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
“____” _____ 2021 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-20М-1,4
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Загорій А.Д.
«____» _____ 2021 р.

Керівник проекту
доктор технічних наук, професор
_____ Олексій СМІРНОВ
«____» _____ 2021 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

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
« 6 » вересня 2021 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Загорію Артему Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API*

2. Керівник роботи *Смірнов Олексій Анатолійович, докт. техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 42-13 від 02.08.2021 року

3. Строк подання студентом роботи до захисту *10.12.2021 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API*

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

1. Призначення та область використання.	*7. Економічна ефективність розробленої*
2.Перегляд аналогічних існуючих систем.	*програми.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та техніки безпеки*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	
6. Наукова новизна	

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

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*3 аркуша*
Показники економічної ефективності	*1 аркуш*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В.	05.10.2021	14.11.2021
Охорона праці	Оришака О.В.	06.10.2021	16.11.2021

7. Дата видачі завдання « 6 » вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2021 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2021 р.	
3.	Розробка моделі компонента	20.10.2021 р.	
4.	Розробка структур даних	25.10.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2021 р.	
6.	Програмування алгоритмів	10.11.2021 р.	
7.	Розрахунок економічної ефективності	13.11.2021 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2021 р.	
9.	Оформлення ПЗ	17.11.2021 р.	
10.	Попередній захист роботи	10.12.2021 р.	

Дата видачі завдання
« 6 » вересня 2021 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2021 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Загорій А.Д. Дослідження та програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Метою розробки є дослідження та програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Об'єктом дослідження є процес кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Предметом дослідження є методи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Методи дослідження базуються на методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

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 XP/Vista/7/8/10.

Програму розроблено в середовищі Delphi.

Ключові слова: комп'ютерна інженерія, захисту доступу, Crypto API

ABSTRACT

Zahorii A.D. Research and software implementation of a cybersecurity system for the protection of software arrays based on the use of the Crypto API library. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2021

In this final qualification work for the second (master's) level of higher education, software has been developed that is designed for a cybersecurity system to protect software arrays based on the use of the Crypto API library.

The purpose of the development is to research and software implementation of a cybersecurity system for the protection of software arrays based on the use of the Crypto API library.

The object of research is the process of cybersecurity to protect software arrays based on the use of the Crypto API library.

The subject of research is cybersecurity methods for protecting software arrays based on the use of the Crypto API library.

Research methods are based on methods of information protection, methods of mathematical statistics, methods of software development.

The result is a software implementation of a cybersecurity system to protect software arrays based on the use of the Crypto API library.

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.

Developed user-friendly interface. Instructions for working with software are given.

The program can be used on an IBM PC with Windows XP / Vista / 7/8/10.

The program is developed in the Delphi environment.

Keywords: computer engineering, access protection, Crypto API

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 3
ВСТУП 4
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 7
1.1 Призначення системи 7
1.2 Область застосування 10
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 11
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти 11
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування 24
2.3 Розгорнута постановка завдання 32
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 33
3.1 Опис функціонування системи 33
3.2 Розробка структурної схеми 39
3.3 Розробка функціональної схеми 41
3.4 Розробка діаграми процесів 46
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ 48
4.1 Розробка блок-схем та опис алгоритмів функціонування системи 48
4.2 Захист розробленого програмного забезпечення 74
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ..... 76
6 НАУКОВА НОВИЗНА 81

7 ЕКОНОМІЧНА ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ 82
7.1 Техніко економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. 82
7.2 Розрахунок трудомісткості розробки програмної продукції 84
7.3 Визначення чисельності виконавців і планового фонду зарплати 86
7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника 91
7.5 Визначення собівартості розробки та ціни програмної продукції. 95
7.6 Визначення об'єму капітальних вкладень та експлуатаційних витрат у споживача програмної продукції. 98
7.7 Визначення експлуатаційних витрат 98
7.8 Визначення економічної ефективності програмної продукції. 100
7.9 Висновок. 102
8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ 103
8.1 Вступ 103
8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером 104
8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста . 105
8.4 Розробка заходів з умов поліпшення охорони праці 109
8.5 Розрахункова частина 109
8.6 Висновки до розділу 112
9 ОСНОВНІ ВИСНОВКИ 113
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 115

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

БД	- база даних
БСА	- блок-схема алгоритму
ВКРМ	- випускна кваліфікаційна робота магістра
ДСТУ	- Державні стандарти України
ЕЦП	- електронно-цифровий підпис
ЕК	- експорт ключів
ЗПД	- збереження підписаних даних
індекс.	- індексований
ІС	- інформаційна система
ІП	- інформаційні потоки
ІМ	- інформаційний масив
КМ	- комп’ютерна мережа
КОК	- ключ обміну ключами
КБ	- кібербезпека
КТЗ	- комплекс технічних засобів
МК	- мікроконтролер
НСВ	- несанкціоноване використання
НСД	- несанкціонований доступ
ОС	- операційна система
ПЗ	- програмне забезпечення
ПК	- персональний комп’ютер
ПП	- програмний продукт
ППЗ	- прикладне програмне забезпечення
СЗ	- система захисту
ТЗ	- технічне завдання
ЦУКС	- центр управління комп’ютерними системами

ВСТУП

Актуальність теми. Інформація про уразливість може бути використана зловмисниками при написанні вірусів. Наприклад, один з найперших відомих мережевих черв'яків (вірус Morіса) в 1988 році використовував такі уразливості як переповнення буфера в Unix-демоні finger для поширення між машинами. Тоді кількість заражених машин склало близько 6 тисяч, а економічний збиток за даними рахункової палати США склав від 10 до 100 млн доларів.

З 2016 комп'ютерні віруси завдали світовій економіці збитків в 450 млрд. доларів. У 2017 збиток від вірусу WannaCry оцінили в 1 млрд. доларів. Випадки зараження були зафіксовані щонайменше в 150 країнах. Вірус застосовував експлойт EternalBlue, що використовує уразливість в протоколі SMB, пов'язану з переповненням буфера. У 2017 році відбулась масштабна хакерська атака з боку Росії проти України та ще 59 країн. Причиною стала компрометація системи оновлення програми M.E.Doc встановленням бекдору та вірусу NotPetya. Збитки підприємств по всьому світу склали 8 млрд. доларів.

Сучасний комп’ютерний світ представляє собою різнобічну і досить складну сукупність обчислювальних засобів, систем обробки інформації, комунікаційних технологій, програмного забезпечення (ПЗ) та високоефективних засобів його проектування. Вся ця багатогранна та взаємопов’язана метасистема вирішує величезне коло проблем в різноманітних сферах людської діяльності: від простого рішення шкільних задач на домашньому персональному комп’ютері (ПК) до управління складними технологічними процесами та космічними технологіями. Тому природним шляхом розвитком інформаційних технологій на даний час є принциповий перехід від відкритого використання інформаційних масивів та КС до захищеного їх використання.

Як показав проведений в період переддипломної практики аналіз провідних комп’ютерних фірм світової спільності, вбудовані в сучасні ОС механізми захисту не забезпечують гарантованого захисту інформації від несанкціонованого доступу. На жаль, це також є типовою тенденцією, отже – під

сумнівом обгрунтованність концепції вбудованої системи захисту в сучасних ОС, а значить виникає необхідність в використанні додаткового захисту інформації.

Враховуючи вищеозначене та той фактор, що світова спільнота вступила в цифрове століття, коли на зміну паперовим носіям інформації прийшли електронні, а особисті контакти все частіше замінюються листуванням по e-mail з засвідченням необхідних документів цифровим електронним підписом, криптографічні алгоритми стають звичним та вкрай необхідним інструментом. Проте ніякий, навіть найкраще реалізований механізм захисту, не зможе забезпечити якісного захисту комп'ютерної інформації в цілому. Захист інформації потребує комплексування різнорідних механізмів в єдину систему. Таким чином, при розробці ВКРМ основним об'єктом нашої уваги будуть вимоги до заданих якостей захищеності, які будуть забезпечуватись сукупністю реалізованих захисних механізмів: блочного шифрування, електронний цифровий підпис зашифрованих файлів, електронний цифровий підпис файлів.

На ринку програмних продуктів (ПП) пропонується досить великий діапазон систем аналогічного спрямування та класу. Але більшість з них на даний час вже зламані та й коштують такі системи досить дорого, що, враховуючи економічний стан нашої країни, робить ці досить привабливі системи недосяжними для багатьох користувачів. Таким чином, розробка комплексних систем захисту на даний час є питанням дійсно актуальним та перспективним.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

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

– Огляд існуючих систем кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

– Дослідження системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

– Програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Об'єктом дослідження є процес кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Предметом дослідження є методи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Методи дослідження базуються на методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

– Розроблено вітчизняний продукт кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

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

Робота апробована на LV Науково-технічна конференція здобувачів вищої освіти «Наука – виробництву», 2021, основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №12.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Згідно технічного завдання (ТЗ) метою виконання ВКРМ є розробка ПЗ системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API. Визначимо сутність поставленої задачі, призначення системи та область її використання.

Достатньо важко дати точну характеристику означення «захист», оскільки воно дуже широко трактується і має на увазі всі аспекти інформаційної безпеки. Захист – це сукупність дій, направлених на протидію спробам зловмисників з джерел зовнішніх та внутрішніх загроз, направлених на злом наявних систем захисту з метою одержання НСД до електронних та програмних ресурсів для несанкціонованого їх використання.

Джерелами зовнішніх загроз є: діяльність конкурентів з перехоплення важливої інформації; навмисні дії з руйнування, знищення або модифікації інформації; стихійні лиха та катастрофи, аварії, екстремальні ситуації.

До джерел внутрішніх загроз відносяться: відсутність координації діяльності підрозділів підприємства у сфері захисту інформації; навмисні дії персоналу по знищенню або модифікації інформації; ненавмисні помилки персоналу, відмови технічних засобів і збої в інформаційних системах; порушення встановлених регламентів збору, накопичення, зберігання, обробки, перетворення, відображення та передачі інформації.

Отже, система, що підлягає розробці, в процесі експлуатації повинна забезпечити виконання наступних функцій:

- Захист внутрішніх сегментів мережі від несанкціонованого доступу з боку користувачів мереж загального користування.

- Криптографічний захист даних, що передаються по каналах зв'язку мереж загального користування між складовими частинами.

- Автоматизована система безпеки повинна працювати на рівні ядра автоматизованої системи мережі таким чином, щоб жодна значуща дія, пов’язана з документообігом в рамках системи, не відбувалася без участі системи безпеки документообігу.

– Схема безпеки буде відділеною від засобів безпеки самої операційної системи, на якій буде реалізована автоматизована система безпеки мережі (КМ), у тому розумінні, що збій або уразливість системи безпеки КМ не повинні впливати на роботу системи, що підлягає розробці. Тому вона повинна забезпечувати замкнуте збереження й передачу даних, пов'язаних з автоматизованою системою КМ таким чином, щоб: неможливо було одержати логічний доступ до зазначених даних поза рамками її роботи; будь-які переміщення даних в системі КМ відбувались під її контролем та з використанням її механізмів захисту.

До погроз безпеки ЕДО слід віднести наступні чинники: злом конфіденційних даних, загублення або знищення даннях, несанкціонована зміна даних, відмова одного з респондентів від поставленого під документом власного ЕЦП. Це дуже вагомий список погроз, тому необхідно визначитись зі стратегією безпеки, реалізацію якої в певній мірі має забезпечити система. До неї слід віднести наступні аспекти: використання методу автентичності користувача електронним цифровим підписом (ЕЦП); шифрування документів блочним шифром на основі використання механізму CryptoAPI ОС Windows; поєднання першого та другого аспектів.

Система орієнтована на локальну мережу організації, яка має декілька територіально віддалених філіалів та використання незахищених ліній глобальної мережі Internet для роботи з іншими респондентами та респондентів з локальною мережею з глобальної мережі.

При розробці ПЗ, згідно ТЗ буде використано криптографічну підсистему ОС Windows, зокрема бібліотеки advapi32.dll, crypt32.dll. Криптографія може служити не тільки для кодування-декодування інформації. Для створення довірчих взаємин необхідна автентифікація. Адже можна довіряти тільки тому і тільки тоді, коли ви абсолютно точно знаєш, з ким маєш справу. Щоб зробити електронні документи дійсними з юридичної точки зору, необхідно передбачити засоби автентифікації автора того чи іншого документа.

Цифрові підписи засновані на технологіях відкритого ключа, таких як RSA, але реалізуються трохи відмінним від стандартного кодування способом. Замість шифрування повідомлення відкритим ключем одержувача, хеш-повідомлення кодується особистим ключем і потім розшифровується за допомогою відкритого ключа відправника, відомого одержувачу. При цьому, будь-який користувач може декодувати хеш-повідомлення, оскільки відкритий ключ (як і припускає його назва) доступний і його можна знайти на сервері каталогів відкритих ключів.

Враховуючи вищеозначене, можемо більш детально означити функції, виконання яких має забезпечити система (виходячи з безпосереднього її призначення):

– забезпечення конфіденційності ЕДО;

– забезпечення відкритості і гласності у прийнятті рішень в межах респондентів локальної мережі;

– забезпечення переходу до більш швидкого, юридично діючого ЕДО;

– забезпечення економії ресурсів за рахунок використання оперативного архіву;

– спрощення процедури економічної діяльності організації: укладання договорів, сертифікації товарів, сплати податків, здійснення експортно-імпортних, митних і транспортних операцій, тощо;

– удосконалення технології документообігу.

Розроблене ПЗ повинне мати високу ступінь мобільності та адаптивності, не бути жорстоко прив'язаним до певного типу ПК і його технічних характеристик і забезпечувати виконання означених ТЗ задач.

1.1 Область застосування

Враховуючи, що сучасний користувач потребує цілеспрямованого та надійного програмного забезпечення з високим рівнем захисту від злому, область застосування системи захисту на основі використання функцій CryptoAPI, яка буде розроблена в процесі виконання ВКРМ, можна охарактеризувати одним містким словом – всюди: установи, організації, підприємства будь-якої форми власності та сфері діяльності, які використовують локальні мережі та незахищені канали глобальних мереж для організації ЕДО, приватний документообіг фізичних осіб, тощо.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Предметом розробки ВКРМ є програмне забезпечення системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки CryptoAPI ОС Windows.

В даний час існують різноманітні технології розробки безпечного програмного забезпечення. Але є набір принципів, що враховуються при будь-якому підході: працездатність і корисність; безпека – можливість захисту від зовнішніх загроз, атак і збереження працездатності після їх виявлення і усунення; надійність – передбачувана, коректна і безвідмовна роботу в разі некоректних вихідних даних; конфіденційність – забезпечення безпечної та коректної роботи з конфіденційною інформацією; чітка організація супроводу програми, контроль прозорості, законності, правильності роботи користувача.

Система, що підлягає розробці, повинна забезпечити виконання наступних функцій: захист електронного документообігу від НСД, модифікації та спроб відмови одного з респондентів від ЕЦП (тобто – власного авторства) в мережі. Окрім цього система, що підлягає розробці, повинна бути інформативною, тобто мати зручний інтерфейс користувача з виведенням результатів роботи та верифікації на екран монітору, друкуючий пристрій або машинний носій інформації.

Для більш детального визначення функцій та структури майбутньої системи, визначення концепції, методики та принципів її побудови, необхідно розглянути існуючі системи – аналоги та провести їх аналіз.

Говорити про те, що інформаційна безпека (ІБ) стала частиною корпоративної культури, у нас в країні можна з великою натяжкою. Необхідність забезпечення ІБ усвідомили тільки великі компанії. Та і вони до недавнього часу проблеми безпеки сприймали виключно як технічні, пов'язані з впровадженням міжмережевих екранів, антивірусного програмного забезпечення, засобів виявлення вторгнень в приватні мережі.

Насправді, по рекомендаціях дослідницьких фірм, від 60 до 80% усіх зусиль із забезпечення безпеки слід направляти на розробку політики безпеки і супутніх їй документів. Чому? Тому, що політика безпеки є найдешевшим і одночасно найефективнішим засобом забезпечення інформаційної безпеки (звичайно, якщо її наслідувати). Крім того, якщо політика сформульована, то вона є і керівництвом по розвитку і вдосконаленню систем захисту.

Конкретні продукти і рішення по інформаційній безпеці удосконалюються рік від року, інтегруються між собою. Усе це відбувається так нестримно, що здається, ніби недалекий той день, коли хто-небудь запропонує універсальний продукт для захисту будь-яких інформаційних систем усіма доступними засобами. Розглянемо найбільш поширені на ринку програмних продуктів системи та програмно-апаратні комплекси аналогічного класу та спрямування.

Криптографічний комплекс "Шифратор IP-потоків" (КК "ШПИЛЬКА"), який має у своєму складі:

– апаратно-програмні комплекси (АПК "ШПИЛЬКА"), які є шифраторами IP – потоків;

– клієнтські програмні комплекси (ПК "Ігла-П"), призначені для роботи на ПК в середовищі Windows з використанням стека мережевих протоколів TCP/IP.

– центр управління ключовою системою.

АПК "Шпилька" будується на базі ПК, містить плату "Кулон -1М", забезпечує захист від НСД при завантаженні системи і для отримання випадкових послідовностей з сертифікованого фізичного датчика, необхідних для реалізації процедур шифрування і автентифікації.

Продуктивність АПК "Шпилька" складає близько 8 Мбіт/с (3000 пакетів/с) при обміні інформацією по мережі "Ethernet/вхід" "Ethernet/вихід " і близько 2 Мбіт/с (750 пакетів/с) при обміні інформацією по інтерфейсу V.35 і протоколу Frame Relay.

Програмний комплекс "Ігла-П" вбудовується в мережеву архітектуру ОС Windows на рівні драйвера мережевої оболонки ndis.sys і перехоплює обмін мережевою інформацією між мережевими протоколами і адаптерами.

ЦУКС створюється на базі ПК і включає:

– автоматизоване робоче місце управління ключовою системою, працююче в середовищі ОС Windows;

– модуль серверної частини;

– серверну програму перегляду протоколів роботи КК "Шпилька".

У КК "ШИП" в якості протоколу передачі даних автентифікації і узгодження сеансового ключа використовується протокол SKIP, що має офіційний статус проекту стандарту Інтернет). Цей протокол був запропонований фірмою Sun і секцією робочої групи інженерної підтримки Інтернет (IETF), працюючій над проблемами забезпечення безпеці стека протоколів TCP/IP (IPSEC).

Протокол SKIP відноситься до родини протоколів TCP/IP і реалізується в нижній частині мережевого рівня. Використання в територіальній мережі протоколу TCP/IP дозволяє шифрувати інформацію на пакетному рівні. Працюючи на мережевому рівні моделі взаємодії відкритих систем, облаштування пакетного шифрування в пункті відправлення шифрує увесь пакет, включаючи заголовок IP, і приписує йому новий заголовок, який є просто IP – адресою облаштування захисту інформації в пункті призначення. Зашифроване повідомлення може передаватися через мережу на основі IP – маршрутизаторів. У пункті призначення пакет розшифровується. Тимчасовий заголовок видаляється, а початковий IP – заголовок знову стає доступним, що і дозволяє довести пакет до адресата.

При використанні протоколу SKIP для організації захищеної взаємодії між абонентами не потрібно ніякого додаткового інформаційного обміну і передачі по каналах зв'язку відкритої інформації. Інформаційна надмірність, що вноситься протоколом, не перевищує 1-2% і, таким чином, не вимагає високих витрат обчислювальних ресурсів.

Криптографічний комплекс "Шифратор – IP – потоків" (КК "ШПИЛЬКА"), що має у своєму складі:

- апаратно-програмні комплекси (АПК "ШПИЛЬКА"), що є шифраторами IP – потоків;

- клієнтські програмні комплекси (ПК "Ігла-П"), призначені для роботи на ПК в середовищі ОС Windows NT з використанням стека мережевих протоколів TCP/IP.

- центр управління ключовою системою.

Ігла-П" вбудовується в мережеву архітектуру ОС Windows на рівні драйвера мережевої оболонки ndis.sys і перехоплює обмін мережевою інформацією міжмережевими протоколами і адаптерами.

ЦУКС створюється на базі ПК і включає:

- автоматизоване робоче місце управління ключовою системою, працююче в робочому середовищі ОС Windows;

- модуль серверної частини;

- серверну програму перегляду протоколів роботи КК "Шпилька".

У системі "Коміта – Кур'єр (tm)" із СКЗІ "Верба-О" для захисту інформації забезпечується виконання наступних заходів :

- усі юридично значимі документи завіряються електронними підписами відповідальних осіб, що дозволяє виключити можливість фальсифікації переданих документів і гарантує їх цілісність і достовірність;

– усі документи обов'язково шифруються перед передачею по каналах зв'язку;

– у повному об'ємі використовуються засоби адміністрування Novell NetWare для обмеженню прав доступу до інформації і NetWare Connect по доступу до комунікацій;

– застосовується оригінальна технологія обмеження доступу до конфіденційної інформації, заснована на застосуванні облаштувань Touch Memory, в які записуються паролі для доступу до засобів комунікацій і закриті ключі користувачів для шифрування ЕЦП;

– ведення електронних журналів реєстрації, що містять повну інформацію про усі створені, передані і прийняті документи у операціоністів, банку, на комунікаційному сервері і у клієнтів;

– архівація усієї інформації в зашифрованому вигляді, що виключає несанкціонований доступ;

– використання внутрішнього оригінального формату для зберігання документів з метою їх захисту від дії комп'ютерних вірусів.

Розглянемо декілька автоматизованих систем захисту електронного документообігу (на основі використання механізму CryptoAPI ОС Windows) лідерів українського ринку програмних продуктів, які впроваджують системи безпечного електронного документообігу.

Компанія «БЕСТ ЗВІТ» є лідером серед українських розробників програмного забезпечення для ведення безпечного електронного документообігу з накладенням ЕЦП. «БЕСТ ЗВІТ» розробляє і технічно підтримує програмне забезпечення, призначене для підготовки будь-якої звітності в електронному вигляді і подачі її в різні контролюючі органи по електронній пошті. Програмні комплекси «БЕСТ ЗВІТ ПЛЮС» і «БЕСТ ЗВІТ КОРПОРАЦІЯ» автоматизують процеси збирання, обробки і консолідації звітності в електронному вигляді. Компанія має великий досвід успішного впровадження своєї системи в ДНАУ, ДСА, Мінфін. Щодня з системою працюють сотні тисяч підприємств та організацій України.

Компанія «Інтелект-Сервіс». Компанія Інтелект-Сервіс, створена в 1994

році, працює на ринках України, Росії, Білорусі, Молдови. За підсумками 2006 року Компанія визнана одним з лідерів Національного бізнес-рейтинга.

Компанія випускає **програмний продукт ІС-ПРО**, який призначений для управління підприємствами різного профілю і форм власності і орієнтована на підприємства виробничої сфери, сфери послуг, торгові компанії, бюджетні організації. Система ІС-ПРО має модульну архітектуру. У її складі виділяються підсистеми, які забезпечують автоматизацію основних бізнес-процесів підприємства. Кожна підсистема володіє відносною самостійністю і може експлуатуватися як у складі комплексу, так і окремо.

Спеціалізований центр сертифікації ключів (СЦСК) ТОВ "НПФ "УНІС".

УНІС реалізує проекти по комплексній автоматизації підприємств і відомств, виконуючи при цьому весь спектр робіт: аналіз предметної області, проектування, розробка, впровадження і супровід інтегрованих автоматизованих систем. Спеціалізація центру сертифікації ключів компанії "УНІС" полягає в тому, що додатково до функцій звичайного центру сертифікації ключів, центр включає підсистеми, які дозволяють ефективно реалізовувати його взаємодію з центрами обробки даних і системами електронного документообігу.

Основним недоліком систем, які ми коротко охарактеризували, є обов’язкове налаштування їх фахівцем та подальше супроводження під час промислової експлуатації.

Природним шляхом розвитку інформаційних технологій на сьогодення є перехід від відкритості до захищеності при побудові інформаційних систем (ІС). На даний час більшість програмних продуктів, що використовуються для побудови ІС, утримують вбудовані механізми захисту. Це відноситься не лише до ОС, але й до СУБД та інших додатків, що свідчить про зростаючу тенденцію посилення ролі механізмів захисту в сучасних інформаційних і мережевих технологіях.

Дану тенденцію найбільш наглядно ілюструє розвиток сучасних операційних систем: ОС Windows, UNIX, LINEX, MSDOS, тощо. Тому доцільно

розглянути та проаналізувати, з ціллю виявлення позитивних якостей та недоліків, їх вбудовані механізми захисту.

Зразу слід відзначити, що аналізувати виконання сучасними ОС вимог, що задаються для класу захищеності автоматизованих систем 1В (захист секретної інформації), не має сенсу. Для більшості ОС або повністю не реалізується основний для даних додатків мандатний механізм керування доступом до ресурсів, або повністю не виконується його важлива вимога: “Повинне здійснюватись керування потоками інформації за допомогою позначок конфіденційності. При цьому, рівень конфіденційності накопичувача повинен бути не нижчим рівня конфіденційності записуваної на нього інформації”. Виходячи з цього, робимо висновок, що аналіз існуючих ОС будемо проводити лише на предмет їх відповідності класу автоматизованих систем 1Г (захист конфіденційності інформації). Окрім цього, вибраний напрямок аналізу відповідає темі БДР, яка підлягає розробці.

В якості альтернативних реалізацій ОС розглянемо родини UNIX і Windows стосовно можливості реалізації ними концепції захисту інформації від НСД та НСВ. Розгляд основних механізмів захисту, що реалізовані в ОС родини Windows, та аналіз захищеності ОС родини Windows показав наступне. Слід відзначити, що в цій ОС ряд об’єктів доступу (пр истрої, реєстр ОС, тощо) не є об’єктами файлової системи. Тому виникає питання, яким чином трактувати вимогу “Система захисту повинна контролювати доступ пойменованих суб’єктів” (користувачів) до пойменованих об’єктів (файлів, програм, полів, тощо). Тобто, не зрозуміло, чи є об’єктами доступу, до яких, виконуючи формальні вимоги, необхідно розмежувати доступ користувачів.

На відміну від родини ОС UNIX, де всі задачі розмежувальної пол ітики доступу до ресурсів вирішуються засобами керування доступом до об’єктів файлової системи, доступ даних в ОС родини Windows розмеж овується власним механізмом доступу до кожного ресурсу. Іншими словами, при розгляді механізмів захисту ОС Windows виникає задача визначення і задання вимог до

повноти розмежування (це визначається тим, що вважати за об’єкт доступу).

Також, як і для ОС UNIX, в ОС Windows основними механізмами з ахисту є:

– ідентифікація та автентифікація користувача при вході в систему;

– розмежування прав доступу до ресурсів, в основі яких лежить реаліз ація дискреційної моделі доступу (окремо до об’єктів файлової системи, до пристроїв, до реєстру ОС, до принтерів, тощо);

– аудит, тобто реєстрація подій.

В цій ОС явно виділені в кращу сторону можл ивості розмежувань прав доступу до файлових об’єктів (для NTFS) – суттєво розширені атрибути доступу, які встановлюються на різні ієрархічні об’єкти файлової системи (логічні диски, каталоги, файли). Наприклад, атрибут “виконання” може встановлюватися і на каталог, тоді він успадковується відповідними файлами (на відміну від ОС UNIX).

При цьому суттєво обмежені можливості управління доступом до інших ресурсів, що підлягають захисту, наприклад – до пристроїв введення (відсутній атрибут “виконання”, тобто, неможливо заборонити запуск несанкціонованої програми з пристрою введення).

Разом з тим ОС Windows має ряд принципових недоліків захисних механізмів, напрямки пов’язаних з можливістю НСД до інформації. При цьому, на відміну від ОС UNIX, в ОС Windows можлива в загальному в ипадку реалізація централізованої схеми адміністрування механізмів захисту чи відповідних формалізованих вимог. Це пов’язано з тим, що в ОС Windows прийнятна інша концепція реалізації розмежувальної політики доступу до ресурсів (для NTFS).

В рамках цієї концепції розмежування для файлу більш пріоритетне, чим для каталогу, а в загальному випадку – розмежування для файлу, що включається, більш пріоритетне, чим для файлу, що включає. Це призводить до того, що користувач, який створює файл і є його “власником”, може призначити будь-які атрибути доступу до такого файлу (тобто, дозволити до нього доступ

будь-якому іншому користувачу). При цьому звернутись до цього файлу може користувач (якому призначить права доступу “власник”) незалежно від встановлених адміністратором атрибутів доступу на каталог, в якому користувач має намір створити файл. Ця проблема безпосередньо пов’язана з реалізованою в ОС Windows концепцією захисту інформації.

Далі, в ОС Windows не в повному обсязі реалізована дискретна модель доступу, частково не можуть розмежовуватися права доступу для користувача “Система”. В ОС присутні не тільки користувачські, але й системні процеси, які запускаються безпосередньо системою. При цьому, доступ системних процесів не може бути розмежованим, відповідно – всі системні процеси, що запускаються, мають необмежений доступ до захищених ресурсів. Саме з цим недоліком системи пов’язана дуже велика кількість атак (наприклад – запуск власного процесу, як системного).

В ОС Windows неможливо також в загальному випадку забезпечити замкнутість (цілісність) програмного середовища. Це пов’язано зовсім з іншими проблемами, чим в ОС UNIX, в яких неможливо встановити атрибут “виконання” на каталог. Розглянемо більш детально це питання. Механізм замкнутості програмного середовища в загальному випадку може бути забезпечений:

– заданням списку дозволених на запуск процесів з наданням можливості користувачам здійснювати запуск процесів тільки з цього списку. При цьому, процеси задаються напівшляховими іменами, причому засобами розмежування доступу забезпечується неможливість їх модернізації користувачем. Даний підхід просто не реалізується вбудованими в ОС механізмами;

– дозволом запуску користувачами програм тільки із заданих каталогів при неможливості модернізувати ці каталоги. Однією з умов коректної реалізації даного підходу є заборона користувачам здійснювати запуск програм не інакше, чим із відповідних каталогів. Некоректність реалізації ОС Windows даного підходу пов’язана з неможливістю встановити атрибут “виконання” на пристрої введення (дисковод/CD-ROM). В зв’язку з цим, при розмежуванні прав доступу,

користувач може запустити несанкціоновану програму з дискети (або з диску CD-ROM). А це дуже розповсюджена атака на ОС даної родини. Тому слід відзначити, що з точки зору забезпеченості замкнутості програмного середовища (тобто, реалізації механізму, який забезпечує запуск користувачами тільки санкціонованих програм/процесів) дії користувача по запуску програм/процесів можуть бути як явними, так і прихованими.

Явні дії визначають запуск процесів/файлів, які однозначно ідентифікуються своїми іменами. Приховані дії дозволяють здійснювати вбудовані в додатки інтерпретатори команд. Прикладом цього можуть бути офісні додатки корпоративної мережі, а прихованими діями – запуск макросів.

В даному випадку ідентифікації підлягає лише власний додаток, наприклад, процес Windows.exe. При цьому він може, окрім своїх регламентованих дій, виконувати ті приховані дії, які задаються макросом, що зберігається в документі, який відкривається. Теж саме можна віднести і до віртуальної машини, яка утримує вбудований інтерпретатор команд, що не забезпечує виконання вимог по ідентифікації програм та особи користувачів, які відправляють документацію в електронному вигляді.

Окрім цього, до недоліків ОС Windows слід віднести наступне:

– неможливість вбудованими механізмами захисту гарантовано видалити залишкову та надлишкову інформацію (відсутні відповідні функції);

– відсутність контролю цілісності файлової системи (вбудовані механізмами захисту контролюють тільки власні системні файли);

– відсутність контролю цілісності програм/додатків перед їх запуском;

– відсутність моніторингу роботи вбудованих механізмів захисту;

– відсутність реєстрації виведення документів на “жорсткі копії”;

– відсутність в повному обсязі механізму керування доступом до хостів.

Окрім цього, існує ще один суттєвий момент: в ОС Windows фільтрації підлягає тільки вхідний доступ до розподіленого ресурсу, а запит доступу на ПК, з якого він здійснений, фільтрації не підлягає. Це принципово, оскільки не

підлягають фільтрації додатки, якими користувач здійснює доступ до розподіленого ресурсу. Це дозволяє успішно виконати атаки на протокол NETBIOS.

Із проведеного аналізу найбільш розповсюдженої ОС Windows, робимо висновок, що багато механізмів, необхідних з точки зору виконання формалізованих вимог щодо захисту інформації/ПЗ, ОС Windows не реалізує зовсім або реалізує частково, але надає користувачу досить потужні механізми захисту, які можна покласти в основу розробки дійсно діє спроможної системи захисту інформації (НАПРИКЛАД CryptoAPI).

Захист ОС UNIX в загальному випадку базується на трьох о сновних механізмах:

– ідентифікації і автентифікації користувача при вході в систему;

– розмежування прав доступу до файлової системи, в основі якого лежить реалізація дискретної моделі доступу;

– аудит, тобто реєстрація подій.

При цьому відзначимо, що для відмінних клонів ОС сімейства UNIX можливості механізмів захисту будуть незначно відмінними, однак будемо розглядати ОС UNIX в загальному випадку, без урахування деяких незначних особливостей окремих ОС цієї родини.

Побудова файлової системи та розмежування доступу до файлових об’єктів має особливості, що присутні даній родині ОС. Розглянемо коротко ці особливості. Всі дискові накопичувачі (томи) об’єднуються в єдину віртуальну файлову систему шляхом операції монтирування тому. При цьому, зміст тому проектується на обраний каталог системи. Елементами файлової системи є також всі пристрої, що підключені до ПК, який підлягає захисту. Тому розмежування доступу до них здійснюється через файлову систему.

Кожний файловий об’єкт має індексний дескриптор (описувач), в якому серед інших даних зберігається інформація про розмежування доступу до даного файлового об’єкту. Права доступу розподілені на три категорії: доступ для

власника, доступ для групи і доступ для інших користувачів. В кожній категорії визначається права на читання, запис і виконання (в випадку каталогу – перегляд).

Користувач має унікальний символьний ідентифікатор (ім'я) і цифровий ідентифікатор (UID). Символьний ідентифікатор пред'являється користувачем при вході в систему, цифровий використовується ОС для визначення прав користувача в системі (доступ до файлів, тощо).

Таким чином, розглянута ОС UNIX є досить привабливою для користувача і в аспекті використання присутніх їй механізмів захисту. Але наряду з виявленими перевагами в процесі проведеного аналізу були визначені наступні недоліки.

Для початку відзначимо, що в ОС UNIX, внаслідок використаної концепції адміністрування (не централізованої), не можливо забезпечити замкнутість (чи цілісність) програмного середовища. Це пов'язано з неможливістю встановлення атрибуту "виконання" на каталог (для каталогу даний атрибут обмежує можливість "огляду" змісту каталогу). Тому при розмежуванні адміністратором доступу користувачів до каталогів, користувач, як "власник" створюваного ним файлу, може занести в свій каталог виконуваний файл і, як його "власник" встановити на файл атрибут "виконання", після чого запустити записану їм програму. Ця проблема безпосередньо пов'язана з реалізуємою в ОС концепцією захисту інформації.

Не в повному обсязі реалізується дискреційна модель доступу, зокрема, не можуть розмежовуватися права доступу для користувачів "root" (UID=0). Тобто, даний суб'єкт доступу виключається із схеми керування доступом до ресурсів. Відповідно всі запущені ним процеси мають необмежений доступ до ресурсів, що підлягають захисту. Саме з цим недоліком системи захисту пов'язана велика кількість атак, направлених на:

– несанкціоноване одержання права root;

– запуск з правами root власного виконуваного файлу (локально чи віддалено впровадженого). При цьому несанкціонована програма одержує повний доступ до ресурсів, що захищаються механізмами захисту ОС.

Окрім цього, в ОС UNIX неможливо вбудованими засобами гарантовано видаляти надлишкову інформацію. Для цього в системі абсолютно відсутні відповідні механізми.

Необхідно також відзначити, що дані ОС не мають можливості контролю цілісності файлових систем, тобто, не утримують відповідних вбудованих засобів. В кращому випадку додатковими утилітами може бути реалізований контроль конфігураційних файлів ОС по розкладу в той час, як самою ва жливою можливістю даного механізму можна вважати контроль цілісності програм (додатків) перед їх запуском, контроль файлів даних користувача, тощо. Моніторинг роботи самих механізмів захисту в системі відсутній.

Що торкається реєстрації (аудиту), то в ОС UNIX не забезп ечується реєстрація видачі документів на “жорстоку копію”, а також інші вимоги до реєстрації подій.

Недоліком також можна визначити і відсутність керування доступом до хостів (розподілених пакетних фільтрів).

Таким чином, із проведеного аналізу робимо наступні висновки:

– в ОС UNIX більшість механізмів, необхідних з точки зору виконання формалізованих вимог, не реалізовані взагалі, або реалізуються частково;

– відсутність моніторингу роботи механізмів захисту створює надзвичайно сприятливі можливості для атак зловмисників з ціллю їх злому.

Таким чином, на основі проведеного аналізу ОС родини UNIX та ОС родини Windows можна зробити наступні висновки:

– більшість сучасних ОС, що є універсальними, не виконують в повн ому обсязі вимог до захисту автоматизованих систем та інформації по класу ІГ. Це означає, що вони не можуть без використання додаткових засобів захисту забезпечити навіть захист конфідиційності інформації;

– існуюча в сучасних ОС концепція захисту побудована на реалізації розподіленої схеми адміністрування механізмів захисту, що не забезпечує виконання формалізованих вимог до основних механізмів захисту;

– для більшості сучасних ОС ха рактерно наступне: повна відсутність мандатного механізму керування доступом до ресурсів, а це означає, що захист секретної інформації (клас 1В) при використанні тільки захисних механізмів ОС неможливий взагалі.

До недоліків слід віднести і наступне:

– розглянуті системи є комплексними, виконуючи і автоматизовані задачі, які не відносяться до сфери захисту документообігу, і тому коштують дорого. Але ж багато користувачів хочуть придбати та експлуатувати лише окремі модулі цих систем, які окремо розробниками цих систем не тиражуються;

– розглянуті системи не можуть бути інтегровані без підтримки розробника (за що також потрібно платити окремо) до складу вже наявних на об'єкті господарської діяльності автоматизованих та автоматичних систем більш високого рівня.

Тому доцільно провести розробку власної системи, яка буде позбавлена вищеозначених недоліків та утримувати виявлені в процесі аналізу позитивні якості.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

При визначені шляхів реалізації задачі захисту інформації з використанням комплексування різнорідних механізмів в єдину систему, було проведено аналіз вирішення конкретної задачі забезпечення безпеки всієї системи в цілому (реалізованої сукупністю механізмів захисту документообігу та програмних масивів) з урахуванням способів можливого подолання визначених нами механізмів в сукупності та кожного окремо (наприклад, з використанням

програм-сніферів, програм підбирання паролів, програм-дизасемблерів, тощо). Виходячи з цього, при визначенні якісних характеристик системи було вирішено використати якісні, тобто, визначені погрози (із числа існуючих згідно даних статистики та потенційно можливих), яким необхідно протидіяти.

Інша сторона проблеми заключається в тому, що використання одного механізму захисту може суттєво змінити вимоги до реалізації іншого механізму. Тому система повинна забезпечити функціонування механізму замкнутості програмного середовища, яка дозволить ліквідувати будь-яку приховану погрозу. При цьому, користувач буде запускати лише санкціоновані для нього процеси, а значить – не зможе запустити програму, яка дозволить одержати несанкціонований документ.

Виходячи з вищеозначеного, формуємо принципи системного підходу до проектування системи:

– кожний модуль системи повинний проектуватись з урахуванням його впливу на безпеку захищаємо об’єкту в цілому та з урахуванням функцій захисту, що реалізуються іншими механізмами;

– розробка системи – це багатокритеріальна задача, тому слід враховувати не тільки відповідний рівень захисту, який надає кожний модуль системи, але і вплив кожного з них на продуктивність захищаємого об’єкта;

– проектування системи повинне здійснюватись з урахуванням потенціальних погроз, які характеризуються неявними каналами НСД. При цьому, розгляду підлягає не сам тип погрози, а характерні ознаки їх виникнення. Система повинна бути направленою не на погрози, а на можливі канали їх виникнення.

Визначимось з якісними вимогами до ПЗ системи, яка підлягає розробці. В процесі теоретичної побудови системи було вирішено за основу взяти наступні пріоритети цих вимог:

– коректність розробленого ПЗ: гарантія одержання вірних результатів при роботі кожної програми, модуль та системи в цілому;

– достовірність: забезпечення раціональної реакції системи на нестандартні ситуації та надання відповідної теоретичної інформації (в разі потреби).

В основу побудови системи вирішено покласти використання вбудованого механізму криптозахисту CryptoAPI ОС Windows. Функції CryptoAPI забезпечують прикладним програмам доступ до криптографічних функцій ОС. Однак вони є лише будуть “передавальною ланкою” в складному ланцюгу обробки інформації. Основну роботу в системі планується покласти на сховані від очей програміста/користувача функції, що будуть входити до спеціалізованих програмних модулів-криптопровайдерів.

Програмна частина криптопровайдера – це dll-файл, підписаний Microsoft, періодично ОС проводить перевірку ЕЦП для виключення можливості підміни криптопровайдера. В додатковому визначені структури CryptoAPI в даному розділі немає необхідності, оскільки вона детально описана в попередніх розділах пояснювальної записки, як визначені і основні функції, які повинна забезпечити в процесі експлуатації система:

– забезпечення криптографічних перетворень: шифрування ключів-симетричних та асиметричних; шифрування файлів/програм; формування (з послідуючим шифруванням) універсального ідентифікатора;

– забезпечення вибору файлів/програм, що підлягають захисту;

– забезпечення роботи з бібліотеками ОС: Crypto API;

– забезпечення обробки виключних ситуацій та помилок з графічним відображенням повідомлення про їх виникнення.

Програмне забезпечення системи буде розроблене на мові програмування DELPHI. Вибір DELPHI обумовлений наступними причинами:

– об’єктно-орієнтована мова програмування DELPHI дозволяє зручно абстрагувати об’єкт проектування і його елементи і організувати класи загального призначення із контролем доступу до даних за рахунок введення

захищених та відкритих членів класу, що ефективно моделюють роботу реальних об’єктів з використанням пересилки інформації між ними;

– у мові DELPHI представлені зручні засоби роботи як на високому рівні, так на середньому та низькому в той час, як інші сучасні мови надають перевагу одному напрямку (або високий рівень із створенням зручного інтерфейсу, або низький рівень із програмуванням мережевого зв’язку з недостатнім забезпеченням функцій інтерфейсу);

– окрім цього, мова DELPHI характеризується: високою продуктивністю; наявністю інтерфейсів до множини СУБД; низькою вартістю; простотою вивчення і використання; стійкою підтримкою об’єктно-орієнтованого програмування; переносимістю; доступністю вихідного коду; доступність підтримки з боку розробників.

Таким чином, вибір мови програмування DELPHI є доцільним, оскільки вона має засоби дуже зручного опису елементів об’єкта та зв’язків між ними та зв’язків між ними та зовнішнім середовищем і тому дозволить розробити ПЗ, яке забезпечить виконання закладених ТЗ функцій.

При побудові ПЗ планується використати стандартні модулі, які виконують наступні функції:

– Menus – модуль, що забезпечує (обслуговує) всі об’єкти та процедури для системи меню DELPHI, а також випадаючи меню та активні елементи рядка статусу, робота з меню;

– Messages, Dialogs – модулі, що забезпечують процедури і функції виклику діалогових вікон;

– SysUtils – модуль, що забезпечує процедури та функції нескладних математичних функцій, функції перетворення з одного типу в інші; деякі тригонометричні та гіперболічні функції; процедури та функції перетворення дат і часу, роботи з рядками та інше;

– Classes – процедури та функції для роботи з класами;

– Variants – процедури та функції перетворення типуVariant в заданий тип;

– Graphics – процедури та функції по роботі з графікою і графічними об’єктами;

– Forms – процедури та функції по роботі з формами, що підключають та забезпечують можливість використання стандартних бібліотечних

– функцій, а саме: підключення діалогових вікон, меню користувача, організації функції натиску кнопок для отримання з клавіатури режиму роботи системи в поточний момент часу, виводу на екран файлу з інформацією про режим роботи системи, підготовки екрану до діалогу з користувачем, тощо. Використання стандартних бібліотек також дозволить полегшити побудову деяких частин програми.

Як ми вже визначились в попередніх розділах пояснювальної записки, основним методом захисту електронного документообігу, який буде надавати система, є організація криптографічного захисту. Але щоб його неможливо було “обійти” з іншого боку – наприклад, перехопити із незахищеної області пам’яті секретні паролі – криптографічні функції повинні бути частиною ОС.

В родині ОС Windows забезпечується реалізація шифрування, генерації ключів та інші криптографічні засоби. Ці функції необхідні для роботи самої ОС, але можуть бути використанні для розробки інших систем захисту. Нами також будуть використані: криптографічний інтерфейс ОС Windows, деякі функції бібліотеки CryptoAPI, які забезпечать прикладним програмам доступ до криптографічних можливостей ОС Windows:

– Crypto Enum Providers (і, резерв, прапорці, тип, ім’я, довжина імені) – для повернення імені і типу і-го порядку криптопровайдера в системі. При цьому нумерація буде починатись з нуля;

– Crypto AcanireContext (провайдер, контейнер, ім’я, тип, прапорці) – виконує підключення до криптопровайдера з заданим типом і ім’ям і повертає його дескриптор (контекст). При підключенні ми будемо передавати функції

прапорець CRYPT VERIFYCONTEXT, який дозволить одержати контекст без підключення до контейнера ключів;

– CryptGetProvParam (провайдер, параметр, розмір даних, прапорці) – для повернення значення вказаного параметра провайдера (наприклад – версії: типу реалізації: програмний, апаратний, змішаний) підтримуваних алгоритмів (PP ENUMA LGS);

– CryptReleaseContext (провайдер, прапорці) – дозволить звільнити дескриптор провайдера.

Кожна з цих функцій, як і більшість і нших функцій Crypto API, повертає логічне значення, яке дорівнює trye при успішному завершенні, і false – якщо виникли помилки. Для одержання коду помилки будемо використовувати функцію GetlastError. Текст процедури, яка буде виводити інформацію про встановленні в системі криптопровайдери, підлягає розробці в процесі написання ВКРМ.

При розробці програмного забезпечення планується взяти за основу в побудові програмного забезпечення системи виявлені позитивні якості розглянутих систем-аналогів.

В розробці ПЗ систем використаємо програмування під ОС WINDOWS з використанням подій WINDOWS та елементів ActiveX, що надасть програмній моделі більшої оглядності та зрозумілості для сучасного користувача, який звик працювати в цій операційній системі. Під час роботи системи необхідно організувати роботу досить потужного інтерфейсу користувача, який забезпечить своєчасне виправлення помилок та надасть, при необхідності, допоміжну інформацію:

– теоретичні відомості по виправленню некоректних дій користувача по відношенню до операційної системи та до системи;

– теоретичні відомості про принципи роботи системи;

– нормативні документи, які регламентують роботу розробника та користувача ПЗ/інформації.

Організація процесу захисту інформаційних масивів від НСД та НСВ повинна відповідати та проводитись згідно діючих державних стандартів, які пред'являють ряд загальних та специфічних вимог, а саме:

– повинні використовуватись сучасні методи організації та керування, включаючи системно-структурний аналіз та моделювання процесів, економічно-математичні методи;

– повинні використовуватись раціональні параметричні та типорозмірні ряди виробів виробництва та засобів контролю;

– повинна бути орієнтація на оптимальний, для конкретних умов рівень автоматизації процесів: контролю, збирання, підготовки, обробки, передачі та представлення інформації.

При розробці програмного забезпечення планується організація системи згідно модульної структури: основна (керуюча) програма та ряд підпрограм загального та спеціального призначення.

Підпрограми загального призначення необхідні для реалізації в системі функцій, які вирішує ПК: інформаційний пошук, кодування, контроль, перетворення інформації, формування вихідних даних для програм керування та контролю, оформлення заданого пакету технічної документації.

Підпрограми спеціального призначення будуть виконувати функції контролю неконкретних дій користувача та оптимізації нестандартних ситуацій в роботі системи. Структурними елементами підпрограм другого типу є модулі програми.

Побудова системи повинна відповідати шести основним принципам: композиції; єдності; відкритості; максимальної універсальності; інформаційної єдності; інваріантності. Розглянемо більш детально кожен з них.

Принцип композиції складається з того, що система, яка підлягає розробці, створюється на базі вже існуючих систем, які були розглянуті в цьому розділі, але з використанням нового типу алгоритмів, які будуть розроблені в процесі розробки ВКРМ та з використанням криптографічної підсистеми ОС Windows 2000. Це дозволить значно зекономити час та кошти при розробці системи.

Принцип єдності допускає, що на усіх стадіях розробки, функціонування та подальшого розвитку системи її цілісність повинна бути забезпечена зв'язками між окремими частинами та компонентами, а саме: методичним забезпеченням; програмним забезпеченням; технічним забезпеченням; інформаційним забезпеченням; організаційним забезпеченням, які, як складові системи, більш достатньо будуть визначені та обґрунтовані у наступних розділах пояснювальної записки дипломного проекту.

Принцип відкритості вимагає, щоб система створювалась та функціонувала як динамічна система, тобто розширяючи та удосконалюючи свої компоненти. Це обумовлено складністю та багатофункціональністю самої системи і наявністю систем-аналогів на ринку програмних продуктів, які постійно удосконалюються. З іншої сторони, постійно вдосконалюють свій стиль роботи та майстерність і зловмисники, що також буде вимагати відповідної доробки системи.

Принцип максимальної універсальності вимагає розробки багатофункціональної мобільної системи. Реалізація цієї задачі передбачає рішення наступних питань: технічного забезпечення системи; математичного забезпечення системи; програмного забезпечення системи. Окрім цього, важливим етапом є розробка організаційних аспектів впровадження та подальшого супроводження системи в промисловій експлуатації.

Принцип інформаційної єдності передбачає уніфікацію інформаційних масивів системи, а також вхідної та вихідної мов описання компонентів всередині системи.

Принцип інваріантності означає, що система, яка підлягає розробці, з точки зору її використання на інших типах технічних засобів, повинна бути універсальною або типовою. Вирішення цієї задачі планується введенням в програмне забезпечення (ПЗ) універсальних або спеціальних модулів. Наявність достатньої кількості такого типу модулів, пакетів прикладних програм дозволить забезпечити закінченість рішення задач системою.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

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

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

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Забезпечення надійного захисту інформації не є разовим заходом, а сукупністю різноманітних заходів. Але традиційна архітектура навіть комплексної системи захисту не є оптимальним рішенням поставленої задачі, адже зловмисники – це професіонали вищого ґатунку, які постійно вдосконалюють свою майстерність та професійний рівень. Це підтверджує той факт, що на даний час всі існуючі системи захисту фактично вже зламані.

Тому при розробці системи захисту було вирішено використати нетрадиційний підхід до побудови її архітектури, в основі якої закладений науковий напрямок “Теорія забезпечення безпеки програм та їх комплексів”.

Використання цього напрямку забезпечує: інтеграцію основних наукових положень прикладної теорії алгоритмів, теорії управління якістю ПЗ, теорії надійності і дозволяє на практиці реалізувати єдині системні позиції попередження випадкового/навмисного розкриття, злому, знищення зберігаємої/обробляємої/захищаємої/траспортуємої інформації/ПП в локальній та глобальній мережах.

Вирішення проблеми забезпечення цілісності і достовірності інформації при її передачі в захищених/незахищених мережах, витікає з необхідності захисту від зловмисних дій, адже несанкціоноване зчитування інформації, яке здійснюється в мережі, направлене:

– на одержання секретної інформації;

– на ідентифікацію інформації, яку запитує користувач;

– на зчитування паролів та їх ототожнення з конкретним користувачем;

– на спотворення або видалення власної інформації сервера та порушення роботи мережі;

– на контроль активності абонентів мережі для одержання побічної інформації про взаємодію користувачів та характер інформації, якою обмінюються абоненти мережі.

Окрім цього, зловмисником:

– можуть бути змінені атрибути (характеристики) електронних документів;

– зловмисник може видати себе за дійсного власника електронного документа або, навпаки, відмовитись від авторства на власний документ/файл;

– зловмисник може дизасемблювати та скопіювати програму з ціллю її подальшого розповсюдження.

Найбільш ефективними методами захисту від вищеозначених зловмисних дій та погроз є криптографічні методи захисту.

Це обумовлено тим, що відомі та поширені способи контролю цілісності програм розроблені на базі використання контрольної суми повздовжнього контролю і контролю на чітність, і, як правило, представляють собою досить прості (але надійні) способи захисту від внесення змін в код програм.

Оскільки область значень контрольної суми досить обмежена, а значення функції контролю на чітність взагалі представлені одним-двома бітами, то для досвідченого порушника не важко найти наступну колізію:

$$f(k_1) = f(k_2), \tag{3.1}$$

де k_1 – код програми без спотворення;

k_2 - код програми, спотворений зловмисником;

f - функція контролю.

В цьому випадку, значення функції для різних аргументів співпадають, а це означає, що тестування не виявить внесених дефектних змін до файлу.

Тому для встановлення достовірності необхідно використати більш складні методи, такі, як автентифікація з використанням криптографічних методів, які віднайдуть також і сліди, що залишив зловмисник.

Найбільш ідеальним варіантом є забезпечення такого ступеню захисту, коли зловмисник не зможе використати в комерційних цілях одержаний файл з однієї причини – він не зможе його одержати.

Термін “криптологія” походить від двох грецьких слів: “крипто” (тайний) та “логос” (вчення). Криптологія, як наука, складається з двох тісно теоретичних і практично пов’язаних дисциплін: криптографії та криптоаналіза.

Криптографія – це наука про способи та методи перетворення (шифрування) інформації з ціллю її захисту від незаконних користувачів. Криптоаналіз займається методами і способами злому шифрів.

Шифр (криптосистема) – метод перетворення інформації з ціллю її захисту від незаконних користувачів. Злом шифру – процес одержання інформації з шифрованого повідомлення без знання використаного шифра.

Шифрування – процес використання шифру до інформації, що підлягає захисту. Дешифрування – процес, зворотний шифруванню.

Вихідний текст, що має сенсове/логічне значення, яке необхідно зашифрувати, називають відкритим текстом. Зашифрований текст, що має вигляд випадкового набору символів/цифр, називається шифротекстом або криптограмою.

Під ключем в криптографії слід розуміти змінний елемент шифру, який використано для шифрування конкретного відкритого тексту.

Зазвичай, атака зловмисника (криптоаналітика) зводиться до спроби розкриття шифру (спроба здійснення атаки) на основі шифротекста. Якщо ж зловмисник має до того ж деякі уривки відкритого тексту і відповідні їм елементи шифротексту, тоді слід чекати спроби здійснення атаки на основі відкритого тексту.

Атака на основі вибраного відкритого тексту заключається в тому, що противник, використавши свій відкритий текст, одержує правильний шифротекст і робить спробу зламати шифр.

Теоретично існує абсолютно стійкій шифр, але єдиним таким шифром є будь-яка форма стрічки одноразового використання, в якій відкритий текст об'єднується з випадковим ключем такої ж довжини.

Цей результат був доказаний К.Шеноном за допомогою розробленого їм теоретико-іформаційного метода дослідження шифрів, але широкого застосування на практиці фактично не одержав із-за великої ступені складності та високої ціни.

В силу даних причин, абсолютно стійкі шифри використовуються тільки в спеціальних мережах для передавання особливо важливої державної інформації. Тому цей метод недоцільно застосовувати в нашій розробці.

Щоб криптозахист не можна було "обійти" з іншої сторони, доцільно використати вбудовані криптографічні можливості ОС Windows. До недоліків слід віднести повноцінне функціонування Crypto API тільки на базі ОС Windows. Тому при виборі операційної системи, на яку буде орієнтована наша розробка, ми зупинимось саме на цій операційній системі.

Функції Crypto API забезпечать прикладним програмам системи доступ до криптографічних можливостей Windows, але вони є лише "передавальною ланкою" в складному ланцюгу обробки інформації.

Основну роботу будуть виконувати сховані від очей програміста функції, які входять в спеціалізовані модулі – провайдери (постачальники) криптографічних послуг, або криптопровайдери, як це показано на рисунку 3.1.

Програмна частина криптопровайдера представляє собою dll-файл, підписаний Microsoft; періодично Windows перевіряє ЕЦП, що виключає можливість заміни (підміни) криптопровайдера. Криптопровайдери відмінні один від одного наступними ознаками: складом функцій (одні – виконують

шифрування, інші – створюють та провіряють ЕЦП, тощо); вимогами до ОС алгоритмами, що здійснюють базові дії (створення ключів, хешування, тощо).

По складу функцій і забезпечуючих їх виконання алгоритмів, криптопровайдери підрозділяються на типи.

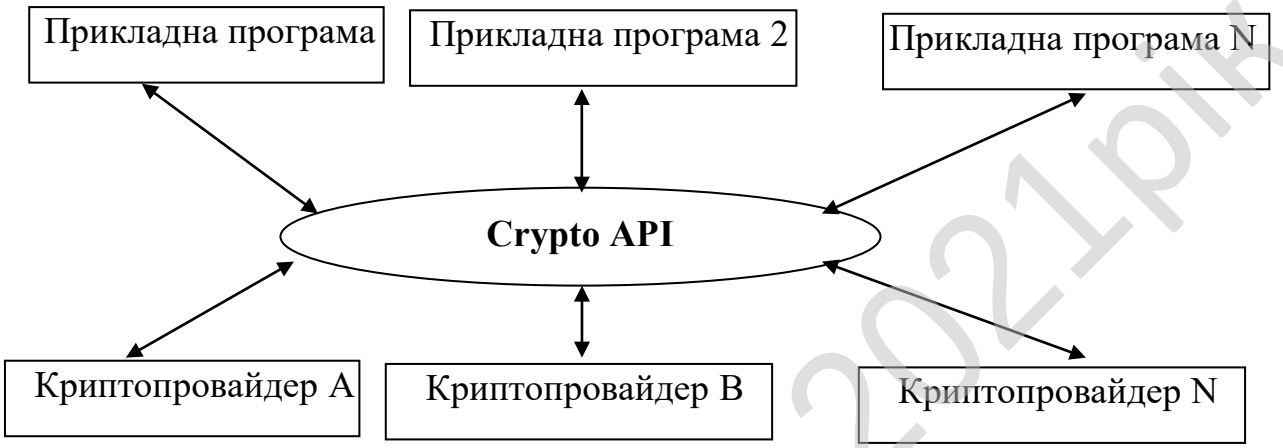

Рисунок 3.1 – Структурна схема взаємозв'язку ПП з криптопровайдерами

Наприклад, CSP типу RROV RSA FULL (підтриму є як шифрування, так і ЕЦП) використовує для обміну ключами і створення ЕЦП алгоритм RSA, для шифрування – алгоритмами RC2 та RC4, для хешування МД5 та SHA.

Використання криптографічних можливостей Windows, по сутті, є роботою програми з графічним пристроєм на низькому рівні, адже криптопровайдер подібний графічному драйверу: він може забезпечувати взаємодію ПЗ з обладнанням (пристрій читання смарт-карт, апаратні датчики випадкових чисел, тощо).

Для виведення інформації на графічний пристрій, додаток не повинен безпосередньо звертатися до драйвера – замість цього в системі необхідно одержати контекст пристрою за допомогою якого і будуть здійсненні всі операції. Це дозволить використовувати при написанні програми графічний пристрій, нічого не знаючи про його програмну реалізацію. Точно такий же принцип використаний і для криптографічних функцій: додаток звертається до

криптопровайдера не напрямки, а через CryptoAPI, але спочатку провівши запит в ОС контексту криптопровайдера.

Таким чином, ОС Windows надає ду же зручний інструмент для організації криптозахисту у вигляді бібліотеки CryptoAPI з набором функцій та процедур з криптопровайдерами, але необхідно розробити алгоритми реалізації основних функцій системи криптозахисту:

– підключення;

– шифрування/розшифрування за допомогою симетричних/асиметричних ключів;

– створення контейнерів ключів;

– створення ключових пар, тощо.

Ці алгоритми об’єднаємо в модуль CryptoAPI.

Після того, як будуть викликані відповідні криптопровайдери (через функції Crypto API), перейдемо до шифрування файлів. Шифрування будемо виконувати з урахуванням вищеозначених вимог, обмежень і таким чином, щоб лише користувач/розробник, знаючий відповідний пароль, міг одержати доступ до них. Більш детально розробка БСА та їх програмування будуть описані в розділі 4 пояснювальної записки.

Для розробки ПЗ підсистеми нами обрано мову програмування DELPHI, обґрунтування цього вибору надається в розділі 2 пояснювальної записки.

Таким чином, нами визначені та обґрунтовані:

– вимоги та обмеження, врахування яких повинна забезпечити система;

– методи, які будуть використані при розробці архітектури системи та її програмного забезпечення;

– функції, виконання яких повинна забезпечити система.

Маємо всі необхідні дані для визначення структури майбутньої системи та розробки її структурної схеми та функціональної схеми.

3.2 Розробка структурної схеми

Основною задачею, що повинна бути виконана згідно технічного завдання та постановки задачі на реалізацію ВКРМ, є розробка програмного забезпечення системи кібербезпеки для захисту інформаційних та програмних масивів на основі використання бібліотеки Crypto API

Розроблене ПЗ повинне забезпечити в процесі експлуатації виконання наступних функцій:

– шифрування/розшифрування об’єкту захисту за допомогою симетричних та асиметричних алгоритмів (використаємо їх комбінацію) та блочних шифрів;

– створення: контейнерів ключів, ключових пар, ключа обміну ключами (КОК), електричного цифрового підпису (ЕЦП);

– створення: цифрових конвертів, сеансових ключів за допомогою блочних шифрів;

– з’єднання з криптопровайдером ОС Windows;

– визначення хеш-функції.

На основі визначених вимог та обмежень до майбутньої системи (підрозділ 3.1 пояснювальної записки) визначимо більш детально компоненти її структури.

Система умовно поділена на дві змістовні частини:

– частина 1 «ПЗ захисту. Клієнтська частина»: бібліотека CryptoAPI, функції шифрування: шифрування блочним шрифтом, шифрування блочним шифром і підпис ЕЦП, підпис файлів ЕЦП; передача на сервер; прийом від сервера; дешифрування одержаних файлів;

– частина 2 “ПЗ захисту. Серверна частина”: передача клієнту списку файлів знаходиться на зберіганні; передача клієнтській частині файлів по запиту; збереження файлів.

В якості технології передачі даних використаємо технологію Socket та протоколи TCP/IP.

Елементи частини 1 та частини 2 певним чином пов'язані, а вихідний елемент є постійним і фіксованим. Разом з тим, вони, як елементи системи, тісно взаємозв'язані, використовуючи в процесі роботи вихідні дані однієї частини, як вхідні дані іншої.

Використання в системі декількох механізмів захисту одночасно до одного і того ж файлу, дозволить розробити дієспроможну та досить потужну систему, призначення якої – забезпечення захисту інформації в мережі від НСД та НСВ.

Написання такого програмного продукту вимагає від програміста-розробника знання принципів побудови комп'ютерних систем захисту програмних продуктів та дотримання вимог, які пред'являються до систем аналогічного класу та спрямування, (описаних в розділі 2 пояснювальної записки).

Під час розробки ПЗ згідно теми ВКРМ, автор має звернути увагу на можливість вирішення програмою найбільшого кола задач, врахувавши при цьому позитивні якості систем-аналогів, розглянутих в розділі 2 пояснювальної записки, та здійснити спробу уникнення виявлених недоліків.

В процесі експлуатації система захисту, на основі використання бібліотеки Crypto AP, має вирішувати наступні основні питання:

– шифрування об'єкту захисту (модуль криптозахисту);

– підписання файлу з елементами захисту ЕЦП (модуль криптографії)

– дешифрування одержаного файлу;

– автентифікація респондента по ЕЦП,

які підлягають програмній реалізації в процесі виконання ВКРМ.

Таким чином, нами визначені складові компонента майбутньої системи, способи зв'язку між ними та чітко визначені функції, які вони будуть виконувати. Тому можемо розробити структурну схему системи, яка наведена на рисунку 3.2.

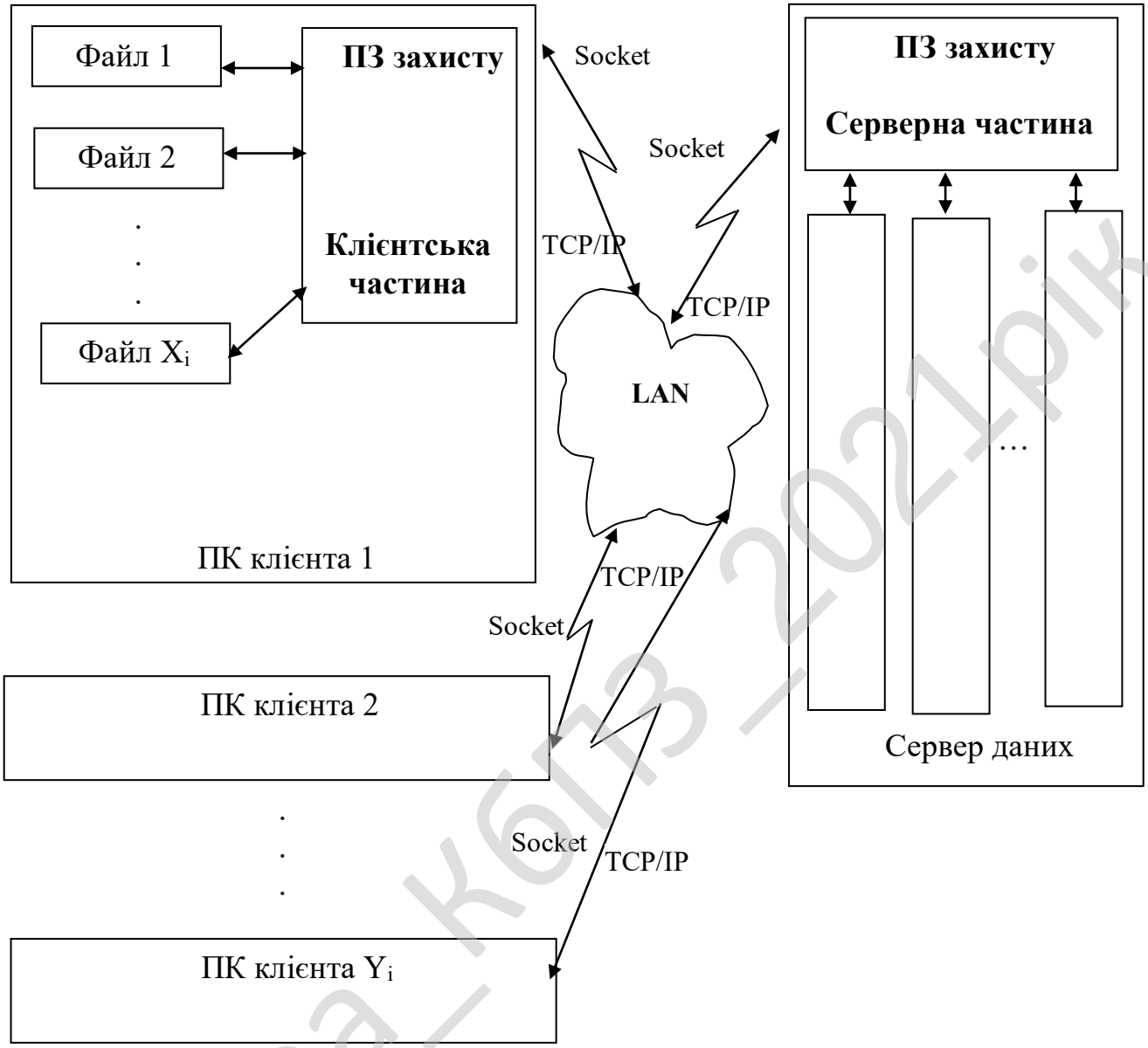

Рисунок 3.2 – Структурна схема системи

3.3 Розробка функціональної схеми

Коли цільові функції системи вже чітко означені, тобто, задача на розробку поставлена, то для одержання текстів вихідних програм необхідно виконати ряд послідовних дій:

– докладно описати задачу;

– виконати аналіз задачі;

– провести інженерну інтерпретацію задачі (з залученням апарату формалізації);

– розробити функціональну схему системи;

– розробити загальну блок-схему алгоритму (БСА) роботи системи;

– розподілити робочі регістри і пам’ять МК ЕОМ;

– розробити тексти робочої програми.

При розробці БСА та текстів програм використаємо метод декомпозиції, при якому вся задача послідовно розподіляється на менші функціональні модулі, кожний з яких можна аналізувати, розробляти та налагоджувати окремо від інших.

При виконанні кожної окремої програми в МК керування передається від одного функціонального модуля до іншого. Схема зв’язаності цих функціональних модулів, кожен з яких реалізує окрему процедуру, створює загальну блок-схему алгоритму (БСА) системи.

Це розподілення задачі на модулі і підмодулі виконується послідовно до такого рівня, коли розробка БСА стає простою і зрозумілою справою. Окрім цього, метод послідовної декомпозиції є достатньо гнучким, що дозволить провести ступінь деталізації БСА у відповідності із складністю процедури.

Розробка БСА функціонального модуля будь-якої програми (прикладної чи керуючої) має яскраво виявлений ітеративний характер, тобто, вимагає багаторазових проб, раніш ніж виникне впевненість, що алгоритм реалізації процедури правильний та завершений. Незалежно від функціонального призначення процедури, при розробці її алгоритму необхідно:

– визначити призначення модуля;

– визначити методи одержання модулем вхідних даних;

– визначити необхідність будь-якої попередньої обробки вхідних даних: маскування, зсунення, масштабування, перекодування. Якщо така необхідність є, то до складу БСА необхідно ввести відповідні оператори;

– визначити метод перетворення вхідних даних в вихідні;

– визначити методи виведення з модуля оброблених даних: передати в пам’ять, передати в спеціальну програму чи в порти введення/виведення даних;

– визначити необхідність постобробки даних, що підлягають виведенню: зміна формату, перекодування, масштабування, маскування;

– проаналізувати одержаний результат, виконати ітеративне корегування алгоритму з ціллю його спрощення та досягнення логічності, стрункості та чіткості графічного образу.

Послідовне використання вищеозначеної поетапної процедури проектування алгоритмів дозволить одержати дійсно працездатне ПЗ. Адже ті зміни, які легко врахувати на етапі планування, зазвичай вимагають великих зусиль та багато часу на етапі реалізації ПЗ та його відлагодження.

Програмне забезпечення, що підлягає розробці, повинне бути компактним та оглядним. Це планується досягнути за рахунок модульної побудови системи у вигляді взаємодії головної програми та підпрограм. При цьому складні програми мають формуватись з менш складних модулів-підпрограм, які, в свою, чергу, є готовими модулями.

Така ієрархічна багаторівнева організація комплексу програм є на практиці основним засобом боротьби зі складністю розв’язуваних проблем. Окрім цього, модульна побудова ПЗ забезпечить високу ступінь мобільності та адаптивності, як це вимагає ТЗ та вимоги до сучасних систем даного класу. Це в подальшому дозволить:

– легко адаптувати систему під ПЗ основної системи на ПК та без суттєвих доробок перейти на новий тип КТЗ;

– без суттєвих доробок та перебудови структури системи розширити її функції шляхом введення нових модулів. Така необхідність може виникнути при переході на обробку нового типу даних або підключенні нового типу обладнання, появи нових видів обмежень нових типів хакерських атак, тощо.

Мінімальна конфігурація комплексу технічних засобів (КТЗ), що забезпечить в повному обсязі функціонування розробленого ПЗ, буде мати наступну конфігурацію:

– персональний комп'ютер (ПК) з обсягом оперативної пам'яті не менше 256 Мб;

– операційна система ОС Windows.

Таким чином, нами визначені складові програмного забезпечення, компоненти системи, тому можемо розробити функціональну схему системи, яка наведена у відповідності з рисунком 3.3.

Якщо клієнту необхідно розмістити файл на сервері, то перед його відправкою будуть виконані наступні перетворення:

– для файлів загального доступу, які потребують автентичності, використовуючи модуль ЕЦП, проводимо їх підписання ЕЦП з наступною передачею на сервер;

– для файлів, які мають комерційн у таємницю і потребують надійного захисту, використовуючи модуль шифрування, проводимо їх шифрування блочним шифром з наступною передачею на сервер;

– для файлів, які потребують високого ступеня захисту, використовуємо два модуля означених вище перетворень в наступній послідовності:

– шифрування блочним шифром → підписання ЕЦП з наступною передачею на сервер.

Операції шифрування/дешифрування/підписання ЕЦП, перевірка ЕЦП виконуються за допомогою бібліотеки CryptoAPI, використовуючи описані в ній функції.

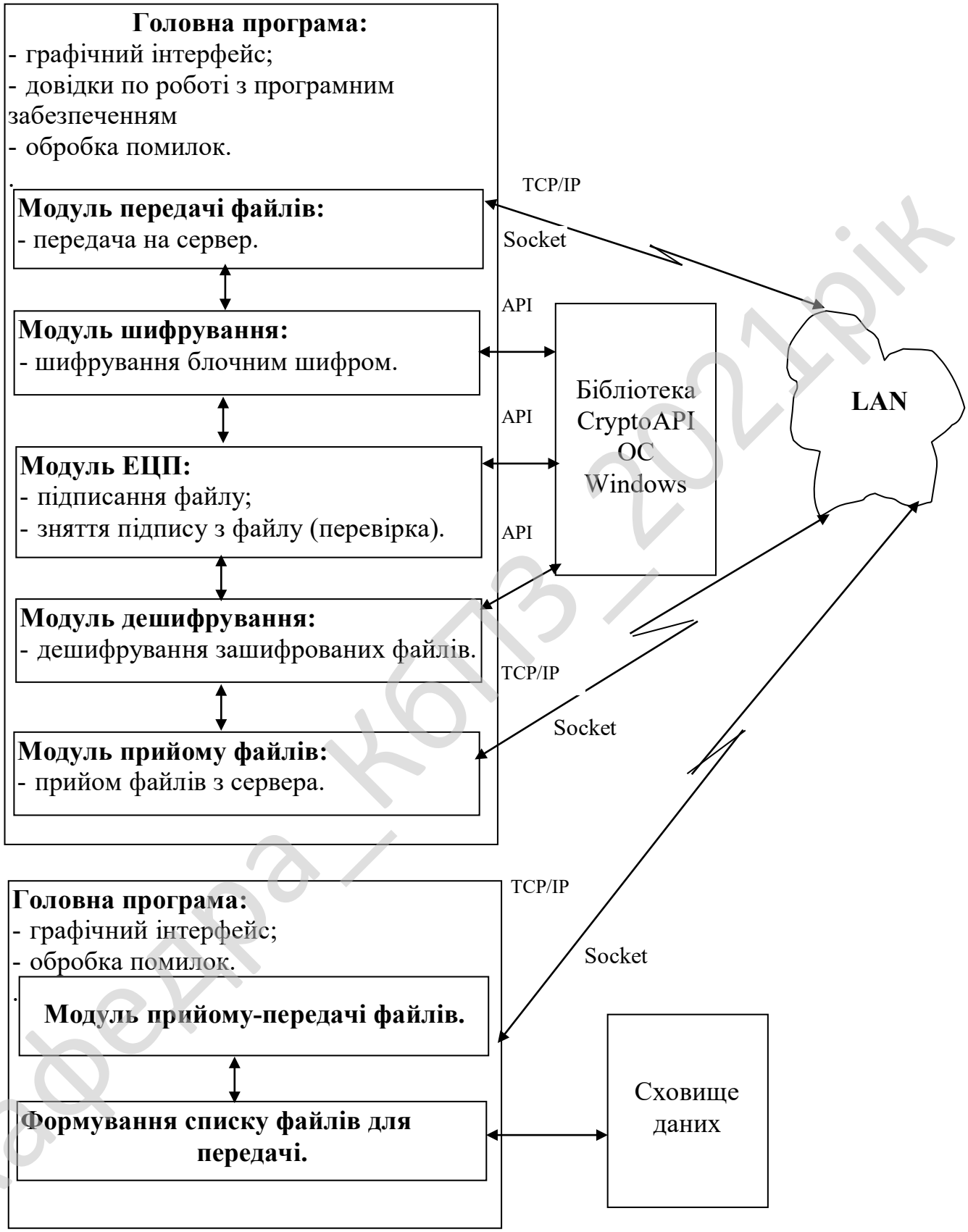

Рисунок 3.3. – Функціональна схема системи

Тому для успішного функціонування розробленого ПЗ, зазначена вище бібліотека повинна бути встановленою (постачатись разом з ПЗ) на ПК користувача/користувачів, який/які будуть використовувати нашу систему в повсякденній роботі.

Для забезпечення функціонування системи системний адміністратор повинен виконати певні налаштування: створення захищеної папки (контейнера) в ОС Windows, в яку поміщаються ним же створені ключеві пари (ключ обміну ключами і ключ підпису). Ці дані є секретними і тому на кожному ПК пари ключів будуть по своєму умісту прив'язаними до конкретного користувача.

Адміністратор системи також визначає та установлює на ПК перелік ключів, які дозволяють розшифрувати певні файли (згідно права доступу до роботи з певними типами файлів користувача ПК).

Модуль приймання файлів відправляє запит до серверної частини і отримує список файлів, дозволених для роботи даному користувачу. Цей модуль забезпечує виконання в системі і функції одержання запитаних і дозволених файлів від сервера по сокетній технології.

3.4 Розробка діаграми процесів

В процесі розробки системи була визначена ієрархія процесів в системі: головні та підлеглі. Для визначення їх взаємодії розроблена діаграма процесів, яка наводиться на рисунку 3.4.

Визначивши і обґрунтувавши:

– проектні рішення щодо побудови системи та розробки її програмного забезпечення;

– побудову структурної та функціональної схем системи діаграми взаємодії процесів в системі;

– комплекс технічних засобів, що забезпечить експлуатацію системи, маємо всі необхідні критерії та дані для розробки програмного забезпечення системи.

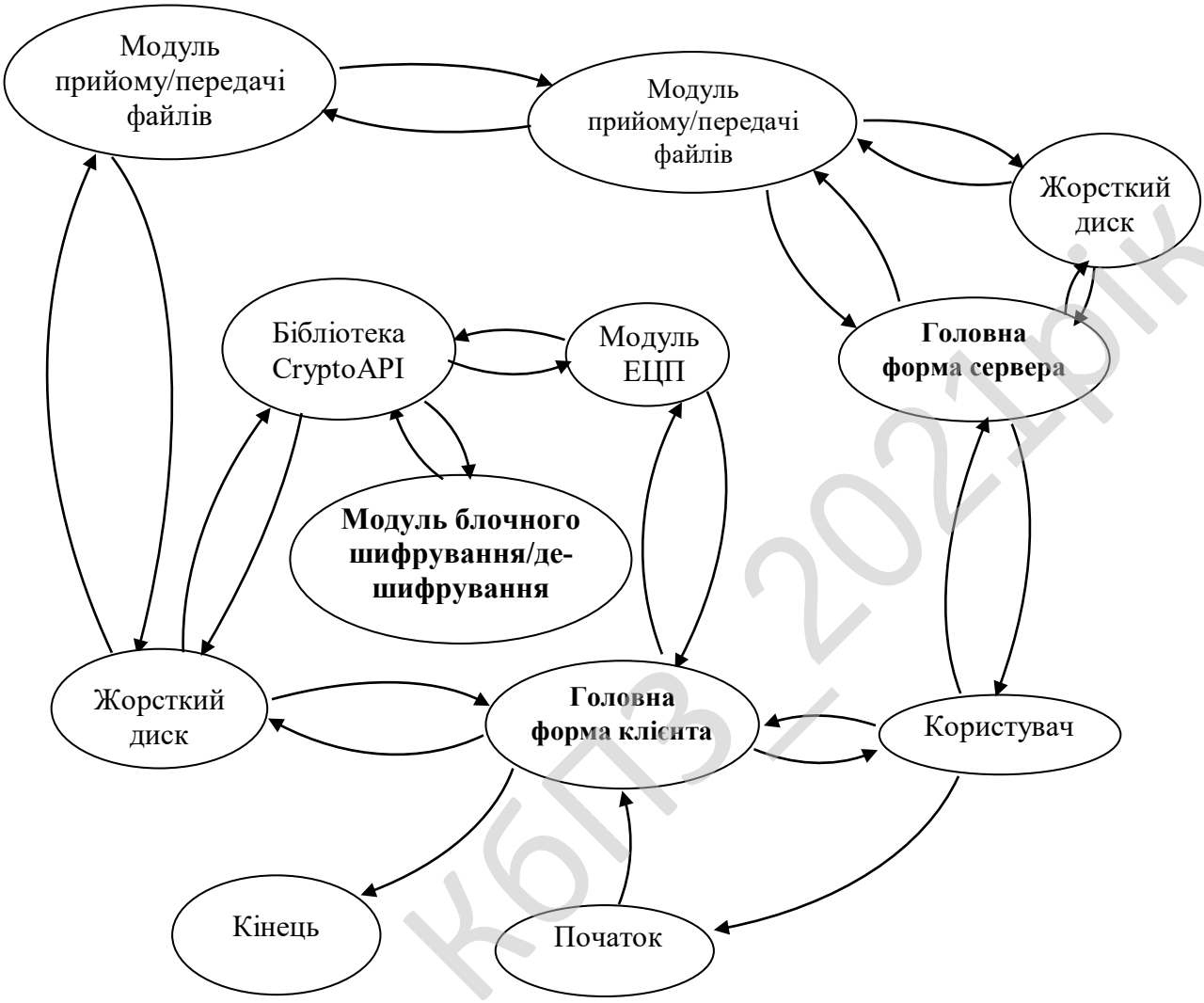

Рисунок 3.4 – Діаграма взаємодії процесів в системі

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

Розроблене в процесі реалізації ВКРМ ПЗ системи забезпечує виконання означених в ТЗ та більш детально визначених в розділі 3 пояснювальної записки функцій, направлених на виконання основної задачі – забезпечення захисту програмних масивів від посягань зловмисників в локальній та глобальній мережах.

При проектуванні системи були враховані наступні аспекти:

– комплексування різнорідних механізмів в єдиній системі;

– взаємний вплив захисних механізмів;

– орієнтація на статистику погроз при визначені ключових механізмів захисту системи;

– ідентифікація та перевірка достовірності суб’єктів доступу при вході в систему по ідентифікатору пароля ключа;

– контроль доступу суб’єктів до ресурсів, які підлягають захисту в відповідності з матрицею доступу.

При реалізації задачі захисту інформації з використанням комплексування декількох механізмів захисту в єдину систему, було проведено аналіз вирішення конкретної задачі з урахуванням способів можливого подолання визначених механізмів в сукупності та кожного окремо (наприклад, з використанням програм-сніферів, програм підбирання паролів, програм-дизасемблерів, тощо). Виходячи з цього, було вирішено використати якісні, тобто, визначені погрози (із числа існуючих згідно даних статистики та потенційно можливих), яким необхідно протидіяти.

Інша сторона проблеми заключається в тому, що використання одного механізму захисту може суттєво змінити вимоги до реалізації іншого механізму. Тому система повинна забезпечити функціонування механізму забезпечення замкнутості програмного середовища, яка дозволить ліквідувати будь-яку приховану погрозу. При цьому, користувач буде запускати лише санкціоновані для нього процеси, а значить – не зможе запустити програму, яка дозволить одержати несанкціоновані інформаційні файли.

Визначимось з якісними вимогами до ПЗ. В процесі теоретичної побудови системи було вирішено за основу взяти наступні пріоритети цих вимог:

– коректність розробленого ПЗ: гарантія одержання вірних результатів при роботі кожної програми, модуля та системи в цілому;

– достовірність: забезпечення раціональної реакції системи на нестандартні ситуації та надання відповідної теоретичної допомоги (в разі потреби).

В основу побудови ПЗ системи покладено використання вбудованого механізму криптозахисту CryptoAPI ОС Windows. Функції CryptoAPI забезпечують прикладним програмам доступ до криптографічного механізму ОС. Однак вони є лише “передавальною ланкою” в складному ланцюгу обробки інформації. Основну роботу в системі виконують сховані від очей системного програміста/користувача функції, що входять до спеціалізованих програмних модулів-криптопровайдерів.

Програмна частина криптопровайдера – це dll-файл, підписаний Microsoft, періодично ОС проводить перевірку ЕЦП для виключення можливості підміни криптопровайдера. В додатковому визначені структури CryptoAPI в даному розділі немає необхідності, оскільки вона детально описана в попередніх розділах пояснювальної записки, як визначені і основні функції, які повинна забезпечити в процесі експлуатації наша система:

– забезпечення криптографічних перетворень: шифрування ключів-симетричних та асиметричних; шифрування файлів блочним шифром; формування ЕЦП та підпис ним файлів; підпис ЕЦП зашифрованих файлів;

– забезпечення вибору файлів, що підлягають захисту;

– забезпечення роботи з бібліотекою Crypto API ОС Windows;

– забезпечення обробки системних подій, виключних ситуацій та помилок з графічним відображенням повідомлення про їх виникнення (ця функція буде присутньою в кожному модулі системи).

В попередніх розділах пояснювальної записки була детально описана структура CryptoAPI, тепер використаємо її в практичних цілях – для побудови алгоритмів системи. Система забезпечує шифрування ключів – відкритого та закритого, файлів, програмних продуктів, підпис зашифрованого файлу ЕЦП.

Шифрування інформативних файлів необхідне для забезпечення захисту від НСД та НСВ при їх транспортуванні в корпоративній мережі, до якої надається доступ користувачам з незахищеної глобальної мережі.

Для шифрування даних використаємо комбінацію з симетричних (закритих) та відкритих алгоритмів, що носить назву асиметричних алгоритмів (ключів).

Симетричність означає, що для шифрування і розшифровки даних використовується один і той же ключ. Для кодування і декодування обидві сторони, що беруть участь в обміні інформацією, повинні дійти згоди про використання ключа ще до початку обміну повідомленнями, але ключ не повинен передаватись тим же способом, що і зашифроване за його допомогою повідомлення.

Пароль (чи ключ) служить для шифрування вихідного повідомлення, зашифрований текст передається через мережу, після чого одержувач розшифровує файл симетричним ключем. Але для цього необхідно спочатку одержати хеш-пароль.

Для створення хеш-паролю створимо хеш-об’єкт, використавши механізм CryptoAPI ОС Windows. Для цього використаємо функцію CryptCreateHash:

(провайдер, ID-алгоритма, ключ, прапорці, хеш),

якій передаємо дескриптор криптопровайдера (дескриптор буде одержаний за допомогою CryptAcquireContext) і ідентифікатор алгоритму хешування. Інші параметри означимо нулями. В результаті одержимо дескриптор хеш-об'єкта. Виконаємо необхідні розрахунки для побудови алгоритму.

Додавання по модулю 2^{32} двох 32 – розрядних чисел. Числа додаються по наступному правилу:

$$A\ [+]\ B = A \langle + \rangle B, \text{ якщо } A \langle + \rangle B < 2^{32};$$
$$A\ [+]\ B = A \langle + \rangle B - 2^{32}, \text{ якщо } A \langle + \rangle B \geq 2^{32}.$$

Символом {+} означимо операцію додавання по модулю 2^{32} двох 32 – розрядних чисел. Правила додавання наступні:

$$A\ \{+\}\ B = A \langle + \rangle B, \text{ якщо } A \langle + \rangle B < 2^{32} - 1;$$
$$A\ \{+\}\ B = A \langle + \rangle B - 2^{32}, \text{ якщо } A \langle + \rangle B < 2^{32} - 1.$$

В усіх режимах роботи алгоритму використовується ключ довжиною 256 бітів, які представляються у вигляді восьми 32-розрядних чисел X (i). Якщо означити ключ W, то

$$W = X(7)\ X(6)\ X(5)\ X(4)\ X(3)\ X(2)\ X(1)\ X(0). \qquad (4.1)$$

Наведений приклад показує обізнаність автора з проблемою розробки алгоритмів криптографії, але при розробці ВКРМ застосуємо для створення ключа функцію CryptDeriveKey:

(провайдер, ID-алгоритма, хеш-об'єкт, ключ, прапорці),

яка використовує хеш-об'єкт в якості вихідних даних і будує означений ключ для алгоритму шифрування, заданого своїм ID. В результаті одержимо дескриптор ключа для шифрування.

При роботі з CryptoAPI будемо мати справу не з самим об'єктом або його адресою, а з дескриптором – цілими числами, які характеризують розташування об'єкта в внутрішніх таблицях криптопровайдера. Ці таблиці розташовані в

захищеній області пам’яті ОС, так що програми зловмисників не зможуть одержати до них доступу.

Алгоритми шифрування розподіляються на блочні та поточні. Перші – обробляють дані відносно великими по розміру блоками (64, 128 біт, тощо), а другі – побітно. Ми використаємо блочні шифри, вони більш надійні, ніж поточні, оскільки кожний блок тексту піддається складним перетворенням, як це показано на рисунку 4.1 (режим зчеплення).

Враховуючи простоту злому, не будемо використовувати спосіб поблочно-окремого шифрування (ECB – electronic codebook) текстів, а використаємо режим зчеплення блоків шифротекста (CBC – cipher block chaining). В цьому режимі шифрування, черговий блок вихідного тексту спочатку комбінується з попереднім блоком шифротекста (за допомогою XOR), а потім одержана послідовність бітів надходить на вхід блочного шифру.

На виході створюється блок шифротекста і він же використовується для шифрування наступного блока. Самий перший блок вихідного тексту також буде скомбінований з деякою послідовністю бітів, але “попереднього блока шифротекста” ще немає; тому режим шифрування зі зворотним зв’язком потребує використання ще одного параметра – він називається ініціалізуючим вектором (IV – initialization vector).

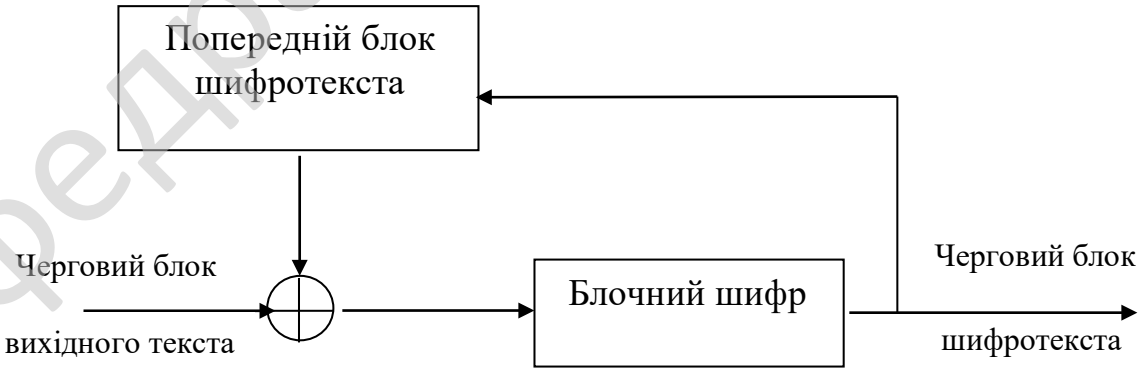

Рисунок 4.1 – Структура блочного шифрування в режимі зчеплення

IV–вектор будемо генерувати окремо за допомогою функції CryptGenRandom і, як і солт-значення, передамо разом з ключем в відкритому вигляді. Розмір IV-вектора дорівнює довжині блока шифра.

Кінцеве шифрування файлів здійснюється функцією CryptEncrypt: (ключ, хеш, фінал, прапорці, дані, розмір-даних, розмір буфера) наступним чином:

– через параметр “ключ” передається дескриптор ключа шифрування;

– параметр хеш використовується, якщо одночасно з шифруванням необхідно вирахувати хеш-значення файла, що підлягає шифруванню;

– параметр фіналу дорівнює true, якщо шифруємий блок текста – останній чи одиночний (шифрування можна здійснювати частинами, викликаючи функцію CryptEncrypt декілька разів);

– значення прапорців повинне бути нульовим;

– параметр “дані” представляє собою адресу буфера, в якому при виклику функції знаходиться вихідний текст, а по завершенню роботи функції – зашифрований текст;

– наступний параметр, описує розмір вхідних/вихідних даних;

– останній параметр задає розмір буфера – якщо в результаті шифрування зашифрований текст не вміститься в буфері, виникає помилка.

Для дешифровки даних використаємо функцію CryptDecrypt:

(ключ, хеш, фінал, прапорці, дані, розмір-даних),

яка є відміною від функції шифрування лише тим, що розмір буфера вказувати не має потреби. Оскільки розмір даних при дешифруванні може тільки зменшитись, то відведеного під них буфера буде явно достатньо.

Для забезпечення цілісності та автентичності програмного коду використаємо асиметричні алгоритми. Цей метод передбачає наявність двох ключів при к ожному сеансі кодування і добре зарекомендував себе навіть у незахищених мережах. Кожен користувач створює два ключі. Кожен ключ являє собою довільний набір цифр обсягом (у деяких випадках) більш ніж у 500 цифр. Обидва ключі зв’язані між собою таким чином, що повідомлення можна зашифрувати за допомогою одного ключа і розшифрувати за допомогою іншого,

однак розшифрувати повідомлення за допомогою ключа, що використовувався для його зашифровки, не можна.

У 1975 році три дослідники з Масачусетського технологічного інституту розробили алгоритм для реалізації методу криптографії на основі відкритого ключа. Криптосистема одержала назву RSA, по перших літерах її авторів – Рона Рівеста (Ron Rivest), Рейді Шаміра (Adi Shamir) і Леонарда Едлмана (Leonard Adleman).

Алгоритм RSA спочатку генерує два різних ключі для кожного користувача. Один із ключів визначений, як відкритий. Цей ключ можна вільно передати кому завгодно будь-яким способом: на дискеті, по електронній пошті чи в роздрукованому вигляді. Відкритий ключ не дозволяє розшифрувати жодне повідомлення, з його допомогою можна лише зашифрувати дані, що потім пересилаються власнику відкритого ключа. Інформацію, закодовану за допомогою відкритого ключа, може декодувати тільки той, у кого є другий ключ (так званий особистий).

Проте алгоритм RSA для кодування повідомлень використовувати не будемо, оскільки це пов’язано з великими витратами часу. Використаємо систему RSA для кодування симетричного (закритого) ключа, за допомогою якого кодується повідомлення. Ця функція використовується в стандарті SSL (Secure Sockets Layer), призначеному для кодування Web-сторінок (адреси відповідних URL починаються з https://, а не з http://). Ключ генерується тільки на Web - браузері, а потім відправляється на Web-сервер.

Для забезпечення безпечної передачі, Web-сервер пересилає свій відкритий ключ Web-браузеру. Web-браузер вибирає симетричний ключ, кодує повідомлення відкритим ключем Web-сервера і відправляє його назад. Тільки Web-сервер може декодувати відкритий ключ за допомогою свого особистого ключа. Таким чином, ключ RSA виступає в ролі “цифрового конверта” для симетричного ключа. Після передачі закритого ключа за допомогою метода RSA,

інформація шифрується з використанням симетричного ключа, оскільки цей спосіб значно швидший.

Подібна система дозволить вибирати симетричні ключі випадковим чином. Якщо хто-небудь і розкриє зашифроване повідомлення, це не дасть ніякої інформації про ключі, за допомогою яких були закодовані інші повідомлення.

В якості ілюстрації наводимо використання алгоритму RSA при розробці ПЗ системи. Ми будемо використовувати його в трьох випадках: для обчислення ключів, для шифрування, для дешифрування.

Обчислення ключів:

– вибір p, q : p і q повинні бути простими числами;

– обчислення: n = p x q:

– обчислення: φ(n) = (p – 1) (q – 1):

– вибір цілого e: gcd φ(n), e) = 1, 1 < e < φ(n));

– обчислення d: d = e-1 mod φ(n);

– відкритий ключ: KU = {e, n};

– особистий ключ: KR = {d, n}/

Шифрування:

– відкритий текст: M < n;

– шифрований текст: З = M2 (mod n).

Дешифрування:

– відкритий текст: З;

– шифрований текст: M = C2 (mod n).

Проведемо експериментальні розрахунки.

Розділити натуральне число n на натуральне число m із залишком означає підібрати невід'ємні цілі числа q і r , що задовольняють двом умовам:

$$n = qm + r \qquad r < m.$$

Натуральне число називається простим, якщо воно ділиться тільки на себе і на одиницю. Перші 30 простих чисел наступні: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 107, 109, 113.

Якщо при діленні на к є N цілі числа n та m, які дають той с амий залишок, то n і m порівнювані по модулю к, це значить, що n = m mod к.

Запишемо: 2 = 17 mod 15. 21 = 1 mod 10. 13 = 9 mod 2.

Два натуральні числа, які не мають спільних, відмінних від одиниці дільників, називаються взаємно простими. Числа 12 и 25 взаємно прості (рисунок 4.2).

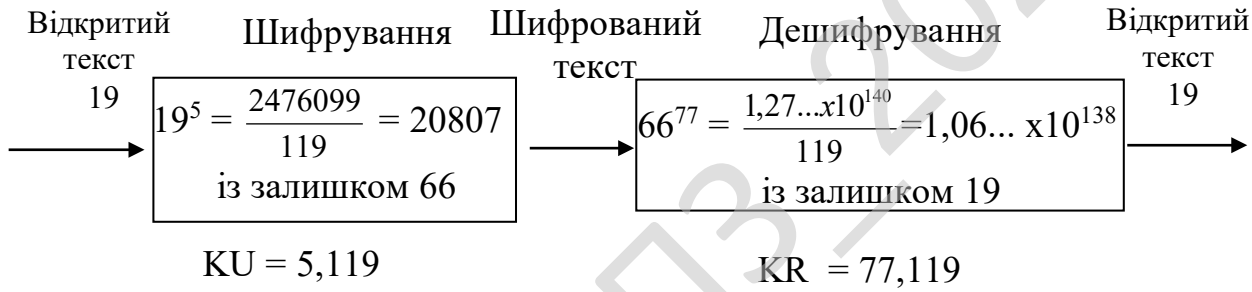

Рисунок 4.2 – Структура взаємно простих чисел

При шифруванні електронних файлів можна кілька разів кодувати симетричний ключ за допомогою різних відкритих ключів. Кожен відкритий ключ належить одному з одержувачів, тому кожний з них зможе розшифрувати файл. Кожен відкритий ключ являє собою “конверт”, що містить той самий ключ для декодування вихідного повідомлення. Ця концепція безпечного обміну інформацією використовується, зокрема, у PGP, а також – аналогічним чином – у кодуванні по алгоритму SSL у Web.

Найбільш істотною перевагою відкритого ключа в криптографії перед закритим полягає в тому, що при його застосовуванні в парі з відкритими, особисті ключі ніколи не висилаються. Завдяки цьому, зазначений тип криптографії забезпечує більш високий рівень таємності і зручності. У системах із закритим ключем ключі підлягають передачі, що дає досить високий ступінь

ризику. Крім того, наявність у системі таких ключів затрудняє реалізацію механізму автентифікації. Система з відкритим ключем дозволяє використовувати для цього цифровий підпис, а в ситуації із закритим ключем для запобігання відмови однієї зі сторін необхідна участь третьої сторони, на яку покладається задача підтвердження автентичності сторін-учасників.

Що стосується недоліків системи з відкритим ключем, то вони зводяться в основному до того, що створені з її застосуванням алгоритми, значно поступаються у швидкості технологіям на основі закритого ключа. Останні швидші, як мінімум, у кілька разів. Отже, кодування з в ідкритим ключем не підходить для великих файлів. Тому варто використати комбінацію обох методів.

При такій комбінації повідомлення шифрується закритим ключем (тому, що на шифрування відкритим ключем іде занадто багато часу) і закритий ключ приєднується до повідомлення, але шифрується відкритим ключем. Такий спосіб одночасно прискорює процес і підвищує безпеку.

Кожний криптопровайдер має свою БД, в якій зберігаються довгочасові ключі користувачів. БД утримує один або більше контейнерів ключів, як це показано на рисунку 4.3.

БД в процесі виконання ВКРМ розроблятись не буде, оскільки використаємо БД вбудованого механізму криптографії ОС WINDOWS. Таким чином, користувач може створити будь-яку кількість контейнерів з різними іменами.

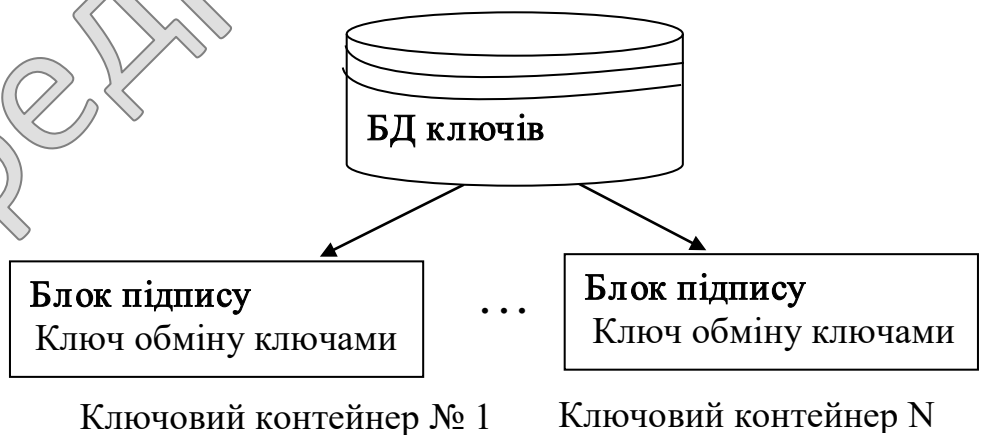

Рисунок 4.3 – Структура БД криптопровайдера

Підключення до контейнеру проводиться одночасно з одержанням контекста криптопровайдера при виклику функції CryptAcguireContext – ім'я контейнера ключів передається функції другим її аргументом. Якщо цей аргумент утримує пустий покажчик (nil), то використовується ім'я реєстрації у користувача в системі.

Для закінчення доступу до контейнера, в останньому аргументі функції передамо прапорець CRYPT VERIFYCONTEXT; для створення нового контейнера використаємо прапорець CRYPT_NEWKEYSET; а для видалення існуючого контейнера (разом з ключами) – CRYPT_DELETEKEYSET.

При цьому вводимо наступні обмеження:

– контейнер може утримувати, як мінімум, дві ключові пари: ключ обміну ключами та ключ підпису;

– ключі, що використовуються для шифрування симетричними алгоритмами, не зберігаються (адже це одноразові, тобто, сеансові ключі).

Після створення контейнера ключів необхідно згенерувати ключові пари обміну ключами та ЕЦП. Для цього використаємо функцію CryptGenKey

(провайдер, алгоритм, прапорці, ключ):

– провайдер – дескриптор криптопровайдера, одержаний в результаті звернення до функції CryptAcguireContext;

– алгоритм – вказує, якому алгоритму шифрування буде відповідати створюваний ключ: кожний провайдер використовує для обміну ключами та ЕЦП строго визначені алгоритми. Ми, використавши провайдер PROV_RSA_FULL, одержимо для реалізації вищеозначених функцій алгоритм RSA;

– прапорці – при створення асиметричних ключів керують їх розміри. Криптопровайдер, який ми використовуємо, дозволяє генерувати ключ обміну ключами довжиною від 384 до 512 біт, в ЕЦП – від 512 до 16384 біт. Чим більше довжина ключа, тим вища ступінь надійності. Якщо користувач не задає довжину ключів, по замовченню – їх розмір буде дорівнювати 512 біт;

– ключ – якщо функція завершена успішно, в цей параметр заноситься дескриптор створеного ключа.

Після створення набору ключів, необхідно забезпечити можливість їх обміну з іншими користувачами (маються на увазі відкриті ключі). Для цього необхідно вилучити їх з БД ключів та записати в файл, який вже можна передавати респондентам.

Експорт ключів виконаємо за допомогою функції CryptExportKey:

(експортований ключ, ключ адресата, формат, прапорці, буфер, розмір буфера):

– експортований ключ-дескриптор потрібного ключа;

– ключ адресата – в випадку збереження відкритого ключа дорівнює нулю, тобто, дані не шифруються;

– формат – вказує на один з можливих форматів експорту (PUBLICKEYBLOB – збереження відкритих ключів в незашифрованому вигляді/ PRIVATEKEYBLOB – збереження ключової пари/SIMPLBLOB – збереження сеансових ключів);

– прапорці – зарезервовані на майбутнє;

– буфер – утримує адрес буфера, в який буде записаний ключовий BLOB;

– розмір буфера – при виклику буфера ця зміна утримує доступний розмір буфера, а по закінченню роботи – кількість експортованих даних.

Експорт ключової пари в цілому, включаючи і закритий ключ, необхідний і в наступних випадках: підпис документів на ПК різного призначення (наприклад, дома і на роботі), для збереження страхової копії, перевірки ЕЦП, дешифровки сеансового ключа, тощо.

Імпорт ключових пар в тільки згенерований контейнер – це самостійна процедура. Для цього необхідно одержати від користувача наступні реквізити: назву контейнера, пароль. Після цього підключитись до провайдера, створити на основі пароля ключ, зчитати з файла імпортовані дані в буфер, після чого використати функцію CryptImportKey:

(провайдер, буфер, довжина, ключі дешифрування, прапорці, імпортований ключ).

Якщо в подальшому є необхідність в експортуванні імпортованої ключової пари, то в параметри прапорця необхідно передати значення CRYPT_EXPORTABLE. Якщо це не буде зроблено, виникне помилка.

При роботі з асиметричними алгоритмами необхідно ввести до складу ПЗ функцію контролю для визначення автентичності ключа. Для цього використаємо побайтову перевірку одержаного ключа на його відповідність з оригіналом, що знаходиться в хазяїна. Відповідно, результати перевірки будуть виведені в вікно перегляду.

Для створення електронного цифрового підпису необхідно обчислити хеш заданого файлу і зашифрувати цей "цифровий відбиток повідомлення" своїм закритим ключем – "підписати". Щоб підпис згодом можна було перевірити, необхідно вказати, який алгоритм хешування використовувався при його створенні. Тому підписане повідомлення буде мати структуру, означену рисунком 4.4.

Щоб перевірити правильність підпису, одержувач підписаного повідомлення повинен мати файл з відкритим ключем підпису відправника. В процесі перевірки підпису цей ключ імпортується всередину криптопровайдера.

Асиметричні алгоритми дозволяють легко обмінюватись ключами шифрування по відкритих каналах зв'язку. Але мають суттєвий недолік – працюють дуже повільно.

Симетричні алгоритми працюють швидко, але потребують наявності захищеного каналу зв'язку і в досить частої заміни ключів. Тому при розробці ПЗ було використано сильні сторони обох підходів: для шифрування повідомлень використовується симетричний алгоритм з випадковим ключем шифрування, який задіяний впродовж сеансу – сеансовим ключем; для дешифровки повідомлення зашифрований сеансовий ключ зберігається разом з повідомленням, створюючи таким чином цифровий конверт (сеансовий ключ шифрується асиметричним алгоритмом з використанням відкритого ключа одержувача повідомлення). При необхідності цифровий конверт може

утримувати сеансовий ключ в декількох примірниках – зашифрований відкритими ключами різних одержувачів, як це показано на рисунку 4.4.

Рисунок 4.4 – Структура цифрового конверта

Система, побудована по такому принципу, буде виконувати всі означені ТЗ функції. Тому, виконавши теоретичну побудову основних алгоритмів системи, переходимо до їх програмної реалізації.

Розроблене ПЗ має модульну структуру побудови, що забезпечує досягнення системою високої ступені мобільності та адаптивності.

До складу ПЗ ввійшли головна програма, яка по суті є диспетчером системи, та програми (модулі), які забезпечують виконання функцій головною програмою.

Розглянемо блок-схему роботи алгоритму головної програми системи (рисунок 4.5). Вона забезпечує в системі:

- керування модулями;
- роботу потужного графічного інтерфейсу;
- обробку помилок;
- організацію довідника користувача.

Одержавши доступ в систему, користувач може вибрати один з трьох режимів роботи системи по захисту файлів чи їх дешифрування.

Розвинений інтерфейс користувача забезпечує високий ступінь візуалізації роботи системи.

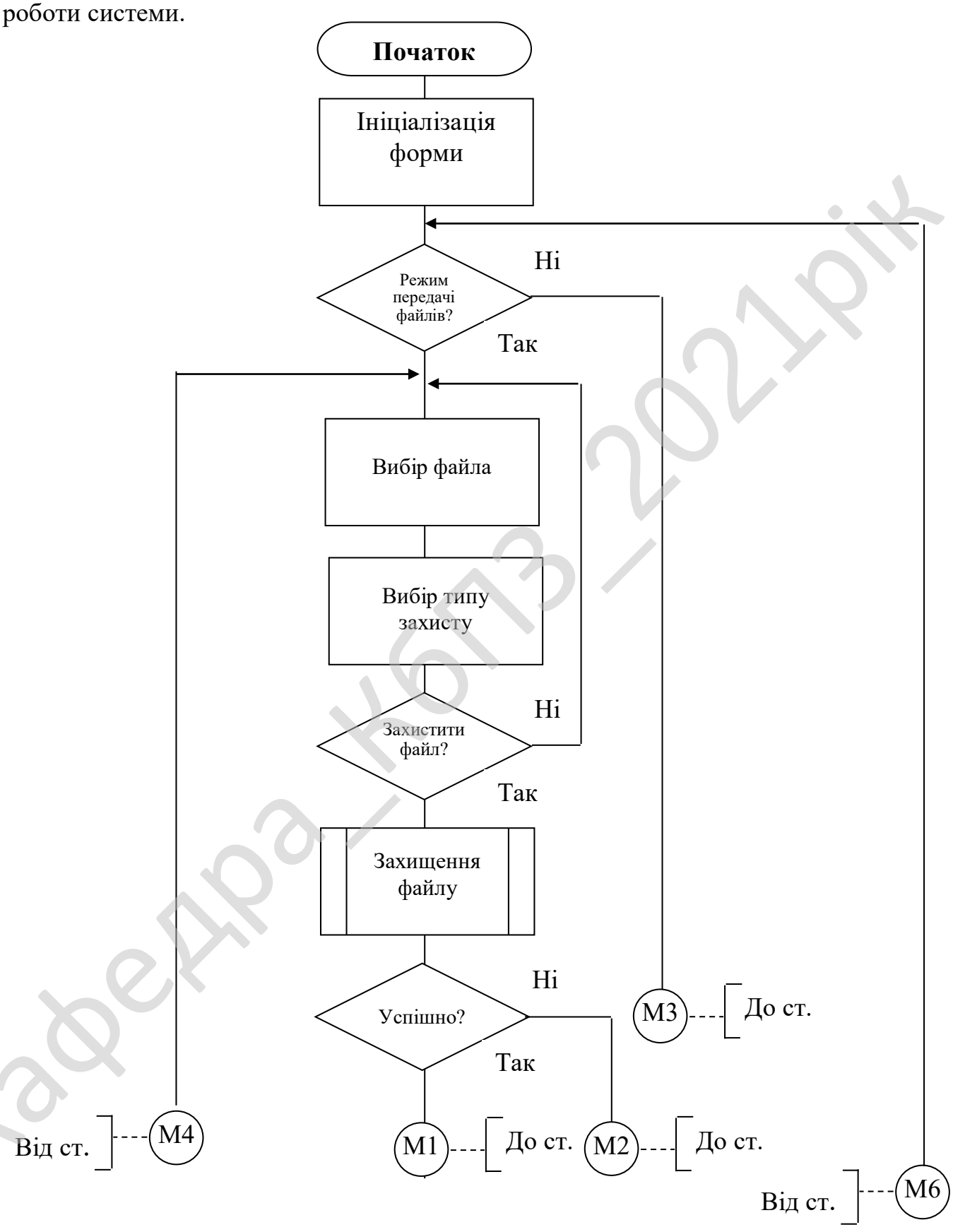

Рисунок 4.5 – Блок-схема алгоритму роботи головної програми

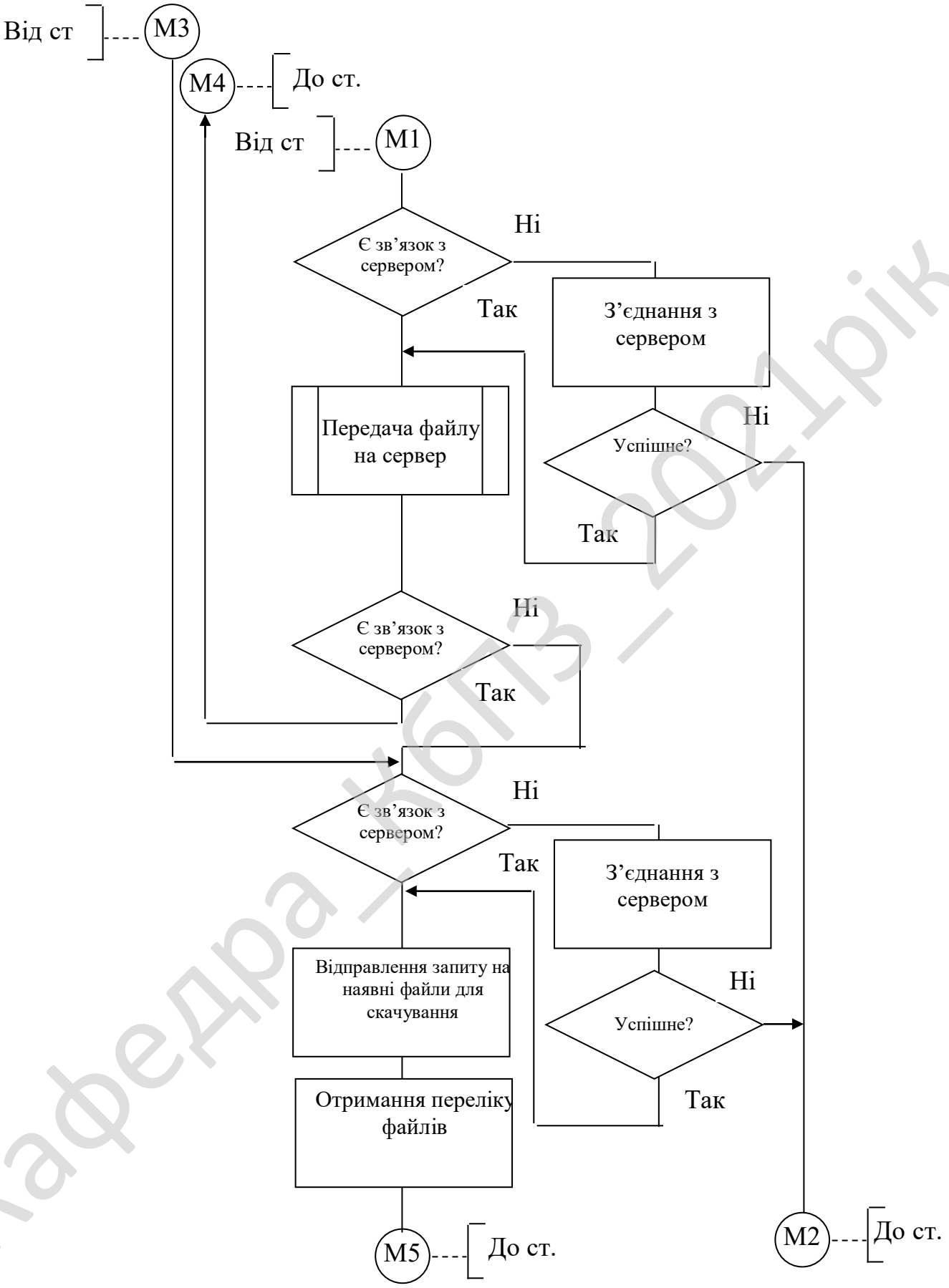

Рисунок 4.5, аркуш 2

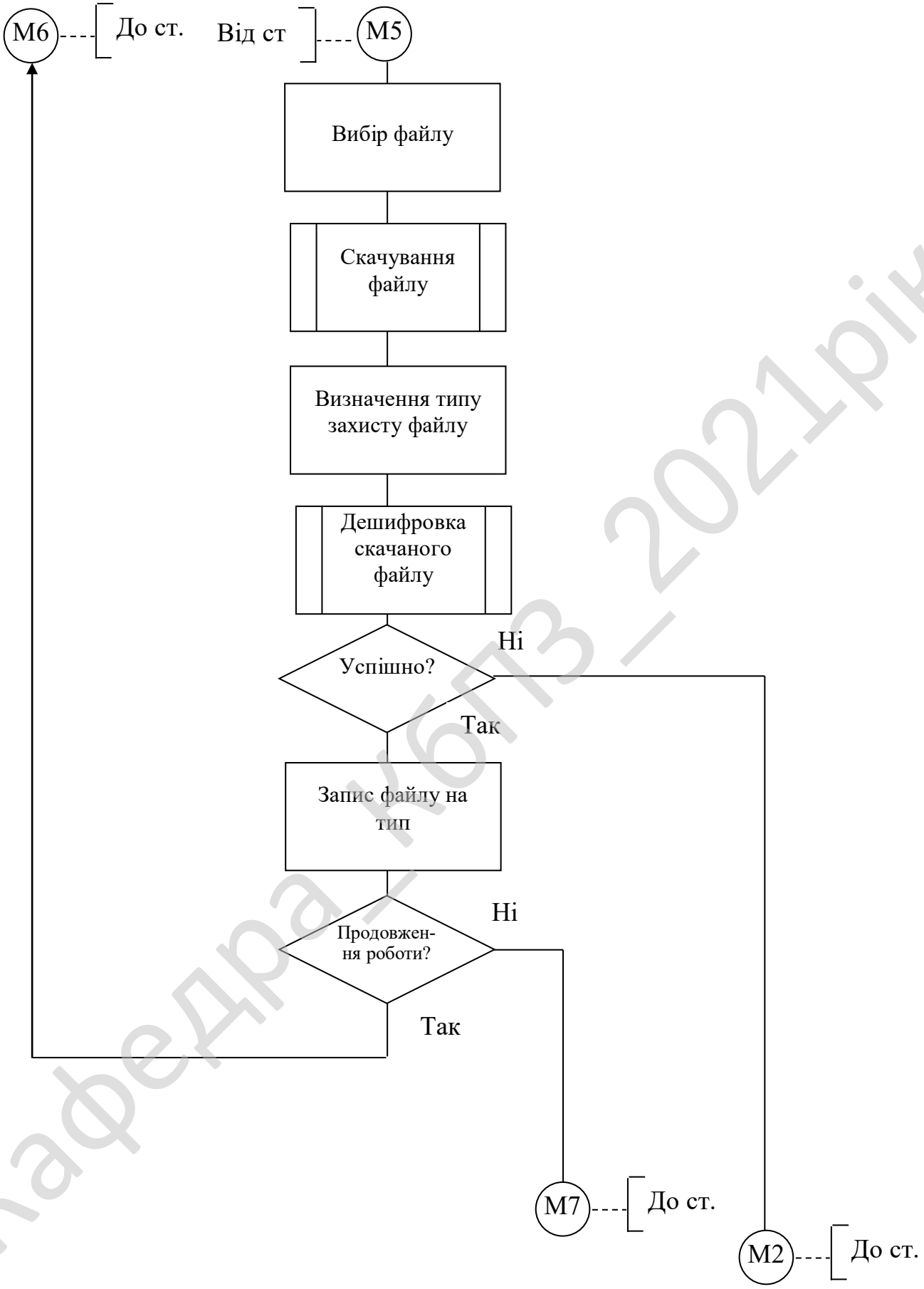

Рисунок 4.5, аркуш 3

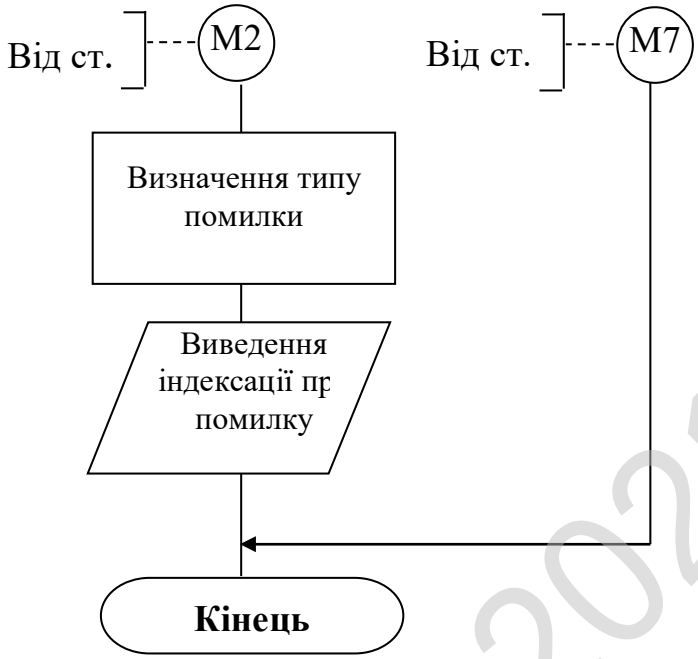

Рисунок 4.5, аркуш 4

Для економії обчислювальних ресурсів при розробці ПЗ використане динамічне розподілення пам’яті. Ті змінні або об’єкти, які вже стають непотрібними, знищуються, а звільнене місце використовується для нових змінних/об’єктів. Для динамічного розподілення виділена спеціальна область пам’яті – heap, а для розподілення – використана процедура GetMen та процедура FreeMen – для її звільнення. Вони мають наступний синтаксис:

```
procedure GetMen (<ім’я покажчика>,
                  <обсяг пам’яті в байтах>);
procedure FreeMen (<ім’я покажчика>,
                  <обсяг пам’яті в байтах>);
```

На відміну від процедур New і Dispoge, ми задаємо не тільки покажчик, в якому встановлюється процедурою GetMen та читається процедурою FreeMen адреса виділеної області пам’яті, але й вказуємо обсяг пам’яті в байтах. Це дозволяє використати як типізовані, так і не типізовані покажчики.

Більш доцільно використати типізовані покажчики, адже вони дозволяють врахувати аспект зміни розміру пам’яті в різних версіях компілятора, використавши функцію SizeOf. Таким чином, будемо мати:

```
GetMen (P, SizeOf (real);
Free Men (P, SizeOf (real);
GetMen (Pr, SizeOf (rec);
Free Men (Pr, SizeOf (rec);
```

де P, Pr – змінні, що є покажчиками на дійсні значення.

В основу побудованої системи покладено використання функцій криптографічна системи CryptoAPI ОС Windows. Код функцій CryptoAPI в ОС утримується в декількох динамічно завантажуваних бібліотеках. Для звернення до таких функцій з прикладних програм, використаємо їх оголошення в програмному коді системи як зовнішніх, а заголовок функції в інтерфейсній частині модуля сформуємо наступним чином:

```
function CryptAcquireContexst
ph PROV: PHCRYPTPROV;
pszContainer: LPCTSTR;
pszpROVIDER: LPCTSTR;
dwProvType: DWORD;
dwFlags: DWORD): BOOL; stdcall.
```

В виконавчу частину замість тіла функції впишемо директиву extern з означенням бібліотеки, в якій утримується функція і її ім’я, тобто:

```
function CryptAcquireContexst; external `advapi32.dll`
name `CryptAcquireContexstA`;
```

Таким чином, маючи опис функції CryptoAPI, можна зібрати заголовки функцій в окремому модулі, який буде забезпечувати взаємодію прикладної програми з криптографічною підсистемою ОС. Але ми не будемо виконувати цю роботу, а використаємо вже готові заготовочні файли (wincrypt.h) для підключення модуля до проекту. Це дозволяє використати не тільки самі функції CryptoAPI, але й мнемонічні константи режимів, ідентифікатори алгоритмів та інші параметри, необхідні для написання програмного коду.

Одним з основних ключових моментів розробки ПЗ системи є створення ЕЦП, блок-схема алгоритму якого наведена на рисунку 4.6.

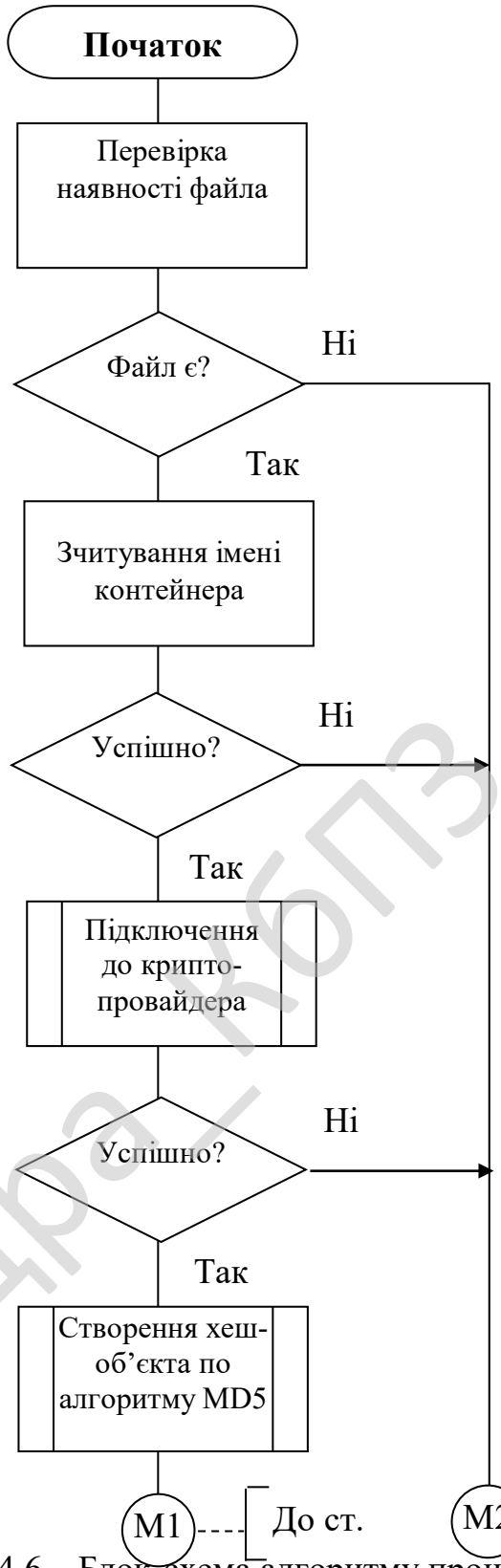

Рисунок 4.6 – Блок-схема алгоритму процедури підпису файлу

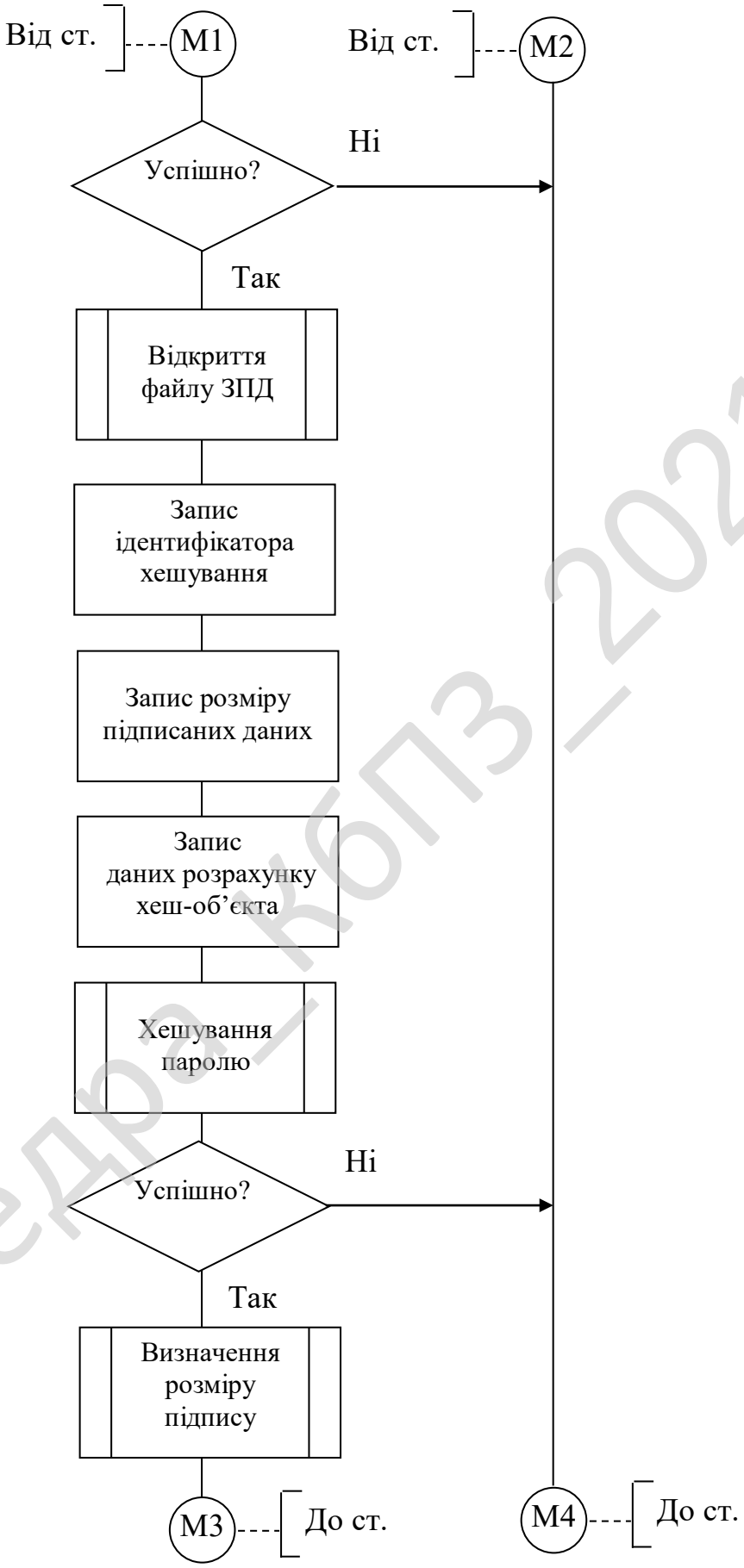

Рисунок 4.6, аркуш 2

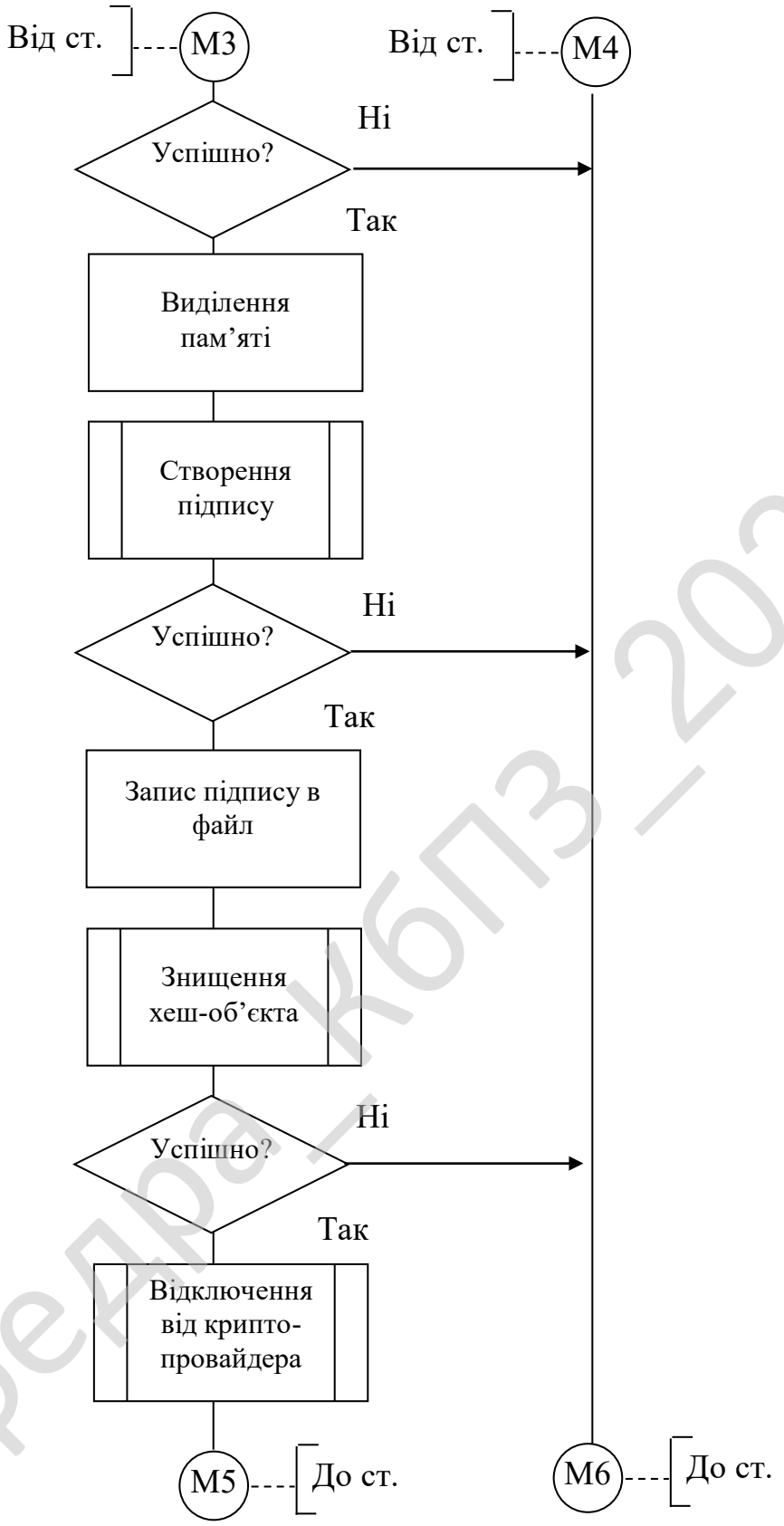

Рисунок 4.6, аркуш 2

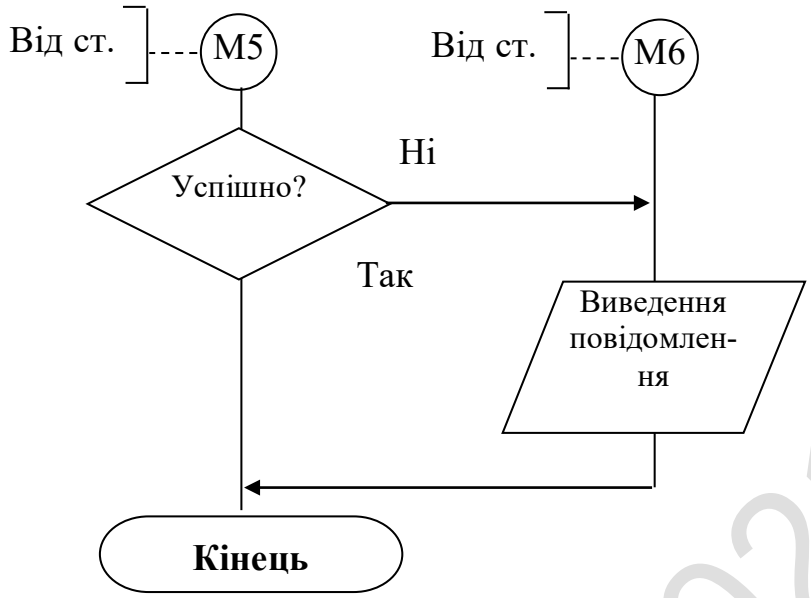

Рисунок 4.6, аркуш 3

Алгоритм роботи вищеозначеної процедури.

Крок 1: початок роботи. Перехід на крок 2.

Крок 2: перевірка на наявність вибраного файлу, який підлягає захисту. Якщо файл є – перехід на крок 3, якщо ні – виведення відповідного повідомлення та вихід на кінець роботи процедури.

Крок 3: зчитування з файла конфігурацій імені контейнера ключів. Перехід на крок 4.

Крок 4: перевірка успішності зчитування. Якщо файл конфігурацій не існує чи порушена його структура – виведення повідомлення та вихід на кінець роботи програми, якщо помилки відсутні – перехід на крок 5.

Крок 5: підключення до криптопровайдера: отримання його контексту, підключення до вибраного контейнера ключів. Перехід на крок 6.

Крок 6: обробка помилок при роботі з криптопровайдером. Якщо помилки є – виведення повідомлення і вихід на кінець роботи програми, якщо ні – перехід на крок 7.

Крок 7: визначення алгоритму хешування – MD5 та створення хеш-об’єкта. Перехід на крок 8.

Крок 8: обробка помилок криптопровайдером. Якщо помилки є – виведення повідомлення і вихід на кінець роботи програми, якщо ні – перехід на крок 9.

Крок 9: відкриття файлу, обраного користувачем для зберігання підписаних даних (ЗПД). Перехід на крок 10.

Крок 10: запис ідентифікатора алгоритму хешування в обраний файл. Перехід на крок 11.

Крок 11: запис розміру підписаних даних. Перехід на крок 12.

Крок 12: запис даних та розрахунки хеш-об'єкта. Перехід на крок 13.

Крок 13: хешуємо пароль. Перехід на крок 14.

Крок 14: обробка помилок при роботі з криптопровайдером. Якщо помилки є – виведення повідомлення і вихід на кінець роботи програми, якщо ні – перехід на крок 15.

Крок 15: визначення розміру підпису. Перехід на крок 16.

Крок 16: обробка помилок при роботі з криптопровайдером. Якщо помилки є – виведення повідомлення і вихід на кінець роботи програми, якщо ні – перехід на крок 17.

Крок 17: виділення динамічної пам'яті. Перехід на крок 18;

Крок 18: обробка помилок при роботі з криптопровайдером. Якщо помилки є – виведення повідомлення і вихід на кінець роботи програми, якщо ні – перехід на крок 19.

Крок 19: запис підпису у файл. Перехід на крок 20.

Крок 20: знищення хеш-об'єкта пам'яті. Перехід на крок 21.

Крок 21: обробка помилок при роботі з криптопровайдером. Якщо помилки є – виведення повідомлення та вихід на кінець роботи програми, якщо ні – перехід на крок 22.

Крок 22: відключення від криптопровайдера. Перехід на крок 23.

Крок 23: обробка помилок при роботі з криптопровайдером. Якщо помилки є – виведення повідомлення і вихід на кінець роботи програми, якщо ні – перехід на крок 24.

Крок 24: кінець роботи програми.

Одним із основних алгоритмів системи, реалізованих в процесі її розробки, є створення ключів та їх шифрування. Після проведення шифрування ключів та їх запису до файлу, для організації подальшої процедури обміну ключами між користувачем системи, до складу розробленого ПЗ введена процедура експортування ключів. Експортуванню підлягають наступні ключі: ключ обміну ключами, ключ підпису.

В якості ілюстрації наводимо фрагмент програмного коду вищеозначеної процедури:

```
procedure TFormLicense.KeyExportVidkr(lv_NameConteiner: String);
{Експортування ключів}
var cont: PChar;
    err: string;
    hProv: HCRYPTPROV;
    key, expKey: HCRYPTKEY;
    pbuf: PBYTE;
    buflen: DWORD;
    f: file;
    lv_f: TextFile;
    hash: HCRYPTHASH;
begin
{ім'я контейнера}
  err := lv_NameConteiner;
cont := StrAlloc(length(err) + 1);
StrPCopy(cont, err);
{Отримуємо контекст криптопровайдера (підключаємося до криптопровайдера)}
if not CryptAcquireContext(@hProv, cont, nil, PROV_RSA_FULL, 0)
then
{обробка помилок}
    begin
        MessageDlg('Помилка при роботі з криптопровайдером: ' +
ErrToStr(GetLastError) +
                   '. Зверніться до розробника! , mtError, [mbOK], 0);
        exit;
       end;
{Експортувати *ключ обміну ключами*}
 repeat
{отримуємо дескриптор ключа}
   if not CryptGetUserKey(hProv, AT_KEYEXCHANGE, @key) the
      begin
{обробка помилок}
        MessageDlg('Помилка при роботі з криптопровайдером ' +
ErrToStr(GetLastError) +
                   '. Зверніться до розробника' , mtError, [mbOK], 0);
        exit;
        end;
```

```
       {визначаємо розмір буфера для експорту ключа}
          {$B-}
          if not CryptExportKey(key, 0, PUBLICKEYBLOB, 0, nil, @bufLen) then
             begin
               {обробка помилок}
               MessageDlg('Помилка при роботі з криптопровайдером: ' +
ErrToStr(GetLastError) +
                          '. Зверніться до розробника ‘ , mtError, [mbOK], 0);
               exit;
             end;
          GetMem(pbuf, bufLen);
       {експортуємо дані}
          if not CryptExportKey(key, 0, PUBLICKEYBLOB, 0, pbuf, @bufLen) then
             begin
               {обробка помилок}
               MessageDlg('Помилка при роботі з криптопровайдером: ' +
ErrToStr(GetLastError) +
                          '. Зверніться до розробника' , mtError, [mbOK], 0);
               exit;
             end;
       {Знищуємо дескриптор *ключа обміну ключами*, при цьому сам ключ не
знищується}
          if not CryptDestroyKey(key) then
             begin
       {обробка помилок}
               MessageDlg('Помилка при роботі з криптопровайдером ' +
ErrToStr(GetLastError) +
                          '. Зверніться до розробника' , mtError, [mbOK], 0);
               exit;
             end;
             AssignFile(lv_f, 'session.log');
             Append(lv_f);
             Writeln(lv_f, DateToStr(Now)+' '+TimeToStr(Now));
             Writeln(lv_f,   'Збереження ключа обміну ключами в файл:
key_vidk.txt');
       {зберігаємо ключ обміну ключами в зовнішньому файлі}
             AssignFile(f, 'key_vidk.txt');
             rewrite(f, 1);
             BlockWrite(f, pbuf^, bufLen);
             CloseFile(f);
             Writeln(lv_f, 'Ключ обміну ключами успішно збережений);
             CloseFile(lv_f);
        until true; {KeyExchange}
```

В наведених основних модулях та процедурах програмного забезпечення підтримується принцип інформативності, а саме: надання користувачеві інформації про виконання встановлених перевірок та результати роботи окремих програм та підпрограм, надання в разі необхідності теоретичних відомостей щодо принципів роботи системи, необхідних для роботи технічних параметрів, тощо. Інформаційні повідомлення в процесі роботи системи виводяться на екран монітора чи роздруковуються (по бажанню користувача).

Таким чином, завдання визначене ТЗ та постановкою задачі, виконане в повному обсязі. Основні етапи розробки ПЗ системи включають:

– розробка структурної і функціональної схем системи для визначення та обґрунтування вибору структурних частин ПЗ та формування функцій;

– теоретичне обґрунтування розробки основних рішень побудови програмного забезпечення та виконання необхідних розрахунків;

– розробка алгоритмів та ПЗ системи.

Розроблене ПЗ має структуру, що забезпечує виконання нею принципів мобільності та адаптивності і цілком дієспроможне. Про це свідчать результати дослідної експлуатації. Розроблено досить потужну систему з додержанням усіх вимог до систем даного класу, яка в подальшому може вдосконалюватись шляхом введення нових функцій – сервісних та робочих. Подальший рекомендований напрямок вдосконалення-використання сертифікатів для підвищення ступені захисту ПЗ при обміні відкритими ключами, тощо.

4.2 Захист розробленого програмного забезпечення

Розроблене програмне забезпечення захистимо за допомогою алгоритму захисту інформації RSA. Спочатку необхідно обчислити пару ключів (секретний ключ і відкритий ключ). Для цього відправник електронних документів обчислює два більших простих числа P и Q, потім знаходить їхній добуток $N = P*Q$ і значення функції $\varphi(N) = (P-1)(Q-1)$. Далі відправник обчислює число E з умов $E < \varphi(N)$, НЗД $(E, \varphi(N)) = 1$ і число D з умов $D < N$, $E*D \equiv 1 (\mathrm{mod}\ \varphi(N))$.

Пари чисел (E, N) є відкритим ключем. Цю пару чисел автор передає партнерам по переписці для перевірки його цифрових підписів. Число D зберігається автором як секретний ключ для підписування.

Допустимо, що відправник хоче підписати повідомлення M перед його відправленням. Спочатку повідомлення M (блок інформації, файл, таблиця) стискають за допомогою геш-функції h(-) у ціле число m: $m = h(M)$.

Потім обчислюють цифровий підпис S під електронним документом M, використовуючи геш-значення m і секретний ключ D: $S = m \pmod N$.

Пари (M, S) передається партнерові-одержувачеві як електронний документ M, підписаний цифровим підписом S, причому підпис S сформований власником секретного ключа D.

Після прийому пари (M, S) одержувач обчислює геш-значення повідомлення M двома різними способами. Насамперед, він відновлює геш-значення m', застосовуючи криптографічне перетворення підпису S з використанням відкритого ключа E: $m' = S^E \pmod N$.

Крім того, він знаходить результат гешування прийнятого повідомлення M с допомогою такої ж геш-функції h(-): $m = h(M)$.

Якщо дотримується рівність обчислених значень, тобто $S^E(\text{mod}) = h(M)$, то одержувач визнає пару (M, S) справжньою. Доведено, що тільки власник секретного ключа D може сформувати цифровий підпис S по документі M, а визначити секретне число D по відкритому числу E не легше, ніж розкласти модуль N на множники. Крім того, можна строго математично довести, що результат перевірки цифрового підпису S буде позитивним тільки в тому випадку, якщо при обчисленні S був використаний секретний ключ D, що відповідає відкритому ключу E. Тому відкритий ключ E іноді називають "ідентифікатором" того, хто підписав.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Згідно з ТЗ в процесі роботи над ВКРМ було розроблене ПЗ системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Розроблена система може бути впроваджена в промислову експлуатацію в структурах будь-якої сфери діяльності людини, де використовується електронний документообіг, який потребує захисту під час транспортування в мережі: локальній чи глобальній. Впровадження любого побічного інструментарію, в тому числі і захисту програмного коду, супроводжується складностями. Вони пов'язані з тим, що, окрім вивчення користувачами питань безпеки інформації, необхідно розширити коло обов'язків супровідників ПЗ та служби технічної підтримки.

Персоналу, що буде супроводжувати ПЗ, необхідно вивчити функції системи, представлені наявною технічною документацією на рівні, що буде достатнім для реалізації функцій і задач інтеграції розробленої системи захисту в наявне програмне забезпечення. Це є досить складною задачею, адже потрібно вбудовувати захист в вихідні тексти програм. Це потягне за собою витрати на додаткове тестування і появу нової версії продукту. Але ця задача середньої складності і, при наявності в організації кваліфікованих розробників ПЗ, вона підлягає вирішенню в термін 3-4 дні.

З ціллю здешевлення загальної вартості впровадження, доцільно провести пробне навчання спеціалістів Замовника і виконати пілотний проект.

Впровадження системи може бути ефективним при умові наявності методичного (МЕТОЗ) та організаційного забезпечення (ОРЗ) системи, що являє собою правила відбору та експлуатації засобів забезпечення ефективної роботи системи.

МЕТОЗ безпосередньо взаємодіє з інформаційним, програмним та технічним забезпеченням системи на етапі промислової експлуатації.

МЕТОЗ на стадії впровадження системи в промислову експлуатацію – це комплекс документів, що описують принципи організації впровадження системи і утримують правила та інструкції по використанню ПЗ, а саме:

– інструкції по використанню програмного забезпечення;

– правила та інструкції по забезпеченню виконання вимог ТБ та промсанітарії при експлуатації системи;

– інструкції користувача, який буде працювати з системою;

– документи з описанням маршруту організації роботи замовника в умовах функціонування системи;

– таблиці нормативної довідкової інформації;

– журнали для реєстрації роботи користувачів, обліку машинного часу, обліку проведених регламентних робіт та інші;

– штатний розклад обслуговуючого персоналу;

– посадові інструкції обслуговуючого персоналу;

– інструкції по супроводженню промислової експлуатації програмного забезпечення;

– заходи щодо впровадження передових методів праці, планів наукової організації праці, принципів морального та матеріального стимулювання робітників.

Організаційне забезпечення – це перелік документів, що регламентують взаємодію та функції підрозділів замовника та матеріально-технічного забезпечення системи, а саме:

– накази та розпорядження по регламентації взаємодії підрозділів, які приймають участь в експлуатації системи;

– накази та розпорядження, які визначають функції підрозділів по експлуатації та обслуговуванню ПЗ та технічного обладнання;

– затверджений графік постачання матеріалів для проведення

регламентних робіт ЗОТ та запасних частин для їх ремонту;

– графіки регламентних робіт.

В процесі дослідної експлуатації системи було доведено, що розроблене ПЗ цілком працездатне і реалізовує поставлені ТЗ задачі в повному обсязі, що підтверджується скриншотами (рисунки 5.1, 5.2, 5.3, 5.4, 5.5).

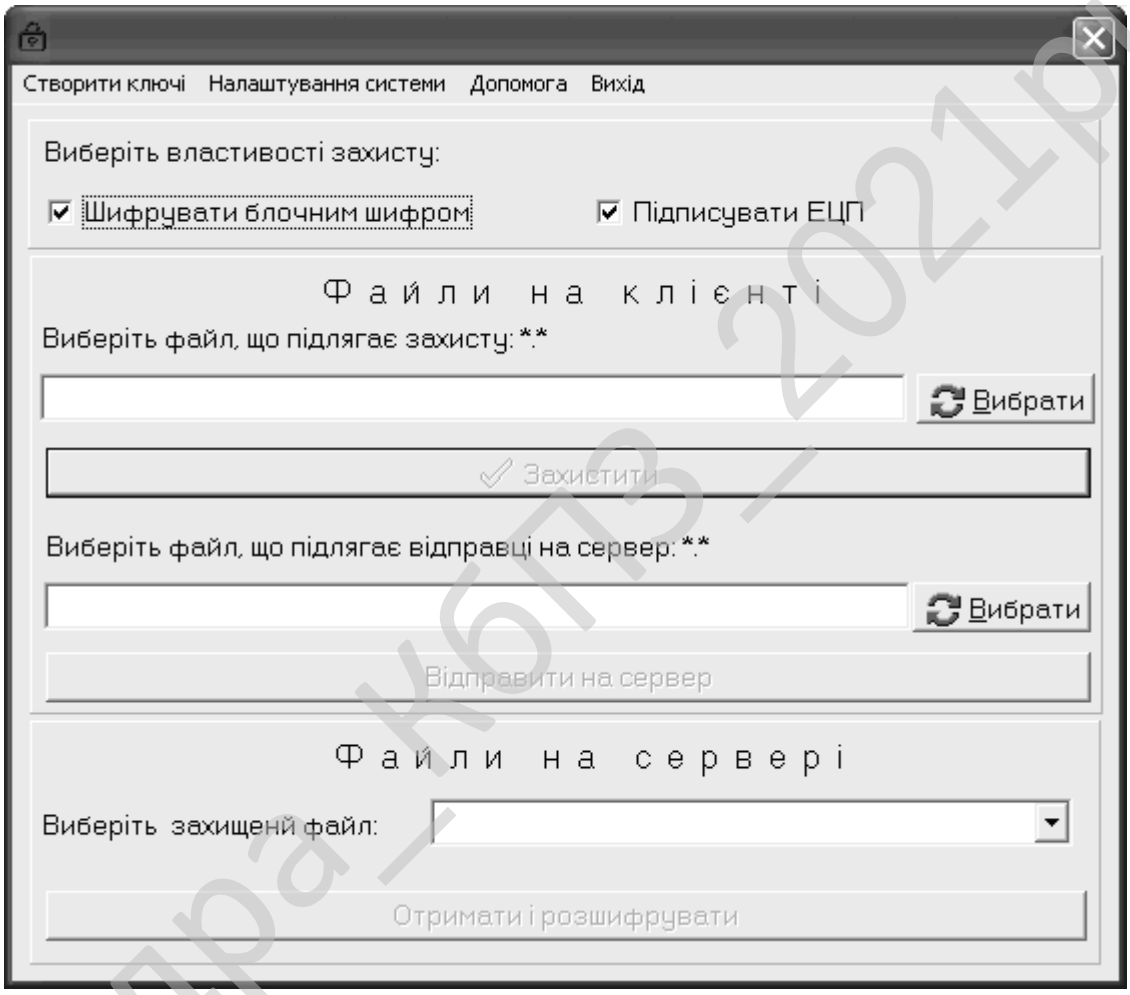

Рисунок 5.1- Головна форма клієнтської частини програми

Методичне та організаційне забезпечення забезпечують виконання плану впровадження. План впровадження передбачає організацію підготовки кадрів, експлуатаційного персоналу та аналіз функціонування системи після виконання всіх робіт із її впровадження і встановлює: обсяги фінансування проектних і

налагоджувальних робіт; терміни постачання технічних засобів та їх монтажу; терміни виконання пуско – налагоджувальних робіт, тощо.

Експлуатацію розробленої системи буде здійснювати персонал підрозділів і відповідних служб організації-замовника, які мають забезпечити виконання наступних задач: технічне супроводження; диспетчеризацію роботи обслуговуючого персоналу в період роботи системи по безпосередньому призначенню; системне забезпечення; супроводження програмних компонентів; обслуговування технічних засобів; організацію робіт по розвитку системи.

Служба, що безпосередньо буде здійснювати експлуатацію розробленої системи, повинна розробити та реалізувати заходи по підготовці організації до введення в дію системи, розробити програми підготовки спеціалістів-користувачів, заходи щодо впровадження системи в промислову експлуатацію.

Рисунок 5.2 – Форма авторського права

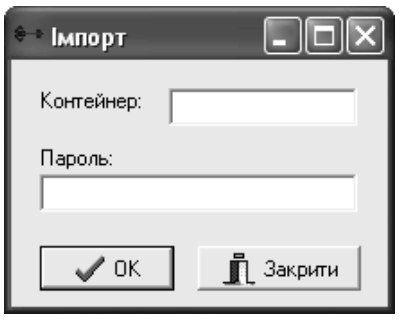

Рисунок 5.3 – Форма імпорту ключів

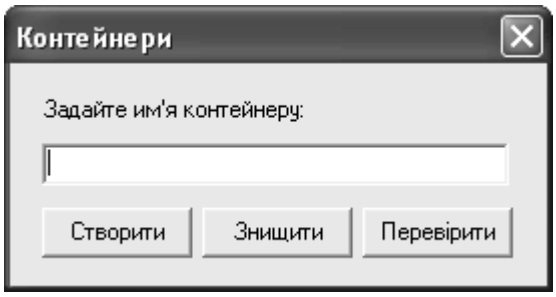

Рисунок 5.4 – Форма створення контейнера

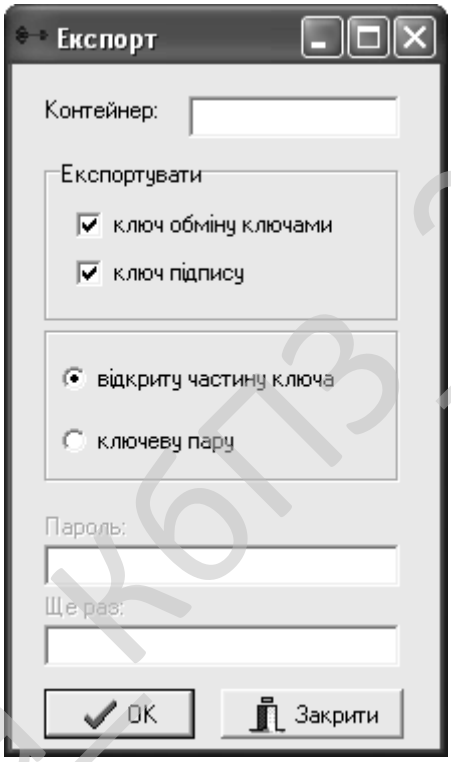

Рисунок 5.5 – Форма експорту ключів

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Метою розробки є дослідження та програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Об'єктом дослідження є процес кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Предметом дослідження є методи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Методи дослідження базуються на методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

– Розроблено вітчизняний продукт кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API, який має більш широкі можливості, на відміну від існуючих аналогів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці).

В магістерській роботі було проведене дослідження та виконана програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API. Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт	N	1
2. Кількість екземплярів програм, шт	Ne	50 (2 ост. цифри № зал*10^1)
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2
7. Кількість макетів вхідної інформації	–	3

Продовження табл. 7.1

1	2	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	1
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	_	2
13. Рівень спрацьованості колективу (1-6)	_	2
14. Ступінь вимірності процесів (1-6)	_	3
15. Необхідна надійність програмного забезпечення (1-6)	_	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	_	2
17. Складність кінцевого програмного продукту (1-6)	_	2
18. Необхідний рівень забезпечення повторного використання (1-6)	_	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	_	2
20. Вимоги до швидкодії ПП (1-6)	_	2
21. Обмеження на розміри основного сховища даних (1-6)	_	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	_	2
23. Професійний рівень аналітиків (1-6)	_	2
24. Професійний рівень програмістів (1-6)	_	2
25. Постійність складу команди розробників (1-6)	_	2
26. Досвід розробки додатків (1-6)	_	2
27. Досвід роботи з обчислювальною платформою (1-6)	_	2

Продовження табл. 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПО для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн	–	50000 (2 ост. цифри № зал*10^4)
33. Норматив додаткової зарплати, % :	Нд	10
34. Норматив відрахувань у соціальні фонди, %	Нс	37
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	55
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що роб ить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \, \text{Size}^B , \qquad (7.1)$$

де A – коефіцієнт Боема, A=2,45; Size – загальний об'єм відлагодженого програмного коду, тис. рядків; B – показник ступеня, що визначається співвідношенням

$$B = 1{,}01 + 0{,}001\sum W_i \qquad (7.2)$$

де Wi – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B=1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}026$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн}=T_{ном}\, ПВ_j, \qquad (7.3)$$

де $ПВj$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн}= 6{,}78\cdot(0{,}88\cdot0{,}93\cdot0{,}88\cdot0{,}91\cdot0{,}95\cdot1\cdot1\cdot0{,}87\cdot1{,}22\cdot1{,}16\cdot1{,}1\cdot1{,}1\cdot$$

$$\cdot1{,}12\cdot1{,}1\cdot1\cdot1{,}1\cdot1{,}1)=9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3CT_{уточн}^{0{,}33+0{,}2(B-1{,}01)}S, \qquad (7.4)$$

де C – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПО згідно встановленим вимогам. Вибираємо в межах (25…350)%

$$T_{РП}=0{,}3\cdot3{,}23\cdot9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)}\cdot83=168 \text{ люд/день}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	168	Ф 7.1-7.4
Впровадження	13	Д13
Всього	209	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де Fpq – плановий фонд робочого часу одного спеціаліста, днів, Тпз – трудомісткість розробки програмного забезпечення люд-дні,

$$Ч = \frac{209 \cdot 1}{60 - 5} = 3{,}8 \text{ ставки}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість обладнання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	385	12	4620	77
Монітор	160	12	1920	32
Клавіатура	140	12	1680	28
Маніпулятор «мишка»	30	12	360	6
Принтер матричний	185	1	185	3
Принтер лазерний	355	2	710	12
Принтер струминний	300	1	300	5
Сканер	155	2	310	5
Концентратор-маршрутизатор	155	2	310	5
Кабельні господарства ЛОМ на 1 м. п.	2,5	100	250	4
Кабельне господарство електромережі	48	50	2400	40
Копіювальний апарат	285	2	570	10
Усього за рік:			$З_ч$	227

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{міс}}{1,2} \quad (7.6)$$

$$Ф_{др}^{с} = \frac{227 \cdot 3}{1,2} = 567,5 \text{ год}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф^{с}_{др}}{F_{др} \cdot T_{зм}} \quad (7.7)$$

$Ч_{ел}$=567,5/(60·8)=1,2 ставки

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів–електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	Кількість штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, серверу доступу АДСЛ (ОС Linux), Wi-Fi налаштування ADSL, VPN, PPPoE, Frame Relay	2	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів д мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	Кількість штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,25
	Підтримка постійних клієнтів	0,5	
	Оформлення договорів, ведення тендерів	0,25	
	Контроль взаєморозрахунків з постачальниками	0,25	
Всього		2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,25
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	

Складемо штатний розклад виконавців:

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньо-місячний оклад, грн.	Всього за період роз-робки, грн.
Керівник (ІТ-менеджер)	1	12000	36000
Продакт-менеджер	0,25	8000	6000
Інженер-програміст	3,8	8000	91200
Інженер-електронщик	1,2	6000	21600
Інженер-системотехнік	0,25	6000	4500
Адміністратор мережі	0,5	6000	9000
Системний програміст	0,25	6000	4500
Дизайнер WEB	0,25	8000	6000
Інженер-верстальник	0,25	6000	4500
Бухгалтер-економіст	0,5	10000	15000
Всього за період розробки	$R_{сп}$=8,25	-	$Ф_{роб}$=198300

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де Фроб – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{198300}{8,25 \cdot 60} = 400 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць. Sy – питома площа на одне робоче місце, $м^2$; $Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних ТОВ науково-дослідницького консалтингового підприємства «Пекторaль» ціна одного квадратного метра площі новобудови, вік якої не перевищує 25 років, по місту складає 800…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 25 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн на одне робоче місце. Тобто

$$І_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де Цм – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 8 \cdot 3500 = 28000 \text{ грн}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7. Дані по оптовій ціні на обладнання та комплектуючі вибирались за прайсом фірми hotline за 16.10.21 – джерело https://hotline.ua

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		10947
Системний блок		7347
Процесор	Socket FM2+ AMD Athlon X4 845 3,5-3,8 GHz 4C 2 MB 65 Вт BOX	1350
Системна плата	ASRock FM2+ FM2A68M-DG3+ A68H/DDR3 1600/DVI+D-Sub+PCI-E16x/4 SATA/2xUSB 3.0/5.1/1G	1200
Відеокарта	GIGABYTE GV-N710D3-1GL GeForc GT710 1 GB DDR3 954/1800 MHz 64-bit D Sub+HDMI+DVI	1150
Жорсткий диск	HDD Seagate Barracuda 750 Gb 7200 32Mb SATAII ST3750528AS (ST3750528AS)	1200
Оперативна пам’ять	DIMM 1024Mb DDR3 PC3-10600 CL9 Transcend JetRam 1333Mhz, 128M x 64, non-Reg., no-ECC , CL 9 (2 модулі)	900
DVD-привод	DVD -RW/+RW , LG SATA SuperMulti Bulk 22x, SecurDisc, black	416
Корпус	GRESSO GE-7525, 500W (120mm big fan), 2xIDE, full-ATX,БЖ 2xSATA, 1xFDD, Air Duct, 2xUSB 2.0, Mic+Audio, silver/black	911
Кардрідер внутрішній	USB 2.0 Card reader STORM CR -35U1A4-B, int. 3.5", 1*USB2.0+AUDIO+1394, multi: All Type Cards, black	220
інше	Клавіатура, мишка	-

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Монітор	LG W2363V-WF Wide LCD 2ms, 70 000:1, 300кд/м2, 170/160, D-Sub / Glossy White	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струменевий	Epson Stylus Photo P50 (C11CA45341) + USI cable	5500
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп’ютери	8	10947	8757,6	96333,6
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	114885,1

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400
Група 4			
3. Обчислювальна техніка	114885	-	-
Всього по групі	114885	50	57442,5
Група 5			
4. Вимірювальні пристрої	5190	-	-
5. Господарський інвентар	28000	-	-
Всього по групі	33190	25	8297,5
Нематеріальні активи			
6. Нематеріальні активи	50000	10	5000
Разом	$К_р$= 1606075		$А_р$= 141140

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_e}, \quad (7.11)$$

де Ne – Кількість екземплярів програм, шт.

$$З_о = 400 \cdot 209/50 = 1672 \text{ грн}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де $Н_q$ – норматив додаткової зарплати, %

$$З_д = 1672 \cdot 10 \cdot 0{,}01 = 167{,}2 \text{ грн}$$

Відрахування на соціальні потреби за нормативом Нс=37% від суми основної та додаткової зарплати

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де $Н_с$ – відрахування на соціальні потреби, %

$$С_{оц} = 0{,}01 \cdot 37(1672+167{,}2) = 681 \text{ грн}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом Нг=15% від основної зарплати

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де $Н_г$ – загальногосподарські витрати, %

$$Г_{осп} = 1672 \cdot 15 \cdot 0{,}01 = 251 \text{ грн}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де $З_{М1}$ – вартість паперу, грн., $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн., $З_{М3}$ – вартість фарби, картріджей, тонеру, грн., $N_е$ – кількість екземплярів програм, шт.

Згідно виданих викладачем норм $n_{міс}$ приймаємо 0,33 пачки паперу на місяць розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=105 грн., визначаємо вартість паперу за період розробки $N_м$ =3 міс:

$$З_{М1} = Ц_п \cdot N_м \, n_{міс}. \quad (7.16)$$

$$З_{М1} = 105 \cdot 3 \cdot 0{,}33 = 105 \text{ грн}.$$

Згідно виданих викладачем норм до вартості запам'ятовуючих пристроїв входить вартість CD дисків в кількості, що дорівнює кількості екземплярів програм та одного DVD диска для збереження резервної копії програми:

$$З_{М2} = \sum Ц_{д}., \quad (7.17)$$

де Цд – вартість дисків CD/DVD: CDR TDK 700Mb, 80Min, 52x Cake box – 8 грн/шт., DVD-R LG 4,7Gb, 16x speed Cake box – 12 грн/шт.

$$З_{М2} = 50 \cdot 8 + 12 = 412 \text{ грн.}$$

Згідно виданих викладачем норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_{з}., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{М3} = 574+558+570=1702 \text{ грн.}$$

$$З_{М} = (105 + 412 + 1702)/50=44 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом (Нп = 15%) від основної зарплати виконавців

$$О_{п} = З_{о} \cdot Н_{п} \cdot 0{,}01, \quad (7.19)$$

де Нп – норматив витрат на освоєння нових мов програмування, %

$$О_{п} = 1672 \cdot 15 \cdot 0{,}01=251 \text{ грн}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 50 прим.)

$$А_{м} = \frac{А_{р} \cdot N_{міс}}{N_{е} \cdot 12}, \quad (7.20)$$

де Ар – загальна річна сума амортизаційних відрахувань, грн.

$$А_{м} = 141140 \cdot 3/(50 \cdot 12)=706 \text{ грн}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції

$$C_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \quad (7.21)$$

$$C_п = 1672+167+681+251+44+251+706 = 3772 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності (Рп) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 55%

$$П_р = 0{,}01 \cdot Р_п \cdot С_п, \quad (7.22)$$

де Рс – рівень рентабельності, %

$$П_р = 0{,}01 \cdot 55 \cdot 3772 = 2075 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1. Основна зарплата виконавців	$З_о$	1672
2. Додаткова зарплата виконавців	$З_д$	167
3. Відрахування на соціальні потреби	$С_{оц}$	681
4. Загальногосподарські витрати	$Г_{осп}$	251
5. Витрати на матеріали	$З_М$	44
6. Освоєння нових операційних систем, мов програмування	$О_п$	251
7. Амортизація основних фондів	$А_м$	706
8. Повна собівартість програмного забезпечення	$С_п$	3772
9. Плановий прибуток	$П_р$	2075
10.Ціна підприємства $Ц_п=С_п+П_р$	$Ц_п$	5847
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	1169,4
12.Відпускна ціна програмної продукції Ц = Цп + ПДВ	$Ц$	7922

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 – Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн	
	Базовий	Новий
Вартість програмної продукції	–	7922
Всього капітальних витрат	–	7922

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на обслуговування	$З_р$	27914	5234
2. Витрати на електроенергію	$З_{ел}$	2565	1282
3. Витрати на амортизацію	$З_{ам}$	0	1981
Всього витрат за рік	$І$	30479	8497

Витрати на оплату праці:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_с), \quad (7.23)$$

де Тр – кількість годин обслуговування системи за рік, *год.*; Зг – заробітна плата обслуговуючого персоналу, *грн / год*.

Після купівлі нового програмного забезпечення кількість годин обслуговування системи зменшилась з 400 годин до 75 годин на рік, тому витрати на обслуговування склали:

$$З_{р\,баз} = 400 \cdot 52 \cdot 1{,}1 \cdot 1{,}22 = 27914 \text{ грн.}$$

до

$$З_{р\,нов} = 75 \cdot 52 \cdot 1{,}1 \cdot 1{,}22 = 5234 \text{ грн.}$$

Витрати на електроенергію визначаються з урахуванням спожитої потужності (Пел) в кіловатах, часу експлуатації технічних засобів (Тр) в годинах та ціни однієї кіловат-години (Цел).

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

Зел баз=·0,475·2455·2,2 = 2565 грн

Зел нов=·0,475·1227·2,2 = 1282 грн

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	7922	–	1980,5
Всього відрахувань	-	–	7922	–	1980,5

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (5847 - 3772) \cdot 50 - (0,05 \cdot 1408000 + 0,5 \cdot 114885 +$$
$$+0,25 \cdot 33190 + 0,1 \cdot 50000) \cdot 3/12 = 68465 \; грн.$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р^*$ – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх строк служби на порядок більший ніж період розробки ПЗ.

$$T_в = \frac{198075}{(5847 - 3772) \cdot 50 \cdot 12 / 3} = 0,5 \; років.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н (K_н - K_б), \quad (7.27)$$

де $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно, $K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються

$$E_{сп} = (30479 - 8497) - 0,25 \cdot 7922 = 20002 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н} \quad (7.28)$$

$$T_{сп} = \frac{7922}{30479 - 8497} = 0,4 \ року$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	50
2. Повна собівартість розробленої програми	Грн.	3772
3. Ціна розробленої програми	Грн.	5847
4. Плановий прибуток від реалізації розробленої програми	Грн.	2075
5. Рентабельність програмної продукції	%	55
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1606075
7. Загальний прибуток від реалізації програмної продукції	Грн.	103750
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	68465
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	0,5
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	7922
11.Величина економічного ефекту у користувача програмної продукції	Грн.	20002
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,4

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Протягом усієї історії людство приділяє прискіпливу увагу безпеці життя. Охорона праці є складовою частиною безпеки життя.

Законом України “Про охорону праці” регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби пальців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робітників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

При розгляді шкідливих чинників роботи програмістів та інших спеціалістів ІТ будемо керуватись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98, та

«Правила охорони праці під час експлуатації електронно-обчислювальних машин» НПАОП 0.00-1.28-10,

Умови праці програміста включають наступні фактори:

- параметри повітряного середовища в приміщенні;
- вентиляція приміщення;
- освітлення приміщення;
- параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення впливу комп'ютера на організм програміста визначимо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста,

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Програміст працює з електронно-обчислювальною машиною (ЕОМ) та іншим обладнанням, яке є джерелом небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. Так як програміст постійно перебуває в приміщенні, тому для комфортних умов праці в цьому приміщенні необхідно створити належний мікроклімат.

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори:

- ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.
- ризик виникнення пожежі;
- негативний вплив на органи зору людини;
- ризики ураження електричним струмом;
- недостатня, або надмірна освітленість робочого місця;
- монотонність праці;

- електромагнітні електромагнітні (у т.ч. високочастотні) випромінювання (коливання);

- несприятливі мікрокліматичні умови;

- нервово-емоційна напруженість праці;

- інтелектуальні навантаження;

- невідповідність ергономічних показників робочого місця діючим вимогам;

- шуми;

- статичні навантаження на кістково-м’язовий апарат;

Результати досліджень показали, що найбільшою мірою негативний фізіологічний вплив на операторів ПК пов'язано з дискомфортними зоровими умовами через неправильно спроектованого освітлення: пряма і відбита від екранів блякліcть, несприятливий розподіл яскравості в полі зору, невірна орієнтація робочого місця щодо світлоприймачів.

Розташовувати обладнане дисплеєм робоче місце необхідно таким чином, щоб в поле зору оператора не потрапляли вікна або освітлювальні прилади. Вони не повинні перебувати і безпосередньо за спиною оператора. Слід домагатися зменшення відображень на екрані від різних джерел штучного і денного світла. Коли штучне світло змішується з природним, рекомендується використовувати лампи, по спектрального складу найбільш близькі до сонячного світла. Співвідношення яскравості екрана і безпосередніх найближчого оточення не повинно перевищувати 3: 1.

Оптимальні значення температури повітря в приміщенні повинні бути 19-23 С. Рекомендована відносна вологість повітря 55%. Швидкість руху повітря не більше 0.1 м / с.

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	6,3
Довжина	8,4
Висота	3,1

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	6,6
Обсяг, V	$м^3$	не менше 20.0	20,5

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

У зазначеному приміщенні працє 8 осіб. За даними, які наведено у табл. 8.1 та табл. 8.2, можна зробити висновок, що площа та об’єм приміщення у розрахунку на одне робоче місце програміста відповідають нормативним вимогам (Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»).

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, яка виконується в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря у приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Волога,%	Швидкість повітря, м/с	Температура,°С	Волога %	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-23	41-59	0,1
Тепла	23-25	50-70	0,1	24-25	50-64	0,12

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У

приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні працюють електродвигуни вентиляторів ЕОМ, а також джерела шуму, перелічені у Табл. 7.7.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

Працю працівника, який постійно працює за комп'ютером, згідно державних будівельних норм ДБН В.2.5 – 28 – 2006 р. (які замінили СНиП 11 -4-79), можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V -го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 лк. Крім того все поле зору повинне бути освітлено достатньо рівномірно – ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як н аслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.4 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

- розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;

- мікроклімат відповідає нормативному значенню;

- акустичні умови роботи не перевищують нормативних значень;

Таким чином можна припустити, що основною причиною можливого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору розтіканню електричного струму на землю).

Так як при ураженні електричним струмом у людини може статися фібриляція шлуночків серця, в організації бажано мати дефібрилятор і підготовлений персонал для роботи з ним.

8.5 Розрахункова частина

Початкові данні для розрахунку штучного захисного заземлення:

Тип заземлення: повторне заземлення нульового дроту на вводі в об'єкт. Напруга – 220/380 В. Розташування заземлюючих електродів – у ряд контуру.

Розрахунок проводиться за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару грунта – чорнозем, нижнього шару грунта – глина. Умовна товщина верхнього шару грунта: $H=0,4$ *м*. Для захисного заземлення: застосовуються ветикальні електроди – куток $D=50*50*5$ *мм.*, довжиною $L=3$ *м.* Відстань між вертикальними заземлювачами (електродами) $A=3$ *м*. Тип горизонтального заземлювача: металева полоса з перетином 40*4 мм. ($b=0,004$ *м.)*. Глибина закладення горизонтального контура заземлення $t= 0,7$ *м*. Опір заземлювача, який нормується: $R_{ЗН} = 4$ *Ом*. Необхідно визначити необхідну кількість вертикальних заземлювачів та длину полоси (горизонтального заземлювача).

Розрахунок захисного заземлення можна автоматизувати за допомогою програми, сирцевий код якої опублікован на стр.13-16 [4], або аналогічної.

Розрахунок.

Відстань від центра вертикального заземлювача до поверхні землі:

$$T=t+L/2=0,7+3/2=2,2 \text{ м.}$$

Розрахунковий питомий опір грунта (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі грунта):

$$\rho = \psi \rho_2 = 1,36*40=54,5 \text{ Ом*м.}$$

де $\psi = 1,36$ – табличне значення коефіцієнта сезонності для відповідної климатичной зони у багатошаровому грунті [3]; $\rho_1 = 50$ *Ом*м.* – табличне значення питомого опору верхнього шару грунта (джерело: https://el-line.ru/udelnoe_soprotivlenie_grunta.shtml); $\rho_2 = 40$ *Ом*м.* – табличне значення питомого опору нижнього шару грунта [3].

Еквівалентний діаметр ветикальньного електрода (кутка):

$$Dв=0,95*b= 0,95*50 = 47,5 \text{ мм.} =0,0475 \text{м.}$$

Опір розтіканню електричного струму одного електрода вертикального заземлювача (з врахуванням заглиблення заземлювача):

$$R_O = 0{,}366(\rho/L)[\lg(2L/D_в)+(1/2)\lg((4T+L)/4T-L))] = 0{,}366(54{,}5/3)[\lg(2*3/0{,}0475)+(1/2)\lg((4*2{,}2+3)/4*2{,}2+3))] = 14{,}92 \text{ Ом.}$$

Відношення $A/L=3/3=1$.

Визначаємо коефіцієнт екранування ветикальньних електродів $К_{ев}=0{,}79$ при орієтовній кількості ветикальньних електродів, яке дорівнює 4 [3].

Визначаємо необхідну кількість вертикальних заземлювачів (без вхарування горизонтального заземлювача), при $R_{ЗН} = 4$ Ом :

$$N=R_O /(К_{ев} R_{ЗН})= 14{,}92/ (0{,}79*4)=4{,}66 \approx 5 \text{ шт.}$$

Визначаємо длину з'єднуючої полоси:

$$L_П =1{,}05*A*N= 1{,}05*3*5=15{,}75\approx 16 \text{ м.}$$

Опір розтіканню електричного струму з'єднуючої полоси:

$$R_П =0{,}366(\rho_2*K_П/L_П)\lg(2(L_П *L_П)/(b*t))=$$

$$=0{,}366(40*5/15{,}75)*[\lg(2*15{,}75*15{,}75)/(0{,}04*0{,}7)]=19{,}02 \text{ Ом.}$$

де $K_П=5$ – табличне значення коефіцієнта сезонності для відповідної климатичной зони з'єднуючої полоси: [3].

Загальний опір розтіканню електричного струму заземлювача:

$$R =(R_O*R_П)/(R_O*\eta_П+ N*R_П*К_{ев})=$$

$$=(14{,}92* 19{,}02)/(14{,}92* 0{,}75+ 4{,}66* 19{,}02* 0{,}79)=3{,}25 \text{ Ом.}$$

де $\eta_П = 0{,}75$ – табличне значення коефіцієнта екранування з'єднуючої полоси [3].

Умова $R \le R_{ЗН}$ виконується $(3{,}25 \le 4)$.

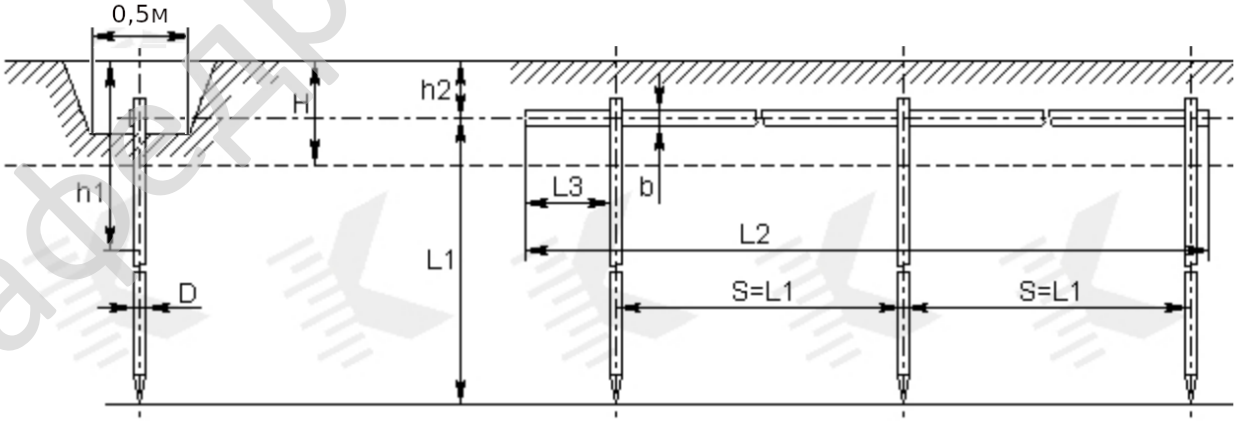

Рисунок 8.1 – Штучне повторне заземлення нульового дроту на вводі в об'єкт

8.6 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

– Досліджена система кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

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

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

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

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

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows XP/Vista/7/8/10.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм RSA.

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.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 20002 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,4 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Загорій А.Д. Дослідження та програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API // Збірник праць молодих науковців ЦНТУ. – Вип. 12. – Кропивницький: ЦНТУ, 2022.

2. А.Ю.Щеглов Защита компьютерной информации от несанкционированного доступа, Наука и техника, Санкт-Петербург, 29\004

3. Дейтел Г., Введение в операционные системы. Т.1, М.: Мир, 2001

4. Хуотаринен А.В., Щеглов А.Ю., Технология физической защиты сетевых устройств, Экономика и производство, № 4, 2002

5. Хуотаринен А.В., Щеглов А.Ю.,, Комплексирование объектной и технической защиты вычислительной сети, BYTE, Россия, № 1, 2000

6. А.А.Тімченко Основа системного проектування та системного аналізу складних об’єктів: Підручник для студентів ВНЗ: У 2-хкн. Кн.1 (А.А.Тимченко; під ред..: В.І.Биков). – К.: Либідь, 2000. – 270 с.

7. Хавьер Пашеку. Програмування Borland Delphi для професіоналів Delphi for. NET DEVELOPER`S Guidle. – М.: Вильямс. 2006. – 944 с.

8. Глонь О.В., Дубовой В.М., Мітюшкін Ю.І. Комп’ютеризовані системи керування: Навчальний посібник. – Вінниця: ВНТУ, 2005. – 157с.

9. Савеленко О.К., Колодочкіна А.В., Лисенко І.А. CASE-ТЕХНОЛОГІЇ У ПРОЕКТУВННІ ІНФОРМАЦІЙНИХ СИСТЕМ. – Кропивницький: ЦНТУ, 2017.- 194 с.

10. Береза А.М. Основи створення інформаційних систем: Навч. посібник. 2 видання,перероблене і доповнене – К.: КНЕУ, 2001. – 270 с.

11. Савеленко О.К., Якименко Н.М., Колодочкіна А.В., Сорокін В.В. Технології проектування комп’ютерних систем: Навчальний посібник. – Кіровоград: КНТУ, 2016. – 237 с.

12. Линда И. Шафер. Роберт Т. Фатрелл, Дональд Ф. Шафер Керування програмними проектами: досягнення оптимальної якості при мінімумі затрат., Видавничий дім «Вільямс». 2003 г. -1136 с.

13. А.В. Колодочкіна, О.К. Савеленко Технології проектування комп'ютерних систем: конспект лекцій для студентів денної та заочної форм навчання напряму підготовки 123 "Комп'ютерна інженерія" – Кіровоград: КНТУ, 2014. – 220 с.

14. Printer Activity Monitor. System Administrator's Guide. [Електроний ресурс]. Режим доступу: http: //www.redline-software.com /eng/support/ docs/ _data/pamHelp.pdf .

15. Наумчук О.М. Основи систем автоматизованого проектування.Інтерактивний комплекс навчально-методичного забезпечення. – Рівне: НУВГП, 2008. – 136 с.

16. Козлов А.П., Кринецький М.І. Основи систем автоматизованого проектування. Конспект лекцій. .К: НАУ, 2003. – 86 с.

17. Саєнко С.Ю., Нечипоренко І.В. Основи САПР. Х.: ХДУХТ, 2017. – 119 с.

18. Кочуров В.А. Инновационные технологии в системах автоматизированного проектирования /В.А. Кочуров, А.В. Бородуля, И.Л. Ковалёва, В.В. Напрасников, С.Е. Пекарчик/. – Учебно-методическое пособие. – Минск: БНТУ, 2017. – 111 с. – ISBN 978-985-550-703-2.

19. Смирнов А.А. Анализ и сравнительное исследование перспективных направлений развития цифровых телекоммуникационных систем и сетей / А.А.Смирнов, В.В.Босько, Е.В.Мелешко // Системи обробки інформації. – Х.: ХУ ПС, 2008. – Вип.7(74). – С.120-123.

20. Смирнов А.А. Усовершенствование метода управления очередями в многопротокольных узлах телекоммуникационной сети / А.А.Смирнов, Е.В.Мелешко // Збірник тез та доповідей другої всеукраїнської науково-

практичної конференції «Системний аналіз. Інформатика. Управління». Запоріжжя. Тези доповідей. Запоріжжя: КПУ, 2011.

21. Смирнов С.А. Метод безопасной маршрутизации метаданных в облачные антивирусные системы / А.К. Дидык, С.А. Смирнов // Информационные технологии в управлении, образовании, науке и промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Видавець Рожко С.Г., 2016. – 566 с.

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

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

24. Смирнов С. А. Исследование показателей качества функционирования интеллектуальных узлов коммутации в телекоммуникационных системах и сетях / Мохамад Абу Таам Гани, А. А. Смирнов, Н. С. Якименко, С. А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України: наук. журн. – Х.: ХУПС, 2014. – № 4(17). – С. 90-95.

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

26. Smirnov S.A. 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.

27. Смирнов С. А. Анализ и исследование методов управления сетевыми ресурсами для обеспечения антивирусной защиты даннях / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Системи озброєння і військова техніка: наук. журн. – Х.: ХУПС, 2015. – № 3(43). – С. 100-107.

28. Смирнов С. А. Исследование эффективности метода управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України: наук. журн. –Х.: ХУПС, 2015. – № 3(20). – С. 134-141.

29. Смирнов С. А. Комплекс gert-моделей технологии облачной антивирусной защиты телекоммуникационной системы / А. А. Смирнов, А. К. Дидык, А. Н. Дреев, С. А. Смирнов // Безпека інформації: наук. – практ. журн. – К.: НАУ, 2015. – Т. 21, № 3. – С. 251-262.

30. Смирнов С. А. Метод безопасной маршрутизации метаданных в облачные антивирусные системы / А. А. Смирнов, А. К. Дидык, С. А. Смирнов // Системи озброєння і військова техніка: наук. журн. – Х.: ХУПС, 2016. – № 2 (46). – С. 146-149.

31. Смирнов С. А. Модели системы нейросетевых экспертов безопасной маршрутизации в облачных антивирусных системах / А. А. Смирнов, А. К. Дидык, А. Н. Дреев, С. А. Смирнов // Системи обробки інформації: зб. наук. праць. – Х.: ХУПС, 2016. – Вип. 3 (140). – С. 36-39.

32. Смирнов С. А. Метод безопасной маршрутизации на базовом множестве путей передачи метаданных в облачные антивирусные системы / В. Л. Бурячок, С. А. Смирнов // Системи управління, навігації та зв'язку. – Полтава, 2016. – Вип. 4(40). – С. 57-62.

33. Смирнов С. А. Способ контроля линий связи телекоммуникационной системы облачного антивируса / А. А. Смирнов, А. К. Дидык, А. Н. Дреев,

С. А. Смирнов // Збірник наукових праць Харківського університету Повітряних Сил. – Харків: ХУПС, 2016. – № 2 (47). – С. 148-152.

34. Смирнов С. А. Дослідження та реалізація GERT-моделі технології розповсюдження комп'ютерних вірусів для захисту телекомунікаційних систем / В. Л. Бурячок, Мохамад Абу Таам Гани, С. А. Смирнов // Інформаційні технології та комп'ютерна інженерія: зб. тез доп. наук.-практ. конф., м. Кіровоград, 4 грудня 2014 р. – Кіровоград: КНТУ, 2014. – С. 168.

35. CAD-CAM & Rapid Prototyping Application Evaluation. Bookboon, 2010. – 174 p. – ISBN: 978-87-7681-676-6

36. Davidson J.K. (Ed.) Models for Computer Aided Tolerancing in Design and Manufacturing. Springer, 2005. – 354 p. – ISBN 978-1402054389.

37. He X. Hua E. Lin Y. Liu X. (Eds.) Computer-Aided Design, Manufacturing, Modeling and Simulation. Verlag C.H.Beck, 2011. – 780 p. – ISBN 978-3319509389.

38. Kurdila A.J., Pardalos P.M., Zabarankin M. (ed.) Robust Optimization-Directed Design . Springer, 2006. – 284 p. – ISBN-10: 0387282637

39. Leon-Rovira N. (Ed.) Trends in Computer Aided Innovation 2007. Springer, 2007. – 229 p. – ISBN 978-0387754550.

40. Luo Y. (Ed.) Cooperative Design Visualization and Engineering, CDVE 2017. Springer, 2017. – 300 p. – ISBN 978-3319668048.

41. Shaler Stidham Jr. O ptimal Design of Queueing Systems. Chapman and Hall/CRC, 2009. 384 p. – ISBN-10: 1584880767

42. Астахова, И.Ф. Компьютерные науки. Деревья, операционные системы, сети / И.Ф. Астахова и др. – М.: Физматлит, 2013. – 88 с.

43. Баринов, В.В. Компьютерные сети: Учебник / В.В. Баринов, И.В. Баринов, А.В. Пролетарский. – М.: Academia, 2018. – 192 с.

44. Баринов, В.В. Компьютерные сети: Учебник / В.В. Баринов. – М.: Академия, 2015. – 256 с.

45. Кузин, А.В. Компьютерные сети: Учебное пособие / А.В. Кузин.. – М.: Форум, НИЦ Инфра-М, 2013. – 192 с.

46. Кузин, А.В. Компьютерные сети: Учебное пособие / А.В. Кузин, Д.А. Кузин. – М.: Форум, 2018. – 704 с.

47. Кузьменко, Н.Г. Компьютерные сети и сетевые технологии / Н.Г. Кузьменко. – СПб.: Наука и техника, 2013. – 368 с.

48. Куроуз, Д. Компьютерные сети. Нисходящий подход / Д. Куроуз, К. Росс. – М.: Эксмо, 2016. – 912 с.

49. Куроуз, Дж. Компьютерные сети: Нисходящий подход / Дж. Куроуз. – М.: Эксмо, 2018. – 800 с.

50. Луганцев, Л.Д. Компьютерные сети / Л.Д. Луганцев. – М.: МГУИЭ, 2001. – 452 с.

51. Максимов, Н.В. Компьютерные сети: Учебное пособие / Н.В. Максимов, И.И. Попов. – М.: Форум, 2017. – 320 с.

52. Максимов, Н.В. Компьютерные сети: Учебное пособие для студентов учреждений среднего профессионального образования / Н.В. Максимов, И.И. Попов. – М.: Форум, НИЦ Инфра-М, 2013. – 464 с.

53. Новожилов, Е.О. Компьютерные сети: Учебное пособие / Е.О. Новожилов. – М.: Academia, 2017. – 288 с.

54. Новожилов, Е.О. Компьютерные сети / Е.О. Новожилов. – М.: Academia, 2016. – 352 с.

55. Новожилов, Е.О. Компьютерные сети: Учебное пособие / Е.О. Новожилов. – М.: Academia, 2016. – 288 с.

56. Новожилов, Е.О. Компьютерные сети: Учебное пособие / Е.О. Новожилов. – М.: Академия, 2018. – 176 с.

57. Новожилов, Е.О. Компьютерные сети. Учебное пособие / Е.О. Новожилов. – М.: Academia, 2016. – 288 с.

58. Олифер, В. Компьютерные сети. Принципы, технологии, протоколы: Учебник / В. Олифер, Н. Олифер. – СПб.: Питер, 2016. – 176 с.

59. Олифер, В. Компьютерные сети. Принципы, технологии, протоколы: Учебник / В. Олифер, Н. Олифер. – СПб.: Питер, 2016. – 318 с.

60. Олифер, В. Компьютерные сети. Принципы, технологии, протоколы: Учебник для ВУЗов / В. Олифер. – СПб.: Питер, 2012. – 944 с.

61. Олифер, В.Г. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. Стандарт третьего поколения / В.Г. Олифер, Н.А. Олифер.. – СПб.: Питер, 2013. – 944 с.

62. Попов, И.И. Компьютерные сети / И.И. Попов, Н.В. Максимов. – М.: Форум, 2004. – 336 с.

63. Прончев, Г.Б. Компьютерные коммуникации. Простейшие вычислительные сети: Учебное пособие / Г.Б. Прончев. – М.: КДУ, 2009. – 64 с.

64. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

65. Про охорону праці: Закон України від 14.10.1992 р. № 2694-XII. Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12#Text

66. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : “Колос”, 1973. – 238 с.

67. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін -т с.-г. машинобуд.; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. – Кіровоград: КІСМ, 1997. – 20 с. Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування..... 2
2 Підстава для розробки..... 2
3 Мета та призначення розробки..... 2
4 Джерела розробки..... 2
5 Технічні вимоги..... 2
5.1 Вміст проекту..... 2
5.2 Показники призначення..... 3
5.3 Вимоги до функціональних характеристик..... 3
5.4 Вимоги до архітектури..... 3
5.5 Вимоги до надійності..... 3
5.6 Умови експлуатації..... 4
5.7 Вимоги до складу та параметрів технічних засобів..... 4
5.8 Вимоги до інформаційної і програмної сумісності..... 4
5.8.1 Обладнання..... 4
5.8.2 Мова програмування..... 4
5.8.3 Вхідні дані..... 5
5.8.4 Вихідні дані..... 5
6 Вимоги до програмної документації..... 5
7 Економічні вимоги..... 5
8 Вимоги щодо охорони праці..... 5
9 Перелік документів, що розробляються..... 6
10 Етапи розробки..... 6
11 Порядок контролю та приймання..... 6

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 42-13 від 02.08.2021 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) 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;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API;

– 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 XP/Vista/7/8/10 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 XP/Vista/7/8/10.

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 Delphi.

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 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2021 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинні бути розглянуті шкідливі і небезпечні фактори при роботі з комп’ютером.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 3 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 121 аркуш.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2021 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 20.12.2021 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

другим (магістерським) рівнем вищої освіти

_____ Смірнов О.А.

Дослідження та програмна реалізація системи кібербезпеки для захисту програмних масивів на основі використання бібліотеки Crypto API

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 36

Літера: РП

Кропивницький – 2021 року

ZaxistInfo.dpr

```
program ZaxistInfo;
//файл проекта програми клієнтської частини
uses
  Forms,
  MainForm in 'MainForm.pas' {Main};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TMain, Main);
  Application.Run;
end.
```

MainForm.pas

```
unit MainForm;
//головний модуль програми 1
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls, Wcrypt2, Menus, ScktComp, ComCtrls;

type
  TMain = class(TForm)
    OpenDialog1: TOpenDialog;
    Panel3: TPanel;
    Label3: TLabel;
    ShifrBlochCheckBox: TCheckBox;
    Panel4: TPanel;
    Label4: TLabel;
    DataNameProjectFile: TEdit;
    BitBtn1: TBitBtn;
    ECP_CheckBox: TCheckBox;
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    Panel1: TPanel;
    ComboBox1: TComboBox;
    Label1: TLabel;
    Label2: TLabel;
    Label5: TLabel;
    ZaxistBtn: TBitBtn;
    Button1: TButton;
    Button2: TButton;
    N4: TMenuItem;
    Label6: TLabel;
    Edit1: TEdit;
    BitBtn2: TBitBtn;
    AlgRadioGroup: TRadioGroup;
    SaveDialog1: TSaveDialog;
    HashRadioGroup: TRadioGroup;
    ClientSocket1: TClientSocket;
    ServerSocket1: TServerSocket;
    Bevel1: TBevel;
    StatusBar1: TStatusBar;
    procedure DataNameEditChange(Sender: TObject);
    procedure OpenBtnClick(Sender: TObject);
    procedure ZaxistBtnClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure ShifrKomentchekBoxClick(Sender: TObject);
    procedure N1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure ServerSocket1ClientRead(Sender: TObject;
      Socket: TCustomWinSocket);
    procedure ServerSocket1ClientConnect(Sender: TObject;
      Socket: TCustomWinSocket);
    procedure ServerSocket1ClientDisconnect(Sender: TObject;
      Socket: TCustomWinSocket);
  private
    { Private declarations }
    plaintext, ciphertext: string;
  public
    { Public declarations }
    lv_Str: array[1..30]of Char;
    lv_parolStr: String;
    lv_prapor,lv_prapor1,lv_prapor2,lv_prapor3 :Boolean;
    function ErrToStr(e: int64): string;
```

```
    function BlockShifr:Boolean;
    function Signing: Boolean;
  end;

var
  Main: TMain;
  StartInfo : TStartupInfo;
  ProcInfo : TProcessInformation;
  lv_f: File;
  lv_parol: Boolean;
  err: string;
  hProv: HCRYPTPROV;
  hash: HCRYPTHASH;
  key: HCRYPTKEY;
  MS: TMemoryStream;


implementation

uses VvodParolForm;

{$R *.dfm}

procedure TMain.DataNameEditChange(Sender: TObject);
{Якщо не вибраний файл, то кнопка Захистити неактивна}
begin
  {Перевірка на наявність вибраного файлу}
if not FileExists(DataNameProjectFile.Text) then
   begin
   DataNameProjectFile.Text:='';
    ShowMessage('Введення імені проекту є обов'язковим');
   end;


 //визначення активності кнопки Захистити
  ZaxistBtn.Enabled :=
  (length(DataNameProjectFile.Text) > 0);


end;

procedure TMain.OpenBtnClick(Sender: TObject);
{Процедура призначена для вибору імені файлу}
begin
if OpenDialog1.Execute then
   DataNameProjectFile.Text := OpenDialog1.FileName;
end;

procedure TMain.ZaxistBtnClick(Sender: TObject);
  var f, outkeyFile: textfile;
      I: Integer;
      lv_String, lv_StrXor: String;
      l: DWORD;
      data: PByte;
      inFile, outFile : file;
begin
  ZaxistBtn.Enabled:= False;
  //перевіряємо вибір властивостей захисту
  if ShifrBlochCheckBox.Checked=False then
   //встановлення прапорців
    begin
      lv_prapor2:=False;
    end
  else
    lv_prapor2:=True;
  if ECP_CheckBox.Checked=False then
    //встановлення прапорців
    begin
      lv_prapor3:=False;
```

```
    end
  else
    lv_prapor3:=True;

 if lv_prapor3 and lv_prapor2 then
    begin
    //шифрування блочним(поточним) шифром
      BlockShifr;
    //підпис файла ЕЦП
      Signing;
    end
  else
    if not lv_prapor3 and lv_prapor2 then
      begin
      //шифрування блочним(поточним) шифром
        BlockShifr;
      end
    else
      if lv_prapor3 and not lv_prapor2 then
        begin
        //ЕЦП
          Signing;
        end
      else
        ShowMessage('Виберіть властивості захисту');

  ZaxistBtn.Enabled:= True;
end;

procedure TMain.FormActivate(Sender: TObject);
begin
  //визначення активності кнопки Захистити
  ZaxistBtn.Enabled :=
  (length(DataNameProjectFile.Text) > 0);
  //визначення властивостей захисту

  if ShifrBlochCheckBox.Checked=False then
   //встановлення прапорців
    begin
      lv_prapor2:=False;
    end
  else
    lv_prapor2:=True;
  if ECP_CheckBox.Checked=False then
    //встановлення прапорців
    begin
      lv_prapor3:=False;
    end
  else
    lv_prapor3:=True;
end;

procedure TMain.ShifrKomentchekBoxClick(Sender: TObject);
begin

  if ShifrBlochCheckBox.Checked=False then
   //встановлення прапорців
    begin
      lv_prapor2:=False;
    end
  else
    begin
      lv_prapor2:=True;
    end;
  if ECP_CheckBox.Checked=False then
    //встановлення прапорців
    begin
      lv_prapor3:=False;
    end
```

```
  else
    lv_prapor3:=True;
 //визначення активності кнопки Захистити
  ZaxistBtn.Enabled :=
  (length(DataNameProjectFile.Text) > 0);
end;

function TMain.ErrToStr(e: int64): string;
{функція переводу помилок в форматі int64 у формат string}
begin
  case e of
    ERROR_BUSY: ErrToStr := 'ERROR_BUSY';
    ERROR_CALL_NOT_IMPLEMENTED: ErrToStr := 'ERROR_CALL_NOT_IMPLEMENTED';
    ERROR_INVALID_HANDLE: ErrToStr := 'ERROR_INVALID_HANDLE';
    ERROR_INVALID_PARAMETER: ErrToStr := 'ERROR_INVALID_PARAMETER';
    ERROR_MORE_DATA: ErrToStr := 'ERROR_MORE_DATA';
    ERROR_NO_MORE_ITEMS: ErrToStr := 'ERROR_NO_MORE_ITEMS';
    ERROR_NOT_ENOUGH_MEMORY: ErrToStr := 'ERROR_NOT_ENOUGH_MEMORY';
    NTE_BAD_ALGID: ErrToStr := 'NTE_BAD_ALGID';
    NTE_BAD_DATA: ErrToStr := 'NTE_BAD_DATA';
    NTE_BAD_FLAGS: ErrToStr := 'NTE_BAD_FLAGS';
    NTE_BAD_HASH: ErrToStr := 'NTE_BAD_HASH';
    NTE_BAD_HASH_STATE: ErrToStr := 'NTE_BAD_HASH_STATE';
    NTE_BAD_KEY: ErrToStr := 'NTE_BAD_KEY';
    NTE_BAD_KEYSET: ErrToStr := 'NTE_BAD_KEYSET';
    NTE_BAD_KEYSET_PARAM: ErrToStr := 'NTE_BAD_KEYSET_PARAM';
    NTE_BAD_LEN: ErrToStr := 'NTE_BAD_LEN';
    NTE_BAD_PROV_TYPE: ErrToStr := 'NTE_BAD_PROV_TYPE';
    NTE_BAD_PUBLIC_KEY: ErrToStr := 'NTE_BAD_PUBLIC_KEY';
    NTE_BAD_SIGNATURE: ErrToStr := 'NTE_BAD_SIGNATURE';
    NTE_BAD_TYPE: ErrToStr := 'NTE_BAD_TYPE';
    NTE_BAD_UID: ErrToStr := 'NTE_BAD_UID';
    NTE_DOUBLE_ENCRYPT: ErrToStr := 'NTE_DOUBLE_ENCRYPT';
    NTE_EXISTS: ErrToStr := 'NTE_EXISTS';
    NTE_FAIL: ErrToStr := 'NTE_FAIL';
    NTE_KEYSET_ENTRY_BAD: ErrToStr := 'NTE_KEYSET_ENTRY_BAD';
    NTE_KEYSET_NOT_DEF: ErrToStr := 'NTE_KEYSET_NOT_DEF';
    NTE_NO_KEY: ErrToStr := 'NTE_NO_KEY';
    NTE_NO_MEMORY: ErrToStr := 'NTE_NO_MEMORY';
    NTE_PROV_DLL_NOT_FOUND: ErrToStr := 'NTE_PROV_DLL_NOT_FOUND';
    NTE_PROV_TYPE_ENTRY_BAD: ErrToStr := 'NTE_PROV_TYPE_ENTRY_BAD';
    NTE_PROV_TYPE_NO_MATCH: ErrToStr := 'NTE_PROV_TYPE_NO_MATCH';
    NTE_PROV_TYPE_NOT_DEF: ErrToStr := 'NTE_PROV_TYPE_NOT_DEF';
    NTE_PROVIDER_DLL_FAIL: ErrToStr := 'NTE_PROVIDER_DLL_FAIL';
    NTE_SIGNATURE_FILE_BAD: ErrToStr := 'NTE_SIGNATURE_FILE_BAD';
  else ErrToStr := 'unknown error';
  end;
end;

procedure TMain.N1Click(Sender: TObject);
//завантаження модуля створення контейнерів та ключів
begin
//заповнення структури функції CreateProcess
  FillChar(StartInfo, Sizeof(StartInfo), #0);
  StartInfo.cb := Sizeof(StartInfo);
  StartInfo.dwFlags := STARTF_USESHOWWINDOW;
  StartInfo.wShowWindow := SW_SHOWNORMAL;
  if not CreateProcess(nil,'KeysAndSigns.exe', nil,
                       nil, false, CREATE_NEW_CONSOLE or HIGH_PRIORITY_CLASS,
                       nil, nil, StartInfo, ProcInfo)
  then ShowMessage(IntToStr(GetLastError));
end;

function TMain.BlockShifr: Boolean;
{Функція шифрування }
var cont: PChar;
    err: string;
    hProv: HCRYPTPROV;
    KeyExchKey, SessionKey: HCRYPTKEY;
```

```
    flag, keyLen: DWORD;
    infile, outfile: file;
    tmp: PBYTE;
    buf: array [0..511] of byte;
    alg: ALG_ID;
    stream: boolean;
begin
 //задаємо контейнер
  err := '1';
  cont := StrAlloc(length(err) + 1);
  StrPCopy(cont, err);

{Підключаємося до криптопровайдера (контейнера)}
  if not CryptAcquireContext(@hProv, cont, nil, PROV_RSA_FULL, 0)
  then
    begin
      MessageDlg('Помилка при роботі з контейнером: ' + ErrToStr(GetLastError),
                         mtError, [mbOK], 0);
      exit;
    end;

begin
{Вибираємо файл для шифрування}
    AssignFile(infile, DataNameProjectFile.Text);
    OpenDialog1.Title := 'Вкажіть файл з відкритим ключем обміну ключами
одержувача';
    if OpenDialog1.Execute then
       begin
         AssignFile(outfile, OpenDialog1.FileName);
         reset(outfile, 1);
         keyLen := FileSize(outfile);
         GetMem(tmp, keyLen);
         BlockRead(outfile, tmp^, keyLen);
         CloseFile(outfile);
       end
    else exit;
{Імпортуємо відкритий ключ підпису відправника}
    if not CryptImportKey(hProv, tmp, keyLen, 0, 0, @KeyExchKey) then
       begin
       MessageDlg('Помилка імпорту ключа: ' + ErrToStr(GetLastError),
                          mtError, [mbOK], 0);
       exit;
       end;
    FreeMem(tmp, keyLen);
{Вибирає ім'я файлу для зашифрованих даних}
    SaveDialog1.Title := 'Задайте ім'я файлу для зашифрованих даних';
    if SaveDialog1.Execute then AssignFile(outfile, SaveDialog1.FileName)
    else exit;
    rewrite(outfile, 1);
{установка алгоритма шифрування}
    case Main.AlgRadioGroup.ItemIndex of
    0: begin
       {алгоритм RC2}
       alg := CALG_RC2;
       {блочний шифр}
       stream := false;
       end;
    1: begin
       {алгоритм RC4}
       alg := CALG_RC4;
       {поточний шифр}
       stream := true;
       end;
    else exit;
    end;
{Створюємо сеансовий ключ}
    if not CryptGenKey(hProv, alg, CRYPT_EXPORTABLE or CRYPT_CREATE_SALT,
                               @SessionKey) then
       begin
```

```
       MessageDlg('Помилка генерації ключа: ' + ErrToStr(GetLastError),
                          mtError, [mbOK], 0);
       exit;
       end;
{встановлюємо розмір буфера}
    keyLen := 128;
    GetMem(tmp, keyLen);
{Ключ у зашифрованому вигляді зберігається в буфері}
    if not CryptExportKey(SessionKey, KeyExchKey, SIMPLEBLOB, 0, tmp, @keyLen)
then
       begin
       MessageDlg('Помилка експорту ключа: ' + ErrToStr(GetLastError),
                          mtError, [mbOK], 0);
       exit;
       end;
{Збереження зашифрованих ключевих даних}
    BlockWrite(outfile, keyLen, 4);     //запис в файл розміра ключа
    BlockWrite(outfile, tmp^, keyLen); //запис самого зашифрованого ключа
{Знищуємо дескриптор ключа}
    if not CryptDestroyKey(KeyExchKey) then
       begin
       MessageDlg('Помилка знищення ключа обміну ключами: ' +
ErrToStr(GetLastError),
                          mtError, [mbOK], 0);
       exit;
       end;
{встановлюємо розмір буфера}
    keyLen := 512;
{Отримуємо солт - значення}
    if not CryptGetKeyParam(SessionKey, KP_SALT, @buf, @keyLen, 0) then
       begin
       MessageDlg('Помилка отримання salt-значення: ' + ErrToStr(GetLastError),
                          mtError, [mbOK], 0);
       exit;
       end;
    BlockWrite(outfile, keyLen, 4);   //записуємо в файл розмір солт-значення
    BlockWrite(outfile, buf, keyLen); //записуємо саме солт-значення
{Створюємо ініціалізуючий вектор
Якщо шифр блочний}
    if not stream then
       begin
       //ГЕНЕРУВАННЯ IV
       {встановлюємо розмір буфера}
       keyLen := 512;
       {уточнюємо розмір солт-значення}
       if not CryptGetKeyParam(SessionKey, KP_IV, @buf, @keyLen, 0) then
          begin
          MessageDlg('Error of IV getting: ' + ErrToStr(GetLastError),
                          mtError, [mbOK], 0);
          exit;
          end;
       {генруємо ініціалізуючий вектор}
       if not CryptGenRandom(hProv, keyLen, @buf) then
          begin
          MessageDlg('Помилка створення IV: ' + ErrToStr(GetLastError),
                          mtError, [mbOK], 0);
          exit;
          end;
       if not CryptSetKeyParam(SessionKey, KP_IV, @buf, 0) then
          begin
          MessageDlg('Помилка встановлення IV: ' + ErrToStr(GetLastError),
                          mtError, [mbOK], 0);
          exit;
          end;
{запис в файл розміра ініціалізуючого вектора}
       BlockWrite(outfile, keyLen, 4);
{запис в файл самого вектора}
       BlockWrite(outfile, buf, keyLen);
       end;
```

```
    reset(infile, 1);
    while not eof(infile) do
          begin
{шифрування і запис до файла}
          BlockRead(infile, buf, 496, keyLen);
          if not CryptEncrypt(SessionKey, 0, eof(infile), 0, @buf, @keyLen, 512)
then
             begin
             MessageDlg('Помилка при шифруванні: ' + ErrToStr(GetLastError),
                         mtError, [mbOK], 0);
             exit;
             end;
          BlockWrite(outfile, buf, keyLen);
          end;
    CloseFile(infile);
    CloseFile(outfile);
    if not CryptDestroyKey(SessionKey) then
       MessageDlg('Помилка знищення сеансового ключа: ' +
ErrToStr(GetLastError),
                         mtError, [mbOK], 0);
    end;
end;

function TMain.Signing: Boolean;
{Функція призначена для підписання файлу}
var cont: PChar;
    err: string;
    hProv: HCRYPTPROV;
    key: HCRYPTKEY;
    alg: ALG_ID;
    hash: HCRYPTHASH;
    infile, outfile: file;
    size: DWORD;
    buf: array [0..511] of byte;
    signature: PBYTE;
begin
  {Перевірка на наявність вибраного файлу}
if not FileExists(DataNameProjectFile.Text) then
   begin
   MessageDlg('Неправильне ім'я файлу!', mtError, [mbOK], 0);
   exit;
   end;
AssignFile(infile, DataNameProjectFile.Text);
//задаємо контейнер
  err := '1';
  cont := StrAlloc(length(err) + 1);
  StrPCopy(cont, err);

{Підключаємось до криптопровайдера і отримуємо
 його контекст - підключаємося до вибраного контейнера}
if not CryptAcquireContext(@hProv, cont, nil, PROV_RSA_FULL, 0)
then
    begin
{обробка помилок}
    case int64(GetLastError) of
    ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
    ERROR_NOT_ENOUGH_MEMORY: err := 'ERROR_NOT_ENOUGH_MEMORY';
    NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
    NTE_BAD_KEYSET: err := 'NTE_BAD_KEYSET';
    NTE_BAD_KEYSET_PARAM: err := 'NTE_BAD_KEYSET_PARAM';
    NTE_BAD_PROV_TYPE: err := 'NTE_BAD_PROV_TYPE';
    NTE_BAD_SIGNATURE: err := 'NTE_BAD_SIGNATURE';
    NTE_EXISTS: err := 'NTE_EXISTS';
    NTE_KEYSET_ENTRY_BAD: err := 'NTE_KEYSET_ENTRY_BAD';
    NTE_KEYSET_NOT_DEF: err := 'NTE_KEYSET_NOT_DEF';
    NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
    NTE_PROV_DLL_NOT_FOUND: err := 'NTE_PROV_DLL_NOT_FOUND';
    NTE_PROV_TYPE_ENTRY_BAD: err := 'NTE_PROV_TYPE_ENTRY_BAD';
    NTE_PROV_TYPE_NO_MATCH: err := 'NTE_PROV_TYPE_NO_MATCH';
```

```
    NTE_PROV_TYPE_NOT_DEF: err := 'NTE_PROV_TYPE_NOT_DEF';
    NTE_PROVIDER_DLL_FAIL: err := 'NTE_PROVIDER_DLL_FAIL';
    NTE_SIGNATURE_FILE_BAD: err := 'NTE_SIGNATURE_FILE_BAD';
    else err := 'Unknown error';
    end;
    MessageDlg('Помилка відкриття контейнеру: ' + err, mtError, [mbOK], 0);
    exit;
    end;
{визначаємо алгоритм хешірування}
case main.HashRadioGroup.ItemIndex of
0: alg := CALG_MD5;
1: alg := CALG_SHA;
end;
{створюємо хеш-об'єкт}
if not CryptCreateHash(hProv, alg, 0, 0, @hash) then
   begin
{обробка помилок}
   case int64(GetLastError) of
   ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
   ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
   ERROR_NOT_ENOUGH_MEMORY: err := 'ERROR_NOT_ENOUGH_MEMORY';
   NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
   NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
   NTE_BAD_KEY: err := 'NTE_BAD_KEY';
   NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
   else err := 'Unknown error';
   end;
   MessageDlg('Помилка створення хеш-об'єкту: ' + err, mtError, [mbOK], 0);
   exit;
   end;
{Визначаємо ім'я файлу для зберігання підписаних даних}
SaveDialog1.Title := 'Задайте ім'я файлу для зберігання підписаних даних';
if SaveDialog1.Execute then
   begin
   AssignFile(outfile, SaveDialog1.FileName);
   rewrite(outfile, 1);
{Записуємо в файл ІДЕНТИФІКАТОР АЛГОРИТМА ХЕШУВАННЯ}
   BlockWrite(outfile, alg, 4);
   reset(infile, 1);
   size := FileSize(infile);
{Записуєм РОЗМІР ПІДПИСАНИХ ДАНИХ}
   BlockWrite(outfile, size, 4);
{Записуємо ДАНІ і РОЗРАХОВУЄМО ХЕШ}
   while not eof(infile) do
         begin
         BlockRead(infile, buf, 512, size);
         BlockWrite(outFile, buf, size);
{Хешуємо пароль}
         if not CryptHashData(hash, @buf, size, 0) then
            begin
{обробка помилок}
            case int64(GetLastError) of
            ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
            ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
            NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
            NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
            NTE_BAD_HASH: err := 'NTE_BAD_HASH';
            NTE_BAD_HASH_STATE: err := 'NTE_BAD_HASH_STATE';
            NTE_BAD_KEY: err := 'NTE_BAD_KEY';
            NTE_BAD_LEN: err := 'NTE_BAD_LEN';
            NTE_BAD_UID: err := 'NTE_BAD_UID';
            NTE_FAIL: err := 'NTE_FAIL';
            NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
            else err := 'Unknown error';
            end;
            MessageDlg('Помилка при хешуванні: ' + err, mtError, [mbOK], 0);
            break;
            end;
         end;
```

```
   CloseFile(infile);
{Визначаємо розмір підпису}
   if not CryptSignHash(hash, AT_SIGNATURE, nil, 0, nil, @size) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      ERROR_MORE_DATA: err := 'ERROR_MORE_DATA';
      NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
      NTE_BAD_HASH: err := 'NTE_BAD_HASH';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_NO_KEY: err := 'NTE_NO_KEY';
      NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка при визначенні розміру підпису: ' + err, mtError,
[mbOK], 0);
      CloseFile(outfile);
      exit;
      end;
{Створюємо підпис}
   GetMem(signature, size);
   if not CryptSignHash(hash, AT_SIGNATURE, nil, 0, signature, @size) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      ERROR_MORE_DATA: err := 'ERROR_MORE_DATA';
      NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
      NTE_BAD_HASH: err := 'NTE_BAD_HASH';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_NO_KEY: err := 'NTE_NO_KEY';
      NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка при підписані хешу: ' + err, mtError, [mbOK], 0);
      CloseFile(outfile);
      exit;
      end;
{Записуємо ПІДПИС}
   BlockWrite(outfile, size, 4);
   BlockWrite(outfile, signature^, size);
   CloseFile(outfile);
   end;
{Знищуємо хеш-об'єкт}
if not CryptDestroyHash(hash) then
   begin
{обробка помилок}
   case int64(GetLastError) of
   ERROR_BUSY: err := 'ERROR_BUSY';
   ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
   ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
   NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
   NTE_BAD_HASH: err := 'NTE_BAD_HASH';
   NTE_BAD_UID: err := 'NTE_BAD_UID';
   else err := 'Unknown error';
   end;
   MessageDlg('Помилка при знищенні хешу: ' + err, mtError, [mbOK], 0);
   end;
{Звільняємо контекст криптопровайдера}
if not CryptReleaseContext(hProv, 0) then
   begin
{обробка помилок}
   case int64(GetLastError) of
   ERROR_BUSY: err := 'ERROR_BUSY';
```

```
   ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
   ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
   NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
   NTE_BAD_UID: err := 'NTE_BAD_UID';
   else err := 'Unknown error';
   end;
   MessageDlg('Помилка при звільненні контексту: ' + err,
       mtError, [mbOK], 0);
   end;
end;

procedure TMain.BitBtn2Click(Sender: TObject);
{Процедура призначена для вибору імені файлу}
begin
if OpenDialog1.Execute then
   Edit1.Text := OpenDialog1.FileName;
end;

procedure TMain.Button1Click(Sender: TObject);
//відправити на сервер
var
  Size: Integer;
  P: ^Byte;
begin
//створюємо буфер для файлу
  MS:=TMemoryStream.Create;
//завантажуємо файл в буфер
  MS.LoadFromFile(Edit1.Text);

//відправляємо інформацію про файл на приймаючу сторону

ServerSocket1.Socket.Connections[0].SendText('file#'+LowerCase(ExtractFileName(E
dit1.Text))+'#'+IntToStr(MS.Size)+'#');
//переходимо на початок файлу
  MS.Position:=0;
//завантажуємо у змінну файл
  P:=MS.Memory;
//відправляємо файл
  Size:=ServerSocket1.Socket.Connections[0].SendBuf(P^,ms.Size);
//виводимо статистику в рядок статусу
  StatusBar1.SimpleText:='Відправлено'+IntToStr(Size)+'з'+IntToStr(MS.Size)+'
байт'
end;

procedure TMain.FormCreate(Sender: TObject);
begin
  //відкриваємо сокет
  ServerSocket1.Open;
end;

procedure TMain.FormDestroy(Sender: TObject);
begin
  //закриваємо сокет
  ServerSocket1.Close;
end;

procedure TMain.ServerSocket1ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  //якщо клієнт прийняв файл, то
  if Socket.ReceiveText='end' then
    begin
      StatusBar1.SimpleText:='Файл відправлений успішно';
      //очистка буфера
      MS.Free;
    end;
end;

procedure TMain.ServerSocket1ClientConnect(Sender: TObject;
```

```
  Socket: TCustomWinSocket);
begin
  StatusBar1.SimpleText:='З'єднання встановлено';
end;

procedure TMain.ServerSocket1ClientDisconnect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  StatusBar1.SimpleText:='З'єднання відсутнє';
end;
end.
```

Container.pas

```
unit Container;
//створення контейнера. Модуль налаштувань.
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, wcrypt2;

type
  TContainersForm = class(TForm)
    Label1: TLabel;
    ContainerEdit: TEdit;
    CreateButton: TButton;
    DeleteButton: TButton;
    VerifyButton: TButton;
    procedure CreateButtonClick(Sender: TObject);
    procedure DeleteButtonClick(Sender: TObject);
    procedure VerifyButtonClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  ContainersForm: TContainersForm;

implementation

{$R *.DFM}

procedure TContainersForm.CreateButtonClick(Sender: TObject);
{Процедура призначена для створення контейнера ключів}
var cont: PChar;
    err: string;
    hProv: HCRYPTPROV;
begin
{Читаємо ім'я контейнеру з поля введення,
якщо поле залишити пустим то використовується
ім'я користувача в системі}
if length(ContainerEdit.Text) = 0
then
    begin
    cont := nil;
    err := 'по замовченню';
    end
else
    begin
    err := ContainerEdit.Text;
    cont := StrAlloc(length(err) + 1);
    StrPCopy(cont, err);
    end;
{Підключаємося до контейнеру і отримуємо контекст криптопровайдера}
if not CryptAcquireContext(@hProv, cont, nil, PROV_RSA_FULL, CRYPT_NEWKEYSET)
then
    begin
{обробка помилок}
    case int64(GetLastError) of
    ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
    ERROR_NOT_ENOUGH_MEMORY: err := 'ERROR_NOT_ENOUGH_MEMORY';
    NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
    NTE_BAD_KEYSET: err := 'NTE_BAD_KEYSET';
    NTE_BAD_KEYSET_PARAM: err := 'NTE_BAD_KEYSET_PARAM';
    NTE_BAD_PROV_TYPE: err := 'NTE_BAD_PROV_TYPE';
    NTE_BAD_SIGNATURE: err := 'NTE_BAD_SIGNATURE';
    NTE_EXISTS: err := 'NTE_EXISTS';
    NTE_KEYSET_ENTRY_BAD: err := 'NTE_KEYSET_ENTRY_BAD';
```

```
    NTE_KEYSET_NOT_DEF: err := 'NTE_KEYSET_NOT_DEF';
    NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
    NTE_PROV_DLL_NOT_FOUND: err := 'NTE_PROV_DLL_NOT_FOUND';
    NTE_PROV_TYPE_ENTRY_BAD: err := 'NTE_PROV_TYPE_ENTRY_BAD';
    NTE_PROV_TYPE_NO_MATCH: err := 'NTE_PROV_TYPE_NO_MATCH';
    NTE_PROV_TYPE_NOT_DEF: err := 'NTE_PROV_TYPE_NOT_DEF';
    NTE_PROVIDER_DLL_FAIL: err := 'NTE_PROVIDER_DLL_FAIL';
    NTE_SIGNATURE_FILE_BAD: err := 'NTE_SIGNATURE_FILE_BAD';
    else err := 'Unknown error';
    end;
    MessageDlg('Помилка створення контейнера: ' + err, mtError, [mbOK], 0);
    exit;
    end
else MessageDlg('Створений контейнер на ім''я ' + err, mtInformation, [mbOK],
0);
{Звільнюємо контекст криптопровайдера}
if not CryptReleaseContext(hProv, 0) then
   begin
{обробка помилок}
   case int64(GetLastError) of
   ERROR_BUSY: err := 'ERROR_BUSY';
   ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
   ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
   NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
   NTE_BAD_UID: err := 'NTE_BAD_UID';
   else err := 'Unknown error';
   end;
   MessageDlg('Помилка при звільненні контексту: ' + err,
       mtError, [mbOK], 0);
   end;
end;

procedure TContainersForm.DeleteButtonClick(Sender: TObject);
{Процедура призначена для знищення контейнера ключів}
var cont: PChar;
    err: string;
    hProv: HCRYPTPROV;
begin
{Отримуємо(читаємо) ім'я контейнеру ключів з поля введення}
if length(ContainerEdit.Text) = 0
then
    begin
    cont := nil;
    err := 'по замовченню';
    end
else
    begin
    err := ContainerEdit.Text;
    cont := StrAlloc(length(err) + 1);
    StrPCopy(cont, err);
    end;
{Знищуємо контейнер ключів разом з ключами}
if not CryptAcquireContext(@hProv, cont, nil, PROV_RSA_FULL, CRYPT_DELETEKEYSET)
then
    begin
{обробка помилок}
    case int64(GetLastError) of
    ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
    ERROR_NOT_ENOUGH_MEMORY: err := 'ERROR_NOT_ENOUGH_MEMORY';
    NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
    NTE_BAD_KEYSET: err := 'NTE_BAD_KEYSET';
    NTE_BAD_KEYSET_PARAM: err := 'NTE_BAD_KEYSET_PARAM';
    NTE_BAD_PROV_TYPE: err := 'NTE_BAD_PROV_TYPE';
    NTE_BAD_SIGNATURE: err := 'NTE_BAD_SIGNATURE';
    NTE_EXISTS: err := 'NTE_EXISTS';
    NTE_KEYSET_ENTRY_BAD: err := 'NTE_KEYSET_ENTRY_BAD';
    NTE_KEYSET_NOT_DEF: err := 'NTE_KEYSET_NOT_DEF';
    NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
    NTE_PROV_DLL_NOT_FOUND: err := 'NTE_PROV_DLL_NOT_FOUND';
```

```
    NTE_PROV_TYPE_ENTRY_BAD: err := 'NTE_PROV_TYPE_ENTRY_BAD';
    NTE_PROV_TYPE_NO_MATCH: err := 'NTE_PROV_TYPE_NO_MATCH';
    NTE_PROV_TYPE_NOT_DEF: err := 'NTE_PROV_TYPE_NOT_DEF';
    NTE_PROVIDER_DLL_FAIL: err := 'NTE_PROVIDER_DLL_FAIL';
    NTE_SIGNATURE_FILE_BAD: err := 'NTE_SIGNATURE_FILE_BAD';
    else err := 'Unknown error';
    end;
    MessageDlg('Помилка видалення контейнеру: ' + err, mtError, [mbOK], 0);
    end
else MessageDlg('Видалений контейнер на ім'я ' + err, mtInformation, [mbOK], 0);
end;

procedure TContainersForm.VerifyButtonClick(Sender: TObject);
{Процедура призначена для верифікації контейнера}
var cont: PChar;
    err: string;
    hProv: HCRYPTPROV;
begin
{Отримуємо (зчитуємо) ім'я контейнера з поля введення}
if length(ContainerEdit.Text) = 0
then
    begin
    cont := nil;
    err := 'по замовченню';
    end
else
    begin
    err := ContainerEdit.Text;
    cont := StrAlloc(length(err) + 1);
    StrPCopy(cont, err);
    end;
{Підключення до контейнеру ключів та
 отримання контексту криптопровайдера}
if not CryptAcquireContext(@hProv, cont, nil, PROV_RSA_FULL, 0)
then
    begin
{обробка помилок}
    case int64(GetLastError) of
    ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
    ERROR_NOT_ENOUGH_MEMORY: err := 'ERROR_NOT_ENOUGH_MEMORY';
    NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
    NTE_BAD_KEYSET: err := 'NTE_BAD_KEYSET';
    NTE_BAD_KEYSET_PARAM: err := 'NTE_BAD_KEYSET_PARAM';
    NTE_BAD_PROV_TYPE: err := 'NTE_BAD_PROV_TYPE';
    NTE_BAD_SIGNATURE: err := 'NTE_BAD_SIGNATURE';
    NTE_EXISTS: err := 'NTE_EXISTS';
    NTE_KEYSET_ENTRY_BAD: err := 'NTE_KEYSET_ENTRY_BAD';
    NTE_KEYSET_NOT_DEF: err := 'NTE_KEYSET_NOT_DEF';
    NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
    NTE_PROV_DLL_NOT_FOUND: err := 'NTE_PROV_DLL_NOT_FOUND';
    NTE_PROV_TYPE_ENTRY_BAD: err := 'NTE_PROV_TYPE_ENTRY_BAD';
    NTE_PROV_TYPE_NO_MATCH: err := 'NTE_PROV_TYPE_NO_MATCH';
    NTE_PROV_TYPE_NOT_DEF: err := 'NTE_PROV_TYPE_NOT_DEF';
    NTE_PROVIDER_DLL_FAIL: err := 'NTE_PROVIDER_DLL_FAIL';
    NTE_SIGNATURE_FILE_BAD: err := 'NTE_SIGNATURE_FILE_BAD';
    else err := 'Unknown error';
    end;
    MessageDlg('Помилка відкриття контейнера: ' + err, mtError, [mbOK], 0);
    exit;
    end
else MessageDlg('Контейнер на ім'я ' + err + ' успішно відкритий',
mtInformation, [mbOK], 0);
{Звільнення контексту криптопровайдера}
if not CryptReleaseContext(hProv, 0) then
   begin
{обробк помилок}
   case int64(GetLastError) of
   ERROR_BUSY: err := 'ERROR_BUSY';
   ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
```

```
    ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
    NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
    NTE_BAD_UID: err := 'NTE_BAD_UID';
    else err := 'Unknown error';
    end;
    MessageDlg('Помилка при звільненні контексту: ' + err,
        mtError, [mbOK], 0);
    end;
end;

end.
```

Signing.pas

```
unit Signing;

interface
//підпис ЕЦП
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, ExtCtrls, wcrypt2;

type
  TSigningForm = class(TForm)
    Label1: TLabel;
    ContainerEdit: TEdit;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    Label2: TLabel;
    DataNameEdit: TEdit;
    HashRadioGroup: TRadioGroup;
    SignBtn: TBitBtn;
    CloseBtn: TBitBtn;
    OpenBtn: TBitBtn;
    procedure CloseBtnClick(Sender: TObject);
    procedure OpenBtnClick(Sender: TObject);
    procedure SignBtnClick(Sender: TObject);
    procedure DataNameEditChange(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  SigningForm: TSigningForm;

implementation

{$R *.DFM}

procedure TSigningForm.CloseBtnClick(Sender: TObject);
{Процедура призначена для закриття форми Підписання файлу}
begin
Close;
end;

procedure TSigningForm.OpenBtnClick(Sender: TObject);
{Процедура призначена для вибору імені файлу}
begin
if OpenDialog1.Execute then
   DataNameEdit.Text := OpenDialog1.FileName;
end;

procedure TSigningForm.SignBtnClick(Sender: TObject);
{Процеду призначена для підписання файлу}
var cont: PChar;
    err: string;
    hProv: HCRYPTPROV;
    key: HCRYPTKEY;
    alg: ALG_ID;
    hash: HCRYPTHASH;
    infile, outfile: file;
    size: DWORD;
    buf: array [0..511] of byte;
    signature: PBYTE;
begin
{Перевірка на наявність вибраного файлу}
if not FileExists(DataNameEdit.Text) then
   begin
   MessageDlg('Неправильне ім'я файлу!', mtError, [mbOK], 0);
```

```
   exit;
   end;
AssignFile(infile, DataNameEdit.Text);
{Зчитуємо ім'я контейнера}
if length(ContainerEdit.Text) = 0
then
    begin
    cont := nil;
    err := 'по замовченню';
    end
else
    begin
    err := ContainerEdit.Text;
    cont := StrAlloc(length(err) + 1);
    StrPCopy(cont, err);
    end;
{Підключаємось до криптопровайдера і отримуємо
 його контекст - підключаємося до вибраного контейнера}
if not CryptAcquireContext(@hProv, cont, nil, PROV_RSA_FULL, 0)
then
    begin
{обробка помилок}
    case int64(GetLastError) of
    ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
    ERROR_NOT_ENOUGH_MEMORY: err := 'ERROR_NOT_ENOUGH_MEMORY';
    NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
    NTE_BAD_KEYSET: err := 'NTE_BAD_KEYSET';
    NTE_BAD_KEYSET_PARAM: err := 'NTE_BAD_KEYSET_PARAM';
    NTE_BAD_PROV_TYPE: err := 'NTE_BAD_PROV_TYPE';
    NTE_BAD_SIGNATURE: err := 'NTE_BAD_SIGNATURE';
    NTE_EXISTS: err := 'NTE_EXISTS';
    NTE_KEYSET_ENTRY_BAD: err := 'NTE_KEYSET_ENTRY_BAD';
    NTE_KEYSET_NOT_DEF: err := 'NTE_KEYSET_NOT_DEF';
    NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
    NTE_PROV_DLL_NOT_FOUND: err := 'NTE_PROV_DLL_NOT_FOUND';
    NTE_PROV_TYPE_ENTRY_BAD: err := 'NTE_PROV_TYPE_ENTRY_BAD';
    NTE_PROV_TYPE_NO_MATCH: err := 'NTE_PROV_TYPE_NO_MATCH';
    NTE_PROV_TYPE_NOT_DEF: err := 'NTE_PROV_TYPE_NOT_DEF';
    NTE_PROVIDER_DLL_FAIL: err := 'NTE_PROVIDER_DLL_FAIL';
    NTE_SIGNATURE_FILE_BAD: err := 'NTE_SIGNATURE_FILE_BAD';
    else err := 'Unknown error';
    end;
    MessageDlg('Помилка відкриття контейнеру: ' + err, mtError, [mbOK], 0);
    exit;
    end;
{визначаємо алгоритм хешування}
case HashRadioGroup.ItemIndex of
0: alg := CALG_MD5;
1: alg := CALG_SHA;
end;
{створюємо хеш-об'єкт}
if not CryptCreateHash(hProv, alg, 0, 0, @hash) then
   begin
{обробка помилок}
   case int64(GetLastError) of
   ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
   ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
   ERROR_NOT_ENOUGH_MEMORY: err := 'ERROR_NOT_ENOUGH_MEMORY';
   NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
   NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
   NTE_BAD_KEY: err := 'NTE_BAD_KEY';
   NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
   else err := 'Unknown error';
   end;
   MessageDlg('Помилка створення хеш-об'єкту: ' + err, mtError, [mbOK], 0);
   exit;
   end;
{Визначаємо ім'я файлу для зберігання підписаних даних}
SaveDialog1.Title := 'Задайте ім'я файлу для зберігання підписаних даних';
```

```
if SaveDialog1.Execute then
   begin
   AssignFile(outfile, SaveDialog1.FileName);
   rewrite(outfile, 1);
{Записуємо в файл ІДЕНТИФІКАТОР АЛГОРИТМА ХЕШУВАННЯ}
   BlockWrite(outfile, alg, 4);
   reset(infile, 1);
   size := FileSize(infile);
{Записуєм РОЗМІР ПІДПИСАНИХ ДАНИХ}
   BlockWrite(outfile, size, 4);
{Записуємо ДАНІ і РОЗРАХОВУЄМО ХЕШ}
   while not eof(infile) do
         begin
         BlockRead(infile, buf, 512, size);
         BlockWrite(outFile, buf, size);
{Хешуємо пароль}
         if not CryptHashData(hash, @buf, size, 0) then
            begin
{обробка помилок}
            case int64(GetLastError) of
            ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
            ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
            NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
            NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
            NTE_BAD_HASH: err := 'NTE_BAD_HASH';
            NTE_BAD_HASH_STATE: err := 'NTE_BAD_HASH_STATE';
            NTE_BAD_KEY: err := 'NTE_BAD_KEY';
            NTE_BAD_LEN: err := 'NTE_BAD_LEN';
            NTE_BAD_UID: err := 'NTE_BAD_UID';
            NTE_FAIL: err := 'NTE_FAIL';
            NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
            else err := 'Unknown error';
            end;
            MessageDlg('Помилка при хешуванні: ' + err, mtError, [mbOK], 0);
            break;
            end;
         end;
   CloseFile(infile);
{Визначаємо розмір підпису}
   if not CryptSignHash(hash, AT_SIGNATURE, nil, 0, nil, @size) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      ERROR_MORE_DATA: err := 'ERROR_MORE_DATA';
      NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
      NTE_BAD_HASH: err := 'NTE_BAD_HASH';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_NO_KEY: err := 'NTE_NO_KEY';
      NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка при визначенні розміру підпису: ' + err, mtError,
[mbOK], 0);
      CloseFile(outfile);
      exit;
      end;
{Створюємо підпис}
   GetMem(signature, size);
   if not CryptSignHash(hash, AT_SIGNATURE, nil, 0, signature, @size) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      ERROR_MORE_DATA: err := 'ERROR_MORE_DATA';
      NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
```

```
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
      NTE_BAD_HASH: err := 'NTE_BAD_HASH';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_NO_KEY: err := 'NTE_NO_KEY';
      NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка при підписані хешу: ' + err, mtError, [mbOK], 0);
      CloseFile(outfile);
      exit;
      end;
{Записуємо ПІДПИС}
   BlockWrite(outfile, size, 4);
   BlockWrite(outfile, signature^, size);
   CloseFile(outfile);
   end;
{Знищуємо хеш-об'єкт}
if not CryptDestroyHash(hash) then
   begin
{обробка помилок}
   case int64(GetLastError) of
   ERROR_BUSY: err := 'ERROR_BUSY';
   ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
   ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
   NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
   NTE_BAD_HASH: err := 'NTE_BAD_HASH';
   NTE_BAD_UID: err := 'NTE_BAD_UID';
   else err := 'Unknown error';
   end;
   MessageDlg('Помилка при знищенні хешу: ' + err, mtError, [mbOK], 0);
   end;
{Звільняємо контекст криптопровайдера}
if not CryptReleaseContext(hProv, 0) then
   begin
{обробка помилок}
   case int64(GetLastError) of
   ERROR_BUSY: err := 'ERROR_BUSY';
   ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
   ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
   NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
   NTE_BAD_UID: err := 'NTE_BAD_UID';
   else err := 'Unknown error';
   end;
   MessageDlg('Помилка при звільненні контексту: ' + err,
       mtError, [mbOK], 0);
   end;
end;

procedure TSigningForm.DataNameEditChange(Sender: TObject);
{Якщо не вибраний файл, то кнопка Підписати неактивна}
begin
SignBtn.Enabled := (length(DataNameEdit.Text) > 0);
end;

end.
```

Generate.pas

```
unit Generate;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComCtrls, StdCtrls, wcrypt2, Buttons;

type
  TGenerateForm = class(TForm)
    Label1: TLabel;
    ContainerEdit: TEdit;
    Label2: TLabel;
    KeyExchLenEdit: TEdit;
    UpDown1: TUpDown;
    Label3: TLabel;
    SignKeyLenEdit: TEdit;
    UpDown2: TUpDown;
    ReportMemo: TMemo;
    KEKCheckBox: TCheckBox;
    SKCheckBox: TCheckBox;
    OKBtn: TBitBtn;
    CloseBtn: TBitBtn;
    ClearBtn: TBitBtn;
    procedure OKBtnClick(Sender: TObject);
    procedure ClearBtnClick(Sender: TObject);
    procedure CloseBtnClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  GenerateForm: TGenerateForm;

implementation

{$R *.DFM}

function algIDtostr(a: DWORD): string;
{функція переводу типу DWORD алгоритму шифрування в string
ІДЕНТИФІКАТОР АЛГОРИТМУ}
begin
case a of
CALG_HMAC: algIDtostr := 'HMAC keyed hash algorithm';
CALG_MD2: algIDtostr := 'MD2 hashing algorithm';
CALG_MD4: algIDtostr := 'MD4 hashing algorithm';
CALG_MD5: algIDtostr := 'MD5 hashing algorithm';
CALG_SHA: algIDtostr := 'SHA hashing algorithm';
CALG_MAC: algIDtostr := 'MAC keyed hash algorithm';
CALG_SSL3_SHAMD5: algIDtostr := 'SSL3 client authentication';
CALG_RSA_SIGN: algIDtostr := 'RSA public-key signature algorithm';
CALG_DSS_SIGN: algIDtostr := 'DSA public-key signature algorithm';
CALG_RSA_KEYX: algIDtostr := 'RSA public-key key exchange algorithm';
CALG_DES: algIDtostr := 'DES encryption algorithm';
CALG_RC2: algIDtostr := 'RC2 block encryption algorithm';
CALG_RC4: algIDtostr := 'RC4 stream encryption algorithm';
CALG_SEAL: algIDtostr := 'SEAL encryption algorithm';
CALG_DH_SF: algIDtostr := 'Diffie-Hellman store and forward key exchange
algorithm';
else algIDtostr := 'unknown algorithm';
end;
end;

procedure TGenerateForm.OKBtnClick(Sender: TObject);
{ГЕНЕРАЦІЯ КЛЮЧА ОБМІНУ КЛЮЧАМИ
```

```
 ГЕНЕРАЦІЯ КЛЮЧА ПІДПИСУ}
var cont: PChar;
    err: string;
    hProv: HCRYPTPROV;
    KeyExchKey, SignKey: HCRYPTKEY;
    flag, keyLen: DWORD;
begin
{Вихід, якщо не вибраний жоден ключ}
if not (KEKCheckBox.Checked or SKCheckBox.Checked) then exit;
{Зчитуємо ім'я контейнера}
if length(ContainerEdit.Text) = 0
then
    begin
    cont := nil;
    err := 'по замовченню';
    end
else
    begin
    err := ContainerEdit.Text;
    cont := StrAlloc(length(err) + 1);
    StrPCopy(cont, err);
    end;
{Отримуємо дескриптор провайдера}
if not CryptAcquireContext(@hProv, cont, nil, PROV_RSA_FULL, 0)
then
    begin
{обробка помилок}
    case int64(GetLastError) of
    ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
    ERROR_NOT_ENOUGH_MEMORY: err := 'ERROR_NOT_ENOUGH_MEMORY';
    NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
    NTE_BAD_KEYSET: err := 'NTE_BAD_KEYSET';
    NTE_BAD_KEYSET_PARAM: err := 'NTE_BAD_KEYSET_PARAM';
    NTE_BAD_PROV_TYPE: err := 'NTE_BAD_PROV_TYPE';
    NTE_BAD_SIGNATURE: err := 'NTE_BAD_SIGNATURE';
    NTE_EXISTS: err := 'NTE_EXISTS';
    NTE_KEYSET_ENTRY_BAD: err := 'NTE_KEYSET_ENTRY_BAD';
    NTE_KEYSET_NOT_DEF: err := 'NTE_KEYSET_NOT_DEF';
    NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
    NTE_PROV_DLL_NOT_FOUND: err := 'NTE_PROV_DLL_NOT_FOUND';
    NTE_PROV_TYPE_ENTRY_BAD: err := 'NTE_PROV_TYPE_ENTRY_BAD';
    NTE_PROV_TYPE_NO_MATCH: err := 'NTE_PROV_TYPE_NO_MATCH';
    NTE_PROV_TYPE_NOT_DEF: err := 'NTE_PROV_TYPE_NOT_DEF';
    NTE_PROVIDER_DLL_FAIL: err := 'NTE_PROVIDER_DLL_FAIL';
    NTE_SIGNATURE_FILE_BAD: err := 'NTE_SIGNATURE_FILE_BAD';
    else err := 'Unknown error';
    end;
    MessageDlg('Помилка відкриття контейнера: ' + err, mtError, [mbOK], 0);
    exit;
    end;
{ГЕНЕРАЦІЯ КЛЮЧА ОБМІНУ КЛЮЧАМИ}
if KEKCheckBox.Checked then
   begin
{зчитуємо довжину ключа і розміщуємо її в старше слово параметра *прапорці*}
    keyLen := strtoint(KeyExchLenEdit.text);
    flag := keyLen shl 16;
{генеруємо ключеву пару обміну ключами}
   if not CryptGenKey(hProv, AT_KEYEXCHANGE, flag, @KeyExchKey) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
      NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_FAIL: err := 'NTE_FAIL';
      else err := 'Unknown error';
      end;
```

```
      MessageDlg('Помилка створення ключа обміну ключами: ' + err,
                         mtError, [mbOK], 0);
      end
   else
       begin
       ReportMemo.Lines.Add('');
       ReportMemo.Lines.Add('Створений ключ обміну ключами:');
       flag := 4;
{отримуємо параметри ключа - ідентифікатор алгоритма}
       if not CryptGetKeyParam(KeyExchKey, KP_KEYLEN, @keyLen, @flag, 0) then
          begin
          case int64(GetLastError) of
          ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
          ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
          ERROR_MORE_DATA: err := 'ERROR_MORE_DATA';
          NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
          NTE_BAD_KEY: err := 'NTE_BAD_KEY';
          NTE_NO_KEY: err := 'NTE_NO_KEY';
          NTE_BAD_TYPE: err := 'NTE_BAD_TYPE';
          NTE_BAD_UID: err := 'NTE_BAD_UID';
          else err := 'Unknown error';
          end;
          MessageDlg('Помилка при отриманні довжини ключа: ' + err,
                          mtError, [mbOK], 0);
          end
       else ReportMemo.Lines.Add('  довжина ключа - ' + inttostr(keyLen));
       flag := 4;
{отримуємо параметри ключа - розмір ключа}
       if not CryptGetKeyParam(KeyExchKey, KP_ALGID, @keyLen, @flag, 0) then
          begin
{обробка помилок}
          case int64(GetLastError) of
          ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
          ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
          ERROR_MORE_DATA: err := 'ERROR_MORE_DATA';
          NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
          NTE_BAD_KEY: err := 'NTE_BAD_KEY';
          NTE_NO_KEY: err := 'NTE_NO_KEY';
          NTE_BAD_TYPE: err := 'NTE_BAD_TYPE';
          NTE_BAD_UID: err := 'NTE_BAD_UID';
          else err := 'Unknown error';
          end;
          MessageDlg('Помилка при отриманні довжини ключа: ' + err,
                          mtError, [mbOK], 0);
          end
       else ReportMemo.Lines.Add('  алгоритм - ' + algIDtostr(keyLen));
       end;
   end;
{ГЕНЕРАЦІЯ КЛЮЧА ПІДПИСУ}
if SKCheckBox.Checked then
   begin
{считуємо довжину ключа і розміщуємо її в старше слово параметра *прапорці*}
   keyLen := strtoint(SignKeyLenEdit.text);
   flag := keyLen shl 16;
{генеруємо ключ підпису}
   if not CryptGenKey(hProv, AT_SIGNATURE, flag, @SignKey) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
      NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_FAIL: err := 'NTE_FAIL';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка при створенні ключа підпису: ' + err,
                         mtError, [mbOK], 0);
```

```
      end
   else
       begin
       ReportMemo.Lines.Add('');
       ReportMemo.Lines.Add('Створений ключ підпису:');
       flag := 4;
{отримання параметрів ключа підпису - розмір ключа}
       if not CryptGetKeyParam(SignKey, KP_KEYLEN, @keyLen, @flag, 0) then
          begin
{обробка помилок}
          case int64(GetLastError) of
          ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
          ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
          ERROR_MORE_DATA: err := 'ERROR_MORE_DATA';
          NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
          NTE_BAD_KEY: err := 'NTE_BAD_KEY';
          NTE_NO_KEY: err := 'NTE_NO_KEY';
          NTE_BAD_TYPE: err := 'NTE_BAD_TYPE';
          NTE_BAD_UID: err := 'NTE_BAD_UID';
          else err := 'Unknown error';
          end;
          MessageDlg('Помилка при отриманні довжини ключа: ' + err,
                         mtError, [mbOK], 0);
          end
       else ReportMemo.Lines.Add('  довжина ключа - ' + inttostr(keyLen));
       flag := 4;
{отримання параметрів ключа підпису - ідентифікатор алгоритма}
       if not CryptGetKeyParam(SignKey, KP_ALGID, @keyLen, @flag, 0) then
          begin
{обробка помилок}
          case int64(GetLastError) of
          ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
          ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
          ERROR_MORE_DATA: err := 'ERROR_MORE_DATA';
          NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
          NTE_BAD_KEY: err := 'NTE_BAD_KEY';
          NTE_NO_KEY: err := 'NTE_NO_KEY';
          NTE_BAD_TYPE: err := 'NTE_BAD_TYPE';
          NTE_BAD_UID: err := 'NTE_BAD_UID';
          else err := 'Unknown error';
          end;
          MessageDlg('Помилка при отриманні довжини ключа: ' + err,
                         mtError, [mbOK], 0);
          end
       else ReportMemo.Lines.Add('  алгоритм - ' + algIDtostr(keyLen));
       end;
   end;
{Звільнюємо контекст криптопровайдера}
if not CryptReleaseContext(hProv, 0) then
   begin
{обробка помилок}
   case int64(GetLastError) of
   ERROR_BUSY: err := 'ERROR_BUSY';
   ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
   ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
   NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
   NTE_BAD_UID: err := 'NTE_BAD_UID';
   else err := 'Unknown error';
   end;
   MessageDlg('Помилка при звільненні контексту: ' + err,
       mtError, [mbOK], 0);
   end;
end;

procedure TGenerateForm.ClearBtnClick(Sender: TObject);
{очистка поля мемо}
begin
ReportMemo.Lines.Clear;
end;
```

```
procedure TGenerateForm.CloseBtnClick(Sender: TObject);
{закриття форми}
begin
Close;
end;

end.
```

KExport.pas

```
unit KExport;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  wcrypt2, StdCtrls, Buttons, ExtCtrls;

type
  TExportForm = class(TForm)
    Label1: TLabel;
    ContainerEdit: TEdit;
    SaveDialog1: TSaveDialog;
    ExportGroupBox: TGroupBox;
    KEKCheckBox: TCheckBox;
    SKCheckBox: TCheckBox;
    OKBtn: TBitBtn;
    CloseBtn: TBitBtn;
    WhatRadioGroup: TRadioGroup;
    PasswLabel: TLabel;
    PasswEdit: TEdit;
    Passw2Edit: TEdit;
    Passw2Label: TLabel;
    procedure CloseBtnClick(Sender: TObject);
    procedure OKBtnClick(Sender: TObject);
    procedure WhatRadioGroupClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  ExportForm: TExportForm;

implementation

{$R *.DFM}

procedure TExportForm.CloseBtnClick(Sender: TObject);
{Закриття форми Експорт і повернення до головної форми}
begin
  Close;
end;

procedure TExportForm.OKBtnClick(Sender: TObject);
{Експортування ключів}
var cont: PChar;
    err: string;
    hProv: HCRYPTPROV;
    key, expKey: HCRYPTKEY;
    pbuf: PBYTE;
    buflen: DWORD;
    f: file;
    hash: HCRYPTHASH;
begin
{Якщо жоден ключ для експорту не вибраний, то вихід}
if not (KEKCheckBox.Checked or SKCheckBox.Checked) then exit;
{Якщо експортується ключова пара, то становиться активним запит Пароль}
if PasswEdit.Enabled and (PasswEdit.Text <> Passw2Edit.Text) then
   begin
   MessageDlg('Помилка при введені паролю! Повторіть введення.', mtError,
[mbOK], 0);
   exit;
   end;
{Зчитуємо ім'я контейнера}
if length(ContainerEdit.Text) = 0
```

```
then
    begin
    cont := nil;
    err := 'по замовченню';
    end
else
    begin
    err := ContainerEdit.Text;
    cont := StrAlloc(length(err) + 1);
    StrPCopy(cont, err);
    end;
{Отримуємо контекст криптопровайдера (підключаємося до криптопровайдера)}
if not CryptAcquireContext(@hProv, cont, nil, PROV_RSA_FULL, 0)
then
{обробка помилок}
    begin
    case int64(GetLastError) of
    ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
    ERROR_NOT_ENOUGH_MEMORY: err := 'ERROR_NOT_ENOUGH_MEMORY';
    NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
    NTE_BAD_KEYSET: err := 'NTE_BAD_KEYSET';
    NTE_BAD_KEYSET_PARAM: err := 'NTE_BAD_KEYSET_PARAM';
    NTE_BAD_PROV_TYPE: err := 'NTE_BAD_PROV_TYPE';
    NTE_BAD_SIGNATURE: err := 'NTE_BAD_SIGNATURE';
    NTE_EXISTS: err := 'NTE_EXISTS';
    NTE_KEYSET_ENTRY_BAD: err := 'NTE_KEYSET_ENTRY_BAD';
    NTE_KEYSET_NOT_DEF: err := 'NTE_KEYSET_NOT_DEF';
    NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
    NTE_PROV_DLL_NOT_FOUND: err := 'NTE_PROV_DLL_NOT_FOUND';
    NTE_PROV_TYPE_ENTRY_BAD: err := 'NTE_PROV_TYPE_ENTRY_BAD';
    NTE_PROV_TYPE_NO_MATCH: err := 'NTE_PROV_TYPE_NO_MATCH';
    NTE_PROV_TYPE_NOT_DEF: err := 'NTE_PROV_TYPE_NOT_DEF';
    NTE_PROVIDER_DLL_FAIL: err := 'NTE_PROVIDER_DLL_FAIL';
    NTE_SIGNATURE_FILE_BAD: err := 'NTE_SIGNATURE_FILE_BAD';
    else err := 'Unknown error';
    end;
    MessageDlg('Помилка відкриття контейнеру: ' + err, mtError, [mbOK], 0);
    exit;
    end;
{Якщо вибрано *ключеву пару*, то створюємо ключ шифрування на основі пароля }
if WhatRadioGroup.ItemIndex > 0 then
   begin
{створюємо хеш-об'єкт}
   if not CryptCreateHash(hProv, CALG_SHA, 0, 0, @hash) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      ERROR_NOT_ENOUGH_MEMORY: err := 'ERROR_NOT_ENOUGH_MEMORY';
      NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
      NTE_BAD_KEY: err := 'NTE_BAD_KEY';
      NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка створеня хеш-об''єкту: ' + err, mtError, [mbOK], 0);
      exit;
      end;
{хешуємо пароль}
   if not CryptHashData(hash, @(PasswEdit.Text[1]), length(PasswEdit.Text), 0)
then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
```

```
      NTE_BAD_HASH: err := 'NTE_BAD_HASH';
      NTE_BAD_HASH_STATE: err := 'NTE_BAD_HASH_STATE';
      NTE_BAD_KEY: err := 'NTE_BAD_KEY';
      NTE_BAD_LEN: err := 'NTE_BAD_LEN';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_FAIL: err := 'NTE_FAIL';
      NTE_NO_MEMORY: err := 'NTE_NO_MEMORY';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка при хешуванні: ' + err, mtError, [mbOK], 0);
      exit;
      end;
{створюємо ключ на основі паролю для потокового шифру RC4}
   if not CryptDeriveKey(hProv, CALG_RC4, hash, 0, @expKey) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
      NTE_BAD_HASH: err := 'NTE_BAD_HASH';
      NTE_BAD_HASH_STATE: err := 'NTE_BAD_HASH_STATE';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_FAIL: err := 'NTE_FAIL';
      else err := 'Unknown error';
      end;
      MessageDlg('Error of CryptHashData: '+err,
                        mtError, [mbOK], 0);
      exit;
      end;
{знищуємо хеш-об'єкт}
   if not CryptDestroyHash(hash) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_BUSY: err := 'ERROR_BUSY';
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      NTE_BAD_ALGID: err := 'NTE_BAD_ALGID';
      NTE_BAD_HASH: err := 'NTE_BAD_HASH';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка при знищенні хешу: ' + err, mtError, [mbOK], 0);
      end;
   end;
{Якщо на панелі Експортувати вибрано *ключ обміну ключами* то}
if KEKCheckBox.Checked then
 repeat
{отримуємо дескриптор ключа)}
   if not CryptGetUserKey(hProv, AT_KEYEXCHANGE, @key) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      NTE_BAD_KEY: err := 'NTE_BAD_KEY';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_NO_KEY: err := 'NTE_NO_KEY';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка при отриманні ключа обміну ключами: ' + err,
                        mtError, [mbOK], 0);
      break;
      end;
{визначаємо розмір буфера для експорта ключа}
   {$B-}
   if ((WhatRadioGroup.ItemIndex = 0) and
```

```
      not CryptExportKey(key, 0, PUBLICKEYBLOB, 0, nil, @bufLen))
      or ((WhatRadioGroup.ItemIndex = 1) and
      not CryptExportKey(key, expKey, PRIVATEKEYBLOB, 0, nil, @bufLen)) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      ERROR_MORE_DATA: err := 'ERROR_MORE_DATA';
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
      NTE_BAD_KEY: err := 'NTE_BAD_KEY';
      NTE_BAD_KEY_STATE: err := 'NTE_BAD_KEY_STATE';
      NTE_BAD_PUBLIC_KEY: err := 'NTE_BAD_PUBLIC_KEY';
      NTE_BAD_TYPE: err := 'NTE_BAD_TYPE';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_NO_KEY: err := 'NTE_NO_KEY';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка отримання розміру BLOB''а ключа обміну ключами: ' +
err,
                        mtError, [mbOK], 0);
      break;
      end;
   GetMem(pbuf, bufLen);
{експортуємо дані}
   if ((WhatRadioGroup.ItemIndex = 0) and
      not CryptExportKey(key, 0, PUBLICKEYBLOB, 0, pbuf, @bufLen))
      or ((WhatRadioGroup.ItemIndex = 1) and
      not CryptExportKey(key, expKey, PRIVATEKEYBLOB, 0, pbuf, @bufLen)) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      ERROR_MORE_DATA: err := 'ERROR_MORE_DATA';
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
      NTE_BAD_KEY: err := 'NTE_BAD_KEY';
      NTE_BAD_KEY_STATE: err := 'NTE_BAD_KEY_STATE';
      NTE_BAD_PUBLIC_KEY: err := 'NTE_BAD_PUBLIC_KEY';
      NTE_BAD_TYPE: err := 'NTE_BAD_TYPE';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_NO_KEY: err := 'NTE_NO_KEY';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка експорту ключа обміну ключами: ' + err,
                        mtError, [mbOK], 0);
      break;
      end;
{Знищуємо дескриптор *ключа обміну ключами*, при цьому сам ключ не знищується}
   if not CryptDestroyKey(key) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      NTE_BAD_KEY: err := 'NTE_BAD_KEY';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка звільнення дескриптора ключа обміну ключами: ' + err,
                        mtError, [mbOK], 0);
      end;
   SaveDialog1.Title := 'Вкажіть файл для збереження ключа обміну ключами';
{зберігаємо ключ обміну ключами в зовнішньому файлі}
   if SaveDialog1.Execute then
      begin
      AssignFile(f, SaveDialog1.FileName);
      rewrite(f, 1);
      BlockWrite(f, pbuf^, bufLen);
```

```
      CloseFile(f);
      MessageDlg('Ключ обміну ключами успішно збережений', mtInformation,
[mbOK], 0);
      end;
 until true; {KeyExchange}

{Якщо на панелі Експортувати вибрано *ключ підпису* то}
if SKCheckBox.Checked then
 repeat
{отримуємо дескриптор ключа (рос. указатель)}
   if not CryptGetUserKey(hProv, AT_SIGNATURE, @key) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      NTE_BAD_KEY: err := 'NTE_BAD_KEY';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_NO_KEY: err := 'NTE_NO_KEY';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка отримання ключа підпису: ' + err,
                        mtError, [mbOK], 0);
      break;
      end;
{визначаємо розмір буфера для експорта ключа}
   {$B-}
   if ((WhatRadioGroup.ItemIndex = 0) and
      not CryptExportKey(key, 0, PUBLICKEYBLOB, 0, nil, @bufLen))
      or ((WhatRadioGroup.ItemIndex = 1) and
      not CryptExportKey(key, expKey, PRIVATEKEYBLOB, 0, nil, @bufLen)) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      ERROR_MORE_DATA: err := 'ERROR_MORE_DATA';
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
      NTE_BAD_KEY: err := 'NTE_BAD_KEY';
      NTE_BAD_KEY_STATE: err := 'NTE_BAD_KEY_STATE';
      NTE_BAD_PUBLIC_KEY: err := 'NTE_BAD_PUBLIC_KEY';
      NTE_BAD_TYPE: err := 'NTE_BAD_TYPE';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      NTE_NO_KEY: err := 'NTE_NO_KEY';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка отримання розміру BLOB''а ключа підпису: ' + err,
                        mtError, [mbOK], 0);
      break;
      end;
   GetMem(pbuf, bufLen);

{експортуємо дані}
   if ((WhatRadioGroup.ItemIndex = 0) and
      not CryptExportKey(key, 0, PUBLICKEYBLOB, 0, pbuf, @bufLen))
      or ((WhatRadioGroup.ItemIndex = 1) and
      not CryptExportKey(key, expKey, PRIVATEKEYBLOB, 0, pbuf, @bufLen)) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      ERROR_MORE_DATA: err := 'ERROR_MORE_DATA';
      NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
      NTE_BAD_KEY: err := 'NTE_BAD_KEY';
      NTE_BAD_KEY_STATE: err := 'NTE_BAD_KEY_STATE';
      NTE_BAD_PUBLIC_KEY: err := 'NTE_BAD_PUBLIC_KEY';
      NTE_BAD_TYPE: err := 'NTE_BAD_TYPE';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
```

```
      NTE_NO_KEY: err := 'NTE_NO_KEY';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка експорта ключа підпису: ' + err,
                          mtError, [mbOK], 0);
      break;
      end;
{Знищуємо дескриптор *ключа підпису*, при цьому сам ключ не знищується}
   if not CryptDestroyKey(key) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      NTE_BAD_KEY: err := 'NTE_BAD_KEY';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка звільнення дескриптора ключа підпису: ' + err,
                          mtError, [mbOK], 0);
      end;
{зберігаємо ключ підпису в зовнішньому файлі}
   SaveDialog1.Title := 'Вкажіть файл для збереження ключа підпису';
   if SaveDialog1.Execute then
      begin
      AssignFile(f, SaveDialog1.FileName);
      rewrite(f, 1);
      BlockWrite(f, pbuf^, bufLen);
      CloseFile(f);
      MessageDlg('Ключ підпису успішно збережений', mtInformation, [mbOK], 0);
      end;
 until true; {Signature}

{Якщо  ключ створювався за допомогою пароля, то знищуємо його}
if WhatRadioGroup.ItemIndex > 0 then
   if not CryptDestroyKey(expKey) then
      begin
{обробка помилок}
      case int64(GetLastError) of
      ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
      ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
      NTE_BAD_KEY: err := 'NTE_BAD_KEY';
      NTE_BAD_UID: err := 'NTE_BAD_UID';
      else err := 'Unknown error';
      end;
      MessageDlg('Помилка знищення ключа шифрування BLOB''а: ' + err,
                          mtError, [mbOK], 0);
      end;
{Знищуємо(звільняємо контекст криптопровайдера) }
if not CryptReleaseContext(hProv, 0) then
   begin
{обробка помилок}
   case int64(GetLastError) of
   ERROR_BUSY: err := 'ERROR_BUSY';
   ERROR_INVALID_HANDLE: err := 'ERROR_INVALID_HANDLE';
   ERROR_INVALID_PARAMETER: err := 'ERROR_INVALID_PARAMETER';
   NTE_BAD_FLAGS: err := 'NTE_BAD_FLAGS';
   NTE_BAD_UID: err := 'NTE_BAD_UID';
   else err := 'Unknown error';
   end;
   MessageDlg('Помилка при звільненні контексту: ' + err,
       mtError, [mbOK], 0);
   end;
end;

procedure TExportForm.WhatRadioGroupClick(Sender: TObject);
{При виборі експортувати ключову пару стає активним запит Пароль}
var e: boolean;
begin
```

```
e := (WhatRadioGroup.ItemIndex > 0);
PasswEdit.Enabled := e;
PasswLabel.Enabled := e;
Passw2Edit.Enabled := e;
Passw2Label.Enabled := e;
end;

end.
```

ServerMain.dpr

```
program ServerMain;
//файл проекту серверна частина
uses
  Forms,
  Server_main in 'Server_main.pas' {Form1};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Server_main.pas

```
unit Server_main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ScktComp, ExtCtrls, ComCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    ServerSocket1: TServerSocket;
    ClientSocket1: TClientSocket;
    Label1: TLabel;
    Label2: TLabel;
    StatusBar1: TStatusBar;
    Memo1: TMemo;
    procedure ClientSocket1Read(Sender: TObject; Socket: TCustomWinSocket);
    procedure writing(Text:string);
    procedure FormCreate(Sender: TObject);//запис даних в буфер
  private
    temp: String;
    Name: string; //ім'я файлу
    Size: Integer;//Розмір файлу
    Receive: Boolean;//Режим роботи - стан прапорця
    MS: TMemoryStream;//буфер для файлу

    { Private declarations }
  public
    IsRecStart: Boolean; // Ознака початку отримання даних
    BmpFile: File; // Файл для передачі
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.ClientSocket1Read(Sender: TObject;
  Socket: TCustomWinSocket);
var
  Rtext: string;
begin
  Rtext:=Socket.ReceiveText;
  if Receive then
    Writing(RText)
  else
    if Copy(Rtext,0,Pos('#',Rtext)-1)='file' then
      begin
        MS:=TMemoryStream.Create;
        Delete(Rtext,1,Pos('#',Rtext));
        Name:= Copy(Rtext,0,Pos('#',Rtext)-1);
        Delete(Rtext,1,Pos('#',Rtext));
        Size:=StrToInt(Copy(Rtext,0,Pos('#',Rtext)-1));
        Delete(Rtext,1,Pos('#',Rtext));
        Label1.Caption:='Розмір файлу: '+IntToStr(Size) +' байт';
        Label2.Caption:='Ім''я файлу: '+ Name;
        Receive:=True;
        Writing(RText);
      end;
end;


procedure TForm1.writing(Text: string);
```

```
begin
  if MS.Size<Size then
  //якщо прийнято байт менший розміру файла, то
  //записуємо у буфер
  MS.Write(Text[1],Length(Text));
  //виводимо дані в рядок статуса
  StatusBar1.SimpleText:='Прийнято'+IntToStr(MS.Size)+'з'+IntToStr(Size);
  //якщо файл прийнятий
  if MS.Size=Size then
    begin
      //встановлюємо прапорець в False
      Receive:=False;
      //на початок буфера
      MS.Position:=0;
      //збереження файлу
      MS.SaveToFile('Arhive\'+Name);
      //відправка успішності прийняття файла
      ClientSocket1.Socket.SendText('end');
      //очистка буфера
      MS.Free;
      //виведення в рядок статуса інформації
      StatusBar1.SimpleText:='Файл одержаний';
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  FindHandle : THandle;
  FindData : TWin32FindData;
  b: boolean;
  s: string;

begin
 Memo1.Clear;
// встанавлюємо атрибути файла
  FindData.dwFileAttributes := FILE_ATTRIBUTE_NORMAL;
  // шукаємо перший файл в заданому каталозі:\
  FindHandle := FindFirstFile('Arhive\*.*', FindData);
  // якщо все ок, тоді продовжуємо
  if FindHandle <> INVALID_HANDLE_VALUE then
  begin
     b := true;
     while b do
     begin
       s := FindData.cFileName;
       // пропускаємо всякі крапки і двокрапки нам не потрібні
       if (s<>'..') and (s<>'.') then
         Memo1.Lines.Add(s);
       b := FindNextFile(FindHandle, FindData);
     end;
  end;
end;

end.
```