру, що вичерпним образом покриває функціональність VPN. Тут трафік всіх додатків, переданий ними через "свої" транспортні порти, перетворюється в IP-пакети, які шифруються, підписуються й туннелюються через недовірені мережі.

При "ізолюючій" політиці безпеки IPsec, коли відкритий IP-трафік повністю захищений, доступ у мережу одержує строго певне коло власників секретних ключів. Пробити цю "броню" сторонньому неможливо.

Технології SSL/TLS VPN (рисунок 1.1, в) одержали поширення відносно недавно, близько 5 років тому. Мотивом для створення VPN-продуктів на основі транспортних протоколів була ідея створити більш просте й дешеве в експлуатації VPN-рішення. Однак тут, як це часто трапляється в сюжеті "хотіли як краще", виникли нюанси. Рішення "розповзлося" у дві сугубо різні архітектури: "псевдо-VPN" і "чесний VPN", які істотно розрізняються по своїх властивостях.

Архітектура "псевдо-VPN" показана на рисунку 1.2. У якості VPN-клієнта в "псевдо-VPN" виступає простий Web-браузер.

Шлюз безпеки на зовнішньому периметрі корпоративної мережі являє собою захищений Web-сервер, що "показує" VPN-клієнтам ресурси мережі, які їм захищаються, у вигляді Web-ресурсів. При цьому ресурси, що захищаються, зовсім не обов'язково повинні являти собою Web-сторінки. Це можуть бути файлові системи, навіть чат, голос і відео. Щоб зробити ці ресурси доступними через браузер, "за" Web-сервером розташовується проксі-сервер, що транслює дані ресурси в HTTP і обернено.

У чому принадність такого рішення? У тім, що для побудови VPN із клієнтської сторони зовсім нічого не потрібно. Браузер є на будь-якій машині. SSL або TLS є майже в кожному браузері.

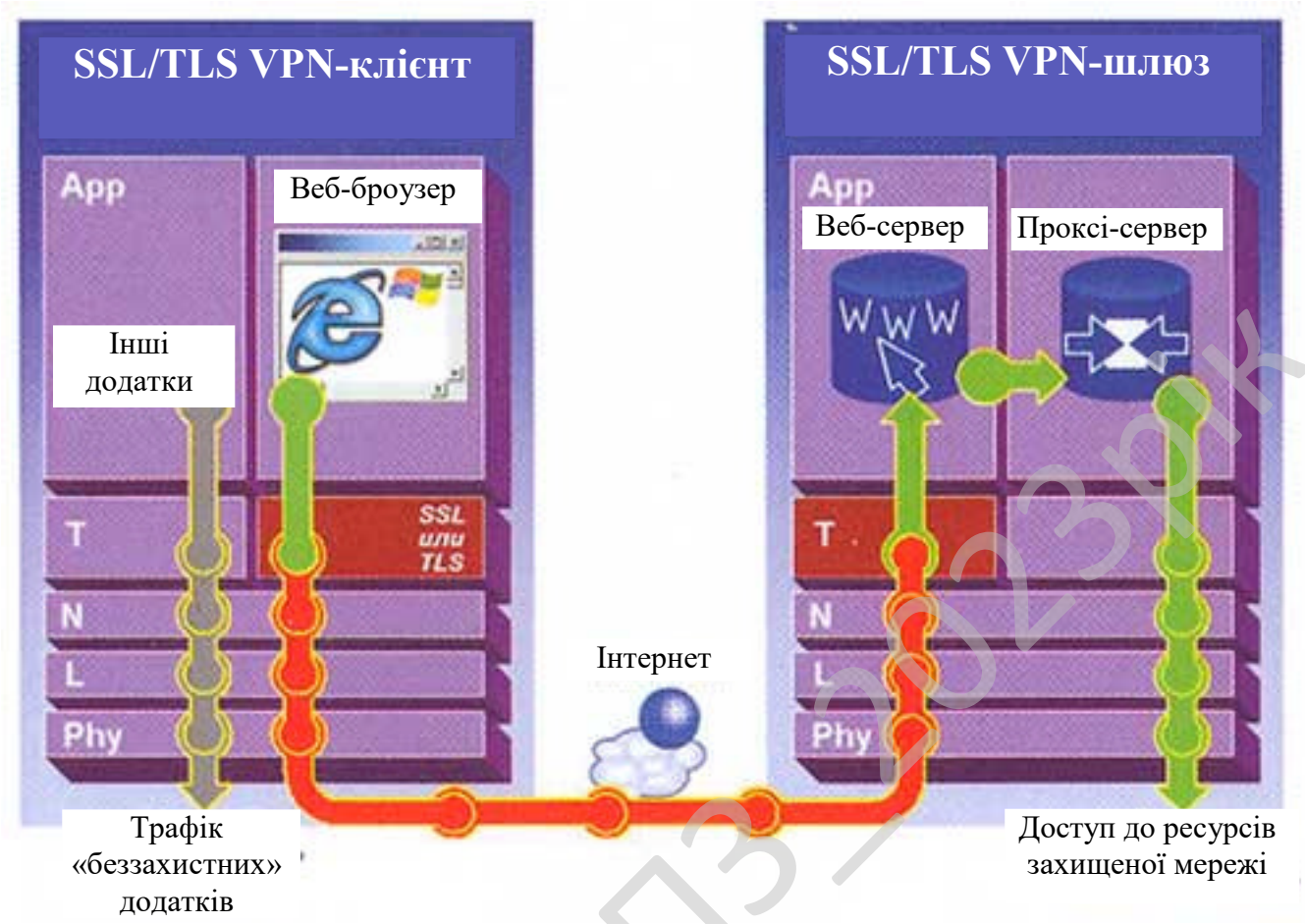

Рисунок 1.2 – Архітектура "псевдо-VPN"

Сертифікат (ключ) не обов'язковий: і SSL і TLS "уміють" виробити тимчасовий сеансовий ключ "на льоту".

Автентифікувати користувача можна пізніше, під захистом сеансового ключа, наприклад, за допомогою пароля.

Основні недоліки:

– це все-таки не VPN. По цитованому вище визначенню, тут користувачі не можуть "взаємодіяти між собою так, як якби вони перебували в єдиній закритій мережі". Вони одержують тільки функціональність доступу до вибраних ресурсів захищеної мережі. Причому, склад "вибраних ресурсів" обмежений можливостями проксі-серверів у складі VPN-шлюзу. Ті ресурси, доступ до яких "відпроксіти" не вдасться, будуть навічно недоступні.

– клієнтський парк також може бути захищений лише почасти. Додатки, у

яких клієнтська частина являє собою не Web-браузер, а щось інше, – беззахисні. І діяльність користувачів за межами подій доступу до Web-VPN-шлюзу повністю непідконтрольна.

Тепер яснішає заклопотаність аналітиків з The Burton group, що рекомендували для захисту критичних ресурсів IPsec – він не робить виключень.

Усвідомлюючи ці недоліки, індустрія SSL/TLS VPN пішла на ускладнення архітектури, пропонуючи "чесний" VPN. Найбільш яскравим прикладом такого рішення є прекрасно пророблений і описаний у декількох книгах проект OpenVPN (http://openvpn.net). Архітектуру цього (і подібних) рішень можна зрозуміти з діаграми на рисунку 1, в. Трафік додатків (усіх або частини) упаковується у відкритий транспортний протокол, потім в IP-пакети, які перехоплюються й повторно інкапсулюються до протоколу транспортного рівня із застосуванням засобів захисту SSL або TLS. Що ми одержуємо в підсумку? Повнофункціональний "чесний" VPN, дуже схожий на рішення IPsec. Однак платою за таке рішення є необхідність установки VPN-клієнта. Нікуди не дінешся – браузер не вміє перехоплювати й упаковувати до транспортного протоколу IP-пакети.

Які переваги "чесного" SSL/TLS VPN? Розроблювачі продуктів у цій технології говорять, що він "набагато легше" IPsec. Я б сказав "трохи легше" – і то з натяжкою:

– Як і в IPsec, на машину віддаленого користувача потрібно встановити VPN-клієнт.

– По розуму, треба б роздати користувачам сертифікати.

– Як і в IPsec, потрібно зконфігурувати політику безпеки.

Властиво, установка й конфігурування VPN-клієнта вважається головними труднощами впровадження IPsec у масовому парку віддалених користувачів. Де ж "полегшення"?

Деяке полегшення є в простоті протоколу й політики безпеки. Деяке полегшення є в ціні: SSL/TLS VPN-продукти бувають трохи дешевші. Але, з

іншого боку, – технічні можливості цих продуктів більш вузькі. Продукти різних виробників (у силу нестандартності інкапсуляції пакетів до транспортного протоколу) практично несумісні, чого не скажеш про IPsec-продукти.

Які технології варто вибрати замовникові, зацікавленому в побудові захищеної системи віддаленого доступу? Неявно відповідь на це питання викладена вище:

1. На порталах масового доступу -"чистий" SSL/TLS, що не є, по суті, VPN.

2. Там, де вимоги безпеки невисокі, а користувачів багато й вони малокваліфіковані, – добре використовувати "псевдо-VPN" на основі SSL або TLS.

3. Коли мова йде про строге корпоративне рішення й про доступ до критичних ресурсів – однозначно варто застосовувати тільки "чесну" VPN-архітектуру. IPsec або SSL/TLS VPN? Отут – вибір на аматора.

1.2 Область застосування

Областю застосування є віддалений доступ до мережі. У сучасному світі стрімко росте число мобільних користувачів інформаційних систем і "інформаційних надомників". Ця тенденція знайшла відбиття навіть у лінгвістиці – у побут технічної англійської мови міцно ввійшли неологізми telecommuter і teleworker.

При цьому, на сьогодні, стали масово доступними комунікаційні ресурси віддаленого доступу. Якщо 5-10 років тому віддалений користувач міг скористатися тільки модемом на лінії, що комутується (який залишається розповсюдженим і недорогим інструментом), то сьогодні практично повсюдно поширені GPRS, xDSL і швидко росте зона покриття сервісів мобільної телефонії 3G, що сполучають високу швидкість передачі даних, надійність каналу й дуже розумні ціни.

Якщо питання комунікацій на сьогодні, таким чином, вирішені, то на передній план виходять питання інформаційної безпеки віддаленого користувача.

У чому ж специфіка віддаленого доступу з погляду інформаційної безпеки? Чим задача захисту віддаленого користувача відрізняється від задачі захисту "внутрішнього" користувача корпоративної мережі?

Далекість робочого місця зовнішнього користувача породжує, у порівнянні з локальним офісним робочим місцем, три додаткових фактори погроз:

1. Мобільний користувач перебуває поза зоною прямого фізичного контролю організації. Потрібний доказ того, що до корпоративного ресурсу підключається саме "свій" користувач, а не зловмисник, "який прикинувся" своїм.

2. Дані віддаленого користувача поширюються по каналах, які перебувають поза зоною контролю організації. Ці дані піддаються перехопленню, несанкціонованій модифікації, "підмішуванню" стороннього трафіку.

3. Для робочого місця організація не може забезпечити фізичну безпеку. Його не може захистити охорона офісу, воно може бути викрадено. Воно може не відповідати вимогам організації по конфігурації.

Виникає питання – якою мірою можна нейтралізувати вплив цих негативних факторів?

Система інформаційної безпеки в IP-мережах застосуванням технологій SSL та IPsec на базі рішень Cisco Systems, розроблювана в бакалаврському проекті, й дозволяє відповісти на це питання.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Складність мережної інфраструктури й непогодженість її компонентів послабляють можливості вашої компанії в плані захисту її інформаційних ресурсів. Спроби об'єднати в одне ціле масу розрізнених точкових продуктів і пристроїв безпеки приводять до появи проломів у системі безпеки й потенційно здатні піддати небажаним ризикам навіть самі захищені середовища.

Максимально комплексний підхід до забезпечення безпеки мережі припускає створення інтегрованого рішення, що сполучить у собі наступні компоненти:

– наявні інфраструктурні пристрої з вбудованими функціями забезпечення безпеки;

– захисні пристрої, що володіють "рідним" мережним інтелектом;

– політику безпеки, що займає відповідне місце в заснованій на співробітництві, адаптивній системі безпеки.

Інтегрована система дозволяє домогтися більш істотного зниження ризиків у порівнянні з будь-яким окремим продуктом або сполученням розрізнених і ізольованих друг від друга захисних пристроїв незалежно від їхньої функціональності й продуктивності. Використання можливостей мережі в ході створення загальної архітектури безпеки дозволяє:

– спростити створювану інфраструктуру;

– забезпечити більш тісну інтеграцію її компонентів;

– скоротити сфери ризику;

– забезпечити більшу прозорість системи безпеки в масштабі всієї

інфраструктури.

Спрощення інфраструктури на основі інтегрованого підходу до будівництва мережі дозволяє створити більш зручну в роботі й керуванні систему безпеки. Крім того, така загальна архітектура безпеки формує платформу для еволюції передових мережних послуг і функціональних можливостей, одночасно забезпечуючи захист ваших інвестицій.

Цей підхід забезпечує високий рівень інтеграції, взаємодії й адаптивності:

– Інтеграція. Кожний мережний елемент виступає як точка захисту. Всі комутатори, маршрутизатори, апаратно-програмні комплекси й кінцеві пристрої реалізують певний об'єм захисної функціональності, включаючи, крім іншого, функції міжмережних екранів і віртуальних приватних мереж, а також можливості перевірки довірчих відносин і ідентифікації. Ця інтеграція охоплює ті технології, які є невід'ємним елементом безпечної роботи мережних пристроїв.

– Взаємодія. Різні мережні компоненти взаємодіють один з одним з метою створення нових засобів захисту. Процес забезпечення безпеки має на увазі координацію функцій кінцевих пристроїв, мережних елементів і застосовуваних політик. Наприклад, у рамках механізму контролю за доступом до мережі (Network Admission Control), допуск кінцевих пристроїв до мережі здійснюється за умови дотримання ними політики безпеки, реалізованої такими мережними пристроями, як маршрутизатори й комутатори.

– Адаптивність. Інноваційні поведінкові методи дозволяють автоматично розпізнавати нові види погроз у міру їхнього виникнення. Між послугами безпеки й мережним інтелектом може існувати взаємна поінформованість, що дозволяє зробити систему безпеки більш ефективною й у режимі, що попереджає, реагувати на нові погрози. Така система ефективно нейтралізує ризики в сфері безпеки завдяки наявності засобів розпізнавання погроз і організації протидії їм на різних мережних рівнях з використанням таких механізмів, як поведінкове розпізнавання, взаємна поінформованість додатків і контроль за доступом до мережі.

У цьому розділі бакалаврського проекту розглянемо рішення в області мережної безпеки на основі продуктів Cisco Systems. Для початку , перелічимо ці рішення.

Стратегія безпечної мережі Self-Defending Network:

– механізми захисту від погроз Threat Defense System;

– керовані послуги багаторівневої безпеки й захисту периметра Layered and Perimeter Security Managed Services;

– засоби ідентифікації Identity & Trust management system;

– засоби забезпечення захисту даних Secure connectivity solution;

– засоби керування розпізнаванням погроз CiscoWorks Security information management solution;

– ситуаційні центри по інформаційній безпеці (Cisco MARS).

Продукти:

– Cisco PIX Firewall;

– Cisco IOS Firewall;

– Cisco IDS/IPS (Service module, Network module);

– Cisco Guard і Traffic anomaly detector;

– Cisco Security agent;

– Content engine network module;

– Cisco Secure Access control server;

– Cisco Security Auditor (діагностика стану захищеності).

Розглянемо перераховані вище рішення в області стратегії безпечної мережі й конкретних продуктів більш докладно.

Стратегія безпечної мережі Self-Defending Network

Механізми захисту від погроз Threat Defense System

Найбільш ефективний захист бізнес-ресурсів підприємств від зловмисників і шкідливих програм досягається тільки у випадку ешелонованої оборони, розподіленої по всій мережі (включаючи і її периметр, і її внутрішні сегменти), а не зосередженої в одній точці. Стратегія Threat Defense System

інтегрує різні захисні механізми в маршрутизатори й комутатори, пропонує виділені захисні пристрої для розмежування доступу (Cisco Pix Firewall і Cisco ASA 5500 Series), відбиття атак і контролю Web-контенту (Cisco IPS 4200 і Cisco Content Engine), а також дозволяє захищати кінцеві пристрої, такі як сервера й робочі станції від широкого спектра погроз (Cisco Security Agent).

Рішення й керовані послуги для захисту від погроз, що входять до складу рішення Enterprise Class Teleworker (ECT), дозволяють здійснювати поточний моніторинг мережі, а також виявляти й класифікувати погрози. Рішення включає контролери поширення інформації про погрози й механізм правив і надає послуги по придушенню атак порушеним вузлам у мережі. Рішення дозволяє забезпечувати постійну безпеку мережі, розпізнавати внутрішні й зовнішні погрози, а також реагувати на будь-які погрози, не повідомляючи про їх користувачів.

Такі інструменти, як Система моніторингу, аналізу й реагування Cisco (CS-MARS) і Cisco Guard/Detector (або Peakflow X/SP), полегшують моніторинг, виявлення й класифікацію погроз. Контролери протоколу поширення інформації про погрози (Threat Information Distribution Protocol, TIDP) і сервіси придушення атак на базі TIDP (TMS) разом створюють механізм поширення інформації про погрози для системи безпеки на базі динамічних списків контролю доступу, системи запобігання вторгнень, а також технології оптимізованої маршрутизації в граничному сегменті.

Встановлене в клієнта устаткування з підтримкою ECT (наприклад, віддалений маршрутизатор) може бути настроєне таким чином, щоб завантажувати повідомлення syslog і запису NetFlow на CS-MARS. Контролери CS-MARS/TIDP здатні опитувати клієнтське встаткування з підтримкою ECT, щоб виявити спрацьовування сигнатур системи виявлення вторгнень і забезпечити придушення погроз за допомогою динамічного поширення цих сигнатур на інші маршрутизатори в мережі й запобігання подальших атак.

Керовані послуги багаторівневої безпеки й захисту периметра Layered and

Perimeter Security Managed Services

Керовані послуги багаторівневої безпеки й захисту периметра забезпечують підтримку інфраструктури захищених локальних і розподілених мереж для рішення на базі ECT і дозволяють створити на базі звичайних мереж інтелектуальні інформаційні мережі. Ці функції безпеки допомагають захистити встановлене в клієнтів устаткування (CPE), IP-пристрої, додатки на базі IP, а також кінцевих користувачів, постачальників і клієнтів.

Концепція багаторівневого захисту забезпечує реалізацію різних механізмів безпеки на рівні клієнтського встаткування, пристроїв і користувачів. У свою чергу, система захисту периметра забезпечує безпеку клієнтського встаткування на границі мережі й дозволяє створити архітектуру інтегрованого й захищеного рішення на базі ECT.

Засоби ідентифікації Identity & Trust management system

Перш ніж користувач, додаток або пристрій одержить доступ до необхідних ресурсів, він повинен бути опізнаний засобами захисту. Саме цю задачу на мережному рівні вирішують технології й засоби вхідні в стратегію Identity & Trust Management System – Cisco Secure Access Control Server (ACS), Cisco Secure User Registration Tool, 802.1x, Network Admission Control (NAC). Стратегія Trust and Identity Solution поширюється на всі елементи інформаційної мережі комп'ютерного підприємства – комутатори й маршрутизатори, ПК і IP-телефони, бездротові точки доступу й клієнти й т.д.

Засоби забезпечення захисту даних Secure connectivity solution

Захищена взаємодія (Secure Connectivity). Розподіленість комп'ютерної системи вимагає забезпечення захисту даних, переданих по відкритих каналах зв'язку (наприклад, через Інтернет). Збереження конфіденційності й цілісності даних є обов'язковим елементом сучасних бізнесів-додатків. Ця вимога досягається за рахунок стратегії Cisco Secure Connectivity System, що, використовуючи механізми автентифікації й приховання переданої інформації, однаково ефективно захищає дані (наприклад, від системи керування

підприємством або від Web-сервера), голос (наприклад, від IP-телефонії) і відео (наприклад, від камер відеоспостереження або в рамках відеоконференції), передані як по провідним, так і по бездротових з'єднаннях. Складовою частиною Secure Connectivity System є такі технології, як IPsec, SSL, SSH, GRE і MPLS.

Засоби керування розпізнаванням погроз CiscoWorks Security information management solution

Будучи важливою складовою Cisco Self-Defending Network, CiscoWorks Security Information Management Solution (SIMS) дозволяє виконувати інтеграцію, кореляцію й аналіз подій безпеки в мережі підприємства для поліпшення видимості мережі й зміцнення системи захисту.

Через різке збільшення числа погроз і атак на інформаційні ресурси, забезпечення безпеки стало першорядною задачею для організацій. Крім відбиття погроз у критично важливих точках мережі, підприємствам також необхідно додержуватися великого числа федеральних і промислових інструкцій, що вимагають застосування й перевірки ефективності засобів керування інформаційною безпекою. Cisco Self-Defending Network дозволяє інтегрувати компоненти захисту й SIM (Security Information Management) по всій мережі, надаючи розподілений захист від погроз. Інтегрована в мережу разом із широким набором мережних пристроїв і продуктів захисту система CiscoWorks SIMS відіграє важливу роль, дозволяючи мережним адміністраторам виконувати централізований моніторинг і аналіз, уживати відповідні дії на основі чітко певних правил обробки інцидентів безпеки й управляти системою захисту мережі підприємства.

CiscoWorks SIMS є критично важливим компонентом системи захисту від погроз Cisco Systems® – набору систем забезпечення інформаційної безпеки й інтелектуальних мережних технологій, що дозволяють виявляти й відбивати погрози безпеки організації як внутрішні, так і зовнішні. Будучи частиною системи захисту від погроз, CiscoWorks SIMS забезпечує всебічний захист інформації в мережі – від центра обробки даних до філій і кінцевих вузлів. Це

допомагає безпеки компанії створити задокументовану мережну інфраструктуру, що може бути доказом відповідності інструкціям і стандартам по забезпеченню катастрофостійкості й проведення хронологічного й судового аналізу атак, що виявляються.

CiscoWorks SIMS надає повний набір можливостей, включаючи наступні:

– Моніторинг подій у реальному масштабі часу. Набір методів кореляції подій для виявлення як відомих, так і невідомих погроз, що забезпечують скорочення числа помилкових спрацьовувань.

– Динамічна візуалізація для швидкого й наочного подання процесів виявлення, відстеження й аналізу погроз.

– Убудована можливість оцінки ризиків для подання уразливостей окремих ресурсів мережі підприємства.

– Комплексний механізм створення звітів і судового аналізу для всіх рівнів операцій системи безпеки.

– Потужна система керування інцидентами безпеки, що забезпечує організацію даних про події безпеки й що дозволяє задіяти механізм відповідної реакції на ці події.

– Моніторинг, створення звітів і керування подіями безпеки на основі керування мережного доступу Network Admission Control (NAC).

– Можливість комплексного моніторингу подій для SAFE Blueprint і систем безпеки від різних вендорів.

Ситуаційні центри по інформаційній безпеці (Cisco MARS)

Ситуаційні центри по інформаційній безпеці (Cisco MARS), що випускаються компанією Cisco Systems, дозволяють не тільки виконати задачу візуалізації рівня захищеності й сигналів тривоги від різних засобів захисту (міжмережні екрани, системи виявлення атак, засоби захисту персональних комп'ютерів і серверів), але й блокують виявлені внутрішні й зовнішні напади. Достоїнство зазначених рішень у тім, що вони можуть управляти не тільки захисними засобами, випущеними компанією Cisco Systems, але й продуктами

інших виробників (ISS, Check Point, Juniper і т.д.).

Продукти

Cisco PIX Firewall

Міжмережний екран Cisco Secure Private Internet Exchange (PIX) Firewall – дозволяє реалізувати захист корпоративних мереж на недосяжному раніше рівні, при цьому простий в експлуатації. Cisco PIX Firewall може забезпечити абсолютну безпеку внутрішньої мережі, повністю сховавши її від зовнішнього миру. На відміну від звичайних proxy-серверів, що виконують обробку кожного мережного пакета окремо з істотним завантаженням центрального процесора, Cisco PIX Firewall використовує спеціальну не UNIX-подібну операційну систему реального часу, що забезпечує більш високу продуктивність.

Основою високої продуктивності міжмережного екрана Cisco PIX Firewall є схема захисту, що базується на застосуванні алгоритму адаптивної безпеки (adaptive security algorithm – ASA), що ефективно приховує адреси користувачів від хакерів. Цей стійкий алгоритм забезпечує безпеку на рівні з'єднання на основі контролю інформації про адреси відправника й одержувача, послідовності нумерації пакетів TCP, номерах портів і додаткових прапорів TCP. Ця інформація зберігається в таблиці, перевірку на відповідність із записами якої проходять всі вхідні пакети.

Доступ через PIX дозволений тільки в тому випадку, якщо з'єднання успішно пройшло ідентифікацію. Цей метод забезпечує прозорий доступ для внутрішніх користувачів і авторизованих зовнішніх користувачів, при цьому повністю захищаючи внутрішню мережу від несанкціонованого доступу.

Завдяки застосуванню технології "наскрізного посередника" (Cut-Through Proxy) брандмауер Cisco PIX Firewall також забезпечує істотну перевагу в продуктивності в порівнянні з екранами-посередниками" на базі ОС UNIX. Як і звичайні proxy-сервери, PIX контролює встановлення з'єднання на рівні додатка.

Після успішного проходження користувачем авторизації доступу, відповідно до прийнятих правил безпеки, PIX забезпечує контроль потоку даних

між абонентами на рівні сесії. Така технологія дозволяє міжмережному екрану PIX працювати значно швидше, ніж звичайні proxy-екрани.

Крім підвищення продуктивності, застосування спеціалізованої убудованої операційної системи реального часу також забезпечує підвищення рівня безпеки. На відміну від операційних систем сімейства UNIX, вихідний текст яких широко доступний, Cisco PIX – власна розробка компанії, створена спеціально для рішення задач забезпечення безпеки. Для підвищення надійності міжмережний екран PIX Firewall передбачає можливість установки в здвоєній конфігурації в режимі "гарячого резервування", за рахунок чого в мережі виключається наявність єдиної точки можливого збою. Якщо два PIX-Екрани будуть працювати в паралельному режимі, і один з них вийде з ладу, то другий у прозорому режимі "підхопить" виконання всіх функцій забезпечення безпеки.

Міжмережний екран Cisco Secure PIX Firewall підтримує більш 500 тисяч одночасних з'єднань і, відповідно, забезпечує підтримку сотень і тисяч користувачів без зниження продуктивності. Повністю завантажений PIX Firewall може забезпечити пропускну здатність до 1,0 Гбіт/с, тобто істотно вище, ніж будь-який міжмережний екран на базі ОС UNIX або ОС Microsoft Windows NT.

Міжмережний екран Cisco Secure PIX Firewall забезпечує низьку вартість використання й супроводу. Користувачі, що не мають спеціальної підготовки можуть швидко настроїти за допомогою простої графічної оболонки PIX Device Manager (PDM), доступ до якої здійснюється за допомогою звичайного web-браузера. PDM – це додаток, що використовує http-сервер, убудований в PIX, і підтримуючий основний набір команд, необхідний для початкового настроювання міжмережного екрана. PDM дозволяє набудовувати PIX практично з будь-якого комп'ютера, для захисту пристрою від "злому" під час конфігурування користувач може використовувати протокол SSL.

Міжмережний екран Cisco Secure PIX Firewall також дозволяє уникнути проблеми недостачі адрес при розширенні й зміні IP-мереж. Технологія трансляції мережних адрес Network Address Translation (NAT) уможливлює

використання в приватній мережі як існуючих адрес, так і резервних адресних просторів. Наприклад, це дозволяє використовувати всього лише одну реальну зовнішню IP-адресу для 64 тисяч вузлів внутрішньої приватної мережі. PIX також може бути настроєний для спільного використання трансльованих і нетрансльованих адрес, дозволяючи використовувати як адресний простір приватної IP-мережі, так і зареєстровані IP-адреси.

У наш час користувачам пропонуються наступні моделі апаратно-програмних міжмережних екранів Cisco Secure PIX Firewall – PIX 501, 506E, 515E, 525 і 535.

Cisco IOS Firewall

Операційна система Cisco – IOS містить у своєму складі різні механізми захисту, включаючи засоби розмежування доступу, розширені списки міжмережного доступу, засоби організації віртуальних корпоративних мереж і т.і.

Firewall Feature Set (FFS) представляє із себе набір додаткових сервісів, які дозволяють істотно наблизити можливості операційної системи IOS до можливостей міжмережного екрана без придбання дорогого додаткового встаткування або програмного забезпечення.

Основними перевагами такого рішення є:

1. Гнучкість – одночасно вирішуються питання маршрутизації, забезпечення захищеного доступу в Інтернет. В описі правил доступу використовується інформація про користувачів, адреси, типи додатків, як для вхідних, так і вихідних з'єднань;

2. Захист інвестицій – додаткові можливості по захисту в маршрутизаторі дозволяє заощадити засоби на навчанні роботі з іншою апаратною платформою;

3. Легкість керування – використання можливостей віддаленого адміністрування дозволяє управляти системою зі свого робочого місця через мережу;

4. Сумісність із іншими продуктами й рішеннями компанії Cisco Systems.

Таблиця 2.1 – Короткий опис основних функціональних можливостей FFS

Заснований на контексті контроль доступу	
Контроль стану з'єднання	Контроль стану й контексту всіх з'єднань через роутер
Протокольно-залежна фільтрація	Облік у правилах фільтрації команд службових протоколів прикладного рівня, виявлення атак на рівні протоколів, динамічне відкриття необхідних для роботи додатків портів
Підтримувані додатки	
TCP/UDP додаток	Telnet, FTP, HTTP, SNMP
FTP протокол	Активний і пасивний режим
Мультимедіа додаток	Контроль потоків службової інформації для правильного відкриття необхідних портів для аудіо/відео з'єднань (включає H.323 додатки, CU-SeeME, RealAudio, VDOLive, Streamworks і ін.)
Контроль SMTP додатків	Виявлення невірних SMTP команд, що дозволяє уникнути необхідності установки поштових серверів у демілітаризованій зоні
Підтримка RCP сервісів	Контроль запитів portmapper на відкриття з'єднань для роботи RCP додатків
Підтримка R-команд	Перевірка відповідей сервера із запитами на встановлення додаткових з'єднань
Підтримка додатків Oracle	Контроль повідомлень про перенапрямок з'єднань від серверної частини Oracle у додатках клієнт-сервер; відкриття портів для роботи клієнтів із сервером
Додатки по підтримці відеоконференцій на основі H.323	Перевірка контрольних повідомлень Q.931 і H.245, які служать для відкриття додаткових UDP з'єднань для передачі відео й аудіо даних

Продовження таблиці 2.1

Виявлення й запобігання атак типу "відмова в обслуговуванні"	
Захист від популярних видів атак	Захист від syn атак, сканування портів, захист від атак з вичерпуваням ресурсів маршрутизатора.
Контроль порядкових номерів	Перевірка порядкових номерів (sequence number) в TCP з'єднаннях для гарантії того, що вони перебувають в очікуваному діапазоні
Статистика з'єднань	Статистика з'єднань, що включає час, адреси джерела/призначення, порти й повне число переданих байт
Настроювання, що рекомендуються, за замовчуванням	Настроювання при завантаженні, вкл/викл source routing, дозв/заборон proxy arp, дозвіл усіх/тільки необхідних додатків, шифрування паролів на роутері за допомогою MD5, списки доступу й паролі до віртуальних терміналів, дозв/заборон автентифікації роутерів у підтримуємих протоколах маршрутизації
Розширена статистика по TCP/UDP з'єднаннях	Статистика доступу користувачів (порти й адреси джерела/призначення)
Блокування Java апплетів	
Установка рівня захисту	Можна настроїти правила фільтрації або повної заборони Java апплетів, що не перебувають усередині архівів або стислих файлів
Оповіщення про атаки в реальному часі	
Розширені можливості	Повідомлення у випадку атак типу "відмова в обслуговуванні" або інших заздалегідь описаних подій через syslog механізм на заданий хост
Сумісність із іншими можливостями Cisco IOS	Сумісність зі списками доступу, трансляцією адрес, рефлексивними списками доступу, технологією шифрування, використовуваною в Cisco
Мережне керування	
Підтримка в ConfigMaker	Програма під Windows95/NT, що полегшує настроювання мережних параметрів, адресації й параметрів FFS

Cisco IDS/IPS (Service module, Network module)

Cisco IDS/IPS є центральним компонентом рішень Cisco Systems по відбиттю атак. Поряд із традиційними механізмами в Cisco IDS/IPS використовуються й унікальні алгоритми, що відслідковують аномалії в мережному трафіку й відхилення від нормального поводження мережних додатків. Це дозволяє виявляти як відомі, так і багато невідомих атак.

Вбудовані технології кореляції подій безпеки Cisco Threat Response, Threat Risk Rating і Meta Event Generator не тільки допомагають істотно знизити число помилкових спрацьовувань, але й дозволяють адміністраторам реагувати лише на дійсно критичні атаки, які можуть завдати серйозної шкоди ресурсам корпоративної мережі.

Основні можливості:

– Широкий спектр алгоритмів виявлення атак (сигнатури, аномалії, евристика, відхилення від RFC і т.п.).

– Захист від методів обходу.

– Можливість роботи одночасно у двох режимах – виявлення й запобігання атак.

– Виявлення атак на IP-телефонію й АСОВІ ТП (SCADA).

– Автоматичний вибір реагування залежно від ступеня погрози.

– Продуктивність – 8 Гбіт/сек у кластері.

– Інтеграція з Cisco Incident Control System.

– Виявлення атак в інкапсульованому трафіку MPLS, GRE, IPv6, Mobile IP-in-IP.

– Інтеграція з Cisco PIX/Cisco ASA 5500 і Cisco Secuirty Agent для блокування атак.

– Підтримка декількох віртуальних сенсорів на одному пристрої.

– Інтеграція з комутаторами й маршрутизаторами для блокування атак шляхом зміни ACL або обмеження швидкості передачі трафіку (Rate Limiting).

– Можливість розподілу навантаження між декількома сенсорами й

забезпечення відказостійкості.

– Вибіркове блокування (не усіх IP-адрес, а тільки атакуючого сервісу).

– Підтримка до 255 VLAN на один інтерфейс сенсора.

– Можливість ефективного запобігання атак у мережах, що комутуються.

– Імпорт даних від сканерів безпеки.

– Механізм OS Fingerprint для визначення релевантності атаки.

– Оцінка ефективності реагування на атаку.

Cisco Guard і Traffic anomaly detector

Модуль виявлення, що вбудовується, атак Cisco Guard і Traffic Anomaly Detector. Основні можливості:

– Унікальна архітектура Multiverification Process (MVP).

– Відсутність зниження продуктивності мережі, що захищається.

– Швидкість обробки трафіку – 1,25 млн пакетів у секунду (можливість масштабування до 10 млн пакетів у секунду шляхом використання кластерів Cisco Guard).

– Число паралельно оброблюваних з'єднань – 1,5 млн.

– Можливість поставки у вигляді виділеного пристрою або модуля для комутатора Cisco Catalyst 6500 або маршрутизатора Cisco 7600.

– Захист від одночасної атаки з боку понад 100 000 зомбі (механізм Zombie Killer).

– Число динамічних фільтрів – 150 000 (додавання 1000 фільтрів у секунду).

– Затримка – менш 1 мсек.

– Централізоване керування й інтеграція з CiscoWorks SIMS.

– Дотримання необхідного рівня SLA.

– Забезпечення послуг аутсорсингу.

Cisco Security agent

Cisco Security Agent (CSA) поєднує різні захисні механізми й функції в одному рішенні:

– запобігання атак;

– персональний міжмережний екран;

– захист від шкідливого коду;

– контроль цілісності;

– блокування витоку інформації через USB -порти й інші зовнішні пристрої;

– обмеження можливостей Інтернет-пейджерів (наприклад, ICQ);

– виявлення перехоплювачів із клавіатури.

Основні можливості:

– Інтеграція з Active Directory, LDAP, NIS.

– Прозорість установки, не потребуючої участі власника комп'ютера.

– Автоматизація створення політик контролю.

– Керування 100.000 агентами з однієї консолі керування.

– Інтеграція з Network Admission Control.

– Функціонування на платформах Windows, Linux, Solaris.

– Блокування витоку конфіденційної інформації через USB, CD/ DVD-RW, зовнішні пристрої й КПК.

– Контроль цілісності файлів і програм.

– Захист від Spyware і rootkits.

– Захист від скачування забороненого контенту (наприклад, MP3 файлів).

– Інвентаризація встановленого програмного забезпечення й збір статистики про роботу додатків.

– Заборона мережної активності хосту або додатків при роботі з типом даних “конфіденційна інформація”.

– Заборона функцій деяких додатків (копіювання в буфер, місце

збереження й т.д.) при роботі з типом даних “конфіденційна інформація

Cisco Secure Access control server

Cisco Secure ACS (ACS) – це сервер керування доступом з високою масштабованістю й високою продуктивністю, що діє як централізований сервер RADIUS і TACACS+. Cisco Secure ACS розширює ресурси забезпечення безпеки доступу, поєднуючи функції автентифікації, користувальницького й адміністративного доступу з політиками включаючи рішення централізованої мережної ідентифікації, що дає більшу гнучкість і мобільність, збільшення рівня безпеки й підвищення результативності роботи користувачів. Це призначає однорідну політику безпеки для всіх, незалежно від того, яким чином вони одержують доступ у мережу. При цьому скорочуються витрати на адміністрування й керування, зв'язані регулюванням мережного доступу користувачів і мережних адміністраторів. Використовуючи центральну базу даних всіх облікових записів користувачів, Cisco Secure ACS централізовано управляє повноваженнями користувачів і розподіляє їх по сотнях або навіть тисячах точок доступу в мережі. Діючи як сервіс аудіта, Cisco Secure ACS надає можливість одержання детальних звітів і моніторингу поводження користувачів мережі, і зберігає записи про кожне одержання доступу й зміни конфігурації пристроїв у масштабах всієї мережі. Ця функціональна можливість придбала особливу важливість для компаній у світлі дотримання положень закону Сарбейнса-Окслі. Cisco Secure ACS підтримує широкі можливості одержання доступу, у тому числі провідні та бездротові LAN, що комутуються, широкополосні, контент, зберігання, телефонію на базі IP (VoIP), міжмережні екрани й VPN.

Cisco Secure ACS – важливий елемент архітектури Cisco Identity-Based Networking Services (IBNS). Cisco IBNS засновані на стандартах забезпечення безпеки портів, зокрема, 802.1x (це стандарт IEEE для керування мережним доступом на базі портів) і на протоколі EAP (Extensible Authentication Protocol), і розширюючи захищену автентифікацію, авторизацію й аудіта (AAA) від

периметра мережі до кожної точки усередині мережі. Нові елементи керування політик (такі як, квотування по користувачах, призначення VLAN, списки контролю доступу) можуть бути застосовані в новій архітектурі, побудованої на розширених функціональних можливостях комутаторів Cisco і бездротових точок доступу що опитують Cisco Secure ACS по протоколі RADIUS.

Крім цього, Cisco Secure ACS є важливим елементом рішення Cisco NAC (Network Admission Control) по керуванню доступом у мережу. NAC Cisco – це галузева ініціатива під патронажем Cisco Systems, у якій використовується мережна інфраструктура для контролю над дотриманням політики безпеки на всіх пристроях, що намагається одержати доступ до ресурсів мережі, таким чином, обмежується збиток, заподіюваний вірусами й хробаками. Використовуючи NAC, клієнти можуть одержати доступ до мережі тільки із припустимих і довірених пристроїв (наприклад, ПК, сервера й кишенькові міні-комп'ютери), і доступ буде обмежений для тих пристроїв, які не відповідають вимогам. Cisco NAC – складова частина ініціативи Cisco по мережах, що самозахищаються (Cisco Self-Defending Network) і фундамент впровадження керування доступом у мережі на 2-ому і 3-ьому рівні мережної моделі. Майбутні фази розширять взаємодію кінцевих точок і ресурсів безпеки мережі, включивши динамічні можливості обмеження поширення інцидентів. Ця інновація дасть довіреним елементам системи можливість повідомляти про нехарактерне поводження джерела, яким можуть виступати несправні або інфіковані системи під час атаки. Таким чином, заражені системи можуть бути динамічно ізольовані від іншої частини мережі, що значно зменшить масштаби поширення вірусів, хробаків і погроз змішаного типу.

Cisco Secure ACS – потужний сервер керування доступом із широким спектром високопродуктивних масштабованих функцій для будь-яких компаній зі зростаючими ресурсами з'єднання по WAN або LAN.

Діагностика стану захищеності (Security Auditor)

Для підвищення рівня захисту як середовища передачі даних, так і

корпоративної мережі комп'ютерного підприємства необхідне використання засобів діагностування стану захищеності й прогнозування його зміни. Для рішення цієї задачі компанія Cisco Systems пропонує як уже згаданий вище ситуаційний центр по інформаційній безпеці, так і спеціальне рішення (Security Auditor), призначене для дистанційної перевірки встаткування й засобів захисту й вироблення рекомендацій з підвищення рівня захищеності.

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

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) 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

Опис Secure Socket Layer

Протокол SSL (Secure Socket Layer – протокол захищених сокетів) був розроблений фірмою Netscape, як протокол забезпечуючий захист даних між сервісними протоколами (такими як HTTP, NNTP, FTP і т.д.) і транспортними протоколами (TCP/IP). Часто для нього використовується абревіатура HTTPS. Саме ця латинська буква "s" перетворює звичайний, не захищений канал передачі даних в Інтернеті по протоколу HTTP, у засекречений або захищений,

Протокол SSL надає "безпечний канал", що має три основні властивості:

– Канал є частним. Шифрування використовується для всіх повідомлень після простого діалогу, що служить для визначення секретного ключа.

– Канал автентифікований. Серверна сторона діалогу завжди автентифікується, у той час як клієнтська – автентифікується опційно.

– Канал надійний. Транспортування повідомлень містить у собі перевірку цілісності (із залученням MAC).

SSL не тільки забезпечує захист даних в Інтернеті, але так само робить упізнання сервера й клієнта (server/client authentication). У цей момент протокол SSL прийнятий W3 консорціумом (W3 Consortium) на розгляд, як основний захисний протокол для клієнтів і серверів (WWW browsers and servers) у мережі Інтернет.

Використання SSL

Найчастіше, цей протокол використовується в складі будь-якого Інтернет-ресурсу, що здійснює маніпуляції з особистими або фінансовими даними його користувачів. Найчастіше, це банки, Інтернет-магазини або будь-які інші віртуальні місця, у яких, приходячі по своїх справах, користувачі,

змушені передавати свої особисті, і найчастіше, секретні дані. Цього може зажадати й проста реєстрація, і процедура оплати якого-небудь товару, або будь-яка інша процедура, при якій користувачі змушені чесно видавати свої паспортні дані, PIN-и й паролі. З'являються два досить вагомих доводи, перший, передану інформацію треба шифрувати, і другий, ми повинні бути впевнені, що передаємо інформацію саме туди, куди потрібно. Саме для рішення цих двох питань і використовується SSL.

HTTP і HTTPS

Спроби розробити універсальний мережний протокол, здатний забезпечити належний рівень безпеки при роботі в Інтернет, вживали досить давно, і досить великою кількістю різних фірм і організацій. HTTP протокол пропонував досить простій, парольний спосіб ідентифікації того або іншого користувача. У момент з'єднання із сервером, користувач уводив пароль, пароль передавався серверу у відкритому, не зашифрованому виді, і далі, перевіривши відповідність пароля й ім'я користувача, сервер відкривав або не відкривав викликане з'єднання.

Далі, у міру розвитку Інтернету, було створено кілька різних безпечних протоколів. Офіційний протокол, розробку якого спонсувала IETF, називався Secure HTTP (SHTTP), Крім нього, розроблялися, і були створені, ще декілька неофіційних проектів, один із яких, за назвою SSL (Secure Sockets Layer), створений Netscape, одержав більшу популярність і широке поширення. Правда, не дивлячись на свою популярність, SSL не є офіційним Інтернет стандартом.

SSL у дії

Головним призначенням SSL-протоколу, є забезпечення приватного й надійного способу обміну інформацією між двома віддалено взаємодіючими додатками. Протокол реалізується у вигляді двошарового (багатошарового) середовища, спеціально призначеного для безпечного переносу секретної інформації, через не засекречені канали зв'язку. Як перший шар, у такому середовищі використовується деякий надійний транспортний протокол; TCP

наприклад. По слову "транспортний", не важко догадатися, що TCP бере на себе функції "несучої", і надалі, стає візником, для всіх лежачих вище шарів (протоколів). Другим по рахунку шаром, що накладається на TCP, є SSL Record Protocol. Разом, ці два шари, TCP і SSL Record Protocol, формують своєрідне ядро SSL. Надалі, це ядро стає первинною герметуючою оболонкою, для всіх наступних більш складних протокольних інфраструктур. У якості однієї з таких структур, використовується SSL Handshake Protocol (Протокол «рукостискання»)– дозволяючий серверу й клієнтові ідентифікувати один одного й погоджувати криптографічні алгоритми й ключі, перед тим як додатки, що працюють на серверній і клієнтській стороні, зможуть почати передачу або прийом інформаційних байтів у захищеному режимі.

Одним з не мало важливих переваг SSL, є його повна програмно-платформна незалежність. Протокол розроблений на принципах переносимості, і ідеологія його побудови, не залежить, від тих додатків, у складі яких він використовується. Крім цього, важливо й те, що поверх протоколу SSL, можуть прозоро накладатися й інші протоколи; або для ще більшого збільшення ступеня захисту цільових інформаційних потоків, або, для адаптації криптографічних здатностей SSL під яку-небудь іншу, цілком певну задачу.

Ви починаєте використовувати SSL у той момент, коли вводите в адресному рядку свого браузера URL, який починається з абревіатури HTTPS, У результаті, ви підключаєтеся до порту за номером 443, що для SSL звичайно використовується за замовчуванням; для стандартного HTTP з'єднання, найчастіше використовується порт 80. У процесі підключення, браузер користувача (надалі клієнт), посилає серверу привітальне повідомлення (hello message). У свою чергу сервер, також повинен посилати клієнтові своє привітальне повідомлення. Привітальні повідомлення, є первинними, ініціалізуючими повідомленнями й містять інформацію, використовувану при подальшому настроюванні секретного каналу, що відкривається. У загальному випадку, привітальне повідомлення встановлює чотири основних параметри:

версія протоколу, ідентифікатор сесії, спосіб шифрування, метод компресії, а також, два спеціально згенерованих випадкові числа; і сервер, і клієнт, генерують такі числа незалежно один від одного, а потім, просто обмінюються ними один з одним.

Після одержання привітального повідомлення від клієнта, сервер відсилає свій сертифікат, якщо такий у нього є. Також, при необхідності, сервер може послати і якесь ключове повідомлення, наприклад у випадку відсутності сертифіката. Якщо сервер авторизований (тобто має відповідний сертифікат), він може зажадати й клієнтський сертифікат, якщо того зажадає обраний спосіб шифрування даних. Після цього, виробляється ще ряд проміжних обмінних операцій, у процесі яких, виробляється остаточне уточнення обраного алгоритму шифрування, ключів і секретів, і далі, сервер посилає клієнтові якесь фінальне повідомлення, після чого обидві сторони приступають до обміну зашифрованої інформації.

Недоліки протоколу

SSL як такий, теоретично, може забезпечити практично повний захист будь-якого Інтернет з'єднання. Але для успішного функціонування SSL, крім нього самого, необхідні також і чисто програмні засоби, що перетворюють технологію SSL у життя. Програми, що так чи інакше використовують SSL протокол, як не дивно, є часом самим уразливим місцем цієї технології. Саме через помилки в цих програмах, можлива майже повна втрата, всіх, досягнутих після використання SSL щитів і заслонів. До таких програмних інструментів, насамперед, відносяться активно використовувані нами Інтернет-браузери.

Одним із самих показових критеріїв рівня захисту, є розмір використовуваних ключів. Чим більше цей розмір, тим відповідно надійніше захист. Браузери в основному використовують три розміри: 40, 56 і 128 біт, відповідно. Причому, 40-а бітний варіант ключа недостатньо надійний. Таким чином, переважніше використовувати саме 128-мі бітні ключі. Стосовно до Internet Explorer від Microsoft, це означає завантаження додаткового пакета

(security pack). Так як інтернаціональні версії цього браузера, завжди забезпечуються винятково 40-а або 56-і бітним захистом, а 128-мі бітний захист, ставиться тільки на північноамериканські версії цього браузера (США, Канада). Для того щоб установити, який саме розмір ключа використовується у вашому браузері, в Netscape Navigator вам досить відкрити підменю "Options/Security Preferences", а в Internet Explorer, підміню "Help/About".

Але розмір ключа, не буде грати вирішальної ролі, якщо в захисті браузера є внутрішній пролом. Повідомлення про відкриття таких проломів, у тих або інших браузерах, з'являються з регулярними інтервалами. Такий пролом нагадує відкриту кватирку в кімнаті, що протоплюється, – все тепло миттєво вивітрюється. Із цього приводу, доречно згадати випадок, який відбувся з Netscape Navigator, у травні 2000 року. Тоді, один корейський студент виявив таку, досить не приємну особливість, цього браузера. При спробі з'єднання із сервером, що володіє не придатним сертифікатом, з подальшою відмовою від продовження такого з'єднання, відбувалося наступне. Netscape, помилково, поміщав цей сертифікат у список придатних, і відповідно, при наступне підключення вже не видавав користувачеві ні яких попереджуючих повідомлень, спокійно підключаючись до цього, не цілком надійного, серверу

Але всі ці і їм подібні діри, не йдуть ні в яке порівняння з тією погрозою, що можуть представляти для користувача вчасно не відкликані сертифікати. Справа в тому, що браузери звичайно поставляються з якимось, цілком певним набором дійсних сертифікатів, але автоматичного механізму перевірки цієї придатності по плину деякого часу – не існує.

Таким чином, можливо, що дія, тим або іншим, використовуваним вашим браузером сертифіката, уже, давно скінчилося; міг минути строк придатності, міг бути загублений контроль над особистим ключем відповідному цьому сертифікату й т.ін. У кожному із цих випадків, сертифікат автоматично озивається, і міститься в спеціальний, так званий revocation list, або список не придатних сертифікатів, створюваний і обновлюваний тим або іншим

сертифікаційному співтовариству (CA). Тепер, якщо не видалити такий сертифікат з вашого браузера, він по колишньому буде значитися як придатний, з усіма наслідками, що випливають звідси.

Варто помітити, що ідея, закладена в протоколі SSL безумовно, гарна. Хоча в неї є й свої плюси, і свої мінуси, але в цілому, цей протокол можна назвати одним з найбільш удалих рішень проблеми захисту користувальницьких даних при їхньому поширенні "відкритим" каналом. Цей протокол цілком би міг стати якоюсь мережною панацеєю. Але, на жаль, практика, показує що ідея це ще не рішення. Без відповідної практичної складової, ідея так і залишається ідеєю, а тому, користувачі безумовно, повинні пам'ятати, що символ замка, що з'являється в рядку стану їхніх Інтернет-браузерів, ще не гарантія того, що всі наші секрети й таємниці перебувають під дійсно надійним захистом

Реалізації SSL

Тепер кілька слів про реалізацію SSL. Найпоширенішим пакетом програм для підтримки SSL є SSLeay. Остання версія (SSLeay v. 0.8.0) підтримує SSLv3. Ця версія доступна у вихідних текстах. Цей пакет призначений для створення й керування різного роду сертифікатами. Так само в його склад входить і бібліотека для підтримки SSL різними програмами. Ця бібліотека необхідна, наприклад, для модуля SSL у розповсюдженому HTTP сервері Apache. Якщо Ви встановлюєте версію, поза США, то особливих проблем з алгоритмом RSA бути не повинно. Але тільки накладається обмеження на довжину ключа в 40 біт. Діє це обмеження й на інший пакет від фірми Netscape – SSLRef. А от якщо комп'ютер з SSLeay перебуває на території США, то за використання алгоритму RSA необхідно заплатити. Але про це потрібно розмовляти із самою фірмою RSA Data Security.

Опис IPsec

В 1994 році Рада по архітектурі Інтернет (IAB – Internet Activities Board – Координаційна рада мережі Internet, технічний орган, відповідальний за розвиток набору протоколів Internet; включає технічні групи IRTF і IETF, кожна

з яких займається рішенням своїх задач) випустив звіт "Безпека архітектури Інтернет". У цьому документі описувалися основні області застосування додаткових засобів безпеки в мережі Інтернет, а саме захист від несанкціонованого моніторингу, підміни пакетів і керування потоками даних. У числі першочергових і найбільш важливих захисних мір вказувалася необхідність розробки концепції й основних механізмів забезпечення цілісності й конфіденційності потоків даних. Оскільки зміна базових протоколів сімейства TCP/IP викликала б повну перебудову мережі Інтернет, була поставлена задача забезпечення безпеки інформаційного обміну у відкритих телекомунікаційних мережах на базі існуючих протоколів. Таким чином, почала створюватися специфікація Secure IP – Захищений IP (протокол захисту мережного трафіку на IP-рівні), додаткова стосовно протоколів IPv4 і IPv6.

Архітектура IPsec

IP Security – це комплект протоколів, що стосуються питань шифрування, автентифікації й забезпечення захисту при транспортуванні IP-пакетів; у його склад входять близько 20 пропозицій по стандартах і 18 RFC.

Специфікація IP Security (відома сьогодні як IPsec) розробляється IETF (Internet Engineering Task Force – проблемна група проектування Internet) У цей час IPsec включає 3 алгоритмо-незалежні базові специфікації, опубліковані в якості RFC-документів "Архітектура безпеки IP", "Автентифікуючий заголовок (AH)", "Інкапсуляція зашифрованих даних (ESP)". Необхідно помітити, що в березні 1998 року Робоча група IP Security Protocol запропонувала нові версії цих специфікацій, що мають у цей час статус попередніх стандартів. Крім цього, існують декілька алгоритмо-залежних специфікацій, що використовують протоколи MD5, SHA, DES.

Робоча група IP Security Protocol розробляє також і протоколи керування ключовою інформацією. У задачу цієї групи входить розробка протоколу IKMP (Internet Key Management Protocol), протоколу керування ключами прикладного рівня, що не залежить від використовуваних протоколів забезпечення безпеки. У

цей час розглядаються концепції керування ключами з використанням специфікації ISAKMP (Internet Security Association and Key Management Protocol) і протоколу Oakley установлення ключа (Oakley Key Determination Protocol). Специфікація ISAKMP описує механізми узгодження атрибутів використовуваних протоколів, у той час як протокол Oakley дозволяє встановлювати сесійні ключі на комп'ютери мережі Інтернет. Розглядаються також можливості використання механізмів керування ключами протоколу SKIP (SKIP команда підготовки наступної команди) Створювані стандарти керування ключовою інформацією, можливо, будуть підтримувати Центри розподілу ключів, аналогічні використовуваним у системі Kerberos.

Гарантії цілісності й конфіденційності даних у специфікації IPsec забезпечуються за рахунок використання механізмів автентифікації й шифрування відповідно. Останні, у свою чергу, засновані на попередньому узгодженні сторонами інформаційного обміну т.зв. "контексту безпеки" – застосовуваних криптографічних алгоритмів, алгоритмів керування ключовою інформацією і їхніми параметрами.

Специфікація IPsec передбачає можливість підтримки сторонами інформаційного обміну різних протоколів і параметрів автентифікації й шифрування пакетів даних, а також різних схем розподілу ключів. При цьому результатом узгодження контексту безпеки є встановлення індексу параметрів безпеки (SPI), що представляє собою покажчик на певний елемент внутрішньої структури сторони інформаційного обміну, що описує можливі набори параметрів безпеки.

По суті, IPsec, працює на третьому рівні, тобто на мережному рівні. У результаті передані IP-пакети захищені прозорим для мережних додатків і інфраструктури чином. На відміну від SSL (Secure Socket Layer), що працює на четвертому (тобто транспортному) рівні й тісніше пов'язаний з більш високими рівнями моделі OSI, IPsec покликаний забезпечити низькорівневий захист.

1. Фізичний Власне кабель або фізичний носій
2. Канальний Передача й прийом пакетів, визначення апаратних адрес
3. Мережний Маршрутизація й ведення обліку
4. Транспортний Забезпечення коректного наскрізного пересилання даних
5. Сеансовий Автентифікація й перевірка повноважень
6. Подання даних Інтерпретація й стиск даних
7. Прикладний Надання послуг на рівні кінцевого користувача: пошта, реєстрація й т.д.

Рисунок 3.1 – Модель OSI/ISO

До IP-даних, готових до передачі по віртуальній приватній мережі, IPsec додає заголовок для ідентифікації захищених пакетів. Перед передачею по Internet ці пакети інкапсулюються в інші IP-пакети. IPsec підтримує кілька типів шифрування, у тому числі DES (DES – Data Encription Standard – стандарт шифрування даних, шифрувальний стандарт (широко використовуваний високонадійний алгоритм для шифрування й дешифрування даних, розроблений U.S. National Bureau of Standards)) і MD5 – дайджест повідомлення.

Щоб установити захищене з'єднання, обоє учасника сеансу повинні мати можливість швидко погодити параметри захисту, такі як алгоритми автентифікації й ключі. IPsec підтримує два типи схем керування ключами, за допомогою яких учасники можуть погодити параметри сеансу. Ця подвійна підтримка викликала певні тертя в робочій групі IETF.

З поточною версією IP, IPv4, можуть бути використані або Internet Secure Association Key Management Protocol (ISAKMP), або Simple Key Management for Internet Protocol. З новою версією IP, IPv6, прийдеться використовувати ISAKMP, відомий зараз як ISAKMP/Oakley. На жаль, дві вищеописані схеми керування ключами багато в чому несумісні, а виходить, гарантувати ітероперабельніть від краю до краю навряд чи можливо.

SA – Security Associations

Security Association (SA) – це з'єднання, що надає служби забезпечення безпеки трафіку, що передається через нього. Два комп'ютери на кожній стороні SA зберігають режим, протокол, алгоритми й ключі, використовувані в SA. Кожний SA використовується тільки в одному напрямку. Для двунаправленого зв'язку потрібно два SA. Кожний SA реалізує один режим і протокол; таким чином, якщо для одного пакета необхідно використовувати два протоколи (як наприклад AH і ESP), то потрібно два SA.

Політика безпеки

Політика безпеки зберігається в SPD (Security Policy Database – База даних політики безпеки). SPD може вказати для пакета даних одну із трьох дій: відкинути пакет, не обробляти пакет за допомогою IPsec, обробити пакет за допомогою IPsec. В останньому випадку SPD також указує, який SA необхідно використовувати (якщо, звичайно підходить SA, що уже був створений) або вказує, з якими параметрами повинен бути створений новий SA.

SPD є дуже гнучким механізмом керування, що допускає дуже гарне керування обробкою кожного пакета. Пакети класифікуються по великій кількості полів, і SPD може перевіряти деякі або всі поля для того, щоб визначити відповідну дію. Це може привести до того, що весь трафік між двома машинами буде передаватися за допомогою одного SA, або окремі SA будуть використовуватися для кожного додатка, або навіть для кожного TCP з'єднання.

ISAKMP/Oakley

Протокол ISAKMP визначає загальну структуру протоколів, які

використовуються для встановлення SA і для виконання інших функцій керування ключами. ISAKMP підтримує кілька Областей Інтерпретації (DOI), однією з яких є IPsec-DOI. ISAKMP не визначає закінчений протокол, а надає "будівельні блоки" для різних DOI і протоколів обміну ключами.

Протокол Oakley – це протокол визначення ключа, що використовує алгоритм заміни ключа Діффі-Хеллмана.

Протокол Oakley підтримує ідеальну пряму безпеку (Perfect Forward Secrecy – PFS), при реалізації якої дозволяється доступ до даних, захищених тільки одним ключем, коли сумнів викликає єдиний ключ. При цьому для обчислення значень додаткових ключів цей ключ захисту повторно ніколи не використовується; крім того, для цього не використовується й вихідний матеріал, що послужив для обчислення даного ключа захисту.

IKE

IKE – протокол обміну ключами за замовчуванням для ISAKMP, на даний момент є єдиним.

IKE перебуває на вершині ISAKMP і виконує, властиво, установлення як ISAKMP SA, так і IPsec SA. IKE підтримує набір різних примітивних функцій для використання в протоколах. Серед них можна виділити хеш-функцію й псевдовипадкову функцію (PRF).

Атаки на AH, ESP і IKE.

Всі види атак на компоненти IPsec можна розділити на наступні групи:

– атаки, що експлуатують кінцівку ресурсів системи (типовий приклад – атака "Відмова в обслуговуванні", Denial-of-service або DoS-атака);

– атаки, що використовують особливості й помилки конкретної реалізації IPsec;

– атаки, засновані на слабостях самих протоколів AH і ESP.

Чисто криптографічні атаки можна не розглядати – обидва протоколи визначають поняття "трансформ", куди приховують всю криптографію. Якщо використовуваний криптоалгоритм стійкий, а певний з ним трансформ не вносить

додаткових слабостей (це не завжди так, тому вірніше розглядати стійкість всієї системи: Протокол – Трансформ – алгоритм), то із цієї сторони все нормально.

Replay Attack – відтворити атаку – нівелюється за рахунок використання Порядкового номера (в одному єдиному випадку це не працює – при використанні ESP без автентифікації й без AH). Далі, порядок виконання дій (спочатку шифрація, потім автентифікація) гарантує швидке відбраковування "поганих" пакетів (більш того, відповідно до останніх досліджень у світі криптографії, саме такий порядок дій найбільш безпечний, зворотний порядок у деяких, правда дуже окремих випадках, може привести до потенційних дір у безпеці; на щастя, ні SSL, ні IKE, ні інші розповсюджені протоколи з порядком дій "спочатку автентифікувати, потім зашифрувати", до цих окремих випадків не відносяться, і, стало бути, цих дір не мають).

Залишається атака Denial-Of-Service (відмова в обслуговуванні). Як відомо, це атака, від якої не існує повного захисту. Проте, швидке відбраковування поганих пакетів і відсутність якої-небудь зовнішньої реакції на них (згідно RFC) дозволяють більш-менш добре справлятися із цією атакою. У принципі, більшості (якщо не всім) відомим мережним атакам (sniffing, spoofing, hijacking і т.п.) AH і ESP при правильному їхньому застосуванні успішно протистоять. З IKE трохи складніше. Протокол дуже складний, важкий для аналізу. Крім того, у силу помилок (у формулі обчислення HASH_R) при його написанні й не зовсім удалих рішень (той же HASH_R і HASH_I) він містить кілька потенційних "дир" (зокрема, у першій фазі не всі Payload у повідомленні автентифікуються), втім, вони не дуже серйозні й ведуть, максимум, до відмови у встановленні з'єднання. Від атак типу replay, spoofing, sniffing, hijacking IKE більш-менш успішно захищається. Із криптографією трохи складніше, – вона не винесена, як в AH і ESP, окремо, а реалізована в самому протоколі. Проте, при використанні стійких алгоритмів і примітивів (PRF), проблем бути не повинно. У якомусь ступені можна розглядати як слабість IPsec те, що в якості єдиного обов'язкового до реалізації криптоалгоритму в нинішніх специфікаціях

вказується DES (це справедливо й для ESP, і для IKE), 56 біт ключа якого вже не вважаються достатніми. Проте, це чисто формальна слабість – самі специфікації є алгоритмо-незалежними, і практично всі відомі вендори давно реалізували 3DES (а деякі вже й AES).Таким чином, при правильній реалізації, найбільш "небезпечною" атакою залишається відмова в обслуговуванні.

Оцінка протоколу

Протокол IPsec одержав неоднозначну оцінку з боку фахівців. З одного боку, відзначається, що протокол IPsec є кращим серед всіх інших протоколів захисту переданих по мережі даних, розроблених раніше (включаючи розроблений Microsoft PPTP (PPTP мережна технологія, що підтримує багатопротокольні віртуальні частні мережі (VPN), дозволяючи віддаленим користувачам безпечно звертатися до корпоративних мереж за допомогою з'єднання, що комутирується, надаваним постачальником послуг Internet (ISP) або за допомогою прямого з'єднання до Internet)). З іншого боку, є присутнім надмірна складність і надмірність протоколу.

На думку аналітиків, протокол є занадто складним, щоб бути безпечним. Зокрема, Niels Ferguson і Bruce Schneier у своїй роботі "A Cryptographic Evaluation of IPsec – Криптографічне обчислення IPsec" відзначають, що вони виявили серйозні проблеми безпеки практично у всіх головних компонентах IPsec.

Автори також відзначають, що набір протоколів вимагає серйозної доробки для того, щоб він забезпечував гарний рівень безпеки. Вони також приводять опис ряду атак, що використовують як слабості загальної схеми обробки даних, так і слабості криптографічних алгоритмів.

У цьому розділі бакалаврського проекту ми розглянули деякі основні моменти, що стосуються протоколу мережної безпеки IPsec. Не зайвим буде відзначити, що протокол IPsec реалізований в операційній системі Windows2000 та вище компанії Microsoft. Приведемо таблицю, у якій відбувається порівняння IPsec і широко розповсюдженого зараз SSL.

Таблиця 3.1 – Класифікація типових віддалених атак на мережі

Особливості	IPsec	SSL
Апаратна незалежність	Так	Так
Код	Не потрібно змін для додатків. Може зажадати доступ до вихідного коду стека TCP/IP.	Потрібні зміни в додатках. Можуть знадобитися нові DLL або доступ до вихідного коду додатків.
Захист	IP пакет цілком. Включає захист для протоколів вищих рівнів.	Тільки рівень додатків.
Фільтрація пакетів	Заснована на автентифікованих заголовках, адресах відправника й одержувача, і т.п. Проста й дешева. Підходить для роутерів.	Заснована на вмісті й семантиці високого рівня. Більш інтелектуальна й більш складна.
Продуктивність	Менше число перемикань контексту й переміщення даних.	Більше число перемикань контексту й переміщення даних. Більші блоки даних можуть прискорити криптографічні операції й забезпечити кращий стиск.
Платформи	Будь-які системи, включаючи роутери	В основному, кінцеві системи (клієнти/сервери), також firewalls.
Firewall/VPN	Весь трафік захищений.	Захищений тільки трафік рівня додатків. ICMP, RSVP, QoS і т.п. можуть бути незахищені.
Прозорість	Для користувачів і додатків.	Тільки для користувачів.
Поточний статус	Стандарт, що з'являється.	Широко використовується WWW браузерами, також використовується деякими іншими продуктами.

3.2 Розробка структурної схеми

На рисунку 3.2 наведена структурна схема розроблювальної системи інформаційної безпеки в IP-мережах застосуванням технологій SSL та IPsec на базі рішень Cisco Systems.

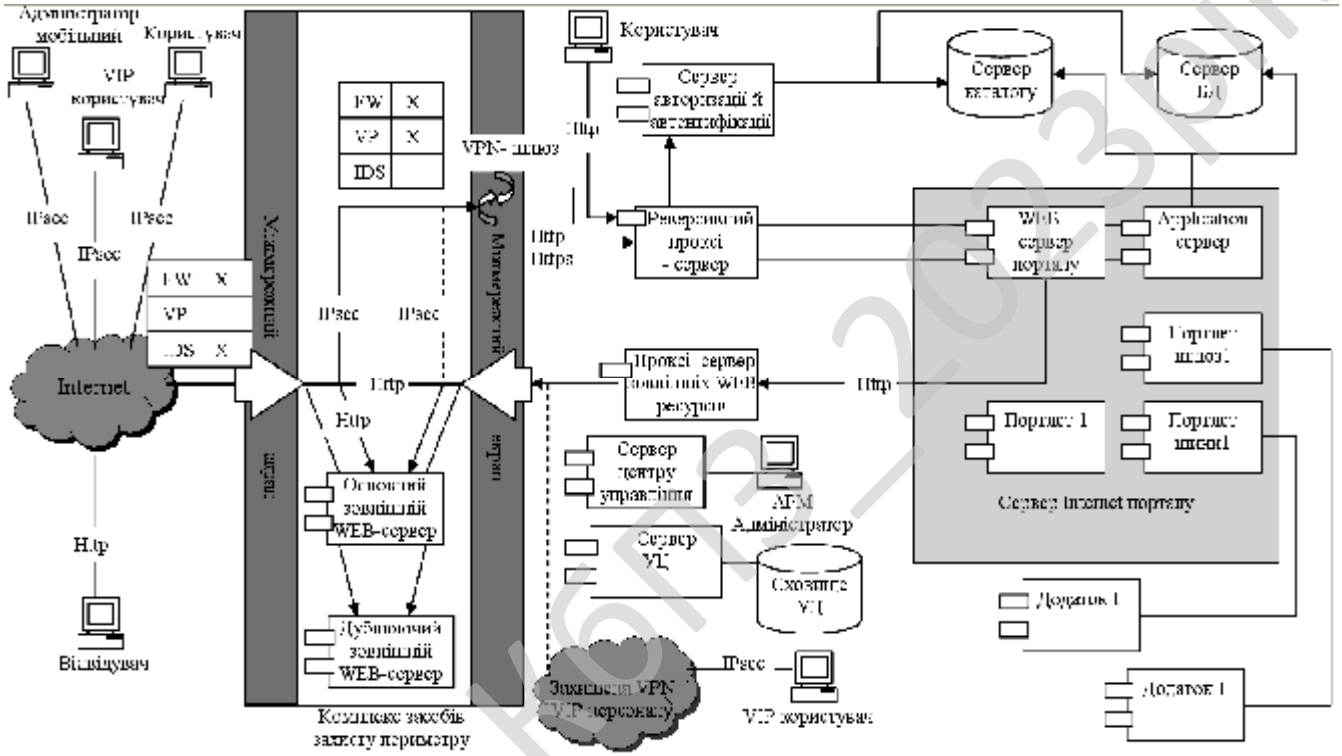

Рисунок 3.2 – Структурна схема системи

На схемі використовуються наступні скороченням та визначення.

УЦ – удостовірюючий центр сертифікатів.

Портлет – змінний компонент користувальницького інтерфейсу будь-якого Веб-порталу, що підключається. Портлет видає фрагменти розмітки, які вбудовуються в сторінку порталу. Найчастіше, сторінка порталу представляється у вигляді набору портлетних вікон, що не перекривають одне одного, кожне з яких відображає портлет. Таким чином, портлет (або сукупність портлетів) представляється у вигляді єдиного веб-додатку, розміщеного на порталі.

Приклади портлетів: e-mail, повідомлення про погоду, останні новини. Завдяки існуючим стандартам портлетів розроблювачі можуть створювати портлети, що вбудовуються в будь-який портал, що відповідає цим стандартам.

VPN – (Virtual Private Network – віртуальна приватна мережа) – логічна мережа, створювана поверх іншої мережі, наприклад Інтернет. Незважаючи на те, що комунікації здійснюються по публічних мережах з використанням небезпечних протоколів, за рахунок шифрування створюються закриті від стороніх канали обміну інформацією. VPN дозволяє об'єднати, наприклад, кілька офісів організації в єдину мережу з використанням для зв'язку між ними непідконтрольних каналів.

IDS – є програмними або апаратними системами, які автоматизують процес перегляду подій, що виникають у комп'ютерній системі або мережі, і аналізують їх з погляду безпеки. Так як кількість мережних атак зростає, IDS стають необхідним доповненням інфраструктури безпеки. Ми розглянемо, для яких цілей призначені IDS, як вибрати й зконфігурувати IDS для конкретних систем і мережних оточень, як обробляти результати роботи IDS і як інтегрувати IDS з іншою інфраструктурою безпеки підприємства. Виявлення проникнення є процесом моніторингу подій, що відбуваються в комп'ютерній системі або мережі, і аналізу їх. Проникнення визначаються як спроби компрометації конфіденційності, цілісності, доступності або обходу механізмів безпеки комп'ютера або мережі. Проникнення можуть здійснюватися як атакуючими, що одержують доступ до систем з Інтернету, так і авторизованими користувачами систем, що намагаються одержати додаткові привілеї, яких у них немає. IDS є програмними або апаратними пристроями, які автоматизують процес моніторингу й аналізу подій, що відбуваються в мережі або системі, з метою виявлення проникнень. IDS складаються із трьох функціональних компонентів: інформаційних джерел, аналізу й відповіді. Система одержує інформацію про подію з одного або декількох джерел інформації, виконує обумовлений конфігурацією аналіз дані події й потім створює спеціальні відповіді – від

найпростіших звітів до активного втручання при визначенні проникнень.

FW – персональний файрвол (персональний брандмауер) – додаток, що виконує роль міжмережного екрана для окремого комп'ютера (звичайно персонального), запущений на цьому ж самому комп'ютері.

Більша частина функцій персонального файрволу дублює функції міжмережного екрану, однак персональний файрвол так само може забезпечувати додаткові можливості:

– Контроль за додатками, що використовують порти. На відміну від звичайних міжмережних екранів, персональний файрвол може визначати не тільки використовуваний протокол і адреси, але й точну назву додатка, що запитує з'єднання (або намагається прослуховувати якийсь порт), зокрема, можливий контроль за незмінністю додатка (у випадку зміни додатка вірусами або троянами, що встановлюються в якості плагінів, мережна активність додатка блокується).

– Призначення роздільних правил різним користувачам без додаткової мережної авторизації.

– Режим навчання, коли при першому зверненні програми до мережних ресурсів користувачеві видається запит (звичайно виду «заборонити завжди, заборонити однократно, завжди дозволити, дозволити однократно, створити правило»).

– Режим змішаної фільтрації (при якій перевіряються різні параметри на різних рівнях мережних протоколів – від другого (перевірка на фальсифікацію MAC-адреси) до 4 (фільтрація портів), і навіть вищестоящих рівнів (фільтрація вмісту веб-сайтів, перевірка пошти, відсівання спама).

При цьому персональний файрвол звичайно не призначений для використання в якості міжмережного екрана й не може здійснювати фільтрацію маршрутизованих і/або трансльованих пакетів, фільтрувати пакети на основі адреси відправника, використовувати різні правила для різних мережних інтерфейсів (звичайно в рамках моделі персонального файрволу вважається, що в

комп'ютера єдина IP-адреса і єдиний зовнішній мережний інтерфейс).

Варто підкреслити, що розроблена, у результаті виконання бакалаврського проектування, система контролює доступ автентифікованних (своїх) користувачів і забороняє доступ сторонніх у захищену зону, що лежить праворуч. Система керування доступом побудована на базі двох проксі-серверів, один із яких (реверсивний) розмежовує доступ зареєстрованих користувачів порталу до його ресурсів, а другий (вихідний) – доступ таких же користувачів у публічну мережу Інтернет. Реверсивний проксі – сервер забезпечує автентифікацію користувачів на основі сертифікатів X.509 і PKI, а також механізм однократної реєстрації. Проксі – сервери управляються сервером автентифікації й авторизації на основі ролей користувачів, зв'язаних з корпоративною організаційною структурою, і матриці доступу, що реалізує механізм рольового доступу до ресурсів порталу. Для відвідувачів порталу (неавторизованих суб'єктів) доступний тільки Web-сервер, розташований у неохоронюваній зоні.

Сервер керування безпекою формує загальну політику безпеки у вигляді сукупності правил мережного доступу до сегментів, хостів, портів, сервісів, і правил проксі-доступу до сервісів, портлетів-додатків і статичних інформаційних ресурсів порталу. Дана політика транслюється в локальні політики FW/VPN-агентів і проксі систем, а потім доставляється й виконується на них.

Відсутність вихідного проксі-сервера в архітектурі порталу пояснюється тим, що в системі не ставилася задача захисту доступу до ресурсів середовища Інтернет, оскільки такий доступ із внутрішньої мережі заборонений. Змінилися й технології захисту мережного рівня – якщо раніше для формування VPN використовувався протокол SKIP (Simple Key management for Internet Protocol), то в цей час застосовується протокол IP Security (IPsec). За минулі роки помітно вдосконалилися засоби виявлення вторгнень, міжмережного екранування, визначення уразливостей і антивірусного захисту. На зміну захищеним смарт-картам прийшли більш зручні захищені USB-токени (ключі на основі USB флеш карт).

Підводячи підсумки, сформулюємо три основних переваги запропонованої архітектури системи захисту інформації.

– Рішення може бути застосоване для широкого кола портальних систем і платформ.

– Не потрібна переробка коду портальних додатків.

– Типові протоколи мережного рівня (IP/IPsec) і прикладного рівня (HTTP/HTTPS) поліпшують інтегрованість засобів захисту й реалізуємість функцій безпеки.

Основне тут те, що дане рішення претендує на роль типового для різноманітних портальних систем, і що воно при інтеграції з існуючими порталами може бути убудоване без переробки коду портальних додатків.

3.3 Розробка функціональної схеми

На рисунку 3.3 наведена функціональна схема системи. У системі використовуються наступні протоколи.

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.

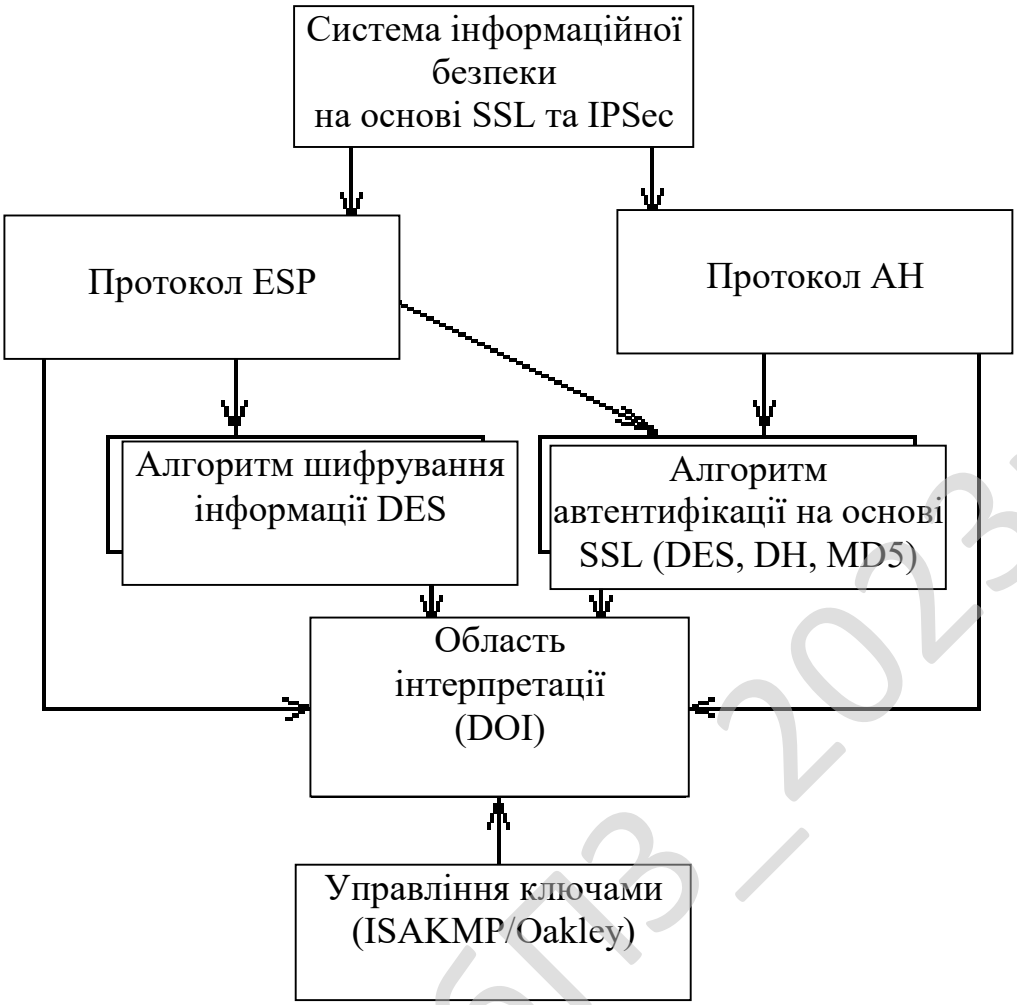

Рисунок 3.3 – Функціональна схема системи

Наступний заголовок	Довжина навантаження	Зарезервовано
Індекс параметрів безпеки (SPI)		
Поле послідовного номеру		
Дані аутентифікації (перемінної довжини)		

Рисунок 3.4 – Формат заголовка AH

Послідовний номер пакета був введений в AH в 1997 році в ході процесу перегляду специфікації IPsec. Значення цього поля формується відправником і служить для захисту від атак, пов'язаних з повторним використанням даних процесу автентифікації. Оскільки мережа Інтернет не гарантує порядок доставки пакетів, одержувач повинен зберігати інформацію про максимальний послідовний номер пакета, що пройшов успішну автентифікацію, і про одержання деякого числа пакетів, що містять попередні послідовні номери (звичайно це число дорівнює 64).

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.

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 шифрування. 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 шифрування. 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 шифрування для декодування інформації транспортного рівня.

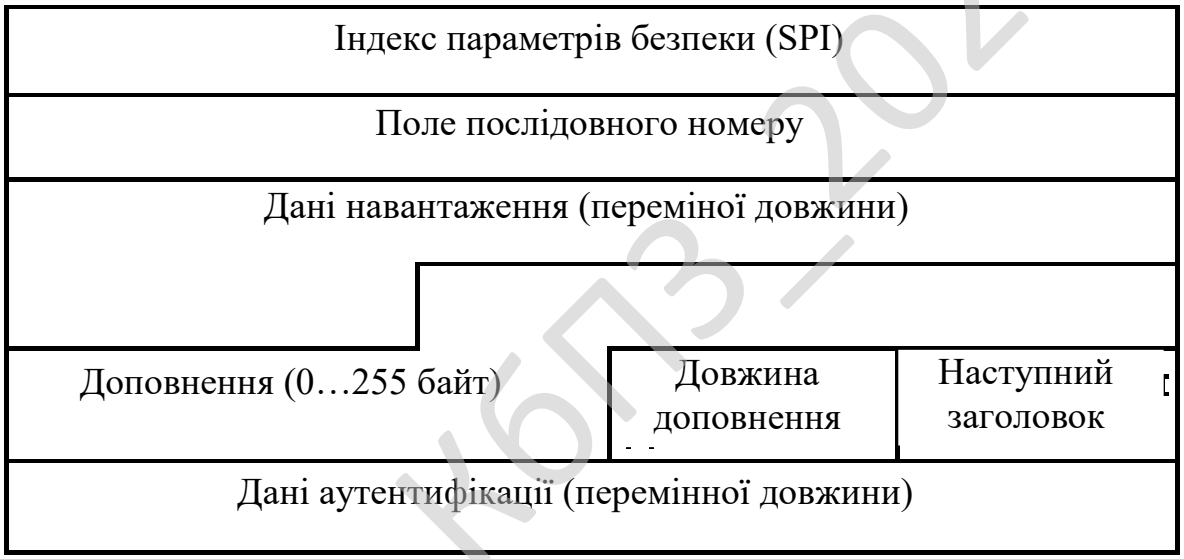

Рисунок 3.5 – Формат заголовка ESP

Розрізняють два режими застосування ESP – транспортний і тонельний.

Транспортний режим – використовується для шифрування поля даних IP пакета, що містить протоколи транспортного рівня (TCP, UDP, ICMP), який, у свою чергу, містить інформацію прикладних служб. Прикладом застосування транспортного режиму є передача електронної пошти. Всі проміжні вузли на маршруті пакета від відправника до одержувача використовують тільки відкриту інформацію мережного рівня й, можливо, деякі опціональні заголовки пакета (в IPv6). Недоліком транспортного режиму є відсутність механізмів приховання конкретних відправника й одержувача пакета, а також можливість проведення

аналізу трафіку. Результатом такого аналізу може стати інформація про об'єми й напрямки передачі інформації, області інтересів абонентів, розташування керівників.

Тонельний режим – припускає шифрування всього пакета, включаючи заголовок мережного рівня. Тонельний режим застосовується якщо буде потреба приховання інформаційного обміну організації із зовнішнім миром. При цьому, адресні поля заголовка мережного рівня пакета, що використовує тонельний режим, заповнюються міжмережним екраном організації й не містять інформації про конкретного відправника пакета. При передачі інформації із зовнішнього миру в локальну мережу організації як адреса призначення використовується мережна адреса міжмережного екрана. Після дешифрування міжмережним екраном початкового заголовка мережного рівня пакет направляється одержувачеві.

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 SSL (Secure Sockets Layer) версії 4.0 замість механізму обміну ключами SSL 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 шифрування, 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 шифрування 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 шифрування 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. Сполучення різних IPsec-механізмів забезпечує цілком безпечні з'єднання як між мережами, так і між кінцевими станціями. Оскільки практично всі постачальники підтримують ці стандарти, рано або пізно це приведе до виникнення середовища для реалізації безпечних з'єднань через Інтернет. Таким чином, протокол IPsec стане основним для безпечної е-комерції в Інтернет.

3.4 Розробка діаграми процесів

На рисунку 3.4 зображена діаграма процесів розробленої системи.

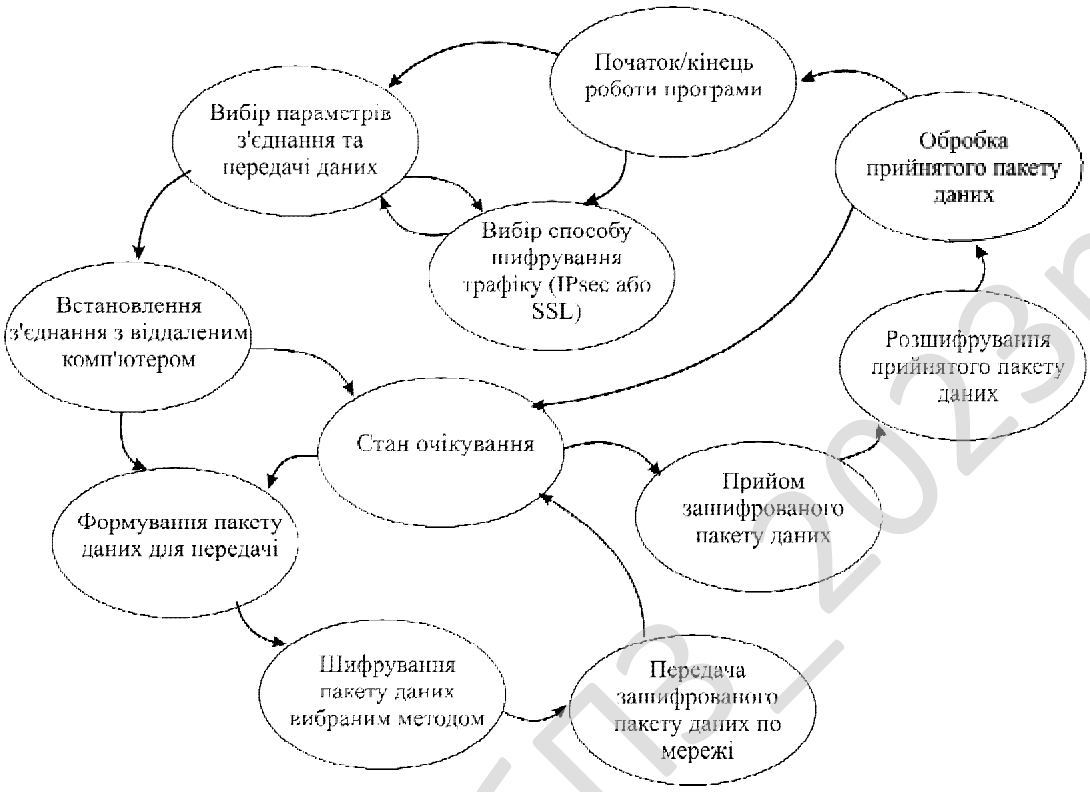

Рисунок 3.4 – Діаграма процесів розробленої системи

Як видно з рисунку, після запуску програми користувач повинен вибрати параметри з'єднання та передачі даних, а також спосіб шифрування трафіку (IPsec або SSL). Потім запускається процес встановлення з'єднання з віддаленим комп'ютером. Після встановлення з'єднання, або користувач запускає процес формування пакету даних для передачі, або система переходить у стан очікування. Після процесу формування пакету даних, відбувається його шифрування та передача по мережі і перехід у стан очікування. Якщо виникає процес прийому зашифрованого пакету даних, то спочатку відбувається його розшифрування, а потім обробка.

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 РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Блок-схема роботи основної програми зображена на рисунку 4.1. З рисунку видно, що спочатку після запуску програми відкривається головне вікно. Потім користувач вводить параметри з'єднання з сервером та намагається здійснити з'єднання з мережею. Якщо спроба вдала і з'яднання відбулося, то користувач або безпосередньо починає роботу у мережі без захисту трафіку, або вмикає шифрування. Для ввімкнення шифрування необхідно ввести пароль та вибрати спосіб шифрування IPsec або SSL. Після цього увесь трафік перед передачею по мережі шифрується вибраним методом. У будь який момент користувач може переглянути стан та параметри з'єднання за допомогою відповідної кнопки. Залежно від того, як адміністратор сконфігурує сервер, захист трафіку для всієї одноадресної передачі даних може бути або запитуваний, але не обов'язковий, або необхідний**.** При використанні цієї моделі, для відповідей на запити безпеки від серверів, клієнтам досить застосування політики за замовчуванням. Після того, як будуть виконані зіставлення безпеки і встановлене з'єднання між сервером і клієнтом, це з'єднання буде залишатися активним ще протягом однієї години після того, як між ними був переданий останній пакет даних. Після закінчення цієї години клієнтом анулюється зіставлення безпеки, і він вертається в початковий стан «тільки відповідь». Якщо згодом клієнт знову відправить пакети відкритим текстом на той же сервер, то сервер відновить захист підключення. Цей спосіб забезпечує безпеку, якщо початкові пакети, що відправляються клієнтом на сервер, не містять конфіденційних даних і якщо політикою сервера дозволений прийом незахищених пакетів, що відправляються клієнтами відкритим текстом.

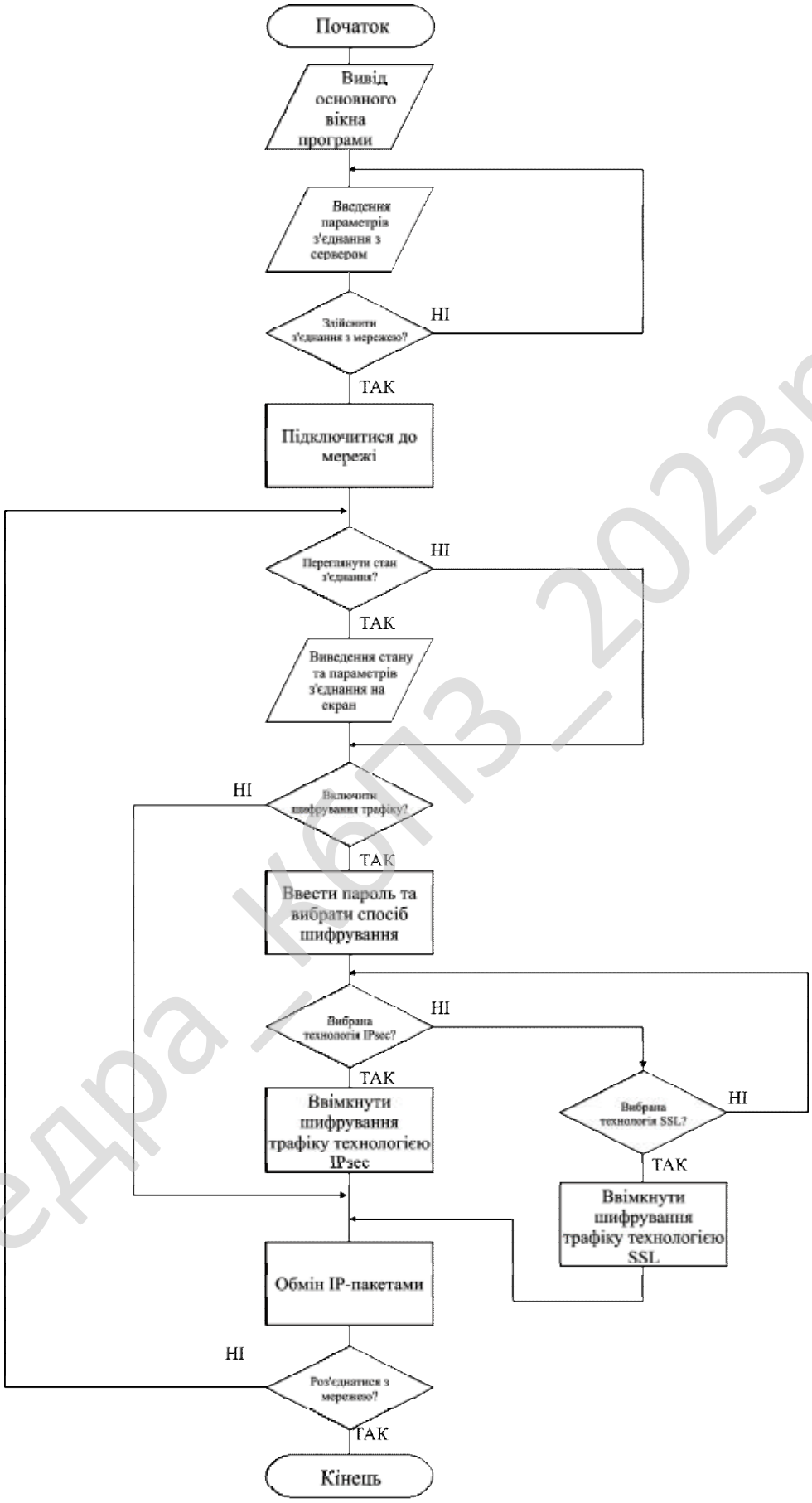

Рисунок 4.1 – Блок-схема алгоритму роботи основної програми

Блок-схема алгоритму шифрування даних за допомогою IPsec зображена на рисунку 4.2.

IPsec опирається на ряд технологічних рішень і методів шифрування, але дію IPsec у загальному можна представити у вигляді наступних головних кроків:

Крок 1. Початок процесу IPsec. Трафік, якому потрібне шифрування відповідно до політики захисту IPsec, погодженої сторонами IPsec, починає IKE-процес.

Крок 2. Перша фаза IKE. IKE-процес виконує автентифікацію сторін IPsec і веде переговори про параметри асоціацій захисту IKE, у результаті чого створюється захищений канал для ведення переговорів про параметри асоціацій захисту IPsec у ході другої фази IKE.

Крок 3. Друга фаза IKE. IKE-процес веде переговори про параметри асоціації захисту IPsec і встановлює відповідні асоціації захисту IPsec для пристроїв сторін, що обмінюються інформацією.

Крок 4. Передача даних. Відбувається обмін даними між сторонами IPsec, що ґрунтується на параметрах IPsec і ключах, збережених у базі даних асоціацій захисту.

Крок 5. Завершення роботи тунелю IPsec. Асоціації захисту IPsec завершують свою роботу або в результаті їхнього видалення, або через перевищення граничного часу їхнього існування.

Розглянемо зазначені кроки докладніше.

Крок 1. Початок процесу IPsec

Тип трафіку, що повинен захищатися засобами IPsec, визначається в рамках політики захисту для VPN. Потім ця політика реалізується у вигляді команд конфігурації інтерфейсів пристроїв кожної сторони IPsec. Наприклад, у маршрутизаторах Cisco і брандмауерах PIX Firewall для визначення трафіку, що підлягає шифруванню, використовують списки доступу. Списки доступу реалізують політику шифрування, наприклад, за допомогою операторів permit, що вказують, що відповідний трафік повинен шифруватися, та операторів deny, що

забороняють шифрування відповідного трафіку. У випадку клієнта Cisco VPN використовуються вікна меню, де вказуються з'єднання, яким повинна забезпечуватися захист IPsec. Коли підлягаючий шифруванню трафік генерується клієнтом IPsec або проходить через нього, клієнт ініціює наступний крок процесу, починаючи першу фазу IKE.

Крок 2. Перша фаза IKE

Головною метою обміну даними, що відбуваються в першій фазі IKE, є автентифікація сторін IPsec і створення захищеного каналу між сторонами, що дозволяє почати обмін IKE. У ході першої фази IKE виконуються наступні дії.

– Ведуться переговори про узгодження політики асоціацій захисту IKE між сторонами, щоб забезпечити захист обміну IKE. Асоціація захисту IKE одержує погоджені параметри IKE і є двосторонньою.

– Виконується обмін Діффі-Хеллмана, у результаті якого вибирається загальний секретний ключ для використання в алгоритмах шифрування IPsec.

– Виконується автентифікація й забезпечується захист сторін IPsec.

– Встановлюється захищений тунель для ведення переговорів про параметри другої фази IKE.

Для першої фази IKE припустимі два режими: основний і енергійний.

Основний режим першої фази IKE (Main Mode)

У цьому режимі виконуються три двосторонніх обміни між ініціатором і респондентом:

1. У ході першого обміну алгоритми, використовувані для захисту зв'язку IKE, погоджуються доти, поки не буде досягнута відповідність для всіх асоціацій захисту IKE сполучених сторін.

2. У процесі другого обміну виконується алгоритм Діффі-Хеллмана, щоб погодити загальний секретний матеріал, на основі якого створюються спільні секретні ключі, передати так звані "оказії" (випадкові значення, що посилаються іншій стороні), підписати їх і повернути назад, щоб довести "свою особистість".

3. У ході третього обміну виконується автентифікація сторони-партнера.

Ідентифікаційним значенням у цьому випадку виступає IP-адреса сторони IPsec у шифрованому вигляді.

Основним результатом цього режиму є узгодження параметрів асоціацій захисту IKE між сторонами з метою створення захищеного каналу для наступних обмінів IKE.

Асоціація захисту IKE визначає параметри обміну IKE: використовуваний метод автентифікації, алгоритми шифрування й хешування, використовувана група Діффі-Хеллмана (одна із двох доступних), максимальний час існування асоціації захисту IKE у секундах або кілобайтах і спільно використовувані секретні значення ключів для алгоритмів шифрування. Асоціації захисту IKE у пристроях кожної зі сторін є двосторонніми.

Енергійний режим першої фази IKE (Aggressive mode)

У даному режимі менше й число обмінів, і число пакетів, що пересилаються при цьому, у результаті чого потрібно менше часу для установки сеансу IPsec. У цьому випадку виконуються наступні дії:

1. Під час першого обміну майже все необхідне включається в пропоновані значення для асоціацій захисту IKE, відкритий ключ Діффі-Хеллмана, оказія, що підписується другою стороною, і пакет ідентифікації, який можна використовувати для того, щоб автентифікувати другу сторону за допомогою третьої сторони.

2. Одержувач відправляє назад усе, що потрібно, щоб завершити обмін. Ініціаторові залишається тільки підтвердити обмін.

Недоліком використання енергійного режиму є те, що обидві сторони обмінюються інформацією до того, як створений захищений канал. Таким чином, можна підключитися до лінії й з'ясувати, хто формує нову асоціацію захисту. З іншого боку, обмін відбувається швидше, ніж в основному режимі. Енергійний режим для обміну IKE звичайно не ініціюється продуктами Cisco, але маршрутизатори Cisco і брандмауери PIX Firewall можуть відповідним чином відповісти стороні IPsec, що ініціювала обмін в енергійному режимі.

Крок 3. Друга фаза IKE (Quick Mode)

Завданням другої фази IKE є узгодження параметрів асоціації захисту IPsec з метою створення тунелю IPsec. У цій фазі виконуються наступні дії.

– Ведуться переговори про параметри асоціації захисту IPsec, що захищаються існуючою асоціацією захисту IKE.

– Встановлюються асоціації захисту IPsec.

– Періодично відновлюються переговори про асоціації захисту IPsec, щоб гарантувати захист.

– У необов'язковому порядку може виконуватися додатковий обмін Діффі-Хеллмана.

Друга фаза IKE виконується тільки у швидкому режимі, після того як у результаті першої фази IKE створюється захищений тунель. Потім ведуться переговори про погоджену політику IPsec, витягає загальний секретний матеріал для роботи алгоритмами захисту IPsec і створюються асоціації захисту IPsec. У швидкому режимі виконується обмін оказіями, які забезпечують захист від відтворення повідомлень. Оказії використовуються для того, щоб гарантувати створення нових секретних ключів і не допустити проведення атак відтворення, у результаті яких супротивник міг би створити "фальшиві" асоціації захисту.

Швидкий режим використовується також для того, щоб домовитися про нові асоціації захисту IPsec, коли виявляється перевищеною межа часу існування старої асоціації захисту IPsec. Базовий варіант швидкого режиму використовується для того, щоб обновити секретний матеріал, призначений для створення спільного секретного ключа на основі значень, отриманих при обміні Діффі-Хеллмана в ході першої фази.

В IPsec є опція PFS (Perfect Forward Secrecy – досконала пряма таємність), що підсилює захист ключів. Якщо політикою IPsec запропоноване використання опції PFS, то для кожного обміну у швидкому режимі потрібен новий обмін Діффі-Хеллмана, що забезпечує нові дані для ключів, у результаті чого дані для ключів будуть мати більшу ентропію ("нерегулярність") і тому більшу стійкість

відносно криптографічних атак. Кожний обмін Діффі-Хеллмана вимагає великої кількості піднесень у ступінь, що збільшує завантаження процесора й знижує загальну продуктивність системи. Асоціації захисту, узгоджувані у швидкому режимі, ідентифікуються IP-адресами IKE-сторін.

Узгодження перетворень IPsec

У ході другої фази в рамках протоколу IKE ведуться переговори про перетворення IPsec (алгоритмах захисту IPsec). IPsec складається із двох головних протоколів захисту й безлічі протоколів підтримки.

Перетворення IPsec і пов'язані з ними алгоритми шифрування є наступними.

– Протокол AH (Authentication Header – заголовок автентифікації). Протокол захисту, що забезпечує автентифікацію й (як опція) сервіс виявлення відтворення. Протокол AH діє як цифровий підпис і гарантує, що дані в пакеті IP не будуть несанкціонованно змінені. Протокол AH не забезпечує сервіс шифрування та дешифрування даних. Даний протокол може використовуватися або самостійно, або разом із протоколом ESP.

– Протокол ESP (Encapsulating Security Payload – включаючий захист корисний вантаж). Протокол захисту, що забезпечує конфіденційність і захист даних, а також (як опція) сервіс автентифікації й виявлення відтворення. Підтримуючі IPsec продукти Cisco використовують ESP для шифрування корисного вантажу IP-пакетів. Протокол ESP може використовуватися самостійно або разом з AH.

– Стандарт DES (Data Encription Standard – стандарт шифрування даних). Алгоритм шифрування й дешифрування даних пакетів. Алгоритм DES використовується як у рамках IPsec, так і IKE. Для алгоритму DES використовується 56-бітовий ключ, що означає не тільки більш високе споживання обчислювальних ресурсів, але й більш надійне шифрування. Алгоритм DES є симетричним алгоритмом шифрування, для якого потрібні ідентичні секретні ключі шифрування в пристроях кожної зі сполучених сторін

IPsec. Для створення симетричних ключів застосовується алгоритм Діффі-Хеллмана. IKE і IPsec використовують алгоритм DES для шифрування повідомлень.

– "Потрійний" DES (3DES). Варіант DES, заснований на використанні трьох ітерацій стандартного DES із трьома різними ключами, що практично потроює стійкість DES. Алгоритм 3DES використовується в рамках IPsec для шифрування й дешифрування потоку даних. Даний алгоритм використовує 168-бітовий ключ, що гарантує високу надійність шифрування. IKE і IPsec використовують алгоритм 3DES для шифрування повідомлень.

При перетворенні IPsec використовується також два стандартних алгоритми хешування, що забезпечують автентифікацію даних.

– Алгоритм MD5 (Message Digest 5). Алгоритм хешування, застосовуваний для автентифікації пакетів даних. У продуктах Cisco використовується обчислюється з допомогою MD5 код HMAC (Hashed Message Authentication Code – хеш-код автентичності повідомлення) – варіант коду автентичності повідомлення, якому забезпечується додатковий захист за допомогою хешування. Хешування являє собою процес однобічного (тобто необоротного) шифрування, у результаті якого для поступаючого на вхід повідомлення довільної довжини виходить код фіксованої довжини. IKE, AH і ESP використовують MD5 для автентифікації даних.

– Алгоритм SHA-1 (Secure Hash Algorithm-1 – захищений алгоритм хешування 1). Алгоритм хешування, використовуваний для автентифікації пакетів даних. В продуктах Cisco застосовується варіант коду HMAC, що обчислюється за допомогою SHA-1. IKE, AH і ESP використовують SHA-1 для автентифікації даних.

– В рамках протоколу IKE симетричні ключі створюються за допомогою алгоритму Діффі-Хеллмана, що використовує DES, 3DES, MD5 і SHA. Протокол Діффі-Хеллмана є криптографічним протоколом, заснованим на застосуванні відкритих ключів. Він дозволяє двом сторонам погодити спільній секретний

ключ, не маючи досить надійного каналу зв'язку. Спільні секретні ключі потрібні для алгоритмів DES і HMAC. Алгоритм Діффі-Хеллмана використовується в рамках IKE для створення сеансових ключів. У продуктах Cisco підтримуються 768- і 1024-бітові групи Діффі-Хеллмана. 1024-бітна група забезпечує більш надійний захист.

Кожній асоціації захисту IPsec привласнюється індекс SPI (Security Parameter Index – індекс параметрів захисту) – число, використовуване для ідентифікації асоціації захисту IPsec. Асоціація захисту IPsec визначає використовуване перетворення IPsec (ESP і/або AH і відповідні алгоритми шифрування й хешування), межу часу існування асоціації захисту IPsec у секундах або кілобайтах, вказує необхідність застосування опції PFS, IP-Адреси сторін, а також спільні значення секретних ключів для алгоритмів шифрування й інші параметри. Всі асоціації захисту IPsec є однобічними.

Один цикл узгодження асоціації захисту IPsec завершується створенням двох асоціацій захисту: однієї вхідної й однієї вихідної.

Протоколи AH і ESP IPsec можуть діяти або в тунельному, або в транспортному режимах. Тунельний режим використовується для зв'язку між шлюзами IPsec, і в цьому випадку засобам IPsec доводиться створювати зовсім новий заголовок IPsec. Транспортний режим звичайно застосовується між клієнтом і сервером VPN, і при цьому використовується існуючий заголовок IP.

Крок 4. Передача даних

Після завершення другої фази IKE і створення асоціацій захисту IPsec у швидкому режимі, починається обмін інформацією через тунель IPsec, що зв'язує сторони IPsec. Пакети шифруються й дешифруються за допомогою алгоритмів шифрування й ключів, зазначених асоціацією захисту IPsec. Асоціація захисту IPsec задає також ліміт часу свого існування в кілобайтах переданих даних або в секундах. Асоціація захисту має спеціальний лічильник, значення якого зменшується на одиницю за кожну секунду або після передачі кожного кілобайта даних.

Крок 5. Завершення роботи тунелю IPsec

Асоціації захисту IPsec завершують свою роботу або через їхнє видалення, або тому, що виявляється перевищена межа часу їхнього існування. Коли асоціації захисту закінчують роботу то ключі, що відповідають їм теж стають недійсними. Якщо для потоку даних потрібні нові асоціації захисту IPsec, у рамках протоколу IKE знову виконується обмін другої фази, а якщо необхідно, то й першої. У результаті успішного їхнього завершення створюються нові асоціації захисту й нові ключі. Нові асоціації захисту можуть створюватися й до витікання часу існування попередніх, щоб потік даних міг рухатися безупинно. Звичайно переговори другої фази виконуються частіше, ніж переговори першої фази.

Мережі на основі IPsec можуть бути побудовані за допомогою самих різних пристроїв Cisco:

- маршрутизаторів Cisco;
- брандмауерів CiscoSecure PIX Firewall;
- програмного забезпечення клієнта CiscoSecure VPN;
- концентраторів Cisco VPN серій 3000 і 5000.

Маршрутизатори Cisco мають вбудовану підтримку VPN з відповідними багатими можливостями програмного забезпечення Cisco IOS, що зменшує складність мережних рішень і знижує загальну вартість VPN при можливості побудови багаторівневого захисту надаваних сервісів.

Брандмауер PIX Firewall є високопродуктивним мережним пристроєм, що може обслуговувати кінцеві точки тунелів, забезпечуючи їм високу пропускну здатність і прекрасні функціональні можливості брандмауера. Програмне забезпечення клієнта CiscoSecure VPN підтримує самі строгі вимоги VPN вилученого доступу для операцій електронної комерції, а також додатків мобільного доступу, пропонуючи закінчену реалізацію стандартів IPsec і забезпечуючи надійну взаємодію маршрутизаторів Cisco і брандмауерів PIX Firewall.

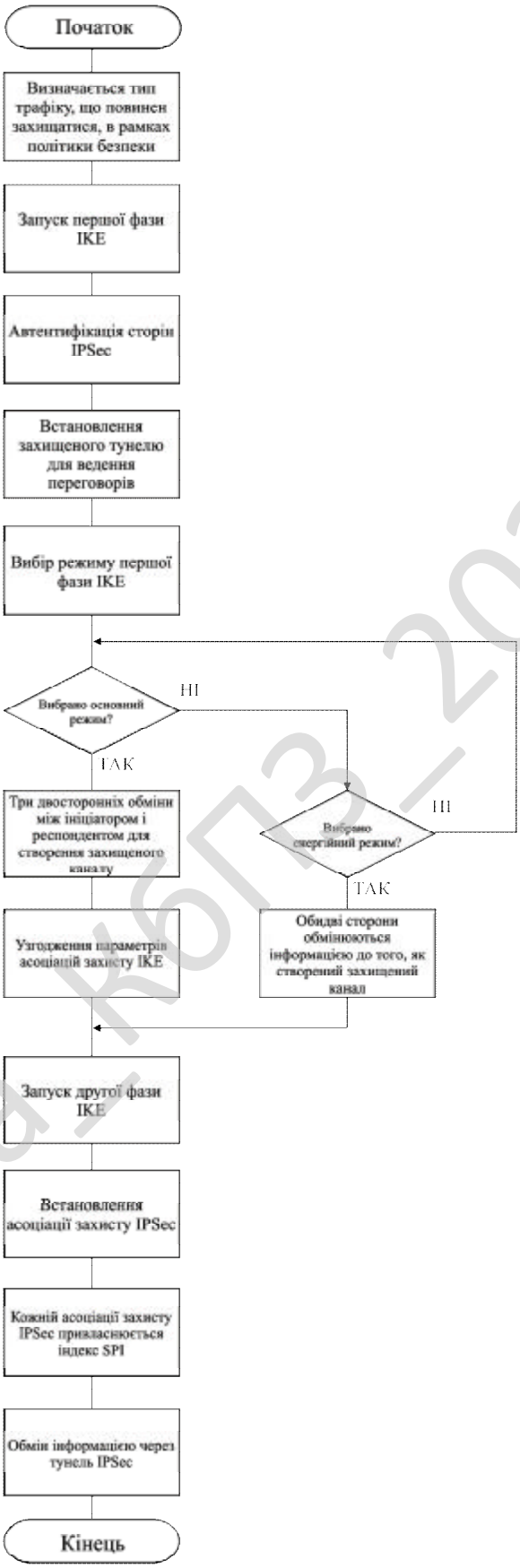

Рисунок 4.2 – Блок-схема алгоритму роботи підпрограми, що реалізує метод захисту інформації IPsec

Блок-схема алгоритму шифрування даних за допомогою SSL зображена на рисунку 4.3.

Протокол діалогу SSL має дві основні фази. Перша фаза використовується для встановлення конфіденційного каналу комунікацій. Друга – служить для автентифікації клієнта.

Фаза 1

Перша фаза є фазою ініціалізації з'єднання, коли обидва партнери посилають повідомлення "hello". Клієнт ініціює діалог посилкою повідомлення CLIENT-HELLO. Сервер одержує повідомлення CLIENT-HELLO, обробляє його й відгукується повідомленням SERVER-HELLO.До цього моменту, як клієнт, так і сервер мають досить інформації, щоб знати, чи потрібний новий майстер-ключ. Коли новий майстер-ключ не потрібний, клієнт і сервер негайно переходять у фазу 2.

Коли потрібний новий майстер-ключ, повідомлення SERVER-HELLO буде містити досить даних, щоб клієнт міг сформувати такий ключ. Сюди входить підписаний сертифікат сервера, список базових шифрів та ідентифікатор з'єднання (останній являє собою випадкове число, сформоване сервером і використовуване протягом сесії). Клієнт генерує майстер-ключ і посилає повідомлення CLIENT-MASTER-KEY (або повідомлення ERROR, якщо інформація сервера вказує, що клієнт і сервер не можуть погодити базовий шифр).

Тут варто помітити, що кожна кінцева точка SSL використовує пару шифрів для кожного з'єднання (тобто всього 4 шифри). На кожній кінцевій точці, один шифр використовується для вихідних комунікацій і один – для вхідних. Коли клієнт або сервер генерує ключ сесії, вони в дійсності формують два ключі, SERVER-READ-KEY (відомий також як CLIENT-WRITE-KEY) і SERVER-WRITE-KEY (відомий також як CLIENT-READ-KEY). Майстер-ключ використовується клієнтом і сервером для генерації різних ключів сесій.

Нарешті, після того як майстер-ключ визначений, сервер посилає клієнтові повідомлення SERVER-VERIFY. Цей заключний крок автентифікує сервер, тому

що тільки сервер, що має відповідний загальнодоступний ключ, може знати майстер-ключ.

Фаза 2

Друга фаза є фазою автентифікації. Сервер уже автентифікований клієнтом на першій фазі, із цієї причини тут здійснюється автентифікація клієнта. При типовому сценарії, серверу необхідно одержати щось від клієнта, і він надсилає запит. Клієнт надішле позитивний відгук, якщо має необхідну інформацію, або надішле повідомлення про помилку, якщо немає. Ця специфікація протоколу не визначає семантику повідомлення ERROR, що посилається у відповідь на запит сервера (наприклад, конкретна реалізація може ігнорувати помилку, закрити з'єднання, і т.д. і, проте, відповідати даній специфікації). Коли один партнер виконав автентифікацію іншого партнера, він посилає повідомлення **finished**. У випадку клієнта повідомлення CLIENT-FINISHED містить зашифровану форму ідентифікатора CONNECTION-ID, що повинен верифікувати сервер. Якщо верифікація зазнає невдачі, сервер посилає повідомлення ERROR. Раз партнер послав повідомлення **finished,** він повинен продовжити сприймати повідомлення доти, поки не одержить повідомлення **finished** від партнера. Як тільки обидва партнери послали й одержали повідомлення **finished**, протокол діалогу SSL закінчив свою роботу. З цього моменту починає працювати прикладний протокол.

Обробка помилок у протоколі з'єднань SSL досить проста. Коли помилка детектована, сторона що виявила її посилає своєму партнерові повідомлення. Помилки, які є невиправними, потрибують від клієнта й сервера розриву з'єднання. Сервер і клієнт повинні "забути" всі ідентифікатори сесії, сполучені з розірваним з'єднанням. Протокол діалогу SSL визначає наступні помилки:

– NO-CIPHER-ERROR – клієнт не може знайти шифр або розмір ключа.

– NO-CERTIFICATE-ERROR – клієнт не має сертифіката.

– BAD-CERTIFICATE-ERROR – сертифікат не дійсний.

– CERTIFICATE-TYPE-ERROR – клієнт/сервер одержав тип сертифіката, який він не підтримує.

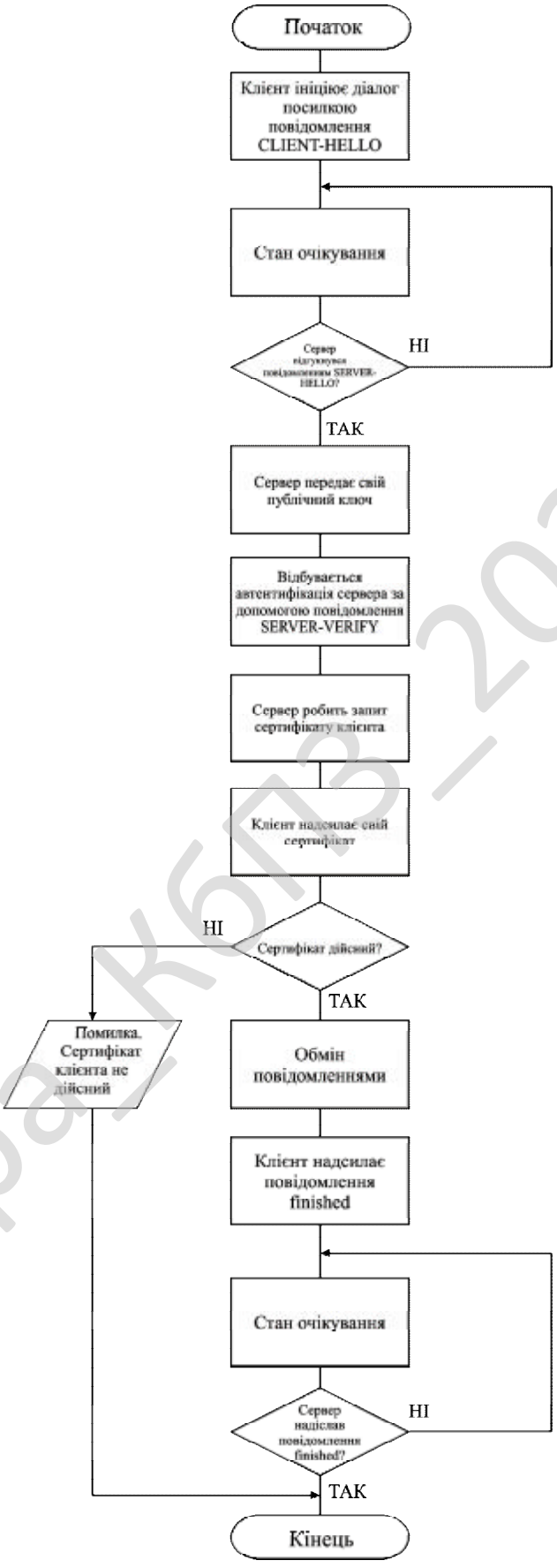

Рисунок 4.3 – Блок-схема алгоритму роботи підпрограми, що реалізує метод захисту інформації SSL

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм Khufu. Khufu – це 64-бітовий блоковий шифр. 64-бітовий відкритий тест спочатку розщеплюється на дві 32-бітові половини, *L* і *R.* Над обома половинами й певними частинами ключа виконується операція XOR. Потім, аналогічно DES, результати проходять деяку послідовність раундів. У кожному раунді молодший значущий байт *L* використовується як вхід S-блоку. У кожного S-блоку 8 вхідних біт і 32 вихідних біта. Далі обраний в S-блоці 32-бітовий елемент піддається операції XOR з *R.* Потім *L* циклічно зрушується на число, кратним восьми биткам, *L* і *R* міняються місцями, і раунд завершується. Сам S-блок не статичний, він міняється кожні вісім раундів. Нарешті, по закінченні останнього раунду, над *L* і *R* виконується операція XOR з іншими частинами ключа, і половини поєднуються, утворюючи блок шифртексту.

Хоча частини ключа використовуються для операції XOR із блоком шифрування на початку й кінці виконання алгоритму, головне призначення ключа – генерація S-блоків. Ці S-блоки секретні, по суті, це частина ключа. Повний розмір ключа алгоритму Khufu дорівнює 512 біт (64 байт), алгоритм надає спосіб генерації S-блоків по ключу.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розроблена система має зручний та інтуїтивно зрозумілий інтерфейс. Після запуску програми на екрані з'являється вікно зображене на рисунку 5.1.

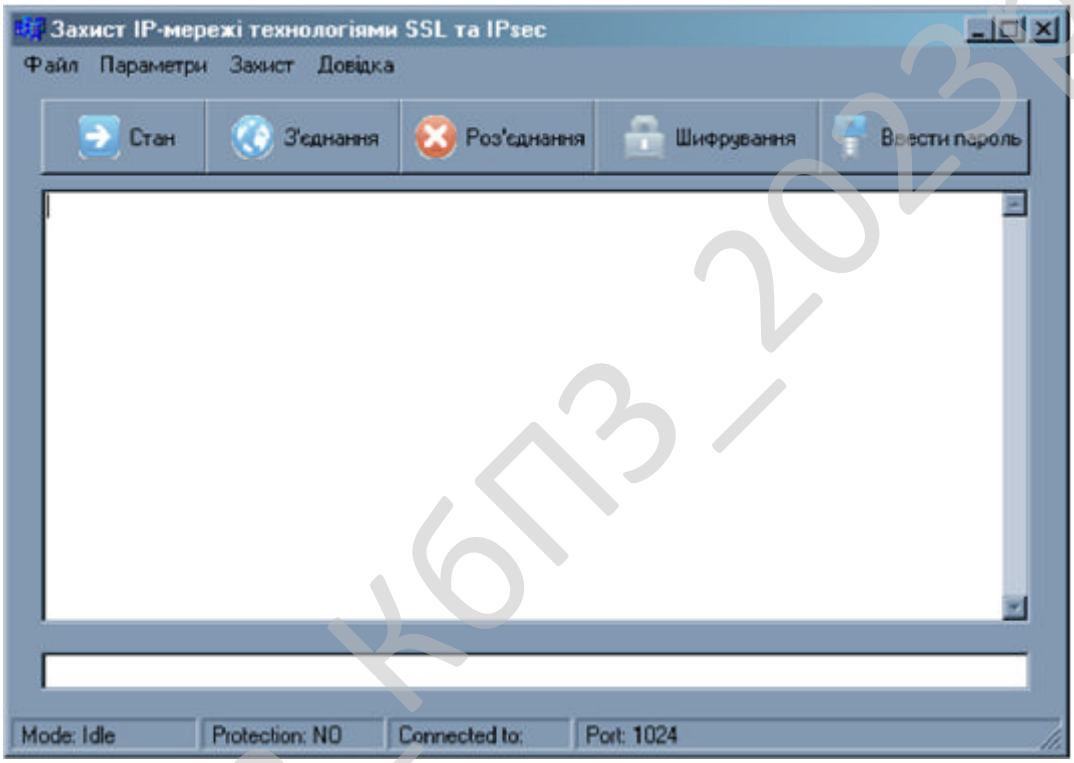

Рисунок 5.1 – Головне вікно програми

Користувач може переглянути стан мережі за допомогою кнопки "Стан". Кнопки "З'єднання" та "Роз'єднання" слугують відповідно для створення та розірвання зв'язку з мережею.

Стан з'єднання, надіслані та прийняті повідомлення відображаються у полі, розташованому в центрі вікна. Параметри з'єднання з сервером встановлюються за допомогою відповідного пункту меню користувача.

Для шифрування IP-трафіку необхідно ввести пароль, вибрати спосіб

шифрування та підтвердити вибір.

Для введення паролю слід натиснути кнопку "Ввести пароль". Після цього з'явиться діалогове вікно зображене на рисунку 5.2.

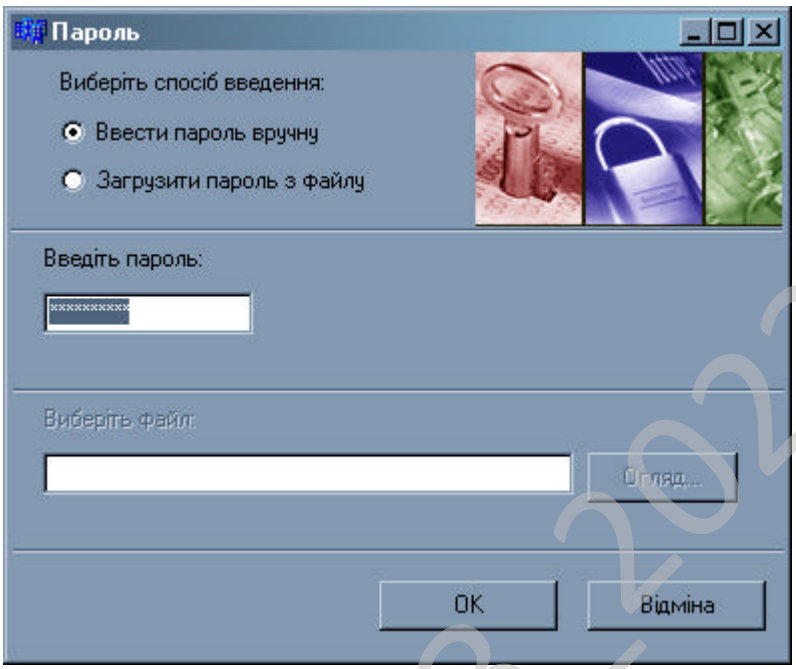

Рисунок 5.2 – Вікно введення паролю

Як видно з рисунку, користувач може ввести пароль вручну, або загрузити його з файлу.

При натисненні кнопки "Шифрування" відкривається діалогове вікно, що дозволяє вибрати способ шифрування та включити режим шифрування. Вимкнути режим шифрування трафіку можна повторним натисненням цієї кнопки.

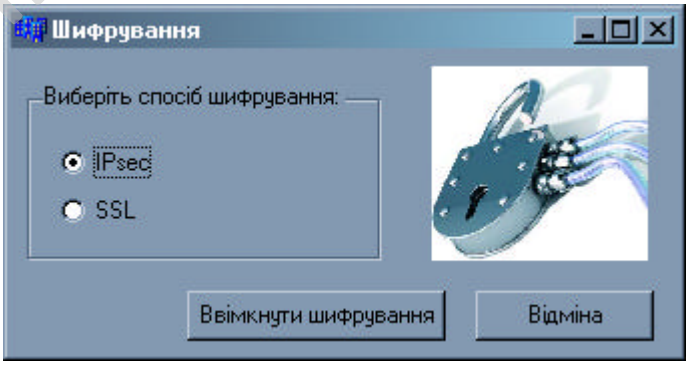

Рисунок 5.3 – Вікно ввімкнення шифрування трафіку

Користувач може вибрати технологію шифрування IPsec або SSL. Для підтвердження свого вибору користувачу слід натиснути кнопку "Ввімкнути шифрування".

Коротку інформацію про програму можна переглянути натиснувши пункт меню Довідка->Про програму… Після цього з'явиться вікно зображене на рисунку 5.4.

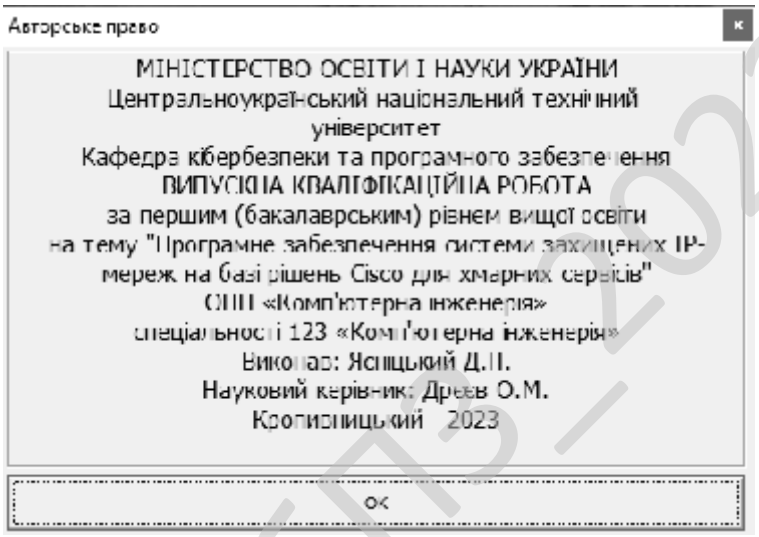

Рисунок 5.4 – Вікно довідки

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

– Досліджена система захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

– На основі отриманих результатів досліджень створена програмна реалізація системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

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 призначені для системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів. 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 Khufu.

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.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Т.В.Смірнова, Є.К. Солових, О.А.Смірнов, «Дослідження стандартів забезпечення кібербезпеки хмарних технологій як сервісів», *X міжнародна науково-технічна конференція «ITSEC»*, Єгипет, м. Шарм-ель-Шейх, 19-24 березня, 2020. с. 36.

2. Т.В.Смірнова, Л.І. Поліщук, О.А.Смірнов, «Аналіз хмарних технологій як сервісів», *XIII всеукраїнська науково-практична конференція «Комп'ютерні інтелектуальні системи та мережі (КІСМ-2020)*». м. Кривий Ріг. 24-26 березня, 2020, С. 210-212.

3. Т.В. Смірнова, Л.І. Поліщук, О.М. Дрєєв, «Застосування сервісу CAEaaS як системи інженерних розрахунків з використанням хмарних технологій», *II Міжнародна науково-практична конференція «Інформаційна безпека та інформаційні технології, Information Security and Information Technologies»*, м. Кропивницький, 2-3 квітня, 2020, с. 52.

4. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, Є.К. Солових, «Інформаційна структура технологічного процесу електродугового напилення». *Всеукраїнська науково-технічна конференція «Стан, досягнення та перспективи інформаційних систем і технологій»*, м. Одеса, 21-22 квітня, 2020, с. 184-186.

5. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, Є.К. Солових, «Реалізація інформаційної структури технологічного процесу електродугового напилення», *XXII Міжнародний науково-практичний семінар «Комбінаторні конфігурації та їх застосування»*, м. Кропивницький, 15-16 травня, 2020, с. 158-162.

6. Т.В. Смірнова, А.В. Щербань, Ю.Ю. Моторін, О.А. Смірнов, «Перспективні напрямки забезпечення кібербезпеки сервісу CAEaaS», *VI Всеукраїнська науково-практична конференція «Перспективні напрями захисту інформації*», м. Одеса, 2-6 вересня 2020, с. 58-59

7. Смирнов А.А. Метод управления доступом в интеллектуальных узлах коммутации / Мохамад Гани Абу Таам, А.А.Смирнов // Информационные технологии и защита информации в информационно-коммуникационных системах: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Вид-во ТОВ «Щедра садиба плюс», 2015. – 486 с.

8. Смирнов А.А. Математическая GERT-модель технологии передачи метаданных в облачные антивирусные системы / В.В.Босько, А.А.Смирнов, И.А.Березюк, Мохамад Гани Абу Таам // Збірник наукових праць "Системи обробки інформації". – Випуск 1(117). – Х.: ХУПС – 2014. – С. 137-141.

9. Смирнов А.А. Структурно-логическая GERT-модель технологии распространения компьютерных вирусов / А.А.Смирнов, И.А.Березюк, Мохамад Гани Абу Таам // Системи управління, навігації та зв'язку. – Випуск 1(29). – П.: ПНТУ. – 2014. – С. 120-125.

10. Смирнов А.А. Сравнительные исследования математических моделей технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / Мохамад Гани Абу Таам, А.А. Смирнов, А.В. Коваленко, С.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 9(125). – Х.: ХУПС – 2014. – С. 105-110.

11. Смирнов А.А. Математическая модель интеллектуального узла коммутации с обслуживанием информационных пакетов различного приоритета / Мохамад Гани Абу Таам, А.А. Смирнов, Н.С. Якименко, С.А. Смирнов // Збірник наукових праць Харківського університету Повітряних Сил. Випуск 4 (41). – Харків: ХУПС. – 2014. – С. 48-52.

12. Смирнов А.А. Исследование показателей качества функционирования интеллектуальных узлов коммутации в телекоммуникационных системах и сетях / Мохамад Гани Абу Таам, А.А. Смирнов, Н.С. Якименко, С.А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 4(17). – Харків: ХУПС. – 2014. – С.90-95.

13. Смирнов А.А. Усовершенствованный алгоритм управления доступом к «облачным» телекоммуникационным ресурсам / Мохамад Гани Абу Таам, А.А. Смирнов, Н.С. Якименко, С.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 1(126). – Х.: ХУПС – 2015. – С. 150-153.

14. Смирнов А.А. Анализ и исследование методов управления сетевыми ресурсами для обеспечения антивирусной защиты данных / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Системи озброєння і військова техніка. – Випуск 3(43) – Х.: ХУПС – 2015. – С. 100-107.

15. Смирнов А.А. Исследование эффективности метода управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 3(19). – Х.: ХУПС. – 2015. – С. 134-141.

16. Mohamad Abou Taam Method of controlling access to intellectual switching nodes of telecommunication networks and systems / A.A. Smirnov, Mohamad Abou Taam, S.A. Smirnov // International Journal of Computational Engineering Research (IJCER). – Volume 5, Issue 5. – India. Delhi. – 2015. – P. 1-7.

17. Смирнов А.А. GERT-модель технологии передачи данных в облачные антивирусные системы / А.А. Смирнов, В.В. Босько, Мохамад Гани Абу Таам // Збірник тез доповідей науково-практичної конференції «Застосування інформаційних технологій у підготовці та діяльності сил охорони правопорядку». м. Харків. 12-13 березня 2014 р. – Харків. АВВ МВС. – 2014. – С. 18-19.

18. Смирнов А.А. Математическое моделирование технологии передачи сигнатур в облачные антивирусные системы / Мохамад Гани Абу Таам, А.А. Смирнов // Збірник тез VI міжнародної науково-практичної конференції “Проблеми і перспективи розвитку ІТ-індустрії”. м. Харків. 17-18 квітня 2014 р. – Харків: ХНЕУ. – 2014. – С. 260.

19. Смирнов А.А. Анализ требований к качеству обслуживания в информационно-телекоммуникационных системах / А.А. Смирнов, Мохамад

Гани Абу Таам // Збірник тез XVI міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 11-12 квітня 2014 р. – Кіровоград: КНТУ. – 2014. – С. 124-126.

20. Смирнов А.А. Дослідження та реалізація GERT-моделі технології розповсюдження комп'ютерних вірусів для захисту телекомунікаційних систем / Мохамад Гані Абу Таам, С.А. Смирнов // Збірник тез науково-практичної конференції «Інформаційні технології та комп'ютерна інженерія». м. Кіровоград. 4 грудня 2014 р. – Кіровоград: КНТУ. – 2014. – С. 168.

21. Смирнов А.А. Исследование математических моделей технологии распространения компьютерных вирусов / А.А. Смирнов, Мохамад Гани Абу Таам, С.А. Смирнов // Збірник наукових праць міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 25-28 лютого 2015 р. – Київ: Європейський університет. – 2015. – С. 90-91.

22. Смирнов А.А. Метод управления доступом к «облачным» ресурсам для защиты телекоммуникационных систем / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез всеукраїнської науково-практичної конференції «Інформаційна безпека держави, суспільства та особистості». м. Кіровоград. 16 квітня 2015. – Кіровоград: КНТУ. – 2015. – С. 50-52.

23. Смирнов А.А. Разработка метода управления доступом в интеллектуальных узлах коммутации / А.А. Смирнов, Мохамад Гани Абу Таам, С.А. Смирнов // Збірник тез VII міжнародної науково-практичної конференції "Проблеми і перспективи розвитку IT-індустрії". м. Харків. 17-18 квітня 2015 р. – Харків: ХНЕУ. – 2015. – С. 14.

24. Смирнов А.А. Реализация метода управления доступом в интеллектуальных узлах коммутации / А.А. Смирнов, Мохамад Гани Абу Таам // Збірник тез XVII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 17-18 квітня 2015 р. – Кіровоград: КНТУ. – 2015. – С. 91-92.

25. Смирнов А.А. Реализация математической модели интеллектуального узла коммутации для обеспечения защищенности телекоммуникационной сети / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез II Міжнародної науково-практичної Інтернет-конференції «Інформаційна та економічна безпека» (INFECO-2015)». м. Харків. 21-22 травня 2015 р. – Харків: ХІБС УБС НБУ. – 2015. – С. 20-24.

26. Смирнов А.А. Разработка математической модели технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Сборник тезисов XI международной конференции "Стратегия качества в промышленности и образовании". г. Варна. Болгария. 01 – 06 июня 2015 г – Варна. ТУВ. – 2015. – С. 488-491

27. Смирнов А.А. Метод управления доступом к облачным телекоммуникационным ресурсам для обеспечения защиты данных / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез Міжнародної науково-практичної конференції «Комп'ютерні технології та інформаційна безпека». м. Кіровоград. 2-3 липня 2015 р. – Кіровоград: КНТУ. – 2015. – С. 4-5.

28. Смирнов А.А. Имитационная модель системы управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез першої всеукраїнської науково-практичної конференції «Перспективні напрями захисту інформації». м. Затока. 7-9 вересня 2015 р. – Одеса: ОНАЗ. – 2015. – С. 90-94.

29. МСЭ-Т Рекомендация G.101. Международные телефонные соединения и цепи – Общие определения //11/2003. [Електронний ресурс]. – Режим доступу до ресурсу: http://www. telecom61.ru›SharedFiles/Download.aspx? …pageid=106

30. Одом Ш. Коммутаторы CISCO / Ш. Одом, Х. Ноттингем – М.:"Кудиц-Образ", 2003. – 528 с.

31. Смірнова Т.В., Дрєєв О.М., «Інформаційна технологія оптимізації технологічного процесу відновлення та зміцнювання поверхонь валів зі сталі як

хмарний сервіс» у *Інформаційна безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка*. – Х. : Вид. Рожко С.Г. 2019. С. 92-107.

32. Т.В.Смірнова, Л.І. Поліщук, «Дослідження хмарних технологій як сервісів для системи інженерних розрахунків» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. С. 106-121.

33. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, «Експертна система оптимізації процесу відновлення та зміцнення поверхонь деталей типу «вал» електродуговим напиленням», *Системи управління, навігації та зв'язку*, № 2 (54). с. 149-154, 2019.

34. Т.В. Смірнова, Є.К. Солових, О.А. Смірнов, О.М. Дрєєв, «Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей», *Центральноукраїнський науковий вісник. Технічні науки.* № 1(32). с. 184-194, 2019.

35. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, Є.К. Солових, «Методи оптимізації технологічних процесів відновлення сталевих покриттів», *Shipbuilding & marine infrastructure / Суднобудування і морська інфраструктура* № 1 (11). с. 48-57, 2019.

36. Т.В.Смірнова, Л.І. Поліщук, О.А.Смірнов, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», *Кібербезпека: освіта, наука, техніка*. № 3(7). С. 43-62. 2020. (Категорія Б)

37. Т.В.Смірнова, «Формалізація та реалізація структури технологічного процесу електродугового напилення для оптимізаційної експертної системи», *Технічні науки та технології*. № 1(19). С. 104-113. 2020. (Категорія Б)

38. Т.В. Смірнова «Формування евристичних правил, бази знань та формалізація структури й правил технологічного процесу для оптимізаційної хмарної інформаційної системи», *Системи управління, навігації та зв'язку*, № 2 (60). с. 101-104, 2020. (Категорія Б)

39. Т.В. Смірнова, О.А. Смірнов, О.М. Дрєєв, С.А. Смірнов, «Використання хмарних експертних систем в сфері інформаційного забезпечення обробки поверхні деталей», *Комп'ютерна інженерія і кібербезпека: досягнення та інновації*, м. Кропивницький. 27-29 листопада, 2018, с. 111-113

40. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов «Захист даних у інформаційній технології відновлення поверхонь деталей у вигляді хмарної платформи як послуги», *Проблеми кібербезпеки інформаційно-телекомунікаційних систем*, м. Київ, 11-12 квітня, 2019, с. 221-224.

41. Т.В. Смірнова, О.М. Дрєєв, Є.К. Солових, «Хмарний сервіс інформаційної технології оптимізації технологічного процесу відновлення та зміцнювання поверхонь зі сталі», *Інформаційна безпека та інформаційні технології, Information Security and Information Technologies*, м. Харків, 24-25 квітня, 2019, с. 36

42. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, «Побудова хмарної експертної системи оптимізації технологічного процесу електродугового напилення сталевих покриттів», *Міжнародний форум з інформаційних систем і технологій INFOS-2019,* м. Харків, 24-27 квітня, 2019, с. 95-98.

43. Т. В. Смирнова, А. А. Смирнов, Е. К. Соловых «Разработка математической модели и программного обеспечения для оптимизации режима электроконтактной обработки газотермических покрытий», *Комплексне забезпечення якості технологічних процесів та систем,* том 2, м. Чернігів, 14 - 16 травня, 2019, с. 78-79.

44. Т.В. Смірнова, О.М. Дрєєв, Є.К. Солових, О.А. Смірнов, «Експертна система інформаційної технології оптимізації абстрактного технологічного процесу як хмарного сервісу», *Інформаційні технології: Наука, техніка, технологія, освіта, здоров'я. MicroCAD-2019,* м. Харків, 15-17 травня, 2019, с. 195.

45. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, «Формування абстрактних експертних систем на основі досліджень відомих експертних систем», *XXI*

Міжнародний науково-практичний семінар «Комбінаторні конфігурації та їх застосування», м. Кропивницький, 17-18 травня, 2019, с. 143-147.

46. Т.В. Смірнова, О.А. Смірнов, «Захист даних у інформаційній технології відновлення поверхонь деталей у вигляді хмарної платформи як послуги», *Інформація, комунікація, суспільство – 2019*, см.т. Чинадієво, 16-18 травня, 2019, с. 25-26

47. Т.В. Смирнова, А.Н. Дреев, А.А. Смирнов, «Информационная технология оптимизации технологического процесса восстановления и упрочнения поверхностей в виде облачного сервиса», *Modern Information, Measurement And Control Systems: Problems And Perspectives (MIMCS 2019)*, Baku, Azerbaijan, 01-02 July, 2019, p. 282.

48. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, «Огляд відомих експертних систем оптимізації технологічних процесів», *Стратегія якості в промисловості і освіті,*м. Варна, Болгарія, 3-6 червня, 2019, с.442-444

49. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, «Формалізація та узагальнення інформаційної моделі технологічних операцій зміцнення та відновлення сталевих поверхонь», *Інтелектуальні системи та інформаційні технології*, м.Одеса, 19-24 серпня, 2019, с. 211-213.

50. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов, «Формування інформаційної моделі технологічного процесу». *V Всеукраїнська науково-практична Інтернет-конференція «Електронні та мехатронні системи: теорія, інновації, практика»*, м. Полтава, 8 листопада, 2019, с. 87-91.

51. T.V. Smirnova, M.S. Chernovol, Ageev M. «Study of the spraying process and the influence of its factors on the properties of electric arc spraying coatings». *Materials of the 20th international scientific and technical seminar «Modern questions of production and repair in industry and in transport»*, *Tbilisi, Georgia*, 23-28 *March* 2020, с. 201-205.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 8-02 від 5.01.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп'ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14" 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Builder C++.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 87 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2023 р.

11.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 1.06.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Дрєєв О.М.

Програмне забезпечення системи захищених IP-мереж на базі рішень Cisco для хмарних сервісів

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 35

Літера: РП

Кропивницький – 2023 року

Основна програма

Файл Project1.cpp основної програми

```
// ---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
// ---------------------------------------------------------------------------
USEFORM("main_form.cpp", MainForm);
USEFORM("about_form.cpp", AboutForm);
USEFORM("password_form.cpp", PasswordDlg);
// ---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
                Application->Initialize();
                Application->CreateForm(__classid(TMainForm), &MainForm);
                Application->CreateForm(__classid(TAboutForm), &AboutForm);
                Application->CreateForm(__classid(TPasswordDlg), &PasswordDlg);
                Application->Run();
        }
        catch (Exception &exception)
        {
                Application->ShowException(&exception);
        }
        catch (...)
        {
                try
                {
                        throw Exception("");
                }
                catch (Exception &exception)
                {
                        Application->ShowException(&exception);
                }
        }
        return 0;
}
// ---------------------------------------------------------------------------
```

Файл main_form.cpp основної програми

```
#include <vcl.h>
#pragma hdrstop
// ---------------------------------------------------------------------------
#include "main_form.h"
#include "about_form.h"
#include "password_form.h"
// ---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
// ---------------------------------------------------------------------------
TMainForm *MainForm;
// ---------------------------------------------------------------------------
__fastcall TMainForm::TMainForm(TComponent* Owner)
        : TForm(Owner)
{
        bIsServer = false;
        bProtection = false;
        bUsingKeybdPassword = true;
        lPasswordSize = 0;
        prgbPasswordBuffer = NULL;
        prgbSendBuffer = NULL;
        prgbPasswordBuffer2 = NULL;
        pInKeyIPsec = NULL;
}
// ---------------------------------------------------------------------------
// Змінює стан Listen...
void __fastcall TMainForm::ChangeListenCondition(TObject *Sender)
{
        if (FileListenItem->Checked)
        {
               //... міняємо іконку на протилежну
               ListenSpeedButton->Glyph = ListenOffImage->Glyph;
        } else
        {
             ListenSpeedButton->Glyph = ListenOnImage->Glyph;
        }
        FileListenItem->Checked = !FileListenItem->Checked;

        if (FileListenItem->Checked)
        {
               ClientSocket->Close();
               ServerSocket->Open();
               StatusBar->Panels->Items[0]->Text = " Mode: Server";
               StatusBar->Panels->Items[2]->Text = " Connected to:";
        }else
        {
               if (ServerSocket->Active)
               {
                       ServerSocket->Close();
               }
               StatusBar->Panels->Items[0]->Text = " Mode: Idle";
               StatusBar->Panels->Items[2]->Text = " Connected to:";
        }
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::FileListenItemClick(TObject *Sender)
{
        // Змінюємо стан Listen...
        ChangeListenCondition(Sender);
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::ListenSpeedButtonClick(TObject *Sender)
{
        // Змінюємо стан Listen...
        ChangeListenCondition(Sender);
```

```
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::FileExitItemClick(TObject *Sender)
{
        // Закриваємо програму
        Close();
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::HelpAboutItemClick(TObject *Sender)
{
        // Відображаємо діалог About
        AboutForm->ShowModal();
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::ConnectTo(TObject *Sender)
{
  // Робимо запит IP-Адреси...
  if (InputQuery("Connect to...", "Address Name:", Server))
  {
     // Якщо рядок був не порожня...
     if (Server.Length() > 0)
     {
       // Якщо в цей момент були на коннекті, рвемо з'єднання...
       if (ClientSocket->Active)
       {
                ClientSocket->Close();
       }
        // Ініціалізуємо параметри сокету...
        ClientSocket->Host = Server;
        ClientSocket->Open();

        FileListenItem->Checked = false;
        ListenSpeedButton->Glyph=MainForm->ListenOffImage->Glyph;
        StatusBar->Panels->Items[0]->Text = " Mode: Client";
     }
  }
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::FileConnectItemClick(TObject *Sender)
{
        ConnectTo(Sender);
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::ConnectSpeedButtonClick(TObject *Sender)
{
        ConnectTo(Sender);
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::FormClose(TObject *Sender, TCloseAction &Action)
{
        ServerSocket->Close();
        ClientSocket->Close();

        // Очищаємо список підключень...
        WipeVCL(Sender);

        // Вивільняємо ресурси
        FreeObjects();
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::ClientSocketConnect(TObject *Sender,
      TCustomWinSocket *Socket)
{
      StatusBar->Panels->Items[2]->Text = " Connected to: " + Socket-
>RemoteHost;
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::Disconnect(TObject *Sender)
{
```

```
        // Закриваємо всі з'єднання...
        ClientSocket->Close();
        ServerSocket->Close();

        FileListenItem->Checked = false;
        ListenSpeedButton->Glyph = ListenOffImage->Glyph;

        StatusBar->Panels->Items[0]->Text = " Mode: Idle";
        bProtection = false;
        StatusBar->Panels->Items[1]->Text = " Protection: NO";
        StatusBar->Panels->Items[2]->Text = " Connected to:";

        // Вивільняємо ресурси
        FreeObjects();
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::FileDisconnectItemClick(TObject *Sender)
{
        Disconnect(Sender);
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::DisconnectSpeedButtonClick(TObject *Sender)
{
        Disconnect(Sender);
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::ClientSocketRead(TObject *Sender,
      TCustomWinSocket *Socket)
{
        // Якщо вікно додатка згорнуте в трей, указуємо на те, що
        // прийшли нові зашифровані повідомлення...
        if (bInTray)
        {
                IconBlinkTimer->Enabled = true;
        }

        // Якщо працюємо в незахищеному режимі
        if (!bProtection)
        {
                ChatMemo->Lines->Add(TimeToStr(Time()) + " (+) " + Socket-
>ReceiveText());
        } else
        {
                // Приймаємо дані...
                Socket->ReceiveBuf(prgbSendBuffer,(EncryptionBlockSize +
INT_LENGTH));
                //... і декодуємо
                DecodeSendBuffer(Sender);
        }
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::ServerSocketClientRead(TObject *Sender,
      TCustomWinSocket *Socket)
{
        // Якщо вікно додатка згорнуте в трей, указуємо на те, що
        // прийшли нові зашифровані повідомлення...
        if (bInTray)
        {
                IconBlinkTimer->Enabled = true;
        }

        // Якщо працюємо в незахищеному режимі
        if (!bProtection)
        {
                ChatMemo->Lines->Add(TimeToStr(Time()) + " (+) " + Socket-
>ReceiveText());
        } else
        {
```

```
                Socket->ReceiveBuf(prgbSendBuffer,(EncryptionBlockSize +
INT_LENGTH));
                DecodeSendBuffer(Sender);
        }
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::ServerSocketAccept(TObject *Sender,
      TCustomWinSocket *Socket)
{
        bIsServer = true;
        StatusBar->Panels->Items[2]->Text = " Connected to: " + Socket-
>RemoteAddress;
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::ClientSocketError(TObject *Sender,
      TCustomWinSocket *Socket, TErrorEvent ErrorEvent, int &ErrorCode)
{
        ChatMemo->Lines->Add("Error connecting to: " + Server);
        StatusBar->Panels->Items[0]->Text = " Mode: Idle";
        StatusBar->Panels->Items[2]->Text = " Connected to:";
        ErrorCode = 0;
}
// ---------------------------------------------------------------------------
// Підготовляє буфер для відправлення
void __fastcall TMainForm::EncodeSendBuffer(TObject *Sender)
{
        // Крок 1 - Конвертуємо дані з Edit-А в char* і додаємо
        //          після записаного рядка випадкові дані
        ConvertAnsiStringToChar(ChatEdit->Text, prgbSendBuffer);

        long int i;

        for (i = (ChatEdit->Text.Length() + 1); i < EncryptionBlockSize; i++)
        {
                prgbSendBuffer[i] = random(256);
        }

        // Крок 2 - Вибираємо випадковий індекс у межах парольного файлу
        //          і "намотуємо", починаючи з його, 1024 бт. парольних даних,
        //          заповнюючи prgbPasswordBuffer2
        long int lRandPos = random(lPasswordSize);

        long int j = lRandPos;

        for (i = 0; i < EncryptionBlockSize; i++)
        {
                prgbPasswordBuffer2[i] = prgbPasswordBuffer[j];
                j++;
                if (j == lPasswordSize)
                {
                        j=0;
                }
        }

        // Крок 3 - Шифрування даних у вихідному буфері
        pInKeyIPsec->Encrypt(prgbSendBuffer, EncryptionBlockSize,
prgbPasswordBuffer2);

        // Крок 4 - Дописуємо в кінець буфера індекс,
        // с якого "намотували" 1 кб. парольних даних
        IntToRGB.IntVar = lRandPos;

        j = EncryptionBlockSize;

        for (i = 0; i < INT_LENGTH; i++)
        {
                prgbSendBuffer[j] = IntToRGB.rgbVar[i];
                j++;
        }
```

```
}
// ---------------------------------------------------------------------------
// Декодує буфер після приймання
void __fastcall TMainForm::DecodeSendBuffer(TObject *Sender)
{
        long int i, j = EncryptionBlockSize;

        // Крок 1 - Довідаємося випадкову позицію в парольному файлі
        for (i = 0; i < INT_LENGTH; i++)
        {
                IntToRGB.rgbVar[i] = prgbSendBuffer[j];
                j++;
        }

        unsigned long int ulRandPos = IntToRGB.IntVar;

        // Якщо парольні файли не відповідають один одному по розміру приховуємо
цей факт
        if (ulRandPos >= lPasswordSize)
        {
                ulRandPos = random(lPasswordSize);
        }

        // Крок 2 - "намотуємо" дані з парольного буфера

        j = ulRandPos;

        for (i = 0; i < EncryptionBlockSize; i++)
        {
                prgbPasswordBuffer2[i] = prgbPasswordBuffer[j];
                j++;
                if (j == lPasswordSize)
                {
                        j=0;
                }
        }

        // Крок 3 - декодуємо дані
        pInKeyIPsec->Decrypt(prgbSendBuffer, EncryptionBlockSize,
prgbPasswordBuffer2);

        // Крок 4 - Після декодування переносимо дані з буфера в рядок
        //          для висновку в ChatMemo
        i = 0;

        // Установлюємо розмір рядка із запасом
        asDecodedStr.SetLength(EncryptionBlockSize);

        while (1)
        {
                asDecodedStr[i + 1]=prgbSendBuffer[i];
                if (
                                (prgbSendBuffer[i] == '\0')
                        ||
                                (i == (EncryptionBlockSize - 1))
                    )
                {
                        break;
                }
                i++;
        }

        asDecodedStr.SetLength((i + 1));

        // Додаємо рядок в Memo
        ChatMemo->Lines->Add(TimeToStr(Time()) + " (+) " + asDecodedStr);
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::ChatEditKeyDown(TObject *Sender, WORD &Key,
```

```
      TShiftState Shift)
{
        if (Key == VK_RETURN)
        {
                if (bIsServer)
                {
                        try
                        {
                                ServerSocket->Socket->Connections[0]->Connected;
                        }
                        catch (...)
                        {
                               ServerSocket->Close();
                               ChatMemo->Lines->Add("Error: Client is not
accessible!");
                               // Перший виклик скидає стан на режим Idle
                               ChangeListenCondition(Sender);
                               // Другий виклик повертає режим Server
                               ChangeListenCondition(Sender);
                               return;
                        }
                        // Якщо працюємо в незахищеному режимі...
                        if (!bProtection)
                        {
                                ServerSocket->Socket->Connections[0]->SendText(
                                ChatEdit->Text);
                        } else
                        {
                                // Підготовляємо буфер для відправлення
                                EncodeSendBuffer(Sender);
                                // Тепер відправляємо підготовлений
prgbSendBuffer
                                ServerSocket->Socket->Connections[0]->SendBuf(
                                prgbSendBuffer,(INT_LENGTH +
EncryptionBlockSize));
                        }
                }
                else
                {
                        if (ClientSocket->Socket->Connected)
                        {
                           // Якщо працюємо в незахищеному режимі...
                           if (!bProtection)
                           {
                                     ClientSocket->Socket->SendText(ChatEdit-
>
                                     Text);
                           } else
                           {
                              // Підготовляємо буфер для відправлення
                              EncodeSendBuffer(Sender);
                              // Тепер відправляємо підготовлений
prgbSendBuffer
                              ClientSocket->Socket->SendBuf(prgbSendBuffer,
                              (INT_LENGTH+EncryptionBlockSize));

                           }
                        } else
                        {
                                ClientSocket->Close();
                                ChatMemo->Lines->Add("Error: Server is not
accessible!");
                                StatusBar->Panels->Items[0]->Text = " Mode:
Idle";
                                StatusBar->Panels->Items[2]->Text = " Connected
to:";
                                return;
                        }
                }
```

```
                ChatMemo->Lines->Add(TimeToStr(Time()) + " (<) " + ChatEdit-
>Text);

                int i;

                // Спочатку повністю затираємо рядок в ChatEdit...
                for (i=1; i <= ChatEdit->Text.Length(); i++)
                {
                        ChatEdit->Text[i] = 0x00;
                }

                // ... а потім забираємо весь текст у ньому
                ChatEdit->Text = "";
        }
}
// ---------------------------------------------------------------------------
// Видає повідомлення про помилку відкриття файлу
void __fastcall TMainForm::FileErrorMessage(char *szFilename,int ErrKind)
{
        std::stringstream msg;
        msg << "Can't open " << szFilename << (ErrKind == TO_READ ? " to read!"
:
         " to write!");
        MessageDlg(msg.str().c_str(), mtError, TMsgDlgButtons() << mbOK, 0);
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::ConvertAnsiStringToChar(AnsiString asStr, char *pCh)
{
        int i = 1;
        while (i <= asStr.Length())
        {
                pCh[ i-1] = asStr[i];
                i++;
        }
        pCh[i - 1] = '\0';
}
// ---------------------------------------------------------------------------
// Повертає довжину файлу
long __fastcall TMainForm::FileLength(char *szFileName)
{
            int fHandle;
            long int lFileSize;

            fHandle = open(szFileName,O_RDONLY);
            lFileSize = filelength(fHandle);
            close(fHandle);

            return lFileSize;
}
// ---------------------------------------------------------------------------
// Забезпечує ініціалізацію:
// 1 - файлового покажчика
// 2 - змінної, що зберігає довжину файлу
// У цілому: бере на себе функції коректного відкриття файлу для читання
FILE* __fastcall TMainForm::OpenFileToRead(char *szFilename,long &lFileSize)
{
            FILE *fHandle;

            // Перевіряємо файл на можливість коректного відкриття...
            if((fHandle = fopen(szFilename,"rb")) == NULL)
            {
                 // Якщо відкрити не можна - виводимо повідомлення про
помилку...
                 FileErrorMessage(szFilename,TO_READ);
                 // ... і виходимо з функції
                 return NULL;
            }

            // ... якщо файл відкрився - все OK, але ще необхідно зробити
```

```
                // визначення його розміру, для цього потрібно перевідкрити його
                // в іншому режимі, тому його тимчасово закриваємо...
                fclose(fHandle);

                // ... і одержуємо розмір файлу.
                lFileSize = FileLength(szFilename);

                // А от і відкриття файлу для роботи з ним...
                fHandle = fopen(szFilename, "rb");

                return fHandle;
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::SetPasswordFileSpeedButtonClick(TObject *Sender)
{
        // Якщо запропоновано одержати пароль із клавіатури...
        if (bUsingKeybdPassword)
        {
                // 1) Відображаємо форму уведення пароля...
                if(
                        (PasswordDlg->ShowModal() == mrOk)
                    &&
                        (PasswordDlg->Password->Text.Length() != 0)
                   )
                {
                        // Установлюємо розмір парольного буфера
                        lPasswordSize = MAX_ENCRYPTION_BLOCK_SIZE;

                        // Створюємо необхідні об'єкти
                        AllocateObjects();

                        //...довідаємося його довжину...
                        int PasswordLen = PasswordDlg->Password->Text.Length();

                        //...копіюємо в парольний буфер...
                        for (int i = 1; i <= PasswordLen; i++)
                        {
                                prgbPasswordBuffer[i - 1] = PasswordDlg-
>Password->Text[i];
                                PasswordDlg->Password->Text[i] = 0x00;
                        }

                        //...і хешуємо пароль
                        pDHash->Hash(prgbPasswordBuffer, PasswordLen);

                        PasswordDlg->Password->Text = "";

                        // Указуємо, що захист використовується
                        bProtection = true;
                        StatusBar->Panels->Items[1]->Text = " Protection: YES";
                }

                return;
        }

        if (MainForm->OpenDialog->Execute())
        {
                // Спочатку вивільняємо ресурси, виділені під старий
                // парольний файл
                FreeObjects();
                bProtection=false;
                StatusBar->Panels->Items[1]->Text = " Protection: NO";

                // Задаємо ім'я максимальної довжини
                char *szPasswordFile = new char [MAX_NAME_LENGTH];
                ConvertAnsiStringToChar(OpenDialog->FileName,szPasswordFile);

                // Відкриваємо парольний файл
                FILE* fPassword = OpenFileToRead(szPasswordFile,lPasswordSize);
```

```
                // Якщо парольний файл не відкрився, виходимо з функції
                if (!fPassword)
                {
                        delete [] szPasswordFile;
                        return;
                }

                // а інакше виділяємо пам'ять під дані парольного файлу
                // і читаємо його в буфер
                AllocateObjects();

                fread(prgbPasswordBuffer,lPasswordSize,1,fPassword);
                fclose(fPassword);

                delete [] szPasswordFile;

                // Указуємо, що захист використовується
                bProtection = true;
                StatusBar->Panels->Items[1]->Text = " Protection: YES";
        }
}
 // ---------------------------------------------------------------------------
-
void __fastcall TMainForm::FileSetPasswordFileItemClick(TObject *Sender)
{
SetPasswordFileSpeedButtonClick(Sender);
}

// ---------------------------------------------------------------------------
// Ліквідує дані переданого масиву char з міркувань
// безпеки
void __fastcall TMainForm::WipeData (char *prgbData, long lDataSize)
{
            long int i;

            for (i = 0; i < lDataSize; i++)
            {
                 prgbData[i] = (char)0x00;
            }
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::AllocateObjects()
{
        prgbPasswordBuffer = new char[lPasswordSize];
        // Якщо не вдалося виділити пам'ять
        if (!prgbPasswordBuffer)
        {
                MessageDlg("Нмає записів в базі даних паролів" ,
                mtError, TMsgDlgButtons() << mbOK, 0);
                return;
        }

        prgbSendBuffer = new char[(MAX_ENCRYPTION_BLOCK_SIZE + INT_LENGTH)];
        if (!prgbSendBuffer)
        {
                MessageDlg("Не може розподілитися пам’ять під prgbSendBuffer!" ,
                mtError, TMsgDlgButtons() << mbOK, 0);
                return;
        }

        prgbPasswordBuffer2 = new char[MAX_ENCRYPTION_BLOCK_SIZE];
        if (!prgbPasswordBuffer2)
        {
                MessageDlg("Не може розподілитися пам’ять під
prgbPasswordBuffer2!" ,
                mtError, TMsgDlgButtons() << mbOK, 0);
                return;
        }
```

```
        // Створюємо екземпляр класу шифрування SSL
        pDHash = new CDHash(MAX_ENCRYPTION_BLOCK_SIZE);
        if (!pDHash)
        {
                MessageDlg("Не може розподілитися пам'ять під pDHash!" ,
                mtError, TMsgDlgButtons() << mbOK, 0);
                return;
        }

        // Створюємо екземпляр класу шифрування
        pInKeyIPsec = new CInKeyIPsec;
        if (!pInKeyIPsec)
        {
                MessageDlg("Не може розподілитися пам'ять під pInKeyIPsec!" ,
                mtError, TMsgDlgButtons() << mbOK, 0);
                return;
        }

        // Необхідно для шифрування
        randomize();
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::FreeObjects()
{
        // Вивільняємо ресурси...
        if (prgbPasswordBuffer)
        {
                WipeData(prgbPasswordBuffer,lPasswordSize);
                delete [] prgbPasswordBuffer;
                prgbPasswordBuffer = NULL;
        }

        if (prgbSendBuffer)
        {
                WipeData(prgbSendBuffer,(EncryptionBlockSize+INT_LENGTH));
                delete [] prgbSendBuffer;
                prgbSendBuffer = NULL;
        }

        if (prgbPasswordBuffer2)
        {
                WipeData(prgbPasswordBuffer2,EncryptionBlockSize); // char* data
                delete [] prgbPasswordBuffer2;
                prgbPasswordBuffer2 = NULL;
        }

        if (pDHash)
        {
                delete pDHash;
                pDHash = NULL;
        }

        if (pInKeyIPsec)
        {
                delete pInKeyIPsec;
                pInKeyIPsec = NULL;
        }
 }
// ---------------------------------------------------------------------------
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
        // Установлюємо початковий стан блоку шифрування
        EncryptionBlockSize = 1024;
        ChatEdit->MaxLength = EncryptionBlockSize;

        // Завантажуємо іконки в ImageList
        IconsImageList->Add(NormalIconImage->Picture->Bitmap, NULL);
        IconsImageList->Add(BlinkIconImage->Picture->Bitmap, NULL);
```

```
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::IconBlinkTimerTimer(TObject *Sender)
{
        // Перекидаємо індекс іконки в масиві іконок
        TrayIcon->IconIndex ^= 0x01;
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::TrayIconClick(TObject *Sender)
{
        // Якщо звернулися до додатка...
        bInTray = false;

        // ...відключаємо таймер...
        IconBlinkTimer->Enabled = false;
        // ... і повертаємо нормальну іконку
        TrayIcon->IconIndex = 0x00;
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::TrayIconMinimize(TObject *Sender)
{
        // Указуємо, що додаток у треї...
        bInTray = true;
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::WipeVCL(TObject *Sender)
{
        int i,j;

        // Спочатку повністю затираємо рядок в ChatEdit...
        for (i=1; i <= ChatEdit->Text.Length(); i++)
        {
                ChatEdit->Text[i] = 0x00;
        }

        // Потім займаємося ChatMemo...
        for (j=0; j < ChatMemo->Lines->Count; j++)
        {
                // Затираємо всі символи в кожному рядку...
                for (i=1; i <= (ChatMemo->Lines->operator [](j).Length()); i++)
                {
                        ChatMemo->Lines->operator [](j)[i] = 0x00;
                }
        }

}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::EditClearClick(TObject *Sender)
{
        // Очищаємо список підключень...
        WipeVCL(Sender);
        ChatMemo->Clear();
        ChatEdit->Text = "";
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::EditTCPPortClick(TObject *Sender)
{
        // Викликаємо діалог установки порту...
          if (InputQuery("Set to...", "TCP Port:", Port))
          {
             unsigned int TCPPort = StrToInt(Port);
             if (
                                (Port.Length() > 0)
                        &&
                                ((TCPPort > 0)
                        &&
                                (TCPPort <= 65535))
                )
             {
```

```
                        ClientSocket->Port = TCPPort;
                        ServerSocket->Port = TCPPort;
                        StatusBar->Panels->Items[3]->Text = " Port: " + Port;
                }
              }
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::N8192bitClick(TObject *Sender)
{
        // Якщо пункт меню ще не виділений, виділяємо
        if (!((TMenuItem*)Sender)->Checked)
        {
                EncryptionBlockSize = 1024;
                ChatEdit->MaxLength = EncryptionBlockSize;

                ((TMenuItem*)Sender)->Checked = true;
        }
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::N16384bitClick(TObject *Sender)
{
        // Якщо пункт меню ще не виділений, виділяємо
        if (!((TMenuItem*)Sender)->Checked)
        {
                EncryptionBlockSize = 2048;
                ChatEdit->MaxLength = EncryptionBlockSize;

                ((TMenuItem*)Sender)->Checked = true;
        }
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::N24576bitClick(TObject *Sender)
{
         // Якщо пункт меню ще не виділений, виділяємо
        if (!((TMenuItem*)Sender)->Checked)
        {
                EncryptionBlockSize = 3072;
                ChatEdit->MaxLength = EncryptionBlockSize;

                ((TMenuItem*)Sender)->Checked = true;
        }
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::N32768bitClick(TObject *Sender)
{
        // Якщо пункт меню ще не виділений, виділяємо
        if (!((TMenuItem*)Sender)->Checked)
        {
                EncryptionBlockSize = 4096;
                ChatEdit->MaxLength = EncryptionBlockSize;

                ((TMenuItem*)Sender)->Checked = true;
        }
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::PasswordModeClick(TObject *Sender)
{
         // Якщо пункт меню ще не виділений, виділяємо
        if (!((TMenuItem*)Sender)->Checked)
        {
                ((TMenuItem*)Sender)->Checked = true;
                 bUsingKeybdPassword = true;
        } else
        {
                ((TMenuItem*)Sender)->Checked = false;
                 bUsingKeybdPassword = false;
        }
}
// ---------------------------------------------------------------------------
```

```
void __fastcall TMainForm::ProtectionResetClick(TObject *Sender)
{
        // Указуємо, що захист відключений...
        bProtection = false;
        StatusBar->Panels->Items[1]->Text = " Protection: NO";

        // Вивільняємо ресурси...
        FreeObjects();
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::ClearClick(TObject *Sender)
{
        // Очищаємо список підключень...
        WipeVCL(Sender);
        ChatMemo->Clear();
        ChatEdit->Text = "";

        // ...відключаємо таймер...
        IconBlinkTimer->Enabled = false;
        // ... і повертаємо нормальну іконку
        TrayIcon->IconIndex = 0x00;
}
// ---------------------------------------------------------------------------
void __fastcall TMainForm::ExitClick(TObject *Sender)
{
        Close();
}
```

Файл main_form.h - бібліотека для файлу main_form.cpp

```
      #ifndef about_form
      #define about_form
      // ---------------------------------------------------------------------------
------
      #include <Classes.hpp>
      #include <Controls.hpp>
      #include <StdCtrls.hpp>
      #include <Forms.hpp>
      #include <ExtCtrls.hpp>
      #include <Graphics.hpp>
      // ---------------------------------------------------------------------------
------
      class TAboutForm : public TForm
      {
      __published:      // IDE-менеджер компонентів
              TButton *OKButton;
              TImage *AboutImage;
              TBevel *AboutBevel;
              TStaticText *VersionStaticText;
              TStaticText *CopyrightStaticText;
              TStaticText *ProtectionVersionStaticText;
              TStaticText *ProtectionVersionStaticText2;
              void __fastcall OKButtonClick(TObject *Sender);
      private:    // задає користувач
      public:           // Задає користувач
              __fastcall TAboutForm(TComponent* Owner);
      };
      // ---------------------------------------------------------------------------
------
      extern PACKAGE TAboutForm *AboutForm;
      // ---------------------------------------------------------------------------
------
      #endif
```

Файл password_form.cpp - вікно введення паролів

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "password_form.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TPasswordDlg *PasswordDlg;
//---------------------------------------------------------------------------
__fastcall TPasswordDlg::TPasswordDlg(TComponent* AOwner)
      : TForm(AOwner)
{
}
//---------------------------------------------------------------------------
```

Файл password_form.h - бібліотека для файлу password_form.cpp

```
// ---------------------------------------------------------------------------
i
#ifndef password_form
#define password_form
// ---------------------------------------------------------------------------
i
#include <vcl\Buttons.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Controls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Graphics.hpp>
#include <vcl\Classes.hpp>
#include <vcl\SysUtils.hpp>
#include <vcl\Windows.hpp>
#include <vcl\System.hpp>
// ---------------------------------------------------------------------------
i
class TPasswordDlg : public TForm
{
__published:
        TLabel *PasswordLabel;
      TEdit *Password;
      TButton *OKBtn;
      TButton *CancelBtn;
private:
public:
      virtual __fastcall TPasswordDlg(TComponent* AOwner);
};
// ---------------------------------------------------------------------------
i
extern PACKAGE TPasswordDlg *PasswordDlg;
// ---------------------------------------------------------------------------
i
#endif
```

Файл inKeyIPsec.cpp - шифрування IPsec

```
#pragma once

#include "inKeyIPsec.h"

CInKeyIPsec::CInKeyIPsec()
{
    this->UpdateProgress = NULL;

    Initialize();
}

CInKeyIPsec::CInKeyIPsec(void (*UpdateProgress)(int progress))
{
    // Підписуємося на функтор-оброблювач відновлення прогресу
    this->UpdateProgress = UpdateProgress;

    Initialize();
}

void CInKeyIPsec::Initialize()
{
      // Ініціалізуємо масив "бітами", які будуть брати участь
    // в операціях виділення відповідних бітів
    rgbBitMask[0] = (char)0x01;
    rgbBitMask[1] = (char)0x02;
    rgbBitMask[2] = (char)0x04;
    rgbBitMask[3] = (char)0x08;
    rgbBitMask[4] = (char)0x10;
    rgbBitMask[5] = (char)0x20;
    rgbBitMask[6] = (char)0x40;
    rgbBitMask[7] = (char)0x80;

    // Ініціалізуємо множину векторів MVS - 56 штук
    prgubMVS = new unsigned char[MVS_NUM * NBITS];
}

CInKeyIPsec::~CInKeyIPsec()
{
    // Деініціалізуємо множину векторів MVS
      if (prgubMVS)
      {
            delete [] prgubMVS;
      }
}

////////////////////////////////////////////////////////////////////////////
// Генерується множина векторів, призначених для зміни
// порядку проходження "стовпців". // РЕКУРСИВНА ЧАСТИНА АЛГОРИТМУ
void CInKeyIPsec::GenerateMVS(char rgbBank[NBITS],int bankLen,
                              char rgbMVVector[NBITS])
{
      int i, j;

      char rgbSavedBank[NBITS];

      // Якщо ми використовували всі символи для генерування векторів,
      // те продовжувати генерування не представляється можливим
      if (bankLen == 0)
      {
            // ТУТ ---> ЗАПИС ГОТОВОГО ВЕКТОРА В БЛОК ВЕКТОРІВ
            for (j = 0; j < NBITS; j++)
            {
                  prgubMVS[cMVS * NBITS + j] = rgbMVVector[j];
            }
            cMVS++;
```

```
                return;
        }

        // Цей цикл породжує множину рекурсивних викликів,
        // додаючи один символ у вектор, і передаючи його долілиць по
        // рекурсії. Так, наприклад, перша гілка рекурсії
        // створить вектор "01234567".

        // Зберігаємо вихідний стан переданого банку
        for (j = 0; j < bankLen; j++)
        {
                rgbSavedBank[j] = rgbBank[j];
        }

        for (i = 0; i < bankLen; i++)
        {
           // Кожний з векторів, одержуваних у різних рекурсивних
           // піддеревах буде відрізнятися від іншого на 1 символ на цієї
           // стадії
           rgbMVVector[NBITS - bankLen] = rgbBank[i];

           // Тепер, коли ОДИН ІЗ СИМВОЛІВ ВИЛУЧЕНИЙ,
           // банк повинен зменшитися на ЦЕЙ СИМВОЛ і
           // потім надійти в скороченому виді в
           // подальшу рекурсію.
           // Переносимо символи вліво, затираючи "дірку"
           // символами, що залишилися в банку
           for (j = i; j < (bankLen - 1); j++)
           {
                     rgbBank[j] = rgbBank[j + 1];
           }

           // РЕКУРСІЯ
           GenerateMVS(rgbBank, (bankLen - 1), rgbMVVector);

           // Тепер відновлюємо той стан банку, у якому
           // він був переданий у рекурсивну функцію
           for (j = 0; j < bankLen; j++)
           {
                     rgbBank[j] = rgbSavedBank[j];
           }
        }
}

////////////////////////////////////////////////////////////////////////////
// Генерується множина векторів, призначених для зміни
// порядку проходження "стовпців".
// BankLen - кількість символів у масиві Bank для генерування
// кожного вектора в масиві. Щораз, коли черговий символ
// породжує піддерево комбінацій, він зникає. Таким чином,
// генеріруемое піддерево вже не буде містити свій
// корінь у жодному з піддерев.
void CInKeyIPsec::StartToGenerateMVS()
{
        // ! ПОТЕНЦІЙНА КРАПКА МОДИФІКУВАННЯ АЛГОРИТМУ
        char rgbBank[NBITS] = {(char)0x00,(char)0x01,(char)0x02,(char)0x03,

(char)0x04,(char)0x05,(char)0x06,(char)0x07};
        // ! ПОТЕНЦІЙНА КРАПКА МОДИФІКУВАННЯ АЛГОРИТМУ

        char rgbMVVector[NBITS];

        // Довжина початкового банку символів
        int bankLen = NBITS;

        // Лічильник кількості записаних векторів для переміщення
        // "стовпців"
        cMVS = 0;
```

```
	// Рекурсивна функція генерування множини векторів
	// для зміни порядку проходження стовпців -
	// заповнює кожний вектор значеннями 0..7
	GenerateMVS(rgbBank, bankLen, rgbMVVector);
}

////////////////////////////////////////////////////////////////////////////
// Ліквідує дані переданого масиву char з міркувань
// безпеки
void CInKeyIPsec::WipeData(char *prgbData, long int lDataSize)
{
	long int i;

	for (i = 0; i < lDataSize; i++)
	{
		prgbData[i] = (char)0x00;
	}
}

////////////////////////////////////////////////////////////////////////////
// Повертає стан необхідного біта з переданого байта
inline char CInKeyIPsec::Getbit(char byte, int nbit)
{
	byte &= rgbBitMask[nbit];

	if (byte != 0)
	{
		return 1;
	} else
	{
		return 0;
	}
}

////////////////////////////////////////////////////////////////////////////
// Установлює стан необхідного біта в переданий байт,
// яке закодовано байтом. Повертає модифікований байт
// !!! Увага !!! Перед накладенням він повинен бути встановлений в 0!
inline char CInKeyIPsec::Putbit(char byte, int nbit, char bit)
{
	if (bit != 0)
	{
		byte |= rgbBitMask[nbit];
	}

	return byte;
}

////////////////////////////////////////////////////////////////////////////
// Перекодування вихідних даних в "рядок"
void CInKeyIPsec::ConvertSourceToBitData(char *prgbSourceBuffer,
                                                    long int lSourceSize, char
*prgbBitData)
{
   long int i, j;

   int nbit;

   // Обробляємо кожний із символів вихідної гами...
   for (i = 0, j = 0; i < lSourceSize; i++)
   {
           // У байті 8 біт, витягаємо кожного...
      for (nbit = 0; nbit < NBITS; nbit++,j++)
      {
             prgbBitData[j] = Getbit(prgbSourceBuffer[i], nbit);
      }
   }
}
```

```
////////////////////////////////////////////////////////////////////////////
// Перекодування дані "рядка" у блок вихідних даних
void CInKeyIPsec::ConvertBitDataToSource(char *prgbSourceBuffer,
                                                    long int lSourceSize, char
*prgbBitData)
{
   long int i, j = 0;

   int nbit;

   // Ураховуємо довісок
   if (bitDataPosition != 0)
   {
            j = NBITS * lSourceSize;
   }

   // Всі вихідні байти повинні бути повернуті на місця...
   for (i = 0; i < lSourceSize; i++)
   {
          // У байті 8 біт, повертаємо на місце кожний...
        for (nbit = 0; nbit < NBITS; nbit++,j++)
        {
              prgbSourceBuffer[i] = Putbit(prgbSourceBuffer[i], nbit,
prgbBitData[j]);
        }
   }
}
////////////////////////////////////////////////////////////////////////
// Обертає prgBitData (ліквідація циклів перемішування IPSec)
void CInKeyIPsec::RotateBitData (long int lSourceSize,char *prgbPasswordBuffer,
                                          char *prgbBitData, long int L, int
mode)
{
      long int i, j;
      int sPartIndex;

      // Індекс, з якого починається "друга частина" даних
      sPartIndex = (unsigned char)prgbPasswordBuffer[L] + 1;

      // Напрямок обертання прямо залежить від режиму роботи
      // програми - шифрування або розшифровка IPSec
      switch (mode)
      {
            case ENCRYPT:
            {
                  // Переносимо "другу частину" даних на перше місце
                  for (i = sPartIndex, j = 0; i < (NBITS * lSourceSize);
i++, j++)
                  {
                        prgbBitData[(bitDataPosition ^ 0x01) * NBITS *
lSourceSize + j] = prgbBitData[bitDataPosition * NBITS * lSourceSize + i];
                  }

                  // Переносимо "першу частину" даних IPSec на друге місце

                  for (i = (sPartIndex - 1); j < (NBITS * lSourceSize); i-
i--, j++)
                  {
                        prgbBitData[(bitDataPosition ^ 0x01) * NBITS *
lSourceSize + j] = prgbBitData[bitDataPosition * NBITS * lSourceSize + i];
                  }

                  // Перекидаємо "покажчик" на дані
                  bitDataPosition ^= 0x01;

                  break;
            }
```

```
            case DECRYPT:
            {
                        // Переносимо "першу частину" даних на перше місце
                        for (i = (NBITS * lSourceSize - sPartIndex), j =
(sPartIndex - 1); j >= 0; i++, j-j--)
                        {
                              prgbBitData[(bitDataPosition ^ 0x01) * NBITS *
lSourceSize + j] = prgbBitData[bitDataPosition * NBITS * lSourceSize + i];
                        }

                        // Переносимо "другу частину" даних на друге місце

                        for (i = 0, j = sPartIndex; j < (NBITS * lSourceSize);
i++, j++)
                        {
                              prgbBitData[(bitDataPosition ^ 0x01) * NBITS *
lSourceSize + j] = prgbBitData[bitDataPosition * NBITS * lSourceSize + i];
                        }

                        // Перекидаємо "покажчик" на дані
                        bitDataPosition ^= 0x01;
            }
      }
}

////////////////////////////////////////////////////////////////////
// УСТАНОВЛЮЄ ІНДЕКС iMVS і РЕАЛІЗУЄ ЙОГО (перемішування бітів)
void CInKeyIPsec::MixBitData(long int lSourceSize, char *prgbPasswordBuffer,
                                    char *prgbBitData, long int L, long int R,
int mode)
{
      int iMVS; // Індекс у масиві покажчиків на char-вектори
                  // перестановок

      int nbit, nbit2 = 0;

      long int i, lIndexTarget, lIndexSource;

      // Цей union необхідний для формування індексу в MVS
      // Індекс складається із двох байтів парольного блоку
      // При наступній ітерації відбувається зсув "рамки зчитування"
      // двох байт по буфері парольного файлу
      static TCharToInt charToInt;

      // Ініціалізуємо "вихідний" блок даних в union-е
      // даними, узятими з парольного блоку даних
      charToInt.rgbVariable[0] = prgbPasswordBuffer[L];
      charToInt.rgbVariable[1] = prgbPasswordBuffer[R];

      // Здійснюємо перетворення Char[2] ---> Int
      iMVS = charToInt.intVariable;

      // Нормалізуємо індекс ВИБОРУ ПОТОЧНОГО ВЕКТОРА
      if (iMVS > (MVS_NUM - 1))
      {
            iMVS -= MVS_NUM;
      }

      // РЕАЛІЗУЄМО ВЕКТОР ПЕРЕСТАНОВКИ ДАНИХ:
      // Необхідно ВСІ N-Е   Б І Т Ы    вихідного блоку
      // помістити в N-ий  "С Т О В П Е Ц Ь" блоку-близнюка
      for (nbit = 0; nbit < NBITS; nbit++)
      {
            // N-Ых бітів стільки, скільки байтів у блоці вихідних даних
            for (i = 0; i < lSourceSize; i++)
            {
                  // При перенесенні даних завжди пишемо в масив-близнюк
                  // НОРМАЛЬНА АДРЕСАЦІЯ - ЧИТАННЯ ПО РЯДКАХ -
nbit*lSourceSize+i
```

```
                  // ТРАНСПОНОВАНА АДРЕСАЦІЯ - ЧИТАННЯ ПО СТОВПЦЯХ -
nbit*i+lSourceSize

                  // Режим адресації прямо залежить від режиму роботи програми -
                  // шифрування або розшифровка
                  switch (mode)
                  {
                        case ENCRYPT:
                        {
                              // Беремо по векторі - пишемо підряд
                              lIndexSource = (long int)((prgubMVS[iMVS * NBITS +
nbit]) * lSourceSize + i);
                              lIndexTarget = nbit * lSourceSize + i;

                              break;
                        }

                        case DECRYPT:
                        {
                              // Беремо підряд - пишемо по векторі
                              lIndexSource = nbit * lSourceSize + i;
                              lIndexTarget = (long int)prgubMVS[iMVS * NBITS +
nbit] * lSourceSize + i;
                        }
                  }

                  // Ураховуємо можливий зсув на другу частину масиву
                  // "бітових" даних
                  // Якщо читаємо із другої частини масиву, те
                  // bitDataPosition != 0, отже, необхідно
                  // додати "довесок" до "вихідного" індексу...
                  if (bitDataPosition != 0)
                  {
                        lIndexSource += (NBITS * lSourceSize);

                  } else
                  // ... а якщо bitDataPosition==0, то для
                  // коректного запису в більше високий шар
                  // також вносимо "довісок"
                  {
                        lIndexTarget += (NBITS * lSourceSize);
                  }

                  // Переміщаємо дані
                  prgbBitData[lIndexTarget] = (prgbBitData[lIndexSource] ^
Getbit(prgbPasswordBuffer[i], nbit2));

                  nbit2++;

                  if (nbit2 >= NBITS)
                  {
                        nbit2 = 0;
                  }
            }
      }
      // "Перекидаємо" покажчик на один із двох логічних підмасивів
      // у масиві prgbBitData
      bitDataPosition ^= (char)0x01;
}

//////////////////////////////////////////////////////////////////////
// Шифрування вихідного блоку даних парольним блоком даних
void CInKeyIPsec::Encrypt(char *prgbSourceBuffer, long int lSourceSize,
                          char *prgbPasswordBuffer)
{
      long int i;

      // Курсори в блоці парольних даних
      long int L, R;
```

```
      char *prgbBitData = new char[2 * NBITS * lSourceSize];


      int mode = ENCRYPT;

    // ГЕНЕРУЄМО MVS - Move Vector
      StartToGenerateMVS();

    // Споконвічно дані лежать на початку масиву (після перекодування)
    bitDataPosition = 0;

      // Курсор R у парольному буфері встає на своє місце...
      R = (lSourceSize - 1);
      // а L - на своє.
      L = 0;

      // Перекодування вихідних даних в "бітове" відображення
      ConvertSourceToBitData(prgbSourceBuffer, lSourceSize, prgbBitData);

      // Очищаємо буфер вихідних даних, щоб забезпечити коректне
      // накладення бітових даних з prgbBitData
      WipeData(prgbSourceBuffer, lSourceSize);

      ////////////////////////////////////////
      // Обробка даних: ШИФРУВАННЯ IPSec
      ////////////////////////////////////////
      // Працюємо доти, поки всі парольні дані IPSec не використані
      for (i = 0; i < lSourceSize; i++)
      {
            // Установка вектора перестановок (циклічний виклик приводить
            // до зміни порядку проходження "бітів")
            MixBitData(lSourceSize, prgbPasswordBuffer,
                              prgbBitData, L, R, mode);

            // Обертаємо бітові дані першим алгоритмом
            RotateBitData(lSourceSize, prgbPasswordBuffer,
                              prgbBitData, L, mode);

            L++;

            R-R--;

            // Обновляємо прогрес
            if (UpdateProgress != NULL)
            {
                  UpdateProgress(i);
            }
      }

      // Перекодування даних з "бітової" форми у вихідну
      ConvertBitDataToSource(prgbSourceBuffer, lSourceSize, prgbBitData);

      // Знищуємо дані...
      WipeData(prgbBitData, 2 * NBITS * lSourceSize);

      if (prgbBitData)
      {
            delete [] prgbBitData;
            prgbBitData = NULL;
      }
}

////////////////////////////////////////////////////////////////////
// Шифрування вихідного блоку даних IPSec парольним блоком даних
void CInKeyIPsec::Decrypt(char *prgbSourceBuffer, long int lSourceSize,
                              char *prgbPasswordBuffer)
{
      long int i;
```

```
	// Курсори в блоці парольних даних
	long int L, R;

	char *prgbBitData = new char[2 * NBITS * lSourceSize];

	int mode = DECRYPT;

// ГЕНЕРУЄМО MVS - Move Vector
	StartToGenerateMVS();

// Споконвічно дані лежать на початку масиву (після перекодування)
bitDataPosition = 0;

	// Курсор R у парольному буфері встає на своє місце...
	L = (lSourceSize - 1);
	// а L - на своє.
	R = 0;

	// Перекодування вихідних даних в "бітове" відображення
	ConvertSourceToBitData(prgbSourceBuffer, lSourceSize, prgbBitData);

	// Очищаємо буфер вихідних даних, щоб забезпечити коректне
	// накладення бітових даних з prgbBitData
	WipeData(prgbSourceBuffer, lSourceSize);

	////////////////////////////////
	// Обробка даних: РОЗШИФРОВКА IPSec
	////////////////////////////////
	// Працюємо доти, поки всі парольні дані IPSec не використані
	for (i = 0; i < lSourceSize; i++)
	{
		// Обертаємо бітові дані першим алгоритмом
		RotateBitData(lSourceSize, prgbPasswordBuffer,
				prgbBitData, L, mode);

		// Установка вектора перестановок (циклічний виклик приводить
		// до зміни порядку проходження "бітів")
		MixBitData(lSourceSize, prgbPasswordBuffer,
				prgbBitData, L, R, mode);

		L-L--;

		R++;

		// Обновляємо прогрес
		if (UpdateProgress != NULL)
		{
			UpdateProgress(i);
		}
	}

	// Перекодування даних з "бітової" форми у вихідну
	ConvertBitDataToSource(prgbSourceBuffer, lSourceSize, prgbBitData);

	// Знищуємо дані...
	WipeData(prgbBitData, 2 * NBITS * lSourceSize);

	if (prgbBitData)
	{
		delete [] prgbBitData;
		prgbBitData = NULL;
	}
}
```

Файл inKeyIPsec.h - бібліотека для файлу inKeyIPsec.cpp

```
#pragma once

#define MVS_NUM 40320  // Кількість перестановок у векторі з 8-і
                              // символів
#define NBITS 8        // MVS_NUM = NBITS! (факторіал)

#define ENCRYPT 1      // Режим: ШИФРУВАТИ
#define DECRYPT 2      // Режим: РОЗШИФРУВАТИ

//////////////////////////////////////////////////////////////////////////
// Цей union необхідний для формування індексу в MVS
// Індекс складається із двох байтів парольного блоку
// При наступній ітерації відбувається зсув "рамки зчитування"
// двох байт по буфері парольного файлу
typedef union
{
      int intVariable;
      char rgbVariable[2];
} TCharToInt;

class CInKeyIPsec
{
public:
      CInKeyIPsec();
      CInKeyIPsec(void (*UpdateProgress)(int progress));
      virtual ~CInKeyIPsec();

      void Encrypt(char *prgbSourceBuffer, long int lSourceSize,
                           char *prgbPasswordBuffer);
      void Decrypt(char *prgbSourceBuffer, long int lSourceSize,
                           char *prgbPasswordBuffer);
private:
      unsigned char *prgubMVS;
    char rgbBitMask[NBITS];
      int cMVS;
      int bitDataPosition;
      void Initialize();
      void GenerateMVS(char rgbBank[NBITS], int bankLen,
                                    char rgbMVVector[NBITS]);
      void StartToGenerateMVS();
      void WipeData(char *prgbData, long int lDataSize);
      inline char Getbit(char byte, int nbit);
      inline char Putbit(char byte, int nbit, char bit);
      void ConvertSourceToBitData(char *prgbSourceBuffer,
                                    long int lSourceSize, char *prgbBitData);
      void ConvertBitDataToSource(char *prgbSourceBuffer,
                                    long int lSourceSize, char *prgbBitData);
      void RotateBitData(long int lSourceSize, char *prgbPasswordBuffer,
                        char *prgbBitData, long int L,int mode);
      void MixBitData(long int lSourceSize, char *prgbPasswordBuffer,
                     char *prgbBitData, long int L, long int R, int mode);
      void (*UpdateProgress)(int progress);
};
```

Файл DSSL.cpp - шифрування SSL

```
#pragma once

#include "DSSL.h"
#include <math.h>

CDSSL::CDSSL()
{
        this->UpdateProgress = 0;
            Initialize(1024);
}

CDSSL::CDSSL(int SSLLen)
{
        this->UpdateProgress = 0;
            Initialize(SSLLen);
}

CDSSL::CDSSL(int SSLLen, void (*UpdateProgress)(int progress))
{
      // Підписуємося на функтор-оброблювач відновлення прогресу
    this->UpdateProgress = UpdateProgress;
      Initialize(SSLLen);
}

void CDSSL::Initialize(int SSLLen)
{
      // Фіксуємо розмір хеша
      SSLGammaLen  = SSLLen;

      // Виділяємо пам'ять під властивості класу шифрування SSL
      prgSourceData = new int[SSLGammaLen];
      prgSSL       = new int[SSLGammaLen];

      // Таблиця перекодування блоку вихідних даних
      int rgCodeTbl[256] =
      {
          34,254, 82,184,160,96,242,222,55,209,212,126,146, 23, 33,43,
          39,134, 59,128,236, 38,155,170, 69,172,252,238, 47,121, 228,
            183,203,135,165,166, 60, 98,7,207,120,189,210,8,226, 41, 72,
            253,71,24,171,196,101,168,169,186,0,46,68, 95,237,65,53,208,
            211, 83,114,157,144,32,193,143,36,250,75,234,49,167,125,141,
              58, 10,103,198,151,109, 37,112, 84,231,224,185,138,152, 94,
            99,139,22,191,136,111,162,227,118,26,102,12,50,132,21,76,213,
            73,197,16,119,48,140,110,54,45,4,148,192,205,158,124,214,180,
              217,230, 86,6, 28,221,107, 2,42,220,201,133, 74, 64, 20,129,
            159,92,66,246,13,216,40, 62,291,31, 3,85,206,145,79,154,142,
            204,117,223,195,244, 90,87,116,104,161,56, 179,77,225,150,5,
              194,174,251,229,115,57,249,215, 19,255,199,147, 70,149, 35,
            245, 63, 11,97,30,27,240,80,122,106,108, 78,173,182,61,130,
              88,219, 25,137, 15,175,190,241,181,239,153,232,1,177,233,14,
          51,164,200, 31,156,247,187, 89, 81,176,188,105,243,127, 17,
            202, 67, 44,235, 9,18,100,52,113,218,123, 93, 91,163,178,248
      };

    prgCodeTbl = new int[ASC_TABLE_SIZE];

    // Переносимо дані зі статичного масиву в динамічний
    for (int i = 0; i < ASC_TABLE_SIZE; i++)
    {
            prgCodeTbl[i] = rgCodeTbl[i];
    }
}

CDSSL::~CDSSL()
{
```

```
	delete [] prgSourceData;

	delete [] prgSSL;

	delete [] prgCodeTbl;
}

// XOR-Подібна операція для цілочислених операндів
inline int CDSSL::Encode(int IB, int PB)
{
	int OB;

	OB = IB - (prgCodeTbl[(unsigned char)PB]);

	if (OB < 0)
	{
	   OB += ASC_TABLE_SIZE;
	}

	return OB;
}

// Ініціалізація структури підключа
void CDSSL::InitSubkey()
{
	for (int i = 0; i < INT_LEN; i++) // Ініціалізуємо структуру
	{
		subkey.Checksums.rgChecksum[i] = 0;
	}
}

// Один крок обертання блоку вихідних даних
void CDSSL::RotateStep(int sourceDataLen)
{
	for (int i = 0; i < (sourceDataLen - 1); i++)
	{
		prgSourceData[i] = prgSourceData[i + 1];
	}
}

// Обертання блоку вихідних даних
void CDSSL::RotateSourceData(int cNumRounds, int sourceDataLen)
{
	int temp;

	// Обертаємо вихідні дані
	for (int i = 0; i < cNumRounds; i++)
	{
		// Зберігаємо нульовий елемент пароля,тому що на його місце у
функції
		// обертання будуть записані дані, що уліво зміщаються
		temp = prgSourceData[0];

		// Один крок обертання вихідних даних
		RotateStep(sourceDataLen);

		// Відновлюємо збережений елемент SSL-Гами в його кінці,
		// (він був "витиснутий" уліво й з'явився праворуч)
		prgSourceData[sourceDataLen - 1] = temp;
	}
}

// Нормування кількості раундів обертання
void CDSSL::NormcNumRounds(int &cNumRounds, int sourceDataLen)
{
	while (cNumRounds > sourceDataLen)
	{
		cNumRounds -= sourceDataLen;
	}
```

```
}

// Нормування індексу таблиці підстановок
inline void CDSSL::NormISubstTbl(int &iSubstTbl)
{
	if (iSubstTbl >= ASC_TABLE_SIZE)
	{
		iSubstTbl -= ASC_TABLE_SIZE;
	}
}

// SSL-Функція, заснована на діленні з остачею
inline int CDSSL::HF1(int k, int size)
{
	return (k % size);
}

// SSL-Функція, заснована на діленні з остачею
// (доповнення до функції HF1 для подвійного шифрування SSL)
inline int CDSSL::HF1_2(int k, int size)
{
	return 1 + (k % (size - 2));
}

// Подвійне шифрування SSL
inline int CDSSL::HFDouble(int k, int size, int numberOfTry)
{
	return (HF1(k, size) + numberOfTry * HF1_2(k, size)) % size;
}

// Нормування індексу в масиві підключа
void CDSSL::NormISubkey(int &iSubkey)
{
	iSubkey = HFDouble(iSubkey, SUBKEY_LEN, nTry1++);
}

// Нормування індексу в масиві SSL-гами
void CDSSL::NormBegCode(int &begCode)
{
	begCode = HFDouble(begCode, SSLGammaLen, nTry2++);
}

// Підготовка блоку вихідних даних до шифруванню SSL
void CDSSL::PrepareSourceData(int sourceDataLen)
{
	int i = 0, j = 0, cNumRounds, iCodeTbl;

	// Циклічне копіювання блоку вихідних даних у масив
	// prgSSL
	while (i < SSLGammaLen)
	{
		// Індекс у таблиці підстановок
		iCodeTbl = prgSourceData[j];

		// Таблична підстановка
		prgSSL[i] = prgCodeTbl[iCodeTbl];

		// Збільшуємо індекс у масиві вих. дан.
		j++;

		// Перевіряємо на закінчення вих. дан.
		if (j >= sourceDataLen)
		{
			// Обертати вих. дан. з одного елемента безглуздо
			if (sourceDataLen > 1)
			{
				// Первісна кількість зрушень вих. дан.
				cNumRounds = (prgSourceData[0] ^ i);
```

```
                        // Нормування кількості кроків обертання вих. дан.
                        NormcNumRounds(cNumRounds, sourceDataLen);

                        // Обертання вих. дан.
                        RotateSourceData(cNumRounds, sourceDataLen);
                  }
                  j = 0; // Обнуляем j (починаємо копіювати вих. дан. з
                         // prgSourceData в prgSSL) з початку
            }
            i++;
      }
}

// Обчислення підключа (використовується для кодування SSL-Гами
// prgSSL. Підключ обчислюється на основі контрольних сум гами
// prgSSL, що підлягає кодуванню)
void CDSSL::CalculateSubkey()
{
      for (int i = 0; i < SSLGammaLen; i++)
      {
            subkey.Checksums.rgChecksum[0] += (prgSSL[i] * prgSSL[i] ^ prgSSL[i]
* i * i * i);
            subkey.Checksums.rgChecksum[1] += (prgSSL[i] * prgSSL[i] * prgSSL[i]
^ prgSSL[i] + i);
            subkey.Checksums.rgChecksum[2] += (prgSSL[i] * prgSSL[i] ^
prgSSL[i]);
            subkey.Checksums.rgChecksum[3] += (prgSSL[i] * prgSSL[i] ^ prgSSL[i]
* i * i);
      }
}

// Перший раунд кодування - починаючи з BegCode і до кінця
void CDSSL::FirstCodeRound(int begCode)
{
      int IB, PB;

      for (int i = begCode; i < (SSLGammaLen - 1); i++)

      {
            IB = prgSSL[i];    // Кодуємий елемент потоку
            PB = prgSSL[i + 1]; // елемент, Що Кодує, потоку
            prgSSL[i] = Encode(IB, PB);
      }
}

// Кодування першого елемента SSL-Гами останнім - реалізуємо
// перенос ОС на початок
inline void CDSSL::MoveCode()
{
   int IB, PB;

   IB = prgSSL[SSLGammaLen - 1];
   PB = prgSSL[0];
   prgSSL[SSLGammaLen - 1] = Encode(IB, PB);
}

// Другий раунд кодування - з початку й до BegCode
void CDSSL::SecondCodeRound(int begCode)
{
      int i, IB, PB;

      // Другий раунд кодування - з початку й до BegCode
      for (i = 0; i < (begCode - 1); i++)
      {
            IB = prgSSL[i];      // Кодуємий елемент потоку
            PB = prgSSL[i + 1];  // елемент, Що Кодує, потоку
            prgSSL[i] = Encode(IB, PB);
      }
}
```

```
// Кодування SSL-гами
void CDSSL::SSLCode()
{
      int begCode = 0;
      iSubkey++;

      // Нормування індексу в масиві підключа
      NormISubkey(iSubkey);

      // Обчислюємо значення індексу початку кодування (BegCode)
      begCode += (unsigned char)(prgSSL[0] + subkey.rgubSubkey[iSubkey]);

    // Нормуємо параметр автокодування
    NormBegCode(begCode);

      // Перший раунд зв'язування - починаючи з BegCode і до кінця
    FirstCodeRound(begCode);

      // Кодування першого елемента останнім - реалізуємо
      // перенос "ОС" на початок SSL-гами
      MoveCode();

      // Другий раунд кодування - з початку й до BegCode
      SecondCodeRound(begCode);
}

// Реалізація ОС при накладенні підключа на SSL-Гаму
void CDSSL::SubkeyCode()
{
      for (int i = 0; i < SUBKEY_LEN; i++)
      {
            subkey.rgubSubkey[i] ^= (unsigned char)rgFeed[i];
      }
}

// Накладення гами підключа на SSL-Гаму
void CDSSL::ImposeSubkeyGamma()
{
      int IB, PB;

      // Буфер ОС по даним підключа з контрольних сум

      // Накладення контрольної суми
      for (int i = 0, j = 0; i < SSLGammaLen; i++, j++)
      {
            if (j >= SUBKEY_LEN)
            {
                  // Реалізація ОС - кодування підключа unSubkey.Subkey[],
                  // який складається з контрольних сум prgSSL, блоком вихідних
                  // даних
                  SubkeyCode();

                  j = 0;
            }

            // Блок кодування гами ключа за допомогою підключа

            // Кодуємий байт основного ключа
            IB = prgSSL[i];
            // байт, що кодує, з підключа
            PB = (int)subkey.rgubSubkey[j];

            // Результат кодування
            prgSSL[i] = Encode(IB, PB);

            // Формуємо буфер ОС
            rgFeed[j] = prgSSL[i];
      }
```

```
}

// Функція шифрування SSL даних
void CDSSL::SSL(char *prgbSourceData, int sourceLen)
{
    int cRounds;

    iSubkey = 0;
      nTry1 = 0;
      nTry2 = 0;

      // Переносимо вихідні дані в масив int (він і буде шифруватися SSL)
      for (int i = 0; i < sourceLen; i++)
      {
            prgSourceData[i] = (unsigned char)prgbSourceData[i];
      }

      // Ініціалізуємо структуру, що містить масив контрольних сум
      // і вхідну в об'єднання Subkey
      InitSubkey();

      // Підготовка вих. дан.: циклічне копіювання його в prgSSL
      PrepareSourceData(sourceLen);

      // Шифрування SSL блоку вихідних даних
      for (cRounds = 0; cRounds < SSLGammaLen; cRounds++)
      {
            // Обчислення підключа
            CalculateSubkey();

            // Кодування SSL-гами XOR-подібною операцією
            SSLCode();

            // Накладення гами підключа
            ImposeSubkeyGamma();

            // Обновляємо прогрес
            if (UpdateProgress != NULL)
            {
                  UpdateProgress(cRounds);
            }
      }

      // Записуємо результат в "цільовий" масив
      for (int i = 0; i < SSLGammaLen; i++)
      {
            prgbSourceData[i] = (char)prgSSL[i];
      }
}
```

Файл DSSL.h - бібліотека для файлу DSSL.cpp

```
#pragma once

#define ASC_TABLE_SIZE 256 // Довжина ASCII-Таблиці
#define SUBKEY_LEN 16      // Довжина підключа unSubkey.Subkey[]
#define MODKEY_LEN 3       // Довжина ключа для кодування модуля
#define INT_LEN 4        // Довжина в байтах змінної типу int
#define NULL 0

// Об'єднання для перетворення чотирьох контрольних сум у підключ,
// складається з 16 байт
typedef union
{
      struct TChecksums // Поле масиву з 4-ьох змінних типи int
      {
            int rgChecksum[4];
      } Checksums;

      unsigned char rgubSubkey[SUBKEY_LEN];  // ...використовується як масив
                                                  // unsigned char з
16
                                             // елементів (4*4=16)
} TSubkey;

// Клас шифрування SSL блоку даних
class CDSSL
{
public:
      CDSSL();
      CDSSL(int SSLLen);
      CDSSL(int SSLLen, void (*UpdateProgress)(int progress));
      virtual ~CDSSL();

      // Функція шифрування SSL даних
      void SSL(char *prgbSourceData, int SourceLen);

private:

      int SSLGammaLen;
      TSubkey subkey;

      int rgFeed[SUBKEY_LEN];

      int *prgSourceData;
      int *prgSSL;
      int iSubkey;
      int *prgCodeTbl;
      int nTry1;
      int nTry2;

      void Initialize(int SSLLen);
      inline int Encode(int IB, int PB);
      void InitSubkey();
      void RotateStep(int sourceDataLen);
      void RotateSourceData(int cNumRounds, int sourceDataLen);
      void NormcNumRounds(int &cNumRounds, int sourceDataLen);
      inline void NormISubstTbl(int &iSubstTbl);
    inline int HF1(int k, int size);
      inline int HF1_2(int k, int size);
      inline int HFDouble(int k, int size, int numberOfTry);
      void NormISubkey(int &iSubkey);
      void NormBegCode(int &begCode);
      void PrepareSourceData(int sourceDataLen);
      void CalculateSubkey();
      void FirstCodeRound(int begCode);
      void MoveCode();
```

```
        void SecondCodeRound(int begCode);
        void SSLCode();
        void SubkeyCode();
        void ImposeSubkeyGamma();
        void (*UpdateProgress)(int progress);
};
```