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

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації”

Виконав здобувач вищої освіти
IV курсу, групи КБ-20-3СК
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Нікіша М.І.
«____» _____ 2023 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Смірнов С.А.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

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

Спеціальність *125 “*Кібербезпека*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Кібербезпека*”*_

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

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

Нікіші Максиму Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації*

2. Керівник роботи *Смірнов Сергій Анатолійович, канд. техн. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 13-02 від 5.01.2023 року

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

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації*

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

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

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

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

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

КАЛЕНДАРНИЙ ПЛАН

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

Дата видачі завдання
« 17 » січня 2023 р.

Підпис керівника
_____ Смірнов С.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2023 р.

Підпис здобувача
_____ Нікіша М.І.
(прізвище та ініціали)

АНОТАЦІЯ

Нікіша М.І. Програмне забезпечення системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації.

Метою розробки є програмне забезпечення системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації.

Результат роботи – програмна реалізація системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації.

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

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

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

Програму розроблено в середовищі Delphi 10.4 Sydney.

Ключові слова: кібербезпека, генерація та розподіл ключів

ABSTRACT

Nikisha M.I. Software of the cyber security system of the center of generation and distribution of keys of the information protection system. 125 Cyber security. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the first (bachelor) level of higher education, software is developed, which is intended for the cyber security system of the center of generation and distribution of keys of the information protection system.

The purpose of the development is the software of the cyber security system of the center of generation and distribution of keys of the information protection system.

The result of the work is the software implementation of the cyber security system of the center of generation and distribution of keys of the information protection system.

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

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

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

The program was developed in the Delphi 10.4 Sydney environment.

Keywords: cyber security, key generation and distribution

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

БД	–	база даних
ЕОМ	–	електронна обчислювальна машина
ЕЦП	–	електронний цифровий підпис
ОС	–	операційна система
ПЗ	–	програмне забезпечення
САС	–	список анульованих сертифікатів
РЦ	–	розподільчий центр
УЦ	–	управляючий центр

ВСТУП

Актуальність теми. Одним із ключових завдань удосконалювання інформаційних комунікацій є завдання побудови безпечної інформаційної системи. Інтерес до неї обумовлений зростаючими обсягами конфіденційної інформації, переданої між учасниками інформаційного обміну, і швидким ростом таких показників інформації, як вартість втрати конфіденційності, вартість схованого порушення цілісності, вартість втрати інформації. Цій проблемі присвячена велика кількість наукових робіт і монографій. Захищеність комунікацій у безпечній інформаційній системі включає забезпечення конфіденційності й цілісності переданої інформації. Ці властивості забезпечуються використовуваними криптографічними системами, успішне функціонування яких припускає використання на приймальній й передавальній сторонах захищеного каналу криптографічних ключів, бінарних наборів достатньої довжини.

Для спрощення процедури генерації й розподілу секретних ключів у криптографії запропоновано й досліджено велику розмаїтість схем попереднього розподілу ключів. У них процедура доставки секретного ключа учасникам інформаційної системи виконується у два етапи: кожному учасникові довіреним центром доставляється пакет ключової інформації (у вигляді набору двійкових слів достатньої довжини), склад якого (у вигляді списку номерів і, можливо, деякої додаткової відкритої інформації про ці набори) публікується. При цьому кожний учасник, знаючи состави пакетів і опубліковані дані, може, використовуючи тільки набори зі свого пакета, обчислити для захищеної комунікації з будь-яким іншим учасником інформаційній системі ключ, що не може обчислити ніякий третій учасник.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації.

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

– Огляд існуючих систем центру генерації та розподілу ключів системи захисту інформації.

– Дослідження системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації.

– Програмна реалізація системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі центру генерації та розподілу ключів системи захисту інформації.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Розвиваючи підхід розробки центрів генерації та розподілу ключів у інформаційній системі, криптографи запропонували й схеми більше загального характеру, що дозволяють попередньо розподіляти ключову інформацію, яка дозволяє обчислювати ключі для спілкування привілейованих груп учасників, недоступні забороненим групам учасників. Але й цей підхід зустрічає труднощі стосовно до великих обчислювальних інформаційних систем, оскільки припускає використання єдиного центра генерації й доставки пакетів ключової інформації кожному учасникові інформаційної системи. Використання для доставки пакетів криптосистем відкритим ключем припускає сертифікацію відритих ключів учасниками.

До 1976 року використовувалися лише симетричні криптосистеми, у яких ключі передавальної й приймаючої сторони повинні бути секретними й практично однаковими, що пов'язане із проблемою доставки ключа від одного учасника іншому або від довіреного центра обом учасникам. З відкриттям У. Діффі, М. Хеллманом публічної криптографії один із ключів може бути відкритим і доставлятися його стороні, що використовує, не по закритому, а автентичному каналу, що вимагає реєстрації цього ключа в центрах сертифікації інфраструктури розподілу відкритих ключів. Варто помітити, що використання криптосистем з відкритим ключем саме по собі не досить для забезпечення захищеності комунікацій у комп'ютерних мережах,

Первісний одноразундовий двосторонній протокол Діффі-Хелмана, що використовує тільки відкриті канали піддається атаці третім учасником, шляхом перехоплення й підміни переданих посилок. Відбиття цієї атаки вимагає використання сертифікованих посилок, переданих через довірені вузли (протокол

Менезеса-кью-Венстоуна), оскільки алгоритми криптографії з відкритим ключем через необхідність виконання алгебраїчних операцій в алгебраїчних структурах високих порядків на три порядки повільніше алгоритмів симетричних криптосистем. Тому криптосистеми з відкритим ключем можуть використовуватися для захисту лише невеликих обсягів інформації й в основному відіграють допоміжну роль, забезпечуючи захист передачі секретних ключів для симетричних криптосистем. Безпосереднє використання цього способу доставки секретного ключа кожним учасником інформаційної системи приводить до багаторазового (по числу учасників) виконання дорогих актів сертифікації й використанню ключів, генеруємих учасниками без належного контролю їхньої якості. Винахід протоколу Kerberos частково спростило проблему, обмеживши число каналів, для яких необхідно розподіляти ключі «зовнішніми» засобами, стосовно домену інформаційної системи, що обслуговується протоколом. При цьому ключі для будь-якої пари учасників інформаційній системі виробляються довіреним центром і доставляються з використанням ключів, генеруємих центром автентифікації даного домена.

Розглянутий стан проблеми розподілу ключів дозволяє укласти, що актуально наступні завдання:

– розробка й обґрунтування архітектурних рішень генерації та нецентралізованого попереднього розподілу ключової інформації в інформаційній системі на основі незалежного попереднього її розподілу в сегментах або доменах інформаційної системи;

– розробка й обґрунтування способу реалізації попереднього розподілу ключової інформації в сегментах або доменах обчислювальної інформаційної системи на основі використання пропонованої модифікації протоколу Kerberos;

– розробка й обґрунтування нових схем попереднього розподілу ключів;

– розробка програмного забезпечення для обчислення пакетів ключової інформації й принципів побудови системного програмного забезпечення для обчислювальних систем з нецентралізованим попереднім розподілом ключів.

1.2 Область застосування

Відправник, що представляє собою джерело повідомлень, і одержувач (приймач зашифрованих повідомлень) домовляються про вибір прийнятного шифру й ключа. Потім відправник шифрує своє повідомлення з використанням обраного алгоритму шифрування й ключа й пересилає отриманий шифротекст по (відкритому) каналу зв'язку. Одержувач розшифровує його, використовуючи шифр і ключ.

Супротивник, швидше за все, може перехопити зашифроване повідомлення, тому що передбачається, що воно передається по відкритому каналу зв'язку. У цьому випадку криптоаналітик супротивника може спробувати розкрити шифротекст.

Будемо припускати, що відправник і одержувач повідомлення використовують досить надійний шифр, і що ймовірність його розкриття невисока.

У цьому випадку безпека шифрування повністю залежить від безпеки ключа. Розкриття ключа приведе до розкриття переданих даних. Таким чином, ключ повинен зберігатися в секреті доти, поки він використовується для закриття даних. Тому для первісного розподілу ключів необхідний надійний канал зв'язку.

Таким чином, принциповою є надійність каналу передачі ключа учасникам секретних переговорів. Самим надійним способом первісного розподілу ключів є обмін ключами при особистій зустрічі абонентів мережі передачі даних. Для доставки ключів можна також використовувати спеціальних кур'єрів.

Якщо в обміні секретними повідомленнями планується участь невеликої кількості сторін, наприклад, двох або трьох, то обоє зазначених способу цілком припустимі. Якщо ж кількість взаємодіючих абонентів велика, то завдання розподілу ключів перетворюється в справжню проблему.

При використанні секретних ключів існують і інші труднощі. Наприклад, ключі повинні час від часу мінятися. Це пов'язане з тим, що чим довше використовується ключ, тим більше ймовірність його компрометації (розкриття). Чим довше використовується ключ, тим більше втрати від його компрометації, тому що тим більша кількість повідомлень зможе розкрити зловмисник при одержанні ключа.

Навіть якщо ключ не буде розкритий, проводити криптоаналіз супротивникові зручніше, маючи у своєму розпорядженні достатню кількість повідомлень, зашифрованих тим самим ключем. Оптимальним уважається використовувати для кожного сеансу обміну зашифрованими повідомленнями свій унікальний ключ – так званий сеансовий ключ. Але де взяти таку кількість ключів для великої телекомунікаційної мережі і як їх розподіляти.

Таким чином, при великій кількості взаємодіючих сторін потрібне попереднє розсилання значної кількості ключів, а також наступне їхнє зберігання й при необхідності – зміна.

Припустимо, у локальній мережі є 100 користувачів. Нехай користувачі мережі бажають обмінюватися секретними даними один з одним за принципом "кожний з кожним". У цьому випадку для кожної пари користувачів необхідний свій секретний ключ для шифрування повідомлень. Зі ста користувачів можна скласти 100x99/2=4950 пар, отже, у системі передачі даних будуть використовуватися 4950 різних секретних ключів. Всі ці ключі повинні бути згенеровані й розподілені надійним образом.

Крім того, кожний зі ста користувачів повинен пам'ятати 99 різних ключів, кожний для певного абонента. Якщо ж в обміні повідомленнями бере участь не сто, а тисяча чоловік, то завдання керування ключами стає надзвичайно складним.

У зв'язку із зазначеними труднощами на практиці застосовуються спеціальні автоматизовані системи керування ключами. Такі системи дозволяють

генерувати ключі, зберігати їх і архівувати, відновлювати загублені ключі, заміняти або вилучати з обігу старі й непотрібні ключі.

Найважливішою частиною системи керування ключами є центр розподілу ключів (Key Distribution Center – KDC), функціями якого є генерація, розподіл і передача ключів.

Фахівцями розроблені спеціальні процедури (або протоколи), які дозволяють центру розподілу ключів доставляти користувачам ключі для проведення окремих сеансів зв'язку (сеансові ключі). На жаль, всі протоколи з використанням симетричного шифрування мають ті або інші недоліки.

Розглянемо один з можливих протоколів обміну ключами. Припустимо, при вступі в співтовариство користувачів мережі обміну даними центром розподілу ключів всім новим абонентам видається індивідуальний секретний ключ. От як може виглядати процедура розподілу секретних ключів для проведення сеансу зв'язку між двома абонентами мережі з використанням центра розподілу ключів (для стислості будемо називати його просто Центром):

1. Абонент А звертається в Центр і запитує сеансовий ключ для зв'язку з абонентом Б.

2. У Центрі створюється випадковий сеансовий ключ. Зашифровуються дві копії цього сеансового ключа – одна з використанням секретного ключа абонента А, інша – з використанням секретного ключа абонента Б. Потім обидві зашифровані копії пересилаються із Центра абонентові А.

3. Абонент А розшифровує свою копію сеансового ключа й пересилає другу зашифровану копію абонентові Б.

4. Абонент Б розшифровує свою копію сеансового ключа.

5. Абоненти А и Б використовують отриманий сеансовий ключ для секретного обміну інформацією.

Зазначений протокол досить простий і може бути автоматизований за допомогою, наприклад, програми передачі даних. Однак наведена процедура розподілу сеансових ключів має кілька явних недоліків.

Першим недоліком даної системи є те, що Центр бере участь у всіх обмінах. Збої в роботі Центра порушать роботу всієї системи.

Другим недоліком є те, центр розподілу ключів повинен зберігати в якому-небудь виді секретні ключі всіх абонентів мережі. Якщо зловмисник знайде доступ до секретних ключів користувачів системи ("зламає" систему, підкупить адміністратора й т.д.), то він зможе читати й змінювати всі передані повідомлення.

І, нарешті, залишається проблема первісного розподілу секретних ключів при вступі користувача в мережу. Первісний секретний ключ повинен бути доставлений по абсолютно надійному каналу зв'язку, інакше весь протокол втрачає всякий зміст.

Добре, якщо первісний ключ може бути виданий особисто новому користувачеві, однак у деяких випадках це неможливо, наприклад, при територіальній розподіленості мережі передачі даних.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Модуль генерації ключів ФАКТОР ТС

МГК призначений для створення закритого ключа й файлу запиту в центр, що засвідчує, на одержання сертифіката відкритого ключа. Дані файли необхідні для роботи СКЗІ програм DiPost і DiSign.

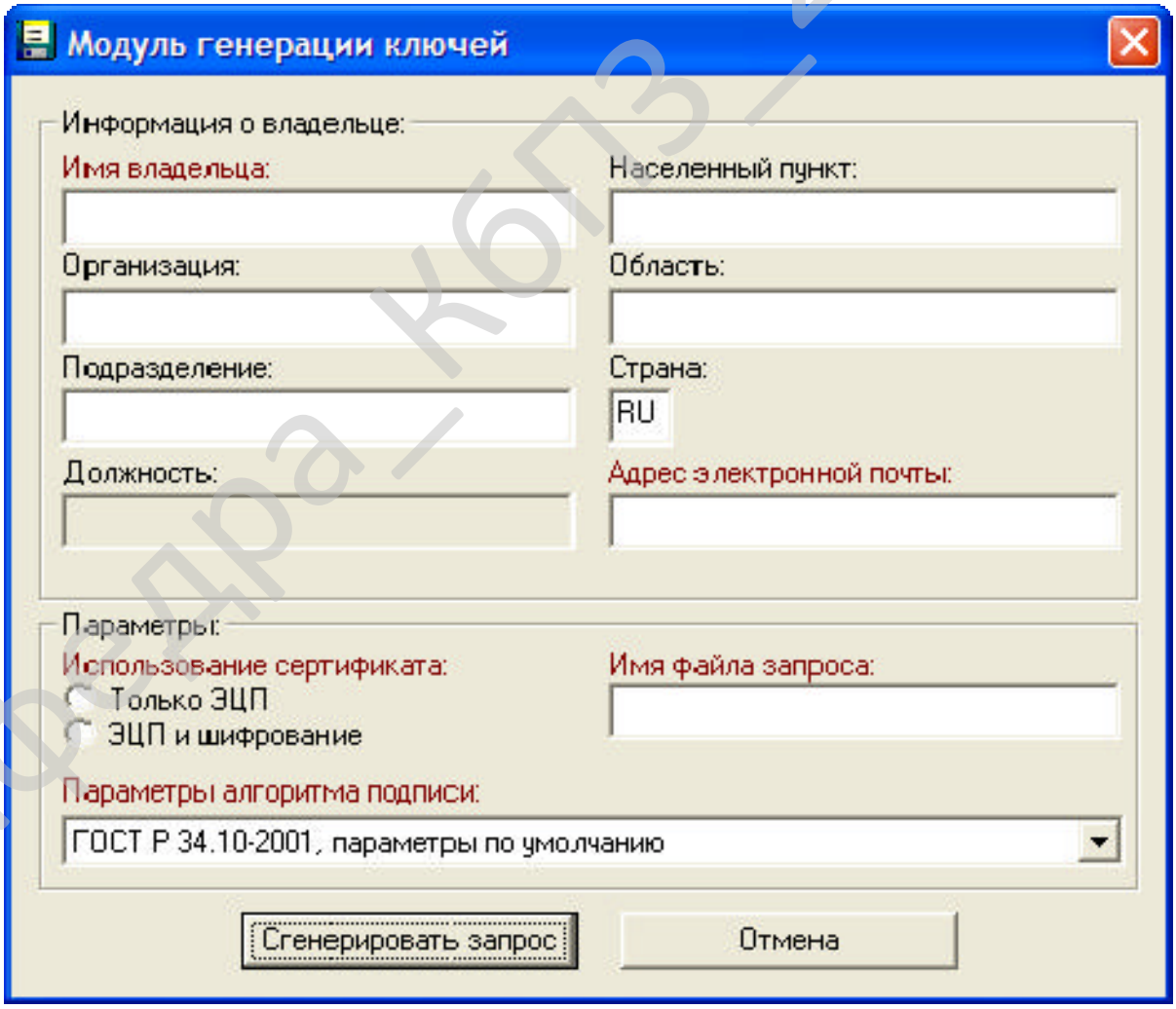

Рисунок 2.1 – Інтерфейс користувача ФАКТОР ТС

Виріб «Автоматизоване робоче місце «Центр генерації ключів», НКБГ.5014306.712 (далі АРМ ГК**)** виконує генерацію ключів для засобів криптографічного захисту інформації **(**СКЗІ**),** використовуваних у технології ДІОНІС.

Зазначені СКЗІ використовують алгоритм шифрування ДСТ 28147:2009 Системи обробки інформації. Захист криптографічна.

Рівень захищеності КС1 гарантує безпечну експлуатацію АРМ ГК в умовах, коли на місцях експлуатації АРМ ГК відсутні внутрішні порушники (допускається наявність тільки зовнішніх порушників).

Рівень захищеності КС2:

– гарантує безпечну експлуатацію АРМ ГК в умовах, коли на місцях експлуатації допускається наявність внутрішнього порушника, що не є користувачем АРМ ГК;

– передбачає експлуатацію АРМ ГК тільки за умови його укомплектування сертифікованим ФСБ апаратним модулем довіреного завантаження (електронним замком);

– передбачає генерацію ключової інформації програмним датчиком випадкових чисел (ПДВЧ), що ініціалізується вихідним матеріалом, надаваним уповноваженою організацією.

Для забезпечення безпечної експлуатації АРМ ГК за рівнем захищеності КС3 потрібно:

– комісійна (спільна) робота відповідальних осіб, допущених до роботи з АРМ ГК;

– зберігання ключового носія з майстер-ключем у сейфі, опечатаному всіма відповідальними особами;

– використання АПМДЗ, що має захист від логічного відключення з BIOS-Setup, що має можливість створення функціонально-замкнутого середовища а також дозволяє виконати вхід в ОС тільки під обліковим записом, що пройшов автентифікацію АПМДЗ.

Для генерації ключів в АРМ ГК крім вихідного матеріалу використовуються випадкові послідовності, вироблювані оператором за допомогою програмно-клавіатурного датчика випадкових чисел (ПКДВЧ).

Генерація ключів і ключових носіїв заданої серії на АРМ ГК виконується в наступному порядку:

1. Генерація ключового носія з майстер-ключем (КНМК) на основі вихідного матеріалу.

2. Генерація ключових носіїв з мережними наборами ключів парного зв'язку (КНПЗ) для вузлів мережі на основі майстра-ключа.

Таким чином, ключовий носій з майстер-ключем (КНМК) виготовляється на АРМ ГК на основі вихідного матеріалу (із ключового носія КНІМ, виготовленого на АРМ ГК). Інформацію, записану на КНМК, АРМ ГК використовує для виготовлення мережних наборів ключів парного зв'язку (КНПЗ).

КНМК виготовляється для кожної серії мережної таблиці тільки один раз і містить всю необхідну інформацію, що дозволяє:

– обчислити ключі парного зв'язку й виготовити ключові носії для всіх вузлів мережі або вибірково для деяких з них;

– у будь-який момент відновити мережний набір і ключовий носій для будь-якого вузла мережі;

– зробити генерацію ключових носіїв для нових вузлів у випадку збільшення розмірності мережі.

На відміну від технологій, що припускають попереднє виготовлення мережної таблиці, у цьому випадку алгоритм роботи АРМ ГК дозволяє уникнути формування й зберігання мережної таблиці на твердому магнітному диску (мережні набори записуються тільки на ключові носії вузлів).

У результаті при виключеному живленні АРМ ГК ні в який момент часу не містить ключової інформації.

Ключові носії КНПЗ повинні бути доставлені на вузли зв'язку обов'язково по захищеному («надійному») каналу зв'язку, наприклад, фельд'єгерською поштою або з використанням телекомунікаційних засобів, але обов'язково із застосуванням незалежного контуру криптографічного захисту.

Програмний засіб криптографічного захисту інформації " Грифон-Б"

Програмний засіб «Грифон-Б» призначено для криптографічного захисту конфіденційної інформації в автоматизованих банківських системах і застосовується для обміну інформацією усередині корпоративної мережі банку, із клієнтами, що працюють по системі " Клієнт-Банк", у системах обслуговування пластикових карт та ін.

ПЗ КЗІ «Грифон-Б» має експертний висновок Департаменту спеціальних телекомунікаційних систем і захисту інформації Служби безпеки України №18/2/ 1-89 від 10 січня 2004 року:

«Криптографічні перетворення, реалізовані в об'єкті експертизи, відповідають вимогам ДСТ 28147:2009, ДСТ 34.310-95 (для текстових документів), ДСТ 34.311-95. Протокол розподілу ключів, реалізований в об'єкті експертизи, відповідає вимогам технічного завдання на ІКР «Розробка програмного засобу криптографічного захисту інформації в автоматизованій банківській системі« шифр " Грифон-Б".

ТОВ СНПФ «АРГУС» здійснює діяльність у сфері криптографічного захисту інформації відповідно до Ліцензії ДСТСЗІ СБУ АА №779028 від 17.08.2004 року.

Функціональним призначенням програмного засобу криптографічного захисту інформації «Грифон-Б» є забезпечення конфіденційності, цілісності й автентичності даних, виконання функцій генерації, сертифікації й розподіли ключів. ПЗ КЗІ «Грифон-Б» призначено для роботи на ІВМ-сумісних персональних комп'ютерах.

До складу програмного засобу входять:

1. Повнофункціональна (включаюча функції генерації загальносистемних параметрів і ключів) програма накладення/перевірки й шифрування електронного цифрового підпису.

2. Користувальницька (не включає функції генерації) програма накладення/перевірки й шифрування електронного цифрового підпису.

3. Тестова версія програми.

4. Користувальницька бібліотека накладення/перевірки підпису у вигляді динамічної поділюваної бібліотеки для SCO UNIX.

5. Програма для генерації ключової інформації ВПС.

Програмний засіб «Грифон-Б» забезпечує реалізацію наступних алгоритмів:

– Криптографічного перетворення відповідно до ДСТ 28147:2009 у режимах простої заміни, гаммування й гаммування зі зворотним зв'язком для областей пам'яті й файлів.

– Формування імітовставки довжиною 32 біт відповідно до ДСТ 28147:2009.

– Гешування відповідно до ДСТ 34.311-95 для областей пам'яті й файлів.

– Генерації секретного ключа електронного цифрового підпису **x** , секретного параметра **k** для реалізації ДСТ 34.310-95, а також генерацію сеансових ключів для реалізації ДСТ 28147:2009.

– Генерації відкритої ключової інформації, обчислення й перевірку електронного цифрового підпису на базі асиметричного криптографічного алгоритму відповідно до ДСТ 34.310-95 для областей пам'яті й файлів.

– Розподілу сеансових ключів відповідно до протоколу обміну ключами на основі алгоритму Діффі-Хеллмана, наведеному в Технічному завданні.

Основні функції програми**:**

– Одержання довідкової інформації.

– Тест гешування, у т.ч. шифрування простою заміною.

– Одержання чисел p, q (512 біт і 1024 біт).

– Тест створення й перевірки ЕЦП.

– Тести швидкодії (шифрування, гешування, генерації чисел, накладення підпису й ін.).

– Просте шифрування рядка або файлу.

– Адресне шифрування рядка або файлу.

– Гешування рядка або файлу.

– Генерація загальносистемних параметрів.

– Генерація макета ключа користувача.

– Генерація ключа користувача.

– Зміна пароля на секретному ключі.

– Підпис рядка або файлу.

– Перевірка підпису.

– Зняття підпису.

– Загальний секретний ключ Z_{AB} по Діффі-Хеллману.

Програмний засіб криптографічного захисту інформації «Грифон-Б» розроблено мовою програмування «С», що дає можливість його перенесення на різні платформи без зміни текстів програм.

Програмний засіб складається з декількох модулів, які компілюються окремо, а потім збираються в статичну бібліотеку для відповідної платформи. Бібліотечні функції викликаються за допомогою керуючої програми з інтерфейсом командного рядка, що може компонуватися у версії з різною функціональністю.

Поставка керуючої програми виробляється в декількох модифікаціях:

– gostfull – повнофункціональна версія (UNIX);

– gostuser – має функції, необхідні для виконання завдань на робочому місці користувача (UNIX, Win32);

– gostargus – генератор ключів адміністратора захисту інформації внутрішньої платіжної системи банку (Win32).

Програма має інтерфейс командного рядка. Керуючі ключі програми визначають тип завдання й чи задають режим її роботи. Можливе виконання декількох завдань одним запуском програми.

Перевірка працездатності програми виконується за допомогою убудованих функцій самоконтролю. Поставляються програми з інтерфейсом командного рядка в декількох модифікаціях: для виконання тестових прикладів, перевірки швидкодії базових алгоритмів, виконання функцій генерації ключів, а також виконання основних операцій, необхідних користувачеві (шифрування, цифрова підпис і ін.).

Програма функціонує в наступних операційних середовищах:

– Windows.

– ОС SCO UNIX.

– ОС Linux Red Hat 6.2.

Для роботи програми також необхідна наявність допоміжної динамічної бібліотеки загального користування ugnr_dll.dll на платформі Win32 або статичної бібліотеки gnr.a на платформах UNIX/Linux.

Програмний засіб «Грифон-Б» не залежить від наявності спеціальних пристроїв зчитування/запису ключів. Передбачено можливість роботи із ключами, розташованими на жорсткому диску, а на платформі Win32 також і з дискети (у тому числі без файлової системи) і із пристроєм “USB Flash”.

Основні функції реалізовані відповідно до міждержавних стандартів:

– ДСТ 28147:2009. Алгоритм криптографічного перетворення.

– ДСТ 34.311-95. Функція гешування.

– ДСТ 34.310-95. Процедура виробітку й перевірки електронного підпису на базі асиметричного криптографічного алгоритму.

Крім того, використана схема розподілу симетричних ключів Діффі-Хеллмана й стандарт X9.17 для генерації сеансових ключів.

Обсяг оброблюваних даних, які підписуються або шифруються, обмежується тільки обсягом доступної оперативної пам'яті (при роботі з областями пам'яті) або вільним місцем на диску (при роботі з файлами).

КриптоПРО OCSP

У процесі керування ключами УЦ має можливість відкликання випущених їм сертифікатів, що необхідно для дострокового припинення їхньої дії, наприклад, у випадку компрометації ключа. Процедура перевірки ЕЦП передбачає крім підтвердження її математичної коректності ще й побудова, і перевірку ланцюжка сертифікатів до довіреного УЦ, а також перевірку статусів сертифікатів у ланцюжку.

Таким чином, перевірка статусів сертифікатів важлива для додатків, що використовують ІОК, оскільки, наприклад, прийняття до обробки підписаного ЕЦП документа, що відповідає сертифікат ключа підпису якого анульований, може бути згодом оскаржене й привести до фінансових втрат.

В ОС Microsoft Windows убудована підтримка технології ІОК. Багато додатків, що працюють під керуванням цих ОС, використовують інтерфейс CryptoAPI для здійснення функцій криптографічного захисту інформації. Функції CryptoAPI використовують, наприклад, що випливають додатки: Internet Explorer, Outlook Express, Outlook, Internet Information Server і ін.

За замовчуванням CryptoAPI здійснює перевірку статусів сертифікатів з використанням СВС (CRL). СВС являє собою список анульованих або припинених сертифікатів, видаваний періодично – наприклад, раз у тиждень. СВС не відбиває інформацію про статуси в реальному часі, а також має ряд інших недоліків, яких позбавлений протокол OCSP – протокол одержання статусу сертифіката в реальному часі.

Лінійка продуктів КриптоПРО OCSP призначена для організації сервера OCSP і для вбудовування функціональності перевірки статусів сертифікатів по протоколі OCSP у різні додатки.

Строге проходження міжнародної рекомендації RFC 2560 "Internet X.509 Public Key Infrastructure Online Certificate Status Protocol – OCSP" забезпечує сумісність реалізації із продуктами інших постачальників.

Лінійка продуктів КриптоПРО OCSP складається з наступних компонентів:

– КриптоПРО OCSP Server.

– КриптоПРО OCSP Client.

– КриптоПРО OCSP SDK.

– КриптоПРО OCSPUTIL.

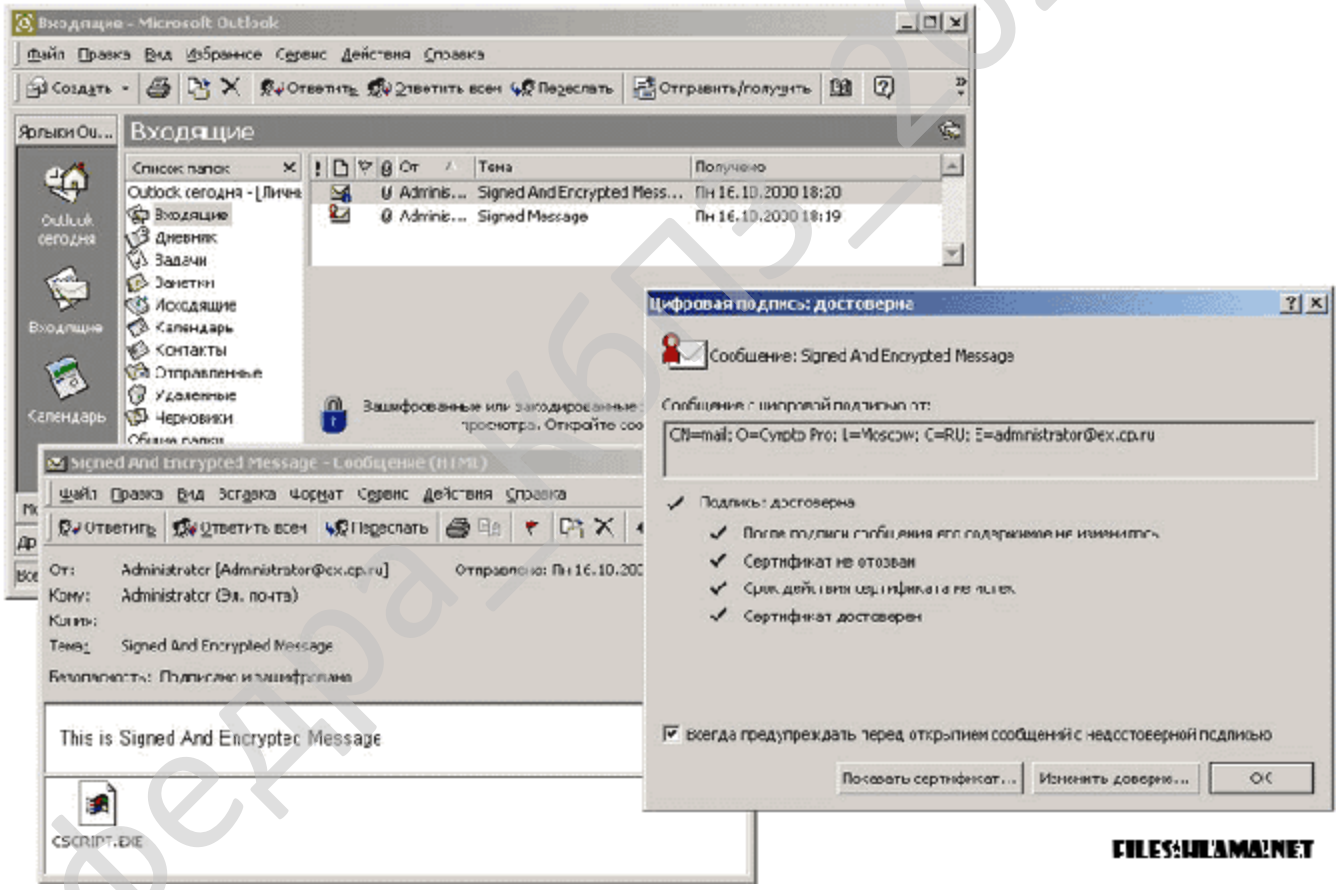

Рисунок 2.2 – Інтерфейс користувача КриптоПРО

КриптоПРО OCSP SDK

Інструментарій розроблювача КриптоПРО OCSP SDK призначений для створення додатків, що використовують протокол OCSP для перевірки статусів сертифікатів, з урахуванням застосування російських криптографічних алгоритмів.

Інструментарій розроблювача КриптоПРО OCSP SDK містить всю необхідну документацію, бібліотеки й вихідні файли для розробки прикладних і серверних додатків, що використовують функції КриптоПРО OCSP Client, і для вбудовування його в настановний пакет.

Інструментарій розроблювача КриптоПРО OCSP SDK функціонує в наступних операційних системах:

– Microsoft Windows.

– ОС сімейства Linux. задовольняючих LSB 3.1 і вище.

– FreeBSD 7.x і вище.

– AIX 5.3 і 6.x.

– Solaris 10 і вище.

Інструментарій розроблювача КриптоПРО OCSP SDK:

– Дозволяє розробити додатки, що використовують функції КриптоПРО OCSP Client, і вмонтувати його в настановний пакет.

– Включає низькорівневий інтерфейс для роботи з ASN.1-структурами протоколу OCSP.

– Установлюється за допомогою Windows Installer.

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland

і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

– Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++,

наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL, включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

- Велика кількість поліпшень STL від Dinkumware.
- Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.
- Поліпшена підтримка Cmake.
- Велика кількість виправлень для підвищення стабільності і якості.

– Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.

– Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

– Додана підтримка Metal Driver GPU для macOS і iOS.

– Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.

– Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

– Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.

– Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.

– Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації.

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

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

б) вибрати та обґрунтувати методику побудови системи кібербезпеки 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 кібербезпеки 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

Комплекс програмних засобів реалізації інфраструктури генерації й розподілу відкритих ключів призначений для забезпечення захисту інформації, оброблюваної в системах обміну повідомленнями (системах електронного документообігу, електронної пошти, автоматизованих банківських системах і т.п.), від погроз порушення її цілісності й конфіденційності, забезпечення автентифікації відправників повідомлень (авторів документів) шляхом використання механізмів криптографічного захисту інформації (електронний цифровий підпис – ЕЦП, шифрування, автентифікація), а також реалізації необхідних для цього функцій генерації ключів, виробітки й керування сертифікатами відкритих ключів (ВідкрКл).

Із цією метою компонента (програмні засоби) комплексу реалізують необхідний перелік функцій генерації й розподілу ключів і сертифікатів, виконання криптографічних перетворень, а також підтримують необхідний інтерфейс для взаємодії із прикладними програмними системами (ППС).

До складу комплексу входять такі програмні засоби (ПЗ):

– ПЗ автоматизованого робочого місця (АРМ) головного центра сертифікації ключів (ГЦСК).

– ПЗ АРМ регіонального центра сертифікації ключів (РЦСК).

– ПЗ АРМ центра реєстрації (ЦР).

– ПЗ генерації ключів і обслуговування сертифікатів користувачів (ПЗ ГКОСП).

Програмні засоби АРМ ГЦСК призначені для керування в ГЦСК процесом виробітку особистого ключа (ОсобКл) і ВідкрКл сертифікації, перевірки отриманих заявок на виробіток сертифікатів і виробітку сертифікатів ВідкрКл,

виконання операцій по обслуговуванню центральної бази даних сертифікатів ВідкрКл користувачів прикладної програмної системи й підтримки із цією метою взаємодії із центральним органом, що засвідчує (ЦЗО) або центром, що засвідчує (ЗЦ), ЦР, РЦСК і користувачами. Програмні засоби АРМ ГЦСК функціонують на робочій станції (РС) локальної обчислювальної мережі (ЛОМ) ГЦСК як окремий додаток.

Програмні засоби АРМ РЦСК призначені для керування в РЦСК процесом виробітку ОсобКл і ВідкрКл сертифікації, перевірки отриманих заявок на виробіток сертифікатів і виробітку сертифікатів ВідкрКл, виконання операцій по обслуговуванню регіональної бази даних (РБД) сертифікатів ВідкрКл користувачів ППС і підтримки із цією метою взаємодії із ГЦСК, ЦР і користувачами. Програмні засоби АРМ РЦСК функціонують на РС ЛОМ РЦСК як окремий додаток.

Програмні засоби АРМ ЦР призначені для керування в ЦР процесом реєстрації заявок на виробіток сертифікатів ВідкрКл користувачів, виробітку ОсобКл і ВідкрКл користувачів і посадових осіб ГЦСК, РЦСК, ЦР, реєстрації заявок, формування запитів і одержання сертифікатів ВідкрКл користувачів від ГЦСК або РЦСК, реєстрації заявок і формування запитів на блокування/ скасування/ поновлення сертифікатів ВідкрКл користувачів, одержання й видачі користувачам сертифікатів і підтримки із цією метою взаємодії із ГЦСК або РЦСК. Програмні засоби АРМ ЦР функціонують на РС ЛОМ ГЦСК/ РЦСК і (у випадку наявності віддалених ЦР) на РС ЛОМ ЦР як окремий додаток.

Програмні засоби ГКОСП призначені для керування безпосередньо на робочих місцях користувачів ППС процесом виробітку ОсобКл і ВідкрКл, підготовки заявок, формування запитів і одержання сертифікатів ВідкрКл користувачів від ГЦСК або РЦСК, підготовки заявок і формування запитів на блокування/ скасування/ поновлення сертифікатів ВідкрКл користувачів, одержання й розміщення в робочій базі даних (БД) відповідних сертифікатів, здійснення операцій по обслуговуванню робочої БД сертифікатів ВідкрКл

користувачів і підтримки із цією метою взаємодії із ГЦСК або РЦСК. ПЗ ГКОСП функціонують на РС користувачів розподіленої ППС як окремий додаток.

Взаємодія між АРМ ГЦСК і АРМ РЦСК здійснюється шляхом обміну файлами, які містять відповідні інформаційні об'єкти (запити, сертифікати, повідомлення), з використанням для передачі файлів засобів електронної пошти або будь-яких інших, що забезпечують можливість обміну файлами, а також (при виконанні реплікації БД) шляхом віддаленого доступу до серверів СУБД РЦСК із боку АРМ ГЦСК.

Взаємодія між АРМ ГЦСК/ РЦСК і АРМ ЦР здійснюється шляхом обміну файлами, які містять відповідні інформаційні об'єкти (запити, сертифікати, повідомлення), з використанням для передачі файлів каналів ЛОМ (для взаємодії з АРМ ЦР, що функціонує в складі ЛОМ ГЦСК/ РЦСК), засобів електронної пошти або будь-яких інших, що забезпечують можливість обміну файлами (для взаємодії з АРМ ЦР, що функціонує в складі віддалених ЦР).

Взаємодія між АРМ ГЦСК/ РЦСК і ПЗ ГКОСП здійснюється шляхом обміну файлами, які містять відповідні інформаційні об'єкти (запити, сертифікати, повідомлення), з використанням для передачі файлів засобів електронної пошти або будь-яких інших, що забезпечують можливість обміну файлами.

Доступ користувачів до сертифікатів ВідкрКл, збереженим у БД публікації сертифікатів (LDAP-каталозі), здійснюється за протоколом LDAPv3 з використанням довільної програми LDAP-клієнта.

Доступ користувачів до сертифікатів, збереженим у каталозі публікації сертифікатів, а також до списків відкликаних сертифікатів (СВС), збереженим у каталозі публікації СВС, здійснюється за протоколом HTTP з використанням довільної програми HTTP-клієнта.

Як носії особистих ключів сертифікації й особистих ключів користувачів можуть використовуватися довільні змінні файлові носії (дискети, пристрою flash drive і т.п.) і пристрою eToken Pro. Реалізований інтерфейс доступу до пристроїв

зберігання ОсобКл забезпечує можливість використання (після розробки відповідного модуля взаємодії) будь-яких інших пристроїв.

Архітектура системи центрів сертифікації

Як ми вже відзначали раніше, необхідною умовою створення глобальної системи автентифікації, заснованої на використанні асиметричної криптографії, є наявність ієрархічної однокореневої системи центрів сертифікації. Основні функції системи ЦС – генерація й розподіл сертифікатів відкритих ключів, відновлення сертифікатів, а також генерація й розподіл списків відкликаних ключів (Certificate Revocation Lists, або скорочено CRL).

У протоколі SET система ЦС має 4-рівневу архітектуру базовану на використанні протоколу X.509.

На верхньому рівні розташовується Корнєвий ЦС (Root Certificate Authority, або скорочено RCA). Він відповідає за генерацію сертифікатів для ЦС наступного нижчележачого рівня Центрів сертифікації міжнародних платіжних систем (Brand Certificate Authority, або скорочено BCA), генерацію сертифікатів для власних відкритих ключів, а також генерацію й розподіл CRL для можливо скомпрометованих ключів ЦС рівня BCA. Оператором RCA є компанія SETCo, спеціально створена для розвитку й поширення стандарту SET.

На другому рівні ієрархії системи ЦС перебувають ЦС платіжних систем. У цей час такі ЦС створені в платіжних системах VISA, Europay/MasterCard, American Express і інших. ЦС рівня BCA відповідає за генерацію сертифікатів для ЦС наступних рівнів – GCA, CCA, MCA, PCA, а також за генерацію, підтримку й поширення CRL для сертифікатів, раніше підписаних даним BCA. Оператором BCA є відповідна платіжна система.

На третьому рівні системи ЦС SET розташовується Геополітичний ЦС (Geo-Political Certificate Authority, або скорочено GCA). Наявність ЦС рівня GCA дозволяє платіжній системі проводити більш гнучку політику генерації й розподілу сертифікатів ключів для ЦС рівня CCA, MCA, PCA в окремих геополітичних зонах земної кулі, а також підвищувати ефективність процедур

генерації, підтримки й поширення CRL по сертифікатах, емітованим GCA. Оператор ЦС рівня GCA визначається правилами відповідної платіжної системи. Наприклад, за правилами систем VISA і MasterCard оператором GCA може бути або сама платіжна система, або Group Member – банк, що має статус Групового учасника платіжної системи.

Нарешті, на четвертому, нижньому рівні системи ЦС SET розташовуються три так званих кінцевих (End-Entity) типи ЦС:

– ЦС для власників платіжних карт (Cardholder Certificate Authority, або скорочено ССА);

– ЦС для ТП (Merchant Certificate Authority, або скорочено МСА);

– ЦС для платіжних шлюзів (Payment Gateway Certificate Authority, або скорочено РСА).

Центри сертифікації рівня End-Entity відповідають за генерацію сертифікатів для основних учасників транзакції ЕК – для власника карти, ТП і платіжного шлюзу. У цьому змісті всі інші ЦС відіграють допоміжну роль, забезпечуючи єдину загальну інфраструктуру центрів довіри, що дозволяє будь-яким двом безпосереднім учасникам транзакції ЕК надійно автентифікувати один одного. Коротко зупинимося на основних функціях ЦС рівня End-Entity.

ЦС рівня ССА відповідає за генерацію й доставку сертифікатів відкритих ключів власників карт. Запити на одержання сертифікатів надходять у ССА від власників карт або через Web-сторінки, або по електронній пошті. Для генерації сертифіката власника карти ССА повинен підтримувати спеціальну процедуру ідентифікації клієнта, певну емітентом карти. ССА також відповідає за поширення серед власників карт списків CRL, згенерованих RCA, BP A, GCA, PCA. Оператором ССА можуть бути банк-емітент карток, для яких випускаються сертифікати, платіжна система або третя сторона, обумовлена правилами конкретної платіжної системи.

ЦС рівня МСА відповідає за генерацію й доставку сертифікатів відкритих ключів ТП. Запити на одержання сертифікатів надходять у МСА від ТП або через

Web-сторінки, або по електронній пошті. Для генерації сертифіката ТП МСА повинен підтримувати спеціальну процедуру ідентифікації ТП, визначену обслуговуючим банком даного ТП. МСА також відповідає за поширення на адресу ТП списків CRL, згенерованих RCA, BCA, GCA, PCA. Оператором МСА можуть бути обслуговуючий банк ТП, платіжна система або третя сторона, обумовлена правилами конкретної платіжної системи.

ЦС рівня РСА відповідає за генерацію й доставку сертифікатів відкритих ключів платіжним шлюзам. РСА також відповідає за генерацію й поширення списку CRL, що містить раніше емітовані даним РСА сертифікати відкритих ключів, для яких відповідні їм закриті ключі виявилися скомпрометованими на момент розсилання CRL.

РСА відповідає за поширення на адресу платіжних шлюзів аркушів CRL, згенерованих RCA, BCA, GCA, PCA. Оператором РСА можуть бути обслуговуючий банк, платіжна система або третя сторона, обумовлена правилами розглянутої платіжної системи.

У протоколі SET використовуються чотири типи пар асиметричних ключів, що відрізняються друг від друга по своєму призначенню:

– ключ для підпису (Digital Signature Key, використовується для ідентифікації власника ключа);

– ключ для шифрування даних (Key Encipherment/Data Encipherment Key, або інакше Key-Exchange Key, ключ, використовуваний для шифрування даних у процесі проведення транзакції ЕК);

– ключ для підписування сертифікатів (Certificate Signature Key);

– ключ для підписування списків відкликаних сертифікатів CRL(CRL Signature Key).

Власникові карти досить мати тільки один ключ типу Digital Signature Key, у той час як РСА для виконання своїх функцій повинен мати ключі всіх чотирьох типів. Далі на прикладах процесів сертифікації власника карти в ССА й

проведення операції покупки буде продемонстроване використання різних типів асиметричних ключів.

Розмаїтість типів ключів, крім того, що підвищує безпеку системи ЕК у цілому, дає більше гнучкості при проектуванні платіжної системи. Це досягається за рахунок того, що для реалізації різних функцій з'являється можливість використовувати ключі різної довжини, що підвищує продуктивність системи в цілому. Наприклад, ключ ТП Key Exchange Key може мати більше коротку довжину в порівнянні із ключем ТП Digital Signature Key – для того, щоб при необхідному рівні криптостійкості зменшити працезатрати на виконання криптографічних операцій на стороні власника карти.

Розмір асиметричних ключів, використовуваних у протоколі SET, не фіксований і може згодом мінятися.

Формат сертифіката відкритого ключа в протоколі SET задовольняє стандарту X.509 v.3. Сертифікат містить наступні дані:

– версію протоколу X.509 (завжди встановлюється значення, рівне 3);

– Serial Number – серійний номер сертифіката – унікальний цілочислений номер сертифіката, що привласнюється ЦС, який видав сертифікат;

– Algorithm Identifier – ідентифікатор алгоритму ЕЦП, використовуваного ЦС для підписування сертифіката;

– Issuer Name – ім'я ЦС, що генерує сертифікат;

– строк початку дії сертифіката;

– строк закінчення дії сертифіката;

– Subject Name – ім'я власника сертифіката;

– ідентифікатор алгоритму, у якому буде використовуватися сертифікуємий ключ;

– значення сертифікуємого відкритого ключа;

– розширення (наприклад, інформація про ключ ЦС – емітента даного сертифіката; рівень власника сертифіката в протоколі SET, тип сертифіката (наприклад, сертифікат власника карти, сертифікат ТП і т.п.) і інше);

– цифровий підпис сертифіката, зроблену з використанням Certificate Signing Key ЦС.

Дійсність SET-сертифікатів засвідчує за допомогою ієрархічного ланцюга перевірок.

Будь-який ЦС, що видав сертифікат наступний за ним в ієрархії ланці, у свою чергу, повинен мати дійсний сертифікат від вищестоящої організації. Посвідчення відбувається шляхом порівняння на предмет рівності вмісту деяких полів сертифіката нижнього рівня й сертифіката більше високого рівня. Рівняються наступні поля:

– поля Issuer Name у сертифікаті нижнього рівня й Subject Name у сертифікаті більше високого рівня;

– поля Certlssuer і CertSerialNumber з X.509 Extensions сертифіката нижнього рівня відповідно з полями Issuer;

– Name і Serial Number у сертифікаті більше високого рівня.

При позитивному результаті порівняння в сертифікаті нижнього рівня перевіряється:

– термін дії сертифіката;

– термін дії ключа, зазначеного в сертифікаті;

– рівень власника сертифіката в ієрархії системи ЦС;

– відповідність типу сертифіката його вмісту;

– використання по призначенню ключа, зазначеного в сертифікаті, і деякі інші поля.

У сертифікаті більше високого рівня перевіряється:

– термін дії сертифіката;

– термін дії ключа, зазначеного в сертифікаті;

– рівень власника сертифіката в ієрархії системи ЦС;

– відповідність типу сертифіката його вмісту;

– використання по призначенню ключа, зазначеного в сертифікаті, і деякі інші поля.

3.2 Розробка структурної схеми

На рисунку 3.1 зображена структурна схема розробленої системи. Основними компонентами розробленого, у ході виконання бакалаврського проектування, комплексу є:

– Центр генерації й сертифікації ключів.

– Центр розподілу ключів.

– Шлюз кодування (Кодуючий модуль).

– Точка реєстрації мобільних клієнтів.

– Мобільний клієнт.

Центр генерації й сертифікації ключів

Центр генерації й сертифікації ключів служить для генерації пар "секретний ключ – відкритий ключ", а також є репозитарієм всіх відомих системі ключів.

Центр генерації ключів установлюється на комп'ютер, що не має мережних з'єднань. Він призначений для виготовлення секретних і відкритих ключів, підготовки ключових дискет, зберігання еталонних копій ключів, а також підпису сертифікатів ключем адміністратора.

Центр розподілу ключів

У завдання Центра розподілу ключів входить роздача ключів і керування контуром безпеки, а також виконання наступних функцій:

– Одержання зі змінного носія відкритих ключів Шлюзів.

– Видача будь-якому Шлюзу сертифікатів відкритих ключів будь-яких інших Шлюзів і інформації про відповідні сегменти мережі.

– Розсилання Шлюзам повідомлень про зміни структури закритої мережі.

– Зберігання інформації про структуру мережі.

Центр реалізований у вигляді програмного комплексу, що виконує функції зберігання й видачі відкритих ключів кодування по мережному запиті від Шлюзів кодування. Центр розподілу ключів може бути встановлений або на окремому (виділеному) комп'ютері, або разом з одним зі Шлюзів кодування.

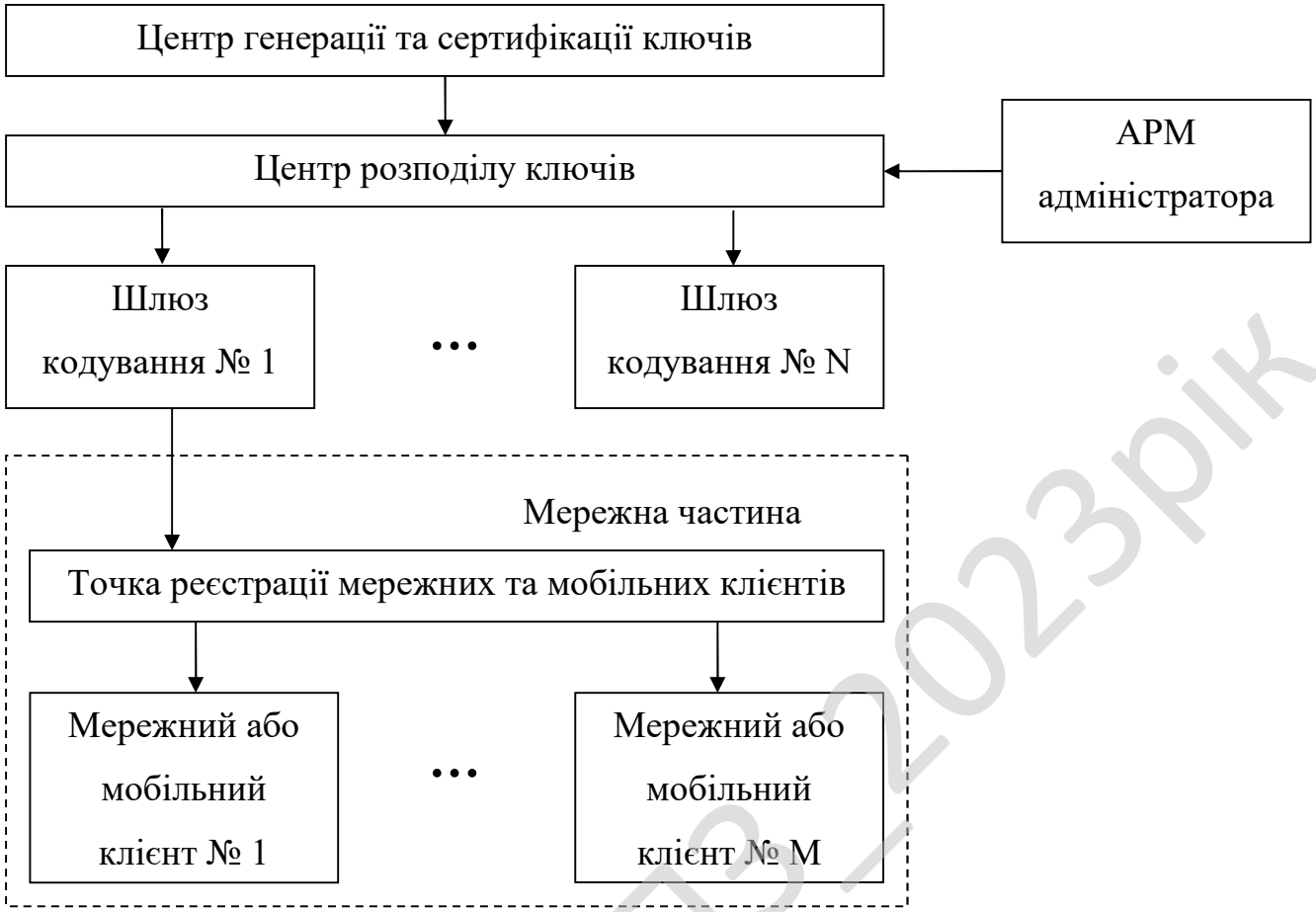

Рисунок 3.1 – Структурна схема системи

Шлюз кодування

Шлюз є основним модулем комплексу, що забезпечує маршрутизацію, фільтрацію й кодування пакетів. Кожний Шлюз призначений для захисту певної групи локальних мереж. На комп'ютері-шлюзі встановлюється ядерний модуль із функціями кодування й декодування й запускається програма автентифікації. Шлюз виконує наступні функції:

– Фільтрація трафіку (розподіл на кодуємий і некодуємий потоки).

– Кодування трафіку (кодуємий потік).

– Автентифікація з іншими Шлюзами.

– Виробіток і виконання процедури зміни сеансових ключів.

– Реєстрація подій у Центрі моніторингу.

– Забезпечення власного захисту.

Точка реєстрації мобільних клієнтів і Мобільний клієнт

Доступ до корпоративних даних, що захищаються, для мобільних абонентів, не підключених до локальних мереж, що захищаються, забезпечується за допомогою таких засобів комплексу , як Точка реєстрації мобільних клієнтів і програмне забезпечення Мобільний клієнт.

Точка реєстрації мобільних клієнтів являє собою спеціальний шлюз кодування для підключення довільної кількості мобільних клієнтів. Мобільний клієнт являє собою програмний модуль, що працює під керуванням ОС Windows і використовуючий апаратні ключі eToken для автентифікації абонента в VPN.

Центр моніторингу

Центр моніторингу являє собою мережне автоматизоване робоче місце із установленим на ньому набором програм, що здійснюють збір і аналіз протоколів, що надходять від всіх модулів комплексу.

Програма контролю цілісності

Комплекс містить у собі засоби формування й перевірки контрольних сум файлів. Ці засоби оформлені у вигляді Програми контролю цілісності. Ця Програма призначена для виявлення змін, додавань і видалень файлів і повідомлення системного адміністратора про ці події.

Автоматизоване робоче місце адміністратора

Настроювання й адміністрування компонентів комплексу здійснюється централізовано з автоматизованого робочого місця (АРМ) адміністратора безпеки за допомогою графічного інтерфейсу або командного рядка. Віддалене керування здійснюється по захищеному каналі. АРМ забезпечує автентифікацію адміністраторів і розмежування доступу до функцій адміністрування.

Можливості:

– Кодування міжмережних потоків. Функції кодування міжмережних інформаційних потоків у відкритих мережах передачі даних виконуються шляхом організації віртуальних захищених мереж (Virtual Private Networks, VPN). Кожна мережа в складі VPN захищена своїм модулем, що кодує, що встановлюється в

точці з'єднання цієї мережі із зовнішніми мережами. Інформація, що захищається, кодується на передавальному модулі й декодується на приймаючому, тобто передається у відкритому виді в межах локальних мереж і в кодованому – за їхніми межами. Кодований трафік передається за протоколом IPsec.

– Створення контуру безпеки. Комплекс кодування міжмережних потоків (ККМП) дозволяє сформувати контур безпеки, що поєднує IP-адреси всіх абонентів, що мають доступ у віртуальну захищену мережу. Абонентами VPN можуть бути цілі мережі, підмережі й окремі робочі станції. Крім того модуль, що кодує, може бути встановлений на окрему робочу станцію.

– Вибіркове кодування трафіку. Для поділу трафіку на кодуємий і некодуємий потоки формується контур безпеки. Модуль, що кодує, комплексу, виділяє пакети, що підлягають кодуванню, на підставі IP-адрес відправника пакета й одержувача пакета, а також перевірки інтерфейсу, через який проходить пакет.

– Керування ключовою системою. У комплексі реалізована асиметрична ключова система, коли потенційні учасники обміну даними використовують пари довгострокових ключів кодування (пари становлять секретний і відкритий ключі). Кодування здійснюється на основі сеансових ключів, автоматично згенерованих за допомогою довгострокових ключів і що мають обмежений час життя. Комплекс здійснює всі необхідні дії по керуванню ключами: генерацію й розподіл довгострокових ключів, виробіток сеансових ключів абонентів, сертифікацію відкритих ключів у довіреному центрі, планову й позаштатну зміну ключів кодування.

– Реєстрація подій, моніторинг і аналіз міжмережних потоків. Комплекс здійснює збір і зберігання статистичної й службової інформації про всі штатні й позаштатні події, що виникають при автентифікації вузлів, передачі кодованої інформації, обмеженні доступу абонентів ЛОМ. Засоби моніторингу проводять

збір протоколів реєстрації від всіх модулів комплексу по кодованому каналі й виконують аналіз цієї інформації.

– Захист з'єднань із мобільними клієнтами. До складу віртуальної захищеної мережі можуть входити мобільні клієнти – віддалені комп'ютери, що підключаються по виділеним або комутуємим каналам зв'язку. Носієм ключової інформації для них є електронний ключ eToken.

3.3 Розробка функціональної схеми

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

Зі схеми ми бачимо, що розроблений програмний комплекс складається з чотирьох функціональних блоків:

– Блок алгоритмів генерації ключів.

– Блок алгоритмів розподілу ключів.

– Блок алгоритмів шифрування, для яких генеруються та розподіляються ключі.

– Блок алгоритмів генерації псевдовипадкових чисел (ГПВЧ).

До алгоритмів генерації ключів відносяться наступні:

– A8.

– RC2.

– GOST P 34.10-94.

До алгоритмів розподілу ключів відносяться наступні:

– Алгоритм Діффі-Хеллмана.

– Алгоритм Ель-Гамаля.

– Архітектура PKI (інфраструктура відкритих ключів).

У блоці алгоритмів шифрування, для яких генеруються та розподіляються ключі, реалізовані наступні криптоалгоритми:

– Gost.

– Cast128.
– Cast256.
– Blowfish.
– IDEA.
– Mars.
– Misty 1.
– RC2.
– RC4.
– RC5.
– RC6.
– FROG.
– Rijndael.
– SAFER.
– SAFER-K40.
– SAFER-SK40.
– SAFER-K64.
– SAFER-SK64.
– SAFER-K128.
– SAFER-SK128.
– TEA.
– TEAN.
– Skipjack.
– SCOP.
– Q128.
– 3Way.
– Twofish.
– Shark.
– Square.

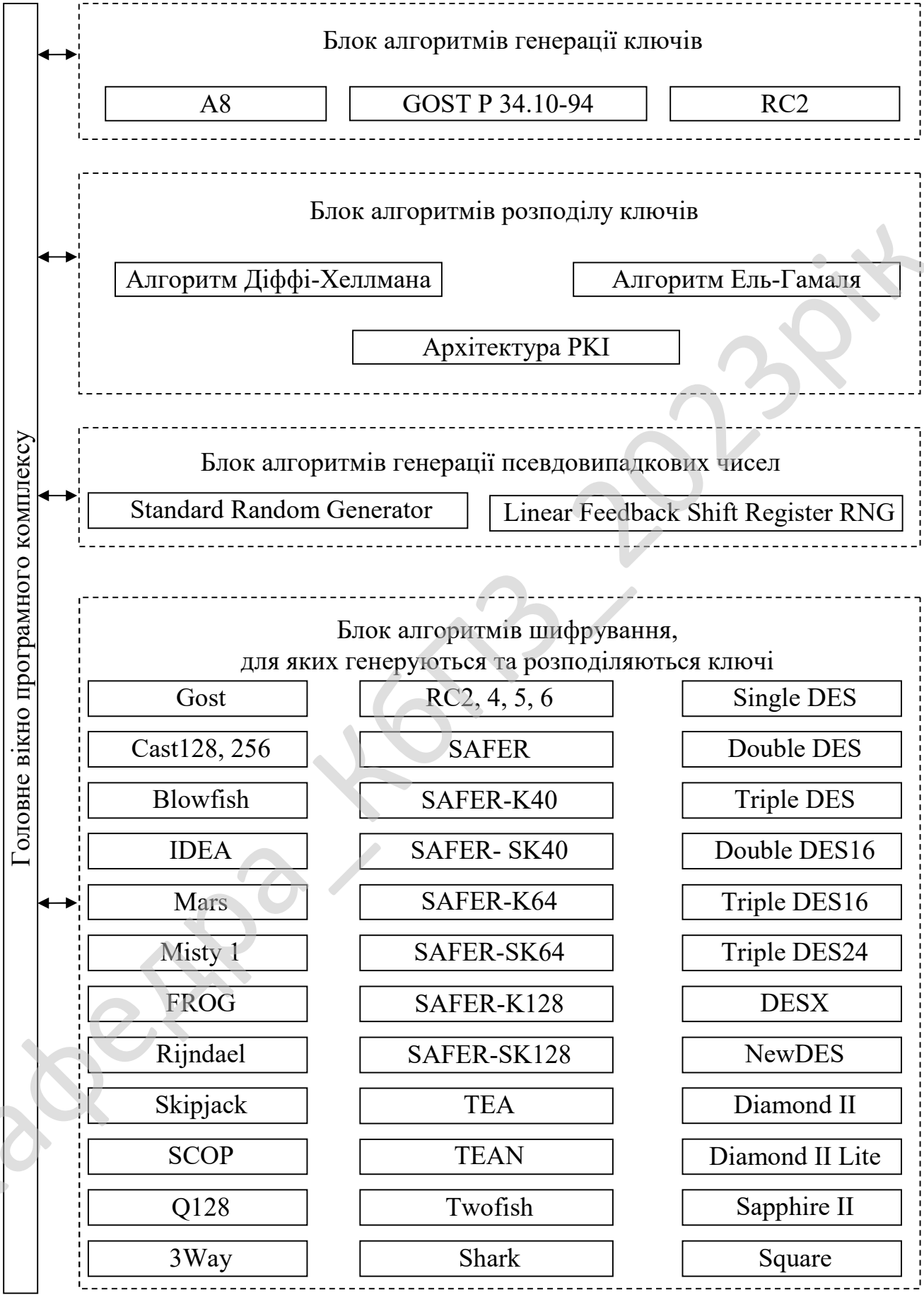

Рисунок 3.2 – Функціональна схема системи

– Single DES.

– Double DES.

– Triple DES.

– Double DES16.

– Triple DES16.

– TripleDES24.

– DESX.

– NewDES.

– Diamond II.

– Diamond II Lite.

– Sapphire II.

У блоці алгоритмів генерації псевдовипадкових чисел реалізовані наступні алгоритми:

– Standard Random Generator.

– Linear Feedback Shift Register RNG з змінним періодом з 2^{64}-1 до 2^{2032}-1.

Розглянемо більш детально деякі з алгоритмів.

DES

DES є класичною мережею Фейштеля із двома гілками (рисунок 3.4). Дані шифруються 64-бітними блоками, використовуючи 56-бітний ключ. Процес шифрування складається із чотирьох етапів. На першому з них виконується початкова перестановка (IP) 64-бітного вихідного тексту (забілювання), під час якої біти переставляються у відповідності зі стандартною таблицею. Наступний етап складається з 16 раундів однієї й тієї ж функції, що використовує операції зсуву й підстановки. На третьому етапі ліва й права половини виходу останньої (16-ої) ітерації міняються місцями. Нарешті, на четвертому етапі виконується перестановка IP^{-1} результату, отриманого на третьому етапі. Перестановка IP^{-1} інверсна початковій перестановці.

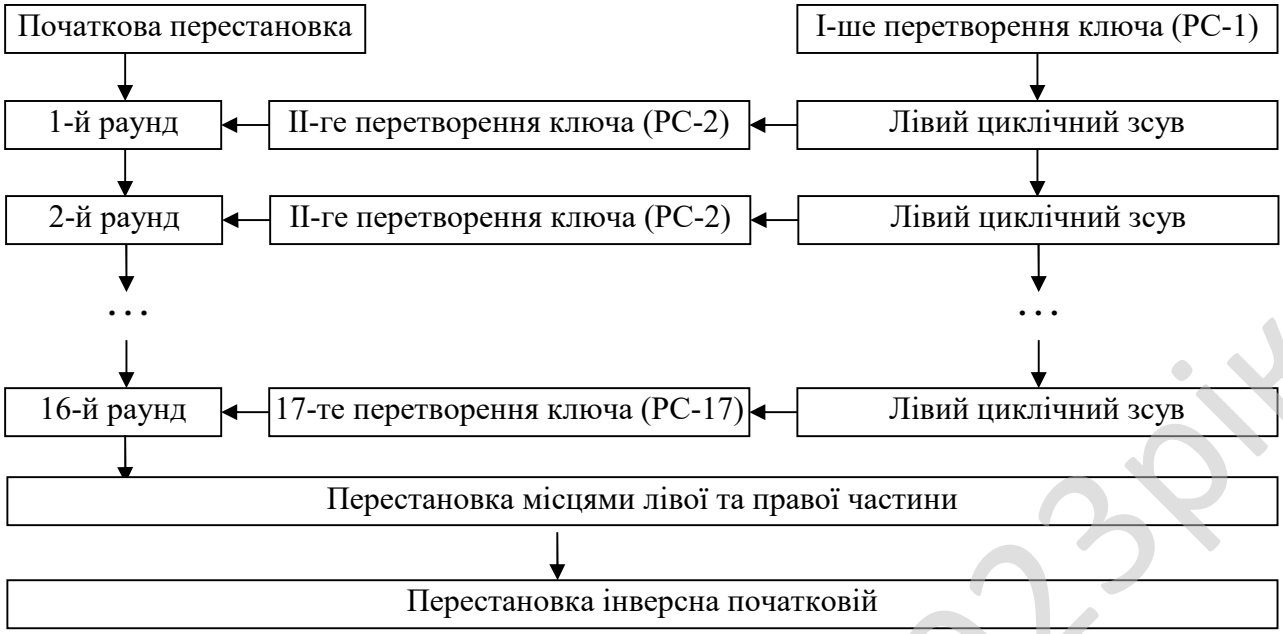

Рисунок 3.3 – Загальна схема DES

У правій частині рисунку 3.3 показаний спосіб, яким використовується 56-бітний ключ. Спочатку ключ подається на вхід функції перестановки. Потім для кожного з 16 раундів підключ K_i є комбінацією лівого циклічного зсуву й перестановки. Функція перестановки та сама для кожного раунду, але підключи K_i для кожного раунду виходять різні внаслідок повторюваного зсуву бітів ключа.

На найпростішому рівні алгоритм не представляє нічого більшого, ніж комбінація двох основних методів шифрування: зсуву й дифузії. Фундаментальним будівельним блоком DES є застосування до тексту одиничної комбінації цих методів (підстановка, а за нею – перестановка), що залежить від ключа. Однакова комбінація методів застосовується до відкритого тексту 16 разів.

Розглянемо більш докладно алгоритм шифрування. DES працює з 64-бітовим блоком відкритого тексту. Після первісної перестановки блок розбивається на праву й ліву половини довжиною по 32 біта. Потім виконується 16 етапів однакових дій, що називаються функцією f, у яких дані поєднуються із ключем. Після шістнадцятого етапу права й ліва половини поєднуються й

алгоритм завершується заключною перестановкою (зворотною відносно первісної).

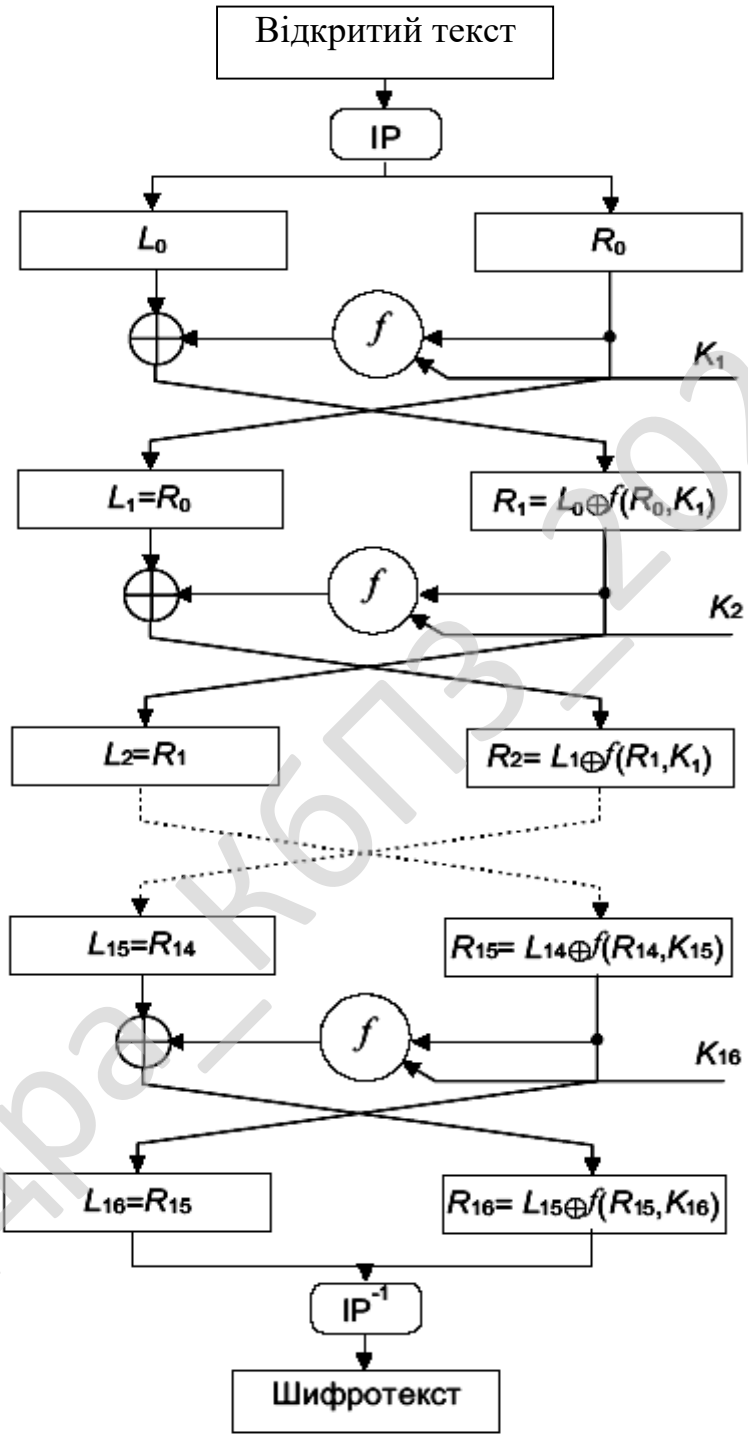

Рисунок 3.4 – Алгоритм DES

На кожному етапі біти ключа зсуваються, і потім з 56 бітів ключа вибираються 48 бітів. Права половина даних збільшується до 48 бітів за допомогою перестановки з розширенням, поєднується за допомогою XOR з 48 бітами зміщеного й переставленого ключа, проходить через 8 S-блоків, утворюючи 32 нових біта, і переставляється знову. Ці чотири операції й виконуються функцією f. Потім результат функції f поєднується з лівою половиною за допомогою іншого XOR. У підсумку цих дій з'являється нова права половина, а стара права половина стає новою лівою. Ці дії повторюються 16 разів, утворюючи 16 етапів DES.

Якщо B_i – це результат і-ої ітерації. L_i, і R_i, – ліва й права половини B_i, K_i – 48-бітовий ключ для етапу i, а f – це функція, що виконує всі підстановки, перестановки й XOR з ключем, то етап можна представити як:

$$L_i = R_{i-1},\ R_i = L_{i-1} \oplus f(R_{i-1}, K_i).$$

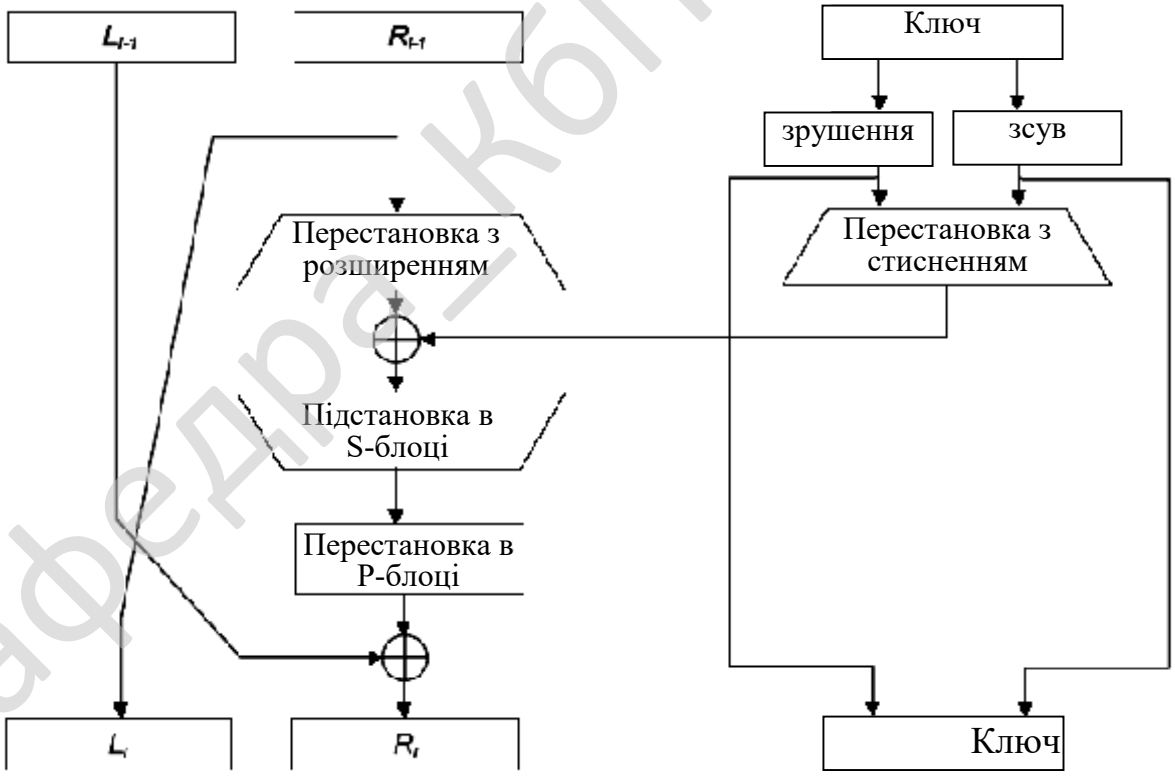

Рисунок 3.5 – Один етап DES.

Таблиця 3.1 – Початкова перестановка шифру DES

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Початкова перестановка й відповідна заключна перестановка не впливають на безпеку DES. Так як програмна реалізація цієї багатобітної перестановки нелегка, у багатьох програмних реалізаціях DES початкова й заключна перестановки не використовуються. Хоча такий новий алгоритм не менш безпечний, ніж DES, він не відповідає стандарту DES і, тому, не може називатися DES.

Перетворення ключа: 64-бітовий ключ DES зменшується до 56-бітового ключа відкиданням кожного восьмого біта. Ці біти використовуються тільки для контролю парності, дозволяючи перевіряти правильність ключа. Після добування 56-бітового ключа для кожного з 16 етапів DES генерується новий 48-бітовий підключ і ці підключі, K_i, визначаються в такий спосіб:

Таблиця 3.2 – Перетворення ключа DES

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

По перше, 56-бітовий ключ ділиться на дві 28-бітових половинки. Потім, половинки циклічно зсуваються ліворуч на один або два біти залежно від етапу.

Таблиця 3.3 – Число бітів зсуву залежно від етапу DES

Етап	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Число	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Після зсуву вибирається 48 з 56 бітів. Так як при цьому не тільки вибирається підмножина бітів, але й змінюється їхній порядок, ця операція називається перестановка зі стиском. Її результатом є набір з 48 бітів. Наприклад, біт зсунуного ключа в позиції 33 переміщається в позицію 35 результату, а 18-й біт зсунуного ключа відкидається.

Таблиця 3.4 – Перестановка зі стиском

14	17	11	24	1	5	3	28	15	6	21	10
23	19	11	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

За допомогою зсуву для кожного підключа використовується відмінна підмножина бітів ключа. Кожний біт використовується приблизно в 14 з 16 підключів, хоча не всі біти використовуються в точності однакове число разів.

Перестановка з розширенням: ця операція розширює праву половину даних, R_i, від 32 до 48 бітів. Так як при цьому не просто повторюються певні біти, але й змінюється їхній порядок, ця операція називається перестановкою з розширенням, У неї дві задачі: привести розмір правої половини у відповідність із ключем для операції XOR і одержати більш довгий результат, який можна буде стиснути в ході операції підстановки. Однак головний криптографічний зміст зовсім в іншому. За рахунок впливу одного біта на дві підстановки швидше зростає залежність бітів результату від бітів вихідних даних. Це називається лавинним ефектом. DES спроектований так, щоб якнайшвидше домогтися залежності кожного біта шифротексту від кожного біта відкритого тексту й кожного біта ключа.

Перестановка з розширенням показана на рисунку 3.7. Іноді вона називається Е-блоком (від expansion). Для кожного 4-бітового вхідного блоку перший і четвертий біт являють собою два біти вихідного блоку, а другий і третій біти – один біт вихідного блоку. В таблиці 3.5 показано, які позиції результату

відповідають яким позиціям вихідних даних. Наприклад, біт вхідного блоку в позиції 3 переміститься в позицію 4 вихідного блоку, а біт вхідного блоку в позиції 21 – у позиції 30 і 32 вихідного блоку.

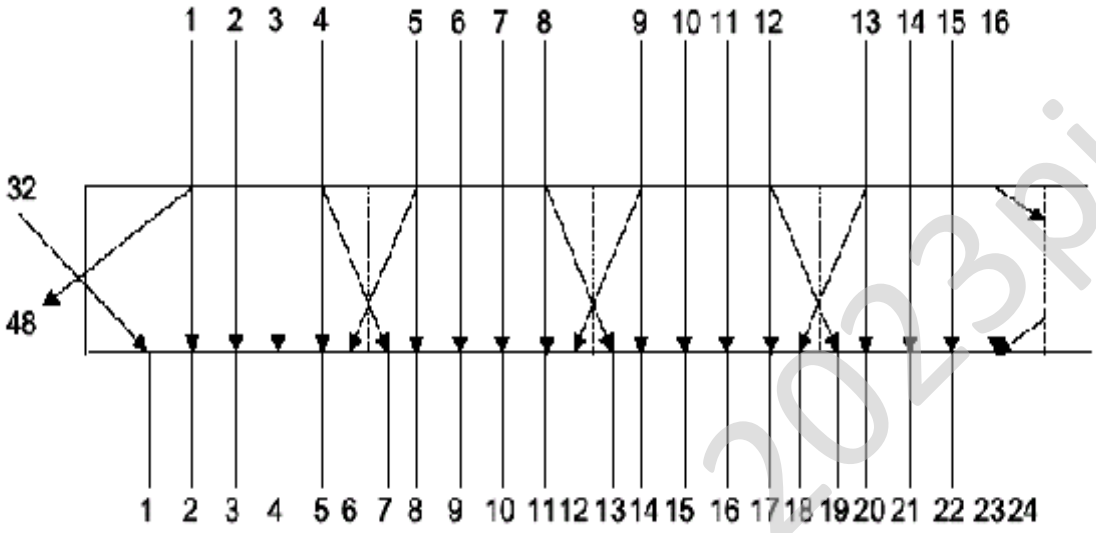

Рисунок 3.6 – Перестановка з розширенням

Хоча вихідний блок більше вхідного, кожний вхідний блок генерує унікальний вихідний блок

Таблиця 3.5 – Перестановка з розширенням DES

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

Підстановка за допомогою S-блоків: Після об'єднання стиснутого блоку з розширеним блоком за допомогою XOR над 48-бітовим результатом виконується операція підстановки. Підстановки здійснюються у вісьмох блоках підстановки, або S-блоках (від Substitution). У кожного S-блоку 6-бітовий вхід і 4-бітовий вихід, усього використовується вісім різних S-блоків. (для восьми S-блоків DES буде потрібно 256 байтів пам'яті.) 48 бітів діляться на вісім 6-бітових підблока.

Кожний окремий підблок обробляється окремим S-блоком: перший підблок – S-блоком 1, другий – S-блоком 2 і так далі.

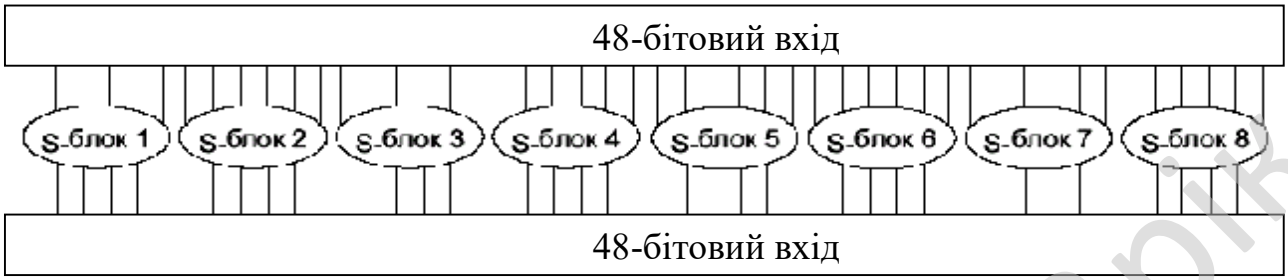

Рисунок 3.7 – Підстановка S-блоку

Кожний S-блок являє собою таблицю з 2 рядків і 16 стовпців. Кожний елемент у блоці є 4– бітовим числом. По 6 вхідних бітах S-блоку визначається, під якими номерами стовпців і рядків шукати вихідне значення.

Перестановка за допомогою P-блоків: 32-бітовий вихід підстановки за допомогою S-блоків, перетасовуються відповідно до P-блоку. Ця перестановка переміщає кожний вхідний біт в іншу позицію, жоден біт не використовується двічі, і жоден біт не ігнорується. Цей процес називається прямою перестановкою або просто перестановкою. Наприклад, біт 21 переміщається в позицію 4, а біт 4 – у позицію 31.

Таблиця 3.6 – перестановка за допомогою P-блоків

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Нарешті, результат перестановки за допомогою P-блоку поєднується за допомогою XOR з лівою половиною первісного 64-бітового блоку. Потім ліва й права половини міняються місцями, і починається наступний етап.

Заключна перестановка є зворотною відносно початкової перестановки. Зверніть увагу, що ліва й права половини не міняються місцями після останнього етапу DES, замість цього об'єднаний блок $R_{16}L_{16}$ використовується як вхід

заключної перестановки. У цьому немає нічого особливого, перестановка половинок з наступним циклічним зсувом привела б до точно такого ж результату. Це зроблено для того, щоб алгоритм можна було використовувати як для шифрування, так і для дешифрування.

Таблиця 3.7 – Заключна перестановка DES

40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

Дешифрування DES

Після всіх підстановок, перестановок, операцій XOR і циклічних зсувів можна припустити, що алгоритм дешифрування різко відрізняється від алгоритму шифрування, і також дуже заплутаний. Навпроти, різні компоненти DES були підібрані так, щоб виконувалася дуже корисна властивість: для шифрування й дешифрування використовується той самий алгоритм. DES дозволяє використовувати для шифрування або дешифрування блоку ту саму функцію. Єдина відмінність полягає в тому, що ключі повинні використовуватися у зворотному порядку. Тобто, якщо на етапах шифрування використовувалися ключі K_1, K_2, K_3, ..., K_{16}, то ключами дешифрування будуть K_{16}, K_{15}, K_{14}, ..., K_1. Алгоритм, що створює ключ для кожного етапу, також циклічний. Ключ зсувається праворуч, а число позицій зсуву дорівнює 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1.

Потрійний DES

У цей час основним недоліком DES вважається маленька довжина ключа, тому вже давно почали розроблятися різні альтернативи цьому алгоритму шифрування. Один з підходів полягає в тому, щоб розробити новий алгоритм, і успішний тому приклад – IDEA. Інший підхід припускає повторне застосування шифрування за допомогою DES з використанням декількох ключів.

Алгоритм Blowfish

Blowfish є мережею Фейштеля, у якої кількість ітерацій дорівнює 16. Довжина блоку дорівнює 64 бітам, ключ може мати будь-яку довжину в межах 448 біт. Хоча перед початком будь-якого шифрування виконується складна фаза ініціалізації, саме шифрування даних виконується досить швидко.

Алгоритм призначений в основному для додатків, у яких ключ міняється нечасто, до того ж існує фаза початкового рукостискання, під час якої відбувається автентифікація сторін і узгодження загальних параметрів і секретів. Класичним прикладом подібних додатків є мережна взаємодія. При реалізації на 32-бітних мікропроцесорах з великим кешем даних Blowfish значно швидше DES.

Алгоритм складається із двох частин:

– розширення ключа;

– шифрування даних.

Розширення ключа перетворить ключ довжиною, принаймні, 448 біт у кілька масивів підключей загальною довжиною 4168 байт.

В основі алгоритму лежить мережа Фейштеля з 16 ітераціями. Кожна ітерація складається з перестановки, що залежить від ключа, і підстановки, що залежить від ключа й даних. Операціями є `XOR` і додавання 32-бітних слів.

Blowfish використовує велику кількість підключей. Ці ключі повинні бути обчислені заздалегідь, до початку будь-якого шифрування або дешифрування даних.

Алгоритм IDEA

IDEA (International Data Encryption Algorithm) є блоковим симетричним алгоритмом шифрування. IDEA є одним з декількох симетричних криптографічних алгоритмів, якими спочатку передбачалося замінити DES.

Принципи розробки

IDEA є блоковим алгоритмом, що використовує 128-бітовий ключ для шифрування даних блоками по 64 біта.Метою розробки IDEA було створення щодо стійкого криптографічного алгоритму з досить простою реалізацією.

Криптографічна стійкість

Наступні характеристики IDEA характеризують його криптографічну стійкість:

1. Довжина блоку: довжина блоку повинна бути достатньою, щоб сховати всі статистичні характеристики вихідного повідомлення. З іншого боку, складність реалізації криптографічної функції зростає експоненціально відповідно до розміру блоку. Використання блоку розміром в 64 біта в 90-і роки означало достатню силу. Більше того, використання режиму шифрування CBC говорить про подальше посилення цього аспекту алгоритму.

2. Довжина ключа: довжина ключа повинна бути досить великою для того, щоб запобігти можливості простого перебору ключа. При довжині ключа 128 біт IDEA вважається досить безпечним.

3. Конфузія: зашифрований текст повинен залежати від ключа складним і заплутаним способом.

4. Дифузія: кожний біт незашифрованого тексту повинен впливати на кожний біт зашифрованого тексту. Поширення одного незашифрованого біта на велику кількість зашифрованих біт приховує статистичну структуру незашифрованого тексту. Визначити, як статистичні характеристики зашифрованого тексту залежать від статистичних характеристик незашифрованого тексту, повинне бути непросто. IDEA із цього погляду є дуже ефективним алгоритмом.

В IDEA два останніх пункти виконуються за допомогою трьох операцій. Це відрізняє його від DES, де все побудовано на використанні операції `XOR` і маленьких нелінійних S-boxes.

Генератори випадкових чисел

Випадкові числа відіграють важливу роль при використанні криптографії в різних мережних додатках, що відносяться до безпеки. Зробимо короткий огляд вимог, пропонованих до випадкових чисел у додатках мережної безпеки, а потім розглянемо кілька способів створення випадкових чисел.

Вимоги до випадкових чисел

Більшість алгоритмів мережної безпеки, заснованих на криптографії, використовують випадкові числа. Двома основними вимогами до послідовності випадкових чисел є випадковість і непередбачуваність.

Випадковість

Звичайно при створенні послідовності псевдовипадкових чисел передбачається, що дана послідовність чисел повинна бути випадкової в деякому певному статистичному змісті. Наступні два критерії використовуються для доказу того, що послідовність чисел є випадковою:

1. Однорідний розподіл: розподіл чисел у послідовності повинне бути однорідним; це означає, що частота появи кожного числа повинна бути приблизно однаковою.

2. Незалежність: жодне значення в послідовності не повинне залежати від інших.

Хоча існують тести, що показують, що послідовність чисел відповідає деякому розподілу, такому як однорідний розподіл, тесту для "доказу" незалежності немає. Проте, можна підібрати набір тестів для доказу того, що послідовність є залежною. Загальна стратегія припускає застосування набору таких тестів доти, поки не буде впевненості, що незалежність існує.

Непередбачуваність

У додатках, таких як взаємна автентифікація й генерація ключа сесії, немає твердої вимоги, щоб послідовність чисел була статистично випадкової, але члени послідовності повинні бути непередбачені. При "правильній" випадковій послідовності кожне число статистично не залежить від інших чисел і, отже, непередбачено. Однак правильні випадкові числа на практиці використовуються досить рідко, частіше послідовність чисел, що повинна бути випадкової, створюється деяким алгоритмом. У цьому випадку необхідно, щоб супротивник не міг угадати наступні елементи послідовності, ґрунтуючись на знанні попередніх елементів і використовуваного алгоритму.

Джерела випадкових чисел

Джерела дійсно випадкових чисел знайти важко. Фізичні генератори шумів, такі як детектори подій іонізуючої радіації, газові розрядні трубки й конденсатор, що тече, можуть бути такими джерелами. Однак ці пристрої в додатках мережної безпеки застосовуються обмежено. Проблеми також викликають грубі атаки на такі пристрої. Альтернативним рішенням є створення набору з великої кількості випадкових чисел і опублікування його в деякій книзі. Проте, і такі набори забезпечують дуже обмежене джерело чисел у порівнянні з тією кількістю, що потрібна додаткам мережної безпеки. Більше того, хоча набори із цих книг дійсно забезпечує статистичну випадковість, вони передбачувані, тому що супротивник може одержати їхню копію.

Таким чином, додатки, що шифрують, використовують для створення випадкових чисел спеціальні алгоритми. Ці алгоритми детерміновані й, отже, створюють послідовність чисел, що не є статистично випадковою. Проте, якщо алгоритм гарний, отримана послідовність буде проходити багато тестів на випадковість. Такі числа часто називають **псевдовипадковими числами**.

Розглянемо кілька алгоритмів генерації випадкових чисел.

Генератори псевдовипадкових чисел

Першою широко використовуваною технологією створення випадкового числа був алгоритм, запропонований Лехмером, що відомий як метод лінійного конгруента. Цей алгоритм параметризується чотирма числами в такий спосіб:

m	Модуль (підстава системи)	$m > 0$
a	Множник	$0 \leq a < m$
c	Збільшення	$0 \leq c < m$
X_0	Початкове значення або зерно (seed)	$0 \leq X_0 < m$

Послідовність випадкових чисел $\{X_n\}$ виходить за допомогою наступної ітераційної рівності:

$$X_{n+1} = (a\,X_n + c) \bmod m.$$

Якщо m, a й c є цілими, то створюється послідовність цілих чисел у діапазоні $0 \leq X_n < m$.

Вибір значень для a, c и m є критичним для розробки гарного генератора випадкових чисел.

Очевидно, що m повинне бути дуже більшим, щоб була можливість створити багато випадкових чисел. Вважається, що m повинне бути приблизно дорівнючим максимальному позитивному цілому числу для даного комп'ютера. Таким чином, звичайно m близько або дорівнює 2^{31}.

Існує три критерії, які використовуються при виборі генератора випадкових чисел:

1. Функція повинна створювати повний період, тобто повинні існувати всі числа між 0 і m до того, як створювані числа почнуть повторюватися.

2. Створювана послідовність повинна з'являтися випадково. Послідовність не є випадкової, тому що вона створюється детерміновано, але різні статистичні тести, які можуть застосовуватися, повинні показувати, що послідовність випадкова.

3. Функція повинна ефективно реалізовуватися на 32-бітних процесорах, а в перспективі на 64-бітних.

Значення a, c и m повинні бути обрані таким чином, щоб ці три критерії виконувалися. Відповідно до першого критерію можна показати, що якщо m є простим і $c = 0$, то при певному значенні a період, створюваний функцією, буде дорівнює m-1. Для 32-бітної арифметики відповідне просте значення $m = 2^{31} - 1$. Таким чином, функція створення псевдовипадкових чисел має вигляд:

$$X_{n+1} = (a\ X_n) \bmod (2^{31} - 1).$$

Тільки невелике число значень a задовольняє всім трьом критеріям. Одне з таких значень є $a = 7^5 = 16807$, що використовувалося в сімействі комп'ютерів IBM 360. Цей генератор широко застосовується й пройшов більше тисячі тестів, більше, ніж всі інші генератори псевдовипадкових чисел.

Сила алгоритму лінійного конгруента в тім, що якщо співмножник і модуль (підстава) відповідним чином підібрані, то результуюча послідовність чисел буде статистично невідзначна від послідовності, що є випадкової з набору 1, 2, ..., m-1. Але не може бути випадковості в послідовності, отриманої з використанням алгоритму, незалежно від вибору початкового значення X_0. Якщо значення обране, то числа, що залишилися, у послідовності будуть визначені. Це завжди враховується при криптоаналізі.

Якщо супротивник знає, що використовується алгоритм лінійного конгруента, і якщо відомі його параметри ($a = 7^5$, $c = 0$, $m = 2^{31} - 1$), то, якщо розкрито одне число, вся послідовність чисел стає відома. Навіть якщо супротивник знає тільки, що використовується алгоритм лінійного конгруента, знання невеликої частини послідовності досить для визначення параметрів алгоритму й всіх наступних чисел. Припустимо, що супротивник може визначити значення X_0, X_1, X_2, X_3. Тоді :

$$X_1 = (a X_0 + c) \bmod m.$$
$$X_2 = (a X_1 + c) \bmod m.$$
$$X_3 = (a X_2 + c) \bmod m.$$

Ці рівності дозволяють знайти a, c и m.

Таким чином, хоча алгоритм і є гарним генератором псевдовипадкової послідовності чисел, бажано, щоб реально використовувана послідовність була непередбаченою, оскільки в цьому випадку знання частини послідовності не дозволить визначити майбутні її елементи. Ця мета може бути досягнута декількома способами. Наприклад, використання внутрішніх системних годинників для модифікації потоку випадкових чисел. Один зі способів застосування годинників складається в перезапуску послідовності після N чисел, використовуючи поточне значення годин за модулем m у якості нового початкового значення. Інший спосіб складається в простому додаванні значення поточного часу до кожного випадкового числа за модулем m.

Криптографічно створені випадкові числа

У криптографічних додатках доцільно шифрувати випадкові числа, що виходять. Найчастіше використовується три способи.

Циклічне шифрування

У цьому випадку застосовується спосіб створення ключа сесії з майстра-ключа. Лічильник з періодом N використовується як вхід у пристрій, що шифрує. Наприклад, у випадку використання 56-бітного ключа DES може застосовуватися лічильник з періодом 2^{56}. Після кожного створеного ключа значення лічильника збільшується на 1. Таким чином, псевдовипадкова послідовність, отримана за даною схемою, має повний період: кожне вихідне значення X_0, X_1,...X_{N-1} засновано на різних значеннях лічильника й, отже, $X_0 \neq X_1 \neq X_{N-1}$. Так як майстер-ключ захищений, легко показати, що будь-який секретний ключ не залежить від знання одного або більше попередніх секретних ключів.

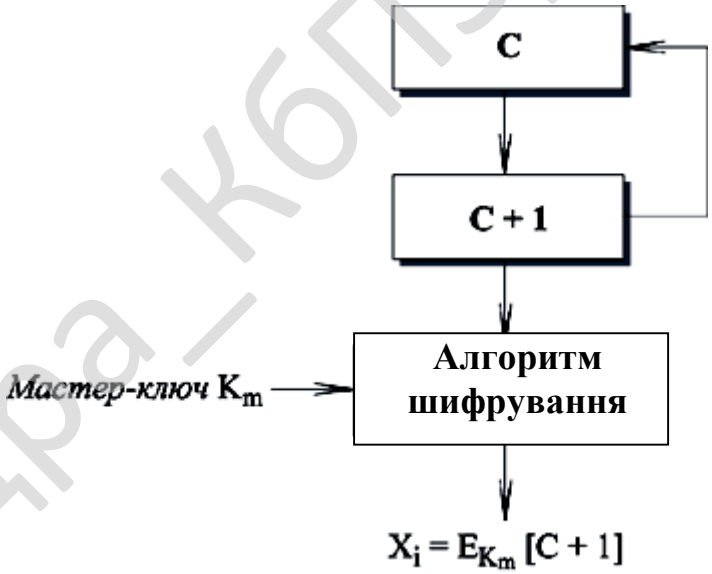

Рисунок 3.8 – Циклічне шифрування

Для подальшого посилення алгоритму вхід повинен бути виходом повноперіодичного генератора псевдовипадкових чисел, а не простою послідовністю.

Режим Output Feedback DES

Режим OFB DES може застосовуватися для генерації ключа, аналогічно тому, як він використовується для потокового шифрування. Помітимо, що виходом кожної стадії шифрування є 64-бітне значення, з якого тільки ліві j бітів подаються назад для шифрування. 64-бітні виходи становлять послідовність псевдовипадкових чисел з гарними статистичними властивостями.

Генератор псевдовипадкових чисел ANSI X9.17

Генератор псевдовипадкових чисел побудований згідно ANSI X9.17. У число додатків, що використовують цю технологію, входять додатки фінансової безпеки й PGP.

Алгоритмом шифрування є потрійний DES. Генератор ANSI X9.17 складається з наступних частин:

1. Вхід: генератором управляють два псевдовипадкових входи. Один є 64-бітним поданням поточної дати й часу, які змінюються щораз при створенні числа. Інший є 64-бітним початковим значенням; воно ініціюється деяким довільним значенням і змінюється в ході генерації послідовності псевдовипадкових чисел.

2. Ключі: генератор використовує три модулі потрійного DES. Всі три використовують ту саму пару 56-бітних ключів, що повинні триматися в секреті й застосовуватися тільки для генерації псевдовипадкового числа.

3. Вихід: вихід складається з 64-бітного псевдовипадкового числа й 64-бітного значення, що буде використовуватися як початкове значення при створенні наступного числа.

DT_i – значення дати й часу на початок і-ої стадії генерації.

V_i – початкове значення для і-ої стадії генерації.

R_i – псевдовипадкове число, створене на і-ії стадії генерації.

K_1, K_2 – ключі, використовувані на кожній стадії.

Тоді: $R_i = EDE_{K1,K2} [EDE_{K1,K2} [DT_i] \oplus V_i]$, $V_{i+1} = EDE_{K1,K2} [EDE_{K1,K2} [DT_i] R_i]$.

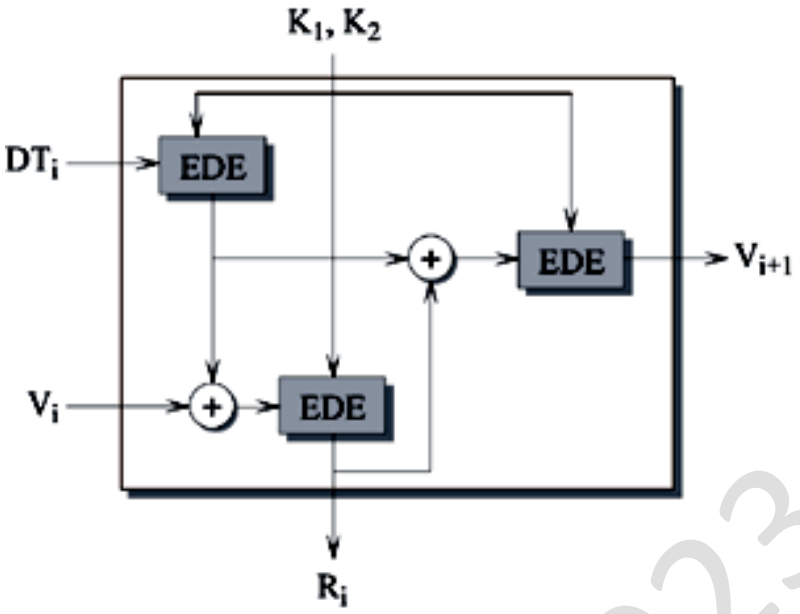

Рисунок 3.9 – Генератор псевдовипадкових чисел ANSI X9.17

Схема включає використання 112-бітного ключа й трьох EDE-шифрувань. На вхід подаються два псевдовипадкових значення: значення дати і часу та початкове значення чергової ітерації, на виході створюються початкове значення для наступної ітерації й чергове псевдовипадкове значення. Навіть якщо псевдовипадкове число R_i буде скомпрометовано, обчислити V_{i+1} з R_i неможливо, і, отже, випливає псевдовипадкове значення R_{i+1}, тому що для одержання V_{i+1} додатково виконуються три операції EDE.

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

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.10. З нього видно, що процеси взаємодіють наступним чином.

Спершу завантажується процес виведення головного вікна програми. Він взаємодіє з наступними процесами:

– Процес шифрування/дешифрування файлів.

– Процес обчислення геш-значень файлів.

– Процес вибору/створення файлів для збереження ключів.

– Процес вибору параметрів системи захисту інформації.

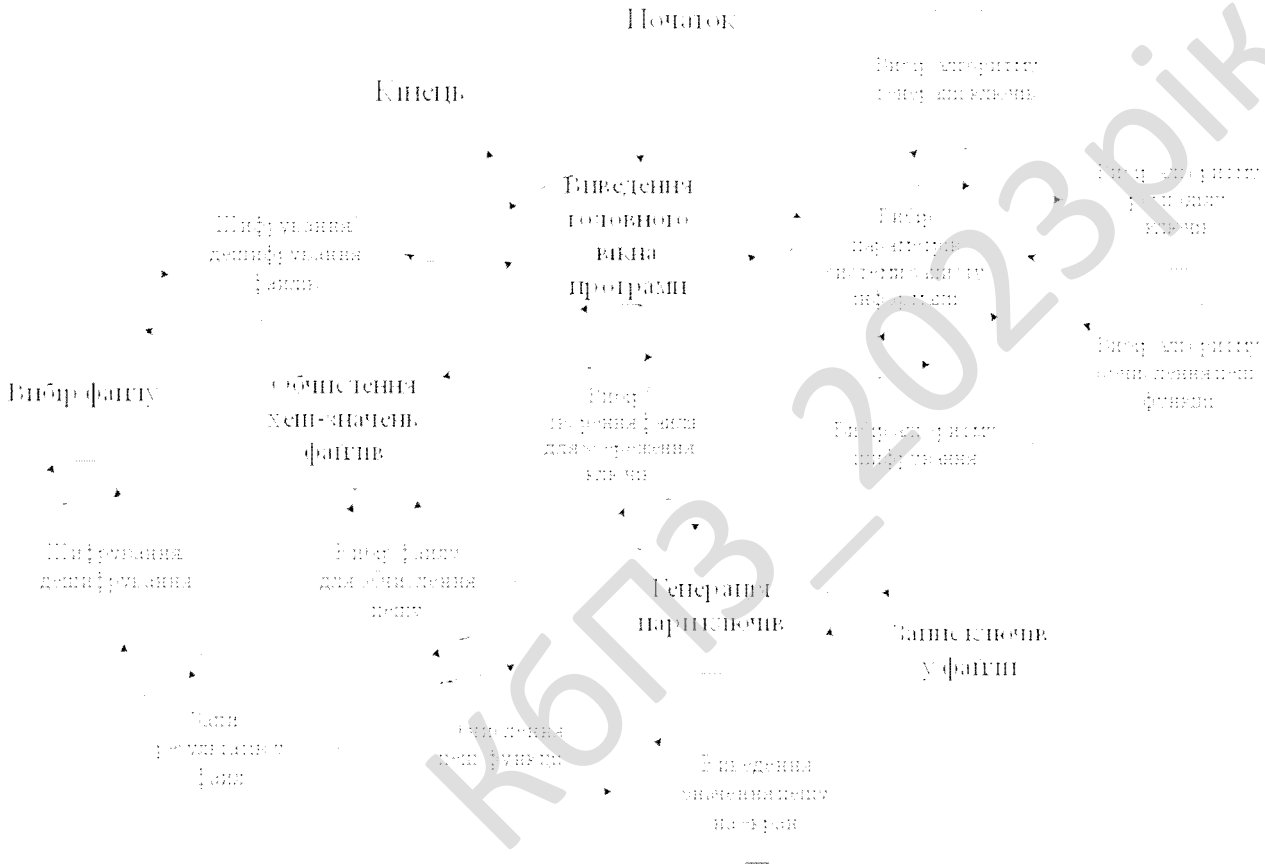

Рисунок 3.10 – Діаграма процесів системи

Процес шифрування/дешифрування файлів взаємодіє з процесом вибору файлів.

Процес вибору файлів взаємодіє з процесом шифрування/дешифрування.

Процес шифрування/дешифрування взаємодіє з процесом запису результату у файл.

Наступний процес обчислення геш-значень файлів взаємодіє з процесом вибору файлів для обчислення гешу.

Процес вибору файлів для обчислення гешу взаємодіє з процесом

обчислення геш-функції.

Процес обчислення геш-функції взаємодіє з процесом виведення значення гешу на екран.

Наступний процес вибору/створення файлів для збереження ключів взаємодіє з процесом генерації пари ключів.

Процес генерації пари ключів взаємодіє з процесом запису ключів у файл.

Процес вибору параметрів системи захисту інформації взаємодіє з наступними процесами:

– Процес вибору алгоритму генерації ключів.

– Процес вибору алгоритму розподілу ключів.

– Процес вибору алгоритму обчислення геш-функції.

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

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків.

Спершу відбувається виведення основного вікна програми. Після цього послідовно виконуються наступні дії:

– Вибір алгоритму генерації ключів.

– Вибір алгоритму розподілу ключів.

– Вибір алгоритму обчислення геш-функцій.

– Вибір алгоритму шифрування.

– Вибір алгоритму створення файлів для збереження ключів.

Якщо необхідно створити ключ, тоді виконуються наступні дії:

– Відбувається генерація відкритого та закритого ключів.

– Записується закритий ключ у файл.

– Записується відкритий ключ у файл.

Якщо необхідно обчислити геш-функцію, тоді відбувається виконання наступних дій:

– Вибір файлу, для обчислення геш-значення.

– Обчислення геш-функції.

– Виведення значення гешу на екран.

Якщо необхідно шифрувати файл, тоді виконуються наступні дії:

– Вибір файлу для шифрування.

– Шифрування файлу, та збереження результатів.

Якщо необхідно дешифрувати файл, тоді виконуються наступні дії:

– Вибір файлу для дешифрування.

– Дешифрування файлу, та збереження результатів.

Після цього користувач обирає, працювати йому далі з програмою, або ні.

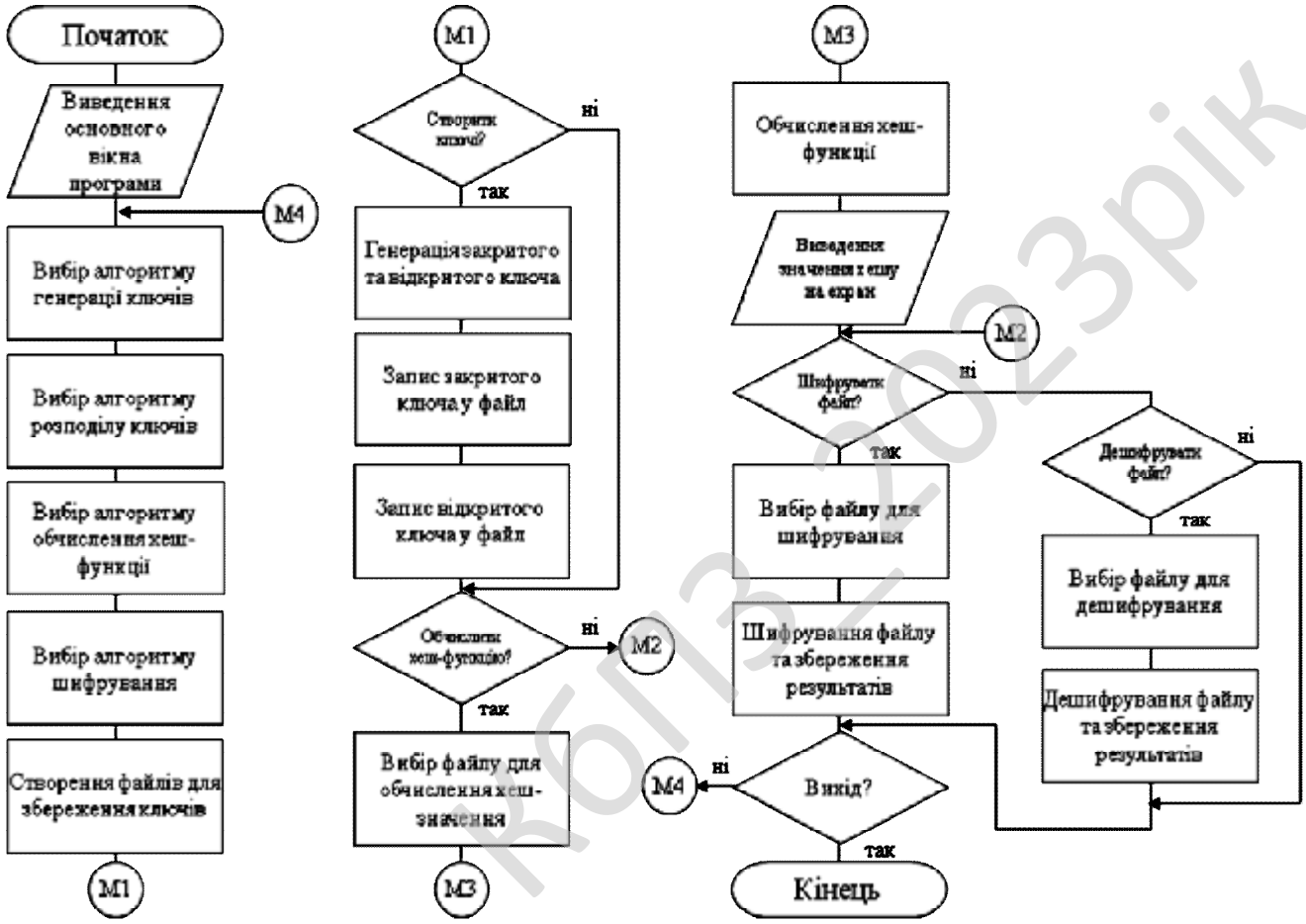

Рисунок 4.1 – Блок-схема роботи основної програми

При зберіганні і передачі даних незрідка виникає вимога захистити їх від небажаного прочитання і модифікації. Якщо завдання захисту від небажаної модифікації вирішується надлишковими кодами, що обговорювалися в попередньому розділі, то з прочитанням все істотно складніше.

Найпростіше забезпечити захист даних, позбавивши потенційних зловмисників доступу до фізичного носія даних або фізичного каналу, по якому відбувається їх передача. На жаль, інколи це нездійсненно (наприклад, якщо обмін даними відбувається по радіоканалу), а частенько просто дуже дорого. В

цьому випадку на допомогу приходять методики, збиральна назва яких – криптографія (тайнопис). На відміну від більшості термінів комп'ютерної лексики це слово не англійського, а грецького походження.

Історія криптографії налічує тисячі років, і багато засадничих принципів сучасної криптографії відомо, можливо, з доісторичних часів, проте, істотний прогрес в теорії шифрування був досягнутий лише відносно недавно, у зв'язку з розробкою сучасної теорії інформації.

Практично всі методи криптографії зводяться до перетворення даних в набір з кінцевої кількості символів і здійснення над цими символами двох основних операцій: підстановки і перестановки. Підстановка полягає в заміні одних символів на інших. Перестановка полягає в зміні порядку символів. Як символи при цьому можуть виступати різні елементи повідомлення – так, при шифруванні повідомлень на природних мовах підстановці і перестановці можуть піддаватися як окремі букви, так і слова або навіть цілі пропозиції (як, наприклад, в алегоричних викладах магічних і священних текстів). У сучасних алгоритмах цим операціям найчастіше піддаються блоки послідовних бітів. Деякі методики можна описати як здійснення операції підстановки над повним повідомленням.

Підстановки і перестановки виробляються по певних правилах. При цьому надія покладається "на те, що ці правила і використовувані в них параметри відомі лише авторові і одержувачеві шифрованого повідомлення і невідомі стороннім особам. У докомп’ютерну еру прагнули засекретити обоє складові процесу шифрування. Зараз для шифрування, як правило, використовують стандартні алгоритми, секретність же повідомлення досягається шляхом засекречування використовуваного алгоритмом параметра, ключа (key).

Прочитання секретного повідомлення сторонньою особою, теоретично, може бути здійснене двома способами: викраданням ключового значення або його вгадуванням шляхом аналізу перехопленої шифровки. Якщо перший захід може запобігти лише фізичним і організаційним захистом, то можливість другого визначається використовуваним алгоритмом. Нижче ми називатимемо процес

аналізу шифровки зломом шифру, а людини, що здійснює цей процес, – зломщиком. По-науковому ця діяльність називається більш нейтрально – криптоаналіз.

Стійкість шифру до пошуку автокореляцій в повідомленні називається криптостойкостыо алгоритму. Навіть при використанні вдалих в цьому сенсі алгоритмів, якщо зломщик знає, що вихідні (нешифровані) дані задовольняють тій або іншій вимозі, наприклад, містять певне слово або забезпечені надлишковим кодом, він може виробити повний перебір простору ключів: перебирати всі значення ключа, що допускаються алгоритмом, поки не буде отримано що задовольняє вимозі повідомлення. При використанні ключів досить великої розрядності така атака виявляється надмірно дорогою, проте прогрес обчислювальної техніки постійно зрушує кордон "достатності" все далі і далі.

Простим і ефективним способом боротьби з такою атакою є розширення простору ключів. Збільшення ключа на один біт наводить до збільшення простору удвічі – таким чином, лінійне зростання розміру ключа забезпечує експоненціальне зростання вартості перебору. Деякі алгоритми шифрування не залежать від розрядності використовуваного ключа – в цьому випадку розширення досягається очевидним способом. Якщо ж в алгоритмі присутня залежність від розрядності, розширити простір можна, всього лише застосувавши до повідомлення декілька різних перетворень, у тому числі і одним алгоритмом, але з різними ключами. Ще один спосіб істотно ускладнити роботу зломщикові – це упаковка повідомлення перед шифруванням і доповнення його випадковими бітами.

Поважно підкреслити, втім, що кількість двійкових розрядів ключа є лише оцінкою об'єму простору ключів зверху, і в багатьох ситуаціях ця оцінка завищена. Деякі алгоритми через свою природу можуть використовувати лише ключі, що задовольняють певній умові, – наприклад, RSA використовує прості Числа. Це різко звужує об'єм роботи по перебору, тому для забезпечення

порівнянною криптостійкості розрядність ключа RSA має бути набагато більше, ніж біля алгоритмів, що допускають довільні ключі.

Низька криптостійкість може бути обумовлена не лише алгоритмом шифрування, але і процедурою вибору ключа: якщо ключ може набувати будь-яких двійкових значень заданої розрядності, але реально для його вибору використовується страждаючий неоднорідністю генератор псевдовипадкових чисел, ми можемо значно скоротити об'єм простору, який реальний повинен буде перебрати зломщик наших повідомлень. Ще гірше ситуація, коли як ключ використовуються слова природної мови, що "легко запам'ятовуються": в цьому випадку реальний об'єм простору ключів навіть досить великої розрядності може вимірюватися всього лише декількома тисячами різних значень.

Якщо ключ породжений надійним генератором випадкових чисел (наприклад, правильно налагодженим оцифровщиком теплового шуму), жодна інформація про автокореляції у вихідному тексті повідомлення зломщика не допоможе: перебираючи повний простір ключів, зломщик вимушений буде перевірити всі повідомлення, співпадаючі по кількості символів з початковим, у тому числі і всі повідомлення, що задовольняють передбачуваному автокореляційному співвідношенню.

Більш практичним виявилося вживання як ключ, псевдовипадкових послідовностей, що породжуються детермінованими алгоритмами. У проміжку між першою і другою світовими війнами широкого поширення набули шифрувальні машини, засновані на механічних генераторах таких послідовностей. Найчастіше використовувалися поєднання, що отримуються при обертанні коліс з взаємно простими кількостями зубців.

Основною небезпекою при використанні таких методів шифрування є можливість визначити поточну точку послідовності – взнавши її (наприклад, по непрямих ознаках здогадавшись, що в даній точці повідомлення має бути таке-то слово, і відновивши елемент ключа, що використався при її шифруванні),

зломщик може продовжити генерацію з тієї ж крапки і розшифрувати весь подальший потік.

У системах цифрового зв'язку широке вживання отримали блокові алгоритми, що виконують над блоками даних фіксованої довжини послідовності – інколи досить складні – перестановок, підстановок і інших операцій, найчастіше двійкових складань даних з ключем по якому-небудь модулю. У операціях використовуються компоненти ключового слова відносно невеликої розрядності.

Шифри з відкритим ключем називаються також двохключовими. Якщо в алгоритмах з прихованим ключем для кодування і декодування повідомлень використовується один і той же ключ, то тут використовуються два ключі: прилюдний і приватний. Для прочитання повідомлення, закодованого прилюдним ключем, необхідний приватний, і навпаки.

Окрім звичайних міркувань криптостійкості, до таких алгоритмів пред'являється додаткова вимога: неможливість відновлення приватного ключа по прилюдному інакше як повним перебором простору ключів.

Двуключові схеми шифрування набагато складніше в розробці, чим схеми з секретним ключем: потрібно знайти перетворення, непіддатливе зверненню за допомогою прилюдного ключа, що застосовувався в нім, але таке, аби із застосуванням приватного ключа його все-таки можна було обернути.

На рисунку 4.2 зображено блок-схему алгоритму роботи підпрограми генерації випадкових чисел лінійним конгруентним методом. Підпрограма працює наступним чином. Лінійний конгруентний метод запропонований Д. Х. Лемером. Основна обчислювальна формула: $x[n+1] = (a\,x[n] + c) \bmod m$.

Алгоритм зациклюється з періодом, що не перевищує деякого m. Коефіцієнти a, m і $x(0)$ можуть приймати довільні цілі значення, за винятком 0. Параметр с може бути також і 0, але в цьому випадку скорочується період.

Крім лінійних існують інші конгруентні методи. Найвідоміші з них : мультиплікативний, мішаний і адитивний. Тепер майже всі стандартні бібліотечні

програми обчислення послідовності рівномірно розподілених випадкових чисел ґрунтуються на конгруентних методах.

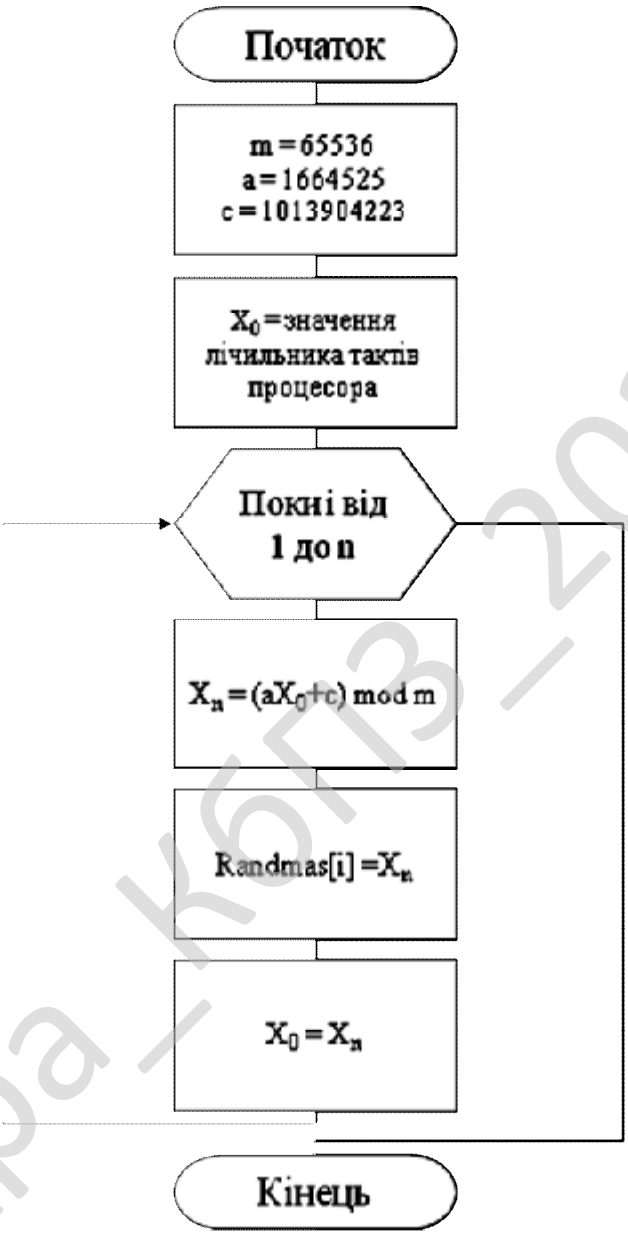

Рисунок 4.2 – Блок-схема алгоритму роботи підпрограми генерації випадкових чисел лінійним конгруентним методом

Очевидно, що *m* повинне бути дуже більшим, щоб була можливість створити багато випадкових чисел. Вважається, що m повинне бути приблизно

дорівнючим максимальному позитивному цілому числу для даного комп'ютера. Таким чином, звичайно m близько або дорівнює 2^{31}.

Існує три критерії, які використовуються при виборі генератора випадкових чисел:

1. Функція повинна створювати повний період, тобто повинні існувати всі числа між 0 і m до того, як створювані числа почнуть повторюватися.

2. Створювана послідовність повинна з'являтися випадково. Послідовність не є випадкової, тому що вона створюється детерміновано, але різні статистичні тести, які можуть застосовуватися, повинні показувати, що послідовність випадкова.

3. Функція повинна ефективно реалізовуватися на 32-бітних процесорах, а в перспективі на 64-бітних.

Значення a, c и m повинні бути обрані таким чином, щоб ці три критерії виконувалися. Відповідно до першого критерію можна показати, що якщо m є простим і $c = 0$, то при певному значенні a період, створюваний функцією, буде дорівнює m-1. Для 32-бітної арифметики відповідне просте значення $m = 2^{31} – 1$. Таким чином, функція створення псевдовипадкових чисел має вигляд:

$$X_{n+1} = (a\, X_n) \bmod (2^{31} – 1).$$

Тільки невелике число значень a задовольняє всім трьом критеріям. Одне з таких значень є $a = 7^5 = 16807$, що використовувалося в сімействі комп'ютерів IBM 360. Цей генератор широко застосовується й пройшов більше тисячі тестів, більше, ніж всі інші генератори псевдовипадкових чисел.

Сила алгоритму лінійного конгруента в тому, що якщо співмножник і модуль (підстава) відповідним чином підібрані, то результуюча послідовність чисел буде статистично невідзначна від послідовності, що є випадкової з набору 1, 2, ..., m-1. Але не може бути випадковості в послідовності, отриманої з використанням алгоритму, незалежно від вибору початкового значення X_0. Якщо значення обране, то числа, що залишилися, у послідовності будуть визначені. Це завжди враховується при криптоаналізі.

Якщо супротивник знає, що використовується алгоритм лінійного конгруента, і якщо відомі його параметри ($a = 7^5$, $c = 0$, $m = 2^{31} - 1$), то, якщо розкрито одне число, вся послідовність чисел стає відома. Навіть якщо супротивник знає тільки, що використовується алгоритм лінійного конгруента, знання невеликої частини послідовності досить для визначення параметрів алгоритму й всіх наступних чисел. Припустимо, що супротивник може визначити значення X_0, X_1, X_2, X_3. Тоді :

$$X_1 = (a\,X_0 + c) \bmod m$$

$$X_2 = (a\,X_1 + c) \bmod m$$

$$X_3 = (a\,X_2 + c) \bmod m$$

Ці рівності дозволяють знайти a, c й m.

Таким чином, хоча алгоритм і є гарним генератором псевдовипадкової послідовності чисел, бажано, щоб реально використовувана послідовність була непередбаченою, оскільки в цьому випадку знання частини послідовності не дозволить визначити майбутні її елементи. Ця мета може бути досягнута декількома способами. Наприклад, використання внутрішніх системних годинників для модифікації потоку випадкових чисел. Один зі способів застосування годинників складається в перезапуску послідовності після N чисел, використовуючи поточне значення годинника за модулем m у якості нового початкового значення. Інший спосіб складається в простому додаванні значення поточного часу до кожного випадкового числа за модулем m.

Розглянемо основні процедури й функції, що забезпечують роботу модулів системи генерації ключів і обміну конфіденційною інформацією. Основна частина цих функцій реалізована в модулі Unit1.

Змінні, використовувані в розглянутому модулі іменовані відповідно до математичної моделі. Так для роботи алгоритму RSA використовуються змінні цілого типу (integer):

P, Q – прості числа для реалізації алгоритму RSA.

N – модуль, по якому виконується шифрування повідомлення.

fn – функція Ейлера.

Ka, Kb – відкритий і особистий ключі користувача системи.

Можливості, надавані користувачам системи клієнтським додатком, визначені в модулі як елементи масиву ClCmds.

ClCmds:array [1..3] of string = ('-msg', '-RSA', '-sig');

Тут:

'-msg' – передача відкритого повідомлення;

'-RSA' – організація процесу шифрування й передачі повідомлення;

'-sig' – реалізація цифрового підпису для повідомлення.

Процедура ChooseCommand (RecText:string) відповідає за обробку команди сервера й вибір потрібної дії.

Центральне місце в реалізації алгоритмів шифрування й цифрового підпису займає функція знаходження НЗД – iNOD.

```
function iNOD (a, b:real):real;
var
      i:real;
      xnod:real;
begin
      if a<b     then xnod:=a
      else xnod:=b;
      i:=xnod;
      while i>0 do
      begin
            if ((iMod (a, i)=0) and (iMod (b, i) = 0))
            then
            begin
                  xnod:=i;
                  i:=0;
            end
            else
            i:= i-1;
      end;
      iNOD:=xnod;
end;
```

Генерація ключів реалізована в процедурі RSA_keys.

```
procedure RSA_keys (P, Q:integer; var N, fn, Ka, Kb:integer);
begin
	N:=P*Q;
	fn:=( P-1)*( Q-1);
	randomize;
	repeat
	begin
		repeat
		Ka:=(Random ( fn-2)+2)
		until (iNod (fn, ka)=1);
		repeat
		Kb:=(Random ( fn-2)+2)
		until (((Ka*Kb) mod fn=1));
	end
	until ((Ka<>Kb) and (iNod (Kb, N)=1));
end;
```

Ключі, згенеровані в процедурі RSA_keys використовуються у функції шифрування повідомлення методом RSA – shifr. Залежно від того, яке значення Ka або Kb було привласнено формальному параметру Kab, буде виконаються або шифрування повідомлення або його розшифровування.

```
function shifr (Kab, N:integer; input:string):string;
var
	i, code:integer;
	code, cod:real;
	res, tmp:string;
	iCode, iKab, j, iRes:TFGInt;
begin
	res:='';
	Base10StringToFGInt (FloatToStr(Kab), iKab);
	Base10StringToFGInt (IntToStr(N), j);
	for i:= 1 to length(input) do
	begin
		code:=ord (input[i]);
		Base10StringToFGInt (FloatToStr(code), iCode);
		FGIntModExp (iCode, iKab, j, iRes);
		FGIntToBase10String (iRes, tmp);
		code:=StrToInt(tmp);
		res:=res+ chr(codes);
	end;
```

```
        shifr:=res;
    end;
```

Для організації цифрового підпису було ухвалене рішення використовувати геш-функцію, що обробляє блоки даних в 64 біт і, використовуючи операцію додавання за модулем 2, на виході видає дайджест довжиною в 8 біт. Тому весь переданий текст необхідно перевести у двійкові коди. Для цього використовувалася функція function ch8to64b (InputText:string):string, що переводить 8 символів таблиці ASCII кодів в 64-бітний блок:

```
function ch8to64b (InputText:string):string;
var
      i, j:integer;
      BinText, tmp:string;
      smb, tsmb:array [1..8] of string;
begin
      for i:=1 to 8 do
      begin
            tsmb[i]:=IntToBin (ord(InputText[i]));
            for j:=1 to 8 do
            begin
                  tmp:=tsmb[i];
                  smb[i]:=smb[i]+tmp [j+24];
            end;
      end;
      for i:= 1 to 8 do
      BinText:=BinText+smb[i];
      ch8to64b:=BinText;
end;
```

Функція function xHash (inp: string):string гешує n-у кількість символів, використовуючи у своїй роботі функцію гешування двох символів.

```
function xHash (inp: string):string;
var
      tmp, res, inp2:string;
      i:integer;
begin
      inp2:=inp;
      while (length(inp) mod 8 <>0) do
      begin
```

```
            np:=inp+'a';
        end;
        tmp:='';
        res:='';
        for i:=1 to length(inp) do
        begin
            tmp:=tmp+inp[i];
            if (i mod 8 = 0) then
            begin
                res:= res+iHash(tmp);
                tmp:='';
            end;
        end;
        xHash:=res;
end;
```

Виконавши гешування повідомлення, необхідно організувати підписання отриманого дайджесту особистим ключем користувача системи, іншими словами необхідно зашифрувати дайджест. Шифрування дайджесту відбувається й використанням функції shifr_hash. Залежно від того, яке значення Ka або Kb було привласнено формальному параметру Keyb, буде виконуватися або шифрування дайджесту повідомлення, або його розшифровування.

```
function shifr_hash (res1:string; Keyb, n:real):string;
var
    i, fx, fcode:integer;
    res2, tmp:string;
    iKab, j, ifx, iRes:TFGInt;
begin
    res2:='';
    Base10StringToFGInt (FloatToStr(Keyb), iKab);
    Base10StringToFGInt (FloatToStr(N), j);
    for i:=1 to length(res1) do
    begin
        fx:=ord (res1 [i]);
        Base10StringToFGInt (FloatToStr(fx), ifx);
        FGIntModExp (ifx, iKab, j, iRes);
        FGIntToBase10String (iRes, tmp);
        fcode:=StrToInt(tmp);
        res2:=res2+chr(fcode);
    end;
    shifr_hash:=res2;
end.
```

Для передачі підписаного повідомлення необхідно приєднати дайджест. Було ухвалене рішення про те, що дайджест повинен передаватися у вигляді тексту для зручності контролю цілісності, тому перехід від бінарного подання дайджесту до символьного (в ASCII-Символах), був виконаний через функцію перекладу в десяткову систему числення BinToInt.

```
Function BinToInt (binText:string):longint;
var
      bin, mult:longint;
      i:integer;
begin
      mult:=1;
      bin:=0;
      for i:=length(binText) downto 1 do
      begin
            if binText[i]='1' then bin:=bin+mult;
            mult:=mult shl 1;
      end;
      BinToInt:=bin;
End.
```

4.2 Захист розробленого програмного забезпечення

Дані в програмі захищаються за допомогою використання алгоритму SHA-3 (Keccak) – алгоритм гешування змінної розрядності, розроблений групою авторів на чолі з Йоаном Дайменом, співавтором Rijndael, автором шифрів MMB, SHARK, Noekeon, SQUARE і BaseKing. 2 жовтня 2012 року Keccak став переможцем конкурсу криптографічних алгоритмів, проведеним Національним інститутом стандартів і технологій США. 5 серпня 2015 року алгоритм затверджено та опубліковано в якості стандарту FIPS 202. У програмній реалізації автори заявляють про 12,5 циклах на байт при виконанні на ПК з процесором Intel Core 2. Проте в апаратних реалізаціях Keccak виявився набагато швидшим, ніж всі інші фіналісти.

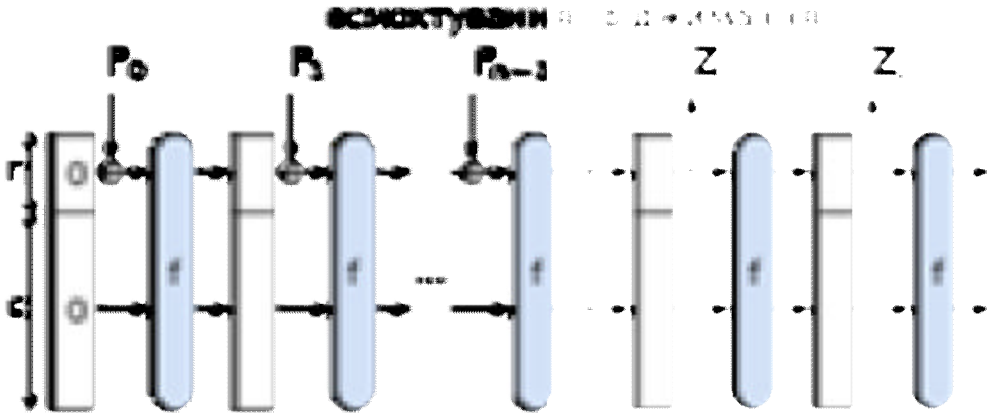

Рисунок 4.3 – Конструкція функції губки, використана в геш-функції

Конструкція функції губки, використана в геш-функції. P_i – вхідні блоки, Z_j – вихід алгоритму. Невикористаний для виведення набір бітів c («capacity») повинен мати значний розмір для досягнення стійкості до атак.

Алгоритм SHA-3 побудований за принципом криптографічної губки (дана структура криптографічних алгоритмів була запропонована авторами алгоритму Keccak раніше).

Геш-функції сімейства SHA-3 побудовані на основі конструкції криптографічної губки, в якій дані спочатку «вбираються» в губку, при якому початкове повідомлення M піддається багатораундовим перестановкам f, потім результат Z «віджимається» з губки. На етапі «вбирання» блоки повідомлення додаються за модулем 2 з підмножиною стану, який потім перетвориться з допомогою функції перестановки f. На етапі «віджимання» вихідні блоки зчитуються з одного і того ж підмножинного стану, зміненого функцією перестановок f. Розмір частини стану, який записується і зчитується, називається «швидкістю» (англ. rate) і позначається r, а розмір частки, яка незаймана введенням / виведенням, називається «ємністю» (англ. capacity) і позначається c.

Алгоритм отримання значення геш-функції можна розділити на кілька етапів:

– Вихідне повідомлення M додається до рядка P довжини, кратній r, за допомогою функції доповнення (pad-функції).

– Рядок P ділиться на n блоків довжини r: P_0, P_1,..., P_{n-1}

– «Всмоктування»: кожен блок P_i доповнюється нулями до рядка довжини b біт і підсумовується по модулю 2 з рядком стану S, де S – рядок довжини b біт (b = r + c). Перед використанням цієї функції всі елементи S дорівнюють нулю. Для кожного наступного блоку стан – рядок, отриманий застосуванням функції перестановок f до результату попереднього кроку.

– «Віджимання»: поки довжина Z менша d (d – кількість біт в результаті геш-функції), до Z додається r перших біт стану S, після кожного додавання до S, застосовується функція перестановок f. Потім S обрізається до довжини d біт

– Рядок Z довжини d біт повертається в якості результату

Завдяки тому, що стан містить c додаткових біт, алгоритм стійкий до атаки подовженням повідомлення, до якої прийняті алгоритми SHA-1 і SHA-2.

У SHA-3 стан S – це масив 5 × 5 слів довжиною w = 64 біта, всього 5 × 5 × 64 = 1600 біт. Також в Keccak можуть використовуватися довжини w, рівні меншим ступенями 2 (від w = 1 до w = 32).

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображений скріншот головного вікна програми. З нього видно, що інтерфейс користувача програми розбито на наступні логічні блоки:

– Блок меню.

– Блок генерації та розподілу ключів.

– Блок реалізації геш-функцій.

– Блок шифрування.

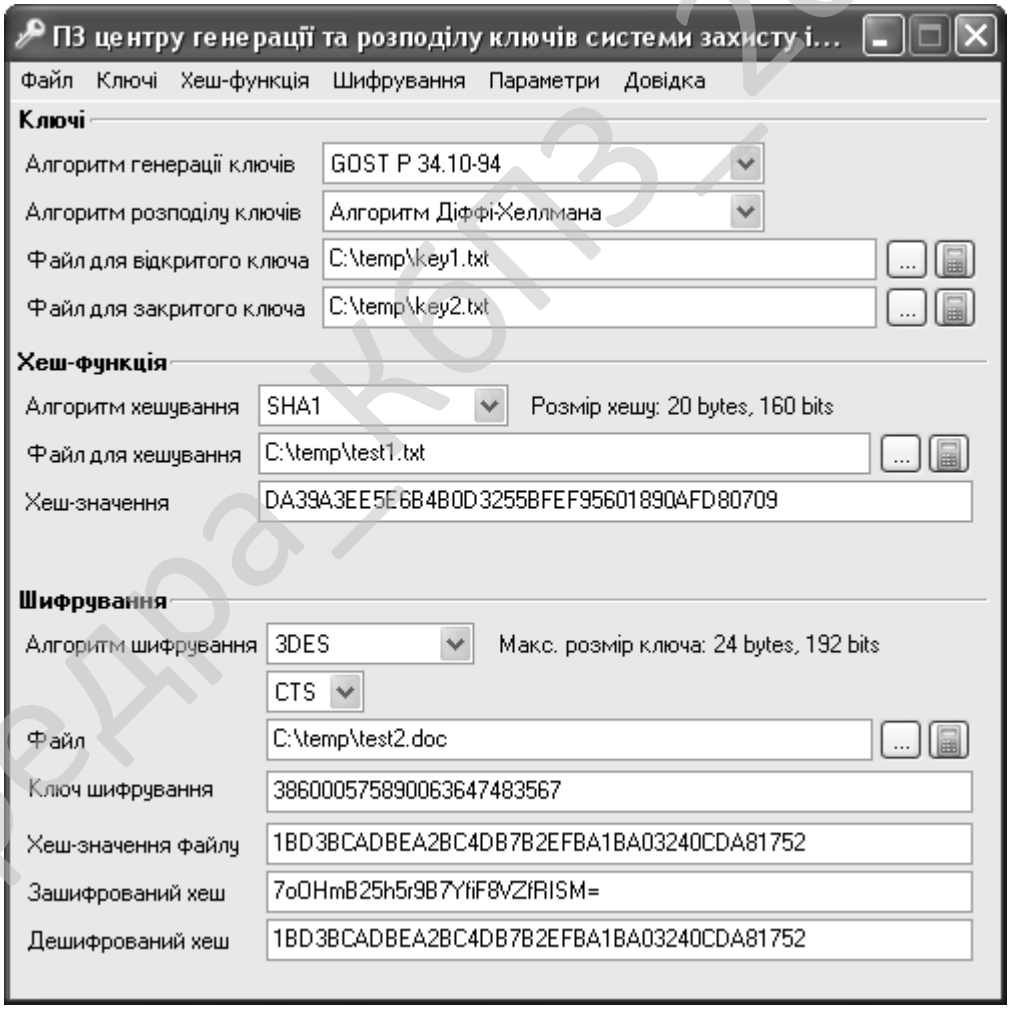

Рисунок 5.1 – Головне вікно програми

Рисунок 5.2 – Довідка

Блок меню складається з наступних елементів:

– Файл.

– Ключі.

– Геш-функція.

– Шифрування.

– Параметри.

– Довідка.

Блок генерації та розподілу ключів з наступних елементів:

– Алгоритм генерації ключів.

– Алгоритм розподілу ключів.

– Файл для збереження відкритого ключа.

– Файл для збереження закритого ключа.

Блок геш-функцій складається з наступних елементів:

– Алгоритм гешування.

– Файл для гешування.

– Геш-значення.

Блок шифрування складається з наступних елементів:

– Алгоритм шифрування.

– Файл, який необхідно зашифрувати, або дешифрувати.

– Ключ шифрування.

– Геш-значення файлу.

– Зашифрований геш.

– Дешифрований геш.

На рисунку 5.2 зображено вікно довідки. На ньому наведено наступні дані:

– Тема проекту.

– Розробник проекту.

– Керівник проекту.

– Місце виконання проекту.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем центру генерації та розподілу ключів системи захисту інформації.

– Досліджена система центру генерації та розподілу ключів системи захисту інформації.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання центру генерації та розподілу ключів системи захисту інформації.

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 10.4 Sydney. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи кібербезпеки центру генерації та розподілу ключів системи захисту

інформації. 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 кібербезпеки й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм SHA-3.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022, pp. 1-12. **(Scopus).**

2. Smirnov O., Smirnova T., Anas M. Al-Oraiqat, Drieiev O., Polishchuk L., Sheroz Khan, Yassin M. Y. Hasan, Aladdein M. Amro, Hazim S. AlRawashdeh «Method for Determining Treated Metal Surface Quality Using Computer Vision Technology». *Sensors (Basel, Switzerland)* Volume 22, Issue 16, 6223, 2022. (**Scopus**).

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

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

5. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». *SN Computer Science,* Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w **(Scopus).**

6. Smirnov O., Kovalenko O., Kovalenko A., Kavun S. «Quantitative Risk Assessment Method Development in the Context of the SDLC-model». *2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T),* 2021, pp. 203-208, doi: 10.1109/PICST54195.2021.9772143 **(Scopus).**

7. Smirnov O., Neskorodieva T., Fedorov E., Rymar P. «Neural Network Modeling Method of Transformations Data of Audit Production with Returnable Waste». *CEUR Workshop Proceedings* Volume 3101, 2021, Pages 192-207. **(Scopus).**

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

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

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

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

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

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

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

15. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. **Springer**, Cham. 2021. pp 557-587. **(Scopus)**.

16. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136. **(Scopus)**.

17. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 366-379. **(Scopus)**.

18. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». *International Journal of Computer Network and Information Security (IJCNIS)*. Vol. 12, No. 3, 2020. PP.33-43. **(Scopus)**.

19. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 633-645. **(Scopus)**.

20. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 646-660., **(Scopus)**.

21. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407. (**Scopus)**.

22. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, Vol 2588, P. 215-227, 2019. **(Scopus)**.

23. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019. **(Scopus)**.

24. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629. **(Scopus)**.

25. Smirnov, O., Kuznetsov, A., Kiian, A., Kuznetsova, K., Ivko, T., Prokopovych-Tkachenko, D., «Soft Decoding Based on Ordered Subsets of Verification Equations of Turbo-Productive Codes», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 873-884. **(Scopus)**.

26. Smirnov, O., Kuznetsov, A., Prokopovych-Tkachenko, D. «Hiding Data in Images Using a Pseudo-Random Sequence». *ISCI'2020: Information Security in Critical Infrastructures.* ***Collective monograph***. Edited by Ivan D. Gorbenko, Victor A. Krasnobayev and Alexandr A. Kuznetsov. ASC Academic Publishing, USA, 2020. pp. 46-59. – ISBN: 978-1-7362833-0-1 (Hardback), ISBN: 978-1-7362833-1-8 (Ebook).

27. Smirnov, O., Kuznetsov, A., Shekhanin, K., Chepurko, I. Detecting Hidden Information in FAT. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures**. Collective monograph**. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 412-429. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

28. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. **Collective monograph**. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

29. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

30. Смірнов О.А., Дрєєва Г.М., «Метод генерування фрактального трафіку за допомогою моделі генератора на графі» у Інформаційна безпека та інформаційні технології: монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139.

31. Смирнов А.А., Коваленко А.В. Комплекс математических моделей технологии тестирования WEB-приложений. Інформаційні технології: сучасний стан та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: ТОВ «ДІСА ПЛЮС», 2018. – 461 с.

32. Смирнов А.А., Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения. Інформаційні технології: проблеми та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: Видавець Рожко С.Г., 2017. – 447 с.

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

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

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

36. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного

процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

37. Смирнов А., Кузнецов А., Кузнецова Т. «Шумоподобные дискретные сигналы для асинхронных систем кодового разделения радиоканалов». *Радиотехника*, № 2(205), 175–183. 2021.

38. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New Technique for Hiding Data in Cover Images Using Adaptively Generated Pseudorandom Sequences». *CEUR Workshop Proceedings* Volume 2732, 2020, Pages 214-227.

39. Смірнов, О.А., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю.Усік П.С., «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв'язку». *Проблеми телекомунікацій*. № 1(26). С. 83-96. 2020.

40. Смирнов А.А, Кузнецов А.А., Киян А.С., Кузнецова Е.А. «Сокрытие данных на основе адресации шумоподобных сигналов». *Всеукраїнський міжвідомчий науково-технічний збірник "Радіотехніка"* – Харьків: ХНУРЕ. – 2020. – Вип. 203. – С. 38-49.

41. Смирнов А.А., Дудан А.В., Смирнова Т.В. «Формализация структуры технологического процесса электродугового напыления». *Сборник научных трудов «Актуальные вопросы машиноведения».* Объединенный институт машиностроения Национальной Академии Наук Беларуси. №9. С. 308-312, 2020.

42. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» *Вісник Черкаського державного технологічного університету. Технічні науки.* №4. С. 103-110. 2020.

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

44. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». *Центральноукраїнський науковий вісник. Технічні науки.* № 2(33). с. 161-172, 2019.

45. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., «Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей». Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

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

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

48. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

49. Смирнов А.А., Лысенко И.А., Информационная технология проектирования тестовых наборов на основе требований к программному обеспечению, Системи управління, навігації та зв'язку. – Випуск 4 (44). – Полтава: ПолтНТУ. – 2017. – С. 112-115.

50. Смірнов О.А., Мелешко Є.В., Хох В.Д., Дослідження методів аудиту систем управління інформаційною безпекою, Системи управління, навігації та зв'язку. – Випуск 1 (41). – Полтава: ПолтНТУ. – 2017. – С. 38-42.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 13-02 від 5.01.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Delphi 10.4 Sydney.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 90 аркушів.

8 Етапи розробки

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

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

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

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

11.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 6.06.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Смірнов С.А.

Програмне забезпечення системи кібербезпеки центру генерації та розподілу ключів системи захисту інформації

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 65

Літера: РП

Кропивницький – 2023 року

Файл Cipher.pas – Генерація та розподіл ключів для криптографічних алгоритмів

```
unit Cipher;

interface

{$I VER.INC}

uses SysUtils, Classes, DECUtil, Hash;

const {ErrorCode's for ECipherException}
  errGeneric        = 0;  {Помилка генерації}
  errInvalidKey     = 1;  {Ключ декодування некоректний}
  errInvalidKeySize = 2;  {Розмір ключа дуже великий}
  errNotInitialized = 3;  {Методи Init() або InitKey() не викликаються}
  errInvalidMACMode = 4;  {CalcMAC не використовує cmECB, cmOFB}
  errCantCalc       = 5;

type
  ECipherException = class(Exception)
  public
    ErrorCode: Integer;
  end;

{Усі класи шифрування у данній бібліотеці мають добрі параметри}
  TCipher_Gost         = class;
  TCipher_Blowfish     = class;
  TCipher_IDEA         = class;
  TCipher_SAFER        = class;
  TCipher_SAFER_K40    = class;
  TCipher_SAFER_SK40   = class;
  TCipher_SAFER_K64    = class;
  TCipher_SAFER_SK64   = class;
  TCipher_SAFER_K128   = class;
  TCipher_SAFER_SK128  = class;
  TCipher_TEA          = class;
  TCipher_TEAN         = class;
  TCipher_SCOP         = class;  {Потоковий шифр}
  TCipher_Q128         = class;
  TCipher_3Way         = class;
  TCipher_Twofish      = class;
  TCipher_Shark        = class;
  TCipher_Square       = class;

  TCipherMode = (cmCTS, cmCBC, cmCFB, cmOFB, cmECB, cmCTSMAC, cmCBCMAC,
cmCFBMAC);
{ режими шифрування:
  cmCTS
  cmCBC     Cipher Block Chaining
  cmCFB     K-bit Cipher Feedback
  cmOFB     K-bit Output Feedback
  cmECB *   Electronic Codebook

  cmCTSMAC  Build a Message Authentication Code у cmCTS режимі
  cmCBCMAC  Build a CBC-MAC
  cmCFBMAC  Build a CFB-MAC
}

  TCipherClass = class of TCipher;

  TCipher = class(TProtection)
  private
    FMode: TCipherMode;
    FHash: THash;
    FHashClass: THashClass;
    FKeySize: Integer;
```

```
    FBufSize: Integer;
    FUserSize: Integer;
    FBuffer: Pointer;
    FVector: Pointer;
    FFeedback: Pointer;
    FUser: Pointer;
    FFlags: Integer;
    function GetHash: THash;
    procedure SetHashClass(Value: THashClass);
    procedure InternalCodeStream(Source, Dest: TStream; DataSize: Integer;
Encode: Boolean);
    procedure InternalCodeFile(const Source, Dest: String; Encode: Boolean);
  protected
    function GetFlag(Index: Integer): Boolean;
    procedure SetFlag(Index: Integer; Value: Boolean); virtual;
{використовуються в методі Init()}
    procedure InitBegin(var Size: Integer);
    procedure InitEnd(IVector: Pointer); virtual;
{повинно анулюватися}
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
virtual;
    class function TestVector: Pointer; virtual;
{анулює TProtection методи}
    procedure CodeInit(Action: TPAction); override;
    procedure CodeDone(Action: TPAction); override;
    procedure CodeBuf(var Buffer; const BufferSize: Integer; Action: TPAction);
override;
{функція шифрування , повинно анулюватися}
    procedure Encode(Data: Pointer); virtual;
{ функція дешифрування, повинно анулюватися}
    procedure Decode(Data: Pointer); virtual;
{особисті дані користувача й буфер}
    property User: Pointer read FUser;
    property Buffer: Pointer read FBuffer;
    property UserSize: Integer read FUserSize;
  public
    constructor Create(const Password: String; AProtection: TProtection);
    destructor Destroy; override;
    class function MaxKeySize: Integer;
{простий тест на коректну роботу}
    class function SelfTest: Boolean;
{ініціалізує вікно шифрування}
    procedure Init(const Key; Size: Integer; IVector: Pointer); virtual;
    procedure InitKey(const Key: String; IVector: Pointer);
{сбрасує Feedbackregister с IVector}
    procedure Done; virtual;
{захищає таємні Data's, Feedback, Buffer, Vector etc.}
    procedure Protect; virtual;

    procedure EncodeBuffer(const Source; var Dest; DataSize: Integer);
    procedure DecodeBuffer(const Source; var Dest; DataSize: Integer);
    function  EncodeString(const Source: String): String;
    function  DecodeString(const Source: String): String;
    procedure EncodeFile(const Source, Dest: String);
    procedure DecodeFile(const Source, Dest: String);
    procedure EncodeStream(const Source, Dest: TStream; DataSize: Integer);
    procedure DecodeStream(const Source, Dest: TStream; DataSize: Integer);

{розраховуєа MAC, Message Authentication Code, використовується в
 cmCBCMAC, cmCTSMAC, cmCFBMAC Modes
 cmCBC, cmCTS, cmCFB Modes }
    function CalcMAC(Format: Integer): String;

{Режим шифрування = cmXXX}
    property Mode: TCipherMode read FMode write FMode;
{поточний Hash-Object, з InitKey()}
    property Hash: THash read GetHash;
{Клас Hash-Object}
    property HashClass: THashClass read FHashClass write SetHashClass;
```

```
{максимальний KeySize та BufSize (Size of Feedback, Buffer та Vector}
    property KeySize: Integer read FKeySize;
    property BufSize: Integer read FBufSize;

{Init() викликається}
    property Initialized: Boolean index 1 read GetFlag write SetFlag;
{актуальний IVector, BufSize Bytes long}
    property Vector: Pointer read FVector;
{ Feedback register, BufSize Bytes long}
    property Feedback: Pointer read FFeedback;
{ Key у функції InitKey }
    property HasHashKey: Boolean index 0 read GetFlag;
  end;

// Опис алгоритмів шифрування

// Алгоритм ГОСТ

  TCipher_Gost = class(TCipher) {russian Cipher}
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
    procedure Encode(Data: Pointer); override;
    procedure Decode(Data: Pointer); override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Алгоритм Blowfish

  TCipher_Blowfish = class(TCipher)
  private
{$IFDEF UseASM}
  {$IFNDEF 486GE}  // не використовується для <= CPU 386
    procedure Encode386(Data: Pointer);
    procedure Decode386(Data: Pointer);
  {$ENDIF}
{$ENDIF}
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
    procedure Encode(Data: Pointer); override;
    procedure Decode(Data: Pointer); override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Алгоритм IDEA

  TCipher_IDEA = class(TCipher) {International Data Encryption Algorithm }
  private
    procedure Cipher(Data, Key: PWordArray);
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
    procedure Encode(Data: Pointer); override;
    procedure Decode(Data: Pointer); override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Вибір режиму алгоритму SAFER

  TSAFERMode = (smDefault, smK40, smK64, smK128, smStrong, smSK40, smSK64,
smSK128);
{smDefault                  Режим побудови KeyLength "Size"
```

```
                          якщо Size <=  5 тоді smK40 використовується
                          якщо Size <=  8 тоді smK64 використовується
                          якщо Size <= 16 тоді smK128 використовується
 smK40       SAFER K-40   Keysize рівний 40bit  ->  5 Byte
 smK64       SAFER K-64   Keysize рівний 64bit  ->  8 Byte
 smK128      SAFER K-128  KeySize рівний 128bit -> 16 Byte
 smStrong                 Режим побудови KeyLength "Size" зафіксований як
smDefault,
                          якщо Size <=  5 тоді smSK40 використовується
                          якщо Size <=  8 тоді smSK64 використовується
                          якщо Size <= 16 тоді smSK128 використовується
                          це Defaultmode для TCipher_SAFER
 smSK40      SAFER SK-40  зафіксовано в версії для K-40 краще з Keyscheduling
 smSK64      SAFER SK-64  зафіксовано в версії для K-64 краще з Keyscheduling
 smSK128     SAFER SK-128 зафіксовано в версії для K-128 краще з Keyscheduling}

// Алгоритм SAFER

  TCipher_SAFER = class(TCipher) {SAFER = Secure And Fast Encryption Routine}
  private
    FRounds: Integer;
    FSAFERMode: TSAFERMode;
    procedure SetRounds(Value: Integer);
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
    procedure Encode(Data: Pointer); override;
    procedure Decode(Data: Pointer); override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
    procedure InitNew(const Key; Size: Integer; IVector: Pointer; SAFERMode:
TSAFERMode);
    property Rounds: Integer read FRounds write SetRounds;
  end;

// Алгоритм SAFER_K40

  TCipher_SAFER_K40 = class(TCipher_SAFER)
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Алгоритм SAFER_SK40

  TCipher_SAFER_SK40 = class(TCipher_SAFER_K40)
  protected
    class function TestVector: Pointer; override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Алгоритм SAFER_K64

  TCipher_SAFER_K64 = class(TCipher_SAFER)
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Алгоритм SAFER_SK64
```

```
  TCipher_SAFER_SK64 = class(TCipher_SAFER_K64)
  protected
    class function TestVector: Pointer; override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Алгоритм SAFER_K128

  TCipher_SAFER_K128 = class(TCipher_SAFER)
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Алгоритм SAFER_SK128

  TCipher_SAFER_SK128 = class(TCipher_SAFER_K128)
  protected
    class function TestVector: Pointer; override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Алгоритм TEA

  TCipher_TEA = class(TCipher) {Tiny Encryption Algorithm}
  private
    FRounds: Integer; {16 - 32, за замовчуванням 16 }
    procedure SetRounds(Value: Integer);
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
    procedure Encode(Data: Pointer); override;
    procedure Decode(Data: Pointer); override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
    property Rounds: Integer read FRounds write SetRounds;
  end;

// Алгоритм TEAN

  TCipher_TEAN = class(TCipher_TEA) {Tiny Encryption Algorithm, розширена версія
}
  protected
    class function TestVector: Pointer; override;
    procedure Encode(Data: Pointer); override;
    procedure Decode(Data: Pointer); override;
  end;

// Алгоритм SCOP

  TCipher_SCOP = class(TCipher) {Потоковий шифр  в блочному режимі}
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
    procedure Encode(Data: Pointer); override;
    procedure Decode(Data: Pointer); override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
    procedure Done; override;
  end;

// Алгоритм Q128
```

```
  TCipher_Q128 = class(TCipher)
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
    procedure Encode(Data: Pointer); override;
    procedure Decode(Data: Pointer); override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Алгоритм 3Way

  TCipher_3Way = class(TCipher)
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
    procedure Encode(Data: Pointer); override;
    procedure Decode(Data: Pointer); override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Алгоритм Twofish

  TCipher_Twofish = class(TCipher)
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
    procedure Encode(Data: Pointer); override;
    procedure Decode(Data: Pointer); override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Алгоритм Shark

  TCipher_Shark = class(TCipher)
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
    procedure Encode(Data: Pointer); override;
    procedure Decode(Data: Pointer); override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

// Алгоритм Square

  TCipher_Square = class(TCipher)
  protected
    class procedure GetContext(var ABufSize, AKeySize, AUserSize: Integer);
override;
    class function TestVector: Pointer; override;
    procedure Encode(Data: Pointer); override;
    procedure Decode(Data: Pointer); override;
  public
    procedure Init(const Key; Size: Integer; IVector: Pointer); override;
  end;

function DefaultCipherClass: TCipherClass;
procedure SetDefaultCipherClass(CipherClass: TCipherClass);
procedure RaiseCipherException(const ErrorCode: Integer; const Msg: String);
function RegisterCipher(const ACipher: TCipherClass; const AName, ADescription:
String): Boolean;
```

```
function UnregisterCipher(const ACipher: TCipherClass): Boolean;
function CipherList: TStrings;
procedure CipherNames(List: TStrings);
function GetCipherClass(const Name: String): TCipherClass;
function GetCipherName(CipherClass: TCipherClass): String;

const
  CheckCipherKeySize: Boolean = False;
{встановлюємо в True raises Exception коли Розмір ключа дуже великий, (Method
Init())
 Та обрізаємо Key, коли False}

implementation

uses DECConst, Windows;

{$I *.inc}
{$I Square.inc}

const
  FDefaultCipherClass : TCipherClass = TCipher_Blowfish;
  FCipherList         : TStringList  = nil;

function DefaultCipherClass: TCipherClass;
begin
  Result := FDefaultCipherClass;
end;

procedure SetDefaultCipherClass(CipherClass: TCipherClass);
begin
  if CipherClass = nil then FDefaultCipherClass := TCipher_Blowfish
    else FDefaultCipherClass := CipherClass;
end;

procedure RaiseCipherException(const ErrorCode: Integer; const Msg: String);
var
  E: ECipherException;
begin
  E := ECipherException.Create(Msg);
  E.ErrorCode := ErrorCode;
  raise E;
end;

function RegisterCipher(const ACipher: TCipherClass; const AName, ADescription:
String): Boolean;
var
  I: Integer;
  S: String;
begin
  Result := False;
  if ACipher = nil then Exit;
  S := Trim(AName);
  if S = '' then
  begin
    S := ACipher.ClassName;
    if S[1] = 'T' then Delete(S, 1, 1);
    I := Pos('_', S);
    if I > 0 then Delete(S, 1, I);
  end;
  S := S + '=' + ADescription;
  I := CipherList.IndexOfObject(Pointer(ACipher));
  if I < 0 then CipherList.AddObject(S, Pointer(ACipher))
    else CipherList[I] := S;
  Result := True;
end;

function UnregisterCipher(const ACipher: TCipherClass): Boolean;
var
  I: Integer;
```

```
begin
  Result := False;
  repeat
    I := CipherList.IndexOfObject(Pointer(ACipher));
    if I < 0 then Break;
    Result := True;
    CipherList.Delete(I);
  until False;
end;

function CipherList: TStrings;
begin
  if not IsObject(FCipherList, TStringList) then FCipherList :=
TStringList.Create;
  Result := FCipherList;
end;

procedure CipherNames(List: TStrings);
var
  I: Integer;
begin
  if not IsObject(List, TStrings) then Exit;
  for I := 0 to CipherList.Count-1 do
    List.AddObject(FCipherList.Names[I], FCipherList.Objects[I]);
end;

function GetCipherClass(const Name: String): TCipherClass;
var
  I: Integer;
  N: String;
begin
  Result := nil;
  N := Name;
  I := Pos('_', N);
  if I > 0 then Delete(N, 1, I);
  for I := 0 to CipherList.Count-1 do
    if AnsiCompareText(N, GetShortClassName(TClass(FCipherList.Objects[I]))) = 0
then
    begin
      Result := TCipherClass(FCipherList.Objects[I]);
      Exit;
    end;
  I := FCipherList.IndexOfName(N);
  if I >= 0 then Result := TCipherClass(FCipherList.Objects[I]);
end;

function GetCipherName(CipherClass: TCipherClass): String;
var
  I: Integer;
begin
  I := CipherList.IndexOfObject(Pointer(CipherClass));
  if I >= 0 then Result := FCipherList.Names[I]
    else Result := GetShortClassName(CipherClass);
end;

function TCipher.GetFlag(Index: Integer): Boolean;
begin
  Result := FFlags and (1 shl Index) <> 0;
end;

procedure TCipher.SetFlag(Index: Integer; Value: Boolean);
begin
  Index := 1 shl Index;
  if Value then FFlags := FFlags or Index
    else FFlags := FFlags and not Index;
end;

procedure TCipher.InitBegin(var Size: Integer);
begin
```

```
  Initialized := False;
  Protect;
  if Size < 0 then Size := 0;
  if Size > KeySize then
    if not CheckCipherKeySize then Size := KeySize
      else RaiseCipherException(errInvalidKeySize, Format(sInvalidKeySize,
[ClassName, 0, KeySize]));
end;

procedure TCipher.InitEnd(IVector: Pointer);
begin
  if IVector = nil then Encode(Vector)
    else Move(IVector^, Vector^, BufSize);
  Move(Vector^, Feedback^, BufSize);
  Initialized := True;
end;

class procedure TCipher.GetContext(var ABufSize, AKeySize, AUserSize: Integer);
begin
  ABufSize := 0;
  AKeySize := 0;
  AUserSize := 0;
end;

class function TCipher.TestVector: Pointer;
begin
  Result := GetTestVector;
end;

procedure TCipher.Encode(Data: Pointer);
begin
end;

procedure TCipher.Decode(Data: Pointer);
begin
end;

constructor TCipher.Create(const Password: String; AProtection: TProtection);
begin
  inherited Create(AProtection);
  FHashClass := DefaultHashClass;
  GetContext(FBufSize, FKeySize, FUserSize);
  GetMem(FVector, FBufSize);
  GetMem(FFeedback, FBufSize);
  GetMem(FBuffer, FBufSize);
  GetMem(FUser, FUserSize);
  Protect;
  if Password <> '' then InitKey(Password, nil);
end;

destructor TCipher.Destroy;
begin
  Protect;
  ReallocMem(FVector, 0);
  ReallocMem(FFeedback, 0);
  ReallocMem(FBuffer, 0);
  ReallocMem(FUser, 0);
  FHash.Release;
  FHash := nil;
  inherited Destroy;
end;

class function TCipher.MaxKeySize: Integer;
var
  Dummy: Integer;
begin
  GetContext(Dummy, Result, Dummy);
end;
```

```
class function TCipher.SelfTest: Boolean;
var
  Data: array[0..63] of Char;
  Key: String;
  SaveKeyCheck: Boolean;
begin
  Result       := InitTestIsOk; {маємо модифікацію testvectors ?}
{ми повинні використовувати ClassName as Key }
  Key          := ClassName;
  SaveKeyCheck := CheckCipherKeySize;
  with Self.Create('', nil) do
  try
    CheckCipherKeySize := False;
    Mode := cmCTS;
    Init(PChar(Key)^, Length(Key), nil);
    EncodeBuffer(GetTestVector^, Data, 32);
    Result := Result and (MemCompare(TestVector, @Data, 32) = 0);
    Done;
    DecodeBuffer(Data, Data, 32);
    Result := Result and (MemCompare(GetTestVector, @Data, 32) = 0);
  finally
    CheckCipherKeySize := SaveKeyCheck;
    Free;
  end;
  FillChar(Data, SizeOf(Data), 0);
end;

procedure TCipher.Init(const Key; Size: Integer; IVector: Pointer);
begin
end;

procedure TCipher.InitKey(const Key: String; IVector: Pointer);
var
  I: Integer;
begin
  Hash.Init;
  Hash.Calc(PChar(Key)^, Length(Key));
  Hash.Done;
  I := Hash.DigestKeySize;
  if I > FKeySize then I := FKeySize; {Обрізаємо великий Keys}
  Init(Hash.DigestKey^, I, IVector);
  EncodeBuffer(Hash.DigestKey^, Hash.DigestKey^, Hash.DigestKeySize);
  Done;
  SetFlag(0, True);
end;

procedure TCipher.Done;
begin
  if MemCompare(FVector, FFeedback, FBufSize) = 0 then Exit;
  Move(FFeedback^, FBuffer^, FBufSize);
  Move(FVector^, FFeedback^, FBufSize);
end;

procedure TCipher.Protect;
begin
  SetFlag(0, False);
  Initialized := False;
  FillChar(FVector^, FBufSize, $AA);
  FillChar(FFeedback^, FBufSize, $AA);
  FillChar(FBuffer^, FBufSize, $AA);
  FillChar(FUser^, FUserSize, $AA);

  FillChar(FVector^, FBufSize, $55);
  FillChar(FFeedback^, FBufSize, $55);
  FillChar(FBuffer^, FBufSize, $55);
  FillChar(FUser^, FUserSize, $55);

  FillChar(FVector^, FBufSize, $FF);
  FillChar(FFeedback^, FBufSize, $FF);
```

```
  FillChar(FBuffer^, FBufSize, 0);
  FillChar(FUser^, FUserSize, 0);
end;

function TCipher.GetHash: THash;
begin
  if not IsObject(FHash, THash) then
  begin
    if FHashClass = nil then FHashClass := DefaultHashClass;
    FHash := FHashClass.Create(nil);
    FHash.AddRef;
  end;
  Result := FHash;
end;

procedure TCipher.SetHashClass(Value: THashClass);
begin
  if Value <> FHashClass then
  begin
    FHash.Release;
    FHash := nil;
    FHashClass := Value;
    if FHashClass = nil then FHashClass := DefaultHashClass;
  end;
end;

procedure TCipher.InternalCodeStream(Source, Dest: TStream; DataSize: Integer;
Encode: Boolean);
const
  maxBufSize = 1024 * 4;
var
  Buf: PChar;
  SPos: Integer;
  DPos: Integer;
  Len: Integer;
  Proc: procedure(const Source; var Dest; DataSize: Integer) of object;
  Size: Integer;
begin
  if Source = nil then Exit;
  if Encode or (Mode in [cmCBCMAC, cmCTSMAC, cmCFBMAC]) then Proc :=
EncodeBuffer
    else Proc := DecodeBuffer;
  if Dest = nil then Dest := Source;
  if DataSize < 0 then
  begin
    DataSize := Source.Size;
    Source.Position := 0;
  end;
  Buf := nil;
  Size := DataSize;
  DoProgress(Self, 0, Size);
  try
    Buf    := AllocMem(maxBufSize);
    DPos   := Dest.Position;
    SPos   := Source.Position;
    if Mode in [cmCTSMAC, cmCBCMAC, cmCFBMAC] then
    begin
      while DataSize > 0 do
      begin
        Len := DataSize;
        if Len > maxBufSize then Len := maxBufSize;
        Len := Source.Read(Buf^, Len);
        if Len <= 0 then Break;
        Proc(Buf^, Buf^, Len);
        Dec(DataSize, Len);
        DoProgress(Self, Size - DataSize, Size);
      end;
    end else
      while DataSize > 0 do
```

```
      begin
        Source.Position := SPos;
        Len := DataSize;
        if Len > maxBufSize then Len := maxBufSize;
        Len := Source.Read(Buf^, Len);
        SPos := Source.Position;
        if Len <= 0 then Break;
        Proc(Buf^, Buf^, Len);
        Dest.Position := DPos;
        Dest.Write(Buf^, Len);
        DPos := Dest.Position;
        Dec(DataSize, Len);
        DoProgress(Self, Size - DataSize, Size);
      end;
  finally
    DoProgress(Self, 0, 0);
    ReallocMem(Buf, 0);
  end;
end;

procedure TCipher.InternalCodeFile(const Source, Dest: String; Encode: Boolean);
var
  S,D: TFileStream;
begin
  S := nil;
  D := nil;
  try
    if Mode in [cmCBCMAC, cmCTSMAC, cmCFBMAC] then
    begin
      S := TFileStream.Create(Source, fmOpenRead or fmShareDenyNone);
      D := S;
    end else
      if (AnsiCompareText(Source, Dest) <> 0) and (Trim(Dest) <> '') then
      begin
        S := TFileStream.Create(Source, fmOpenRead or fmShareDenyNone);
        D := TFileStream.Create(Dest, fmCreate);
      end else
      begin
        S := TFileStream.Create(Source, fmOpenReadWrite);
        D := S;
      end;
    InternalCodeStream(S, D, -1, Encode);
  finally
    S.Free;
    if S <> D then
    begin
{$IFDEF VER_D3H}
      D.Size := D.Position;
{$ENDIF}
      D.Free;
    end;
  end;
end;

procedure TCipher.EncodeStream(const Source, Dest: TStream; DataSize: Integer);
begin
  InternalCodeStream(Source, Dest, DataSize, True);
end;

procedure TCipher.DecodeStream(const Source, Dest: TStream; DataSize: Integer);
begin
  InternalCodeStream(Source, Dest, DataSize, False);
end;

procedure TCipher.EncodeFile(const Source, Dest: String);
begin
  InternalCodeFile(Source, Dest, True);
end;
```

```
procedure TCipher.DecodeFile(const Source, Dest: String);
begin
  InternalCodeFile(Source, Dest, False);
end;

function TCipher.EncodeString(const Source: String): String;
begin
  SetLength(Result, Length(Source));
  EncodeBuffer(PChar(Source)^, PChar(Result)^, Length(Source));
  if Mode in [cmCBCMAC, cmCTSMAC, cmCFBMAC] then Result := '';
end;

function TCipher.DecodeString(const Source: String): String;
begin
  SetLength(Result, Length(Source));
  DecodeBuffer(PChar(Source)^, PChar(Result)^, Length(Source));
  if Mode in [cmCBCMAC, cmCTSMAC, cmCFBMAC] then Result := '';
end;

procedure TCipher.EncodeBuffer(const Source; var Dest; DataSize: Integer);
var
  S,D,F: PByte;
begin
  if not Initialized then
    RaiseCipherException(errNotInitialized, Format(sNotInitialized,
[ClassName]));
  S := @Source;
  D := @Dest;
  case FMode of
    cmECB:
      begin
        if S <> D then Move(S^, D^, DataSize);
        while DataSize >= FBufSize do
        begin
          Encode(D);
          Inc(D, FBufSize);
          Dec(DataSize, FBufSize);
        end;
        if DataSize > 0 then
        begin
          Move(D^, FBuffer^, DataSize);
          Encode(FBuffer);
          Move(FBuffer^, D^, DataSize);
        end;
      end;
    cmCTS:
      begin
        while DataSize >= FBufSize do
        begin
          XORBuffers(S, FFeedback, FBufSize, D);
          Encode(D);
          XORBuffers(D, FFeedback, FBufSize, FFeedback);
          Inc(S, FBufSize);
          Inc(D, FBufSize);
          Dec(DataSize, FBufSize);
        end;
        if DataSize > 0 then
        begin
          Move(FFeedback^, FBuffer^, FBufSize);
          Encode(FBuffer);
          XORBuffers(S, FBuffer, DataSize, D);
          XORBuffers(FBuffer, FFeedback, FBufSize, FFeedback);
        end;
      end;
    cmCBC:
      begin
        F := FFeedback;
        while DataSize >= FBufSize do
        begin
```

```
      XORBuffers(S, F, FBufSize, D);
      Encode(D);
      F := D;
      Inc(S, FBufSize);
      Inc(D, FBufSize);
      Dec(DataSize, FBufSize);
    end;
    Move(F^, FFeedback^, FBufSize);
    if DataSize > 0 then
    begin
      Move(FFeedback^, FBuffer^, FBufSize);
      Encode(FBuffer);
      XORBuffers(S, FBuffer, DataSize, D);
      XORBuffers(FBuffer, FFeedback, FBufSize, FFeedback);
    end;
  end;
cmCFB:
  while DataSize > 0 do
  begin
    Move(FFeedback^, FBuffer^, FBufSize);
    Encode(FBuffer);
    D^ := S^ xor PByte(FBuffer)^;
    Move(PByteArray(FFeedback)[1], FFeedback^, FBufSize-1);
    PByteArray(FFeedback)[FBufSize-1] := D^;
    Inc(D);
    Inc(S);
    Dec(DataSize);
  end;
cmOFB:
  while DataSize > 0 do
  begin
    Move(FFeedback^, FBuffer^, FBufSize);
    Encode(FBuffer);
    D^ := S^ xor PByte(FBuffer)^;
    Move(PByteArray(FFeedback)[1], FFeedback^, FBufSize-1);
    PByteArray(FFeedback)[FBufSize-1] := PByte(FBuffer)^;
    Inc(D);
    Inc(S);
    Dec(DataSize);
  end;
cmCTSMAC:
  begin
    while DataSize >= FBufSize do
    begin
      XORBuffers(S, FFeedback, FBufSize, FBuffer);
      Encode(FBuffer);
      XORBuffers(FBuffer, FFeedback, FBufSize, FFeedback);
      Inc(S, FBufSize);
      Dec(DataSize, FBufSize);
    end;
    if DataSize > 0 then
    begin
      Move(FFeedback^, FBuffer^, FBufSize);
      Encode(FBuffer);
      XORBuffers(FBuffer, FFeedback, FBufSize, FFeedback);
    end;
  end;
cmCBCMAC:
  begin
    while DataSize >= FBufSize do
    begin
      XORBuffers(S, FFeedback, FBufSize, FBuffer);
      Encode(FBuffer);
      Move(FBuffer^, FFeedback^, FBufSize);
      Inc(S, FBufSize);
      Dec(DataSize, FBufSize);
    end;
    if DataSize > 0 then
    begin
```

```
          Move(FFeedback^, FBuffer^, FBufSize);
          Encode(FBuffer);
          XORBuffers(FBuffer, FFeedback, FBufSize, FFeedback);
        end;
      end;
    cmCFBMAC:
      while DataSize > 0 do
      begin
        Move(FFeedback^, FBuffer^, FBufSize);
        Encode(FBuffer);
        Move(PByteArray(FFeedback)[1], FFeedback^, FBufSize-1);
        PByteArray(FFeedback)[FBufSize-1] := S^ xor PByte(FBuffer)^;
        Inc(S);
        Dec(DataSize);
      end;
  end;
end;

procedure TCipher.DecodeBuffer(const Source; var Dest; DataSize: Integer);
var
  S,D,F,B: PByte;
begin
  if not Initialized then
    RaiseCipherException(errNotInitialized, Format(sNotInitialized,
[ClassName]));
  S := @Source;
  D := @Dest;
  case FMode of
    cmECB:
      begin
        if S <> D then Move(S^, D^, DataSize);
        while DataSize >= FBufSize do
        begin
          Decode(D);
          Inc(D, FBufSize);
          Dec(DataSize, FBufSize);
        end;
        if DataSize > 0 then
        begin
          Move(D^, FBuffer^, DataSize);
          Encode(FBuffer);
          Move(FBuffer^, D^, DataSize);
        end;
      end;
    cmCTS:
      begin
        if S <> D then Move(S^, D^, DataSize);
        F := FFeedback;
        B := FBuffer;
        while DataSize >= FBufSize do
        begin
          XORBuffers(D, F, FBufSize, B);
          Decode(D);
          XORBuffers(D, F, FBufSize, D);
          S := B;
          B := F;
          F := S;
          Inc(D, FBufSize);
          Dec(DataSize, FBufSize);
        end;
        if F <> FFeedback then Move(F^, FFeedback^, FBufSize);
        if DataSize > 0 then
        begin
          Move(FFeedback^, FBuffer^, FBufSize);
          Encode(FBuffer);
          XORBuffers(FBuffer, D, DataSize, D);
          XORBuffers(FBuffer, FFeedback, FBufSize, FFeedback);
        end;
      end;
```

```
    cmCBC:
      begin
        if S <> D then Move(S^, D^, DataSize);
        F := FFeedback;
        B := FBuffer;
        while DataSize >= FBufSize do
        begin
          Move(D^, B^, FBufSize);
          Decode(D);
          XORBuffers(F, D, FBufSize, D);
          S := B;
          B := F;
          F := S;
          Inc(D, FBufSize);
          Dec(DataSize, FBufSize);
        end;
        if F <> FFeedback then Move(F^, FFeedback^, FBufSize);
        if DataSize > 0 then
        begin
          Move(FFeedback^, FBuffer^, FBufSize);
          Encode(FBuffer);
          XORBuffers(D, FBuffer, DataSize, D);
          XORBuffers(FBuffer, FFeedback, FBufSize, FFeedback);
        end;
      end;
    cmCFB:
      while DataSize > 0 do
      begin
        Move(FFeedback^, FBuffer^, FBufSize);
        Encode(FBuffer);
        Move(PByteArray(FFeedback)[1], FFeedback^, FBufSize-1);
        PByteArray(FFeedback)[FBufSize-1] := S^;
        D^ := S^ xor PByte(FBuffer)^;
        Inc(D);
        Inc(S);
        Dec(DataSize);
      end;
    cmOFB:
      while DataSize > 0 do
      begin
        Move(FFeedback^, FBuffer^, FBufSize);
        Encode(FBuffer);
        D^ := S^ xor PByte(FBuffer)^;
        Move(PByteArray(FFeedback)[1], FFeedback^, FBufSize-1);
        PByteArray(FFeedback)[FBufSize-1] := PByte(FBuffer)^;
        Inc(D);
        Inc(S);
        Dec(DataSize);
      end;
    cmCTSMAC, cmCBCMAC, cmCFBMAC:
      begin
        EncodeBuffer(Source, Dest, DataSize);
        Exit;
      end;
  end;
end;

procedure TCipher.CodeInit(Action: TPAction);
begin
  if not Initialized then
    RaiseCipherException(errNotInitialized, Format(sNotInitialized,
[ClassName]));
{  if (Mode in [cmCBCMAC, cmCTSMAC, cmCFBMAC]) <> (Action = paCalc) then
    RaiseCipherException(errCantCalc, Format(sCantCalc, [ClassName]));}
  if Action <> paCalc then
    if Action <> paWipe then Done
      else RndXORBuffer(RndTimeSeed, FFeedback^, FBufSize);
  inherited CodeInit(Action);
end;
```

```
procedure TCipher.CodeDone(Action: TPAction);
begin
  inherited CodeDone(Action);
  if Action <> paCalc then
    if Action <> paWipe then Done
      else RndXORBuffer(RndTimeSeed, FFeedback^, FBufSize);
end;

procedure TCipher.CodeBuf(var Buffer; const BufferSize: Integer; Action:
TPAction);
begin
  if Action = paDecode then
  begin
    if Action in Actions then
      DecodeBuffer(Buffer, Buffer, BufferSize);
    inherited CodeBuf(Buffer, BufferSize, Action);
  end else
  begin
    inherited CodeBuf(Buffer, BufferSize, Action);
    if Action in Actions then
      EncodeBuffer(Buffer, Buffer, BufferSize);
  end;
end;

function TCipher.CalcMAC(Format: Integer): String;
var
  B: PByteArray;
begin
  if Mode in [cmECB, cmOFB] then
    RaiseCipherException(errInvalidMACMode, sInvalidMACMode);
  Done;
  B := AllocMem(FBufSize);
  try
    Move(FBuffer^, B^, FBufSize);
    EncodeBuffer(B^, B^, FBufSize);
    SetLength(Result, FBufSize);
    Move(FFeedback^, PChar(Result)^, FBufSize);
    if Protection <> nil then Result := Protection.CodeString(Result,
paScramble, Format)
      else Result := StrToFormat(PChar(Result), Length(Result), Format);
  finally
    ReallocMem(B, 0);
    Done;
  end;
end;

class procedure TCipher_Gost.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  ABufSize := 8;
  AKeySize := 32;
  AUserSize := 32;
end;

class function TCipher_Gost.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    0B3h,003h,0A0h,03Fh,0B5h,07Bh,091h,04Dh
         DB    097h,051h,024h,040h,0BDh,0CFh,025h,015h
         DB    034h,005h,09Ch,0F8h,0ABh,010h,086h,09Fh
         DB    0F2h,080h,047h,084h,047h,09Bh,01Ah,0D1h
end;

type
  PCipherRec = ^TCipherRec;
  TCipherRec = packed record
                 case Integer of
```

```
                    0: (X: array[0..7] of Byte);
                    1: (A, B: LongWord);
                end;

procedure TCipher_Gost.Encode(Data: Pointer);
var
  I,A,B,T: LongWord;
  K: PIntArray;
begin
  K := User;
  A := PCipherRec(Data).A;
  B := PCipherRec(Data).B;
  for I := 0 to 11 do
  begin
    if I and 3 = 0 then K := User;
    T := A + K[0];
    B := B xor Gost_Data[0, T        and $FF] xor
               Gost_Data[1, T shr  8 and $FF] xor
               Gost_Data[2, T shr 16 and $FF] xor
               Gost_Data[3, T shr 24        ];
    T := B + K[1];
    A := A xor Gost_Data[0, T        and $FF] xor
               Gost_Data[1, T shr  8 and $FF] xor
               Gost_Data[2, T shr 16 and $FF] xor
               Gost_Data[3, T shr 24        ];
    Inc(PInteger(K), 2);
  end;
  K := @PIntArray(User)[6];
  for I := 0 to 3 do
  begin
    T := A + K[1];
    B := B xor Gost_Data[0, T        and $FF] xor
               Gost_Data[1, T shr  8 and $FF] xor
               Gost_Data[2, T shr 16 and $FF] xor
               Gost_Data[3, T shr 24        ];
    T := B + K[0];
    A := A xor Gost_Data[0, T        and $FF] xor
               Gost_Data[1, T shr  8 and $FF] xor
               Gost_Data[2, T shr 16 and $FF] xor
               Gost_Data[3, T shr 24        ];
    Dec(PInteger(K), 2);
  end;
  PCipherRec(Data).A := B;
  PCipherRec(Data).B := A;
end;

procedure TCipher_Gost.Decode(Data: Pointer);
var
  I,A,B,T: LongWord;
  K: PIntArray;
begin
  A := PCipherRec(Data).A;
  B := PCipherRec(Data).B;
  K := User;
  for I := 0 to 3 do
  begin
    T := A + K[0];
    B := B xor Gost_Data[0, T and $FF] xor
               Gost_Data[1, T shr  8 and $FF] xor
               Gost_Data[2, T shr 16 and $FF] xor
               Gost_Data[3, T shr 24];
    T := B + K[1];
    A := A xor Gost_Data[0, T and $FF] xor
               Gost_Data[1, T shr  8 and $FF] xor
               Gost_Data[2, T shr 16 and $FF] xor
               Gost_Data[3, T shr 24];
    Inc(PInteger(K), 2);
  end;
  for I := 0 to 11 do
```

```
  begin
    if I and 3 = 0 then K := @PIntArray(User)[6];
    T := A + K[1];
    B := B xor Gost_Data[0, T and $FF] xor
               Gost_Data[1, T shr  8 and $FF] xor
               Gost_Data[2, T shr 16 and $FF] xor
               Gost_Data[3, T shr 24];
    T := B + K[0];
    A := A xor Gost_Data[0, T and $FF] xor
               Gost_Data[1, T shr  8 and $FF] xor
               Gost_Data[2, T shr 16 and $FF] xor
               Gost_Data[3, T shr 24];
    Dec(PInteger(K), 2);
  end;
  PCipherRec(Data).A := B;
  PCipherRec(Data).B := A;
end;

procedure TCipher_Gost.Init(const Key; Size: Integer; IVector: Pointer);
begin
  InitBegin(Size);
  Move(Key, User^, Size);
  InitEnd(IVector);
end;

class procedure TCipher_Blowfish.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  ABufSize := 8;
  AKeySize := 56;
  AUserSize := SizeOf(Blowfish_Data) + SizeOf(Blowfish_Key);
end;

class function TCipher_Blowfish.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    019h,071h,0CAh,0CDh,02Bh,09Ch,085h,029h
         DB    0DAh,081h,047h,0B7h,0EBh,0CEh,016h,0C6h
         DB    091h,00Eh,01Dh,0C8h,040h,012h,03Eh,035h
         DB    070h,0EDh,0BCh,096h,04Ch,013h,0D0h,0B8h
end;

type
  PBlowfish = ^TBlowfish;
  TBlowfish = array[0..3, 0..255] of LongWord;

{$IFDEF UseASM}
  {$IFNDEF 486GE}  // не використовується для <= CPU 386
procedure TCipher_Blowfish.Encode386(Data: Pointer);
asm  // спеціально для  CPU < 486
         PUSH   EDI
         PUSH   ESI
         PUSH   EBX
         PUSH   EBP
         PUSH   EDX

         MOV    ESI,[EAX].TCipher_Blowfish.FUser

         MOV    EBX,[EDX]         // A
         MOV    EDX,[EDX + 4]     // B

         XCHG   BL,BH       // це BSWAP EBX,EDX
         XCHG   DL,DH
         ROL    EBX,16
         ROL    EDX,16
         XCHG   BL,BH
         XCHG   DL,DH
```

```
        XOR     EBX,[ESI + 4 * 256 * 4]
        XOR     EDI,EDI

@@1:    MOV     EAX,EBX
        SHR     EBX,16

        MOVZX   ECX,BH
        MOV     EBP,[ESI + ECX * 4 + 1024 * 0]
        MOVZX   ECX,BL
        ADD     EBP,[ESI + ECX * 4 + 1024 * 1]

        MOVZX   ECX,AH
        XOR     EBP,[ESI + ECX * 4 + 1024 * 2]
        MOVZX   ECX,AL
        ADD     EBP,[ESI + ECX * 4 + 1024 * 3]
        XOR     EDX,[ESI + 4 * 256 * 4 + 4 + EDI * 4]

        XOR     EBP,EDX
        MOV     EDX,EAX
        MOV     EBX,EBP
        INC     EDI
        TEST    EDI,010h
        JZ      @@1

        POP     EAX
        XOR     EDX,[ESI + 4 * 256 * 4 + 17 * 4]

        XCHG    BL,BH         // це BSWAP EBX,EDX
        XCHG    DL,DH
        ROL     EBX,16
        ROL     EDX,16
        XCHG    BL,BH
        XCHG    DL,DH

        MOV     [EAX],EDX
        MOV     [EAX + 4],EBX

        POP     EBP
        POP     EBX
        POP     ESI
        POP     EDI
end;

procedure TCipher_Blowfish.Decode386(Data: Pointer);
asm // спеціально для  CPU < 486
        PUSH    EDI
        PUSH    ESI
        PUSH    EBX
        PUSH    EBP
        PUSH    EDX

        MOV     ESI,[EAX].TCipher_Blowfish.FUser

        MOV     EBX,[EDX]         // A
        MOV     EDX,[EDX + 4]     // B

        XCHG    BL,BH
        XCHG    DL,DH
        ROL     EBX,16
        ROL     EDX,16
        XCHG    BL,BH
        XCHG    DL,DH

        XOR     EBX,[ESI + 4 * 256 * 4 + 17 * 4]

        MOV     EDI,16

@@1:    MOV     EAX,EBX
        SHR     EBX,16
```

```
        MOVZX  ECX,BH
        MOV    EBP,[ESI + ECX * 4 + 1024 * 0]
        MOVZX  ECX,BL
        ADD    EBP,[ESI + ECX * 4 + 1024 * 1]

        MOVZX  ECX,AH
        XOR    EBP,[ESI + ECX * 4 + 1024 * 2]
        MOVZX  ECX,AL
        ADD    EBP,[ESI + ECX * 4 + 1024 * 3]
        XOR    EDX,[ESI + 4 * 256 * 4 + EDI * 4]

        XOR    EBP,EDX
        MOV    EDX,EAX
        MOV    EBX,EBP

        DEC    EDI
        JNZ    @@1

        POP    EAX
        XOR    EDX,[ESI + 4 * 256 * 4]

        XCHG   BL,BH        // BSWAP
        XCHG   DL,DH
        ROL    EBX,16
        ROL    EDX,16
        XCHG   BL,BH
        XCHG   DL,DH

        MOV    [EAX],EDX
        MOV    [EAX + 4],EBX

        POP    EBP
        POP    EBX
        POP    ESI
        POP    EDI
end;
  {$ENDIF} //486GE
{$ENDIF}

procedure TCipher_Blowfish.Encode(Data: Pointer);
{$IFDEF UseASM}  // спеціально для   CPU >= 486
asm
        PUSH   EDI
        PUSH   ESI
        PUSH   EBX
        PUSH   EBP
        PUSH   EDX

        MOV    ESI,[EAX].TCipher_Blowfish.FUser
        MOV    EBX,[EDX]         // A
        MOV    EBP,[EDX + 4]     // B

        BSWAP  EBX               // CPU >= 486
        BSWAP  EBP

        XOR    EDI,EDI
        XOR    EBX,[ESI + 4 * 256 * 4]
//      XOR    ECX,ECX
@@1:

        MOV    EAX,EBX
        SHR    EBX,16
        MOVZX  ECX,BH     // це прискорює для AMD Chips,
//        MOV    CL,BH    // це прискорює для PII's
        MOV    EDX,[ESI + ECX * 4 + 1024 * 0]
        MOVZX  ECX,BL
//        MOV    CL,BL
        ADD    EDX,[ESI + ECX * 4 + 1024 * 1]
```

```
        MOVZX  ECX,AH
//        MOV    CL,AH
        XOR    EDX,[ESI + ECX * 4 + 1024 * 2]
        MOVZX  ECX,AL
//        MOV    CL,AL
        ADD    EDX,[ESI + ECX * 4 + 1024 * 3]
        XOR    EBP,[ESI + 4 * 256 * 4 + 4 + EDI * 4]

        INC    EDI
        XOR    EDX,EBP
        TEST   EDI,010h
        MOV    EBP,EAX
        MOV    EBX,EDX
        JZ     @@1

        POP    EAX
        XOR    EBP,[ESI + 4 * 256 * 4 + 17 * 4]

        BSWAP  EBX
        BSWAP  EBP

        MOV    [EAX],EBP
        MOV    [EAX + 4],EBX

        POP    EBP
        POP    EBX
        POP    ESI
        POP    EDI
end;
{$ELSE}
var
  I,A,B: LongWord;
  P: PIntArray;
  D: PBlowfish;
begin
  D := User;
  P := Pointer(PChar(User) + SizeOf(Blowfish_Data));
  A := SwapInteger(PCipherRec(Data).A) xor P[0]; Inc(PInteger(P));
  B := SwapInteger(PCipherRec(Data).B);
  for I := 0 to 7 do
  begin
    B := B xor P[0] xor (D[0, A shr 24        ] +
                         D[1, A shr 16 and $FF] xor
                         D[2, A shr  8 and $FF] +
                         D[3, A        and $FF]);

    A := A xor P[1] xor (D[0, B shr 24        ] +
                         D[1, B shr 16 and $FF] xor
                         D[2, B shr  8 and $FF] +
                         D[3, B        and $FF]);
    Inc(PInteger(P), 2);
  end;
  PCipherRec(Data).A := SwapInteger(B xor P[0]);
  PCipherRec(Data).B := SwapInteger(A);
end;
{$ENDIF}

procedure TCipher_Blowfish.Decode(Data: Pointer);
{$IFDEF UseASM}
asm
        PUSH   EDI
        PUSH   ESI
        PUSH   EBX
        PUSH   EBP
        PUSH   EDX

        MOV    ESI,[EAX].TCipher_Blowfish.FUser
        MOV    EBX,[EDX]         // A
```

```
        MOV    EBP,[EDX + 4]     // B

        BSWAP  EBX
        BSWAP  EBP

        XOR    EBX,[ESI + 4 * 256 * 4 + 17 * 4]
        MOV    EDI,16
//        XOR    ECX,ECX

@@1:    MOV    EAX,EBX
        SHR    EBX,16

        MOVZX  ECX,BH
//        MOV    CL,BH
        MOV    EDX,[ESI + ECX * 4 + 1024 * 0]
        MOVZX  ECX,BL
//        MOV    CL,BL
        ADD    EDX,[ESI + ECX * 4 + 1024 * 1]

        MOVZX  ECX,AH
//        MOV    CL,AH
        XOR    EDX,[ESI + ECX * 4 + 1024 * 2]
        MOVZX  ECX,AL
//        MOV    CL,AL
        ADD    EDX,[ESI + ECX * 4 + 1024 * 3]
        XOR    EBP,[ESI + 4 * 256 * 4 + EDI * 4]

        XOR    EDX,EBP
        DEC    EDI
        MOV    EBP,EAX
        MOV    EBX,EDX
        JNZ    @@1

        POP    EAX
        XOR    EBP,[ESI + 4 * 256 * 4]

        BSWAP  EBX
        BSWAP  EBP

        MOV    [EAX],EBP
        MOV    [EAX + 4],EBX

        POP    EBP
        POP    EBX
        POP    ESI
        POP    EDI
end;
{$ELSE}
var
  I,A,B: LongWord;
  P: PIntArray;
  D: PBlowfish;
begin
  D := User;
  P := Pointer(PChar(User) + SizeOf(Blowfish_Data) + SizeOf(Blowfish_Key) -
SizeOf(Integer));
  A := SwapInteger(PCipherRec(Data).A) xor P[0];
  B := SwapInteger(PCipherRec(Data).B);
  for I := 0 to 7 do
  begin
    Dec(PInteger(P), 2);
    B := B xor P[1] xor (D[0, A shr 24        ] +
                         D[1, A shr 16 and $FF] xor
                         D[2, A shr  8 and $FF] +
                         D[3, A        and $FF]);
    A := A xor P[0] xor (D[0, B shr 24        ] +
                         D[1, B shr 16 and $FF] xor
                         D[2, B shr  8 and $FF] +
                         D[3, B        and $FF]);
```

```
  end;
  Dec(PInteger(P));
  PCipherRec(Data).A := SwapInteger(B xor P[0]);
  PCipherRec(Data).B := SwapInteger(A);
end;
{$ENDIF}

procedure TCipher_Blowfish.Init(const Key; Size: Integer; IVector: Pointer);
var
  I,J: Integer;
  B: array[0..7] of Byte;
  K: PByteArray;
  P: PIntArray;
  S: PBlowfish;
begin
  InitBegin(Size);
  K := @Key;
  S := User;
  P := Pointer(PChar(User) + SizeOf(Blowfish_Data));
  Move(Blowfish_Data, S^, SizeOf(Blowfish_Data));
  Move(Blowfish_Key, P^, Sizeof(Blowfish_Key));
  J := 0;
  for I := 0 to 17 do
  begin
    P[I] := P[I] xor (K[(J + 0) mod Size] shl 24 +
                      K[(J + 1) mod Size] shl 16 +
                      K[(J + 2) mod Size] shl  8 +
                      K[(J + 3) mod Size]);
    J := (J + 4) mod Size;
  end;
  FillChar(B, SizeOf(B), 0);
  for I := 0 to 8 do
  begin
    Encode(@B);
    P[I * 2]     := SwapInteger(PCipherRec(@B).A);
    P[I * 2 + 1] := SwapInteger(PCipherRec(@B).B);
  end;
  for I := 0 to 3 do
    for J := 0 to 127 do
    begin
      Encode(@B);
      S[I, J * 2]    := SwapInteger(PCipherRec(@B).A);
      S[I, J * 2 +1] := SwapInteger(PCipherRec(@B).B);
    end;

  FillChar(B, SizeOf(B), 0);
  InitEnd(IVector);
end;

class procedure TCipher_IDEA.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  ABufSize := 8;
  AKeySize := 16;
  AUserSize := 208;
end;

class function TCipher_IDEA.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    08Ch,065h,0CAh,0D8h,043h,0E7h,099h,093h
         DB    0EDh,041h,0EAh,048h,0FDh,066h,050h,094h
         DB    0A2h,025h,06Dh,0D7h,0B1h,0D0h,09Ah,023h
         DB    03Dh,0D2h,0E8h,0ECh,0C9h,045h,07Fh,07Eh
end;

function IDEAMul(X, Y: LongWord): LongWord; assembler; register;
asm
```

```
      AND     EAX,0FFFFh
      JZ      @@1
      AND     EDX,0FFFFh
      JZ      @@1
      MUL     EDX
      MOV     ECX,EAX
      MOV     EDX,EAX
      SHR     EDX,16
      SUB     EAX,EDX
      CMP     AX,CX
      JNA     @@2
      INC     EAX
@@2:  RET
@@1:  MOV     ECX,1
      SUB     ECX,EAX
      SUB     ECX,EDX
      MOV     EAX,ECX
end;

procedure TCipher_IDEA.Cipher(Data, Key: PWordArray);
var
  I: LongWord;
  X,Y,A,B,C,D: LongWord;
begin
  I := SwapInteger(PIntArray(Data)[0]);
  A := LongRec(I).Hi;
  B := LongRec(I).Lo;
  I := SwapInteger(PIntArray(Data)[1]);
  C := LongRec(I).Hi;
  D := LongRec(I).Lo;
  for I := 0 to 7 do
  begin
    A := IDEAMul(A, Key[0]);
    Inc(B, Key[1]);
    Inc(C, Key[2]);
    D := IDEAMul(D, Key[3]);
    Y := C xor A;
    Y := IDEAMul(Y, Key[4]);
    X := B xor D + Y;
    X := IDEAMul(X, Key[5]);
    Inc(Y, X);
    A := A xor X;
    D := D xor Y;
    Y := B xor Y;
    B := C xor X;
    C := Y;
    Inc(PWord(Key), 6);
  end;
  LongRec(I).Hi := IDEAMul(A, Key[0]);
  LongRec(I).Lo := C + Key[1];
  PIntArray(Data)[0] := SwapInteger(I);
  LongRec(I).Hi := B + Key[2];
  LongRec(I).Lo := IDEAMul(D, Key[3]);
  PIntArray(Data)[1] := SwapInteger(I);
end;

procedure TCipher_IDEA.Encode(Data: Pointer);
begin
  Cipher(Data, User);
end;

procedure TCipher_IDEA.Decode(Data: Pointer);
begin
  Cipher(Data, @PIntArray(User)[26]);
end;

procedure TCipher_IDEA.Init(const Key; Size: Integer; IVector: Pointer);

  function IDEAInv(X: Word): Word;
```

```
  var
    A, B, C, D: Word;
  begin
    if X <= 1 then
    begin
      Result := X;
      Exit;
    end;
    A := 1;
    B := $10001 div X;
    C := $10001 mod X;
    while C <> 1 do
    begin
      D := X div C;
      X := X mod C;
      Inc(A, B * D);
      if X = 1 then
      begin
        Result := A;
        Exit;
      end;
      D := C div X;
      C := C mod X;
      Inc(B, A * D);
    end;
    Result := 1 - B;
  end;

var
  I: Integer;
  E: PWordArray;
  A,B,C: Word;
  K,D: PWordArray;
begin
  InitBegin(Size);
  E := User;
  Move(Key, E^, Size);
  for I := 0 to 7 do E[I] := Swap(E[I]);
  for I := 0 to 39 do
    E[I + 8] := E[I and not 7 + (I + 1) and 7] shl 9 or
                E[I and not 7 + (I + 2) and 7] shr 7;
  for I := 41 to 44 do
    E[I + 7] := E[I] shl 9 or E[I + 1] shr 7;
  K  := E;
  D  := @E[100];
  A  := IDEAInv(K[0]);
  B  := 0 - K[1];
  C  := 0 - K[2];
  D[3] := IDEAInv(K[3]);
  D[2] := C;
  D[1] := B;
  D[0] := A;
  Inc(PWord(K), 4);
  for I := 1 to 8 do
  begin
    Dec(PWord(D), 6);
    A    := K[0];
    D[5] := K[1];
    D[4] := A;
    A    := IDEAInv(K[2]);
    B    := 0 - K[3];
    C    := 0 - K[4];
    D[3] := IDEAInv(K[5]);
    D[2] := B;
    D[1] := C;
    D[0] := A;
    Inc(PWord(K), 6);
  end;
  A := D[2]; D[2] := D[1]; D[1] := A;
```

```
  InitEnd(IVector);
end;

type
  PSAFERRec = ^TSAFERRec;
  TSAFERRec = packed record
                case Integer of
                  0: (A,B,C,D,E,F,G,H: Byte);
                  1: (X,Y: Integer);
              end;

procedure TCipher_SAFER.SetRounds(Value: Integer);
begin
  if (Value < 4) or (Value > 13) then
    case FSaferMode of  {Визначений раунд }
      smK40, smSK40: Value := 5;
      smK64, smSK64: Value := 6;
      smK128, smSK128: Value := 10;
    else
      Value := 8;
    end;
  FRounds := Value;
end;

class procedure TCipher_SAFER.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  ABufSize := 8;
  AKeySize := 16;
  AUserSize := 768;
end;

class function TCipher_SAFER.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    000h,03Dh,049h,020h,073h,063h,085h,0AAh
         DB    0D9h,0C2h,00Ah,0DEh,07Eh,09Eh,0E9h,0ABh
         DB    024h,0D0h,074h,034h,047h,07Eh,021h,01Dh
         DB    055h,0F9h,035h,028h,098h,084h,0A8h,075h
end;

procedure TCipher_SAFER.Encode(Data: Pointer);
var
  Exp,Log,Key: PByteArray;
  I: Integer;
  T: Byte;
begin
  Exp := User;
  Log := Pointer(PChar(User) + 256);
  Key := Pointer(PChar(User) + 512);
  with PSAFERRec(Data)^ do
  begin
    for I := 1 to FRounds do
    begin
      A := A xor Key[0];
      B := B  +  Key[1];
      C := C  +  Key[2];
      D := D xor Key[3];
      E := E xor Key[4];
      F := F  +  Key[5];
      G := G  +  Key[6];
      H := H xor Key[7];

      A := Exp[A]  +  Key[8];
      B := Log[B] xor Key[9];
      C := Log[C] xor Key[10];
      D := Exp[D]  +  Key[11];
      E := Exp[E]  +  Key[12];
```

```
      F := Log[F] xor Key[13];
      G := Log[G] xor Key[14];
      H := Exp[H]  +  Key[15];

      Inc(B, A); Inc(A, B);
      Inc(D, C); Inc(C, D);
      Inc(F, E); Inc(E, F);
      Inc(H, G); Inc(G, H);

      Inc(C, A); Inc(A, C);
      Inc(G, E); Inc(E, G);
      Inc(D, B); Inc(B, D);
      Inc(H, F); Inc(F, H);

      Inc(E, A); Inc(A, E);
      Inc(F, B); Inc(B, F);
      Inc(G, C); Inc(C, G);
      Inc(H, D); Inc(D, H);

      T := B; B := E; E := C; C := T;
      T := D; D := F; F := G; G := T;

      Inc(PByte(Key), 16);
    end;
    A := A xor Key[0];
    B := B  +  Key[1];
    C := C  +  Key[2];
    D := D xor Key[3];
    E := E xor Key[4];
    F := F  +  Key[5];
    G := G  +  Key[6];
    H := H xor Key[7];
  end;
end;

procedure TCipher_SAFER.Decode(Data: Pointer);
var
  Exp,Log,Key: PByteArray;
  I: Integer;
  T: Byte;
begin
  Exp := User;
  Log := Pointer(PChar(User) + 256);
  Key := Pointer(PChar(User) + 504 + 8 * (FRounds * 2 + 1));
  with PSAFERRec(Data)^ do
  begin
    H := H xor Key[7];
    G := G  -  Key[6];
    F := F  -  Key[5];
    E := E xor Key[4];
    D := D xor Key[3];
    C := C  -  Key[2];
    B := B  -  Key[1];
    A := A xor Key[0];

    for I := 1 to FRounds do
    begin
      Dec(PByte(Key), 16);
      T := E; E := B; B := C; C := T;
      T := F; F := D; D := G; G := T;

      Dec(A, E); Dec(E, A);
      Dec(B, F); Dec(F, B);
      Dec(C, G); Dec(G, C);
      Dec(D, H); Dec(H, D);

      Dec(A, C); Dec(C, A);
      Dec(E, G); Dec(G, E);
      Dec(B, D); Dec(D, B);
```

```
      Dec(F, H); Dec(H, F);

      Dec(A, B); Dec(B, A);
      Dec(C, D); Dec(D, C);
      Dec(E, F); Dec(F, E);
      Dec(G, H); Dec(H, G);

      H := H  -  Key[15];
      G := G xor Key[14];
      F := F xor Key[13];
      E := E  -  Key[12];
      D := D  -  Key[11];
      C := C xor Key[10];
      B := B xor Key[9];
      A := A  -  Key[8];

      H := Log[H] xor Key[7];
      G := Exp[G]  -  Key[6];
      F := Exp[F]  -  Key[5];
      E := Log[E] xor Key[4];
      D := Log[D] xor Key[3];
      C := Exp[C]  -  Key[2];
      B := Exp[B]  -  Key[1];
      A := Log[A] xor Key[0];
    end;
  end;
end;

procedure TCipher_SAFER.Init(const Key; Size: Integer; IVector: Pointer);
begin
  InitNew(Key, Size, IVector, smStrong);
end;

procedure TCipher_SAFER.InitNew(const Key; Size: Integer; IVector: Pointer;
SAFERMode: TSAFERMode);

  procedure InitTab;
  var
    I,E: Integer;
    Exp: PByte;
    Log: PByteArray;
  begin
    Exp := User;
    Log := Pointer(PChar(User) + 256);
    E   := 1;
    for I := 0 to 255 do
    begin
      Exp^ := E and $FF;
      Log[E and $FF] := I;
      E := (E * 45) mod 257;
      Inc(Exp);
    end;
  end;

  procedure InitKey;

    function ROR3(Value: Byte): Byte; assembler;
    asm
      ROR AL,3
    end;

    function ROL6(Value: Byte): Byte; assembler;
    asm
      ROL AL,6
    end;

  var
    D: PByte;
    Exp: PByteArray;
```

```
    Strong: Boolean;
    K: array[Boolean, 0..8] of Byte;
    I,J: Integer;
  begin
    Strong := FSAFERMode in [smStrong, smSK40, smSK64, smSK128];
    Exp := User;
    D := User;
    Inc(D, 512);
    FillChar(K, SizeOf(K), 0);
{Установка ключа A}
    I := Size;
    if I > 8 then I := 8;
    Move(Key, K[False], I);
{Установка ключа для K-40, SK-40}
    if FSAFERMode in [smK40, smSK40] then
    begin
      K[False, 5] := K[False, 0] xor K[False, 2] xor 129;
      K[False, 6] := K[False, 0] xor K[False, 3] xor K[False, 4] xor 66;
      K[False, 7] := K[False, 1] xor K[False, 2] xor K[False, 4] xor 36;
      K[False, 8] := K[False, 1] xor K[False, 3] xor 24;
      Move(K[False], K[True], SizeOf(K[False]));
    end else
    begin
      if Size > 8 then
      begin
        I := Size - 8;
        if I > 8 then I := 8;
        Move(TByteArray(Key)[8], K[True], I);
      end else Move(K[False], K[True], 9);
      for I := 0 to 7 do
      begin
        K[False, 8] := K[False, 8] xor K[False, I];
        K[True, 8]  := K[True, 8]  xor K[True, I];
      end;
    end;
{Установка KeyData}
    Move(K[True], D^, 8);
    Inc(D, 8);

    for I := 0 to 8 do K[False, I] := ROR3(K[False, I]);

    for I := 1 to FRounds do
    begin
      for J := 0 to 8 do
      begin
        K[False, J] := ROL6(K[False, J]);
        K[True, J] := ROL6(K[True, J]);
      end;
      for J := 0 to 7 do
      begin
        if Strong then D^ := K[False, (J + I * 2 -1) mod 9] + Exp[Exp[18 * I + J
+1]]
          else D^ := K[False, J] + Exp[Exp[18 * I + J +1]];
        Inc(D);
      end;
      for J := 0 to 7 do
      begin
        if Strong then D^ := K[True, (J + I * 2) mod 9] + Exp[Exp[18 * I + J
+10]]
          else D^ := K[True, J] + Exp[Exp[18 * I + J +10]];
        Inc(D);
      end;
    end;
    FillChar(K, SizeOf(K), 0);
  end;

begin
  InitBegin(Size);
  FSAFERMode := SAFERMode;
```

```
  if SAFERMode = smDefault then
    if Size <= 5 then FSAFERMode := smK40 else
      if Size <= 8 then FSAFERMode := smK64 else FSAFERMode := smK128
  else
    if SAFERMode = smStrong then
      if Size <= 5 then FSAFERMode := smSK40 else
        if Size <= 8 then FSAFERMode := smSK64 else FSAFERMode := smSK128;
  SetRounds(FRounds);
  InitTab;
  InitKey;
  InitEnd(IVector);
end;

class procedure TCipher_SAFER_K40.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  inherited GetContext(ABufSize, AKeySize, AUserSize);
  AKeySize := 5;
end;

class function TCipher_SAFER_K40.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    005h,0B4h,019h,057h,026h,05Ch,013h,060h
         DB    0A0h,082h,094h,045h,0D6h,0A5h,046h,0D8h
         DB    073h,050h,096h,080h,04Fh,06Dh,0F7h,0E5h
         DB    0C8h,01Ah,0EFh,044h,04Ch,0B4h,059h,013h
end;

procedure TCipher_SAFER_K40.Init(const Key; Size: Integer; IVector: Pointer);
begin
  InitNew(Key, Size, IVector, smK40);
end;

class function TCipher_SAFER_SK40.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    0D9h,003h,003h,06Dh,018h,038h,0D1h,0C1h
         DB    089h,0E8h,038h,012h,07Fh,028h,0FCh,0C7h
         DB    0C5h,00Bh,0B7h,0C4h,0DBh,021h,0A4h,031h
         DB    020h,008h,08Ah,077h,0F7h,0DFh,026h,0FFh
end;

procedure TCipher_SAFER_SK40.Init(const Key; Size: Integer; IVector: Pointer);
begin
  InitNew(Key, Size, IVector, smSK40);
end;

class procedure TCipher_SAFER_K64.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  inherited GetContext(ABufSize, AKeySize, AUserSize);
  AKeySize := 8;
end;

class function TCipher_SAFER_K64.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    08Ch,0B2h,032h,0F0h,00Eh,0C2h,0DAh,0CBh
         DB    039h,008h,02Dh,05Ch,093h,0FFh,0CEh,0F3h
         DB    08Fh,01Fh,0B7h,02Ch,0C5h,0C7h,0A7h,0E9h
         DB    089h,0BEh,061h,08Bh,000h,0E6h,09Fh,00Eh
end;

procedure TCipher_SAFER_K64.Init(const Key; Size: Integer; IVector: Pointer);
begin
```

```
  InitNew(Key, Size, IVector, smK64);
end;

class function TCipher_SAFER_SK64.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    0DDh,09Ch,01Ah,0D6h,029h,00Ch,0EEh,04Fh
         DB    0E5h,04Bh,0C0h,055h,0BFh,022h,00Eh,0BCh
         DB    019h,041h,078h,0CFh,094h,0DBh,02Fh,039h
         DB    06Bh,01Eh,0A7h,0CAh,04Bh,05Fh,077h,0E0h
end;

procedure TCipher_SAFER_SK64.Init(const Key; Size: Integer; IVector: Pointer);
begin
  InitNew(Key, Size, IVector, smSK64);
end;

class procedure TCipher_SAFER_K128.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  inherited GetContext(ABufSize, AKeySize, AUserSize);
  AKeySize := 16;
end;

class function TCipher_SAFER_K128.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    00Ch,0A9h,070h,0B9h,0F3h,014h,087h,0D9h
         DB    09Eh,05Eh,078h,031h,074h,0DFh,0A8h,0BBh
         DB    03Dh,040h,0A5h,0D9h,08Ch,07Ch,004h,0B7h
         DB    09Ch,001h,0DAh,063h,0ABh,026h,035h,0BCh
end;

procedure TCipher_SAFER_K128.Init(const Key; Size: Integer; IVector: Pointer);
begin
  InitNew(Key, Size, IVector, smK128);
end;

class function TCipher_SAFER_SK128.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    0C8h,0A6h,070h,033h,029h,038h,038h,02Bh
         DB    069h,0ACh,061h,072h,08Fh,0DCh,09Fh,0A4h
         DB    09Eh,06Fh,0C4h,053h,0D8h,089h,0FFh,042h
         DB    072h,009h,07Dh,0CDh,0D0h,0EAh,07Eh,028h
end;

procedure TCipher_SAFER_SK128.Init(const Key; Size: Integer; IVector: Pointer);
begin
  InitNew(Key, Size, IVector, smSK128);
end;

type
  PTEARec = ^TTEARec;
  TTEARec = packed record
              A,B,C,D: LongWord;
            end;

const
  TEA_Delta = $9E3779B9;

procedure TCipher_TEA.SetRounds(Value: Integer);
begin
  FRounds := Value;
  if FRounds < 16 then FRounds := 16 else
    if FRounds > 32 then FRounds := 32;
```

```
end;

class procedure TCipher_TEA.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  ABufSize := 8;
  AKeySize := 16;
  AUserSize := 32;
end;

class function TCipher_TEA.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    0B7h,0B8h,0AAh,0BBh,026h,04Bh,006h,0F9h
         DB    070h,086h,0B0h,0E4h,056h,004h,029h,0CCh
         DB    0BFh,055h,0EAh,04Eh,0EFh,059h,026h,018h
         DB    019h,0B0h,003h,07Ch,029h,08Ch,0E2h,077h
end;

procedure TCipher_TEA.Encode(Data: Pointer);
{$IFDEF UseASM}
asm
         PUSH  EDI
         PUSH  ESI
         PUSH  EBX
         PUSH  EBP
         PUSH  EDX

         MOV   EBX,[EDX]       // X
         MOV   EDX,[EDX + 4]   // Y
         XOR   EDI,EDI         // Sum

         MOV   ESI,[EAX].TCipher_TEA.FUser  // Користувач
         MOV   ECX,[EAX].TCipher_TEA.FRounds  // Раунди

@@1:     ADD   EDI,TEA_Delta

         MOV   EAX,EDX
         MOV   EBP,EDX
         SHL   EAX,4
         SHR   EBP,5
         ADD   EAX,[ESI]
         ADD   EBP,[ESI +  4]
         XOR   EAX,EDX
         ADD   EAX,EDI

         XOR   EAX,EBP
         ADD   EAX,EBX
         MOV   EBX,EAX
         SHL   EAX,4
         MOV   EBP,EBX
         SHR   EBP,5
         ADD   EAX,[ESI +  8]
         XOR   EAX,EBX
         ADD   EBP,[ESI + 12]
         ADD   EAX,EDI

         XOR   EAX,EBP
         ADD   EDX,EAX

         DEC   ECX
         JNZ   @@1

         POP   EAX
         MOV   [EAX],EBX
         MOV   [EAX + 4],EDX

         POP   EBP
```

```
          POP   EBX
          POP   ESI
          POP   EDI
end;
{$ELSE}
var
  I,Sum,X,Y: LongWord;
begin
  Sum := 0;
  X := PTEARec(Data).A;
  Y := PTEARec(Data).B;
  with PTEARec(User)^ do
    for I := 1 to FRounds do
    begin
      Inc(Sum, TEA_Delta);
      Inc(X, (Y shl 4 + A) xor Y + Sum xor (Y shr 5 + B));
      Inc(Y, (X shl 4 + C) xor X + Sum xor (X shr 5 + D));
    end;
  PTEARec(Data).A := X;
  PTEARec(Data).B := Y;
end;
{$ENDIF}

procedure TCipher_TEA.Decode(Data: Pointer);
{$IFDEF UseASM}
asm
          PUSH  EDI
          PUSH  ESI
          PUSH  EBX
          PUSH  EBP
          PUSH  EDX

          MOV   EBX,[EDX]       // X
          MOV   EDX,[EDX + 4]   // Y

          MOV   ESI,[EAX].TCipher_TEA.FUser  // Користувач
          MOV   EDI,TEA_Delta
          MOV   ECX,[EAX].TCipher_TEA.FRounds  // Раунди
          IMUL  EDI,ECX

@@1:      MOV   EAX,EBX
          MOV   EBP,EBX
          SHL   EAX,4
          SHR   EBP,5
          ADD   EAX,[ESI +  8]
          ADD   EBP,[ESI + 12]
          XOR   EAX,EBX
          ADD   EAX,EDI
          XOR   EAX,EBP
          SUB   EDX,EAX
          MOV   EAX,EDX
          SHL   EAX,4
          MOV   EBP,EDX
          SHR   EBP,5
          ADD   EAX,[ESI]
          XOR   EAX,EDX
          ADD   EBP,[ESI + 4]
          ADD   EAX,EDI

          XOR   EAX,EBP
          SUB   EDI,TEA_Delta
          SUB   EBX,EAX

          DEC   ECX
          JNZ   @@1

          POP   EAX
          MOV   [EAX],EBX
          MOV   [EAX + 4],EDX
```

```
          POP   EBP
          POP   EBX
          POP   ESI
          POP   EDI
end;
{$ELSE}
var
  I,Sum,X,Y: LongWord;
begin
  Sum := TEA_Delta * LongWord(FRounds);
  X := PTEARec(Data).A;
  Y := PTEARec(Data).B;
  with PTEARec(User)^ do
    for I := 1 to FRounds do
    begin
      Dec(Y, (X shl 4 + C) xor X + Sum xor (X shr 5 + D));
      Dec(X, (Y shl 4 + A) xor Y + Sum xor (Y shr 5 + B));
      Dec(Sum, TEA_Delta);
    end;
  PTEARec(Data).A := X;
  PTEARec(Data).B := Y;
end;
{$ENDIF}

procedure TCipher_TEA.Init(const Key; Size: Integer; IVector: Pointer);
begin
  InitBegin(Size);
  Move(Key, User^, Size);
  SetRounds(FRounds);
  InitEnd(IVector);
end;

class function TCipher_TEAN.TestVector: Pointer;
asm
          MOV   EAX,OFFSET @Vector
          RET
@Vector:  DB    0CDh,07Eh,0BBh,0A2h,092h,01Ah,04Bh,03Bh
          DB    0E2h,09Eh,062h,0CFh,0F7h,01Dh,0A5h,0DFh
          DB    063h,033h,094h,029h,0E2h,036h,07Ch,066h
          DB    03Fh,0F8h,01Ah,0F9h,002h,078h,0BFh,0A1h
end;

procedure TCipher_TEAN.Encode(Data: Pointer);
var
  I,Sum,X,Y: LongWord;
  K: PIntArray;
begin
  Sum := 0;
  X := PTEARec(Data).A;
  Y := PTEARec(Data).B;
  K := User;
  for I := 1 to FRounds do
  begin
    Inc(X, (Y shl 4 xor Y shr 5) + (Y xor Sum) + K[Sum and 3]);
    Inc(Sum, TEA_Delta);
    Inc(Y, (X shl 4 xor X shr 5) + (X xor Sum) + K[Sum shr 11 and 3]);
  end;
  PTEARec(Data).A := X;
  PTEARec(Data).B := Y;
end;

procedure TCipher_TEAN.Decode(Data: Pointer);
var
  I,Sum,X,Y: LongWord;
  K: PIntArray;
begin
  Sum := TEA_Delta * LongWord(FRounds);
  X := PTEARec(Data).A;
```

```
  Y := PTEARec(Data).B;
  K := User;
  with PTEARec(User)^ do
    for I := 1 to FRounds do
    begin
      Dec(Y, (X shl 4 xor X shr 5) + (X xor Sum) + K[Sum shr 11 and 3]);
      Dec(Sum, TEA_Delta);
      Dec(X, (Y shl 4 xor Y shr 5) + (Y xor Sum) + K[Sum and 3]);
    end;
  PTEARec(Data).A := X;
  PTEARec(Data).B := Y;
end;

const
  SCOP_SIZE = 32; {is the Maximum}

class procedure TCipher_SCOP.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  ABufSize := SCOP_SIZE * SizeOf(Integer);
  AKeySize := 48;
  AUserSize := (384 * 4 + 4 * SizeOf(Integer)) * 2;
end;

class function TCipher_SCOP.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    014h,0C0h,009h,0E8h,073h,0B6h,053h,092h
         DB    08Bh,013h,069h,0A9h,0F2h,099h,0FEh,05Eh
         DB    0EEh,03Bh,0FDh,0C1h,050h,059h,00Eh,094h
         DB    062h,017h,008h,01Eh,0A4h,01Ah,04Dh,08Fh
end;

procedure TCipher_SCOP.Encode(Data: Pointer);
var
  I, J, W: Byte;
  T, T1, T2, T3: Integer;
  P: PIntArray;
  B: PInteger;
begin
  P  := User;
  I  := P[0];
  J  := P[1];
  T3 := P[3];
  P  := @P[4 + 128];
  B  := Data;
  for W := 1 to SCOP_SIZE do
  begin
    T1 := P[J];
    Inc(J, T3);
    T := P[I - 128];
    T2 := P[J];
    Inc(I);
    T3 := T2 + T;
    P[J] := T3;
    Inc(J, T2);
    Inc(B^, T1 + T2);
    Inc(B);
  end;
end;

procedure TCipher_SCOP.Decode(Data: Pointer);
var
  I, J, W: Byte;
  T, T1, T2, T3: Integer;
  P: PIntArray;
  B: PInteger;
begin
```

```
  P  := User;
  I  := P[0];
  J  := P[1];
  T3 := P[3];
  P  := @P[4 + 128];
  B  := Data;
  for W := 1 to SCOP_SIZE do
  begin
    T1 := P[J];
    Inc(J, T3);
    T := P[I - 128];
    T2 := P[J];
    Inc(I);
    T3 := T2 + T;
    P[J] := T3;
    Inc(J, T2);
    Dec(B^, T1 + T2);
    Inc(B);
  end;
end;

procedure TCipher_SCOP.Init(const Key; Size: Integer; IVector: Pointer);
var
  Init_State: packed record
                Coef: array[0..7, 0..3] of Byte;
                X: array[0..3] of LongWord;
              end;

  procedure ExpandKey;
  var
    P: PByteArray;
    I,C: Integer;
  begin
    C := 1;
    P := @Init_State;
    Move(Key, P^, Size);
    for I := Size to 47 do P[I] := P[I - Size] + P[I - Size +1];
    for I := 0 to 31 do
      if P[I] = 0 then
      begin
        P[I] := C;
        Inc(C);
      end;
  end;

  procedure GP8(Data: PIntArray);
  var
    I,I2: Integer;
    NewX: array[0..3] of LongWord;
    X1,X2,X3,X4: LongWord;
    Y1,Y2: LongWord;
  begin
    I := 0;
    while I < 8 do
    begin
      I2 := I shr 1;
      X1 := Init_State.X[I2] shr 16;
      X2 := X1 * X1;
      X3 := X2 * X1;
      X4 := X3 * X1;
      Y1 := Init_State.Coef[I][0] * X4 +
            Init_State.Coef[I][1] * X3 +
            Init_State.Coef[I][2] * X2 +
            Init_State.Coef[I][3] * X1 + 1;
      X1 := Init_State.X[I2] and $FFFF;
      X2 := X1 * X1;
      X3 := X2 * X1;
      X4 := X3 * X1;
      Y2 := Init_State.Coef[I +1][0] * X4 +
```

```
            Init_State.Coef[I +2][1] * X3 +
            Init_State.Coef[I +3][2] * X2 +
            Init_State.Coef[I +4][3] * X1 + 1;
      Data[I2] := Y1 shl 16 or Y2 and $FFFF;
      NewX[I2] := Y1 and $FFFF0000 or Y2 shr 16;
      Inc(I, 2);
    end;
    Init_State.X[0] := NewX[0] shr 16 or NewX[3] shl 16;
    Init_State.X[1] := NewX[0] shl 16 or NewX[1] shr 16;
    Init_State.X[2] := NewX[1] shl 16 or NewX[2] shr 16;
    Init_State.X[3] := NewX[2] shl 16 or NewX[3] shr 16;
  end;

var
  I,J: Integer;
  T: array[0..3] of Integer;
  P: PIntArray;
begin
  InitBegin(Size);
  FillChar(Init_State, SizeOf(Init_State), 0);
  FillChar(T, SizeOf(T), 0);
  P := Pointer(PChar(User) + 12);
  ExpandKey;
  for I := 0 to 7 do GP8(@T);
  for I := 0 to 11 do
  begin
    for J := 0 to 7 do GP8(@P[I * 32 + J * 4]);
    GP8(@T);
  end;
  GP8(@T);
  I := T[3] and $7F;
  P[I] := P[I] or 1;
  P := User;
  P[0] := T[3] shr 24;
  P[1] := T[3] shr 16;
  P[2] := T[3] shr  8;
  FillChar(Init_State, SizeOf(Init_State), 0);
  InitEnd(IVector);
  P := Pointer(PChar(User) + FUserSize shr 1);
  Move(User^, P^, FUserSize shr 1);
end;

procedure TCipher_SCOP.Done;
begin
  inherited Done;
  Move(PByteArray(User)[FUserSize shr 1], User^, FUserSize shr 1);
end;

class procedure TCipher_Q128.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  ABufSize := 16;
  AKeySize := 16;
  AUserSize := 256;
end;

class function TCipher_Q128.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    099h,0AAh,0D0h,03Dh,0CAh,014h,04Eh,02Ah
         DB    0F8h,01Eh,001h,0A0h,0EAh,0ABh,09Fh,048h
         DB    023h,02Dh,059h,054h,054h,07Eh,02Bh,012h
         DB    086h,080h,0E8h,033h,0EBh,0E1h,05Eh,0AEh
end;

procedure TCipher_Q128.Encode(Data: Pointer);
{$IFDEF UseASM}
asm
```

```
        PUSH    ESI
        PUSH    EDI
        PUSH    EBX
        PUSH    EBP
        PUSH    EDX

        MOV     EDI,[EAX].TCipher_Q128.FUser

        MOV     EAX,[EDX]       // B0
        MOV     EBX,[EDX +  4]  // B1
        MOV     ECX,[EDX +  8]  // B2
        MOV     EDX,[EDX + 12]  // B3

        MOV     EBP,16

@@1:    MOV     ESI,EAX
        AND     EAX,03FFh
        MOV     EAX,[EAX * 4 + OFFSET Q128_DATA]
        ROL     ESI,10
        ADD     EAX,[EDI]
        XOR     EAX,EBX

        MOV     EBX,EAX
        AND     EAX,03FFh
        MOV     EAX,[EAX * 4 + OFFSET Q128_DATA]
        ROL     EBX,10
        ADD     EAX,[EDI + 4]
        XOR     EAX,ECX

        MOV     ECX,EAX
        AND     EAX,03FFh
        MOV     EAX,[EAX * 4 + OFFSET Q128_DATA]
        ROL     ECX,10
        ADD     EAX,[EDI + 8]
        XOR     EAX,EDX

        MOV     EDX,EAX
        AND     EAX,03FFh
        MOV     EAX,[EAX * 4 + OFFSET Q128_DATA]
        ROL     EDX,10
        ADD     EAX,[EDI + 12]
        XOR     EAX,ESI

        ADD     EDI,16

        DEC     EBP
        JNZ     @@1

        POP     ESI

        MOV     [ESI],EAX       // B0
        MOV     [ESI +  4],EBX  // B1
        MOV     [ESI +  8],ECX  // B2
        MOV     [ESI + 12],EDX  // B3


        POP     EBP
        POP     EBX
        POP     EDI
        POP     ESI
end;
{$ELSE}
var
  D: PInteger;
  B0, B1, B2, B3, I: LongWord;
begin
  D  := User;
  B0 := PIntArray(Data)[0];
  B1 := PIntArray(Data)[1];
```

```
  B2 := PIntArray(Data)[2];
  B3 := PIntArray(Data)[3];
  for I := 1 to 16 do
  begin
    B1 := B1 xor (Q128_Data[B0 and $03FF] + D^); Inc(D); B0 := B0 shl 10 or B0
shr 22;
    B2 := B2 xor (Q128_Data[B1 and $03FF] + D^); Inc(D); B1 := B1 shl 10 or B1
shr 22;
    B3 := B3 xor (Q128_Data[B2 and $03FF] + D^); Inc(D); B2 := B2 shl 10 or B2
shr 22;
    B0 := B0 xor (Q128_Data[B3 and $03FF] + D^); Inc(D); B3 := B3 shl 10 or B3
shr 22;
  end;
  PIntArray(Data)[0] := B0;
  PIntArray(Data)[1] := B1;
  PIntArray(Data)[2] := B2;
  PIntArray(Data)[3] := B3;
end;
{$ENDIF}
procedure TCipher_Q128.Decode(Data: Pointer);
{$IFDEF UseASM}
asm
       PUSH   ESI
       PUSH   EDI
       PUSH   EBX
       PUSH   EBP
       PUSH   EDX

       MOV    EDI,[EAX].TCipher_Q128.FUser
       LEA    EDI,[EDI + 64 * 4]

       MOV    ESI,[EDX]       // B0
       MOV    EBX,[EDX +  4]  // B1
       MOV    ECX,[EDX +  8]  // B2
       MOV    EDX,[EDX + 12]  // B3

       MOV    EBP,16

@@1:   SUB    EDI,16

       ROR    EDX,10
       MOV    EAX,EDX
       AND    EAX,03FFh
       MOV    EAX,[EAX * 4 + OFFSET Q128_DATA]
       ADD    EAX,[EDI + 12]
       XOR    ESI,EAX

       ROR    ECX,10
       MOV    EAX,ECX
       AND    EAX,03FFh
       MOV    EAX,[EAX * 4 + OFFSET Q128_DATA]
       ADD    EAX,[EDI +  8]
       XOR    EDX,EAX

       ROR    EBX,10
       MOV    EAX,EBX
       AND    EAX,03FFh
       MOV    EAX,[EAX * 4 + OFFSET Q128_DATA]
       ADD    EAX,[EDI +  4]
       XOR    ECX,EAX

       ROR    ESI,10
       MOV    EAX,ESI
       AND    EAX,03FFh
       MOV    EAX,[EAX * 4 + OFFSET Q128_DATA]
       ADD    EAX,[EDI]
       XOR    EBX,EAX

       DEC    EBP
```

```
        JNZ     @@1

        POP     EAX

        MOV     [EAX],ESI       // B0
        MOV     [EAX +  4],EBX  // B1
        MOV     [EAX +  8],ECX  // B2
        MOV     [EAX + 12],EDX  // B3


        POP     EBP
        POP     EBX
        POP     EDI
        POP     ESI
end;
{$ELSE}
var
  D: PInteger;
  B0, B1, B2, B3, I: LongWord;
begin
  D  := @PIntArray(User)[63];
  B0 := PIntArray(Data)[0];
  B1 := PIntArray(Data)[1];
  B2 := PIntArray(Data)[2];
  B3 := PIntArray(Data)[3];
  for I := 1 to 16 do
  begin
    B3 := B3 shr 10 or B3 shl 22; B0 := B0 xor (Q128_Data[B3 and $03FF] + D^);
Dec(D);
    B2 := B2 shr 10 or B2 shl 22; B3 := B3 xor (Q128_Data[B2 and $03FF] + D^);
Dec(D);
    B1 := B1 shr 10 or B1 shl 22; B2 := B2 xor (Q128_Data[B1 and $03FF] + D^);
Dec(D);
    B0 := B0 shr 10 or B0 shl 22; B1 := B1 xor (Q128_Data[B0 and $03FF] + D^);
Dec(D);
  end;
  PIntArray(Data)[0] := B0;
  PIntArray(Data)[1] := B1;
  PIntArray(Data)[2] := B2;
  PIntArray(Data)[3] := B3;
end;
{$ENDIF}

procedure TCipher_Q128.Init(const Key; Size: Integer; IVector: Pointer);
var
  K: array[0..3] of LongWord;
  I: Integer;
  D: PInteger;
begin
  InitBegin(Size);
  FillChar(K, SizeOf(K), 0);
  Move(Key, K, Size);
  D := User;
  for I := 19 downto 1 do
  begin
    K[1] := K[1] xor Q128_Data[K[0] and $03FF]; K[0] := K[0] shr 10 or K[0] shl
22;
    K[2] := K[2] xor Q128_Data[K[1] and $03FF]; K[1] := K[1] shr 10 or K[1] shl
22;
    K[3] := K[3] xor Q128_Data[K[2] and $03FF]; K[2] := K[2] shr 10 or K[2] shl
22;
    K[0] := K[0] xor Q128_Data[K[3] and $03FF]; K[3] := K[3] shr 10 or K[3] shl
22;
    if I <= 16 then
    begin
      D^ := K[0]; Inc(D);
      D^ := K[1]; Inc(D);
      D^ := K[2]; Inc(D);
      D^ := K[3]; Inc(D);
```

```
    end;
  end;
  FillChar(K, SizeOf(K), 0);
  InitEnd(IVector);
end;

type
  P3Way_Key = ^T3Way_Key;
  T3Way_Key = packed record
                E_Key: array[0..2] of Integer;
                E_Data: array[0..11] of Integer;
                D_Key: array[0..2] of Integer;
                D_Data: array[0..11] of Integer;
              end;

class procedure TCipher_3Way.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  ABufSize := 12;
  AKeySize := 12;
  AUserSize := SizeOf(T3Way_Key);
end;

class function TCipher_3Way.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    077h,0FCh,077h,094h,07Ch,08Fh,0DEh,021h
         DB    0E9h,081h,0DFh,02Ah,0B1h,0BCh,07Eh,0F8h
         DB    0A3h,0B6h,044h,04Bh,0B6h,0FCh,079h,0C4h
         DB    09Bh,068h,04Fh,009h,0C7h,0BFh,00Eh,005h
end;

procedure TCipher_3Way.Encode(Data: Pointer);
var
  I: Integer;
  A0,A1,A2: LongWord;
  B0,B1,B2: LongWord;
  K0,K1,K2: LongWord;
  E: PLongWord;
begin
  with P3Way_Key(User)^ do
  begin
    K0 := E_Key[0];
    K1 := E_Key[1];
    K2 := E_Key[2];
    E  := @E_Data;
  end;
  A0 := PIntArray(Data)[0];
  A1 := PIntArray(Data)[1];
  A2 := PIntArray(Data)[2];
  for I := 0 to 10 do
  begin
    A0 := A0 xor K0 xor E^ shl 16;
    A1 := A1 xor K1;
    A2 := A2 xor K2 xor E^;
    Inc(E);

    B0 := A0 xor A0 shr 16 xor A1 shl 16 xor A1 shr 16 xor A2 shl 16 xor
                A1 shr 24 xor A2 shl  8 xor A2 shr  8 xor A0 shl 24 xor
                A2 shr 16 xor A0 shl 16 xor A2 shr 24 xor A0 shl  8;
    B1 := A1 xor A1 shr 16 xor A2 shl 16 xor A2 shr 16 xor A0 shl 16 xor
                A2 shr 24 xor A0 shl  8 xor A0 shr  8 xor A1 shl 24 xor
                A0 shr 16 xor A1 shl 16 xor A0 shr 24 xor A1 shl  8;
    B2 := A2 xor A2 shr 16 xor A0 shl 16 xor A0 shr 16 xor A1 shl 16 xor
                A0 shr 24 xor A1 shl  8 xor A1 shr  8 xor A2 shl 24 xor
                A1 shr 16 xor A2 shl 16 xor A1 shr 24 xor A2 shl  8;
    asm
      ROR  B0,10
```

```
      ROL  B2,1
    end;
    A0 := B0 xor (B1 or not B2);
    A1 := B1 xor (B2 or not B0);
    A2 := B2 xor (B0 or not B1);
    asm
      ROL A0,1
      ROR A2,10
    end;
  end;
  A0 := A0 xor K0 xor E^ shl 16;
  A1 := A1 xor K1;
  A2 := A2 xor K2 xor E^;
  PIntArray(Data)[0] := A0 xor A0 shr 16 xor A1 shl 16 xor A1 shr 16 xor A2 shl
16 xor
                              A1 shr 24 xor A2 shl  8 xor A2 shr  8 xor A0 shl
24 xor
                              A2 shr 16 xor A0 shl 16 xor A2 shr 24 xor A0 shl
8;
  PIntArray(Data)[1] := A1 xor A1 shr 16 xor A2 shl 16 xor A2 shr 16 xor A0 shl
16 xor
                              A2 shr 24 xor A0 shl  8 xor A0 shr  8 xor A1 shl
24 xor
                              A0 shr 16 xor A1 shl 16 xor A0 shr 24 xor A1 shl
8;
  PIntArray(Data)[2] := A2 xor A2 shr 16 xor A0 shl 16 xor A0 shr 16 xor A1 shl
16 xor
                              A0 shr 24 xor A1 shl  8 xor A1 shr  8 xor A2 shl
24 xor
                              A1 shr 16 xor A2 shl 16 xor A1 shr 24 xor A2 shl
8;
end;

procedure TCipher_3Way.Decode(Data: Pointer);
var
  I: Integer;
  A0,A1,A2: LongWord;
  B0,B1,B2: LongWord;
  K0,K1,K2: LongWord;
  E: PLongWord;
begin
  with P3Way_Key(User)^ do
  begin
    K0 := D_Key[0];
    K1 := D_Key[1];
    K2 := D_Key[2];
    E  := @D_Data;
  end;
  A0 := SwapBits(PIntArray(Data)[2]);
  A1 := SwapBits(PIntArray(Data)[1]);
  A2 := SwapBits(PIntArray(Data)[0]);
  for I := 0 to 10 do
  begin
    A0 := A0 xor K0 xor E^ shl 16;
    A1 := A1 xor K1;
    A2 := A2 xor K2 xor E^;
    Inc(E);

    B0 := A0 xor A0 shr 16 xor A1 shl 16 xor A1 shr 16 xor A2 shl 16 xor
                A1 shr 24 xor A2 shl  8 xor A2 shr  8 xor A0 shl 24 xor
                A2 shr 16 xor A0 shl 16 xor A2 shr 24 xor A0 shl  8;
    B1 := A1 xor A1 shr 16 xor A2 shl 16 xor A2 shr 16 xor A0 shl 16 xor
                A2 shr 24 xor A0 shl  8 xor A0 shr  8 xor A1 shl 24 xor
                A0 shr 16 xor A1 shl 16 xor A0 shr 24 xor A1 shl  8;
    B2 := A2 xor A2 shr 16 xor A0 shl 16 xor A0 shr 16 xor A1 shl 16 xor
                A0 shr 24 xor A1 shl  8 xor A1 shr  8 xor A2 shl 24 xor
                A1 shr 16 xor A2 shl 16 xor A1 shr 24 xor A2 shl  8;
    asm
      ROR B0,10
```

```
      ROL B2,1
    end;
    A0 := B0 xor (B1 or not B2);
    A1 := B1 xor (B2 or not B0);
    A2 := B2 xor (B0 or not B1);
    asm
      ROL A0,1
      ROR A2,10
    end;
  end;
  A0 := A0 xor K0 xor E^ shl 16;
  A1 := A1 xor K1;
  A2 := A2 xor K2 xor E^;
  B0 := A0 xor A0 shr 16 xor A1 shl 16 xor A1 shr 16 xor A2 shl 16 xor
              A1 shr 24 xor A2 shl  8 xor A2 shr  8 xor A0 shl 24 xor
              A2 shr 16 xor A0 shl 16 xor A2 shr 24 xor A0 shl  8;
  B1 := A1 xor A1 shr 16 xor A2 shl 16 xor A2 shr 16 xor A0 shl 16 xor
              A2 shr 24 xor A0 shl  8 xor A0 shr  8 xor A1 shl 24 xor
              A0 shr 16 xor A1 shl 16 xor A0 shr 24 xor A1 shl  8;
  B2 := A2 xor A2 shr 16 xor A0 shl 16 xor A0 shr 16 xor A1 shl 16 xor
              A0 shr 24 xor A1 shl  8 xor A1 shr  8 xor A2 shl 24 xor
              A1 shr 16 xor A2 shl 16 xor A1 shr 24 xor A2 shl  8;

  PIntArray(Data)[2] := SwapBits(B0);
  PIntArray(Data)[1] := SwapBits(B1);
  PIntArray(Data)[0] := SwapBits(B2);
end;

procedure TCipher_3Way.Init(const Key; Size: Integer; IVector: Pointer);

  procedure RANDGenerate(Start: Integer; var P: Array of Integer);
  var
    I: Integer;
  begin
    for I := 0 to 11 do
    begin
      P[I] := Start;
      Start := Start shl 1;
      if Start and $10000 <> 0 then Start := Start xor $11011;
    end;
  end;

var
  A0, A1, A2: Integer;
  B0, B1, B2: Integer;
begin
  InitBegin(Size);
  with P3Way_Key(User)^ do
  begin
    Move(Key, E_Key, Size);
    Move(Key, D_Key, Size);
    RANDGenerate($0B0B, E_Data);
    RANDGenerate($B1B1, D_Data);

    A0 := D_Key[0]; A1 := D_Key[1]; A2 := D_Key[2];
    B0 := A0 xor A0 shr 16 xor A1 shl 16 xor A1 shr 16 xor A2 shl 16 xor
                A1 shr 24 xor A2 shl  8 xor A2 shr  8 xor A0 shl 24 xor
                A2 shr 16 xor A0 shl 16 xor A2 shr 24 xor A0 shl  8;
    B1 := A1 xor A1 shr 16 xor A2 shl 16 xor A2 shr 16 xor A0 shl 16 xor
                A2 shr 24 xor A0 shl  8 xor A0 shr  8 xor A1 shl 24 xor
                A0 shr 16 xor A1 shl 16 xor A0 shr 24 xor A1 shl  8;
    B2 := A2 xor A2 shr 16 xor A0 shl 16 xor A0 shr 16 xor A1 shl 16 xor
                A0 shr 24 xor A1 shl  8 xor A1 shr  8 xor A2 shl 24 xor
                A1 shr 16 xor A2 shl 16 xor A1 shr 24 xor A2 shl  8;
    D_Key[2] := SwapBits(B0); D_Key[1] := SwapBits(B1); D_Key[0] :=
SwapBits(B2);
  end;
  InitEnd(IVector);
end;
```

```
class procedure TCipher_Twofish.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  ABufSize := 16;
  AKeySize := 32;
  AUserSize := 4256;
end;

class function TCipher_Twofish.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    0A5h,053h,057h,003h,0EFh,033h,048h,079h
         DB    09Fh,022h,0B4h,054h,097h,005h,084h,019h
         DB    087h,0BDh,083h,01Ch,04Dh,0AEh,012h,013h
         DB    060h,07Ch,07Ch,0D1h,098h,045h,002h,019h
end;

type
  PTwofishBox = ^TTwofishBox;
  TTwofishBox = array[0..3, 0..255] of Longword;

  TLongRec = record
               case Integer of
                 0: (L: Longword);
                 1: (A,B,C,D: Byte);
             end;

procedure TCipher_Twofish.Encode(Data: Pointer);
var
  S: PIntArray;
  Box: PTwofishBox;
  I,X,Y: LongWord;
  A,B,C,D: TLongRec;
begin
  S   := User;
  A.L := PIntArray(Data)[0] xor S[0];
  B.L := PIntArray(Data)[1] xor S[1];
  C.L := PIntArray(Data)[2] xor S[2];
  D.L := PIntArray(Data)[3] xor S[3];

  S   := @PIntArray(User)[8];
  Box := @PIntArray(User)[40];
  for I := 0 to 7 do
  begin
    X := Box[0, A.A] xor Box[1, A.B] xor Box[2, A.C] xor Box[3, A.D];
    Y := Box[1, B.A] xor Box[2, B.B] xor Box[3, B.C] xor Box[0, B.D];
    asm ROL  D.L,1 end;
    C.L := C.L xor (X + Y       + S[0]);
    D.L := D.L xor (X + Y shl 1 + S[1]);
    asm ROR  C.L,1 end;

    X := Box[0, C.A] xor Box[1, C.B] xor Box[2, C.C] xor Box[3, C.D];
    Y := Box[1, D.A] xor Box[2, D.B] xor Box[3, D.C] xor Box[0, D.D];
    asm ROL  B.L,1 end;
    A.L := A.L xor (X + Y       + S[2]);
    B.L := B.L xor (X + Y shl 1 + S[3]);
    asm ROR  A.L,1 end;
    Inc(PInteger(S), 4);
  end;
  S := User;
  PIntArray(Data)[0] := C.L xor S[4];
  PIntArray(Data)[1] := D.L xor S[5];
  PIntArray(Data)[2] := A.L xor S[6];
  PIntArray(Data)[3] := B.L xor S[7];
end;

procedure TCipher_Twofish.Decode(Data: Pointer);
```

```
var
  S: PIntArray;
  Box: PTwofishBox;
  I,X,Y: LongWord;
  A,B,C,D: TLongRec;
begin
  S := User;
  Box := @PIntArray(User)[40];
  C.L := PIntArray(Data)[0] xor S[4];
  D.L := PIntArray(Data)[1] xor S[5];
  A.L := PIntArray(Data)[2] xor S[6];
  B.L := PIntArray(Data)[3] xor S[7];
  S := @PIntArray(User)[36];
  for I := 0 to 7 do
  begin
    X := Box[0, C.A] xor Box[1, C.B] xor Box[2, C.C] xor Box[3, C.D];
    Y := Box[0, D.D] xor Box[1, D.A] xor Box[2, D.B] xor Box[3, D.C];
    asm ROL  A.L,1 end;
    B.L := B.L xor (X + Y shl 1 + S[3]);
    A.L := A.L xor (X + Y       + S[2]);
    asm ROR  B.L,1 end;

    X := Box[0, A.A] xor Box[1, A.B] xor Box[2, A.C] xor Box[3, A.D];
    Y := Box[0, B.D] xor Box[1, B.A] xor Box[2, B.B] xor Box[3, B.C];
    asm ROL  C.L,1 end;
    D.L := D.L xor (X + Y shl 1 + S[1]);
    C.L := C.L xor (X + Y       + S[0]);
    asm ROR  D.L,1 end;
    Dec(PByte(S),16);
  end;
  S := User;
  PIntArray(Data)[0] := A.L xor S[0];
  PIntArray(Data)[1] := B.L xor S[1];
  PIntArray(Data)[2] := C.L xor S[2];
  PIntArray(Data)[3] := D.L xor S[3];
end;

procedure TCipher_Twofish.Init(const Key; Size: Integer; IVector: Pointer);
var
  BoxKey: array[0..3] of TLongRec;
  SubKey: PIntArray;
  Box: PTwofishBox;

  procedure SetupKey;

    function Encode(K0, K1: Integer): Integer;
    var
      R, I, J, G2, G3: Integer;
      B: byte;
    begin
      R := 0;
      for I := 0 to 1 do
      begin
        if I <> 0 then R := R xor K0 else R := R xor K1;
        for J := 0 to 3 do
        begin
          B := R shr 24;
          if B and $80 <> 0 then G2 := (B shl 1 xor $014D) and $FF
            else G2 := B shl 1 and $FF;
          if B and 1 <> 0 then G3 := (B shr 1 and $7F) xor $014D shr 1 xor G2
            else G3 := (B shr 1 and $7F) xor G2;
          R := R shl 8 xor G3 shl 24 xor G2 shl 16 xor G3 shl 8 xor B;
        end;
      end;
      Result := R;
    end;

    function F32(X: Integer; K: array of Integer): Integer;
    var
```

```
    A, B, C, D: Integer;
  begin
    A := X and $FF;
    B := X shr  8 and $FF;
    C := X shr 16 and $FF;
    D := X shr 24;
    if Size = 32 then
    begin
      A := Twofish_8x8[1, A] xor K[3] and $FF;
      B := Twofish_8x8[0, B] xor K[3] shr  8 and $FF;
      C := Twofish_8x8[0, C] xor K[3] shr 16 and $FF;
      D := Twofish_8x8[1, D] xor K[3] shr 24;
    end;
    if Size >= 24 then
    begin
      A := Twofish_8x8[1, A] xor K[2] and $FF;
      B := Twofish_8x8[1, B] xor K[2] shr  8 and $FF;
      C := Twofish_8x8[0, C] xor K[2] shr 16 and $FF;
      D := Twofish_8x8[0, D] xor K[2] shr 24;
    end;
    A := Twofish_8x8[0, A] xor K[1] and $FF;
    B := Twofish_8x8[1, B] xor K[1] shr  8 and $FF;
    C := Twofish_8x8[0, C] xor K[1] shr 16 and $FF;
    D := Twofish_8x8[1, D] xor K[1] shr 24;

    A := Twofish_8x8[0, A] xor K[0] and $FF;
    B := Twofish_8x8[0, B] xor K[0] shr  8 and $FF;
    C := Twofish_8x8[1, C] xor K[0] shr 16 and $FF;
    D := Twofish_8x8[1, D] xor K[0] shr 24;

    Result := Twofish_Data[0, A] xor Twofish_Data[1, B] xor
              Twofish_Data[2, C] xor Twofish_Data[3, D];
  end;

var
  I,J,A,B: Integer;
  E,O: array[0..3] of Integer;
  K: array[0..7] of Integer;
begin
  FillChar(K, SizeOf(K), 0);
  Move(Key, K, Size);
  if Size <= 16 then Size := 16 else
    if Size <= 24 then Size := 24
      else Size := 32;
  J := Size shr 3 - 1;
  for I := 0 to J do
  begin
    E[I] := K[I shl 1];
    O[I] := K[I shl 1 + 1];
    BoxKey[J].L := Encode(E[I], O[I]);
    Dec(J);
  end;
  J := 0;
  for I := 0 to 19 do
  begin
    A := F32(J, E);
    B := ROL(F32(J + $01010101, O), 8);
    SubKey[I shl 1] := A + B;
    B := A + B shr 1;
    SubKey[I shl 1 + 1] := ROL(B, 9);
    Inc(J, $02020202);
  end;
end;

procedure DoXOR(D, S: PIntArray; Value: LongWord);
var
  I: LongWord;
begin
  Value := (Value and $FF) * $01010101;
```

```
    for I := 0 to 63 do D[I] := S[I] xor Value;
  end;

  procedure SetupBox128;
  var
    L: array[0..255] of Byte;
    A,I: Integer;
  begin
    DoXOR(@L, @Twofish_8x8[0], BoxKey[1].L);
    A := BoxKey[0].A;
    for I := 0 to 255 do
      Box[0, I] := Twofish_Data[0, Twofish_8x8[0, L[I]] xor A];
    DoXOR(@L, @Twofish_8x8[1], BoxKey[1].L shr 8);
    A := BoxKey[0].B;
    for I := 0 to 255 do
      Box[1, I] := Twofish_Data[1, Twofish_8x8[0, L[I]] xor A];
    DoXOR(@L, @Twofish_8x8[0], BoxKey[1].L shr 16);
    A := BoxKey[0].C;
    for I := 0 to 255 do
      Box[2, I] := Twofish_Data[2, Twofish_8x8[1, L[I]] xor A];
    DoXOR(@L, @Twofish_8x8[1], BoxKey[1].L shr 24);
    A := BoxKey[0].D;
    for I := 0 to 255 do
      Box[3, I] := Twofish_Data[3, Twofish_8x8[1, L[I]] xor A];
  end;

  procedure SetupBox192;
  var
    L: array[0..255] of Byte;
    A,B,I: Integer;
  begin
    DoXOR(@L, @Twofish_8x8[1], BoxKey[2].L);
    A := BoxKey[0].A;
    B := BoxKey[1].A;
    for I := 0 to 255 do
      Box[0, I] := Twofish_Data[0, Twofish_8x8[0, Twofish_8x8[0, L[I]] xor B]
xor A];
    DoXOR(@L, @Twofish_8x8[1], BoxKey[2].L shr 8);
    A := BoxKey[0].B;
    B := BoxKey[1].B;
    for I := 0 to 255 do
      Box[1, I] := Twofish_Data[1, Twofish_8x8[0, Twofish_8x8[1, L[I]] xor B]
xor A];
    DoXOR(@L, @Twofish_8x8[0], BoxKey[2].L shr 16);
    A := BoxKey[0].C;
    B := BoxKey[1].C;
    for I := 0 to 255 do
      Box[2, I] := Twofish_Data[2, Twofish_8x8[1, Twofish_8x8[0, L[I]] xor B]
xor A];
    DoXOR(@L ,@Twofish_8x8[0], BoxKey[2].L shr 24);
    A := BoxKey[0].D;
    B := BoxKey[1].D;
    for I := 0 to 255 do
      Box[3, I] := Twofish_Data[3, Twofish_8x8[1, Twofish_8x8[1, L[I]] xor B]
xor A];
  end;

  procedure SetupBox256;
  var
    L: array[0..255] of Byte;
    K: array[0..255] of Byte;
    A,B,I: Integer;
  begin
    DoXOR(@K, @Twofish_8x8[1], BoxKey[3].L);
    for I := 0 to 255 do L[I] := Twofish_8x8[1, K[I]];
    DoXOR(@L, @L, BoxKey[2].L);
    A := BoxKey[0].A;
    B := BoxKey[1].A;
    for I := 0 to 255 do
```

```
      Box[0, I] := Twofish_Data[0, Twofish_8x8[0, Twofish_8x8[0, L[I]] xor B]
xor A];
    DoXOR(@K, @Twofish_8x8[0], BoxKey[3].L shr 8);
    for I := 0 to 255 do L[I] := Twofish_8x8[1, K[I]];
    DoXOR(@L, @L, BoxKey[2].L shr 8);
    A := BoxKey[0].B;
    B := BoxKey[1].B;
    for I := 0 to 255 do
      Box[1, I] := Twofish_Data[1, Twofish_8x8[0, Twofish_8x8[1, L[I]] xor B]
xor A];
    DoXOR(@K, @Twofish_8x8[0],BoxKey[3].L shr 16);
    for I := 0 to 255 do L[I] := Twofish_8x8[0, K[I]];
    DoXOR(@L, @L, BoxKey[2].L shr 16);
    A := BoxKey[0].C;
    B := BoxKey[1].C;
    for I := 0 to 255 do
      Box[2, I] := Twofish_Data[2, Twofish_8x8[1, Twofish_8x8[0, L[I]] xor B]
xor A];
    DoXOR(@K, @Twofish_8x8[1], BoxKey[3].L shr 24);
    for I := 0 to 255 do L[I] := Twofish_8x8[0, K[I]];
    DoXOR(@L, @L, BoxKey[2].L shr 24);
    A := BoxKey[0].D;
    B := BoxKey[1].D;
    for I := 0 to 255 do
      Box[3, I] := Twofish_Data[3, Twofish_8x8[1, Twofish_8x8[1, L[I]] xor B]
xor A];
  end;

begin
  InitBegin(Size);
  SubKey := User;
  Box    := @SubKey[40];
  SetupKey;
  if Size = 16 then SetupBox128 else
    if Size = 24 then SetupBox192
      else SetupBox256;
  InitEnd(IVector);
end;

class procedure TCipher_Shark.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  ABufSize := 8;
  AKeySize := 16;
  AUserSize := 112;
end;

class function TCipher_Shark.TestVector: Pointer;
asm
        MOV   EAX,OFFSET @Vector
        RET
@Vector: DB    0D9h,065h,021h,0AAh,0C0h,0C3h,084h,060h
        DB    09Dh,0CEh,01Fh,08Bh,0FBh,0ABh,018h,03Fh
        DB    0A1h,021h,0ACh,0F8h,053h,049h,0C0h,06Fh
        DB    027h,03Ah,089h,015h,0D3h,07Ah,0E9h,00Bh
end;

{$IFDEF VER_D4H} // >= D4
  {.$DEFINE Shark64}
{$ENDIF}

type
  PInt64        = ^TInt64;
{$IFDEF Shark64}
  TInt64        = Int64;
{$ELSE}
  TInt64        = packed record
                    L,R: Integer;
                  end;
```

```
{$ENDIF}

  PInt64Array = ^TInt64Array;
  TInt64Array = array[0..1023] of TInt64;

{$IFDEF Shark64}
  TShark_Data = array[0..7, 0..255] of Int64;
{$ENDIF}

procedure TCipher_Shark.Encode(Data: Pointer);
var
  I,T: Integer;
{$IFDEF Shark64}
  D: TInt64;
  K: PInt64;
{$ELSE}
  L,R: LongWord;
  K: PIntArray;
{$ENDIF}
begin
  K := User;
{$IFDEF Shark64}
  D := PInt64(Data)^;
  for I := 0 to 4 do
  begin
    D := D xor K^; Inc(K);
    D := TShark_Data(Shark_CE)[0, D shr 56 and $FF] xor
         TShark_Data(Shark_CE)[1, D shr 48 and $FF] xor
         TShark_Data(Shark_CE)[2, D shr 40 and $FF] xor
         TShark_Data(Shark_CE)[3, D shr 32 and $FF] xor
         TShark_Data(Shark_CE)[4, D shr 24 and $FF] xor
         TShark_Data(Shark_CE)[5, D shr 16 and $FF] xor
         TShark_Data(Shark_CE)[6, D shr  8 and $FF] xor
         TShark_Data(Shark_CE)[7, D        and $FF];
  end;
  D := D xor K^; Inc(K);
  D := (Int64(Shark_SE[D shr 56 and $FF]) shl 56) xor
       (Int64(Shark_SE[D shr 48 and $FF]) shl 48) xor
       (Int64(Shark_SE[D shr 40 and $FF]) shl 40) xor
       (Int64(Shark_SE[D shr 32 and $FF]) shl 32) xor
       (Int64(Shark_SE[D shr 24 and $FF]) shl 24) xor
       (Int64(Shark_SE[D shr 16 and $FF]) shl 16) xor
       (Int64(Shark_SE[D shr  8 and $FF]) shl  8) xor
       (Int64(Shark_SE[D        and $FF]));
  PInt64(Data)^ := D xor K^;
{$ELSE}
  L := PInt64(Data).L;
  R := PInt64(Data).R;
  for I := 0 to 4 do
  begin
    L := L xor K[0];
    R := R xor K[1];
    Inc(PInteger(K), 2);
    T := Shark_CE[0, R shr 23 and $1FE] xor
         Shark_CE[1, R shr 15 and $1FE] xor
         Shark_CE[2, R shr  7 and $1FE] xor
         Shark_CE[3, R shl  1 and $1FE] xor
         Shark_CE[4, L shr 23 and $1FE] xor
         Shark_CE[5, L shr 15 and $1FE] xor
         Shark_CE[6, L shr  7 and $1FE] xor
         Shark_CE[7, L shl  1 and $1FE];
    R := Shark_CE[0, R shr 23 and $1FE or 1] xor
         Shark_CE[1, R shr 15 and $1FE or 1] xor
         Shark_CE[2, R shr  7 and $1FE or 1] xor
         Shark_CE[3, R shl  1 and $1FE or 1] xor
         Shark_CE[4, L shr 23 and $1FE or 1] xor
         Shark_CE[5, L shr 15 and $1FE or 1] xor
         Shark_CE[6, L shr  7 and $1FE or 1] xor
         Shark_CE[7, L shl  1 and $1FE or 1];
```

```
    L := T;
  end;
  L := L xor K[0];
  R := R xor K[1];
  Inc(PInteger(K), 2);
  L := LongWord(Shark_SE[L shr 24        ]) shl 24 xor
       LongWord(Shark_SE[L shr 16 and $FF]) shl 16 xor
       LongWord(Shark_SE[L shr  8 and $FF]) shl  8 xor
       LongWord(Shark_SE[L        and $FF]);
  R := LongWord(Shark_SE[R shr 24        ]) shl 24 xor
       LongWord(Shark_SE[R shr 16 and $FF]) shl 16 xor
       LongWord(Shark_SE[R shr  8 and $FF]) shl  8 xor
       LongWord(Shark_SE[R        and $FF]);
  PInt64(Data).L := L xor K[0];
  PInt64(Data).R := R xor K[1];
{$ENDIF}
end;

procedure TCipher_Shark.Decode(Data: Pointer);
var
  I,T: Integer;
{$IFDEF Shark64}
  D: TInt64;
  K: PInt64;
{$ELSE}
  R,L: LongWord;
  K: PIntArray;
{$ENDIF}
begin
  K := User;
{$IFDEF Shark64}
  Inc(K, 7);
  D := PInt64(Data)^;
  for I := 0 to 4 do
  begin
    D := D xor K^; Inc(K);
    D := TShark_Data(Shark_CD)[0, D shr 56 and $FF] xor
         TShark_Data(Shark_CD)[1, D shr 48 and $FF] xor
         TShark_Data(Shark_CD)[2, D shr 40 and $FF] xor
         TShark_Data(Shark_CD)[3, D shr 32 and $FF] xor

         TShark_Data(Shark_CD)[4, D shr 24 and $FF] xor
         TShark_Data(Shark_CD)[5, D shr 16 and $FF] xor
         TShark_Data(Shark_CD)[6, D shr  8 and $FF] xor
         TShark_Data(Shark_CD)[7, D        and $FF];
  end;
  D := D xor K^; Inc(K);
  D := (Int64(Shark_SD[D shr 56 and $FF]) shl 56) xor
       (Int64(Shark_SD[D shr 48 and $FF]) shl 48) xor
       (Int64(Shark_SD[D shr 40 and $FF]) shl 40) xor
       (Int64(Shark_SD[D shr 32 and $FF]) shl 32) xor
       (Int64(Shark_SD[D shr 24 and $FF]) shl 24) xor
       (Int64(Shark_SD[D shr 16 and $FF]) shl 16) xor
       (Int64(Shark_SD[D shr  8 and $FF]) shl  8) xor
       (Int64(Shark_SD[D        and $FF]));
  PInt64(Data)^ := D xor K^;
{$ELSE}
  Inc(PInteger(K), 14);
  L := PInt64(Data).L;
  R := PInt64(Data).R;
  for I := 0 to 4 do
  begin
    L := L xor K[0];
    R := R xor K[1];
    Inc(PInteger(K), 2);
    T := Shark_CD[0, R shr 23 and $1FE] xor
         Shark_CD[1, R shr 15 and $1FE] xor
         Shark_CD[2, R shr  7 and $1FE] xor
         Shark_CD[3, R shl  1 and $1FE] xor
```

```
          Shark_CD[4, L shr 23 and $1FE] xor
          Shark_CD[5, L shr 15 and $1FE] xor
          Shark_CD[6, L shr  7 and $1FE] xor
          Shark_CD[7, L shl  1 and $1FE];
      R := Shark_CD[0, R shr 23 and $1FE or 1] xor
          Shark_CD[1, R shr 15 and $1FE or 1] xor
          Shark_CD[2, R shr  7 and $1FE or 1] xor
          Shark_CD[3, R shl  1 and $1FE or 1] xor
          Shark_CD[4, L shr 23 and $1FE or 1] xor
          Shark_CD[5, L shr 15 and $1FE or 1] xor
          Shark_CD[6, L shr  7 and $1FE or 1] xor
          Shark_CD[7, L shl  1 and $1FE or 1];
      L := T;
    end;
    L := L xor K[0];
    R := R xor K[1];
    Inc(PInteger(K), 2);
    L := Integer(Shark_SD[L shr 24        ]) shl 24 xor
         Integer(Shark_SD[L shr 16 and $FF]) shl 16 xor
         Integer(Shark_SD[L shr  8 and $FF]) shl  8 xor
         Integer(Shark_SD[L        and $FF]);
    R := Integer(Shark_SD[R shr 24        ]) shl 24 xor
         Integer(Shark_SD[R shr 16 and $FF]) shl 16 xor
         Integer(Shark_SD[R shr  8 and $FF]) shl  8 xor
         Integer(Shark_SD[R        and $FF]);
    PInt64(Data).L := L xor K[0];
    PInt64(Data).R := R xor K[1];
  {$ENDIF}
  end;

  procedure TCipher_Shark.Init(const Key; Size: Integer; IVector: Pointer);
  var
    Log, ALog: array[0..255] of Byte;

    procedure InitLog;
    var
      I, J: Word;
    begin
      ALog[0] := 1;
      for I := 1 to 255 do
      begin
        J := ALog[I-1] shl 1;
        if J and $100 <> 0 then J := J xor $01F5;
        ALog[I] := J;
      end;
      for I := 1 to 254 do Log[ALog[I]] := I;
    end;

    function Transform(A: TInt64): TInt64;
    type
      TInt64Rec = packed record
                    Lo, Hi: Integer;
                  end;

      function Mul(A, B: Integer): Byte;
      begin
        Result := ALog[(Log[A] + Log[B]) mod 255];
      end;

    var
      I,J: Byte;
      K,T: array[0..7] of Byte;
    begin
  {$IFDEF Shark64}
      Move(TInt64Rec(A).Hi, K[0], 4);
      Move(TInt64Rec(A).Lo, K[4], 4);
      SwapIntegerBuffer(@K, @K, 2);
  {$ELSE}
      Move(A.R, K[0], 4);
```

```
    Move(A.L, K[4], 4);
    SwapIntegerBuffer(@K, @K, 2);
{$ENDIF}
    for I := 0 to 7 do
    begin
      T[I] := Mul(Shark_I[I, 0], K[0]);
      for J := 1 to 7 do T[I] := T[I] xor Mul(Shark_I[I, J], K[J]);
    end;
{$IFDEF Shark64}
    Result := T[0];
    for I := 1 to 7 do Result := Result shl 8 xor T[I];
{$ELSE}
    Result.L := T[0];
    Result.R := 0;
    for I := 1 to 7 do
    begin
      Result.R := Result.R shl 8 or Result.L shr 24;
      Result.L := Result.L shl 8 xor T[I];
    end;
{$ENDIF}
  end;

  function Shark(D: TInt64; K: PInt64): TInt64;
  var
    R,T: Integer;
  begin
{$IFDEF Shark64}
    for R := 0 to 4 do
    begin
      D := D xor K^; Inc(K);
      D := TShark_Data(Shark_CE)[0, D shr 56 and $FF] xor
           TShark_Data(Shark_CE)[1, D shr 48 and $FF] xor
           TShark_Data(Shark_CE)[2, D shr 40 and $FF] xor
           TShark_Data(Shark_CE)[3, D shr 32 and $FF] xor
           TShark_Data(Shark_CE)[4, D shr 24 and $FF] xor
           TShark_Data(Shark_CE)[5, D shr 16 and $FF] xor
           TShark_Data(Shark_CE)[6, D shr  8 and $FF] xor
           TShark_Data(Shark_CE)[7, D        and $FF];
    end;
    D := D xor K^; Inc(K);
    D := (Int64(Shark_SE[D shr 56 and $FF]) shl 56) xor
         (Int64(Shark_SE[D shr 48 and $FF]) shl 48) xor
         (Int64(Shark_SE[D shr 40 and $FF]) shl 40) xor
         (Int64(Shark_SE[D shr 32 and $FF]) shl 32) xor
         (Int64(Shark_SE[D shr 24 and $FF]) shl 24) xor
         (Int64(Shark_SE[D shr 16 and $FF]) shl 16) xor
         (Int64(Shark_SE[D shr  8 and $FF]) shl  8) xor
         (Int64(Shark_SE[D        and $FF]));
    Result := D xor K^;
{$ELSE}
    for R := 0 to 4 do
    begin
      D.L := D.L xor K.L;
      D.R := D.R xor K.R;
      Inc(K);
      T   := Shark_CE[0, D.R shr 23 and $1FE] xor
             Shark_CE[1, D.R shr 15 and $1FE] xor
             Shark_CE[2, D.R shr  7 and $1FE] xor
             Shark_CE[3, D.R shl  1 and $1FE] xor
             Shark_CE[4, D.L shr 23 and $1FE] xor
             Shark_CE[5, D.L shr 15 and $1FE] xor
             Shark_CE[6, D.L shr  7 and $1FE] xor
             Shark_CE[7, D.L shl  1 and $1FE];

      D.R := Shark_CE[0, D.R shr 23 and $1FE or 1] xor
             Shark_CE[1, D.R shr 15 and $1FE or 1] xor
             Shark_CE[2, D.R shr  7 and $1FE or 1] xor
             Shark_CE[3, D.R shl  1 and $1FE or 1] xor
             Shark_CE[4, D.L shr 23 and $1FE or 1] xor
```

```
              Shark_CE[5, D.L shr 15 and $1FE or 1] xor
              Shark_CE[6, D.L shr  7 and $1FE or 1] xor
              Shark_CE[7, D.L shl  1 and $1FE or 1];
      D.L := T;
    end;
    D.L := D.L xor K.L;
    D.R := D.R xor K.R;
    Inc(K);
    D.L := Integer(Shark_SE[D.L shr 24 and $FF]) shl 24 xor
           Integer(Shark_SE[D.L shr 16 and $FF]) shl 16 xor
           Integer(Shark_SE[D.L shr  8 and $FF]) shl  8 xor
           Integer(Shark_SE[D.L        and $FF]);
    D.R := Integer(Shark_SE[D.R shr 24 and $FF]) shl 24 xor
           Integer(Shark_SE[D.R shr 16 and $FF]) shl 16 xor
           Integer(Shark_SE[D.R shr  8 and $FF]) shl  8 xor
           Integer(Shark_SE[D.R        and $FF]);
    Result.L := D.L xor K.L;
    Result.R := D.R xor K.R;
{$ENDIF}
  end;

var
  T: array[0..6] of TInt64;
  A: array[0..6] of TInt64;
  K: array[0..15] of Byte;
  I,J,R: Byte;
  E,D: PInt64Array;
  L: TInt64;
begin
  InitBegin(Size);
  FillChar(K, SizeOf(K), 0);
  Move(Key, K, Size);
  InitLog;
  E := User;
  D := @E[7];
  Move(Shark_CE[0], T, SizeOf(T));
  T[6] := Transform(T[6]);
  I := 0;
{$IFDEF Shark64}
  for R := 0 to 6 do
  begin
    Inc(I);
    A[R] := K[I and $F];
    for J := 1 to 7 do
    begin
      Inc(I);
      A[R] := A[R] shl 8 or K[I and $F];
    end;
  end;
  E[0] := A[0] xor Shark(0, @T);
  for R := 1 to 6 do E[R] := A[R] xor Shark(E[R - 1], @T);
{$ELSE}
  for R := 0 to 6 do
  begin
    Inc(I);
    A[R].L := K[I and $F];
    A[R].R := 0;
    for J := 1 to 7 do
    begin
      Inc(I);
      A[R].R := A[R].R shl 8 or A[R].L shr 24;
      A[R].L := A[R].L shl 8 or K[I and $F];
    end;
  end;
  L.L := 0;
  L.R := 0;
  L := Shark(L, @T);
  E[0].L := A[0].L xor L.L;
  E[0].R := A[0].R xor L.R;
```

```
  for R := 1 to 6 do
  begin
    L := Shark(E[R - 1], @T);
    E[R].L := A[R].L xor L.L;
    E[R].R := A[R].R xor L.R;
  end;
{$ENDIF}

  E[6] := Transform(E[6]);
  D[0] := E[6];
  D[6] := E[0];
  for R := 1 to 5 do D[R] := Transform(E[6-R]);

  FillChar(Log, SizeOf(Log), 0);
  FillChar(ALog, SizeOf(ALog), 0);
  FillChar(T, SizeOf(T), 0);
  FillChar(A, SizeOf(A), 0);
  FillChar(K, SizeOf(K), 0);
  InitEnd(IVector);
end;

class procedure TCipher_Square.GetContext(var ABufSize, AKeySize, AUserSize:
Integer);
begin
  ABufSize := 16;
  AKeySize := 16;
  AUserSize := 9 * 4 * 2 * SizeOf(LongWord);
end;

class function TCipher_Square.TestVector: Pointer;
asm
         MOV   EAX,OFFSET @Vector
         RET
@Vector: DB    043h,09Ch,0A6h,0C4h,067h,0E8h,02Eh,047h
         DB    022h,095h,066h,085h,006h,039h,06Ah,0C9h
         DB    018h,021h,020h,0F7h,044h,036h,0F1h,061h
         DB    07Dh,014h,090h,0B1h,0A9h,068h,056h,0C7h
end;

procedure TCipher_Square.Encode(Data: Pointer);
var
  Key: PIntArray;
  A,B,C,D: LongWord;
  AA,BB,CC: LongWord;
  I: Integer;
begin
  Key := User;
  A := PIntArray(Data)[0] xor Key[0];
  B := PIntArray(Data)[1] xor Key[1];
  C := PIntArray(Data)[2] xor Key[2];
  D := PIntArray(Data)[3] xor Key[3];
  Inc(PInteger(Key), 4);
  for I := 0 to 6 do
  begin
    AA := Square_TE[0, A        and $FF] xor
          Square_TE[1, B        and $FF] xor
          Square_TE[2, C        and $FF] xor
          Square_TE[3, D        and $FF] xor Key[0];
    BB := Square_TE[0, A shr  8 and $FF] xor
          Square_TE[1, B shr  8 and $FF] xor
          Square_TE[2, C shr  8 and $FF] xor
          Square_TE[3, D shr  8 and $FF] xor Key[1];
    CC := Square_TE[0, A shr 16 and $FF] xor
          Square_TE[1, B shr 16 and $FF] xor
          Square_TE[2, C shr 16 and $FF] xor
          Square_TE[3, D shr 16 and $FF] xor Key[2];
    D  := Square_TE[0, A shr 24        ] xor
          Square_TE[1, B shr 24        ] xor
          Square_TE[2, C shr 24        ] xor
```

```
          Square_TE[3, D shr 24        ] xor Key[3];

    Inc(PInteger(Key), 4);

    A := AA; B := BB; C := CC;
  end;

  PIntArray(Data)[0] := LongWord(Square_SE[A        and $FF])        xor
                        LongWord(Square_SE[B        and $FF]) shl  8 xor
                        LongWord(Square_SE[C        and $FF]) shl 16 xor
                        LongWord(Square_SE[D        and $FF]) shl 24 xor Key[0];
  PIntArray(Data)[1] := LongWord(Square_SE[A shr  8 and $FF])        xor
                        LongWord(Square_SE[B shr  8 and $FF]) shl  8 xor
                        LongWord(Square_SE[C shr  8 and $FF]) shl 16 xor
                        LongWord(Square_SE[D shr  8 and $FF]) shl 24 xor Key[1];
  PIntArray(Data)[2] := LongWord(Square_SE[A shr 16 and $FF])        xor
                        LongWord(Square_SE[B shr 16 and $FF]) shl  8 xor
                        LongWord(Square_SE[C shr 16 and $FF]) shl 16 xor
                        LongWord(Square_SE[D shr 16 and $FF]) shl 24 xor Key[2];
  PIntArray(Data)[3] := LongWord(Square_SE[A shr 24        ])        xor
                        LongWord(Square_SE[B shr 24        ]) shl  8 xor
                        LongWord(Square_SE[C shr 24        ]) shl 16 xor
                        LongWord(Square_SE[D shr 24        ]) shl 24 xor Key[3];
end;

procedure TCipher_Square.Decode(Data: Pointer);
var
  Key: PIntArray;
  A,B,C,D: LongWord;
  AA,BB,CC: LongWord;
  I: Integer;
begin
  Key := @PIntArray(User)[9 * 4];
  A := PIntArray(Data)[0] xor Key[0];
  B := PIntArray(Data)[1] xor Key[1];
  C := PIntArray(Data)[2] xor Key[2];
  D := PIntArray(Data)[3] xor Key[3];
  Inc(PInteger(Key), 4);

  for I := 0 to 6 do
  begin
    AA := Square_TD[0, A        and $FF] xor
          Square_TD[1, B        and $FF] xor
          Square_TD[2, C        and $FF] xor
          Square_TD[3, D        and $FF] xor Key[0];
    BB := Square_TD[0, A shr  8 and $FF] xor
          Square_TD[1, B shr  8 and $FF] xor
          Square_TD[2, C shr  8 and $FF] xor
          Square_TD[3, D shr  8 and $FF] xor Key[1];
    CC := Square_TD[0, A shr 16 and $FF] xor
          Square_TD[1, B shr 16 and $FF] xor
          Square_TD[2, C shr 16 and $FF] xor
          Square_TD[3, D shr 16 and $FF] xor Key[2];
    D  := Square_TD[0, A shr 24        ] xor
          Square_TD[1, B shr 24        ] xor
          Square_TD[2, C shr 24        ] xor
          Square_TD[3, D shr 24        ] xor Key[3];

    Inc(PInteger(Key), 4);
    A := AA; B := BB; C := CC;
  end;

  PIntArray(Data)[0] := LongWord(Square_SD[A        and $FF])        xor
                        LongWord(Square_SD[B        and $FF]) shl  8 xor
                        LongWord(Square_SD[C        and $FF]) shl 16 xor
                        LongWord(Square_SD[D        and $FF]) shl 24 xor Key[0];
  PIntArray(Data)[1] := LongWord(Square_SD[A shr  8 and $FF])        xor
                        LongWord(Square_SD[B shr  8 and $FF]) shl  8 xor
                        LongWord(Square_SD[C shr  8 and $FF]) shl 16 xor
```

```
                        LongWord(Square_SD[D shr  8 and $FF]) shl 24 xor Key[1];
  PIntArray(Data)[2] := LongWord(Square_SD[A shr 16 and $FF])        xor
                        LongWord(Square_SD[B shr 16 and $FF]) shl  8 xor
                        LongWord(Square_SD[C shr 16 and $FF]) shl 16 xor
                        LongWord(Square_SD[D shr 16 and $FF]) shl 24 xor Key[2];
  PIntArray(Data)[3] := LongWord(Square_SD[A shr 24        ])        xor
                        LongWord(Square_SD[B shr 24        ]) shl  8 xor
                        LongWord(Square_SD[C shr 24        ]) shl 16 xor
                        LongWord(Square_SD[D shr 24        ]) shl 24 xor Key[3];
end;

procedure TCipher_Square.Init(const Key; Size: Integer; IVector: Pointer);
type
  PSquare_Key = ^TSquare_Key;
  TSquare_Key = array[0..8, 0..3] of LongWord;
var
  E,D: PSquare_Key;
  T,I: Integer;
begin
  InitBegin(Size);
  E := User;
  D := User; Inc(D);
  Move(Key, E^, Size);
  for T := 1 to 8 do
  begin
    E[T, 0] := E[T -1, 0] xor ROR(E[T -1, 3], 8) xor 1 shl (T - 1); D[8 -T, 0]
:= E[T, 0];
    E[T, 1] := E[T -1, 1] xor E[T, 0];                              D[8 -T, 1]
:= E[T, 1];
    E[T, 2] := E[T -1, 2] xor E[T, 1];                              D[8 -T, 2]
:= E[T, 2];
    E[T, 3] := E[T -1, 3] xor E[T, 2];                              D[8 -T, 3]
:= E[T, 3];
    for I := 0 to 3 do
      E[T -1, I] :=     Square_PHI[E[T -1, I]        and $FF]      xor
                    ROL(Square_PHI[E[T -1, I] shr  8 and $FF],  8) xor
                    ROL(Square_PHI[E[T -1, I] shr 16 and $FF], 16) xor
                    ROL(Square_PHI[E[T -1, I] shr 24        ], 24);
  end;
  D[8] := E[0];
  InitEnd(IVector);
end;

{$IFDEF UseASM}
  {$IFNDEF 486GE}  // не використовується для <= CPU 386

procedure FindVirtualMethodAndChange(AClass: TClass; MethodAddr, NewAddress:
Pointer);
// MethodAddr повинно явно існувати
type
  PPointer = ^Pointer;
const
  PageSize = SizeOf(Pointer);
var
  Table: PPointer;
  SaveFlag: DWORD;
begin
  Table := PPointer(AClass);
  while Table^ <> MethodAddr do Inc(Table);
  if VirtualProtect(Table, PageSize, PAGE_EXECUTE_READWRITE, @SaveFlag) then
  try
    Table^ := NewAddress;
  finally
    VirtualProtect(Table, PageSize, SaveFlag, @SaveFlag);
  end;
end;
  {$ENDIF}
{$ENDIF}
```

```
{$IFDEF VER_D3H}
procedure ModuleUnload(Module: Integer);
var
  I: Integer;
begin
  if IsObject(FCipherList, TStringList) then
    for I := FCipherList.Count-1 downto 0 do
      if FindClassHInstance(TClass(FCipherList.Objects[I])) = Module then
        FCipherList.Delete(I);
end;
{$ENDIF}

initialization
{$IFDEF UseASM}
  {$IFNDEF 486GE}  // не використовується для <= CPU 386
  if CPUType <= 3 then  // CPU <= 386
  begin
    FindVirtualMethodAndChange(TCipher_Blowfish, @TCipher_Blowfish.Encode,
                                                 @TCipher_Blowfish.Encode386);
    FindVirtualMethodAndChange(TCipher_Blowfish, @TCipher_Blowfish.Decode,
                                                 @TCipher_Blowfish.Decode386);
  end;
  {$ENDIF}
{$ENDIF}
{$IFDEF VER_D3H}
  AddModuleUnloadProc(ModuleUnload);
{$ENDIF}
{$IFNDEF ManualRegisterClasses}
  RegisterCipher(TCipher_3Way, '', '');
  RegisterCipher(TCipher_Blowfish, '', '');
  RegisterCipher(TCipher_Gost, '', '');
  RegisterCipher(TCipher_IDEA, '', 'Не для комерційного використання');
  RegisterCipher(TCipher_Q128, '', '');
  RegisterCipher(TCipher_SAFER_K40, 'SAFER-K40', '');
  RegisterCipher(TCipher_SAFER_SK40, 'SAFER-SK40', 'Keyscheduling');
  RegisterCipher(TCipher_SAFER_K64, 'SAFER-K64', '');
  RegisterCipher(TCipher_SAFER_SK64, 'SAFER-SK64', 'Keyscheduling');
  RegisterCipher(TCipher_SAFER_K128, 'SAFER-K128', '');
  RegisterCipher(TCipher_SAFER_SK128, 'SAFER-SK128', 'Keyscheduling');
  RegisterCipher(TCipher_SCOP, '', '');
  RegisterCipher(TCipher_Shark, '', '');
  RegisterCipher(TCipher_Square, '', '');
  RegisterCipher(TCipher_TEA, 'TEA', '');
  RegisterCipher(TCipher_TEAN, 'TEA extended', '');
  RegisterCipher(TCipher_Twofish, '', '');
{$ENDIF}
finalization
{$IFDEF VER_D3H}
  RemoveModuleUnloadProc(ModuleUnload);
{$ENDIF}
  FCipherList.Free;
  FCipherList := nil;
end.
```

Файл Main.pas – основна програма

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, Buttons, XPMan;

type
  TMainForm = class(TForm)
    PHash: TPanel;
    Label1: TLabel;
    Algorithm: TLabel;
    CBHash: TComboBox;
    EHashFile: TEdit;
    Label2: TLabel;
    BtnHashFile: TBitBtn;
    BtnCalcHash: TBitBtn;
    OpenDialog: TOpenDialog;
    LhashInfo: TLabel;
    Label3: TLabel;
    EDigest: TEdit;
    PCipher: TPanel;
    Label4: TLabel;
    Label5: TLabel;
    CBCipher: TComboBox;
    Label6: TLabel;
    ECipherFile: TEdit;
    BtnCipherFile: TBitBtn;
    BtnCalcCipher: TBitBtn;
    LCipherInfo: TLabel;
    EPassword: TEdit;
    Label7: TLabel;
    Label8: TLabel;
    EHashInput: TEdit;
    Label9: TLabel;
    EHashENC: TEdit;
    Label10: TLabel;
    EhashDEC: TEdit;
    CBMode: TComboBox;
    XPManifest1: TXPManifest;
    SpeedButton1: TSpeedButton;
    Bevel1: TBevel;
    Bevel2: TBevel;
    procedure FormCreate(Sender: TObject);
    procedure BtnHashFileClick(Sender: TObject);
    procedure BtnCalcHashClick(Sender: TObject);
    procedure BtnCipherFileClick(Sender: TObject);
    procedure BtnCalcCipherClick(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  MainForm: TMainForm;

implementation

uses about;

{$R *.DFM}
```

```
{Імпортуємо прототипи з DEC1.DLL}

const
  DEC1DLL = 'DEC1.DLL';

type
  TCipherHandle = Integer;
  THashHandle   = Integer;
  TEnumProc     = function(const Name: PChar; ID, MaxKeySize: Integer; Data:
Pointer): Bool; stdcall;

const
{Режими шифрування для Cipher_Create()}
  cm_CTS        = 0;
  cm_CBC        = 1;
  cm_CFB        = 2;
  cm_OFB        = 3;
  cm_ECB        = 4;

function Cipher_GetID(Name: PChar): Integer; stdcall; external DEC1DLL;
function Cipher_Create(ID, Mode: Integer): TCipherHandle; stdcall; external
DEC1DLL;
function Cipher_Delete(Handle: TCipherHandle): Integer; stdcall; external
DEC1DLL;
function Cipher_Encode(Handle: TCipherHandle; Source, Dest: PChar; Size:
Integer): Integer; stdcall; external DEC1DLL;
function Cipher_Decode(Handle: TCipherHandle; Source, Dest: PChar; Size:
Integer): Integer; stdcall; external DEC1DLL;
function Cipher_Init(Handle: TCipherHandle; Key: Pointer; KeyLen: Integer;
IVector: Pointer): Integer; stdcall; external DEC1DLL;
function Cipher_InitKey(Handle: TCipherHandle; Key: PChar; IVector: Pointer):
Integer; stdcall; external DEC1DLL;
function Cipher_Done(Handle: TCipherHandle): Integer; stdcall; external DEC1DLL;
function Cipher_Protect(Handle: TCipherHandle): Integer; stdcall; external
DEC1DLL;
function Cipher_GetMaxKeySize(Handle: TCipherHandle): Integer; stdcall; external
DEC1DLL;
function Cipher_SetHash(Handle: TCipherHandle; Hash_ID: Integer): Integer;
stdcall; external DEC1DLL;
function Cipher_GetHash(Handle: TCipherHandle): Integer; stdcall; external
DEC1DLL;
procedure Cipher_EnumNames(Proc: TEnumProc; UserData: Pointer); stdcall;
external DEC1DLL;

function Hash_GetID(Name: PChar): Integer; stdcall; external DEC1DLL;
function Hash_Create(ID: Integer): THashHandle; stdcall; external DEC1DLL;
function Hash_Delete(Handle: THashHandle): Integer; stdcall; external DEC1DLL;
function Hash_Init(Handle: THashHandle): Integer; stdcall; external DEC1DLL;
function Hash_Done(Handle: THashHandle; Digest: PChar; DigestSize: Integer):
Integer; stdcall; external DEC1DLL;
function Hash_Update(Handle: THashHandle; Source: PChar; SourceLen: Integer):
Integer; stdcall; external DEC1DLL;
function Hash_GetMaxDigestSize(Handle: THashHandle): Integer; stdcall; external
DEC1DLL;
function Hash_CalcFile(ID: Integer; FileName, Digest: PChar; MaxDigestLen:
Integer): Integer; stdcall; external DEC1DLL;
procedure Hash_EnumNames(Proc: TEnumProc; UserData: Pointer); stdcall; external
DEC1DLL;

function StrToBase64(Dest, Source: PChar; Len, MaxLen: Integer): Integer;
stdcall; external DEC1DLL;
function Base64ToStr(Dest, Source: PChar; Len, MaxLen: Integer): Integer;
stdcall; external DEC1DLL;
function StrToBase16(Dest, Source: PChar; Len, MaxLen: Integer): Integer;
stdcall; external DEC1DLL;
function Base16ToStr(Dest, Source: PChar; Len, MaxLen: Integer): Integer;
stdcall; external DEC1DLL;

{Кінець імпорту}
```

```
procedure TMainForm.FormCreate(Sender: TObject);

  function EnumHash(const Name: PChar; ID, MaxKeySize: Integer; Combo:
TComboBox): Bool; stdcall;
  {ітерації з ID=0 до HashCount-1}
  begin
    Result := True;
    Combo.Items.AddObject(Name, Pointer(MaxKeySize));
  end;

  function EnumCipher(const Name: PChar; ID, MaxKeySize: Integer; Combo:
TComboBox): Bool; stdcall;
  begin
    Result := True;
    Combo.Items.AddObject(Name, Pointer(MaxKeySize));
  end;

begin
  EHashFile.Text := ParamStr(0);
  ECipherFile.Text := ParamStr(0);

  Hash_EnumNames(@EnumHash, CBHash);
  CBHash.ItemIndex := 0;
  BtnCalcHashClick(nil);

  Cipher_EnumNames(@EnumCipher, CBCipher);
  CBCipher.ItemIndex := 0;
  CBMode.ItemIndex := 0;
  BtnCalcCipherClick(nil);
end;

procedure TMainForm.BtnHashFileClick(Sender: TObject);
begin
  if OpenDialog.Execute then
  begin
    EHashFile.Text := OpenDialog.FileName;
    BtnCalcHashClick(nil);
  end;
end;

procedure TMainForm.BtnCalcHashClick(Sender: TObject);
var
  Handle: THashHandle;
  Len: Integer;
  S: TFileStream;
  Buffer: array[0..1023] of Char;
  Digest: String;
begin
  if FileExists(EHashFile.Text) and (CBHash.ItemIndex >= 0) then
  begin
    Len := Integer(CBHash.Items.Objects[CBHash.ItemIndex]);
    LHashInfo.Caption := Format('Розмір хешу: %d bytes, %d bits', [Len, Len *
8]);
    SetLength(Digest, Len);

    Handle := Hash_Create(CBHash.ItemIndex);
    if Hash_Init(Handle) = 0 then
    try
      S := TFileStream.Create(EHashFile.Text, fmOpenRead or fmShareDenyNone);
      try
        repeat
          Len := S.Read(Buffer, Sizeof(Buffer));
          Hash_Update(Handle, Buffer, Len);
        until Len <= 0;
        Hash_Done(Handle, PChar(Digest), Length(Digest));

        StrToBase16(Buffer, PChar(Digest), Length(Digest), SizeOf(Buffer));
        EDigest.Text := StrPas(Buffer);
```

```
      finally
        S.Free;
      end;
    finally
      Hash_Delete(Handle);
    end else EDigest.Text := 'Помилка введення';
  end;
end;

procedure TMainForm.BtnCipherFileClick(Sender: TObject);
begin
  if OpenDialog.Execute then
  begin
    ECipherFile.Text := OpenDialog.FileName;
    BtnCalcCipherClick(nil);
  end;
end;

procedure TMainForm.BtnCalcCipherClick(Sender: TObject);

  procedure HashBase64(const FileName: String; Output: TEdit);
  var
    Digest: String;
    Len: Integer;
  begin
    SetLength(Digest, 1024);
    Len := Hash_CalcFile(CBHash.ItemIndex, PChar(FileName), PChar(Digest),
Length(Digest));
    if (Len > 0) and (StrToBase64(PChar(Digest), PChar(Digest), Len,
Length(Digest)) = 0) then
      Output.Text := Digest
    else Output.Text := 'Error';
  end;

var
  Len: Integer;
  Handle: TCipherHandle;
  S,D: TFileStream;
  Buffer: array[0..1023] of Char;
begin
  if FileExists(ECipherFile.Text) and (CBCipher.ItemIndex >= 0) then
  try
    Screen.Cursor := crHourGlass;

    Len := Integer(CBCipher.Items.Objects[CBCipher.ItemIndex]);
    LCipherInfo.Caption := Format('Макс. розмір ключа: %d bytes, %d bits', [Len,
Len * 8]);

{створюємо шифр}
    Handle := Cipher_Create(CBCipher.ItemIndex, CBMode.ItemIndex);
{встановлюємо ключ шифрування хеш функції  SHA1}
    Cipher_SetHash(Handle, CBHash.ItemIndex);
    try
{встановлюємо фази шифрування}
      Cipher_InitKey(Handle, PChar(EPassword.Text), nil);
{open Source & Desfile, read in, encode}
      S := nil;
      D := nil;
      try
        S := TFileStream.Create(ECipherFile.Text, fmOpenRead or
fmShareDenyNone);
        D := TFileStream.Create(ChangeFileExt(ParamStr(0), '.ENC'), fmCreate);
        repeat
          Len := S.Read(Buffer, SizeOf(Buffer));
          Cipher_Encode(Handle, Buffer, Buffer, Len);
          D.Write(Buffer, Len);
        until Len <= 0;
      finally
        Cipher_Protect(Handle);
```

```
        S.Free;
        D.Free;
      end;
{і тепер назад, дешифруємо}
      Cipher_InitKey(Handle, PChar(EPassword.Text), nil);
      S := nil;
      D := nil;
      try
        S := TFileStream.Create(ChangeFileExt(ParamStr(0), '.ENC'), fmOpenRead
or fmShareDenyNone);
        D := TFileStream.Create(ChangeFileExt(ParamStr(0), '.DEC'), fmCreate);
        repeat
          Len := S.Read(Buffer, SizeOf(Buffer));
          Cipher_Decode(Handle, Buffer, Buffer, Len);
          D.Write(Buffer, Len);
        until Len <= 0;
      finally
        Cipher_Protect(Handle);
        S.Free;
        D.Free;
      end;
    finally
      Cipher_Delete(Handle);
    end;

{перевіряємо роботу}
    HashBase64(ECipherFile.Text, EHashInput);
    HashBase64(ChangeFileExt(ParamStr(0), '.ENC'), EHashENC);
    HashBase64(ChangeFileExt(ParamStr(0), '.DEC'), EHashDEC);
    if EHashInput.Text <> EHashDEC.Text then EHashDEC.Color := clRed
      else EHashDEC.Color := clWindow;
  finally
    Screen.Cursor := crDefault;
  end;
end;

procedure TMainForm.SpeedButton1Click(Sender: TObject);
begin
Form1.Show;
end;

end.
```

Файл about.pas – довідка

```
unit about;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    Image1: TImage;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
Form1.Close;
end;

procedure TfmAbout.FormCreate(Sender: TObject);
begin
Label1.Caption:='БАКАЛАВРСЬКИЙ ПРОЕКТ';
Label2.Caption:='на тему:';
Label3.Caption:='Програмне забезпечення системи кібербезпеки центру генерації та
розподілу ключів системи захисту інформації';
Label5.Caption:='Керівник: Смірнов С.А.';
Label6.Caption:='Розробив: студент Нікіша Максим Ігорович';
Label7.Caption:='гр. КБ-20-3СК';
Label8.Caption:='м. Кропивницький 2023';
end;

end.
```