МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

МІЖНАРОДНИЙ ЕКОНОМІКО-ГУМАНІТАРНИЙ УНІВЕРСИТЕТ ІМЕНІ АКАДЕМІКА СТЕПАНА ДЕМ'ЯНЧУКА

І.В. УКРАЇНЕЦЬ

АНАЛІЗ І ДОСЛІДЖЕННЯ КРИПТОГРАФІЧНИХ ЗАСОБІВ ЗАХИСТУ ІНФОРМАЦІЇ НА БАЗІ «УКРГАЗБАНК»

Науковий керівник: Р.М.Літнарович,
доцент, кандидат технічних наук

Рівне, 2012

ІРИНА ВАСИЛІВНА УКРАЇНЕЦЬ, МАГІСТРАНТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

УДК 614.2

Українець І.В. Аналіз і дослідження криптографічних засобів захисту інформації на базі «Укргазбанк». Науковий керівник Р.М.Літнарович. МЕГУ, Рівне, 2011.- 114 с.

Рецензенти **:** В.Г.Бурачек, доктор технічних наук, професор
В.О.Боровий, доктор технічних наук, професор
……..Є.С.Парняков, доктор технічних наук, професор
Відповідальний за випуск: Й.В.Джунь, доктор фіз.-мат. наук, професор

Важливість і актуальність питань захисту інформації вже давно вийшли на одне з перших місць серед інших завдань, що вирішуються в процесі проектування, створення та використання сучасних інформаційних систем. Причини такої підвищеної уваги до цієї проблеми цілком очевидні - від якості заходів захисту інформації безпосередньо залежить економічна безпека організації.

Ключові слова – інформація, захист, криптографія, безпека.

Важность и актуальность вопросов защиты информации уже давно вышли на одно из первых мест среди других заданий, которые решаются в процессе проектирования, создания и использования современных информационных систем. Причины такого повышенного внимания к этой проблеме полностью очевидны - от качества мероприятий защиты информации непосредственно зависит экономическая безопасность организации.

Ключевые слова - информация, защита, криптография, безопасность.

Importance and actuality of questions of priv already a long ago went out on one of the first places among other tasks, which decide in the process of planning, creation and use of the modern informative systems. Reasons of such enhanceable attention to this problem are fully obvious - on quality of measures of priv economic security of organization depends directly.

Keywords - information, defence, cryptography, safety.

ЗМІСТ

Вступ……5

Розділ 1. Криптографія та її практичне застосування у сфері інформаційної безпеки……8

1.1. Криптографічні засоби захисту інформації……8

1.2. Математичні основи криптографічного захисту інформації……13

1.3. Порівняльний аналіз симетричних та асиметричних алгоритмів……21

Розділ 2. Огляд криптографічних алгоритмів захисту інформації……26

2.1. Шифр DES……26

2.2. Шифр ГОСТ……35

2.3. Шифр Blowfish……38

…2.4. Шифр RSA……43

Розділ 3. Програмна реалізація алгоритму……47

…3.1. Опис алгоритму реалізації системи……47

…3.2. Середовище розробки……51

…3.3. Інструкція користувачеві по використанню……56

Висновки……67

Список використаної літератури……69

Додатки……74

Додаток 1. Реалізація подій та методів головної форми вікна……74

Додаток 2. Реалізація об’єктів шифрування даних……90

Додаток 3. Програмування додаткових інтерфейсів……102

ВСТУП

Організація магістерської роботи спрямована на дослідження та аналіз криптографічних засобів захисту інформації на базі УКРГАЗ Банку. Широке використання обчислювальних мереж, призводить до того, що з’являється велика можливість для несанкціонованого доступу до переданої інформації.

Останнім часом зріс інтерес до питань захисту інформації. це пов'язують з тим, що стали більш широко використовуватися обчислювальні мережі, що призводить до того, що з'являються великі можливості для несанкціонованого доступу до переданої інформації.

Важливість і актуальність питань захисту інформації вже давно вийшли на одне з перших місць серед інших завдань, що вирішуються в процесі проектування, створення та використання сучасних інформаційних систем. Причини такої підвищеної уваги до цієї проблеми цілком очевидні - від якості заходів захисту інформації безпосередньо залежить економічна безпека організації.

Банківська інформація завжди була об'єктом пильної уваги всякого роду зловмисників. У системі забезпечення безпеки банку захист інформації відіграє найбільш важливу роль.

Сучасні технології дають банкам переваги в організації систем доставки товарів і послуг. Використання електронних засобів зв'язку дозволяє реалізувати:

- ✓ електронні платежі і розрахунки в точці продажу;
- ✓ клієнтські термінали, які здійснюють прямий зв'язок з банком;
- ✓ домашнє банківське обслуговування за допомогою персонального комп'ютера або телефону;
- ✓ обмін електронними даними в мережі з розширеним набором послуг;
- ✓ технології електронних банківських карт, включаючи магнітні та електронні пластикові карти.

Актуальність роботи викликана потребою створення програмної системи захисту інформації у комп'ютерних мережах на основі криптографічних засобів захисту інформації, які забезпечують найбільший ступінь захисту інформації. Проектування та розробка системи захисту інформації створена на основі новітньої платформи для розробки Visual Studio 2010, що є лідером серед засобів для розробки складних програмно-інженерних систем.

В магістерській роботі за основу взято симетричну та асиметричну криптографію. Особливу увагу приділено алгоритмам захисту DES, ГОСТ, Blowfish та RSA. Велику увагу у роботі приділено сліпому цифровому підпису. Сліпий цифровий підпис реалізовано за допомогою алгоритму RSA. Найбільш широке застосування протокол сліпих підписів знайшов у сфері цифрових грошей.

Мета роботи — розробити криптографічно-стійку систему захисту інформації, що представляється у вигляді програмної системи, що дозволяє проводити кодування та передачу інформаційних ресурсів у мережі. Основними завданнями для досягнення мети стали:

- Дослідження проблеми захисту інформації на основі криптографічних методів;
- Розробити програмну систему захисту інформації для зберігання обробки та використання закодованої інформації;
- Аналіз симетричної та асиметричної криптографії, дослідження роботи шифрів блочного типу. Виділення найбільш надійних шифрів захисту інформації;
- Організувати базу даних користувачів для обміну повідомлень із відповідним цифровим підписом кожного із користувачів системи;
- Реалізувати програмну систему у середовищі Visual Studio 2010 на мові програмування C#.

Наукова новизна полягає у реалізації системи захисту інформації, що дозволяє проводити обмін, зберігання обробку інформаційних ресурсів із

врахуванням крипто-стійкості та відповідної надійності для банківських систем. Систему захисту інформації розроблено за допомогою новітньої мови програмування C#, що дозволило врахувати можливості передачі та ідентифікації електронних повідомлень між користувачами системи.

Практична значимість роботи полягає в розробці програмного продукту, який є перевірений, протестований та впроваджений на базі УКРГАЗ Банку. Розроблена криптографічна система захисту інформації відповідає усім вимогам, які були поставлені до даного програмного забезпечення. Перевагою системи є забезпечення можливості її роботи на будь-якому ПК. Головною вимогою ставиться наявність NET Framework 4.0.

РОЗДІЛ 1. КРИПТОГРАФІЯ ТА ЇЇ ПРАКТИЧНЕ ЗАСТОСУВАННЯ У СФЕРІ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

1.1. Криптографічні засоби захисту інформації

Криптографічний захист інформації — вид захисту, що реалізовується за допомогою перетворень інформації з використанням спеціальних (ключових) даних з метою приховування (або відновлення) змісту інформації, підтвердження її справжності, цілісності, авторства тощо.

Завдання криптографії є пошуком і дослідженням математичних методів перетворення інформації. Криптографія дозволяє перетворити інформацію таким чином, що її прочитання (відновлення) стає можливим тільки при знанні ключа.

Криптографія — наука про математичні методи забезпечення конфіденційності (неможливості прочитання інформації стороннім) і автентичності (цілісності і справжності) інформації. ***Відкритий текст*** (plaintext, clear text) – це інформація, що може бути прочитана, осмислена і зрозуміла без будь-яких спеціальних заходів. Шифрування дозволяє сховати інформацію від тих, для кого вона не призначається, незважаючи на те, що вони можуть бачити сам шифротекст. Протилежний процес перетворення шифротексту в його вихідний вид називається розшифруванням або ***дешифруванням.***

Ключ – конкретний таємний стан деяких параметрів алгоритму криптографічного перетворення даних, що забезпечує вибір одного варіанту із сукупності всіляких для даного алгоритму.

Під ***шифром*** розуміється сукупність зворотних перетворень безлічі відкритих даних на безліч зашифрованих даних, заданих алгоритмом криптографічного перетворення[33].

Криптостійкістю називається характеристика шифру, визначальна його стійкість до дешифрування. Звичайно ця характеристика визначається періодом часу, необхідним для дешифрування.

За принципами використання криптографічний захист може бути вбудованим у платіжну систему або бути додатковим механізмом, який може відключатися. Є дві групи криптографічних алгоритмів:

1) загальні:
    a. симетричні;
    b. асиметричні;
2) спеціальні.

Криптографічні алгоритми застосовують із метою:

- шифрування інформації;
- захисту даних і повідомлень (інформації) від модифікації або підробки.

На протидію загрозам та з метою мінімізації можливих збитків користувачів і власників платіжної системи спрямовані чотири групи заходів:

1) правові;

2) морально-етичні;

3) адміністративні;

4) фізичні.

Особливу увагу заслуговують адміністративні та фізичні засоби захисту інформації.

Адміністративні заходи — це заходи організаційного характеру, які регламентують процес функціонування системи обробки платіжної інформації, використання її ресурсів, діяльність персоналу тощо[15]. До них можна віднести:

- організацію перепускного режиму до приміщень, де розміщені основні обчислювальні засоби для обробки платіжної інформації;
- дотримання правил обробки інформації та інструкцій для обслуговуючого персоналу під час обробки платіжної інформації;
- організацію розподілу доступу і зберігання паролів та криптографічних ключів;

- організацію обліку, зберігання та знищення документів та носіїв з конфіденційною інформацією;
- організацію підготовки користувачів платіжної системи й обслуговуючого персоналу та контролю за їх роботою;
- порядок внесення змін до програмних і апаратних засобів платіжної системи тощо.

Фізичний захист систем електронних розрахунків потребує виконання вимог щодо безпечного та надійного функціонування ключових обчислювальних машин платіжної системи. При централізованій обробці платіжних документів особливу увагу треба приділяти фізичній охороні та пропускному режиму будівель, де розміщена основна обчислювальна техніка. Приміщення з обчислювальною технікою повинні бути обладнані кількома рівнями охоронної сигналізації, пожежною сигналізацією, бажано встановити охоронне телебачення для здійснення постійного контролю за обчислювальною технікою.

Криптографічний захист може здійснюватися в процесі передачі інформації каналами зв'язку та при її обробці на робочих станціях і серверах.

При передачі інформації каналами зв'язку висуваються наступні вимоги:

- забезпечення конфіденційності інформації;
- забезпечення цілісності інформації;
- автентичність сторін інформаційного обміну.

Конфіденційність інформації забезпечується симетричним (алгоритми ГОСТ 28147-89, DES, 3DES, AES, IDEA, Blowfish) та асиметричним (алгоритми RSA, El Gamal) шифруванням[13]. Цілісність інформації та автентичність сторін досягається використанням хеш-функцій та технологій цифрового підпису. Сукупність технологій, що забезпечують конфіденційність та цілісність інформації при її передачі незахищеними каналами зв'язку, отримала назву віртуальних приватних мереж (VPN – Virtual Private Network)[31].

В процесі мережевої взаємодії захист інформації, зокрема, забезпечується за допомогою протоколів SSL, SSH, S-HTTP, IPSec, тощо. Автентичність сторін інформаційного обміну досягається за рахунок використання протоколів X.509, RADIUS, TACACS+ та інших. Реалізація даних технологій може здійснюватися програмними та програмно-апаратними засобами[39].

Захист інформації на робочих станціях і серверах може реалізовуватися за допомогою шифрування на рівні файлової системи, криптографічних методів перевірки автентичності (цифрові сертифікати, одноразові паролі тощо), криптографічних засобів перевірки цілісності (контрольні суми).

На сьогодні у сфері криптографічного захисту інформації реалізовуються такі основні завдання:

- участь у формуванні та реалізація державної політики у сфері криптографічного захисту інформації;
- розробка нормативно-правових актів, організація та проведення ліцензування, сертифікації та державної експертизи у сфері криптографічного захисту інформації;
- здійснення контролю за додержанням вимог законодавства у сфері надання послуг електронного цифрового підпису (ЕЦП).

Формування та реалізація державної політики у сфері криптографічного захисту інформації (КЗІ) здійснюється за наступними напрямами[15]:

- нормативно-правове забезпечення функціонування та розвитку системи криптографічного захисту інформації;
- організація та супроводження заходів у рамках державних цільових програм;
- ліцензування господарської діяльності у сфері КЗІ;
- координація робіт з проведення сертифікації засобів КЗІ, державної експертизи у сфері КЗІ, тематичних досліджень та допуску до експлуатації засобів КЗІ;
- організація робіт з проведення сертифікації засобів КЗІ;
- організація та проведення робіт з державної експертизи у сфері КЗІ, тематичних досліджень та допуску до експлуатації засобів КЗІ;
- участь у заходах щодо державного контролю за міжнародними передачами товарів подвійного використання;

- розроблення криптографічних алгоритмів та криптографічних протоколів;

- централізований облік спеціальних виробів.

Розвиток системи криптографічного захисту інформації, організація і налагодження виробництва вітчизняних захищених засобів і технологій для оброблення інформації – це необхідні напрямки діяльності у сфері забезпечення національної безпеки України[21].

На сьогодні як результат діяльності у сфері КЗІ:

- ✓ розгорнуто виробництво національних ключових документів (шифрів) для використання у системах спеціального зв'язку України;
- ✓ розроблено сімейство національних криптографічних алгоритмів та криптографічних протоколів, на основі яких можуть створюватись найсучасніші засоби захисту інформації;
- ✓ розроблено типові рішення, які дозволяють забезпечити створення уніфікованого ряду засобів шифрування і засекречування та розгорнути їх серійне виробництво;
- ✓ розроблено та серійно виготовляються вітчизняні конкурентно спроможні засоби захисту інформації, які за своїми властивостями не поступаються кращим світовим зразкам; створена нормативно-правова база та побудована організаційно-технічна інфраструктура для організації та застосування електронного цифрового підпису як аналога власноручного підпису.

За останні роки в державі забезпечено сталий розвиток інфраструктури електронного цифрового підпису відповідно до положень Закону України «Про електронний цифровий підпис».

Однією із складових реалізації державної політики у сфері криптографічного захисту інформації є проведення ліцензування та сертифікації засобів КЗІ.

На сьогодні ліцензії у сфері КЗІ отримали понад 300 суб'єктів господарювання. Регулярно здійснюється перевірка дотримання суб'єктами господарської діяльності ліцензійних умов і правил провадження діяльності у сфері КЗІ.

Створені системи сертифікації й експертизи систем та засобів КЗІ та забезпечується їх діяльність.

Виконуються роботи, пов'язані із проведенням досліджень та підтвердженням якісних характеристик засобів, комплексів та систем криптографічного захисту інформації[22].

1.2. Математичні основи криптографічного захисту інформації

Криптографічний захист базується на основі використання математичних методів перетворення інформації. Оскільки написання програмного продукту для захисту інформації виконано на мові програмування C# наведемо перелік операцій над бітами (табл. 1.1).

Усі операції для перетворення інформації на основі симетричної криптографії базуються на побітових операціях. В процесі шифрування інформації найбільш часто використовується операція побітового додавання за модулем два. Дана операція в мові програмування C# задається за допомогою операції «виключення або» ^. Дану операцію будемо означати знаком $\oplus$. Операція виконує наступне перетворення бітів:

$$0 \oplus 0 = 0,\ 0 \oplus 1 = 1,\ 1 \oplus 0 = 1,\ 1 \oplus 1 = 0.$$

Дана операція використовується у шифрі одноразового блокноту[10].

Таблиця 1.1.

Основні операції роботи з бітами

Операція	Опис
<<	Побітовий зсув в вліво
>>	Побітовий зсув в право
|	Операція диз'юнкції
^	Операція еквіваленції
&	Операція кон'юнкції

Опишемо детально даний алгоритм роботи шифру одноразового блокноту. Перед шифруванням повідомлення M записується у двійковій формі. Ключем K служить довільне двійкове слово однакової з M довжини.

Криптотекст С отримують побітовим додаванням повідомлення і ключа, тобто $C = M \oplus K$.

Дешифрування у шифрі одноразового блокноту збігається із шифруванням — щоб отримати вихідне повідомлення М, слід додати до криптотексту С той же ключ К. Це легко обґрунтувати: оскільки $C = M \oplus K$, тоді справедливе наступне співідношення

$$C \oplus K = (M \oplus K) \oplus K = M \oplus (K \oplus K) = M \oplus 0 = M$$

Алгоритм, який полягає у шифруванні повідомлення за допомогою одного шифру, а потім застосування до отриманого крипто тексту, ще одного шифру називається композицією або добутком цих двох шифрів[50].

Частковим випадком композиції двох шифрів є послідовне застосування двічі одного і того ж шифру. Інтуїтивним виглядає припущення, що подвійне шифрування збільшує надійність.

Перспективнішим видається подвійне шифрування з незалежним вибором ключа кожного разу. Тобто, за допомогою алгоритму шифрування Е повідомлення М перетворюється у криптотекст $C = E_{K_2}(E_{K_1}(M))$, де два ключі K_1 та K_2 вибираються незалежно один від одного. Дешифрування не складає труднощів: $M = D_{K_1}(D_{K_1}(C))$ (належить звернути увагу на зміну порядку, в якому застосовуються ключі).

Припустимо, що ключем служить двійкове слово довжиною n. В цьому разі всі можливі ключі є 2^n, а всі можливі пари ключів є 2^{2n}. Однак це не означає, що ламання двократного шифру займатиме 2^{2n} кроків замість 2^n. Є спосіб оптимізації перебірної процедури, так звана зустрічна атака, що займає кілька кроків пропорційну до 2^n, але взамін вимагає машинної пам'яті такого ж порядку. Це атака з відомим відкритим текстом, але для проведення якої потрібно знати якусь пару з повідомлення М та відповідного йому криптотексту С.

Алгоритм Евкліда. Для будь-якого цілого a і натурального b однозначно визначені цілі числа q і r такі, що $a = bq + r$ і $0 \le r \le b$. Число q

називають часткою, а r — остачею від ділення a на b [56]. Наприклад, рівність $-20=(-1)\cdot 67+47$ означає, що -20 при ділені на 67 дає остачу -1 і остачу 47. Для остачі будемо вживати таке позначення: $r=a \bmod b$. Число r будемо також називати (зведеним) зведеним лишком числа a за модулем b. Якщо $r=0$, то кажуть, що a ділиться на b (націло або без остачі) і пишеться $a \mid b$. Кажуть також, що b ділить a, називають b дільником числа a, а a — кратним числа b. Запис $a \nmid b$ означатиме, що a не ділиться на b. Для цілих a і b через НСД (a,b) позначатимемо їх найбільший спільний дільник (НСД) — найбільше з поміж таких c, що одночасно $c \mid a$ і $c \mid b$ (хоча б одне із a і b вважається відмінним від нуля). Числа a і b називаються взаємо простими, якщо НСД $(a,b)=0$.

Алгоритм Евкліда для знаходження НСД двох натуральних чисел a і b ґрунтується на співвідношеннях

НСД $(a,b)=$ НСД $(a, a \bmod b)$ для $a \geq b$ (1)

НСД $(a,0)=a$ (2)

Спершу продемонструємо ідею цього алгоритму на прикладі.

Щоб знайти НСД(211,79), застосуємо послідовність (1) аж доки не опинимося у ситуації, коли можна буде використати (2). Таким чином робота алгоритму зводиться до кількаразового ділення з остачею.

$$211=79\cdot 2+53$$

$$79=53\cdot 1+26$$

$$53=26\cdot 2+1$$

$$26=1\cdot 26+0$$

Маємо НСД(211,79) = НСД(79,53) = НСД(53,26) = НСД(26,1) = НСД(1,0) = 1.

Опишемо, що відбувається в загальному випадку.

Алгоритм Евкліда обчислення НСД $(a,b), a \geq b.$

- Покласти $r_0=a,\ r_1=b,\ i=1.$
- Виконати i-ий крок, що полягає в ділені з остачею:

$$r_{i-1} = r_i \cdot q + r_{i+1}.$$

- Якщо $r_{i+1} > 0,$ то збільшити i на 1 і перейти до виконання наступного кроку згідно з попереднім пунктом. Якщо $r_{i+1} = 0$, то завершити роботу із результатом НСД$(a,b) = r_i$.

Коректність. Алгоритм завершує роботу, як тільки $r_{i+1} = 0$ [56]. Рано чи пізно це трапиться, оскільки $0 \le r_{i+1} < r_i$ для кожного i. Припустимо так сталося на m-ому кроці, тобто $r_{m+1} = 0$. Результатом роботи алгоритму є значення r_m. Рівність $r_m = \text{НСД}(a,b)$ випливає з таких міркувань. Співвідношення НСД(a,b) = НСД(r_{i-1}, r_i) для $1 \le i \le m+1$ доводиться індукцією з використанням (1). При $i = m+1$ із врахуванням (2) отримуємо НСД(a,b) = НСД$(r_m, 0) = r_m$.

Ефективність. Під ефективністю алгоритму Евкліда розуміють те, що кількість кроків m, які виконуються на вході $a,b,$ не є надто великою [10]. Із співвідношення $0 \le r_{i+1} < r_i$ випливає, що $m \le b.$ Однак ця оцінка є незадовільною. Скажімо, для чисел a і b з якоюсь сотнею цифр у десятковому записі кожного, нам гарантовано лише, що на знаходження їх НСД буде витрачено не більше 10^{100} кроків. Але ж найбільш швидкодіючі ЕОМ для виконання такої кількості кроків потрібен фантастично великий обсяг часу.

На щастя, оцінку на m можна суттєво поліпшити. Покажемо, що $m \le 2\log_2 b + 1$. Для цього використаємо нерівність $r_{i+1} < r_{i-1}/2$. Вона доводиться розглядом двох випадків: якщо $r_i < r_{i-1}/2$, то виконується нерівність $r_{i+1} < r_i$; якщо $r_i \ge r_{i-1}/2$, то виконується рівність $r_{i+1} = r_{i-1} \bmod r_i$. Таким чином, кожні два кроки зменшуються r_{i+1} принаймні в двічі, і не пізніше $2\log_2 b + 1$ кроків ми прийдемо до $r_{i+1} = 0$.

Алгоритм Евкліда дає такий наслідок.

Твердження. Для кожної пари взаємно простих чисел a і b можна знайти такі цілі u і v, що $ua + vb = 1$ [10].

Зазначимо, що алгоритм Евкліда дає ефективний спосіб знаходження коефіцієнтів u і v для заданої пари a, b.

Нехай $a = 211,\ b = 79$. Протокол роботи алгоритму Евкліда виписаний вище. Рухаючись знизу в вверх, отримаємо

$$1 = 1 \cdot 53 + (-2) \cdot 26 = 1 \cdot 53 + (-2) \cdot (79 - 1 \cdot 53) = (-2) \cdot 79 + 3 \cdot 53 = (-2) \cdot 79 + \\ + 3 \cdot (211 - 2 \cdot 79) = 3 \cdot 211 + (-8) \cdot 79.$$

Отже, $u = 3$ і $v = -8$.

Конгруенція.

Ми домовилися позначати остачу від ділення цілого a на натуральне b через $a \bmod b$. Позначення $\bmod$ ми будемо вживати і в іншому значенні. А саме, ми будемо писати $x \equiv y (\bmod n)$, якщо цілі x і y при ділені на натуральне n дають однакову остачу (тобто, $x \bmod n = y \bmod n$). Такі x і y називаються конгруентними або рівними за модулем n. Відношення між x і y називається конгруенцією або порівнянням [54].

Твердження.

Наступні три умови еквівалентні.

1) $x \equiv y (\bmod n)$.
2) $x = y + kn$ для деякого цілого k.
3) $n \mid (x - y)$.

Для натурального n через Z_n позначаємо множину $\{0,1,\ldots,n-1\}$, наділену операціями додавання та множення за модулем n. Сумою x і y із Z_n є $(x+y) \bmod n$, а їх добуток є $(x \cdot y) \bmod n$. Відносні цих операцій Z_n є комутативним кільцем з одиницею, яке називається кільцем зведених лишків за модулем n. Через Z_n^* позначатимемо мультиплікативну групу елементів, для яких в Z_n є обернений відносно множення.

Твердження. Z_n^* складається з елементів x, взаємно простих з n, і лише з них.

Елемент, обернений до $x \in Z_n^*$ відносно множення, будемо позначати через $x^{-1} \bmod n$ або просто x^{-1}. Ділення на x в Z_n означатиме множення на x^{-1}. Елемент, обернений до $x \in Z_n$ відносно додавання будемо позначати через $-x$. Зокрема, $-1 = n-1$ в Z_n. Як звичайно, в Z_n можна ввести операцію віднімання: $x - y = x + (-y)$.

Нехай ми хочемо знайти елемент, обернений до 79 в Z_{211}^*. Вище було отримано рівність $1 = 3 \cdot 211 + (-8) \cdot 79$. З неї негайно випливає $(-8) \cdot 79 \equiv 1 (\bmod 211)$. Отже $79^{-1} \bmod 211 = (-8) \bmod 211 = 203$.

Наслідок. Для простого модуля p, кільця Z_n є полем [28].

Через ш(n) позначаємо порядок групи Z_n^*. Іншими словами, значення ш(n) дорівнює кількості натуральних чисел, що не перевищують n і взаємо прості з n. ш називається функцією Ойлера.

Теорема Ойлера. Для взаємно простих цілих x і натурального n справедлива конгруенція $x^{ш(n)} \equiv 1 (\bmod n)$.

Як легко бачити, ш(p)= p - 1 для простого p. Звідси отриманий такий наслідок теореми Ойлера.

Мала ТЕОРЕМА Ферма.

Якщо ціле x не ділиться на просте p, то $x^{p-1} = 1 (\bmod p)$.

Китайська ТЕОРЕМА про остачі.

Для будь-якої пари взаємно простих натуральних чисел n_1 і n_2 та для будь-якої пари цілих чисел x_1 і x_2, можна знайти таке ціле x, де $x \equiv x_1 (\bmod n_1)$ і $x \equiv x_2 (\bmod n_2)$.

Твердження.

Нехай $n = n_1 n_2$, де n_1 і n_2 взаємно прості. Тоді відображення $f : Z_n \to Z_{n_1} \oplus Z_{n_2}$. Задане співвідношенням

$$f(x) = (x \bmod n_1, x \bmod n_2),$$

є ізоморфізмом кілець.

Таким чином ми навели приклад обчислення НСД двох натуральних чисел з використанням алгоритму Евкліда. Наведемо приклад функції реалізацію даного методу на мові C#:

```
double NSD(int r0,int r1)
{
int riplus1=r0;
      int qi=r1;
      while(riplus1>0)
      {
            riplus1=r0%r1;
            qi=r0/r1;
            r0=r1;
            r1=riplus1;
}
return r0;
}
```

Генерація простих чисел.

Для алгоритмів з відкритим ключем потрібні прості числа [42]. Їх потрібна множина для довільної достатньо великої мережі.

Генерування випадкових чисел із спробою їх розкладу на множники — це не правильний спосіб пошуку простих чисел. Існують різноманітні імовірності перевірки простих чисел, із завданням ступеню достовірності. При умові, що ця «достовірність» досить велика, такі спроби перевірки достатньо великі.

Роберт Соловей і Фолькер Штрасен розробили алгоритм імовірності перевірки простоти числа [56]. Для перевірки простоти числа p цей алгоритм використовує символ Якобі:

1. Вибір випадкового числа a, менше p.
2. Якщо НСД(a, p), то p не проходить превірку і являється основним.
3. Обчислюємо $j = a(p-1)/2 \bmod p$.
4. Обчислюємо сивол Якобі $J(a, p)$.
5. Якщо $j \neq J(a, p)$, то число p більш усього не являється простим.
6. Якщо $j = J(a, p)$, то імовірно, що число p не являється простим, не більше 50 відсотків.

Число a, яке не показує, що p імовірно не являється простим числом, називається свідком. Якщо p — складне число, імовірність випадкового числа a бути свідком не нижче 50 відсотків. Повторивши цю перевірку t раз з t різних значень a. Імовірність того, що остаток числа перевищить всі t перевірки не перевищує $(1/2)^t$.

Інший більш простий спосіб був розроблений Леманом [54]. Послідовність дій при первірці простоти числа p:

1. Вибрати випадкове число a, менше p.
2. Обчислити $a^{(p-1)/2} \bmod p$.
3. Якщо $a^{(p-1)/2} \neq 1$ або $-1 (\bmod p)$, то p не являється простим.
4. Якщо $a^{(p-1)/2} = 1$, то імовірність того, що число p не являється простим, не більше 50 відсотків.

Дискретні логарифми в скінченному полі.

В якості однонапровної функції в криптографії часто використовують піднесення в степінь по модулю [56]. Легко обчислити:

$$a^x \bmod n$$

Задачою, зворотнього піднесення в степінь по модулю, являється пошук дискретного логарифму, а це уже задача:

Нехай x, для якого $a^x \equiv b (\bmod n)$.

Наприклад:

Якщо , то $x = 6$

Рішення існує не для всіх дискретних логарифмів. Легко помітити, що наступне рівняння не має рішення

$3^x \equiv 7 \bmod 13$

Ще складніше цю задачу рішити для 1024-бітових чисел.

1.3. Порівняльний аналіз симетричних та асиметричних алгоритмів

З метою визначення найдоцільнішого криптографічного алгоритму при проектуванні системи захисту інформації, зроблено порівняльний аналіз алгоритмів шифрування з огляду на наступні критерії:

- ✓ високий рівень захисту даних проти дешифрування та можливої модифікації;
- ✓ захищеність інформації повинна базуватися тільки на знанні ключа та не залежати від того, відомий алгоритм чи ні;
- ✓ невеликі зміни вихідного тексту або ключа має приводити до значних змін шифрованого тексту;
- ✓ область значень ключа має виключати можливість дешифрування даних шляхом перебору значень ключа;
- ✓ економічність реалізації алгоритму при достатній швидкодії;
- ✓ вартість дешифрування даних без відомого ключа має перевищувати вартість самих даних.

Відомо, що шифрування й дешифрування даних відбувається за допомогою симетричних і асиметричних криптосистем, причому до появи останніх єдиними існуючими були симетричні криптосистеми.

Симетричні криптосистеми – це спосіб шифрування, у якому один й той самий криптографічний ключ, що обирається перед обміном інформації та зберігається в секреті, застосовується як для шифрування, так і для дешифрування, при цьому інформація може шифруватися потоком або блоками [52]. Першим блоковим шифром, що широко використовується на практиці, став DES (Data Encryption Standart). Згодом з'явилося достатня кількість блокових алгоритмів - IDEA, Blowfish, радянський ДЕРЖСТАНДАРТ ГОСТ 28147-89 та інші. При блоковому шифруванні інформація розбивається на блоки й шифрується по 64 або 128 біт, або блоками змінної довжини. При цьому по особливій системі за допомогою циклів переміщення й підстановки (раундів) до кожного з блоків застосовується ключ. Лавинний ефект, що виникає в результаті повторення

раундів, призводить до втрати бітів між блоками відкритих і зашифрованих даних.

Як відомо, за класифікацією блокові шифри бувають двох основних видів: шифр перестановки (transposition, permutation, P-Блоки) і шифр заміни (підстановки, substitution, S-Блоки). Шифри перестановки в новій послідовності переставляють елементи відкритих даних (шифри горизонтальної, вертикальної, подвійної перестановки, ґрати, лабіринти, лозунгові й ін.).

Шифри заміни заміняють за певним правилом елементи відкритих даних на інші елементи. Бувають шифри простої, складної, парної заміни, буквено-складове шифрування й шифри колонної заміни. Шифри заміни діляться на дві групи: моноалфавітні й поліалфавітні. У моноалфавітних шифрах буква вихідного тексту заміняється на іншу, заздалегідь відому букву (код Цезаря). У поліалфавітних шифрах деякий символ вихідного повідомлення в кожному випадку його появи послідовно заміняють одним із символів деякого набору (шифр Відженера, циліндр Джефферсона, диск Уетстоуна, Enigma). У даний час у криптографічних системах використовуються обидва способи шифрування (заміни й перестановки). Вони більше стійкі, ніж системи, що використовують тільки заміни або тільки перестановки.

Більшість сучасних стійких симетричних алгоритмів використовують ключ довжини 64-256 біт (8-32 байта). У таблиці 1.1 наведено основні використовувані в цей час алгоритми, довжини блоку й довжини ключа.

Таблиця 1.1

Алгоритм	Довжина ключа (у бітах)	Довжина блоку (у бітах)
DES	64	64
Blowfish	Змінна, до 448 біт	64
IDEA	128	64
RC5	змінна	Змінна
ДЕРЖСТАНДАРТ ГОСТ 28147-89	256	64

Слід зазначити, що крім блокових шифрів активно використовуються і потокові шифри. Вони, як і блокові шифри, використовують симетричний ключ, але виконують шифрування вхідного потоку побайтно або іноді побітно. Ідея потокового шифру полягає в тому, що на основі симетричного ключа виробляється ключова послідовність або гама. Принцип шифрування гамуванням - це генерація гами шифру за допомогою датчика псевдовипадкових чисел і накладення отриманої гами на відриті дані оберненим образом (наприклад, використовуючи додавання за модулем 2) [10]. Процес дешифрування даних – це повторна генерація гами шифру при відомому ключу й накладення такої гами на зашифровані дані. Потокові шифри бувають із одноразовим або нескінченним ключем (infinite-key cipher), з кінцевим ключем (система Вернама - Vernam) також на основі генератора псевдовипадкових чисел (ПСЧ).

Потокові шифри, як правило, більше продуктивні, чим блокові й використовуються для шифрування мови, мережного трафіку та інших даних із заздалегідь відомою довжиною [39]. Але потокові шифри не повною мірою підходять для програмних реалізацій, оскільки шифрують і дешифрують лише по одному біту даних. Блокові ж шифри легко реалізовувати програмно, оскільки вони дозволяють уникнути значних маніпуляцій з бітами й оперують зручними для комп'ютера блоками даних. З іншого боку, потокові шифри простіші за блокові для апаратної реалізації.

Але, якими б складними та надійними не були симетричні криптографічні системи, їхнє слабке місце при практичній реалізації - проблема розподілу ключів. Для того, щоб був можливий обмін конфіденційною інформацією між двома суб'єктами інформаційних відносин, ключ повинен бути згенерований одним з них, а потім певним чином, знову ж у конфіденційному порядку, переданий іншому. Для вирішення цієї проблеми на основі результатів, отриманих класичною й сучасною алгеброю, у сімдесятих роках минулого століття було запропоновано абсолютно нову криптографію – криптографію з відкритим

ключом. Її ще називають «відкритою криптографією», «несиметричною криптографією» або «асиметричною криптографією». Саме в асиметричних алгоритмах шифрування для закриття інформації використовують один ключ (відкритий), а для розшифровування - інший (секретний) [42]. Ці ключі різні й не можуть бути отримані один з іншого. Першим алгоритмом асиметричного шифрування був алгоритм, створений Вітфілдом Діффі й Мартіном Хеллманом. Діффі й Хелман запропонували для створення криптографічних систем з відкритим ключом функцію дискретного піднесення до степеня. Автори статті не ставили за мету в даному огляді наводити математичні викладки й формули, однак слід відзначити, що криптографічна стійкість алгоритму Діффі — Хеллмана заснована на передбачуваній складності проблеми дискретного логарифмування [54]. Тобто, необерненість перетворення в цьому випадку забезпечується тим, що досить легко обчислити показникову функцію в кінцевому полі, що складається з p елементів. Обчислення ж логарифмів у таких полях - досить трудомістка операція [56].

Схожим на алгоритм Діффі — Хеллмана є алгоритм Ель-Гамаля, криптостійкість якого заснована на обчислювальній складності завдання логарифмування цілих чисел у кінцевих полях. Однак, схема Ель Гамаля має певні недоліки, серед яких- відсутність семантичної стійкості та подільність шифру. Як засіб усунення цих недоліків можна використати об’єднання схеми Ель – Гамаля з цифровим підписом Шнорра, що дозволить не тільки шифрувати повідомлення, а й аутентифікувати його.

Найвідомішим несиметричним алгоритмом на сьогоднішній день є алгоритм, запропонований Ривестом, Шаміром і Адельманом, - алгоритм RSA. Безпека алгоритму RSA заснована на труднощі вирішення задачі розкладання чисел на прості множники: як відомо, час виконання найкращих з існуючих алгоритмів розкладання, наприклад, при $n \geq 10^{145}$ виходить за межі сучасних технологічних можливостей [42]. Як варіант RSA можливо використовувати криптосистему з функцією Кармайля замість функції

Ейлера. RSA може застосовуватися не тільки для шифрування, але й для цифрового підпису. Також він використовується у відкритій системі шифрування PGP і інших системах шифрування (приміром, DarkCryptTC і формат xdc) у сполученні із симетричними алгоритмами. Але важливою проблемою практичної реалізації RSA є генерація великих простих чисел.

Отже, розглянувши основні види криптографічних систем та проаналізувавши їх особливості, можна зробити висновок, що, з огляду на наведені вище критерії, доцільно використовувати асиметричні методи шифрування даних. Хоча несиметрична криптографія є досить повільною у порівнянні зі швидкими і перевіреними часом і практикою симетричних алгоритмів, все ж таки використання несиметричної криптографії радикально спрощує процедуру розподілу ключів між учасниками інформаційних відносин. До того ж, за допомогою відкритого й секретного ключів стає можливим використання електронно-цифрового підпису. Але, в той же час, хоча описані асиметричні алгоритми дозволяють обійти проблему схованої передачі ключа, необхідність аутентифікації залишається. Без додаткових засобів, один з користувачів не може бути впевнений, що він обмінявся ключами саме з тим користувачем, який йому потрібний. Небезпека імітації в цьому випадку залишається.

РОЗДІЛ 2. ОГЛЯД КРИПТОГРАФІЧНИХ АЛГОРИТМІВ ЗАХИСТУ ІНФОРМАЦІЇ

Для організації програмної системи захисту інформації були взяті симетричні та асиметричні алгоритми. У них ми реалізуємо такі шифри: DES, ГОСТ, Blowfish та RSA. До симетричних алгоритмів відносяться: DES, ГОСТ, Blowfish. Дані шифри для кодування та декодування інформації використовують один і той же ключ. Кожен із яких має свої переваги і відповідну криптостійкість до різних видів атак. Найбільш високу криптостійсть серед симетричних алгоритмів має шифр Blowfish, який є вбудований у операційну систему UNIX [56].

До асиметричних алгоритмів відноситься шифр RSA. На даний момент часу алгоритм RSA є найбільш розвопвсюдженою криптосистемою. Даний алгоритм включений у прийняті раніше криптографічні стандарти, які виникли до його створення. Даний алгоритм є достатньо простим у розуміні та реалізації. Даний алгоритм представлений стандартом майже у всіх країнах світу. Французьке банківське співтовариствов прийняло RSA в якості стандарта, таке ж рішення прийняли і австралійці [53].

2.1. Шифр DES

Стандарт шифрування даних DES (Data Encryption Standard) опублікований в 1977 р. Національним бюро стандартів США. Стандарт DES призначений для захисту від несанкціонованого доступу до важливої, але несекретної інформації в державних і комерційних організаціях США. Алгоритм, покладений в основу стандарту, розповсюджувався достатньо швидко, і вже в 1980 р. був схвалений Національним інститутом стандартів і технологій США (НІСТ). З'являється програмне забезпечення і спеціалізовані МІКРОЕОМ, призначені для шифрування і розкодування інформації в мережах передачі даних.

До теперішнього часу DES є найбільш поширеним алгоритмом, використовуваним в системах захисту комерційної інформації.

Опишемо сам алгоритм кодування інформації шифром DES.

DES є блоковим шифром, шифруючий дані 64-бітовими блоками. З одного кінця алгоритму вводитися 64-бітовий блок відкритого тексту, а з іншого кінця виходить 64-бітовий блок шифротекста.

1. Довжина ключа рівна 56 бітам. Ключ зазвичай представляється 64-бітним числом, але кожний восьмий біт використовується для перевірки парності та ігнорується. Біти парності є найменшими значущими бітами байтів ключа. Ключ, який може бути будь-яким 56-бітним числом, можна змінити в будь-який момент часу [62].

Фундаментальним будівельним блоком DES є комбінація підстановок і перестановок. DES складається з 16 циклів (рис. 2.1).

Основні риси шифру DES визначаються перед усім тим, що він – узагальнення шифру Фейстеля і крім того, в ньому:

- число раундів S рівне 16
- довжина блоку n – 64 біти
- розмір ключа – 56 бітів
- кожний з підключів $k_1, k_2, ..., k_{16}$ нараховує 48 бітів.

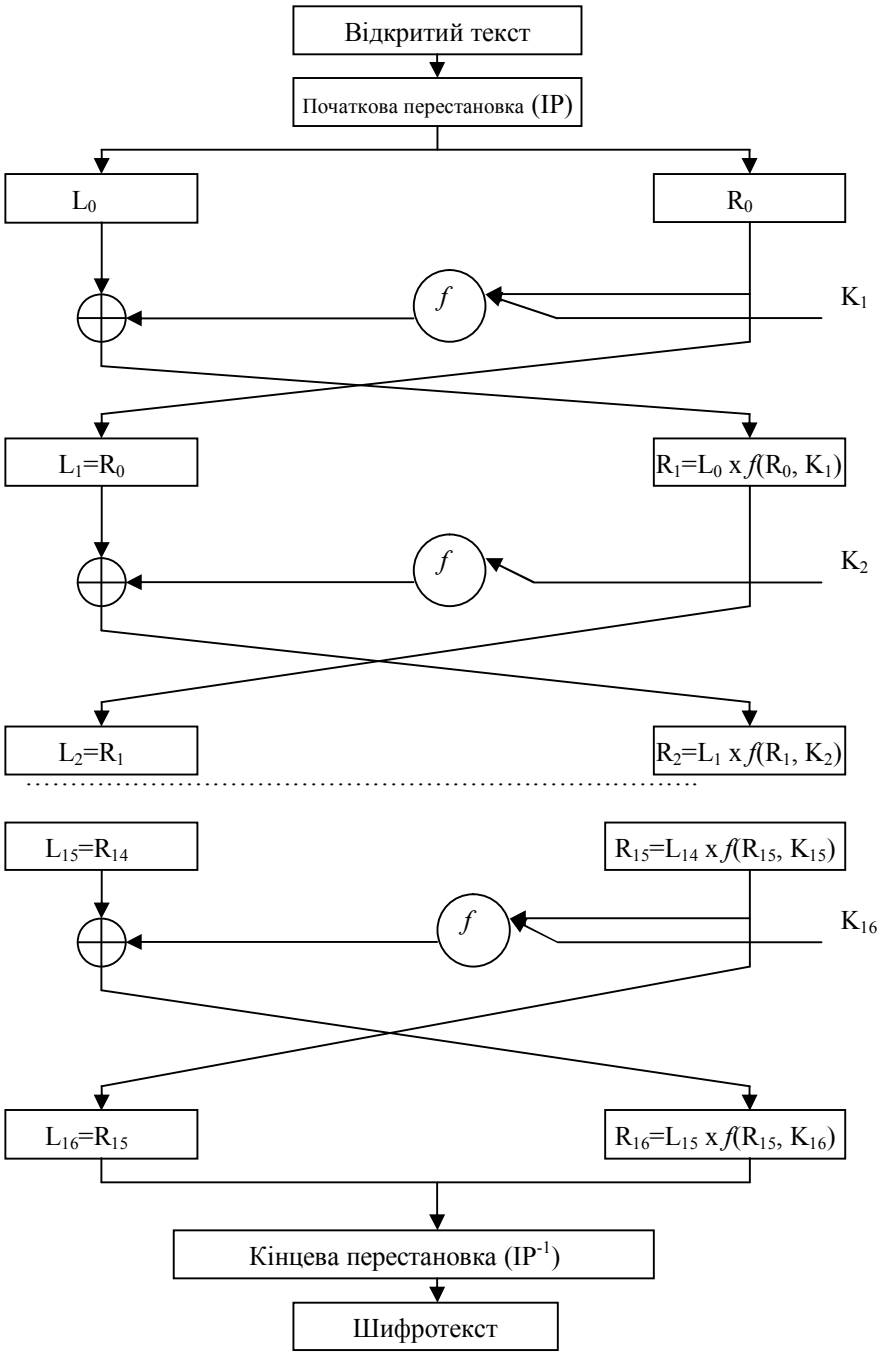

Рис. 2.1. Схема алгоритму DES

Загалом цикл перетворення представлений на рис. 2.2. Якщо L_i і R_i – ліва і права половини, отримані в результаті *i*-ї ітерації, K_i – 48-бітовий ключ для циклу *i*, а *f* – функція, що виконує всі підстановки, перестановки і XOR з ключем, то один цикл перетворення можна представити як $(L_i, R_i) = (R_{i-1}, L_{i-1}$ (XOR) $f(R_{i-1}, K_i))$.

DES є шифром Фейстеля [56] і сконструйований так, щоб виконувалася корисна властивість: для шифрування і дешифровки використовується один і

той же алгоритм. Єдина відмінність полягає в тому, що ключі повинні використовуватися в зворотному порядку.

Тобто якщо при шифруванні використовувалися ключі K_1, K_2 .,K_{16}, то ключами дешифрування будуть K_{16}, K_{15} ., K_1. Алгоритм використовує тільки стандартну арифметику 64-бітових чисел і логічні операції, тому легко реалізується на апаратному рівні.

DES працює з 64-бітовими блоками відкритого тексту. Після первинної перестановки блок розбивається на праву і ліву половини завдовжки по 32 біта. Потім виконуються 16 перетворень (функція *f*), в яких дані об'єднуються з ключом. Після шістнадцятого циклу права і ліва половини об’єднуються, і алгоритм завершується завершальною перестановкою (зворотною по відношенню до первинної). На кожному циклі (рис. 2.2) біти ключа зрушуються, і потім з 56 бітів ключа вибираються 48 бітів. Права половина даних збільшується до 48 бітів за допомогою перестановки з розширенням, об'єднується за допомогою XOR з 48 бітами зміщеного і переставленого ключа, проходить через S-блоки, утворюючи 32 нових біта, і переставляються знову. Ці чотири операції і виконуються функцією *f*.

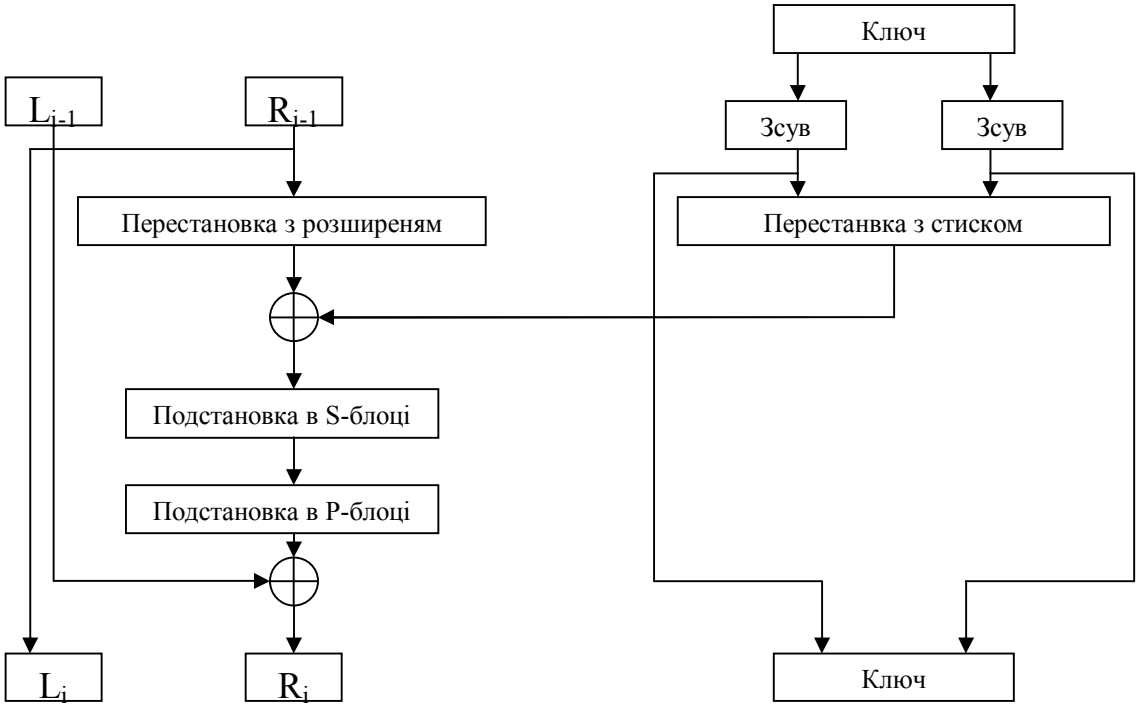

Рис. 2.2. Реалізація функції *f* при кодуванні

Потім результат функції *f* об'єднується з лівою половиною за допомогою іншого XOR. У результаті цих дій з'являється нова права половина, а стара стає новою лівою половиною. Ці дії повторюються 16 разів, утворюючи 16 циклів DES[52].

Шифр DES перетворює відкритий текст з 64 бітів наступним чином (рис. 2.1.):

- виконує початкову перестановку (IP);
- розділяє блок на ліву та праву частини;
- виконує 16 раундів з одним і тим самим набором дій;
- з'єднує половини блоку;
- виконує кінцеву перестановку (IP^{-1}).

Початкова перестановка алгоритму DES визначається таблицею 2.1. Цю, та інші таблиці, що відображають перестановки, необхідно читати зліва направо та зверху вниз. Так, число 58, що розміщується в першому рядку та першому стовпці таблиці, означає, що IP переміщує 58 біт вхідних даних на перше місце. Аналогічно, відповідно до цієї таблиці, другий біт переміщується на позицію 50, і т.д.

Таблиця 2.1. Початкова перестановка IP.

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

Зворотна перестановка задається таблицею 2.2.

В кожному раунді алгоритму DES вона складається з шести кроків:

1. Перестановка з розширенням E. Перша половина з 32 бітів розтягується до 48 бітів і перемішується. Це допомагає розсіюванню зв'язку між вхідними бітами та вихідними. Перестановка з розширенням (що відрізняється від початкової) обирається так, щоб один вхідний біт

впливав на дві заміни через S-блоки. Це допомагає поширювати залежності і створює лавинний ефект (незначна різниця між двома наборами вхідних даних перетворюється в велику на виході.) Перестановка E представлена таблицею 2.3. Кожний рядок в ній відповідає бітам, що входять в відповідний S-блок на наступному кроці[50].

Таблиця 2.2. Перестановка, зворотна до початкової.

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

Опишемо дію функції шифрування f (рис. 2.3).

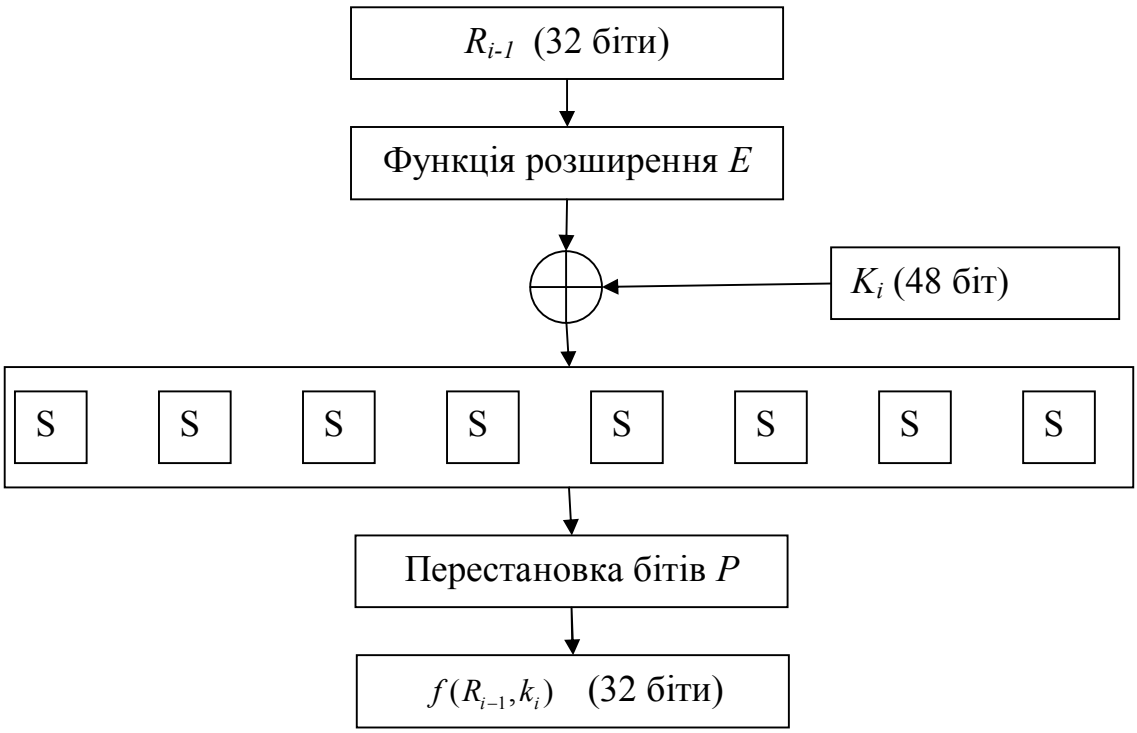

Рис. 2.3. Схема визначення функції шифрування f

Таблиця 2.3. Перестановка E.

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

2. Складання з підключем. До рядку з 48 бітів, що отримана після перестановки з розширенням, та підключа (його довжина також 48 бітів) застосовується операція виключного АБО, тобто, кожна пара відповідних бітів складається за модулем 2. Відмітимо, що підключі використовуються лише в цьому місці алгоритму.
3. Розділення. Результат попереднього кроку розкладається на 6 частин по 8 бітів в кожній.
4. S-блок. Кожний 6-бітний шматок передається в один з восьми S-блоків (блоків підстановки), де вони перетворюються в набір з 4 бітів. S-блоки – нелінійні компоненти DES і саме вони дають основний вклад в криптостійкість шифру. Кожний S-блок представляє собою пошукову таблицю з чотирьох рядків та 16 стовпців. Шість вхідних в S-блок бітів визначають, який рядок і який стовпчик необхідно використати для заміни. Перший та шостий біт задають номер рядка, а інші – номер стовпчика. Вихід S-блоку – значення відповідної клітинки таблиці. Вміст восьми S-блоків алгоритму представлено у таблиці 2.4.

Таблиця 2.4. S-блоки.

S1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	5	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	5	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8

	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	15	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	5	6	11	12	7	2	14
S5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

2. P-блок. На цей момент у нас є вісім груп 4-бітних елементів, які комбінуються тут в 32-бітний рядок та перемішуються, формуючи вихід функції f. Перестановку подано в таблиці 2.5 [45].

Таблиця 2.5. Перестановка в P-блоці.

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

На кожній ітерації використовується поточне значення ключа k_i (48 біт), що отримується з вихідного ключа k наступним чином.

Спочатку користувачі обирають сам ключ k, що містить 56 випадкових значущих бітів. Вісім бітів, що знаходяться в позиціях 8, 16, …, 64, додаються в ключ таким чином, щоб кожний байт містив непарне число одиниць. Це використовується для виявлення помилок при обміні та зберіганні ключів. Значущі 56 біт ключа піддаються перестановці, поданій в таблиці 2.6.

Таблиця 2.6. Перестановка PC-1.

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

Дана перестановка визначається двома блоками C_0 та D_0 по 28 біт в кожній (вони займають відповідно верхню та нижню частини таблиці). На наступних ітераціях визначаються наступні C_i та D_i, що отримуються одним або двома лівими циклічними зсувами відповідно таблиці 2.7.

Таблиця 2.7. Зсуви

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

Тепер, визначимо ключі $k_i, 1 \le i \le 16$. Ключ k_i складається з 48 бітів, що обираються із блоку $C_i D_i$ відповідно до таблиці 2.8 [2].

Таблиця 2.8. Перестановка PC-2.

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	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

Кінцева перестановка IP^{-1} застосовується лише до останнього блоку та використовується для відновлення позиції. Вона являється оберненою до IP. Кінцева перестановка визначається таблицею 2.9.

Таблиця 2.9. Кінцев аперестановка IP^{-1}

40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	15	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

При розшифруванні даних всі дії виконуються в зворотному порядку.

2.2. Шифр ГОСТ

Серед поширених криптографічних алгоритмів блочного типу слід назвати алгоритм ГОСТ, який широко використовується для захисту інформації у різних сферах людської діяльності.

ГОСТ – радянський та російський стандарт симетричного шифрування, введений в 1990 році. Повна назва – «ГОСТ 28147-89 Системи обробки інформації Захист криптографічний. Алгоритм криптографічного перетворення» [62].

Алгоритм ГОСТ використовує ключ розміром 256 біт, завдяки чому він має досить високу крипостійкість. Для його розкриття потрібно використати 2^{256} переборів, не враховуючи S-блоків. Він також є досить стійким до диференціального та лінійного розкриття [56].

ГОСТ передбачає три режими шифрування (проста заміна, гамування, гамування із зворотнім зв’язком) та один режим вироблення імітовставки. Перший з режимів шифрування призначений для шифрування ключової інформації та не може використовуватися для шифрування інших даних, для цього передбачені два інших режими шифрування. Режим вироблення імітовставки (криптографічної контрольної комбінації) призначений для імітозахисту даних, що зашифровуються, тобто, для захисту від випадкових чи усвідомлених несанкціонованих змін.

ГОСТ представляє 64-бітовим алгоритмом з 256-бітовим ключем. ГОСТ також використовує додатковий ключ, який розглядається нижче. Процес роботи алгоритму полягає в 32 етапах послідовного виконання простого алгоритму шифрування [52].

Для шифрування тест умовно розбивається на блоки по 64 біти. Кожен з блоків умовно складається з лівої половини L та правої половини R. На етапі і використовується ключ K_i. На етапі шифрування алгоритм ГОСТ реалізується як:

$$L_i = R_{i\text{-}1}$$

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

Етап ГОСТ показаний на рис. 2.4.

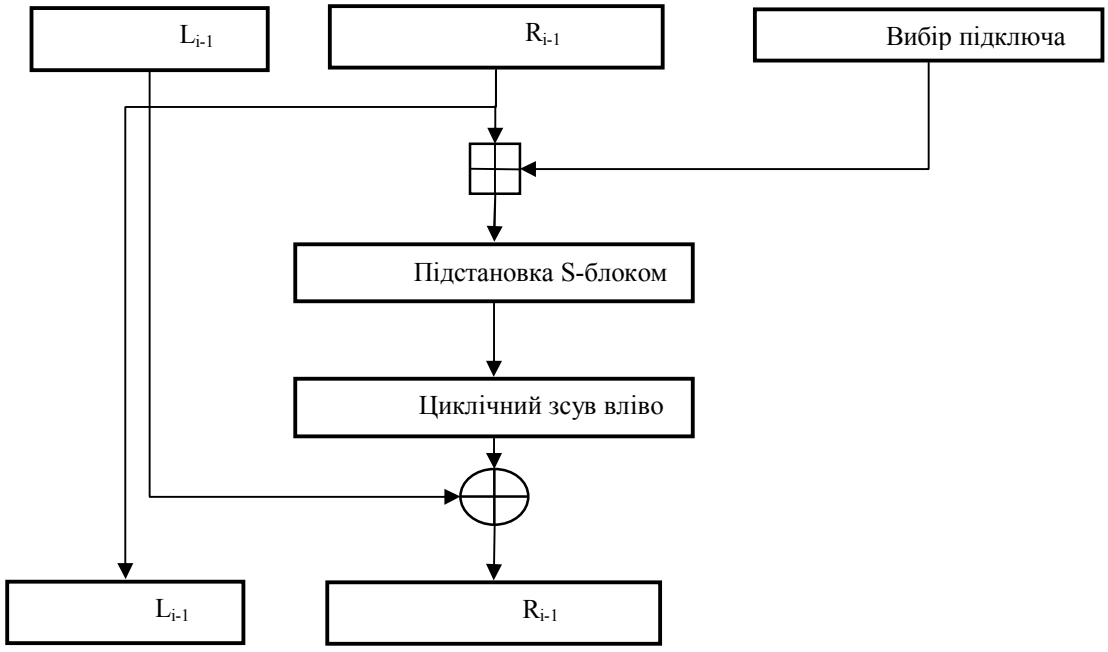

Рис. 2.4. Етап ГОСТ

Функція F проста. Спочатку права половина і i-тий підключ складаються по модулю 2^{32}. Результат розбивається на вісім 4-бітових кусочки, кожен із яких поступає на вхід його S-блоку. Використовуються вісім різних S-блоків, перших 4 біти попадають в перший S-блок, другі 4 біти

— в другій S-блок, і так далі. Кожний S-блок представляє собою перестановку чисел від 0 до 15. Наприклад, S-блок може виглядати так:

7, 10, 2, 4, 15, 9, 0, 3, 6, 12, 5, 13, 1, 8, 11

В цьому випадку, якщо на вході S-блоку 0, то на виході 7. Якщо на вході 1 то виході 10, і так далі. Всі S-блоків різні, вони фактично являються допоміжним матеріалом для ключа. Всі вісім S-блоків різні, вони фактично являються додатковим ключовим матеріалом. S-блоки повинні зберігатися у секреті.

Виходи всіх восьми S-блоків об’єднуються в 32-бітове слово, а далі всі слова циклічно зсуваються на 11 біт. Нарешті результат об'єднується за допомогою XOR з лівою половиною, і виходить нова права половина, а права половина стає новою лівою половиною. Виконується це 32 рази.

Генерація підключів проста. 256-бітовий ключ розбивається на вісім 32-бітових блоків: $k_1, k_2, ..., k_8$ [56]. На кожному етапі використовується свій підключ, як показано в табл. 2.10. Дешифрування виконується також, як і шифрування, але інвертується порядок підключів k_i.

Таблиця 2.10

Використання підключів на різних етапах ГОСТ

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

Верхній рядок в табл. 1 позначає етап, а нижній, який ключ використовується.

Стандарт ГОСТ не визначає спосіб генерації S-блоків. Виробник створює перестановки S-блоку самостійно за допомогою генератора випадкових чисел. Приклад S-блоків наведений у Таблиці 2.11.

Таблиця 2.11

S-блоків ГОСТ

S-блок 1:															
4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
S-блок 2:															
14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
S-блок 3:															
5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11
S-блок 4:															
7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
S-блок 5:															
6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
S-блок 6:															
4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
S-блок 7:															
13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
S-блок 8:															
1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

2.3. Шифр Blowfish

Blowfish оптимізований для тих додатків, в яких немає частої зміни ключів, таких як лінії зв'язку або програма автоматичного шифрування файлів. При реалізації на 32-бітових мікропроцесорах з великим кешем даних, таких як Pentium і PowerPC, Blowfish помітно швидше за DES [56]. Blowfish не підходить для використання в додатках з частою зміною ключів, наприклад, при комутації пакетів, або для використання як одно направлена хеш-функція. Великі вимоги до пам'яті роблять неможливим використання цього алгоритму в інтелектуальній платні.

Шифрування даних складається з простої функції, що послідовно виконується 16 разів. Кожен етап складається із залежної від ключа перестановки і залежної від ключа і даних підстановки. Використовуються тільки складання і XOR 32-бітових слів. Єдиними додатковими операціями на кожному етапі є чотири витягання даних з індексованого масиву [45].

У Blowfish використовується багато підключів. Ці підключі повинні бути розраховані до початку шифрування або дешифрування даних.

P-масив складається з 18 32-бітових підключів:

P1,P2..,P18

Кожний з чотирьох 32-бітових S-блоків містить 256 елементів:

$S_{1,0}, S_{1,1}, \ldots, S_{1,255}$

$S_{2,0}, S_{2,2}, \ldots, S_{2,255}$

$S_{3,0}, S_{3,3}, \ldots, S_{3,255}$

$S_{4,0}, S_{4,4}, \ldots, S_{4,255}$

Точний метод, використовуваний при обчисленні цих підключів описаний нижче.

Для шифрування тест умовно розбивається на блоки по 64 біти. Кожен з блоків умовно складається з лівої половини L та правої половини R. На етапі і використовується ключ P_i. На етапі шифрування алгоритм реалізується як:

Для i=1 до 16

$L = L \oplus P_i$

$R = F(L) \oplus R$

Переставити L і R (крім кінцевого етапу).

$R = R \oplus P_{17}$

$L = L \oplus P_{18}$

Об’єднуються L і R.

Процес шифрування проводиться у 18 етапів (рис. 2.4).

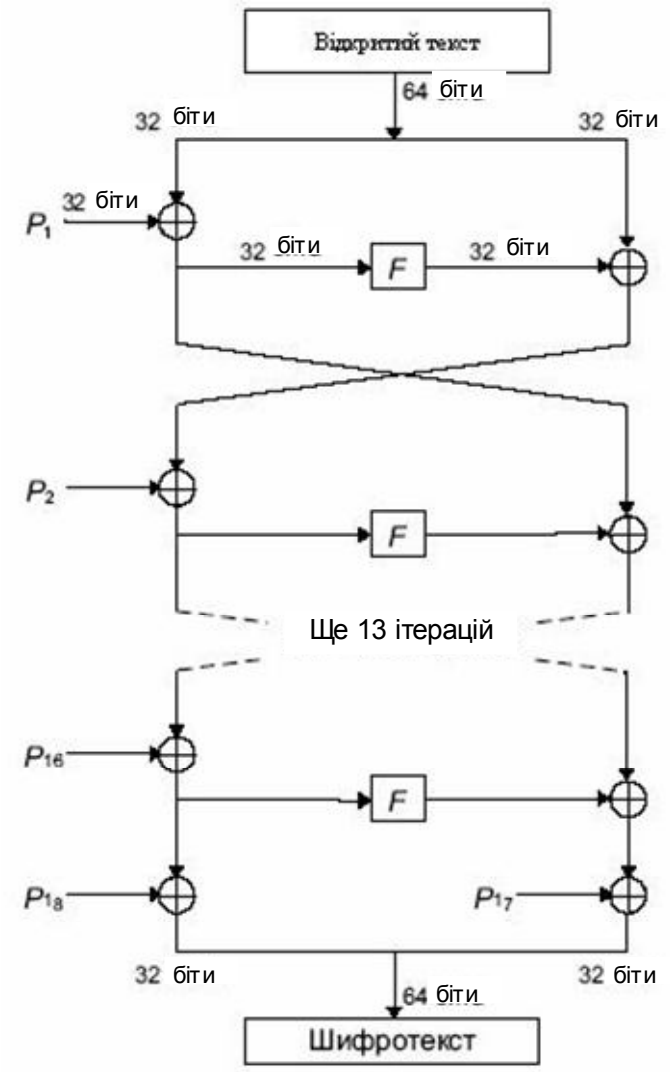

Рис. 2.4.Етапи шифрування Blowfish

Функція F є наступною(рис. 2.5):

Розділяємо L на чотири 8-бітових частини: a, b, c і d

$$F(L) = ((S_{1,a} + S_{2,b} \bmod 2^{32}) \oplus S_{3,c}) + S_{4,d} \bmod 2^{32}$$

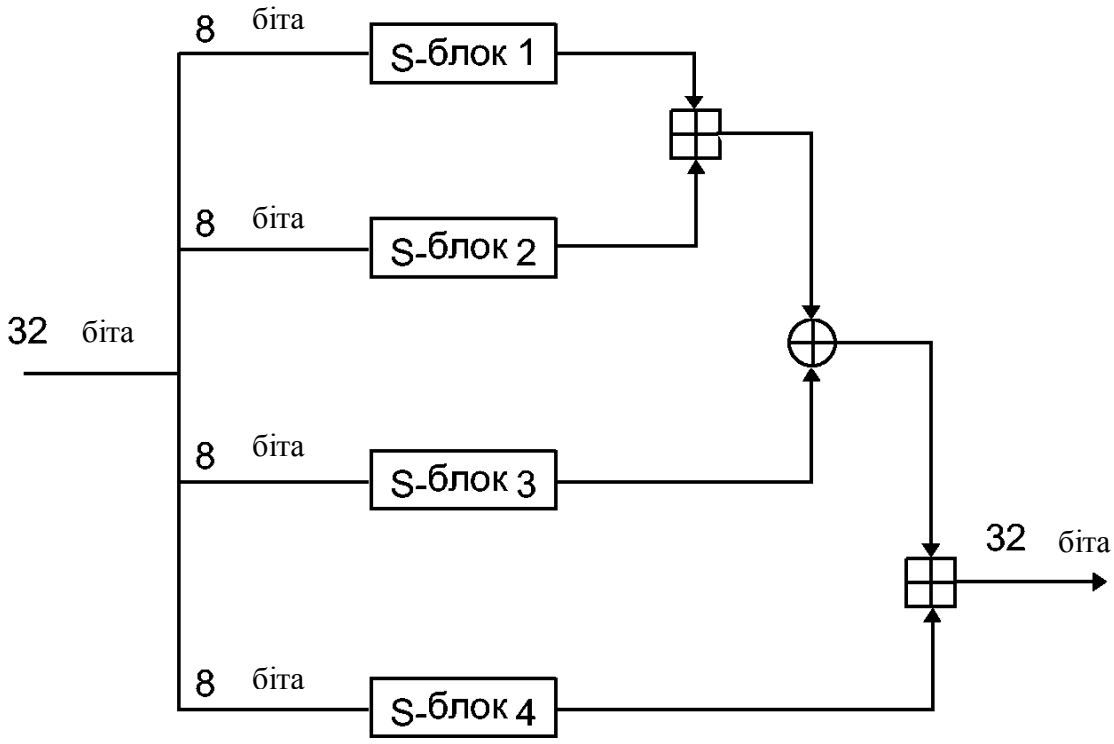

Рис. 2.5. Функція F

Дешифрування виконується точно так само, як шифрування але P_i беруться у зворотному порядку.

Blowfish — блочний шифр з ключом змінної довжини. Ключ може змінюватися в межах від 32 до 448 біт [52]. У програмній реалізації було збільшено мінімальний розмір ключа для забезпечення оптимальної надійності.

Шифрування: Blowfish - мережа Feistel, яка включає 16 ітерацій.

Реалізації Blowfish, які вимагають найбільших швидкостей, не повинні організовуватися в циклі, що гарантує присутність всіх підключів в кеші.

Вірогідність помилок в реалізації алгоритму мала, оскільки самі S блоки і P-блок дуже прості. 64-бітовий блок насправді розбивається на 32-бітові слова, що сумісно з існуючими алгоритмами. У Blowfish можна легко підвищувати розмір блоку до 128-біт, і знижувати розмір блоку. Криптоаналіз

варіантів міні-Blowfish може бути значно простіше, ніж криптоаналіз повної версії [56].

Мережа Feistel, складова тіла Blowfish, розроблена так, щоб при найбільшій простоті вона зберігала свої криптографічні властивості.

У проекті алгоритму існує два способи гарантувати, що ключ є достатньо довгим, щоб досягти потрібного рівня захисту. Перший в тому, щоб ретельно розробити алгоритм так, щоб ентропія випадково введеного ключа зберігалася, оскільки тоді не існує іншого методу криптоаналізу, окрім атаки в лоб". Інший спосіб - в збільшенні довжини ключа настільки, що зменшення ефективності ключа на декілька бітів не дає великих переваг. Оскільки Blowfish розроблений для великих процесорів з великим розміром пам'яті ми вибрали останній спосіб.

Менша кількість ітерацій, імовірно, можливе зменшення кількості ітерацій з 16 до 8 без сильного ослаблення захисту. Число ітерацій потрібних для захисту може залежати від довжини ключа. Звертаємо увагу на те, що з наявною процедурою породження підключа, алгоритм з 8 ітераціями не може обробляти ключ завдовжки більше 192 біт [45].

Безперервне обчислення підключа. Поточний метод обчислення підключів вимагає, щоб всі підключі обчислювалися перед будь-яким шифруванням. Фактично неможливо обчислити останній підключ останнього S-блока без обчислення перед цим попереднього ключа. Був би привабливіший альтернативний метод обчислення підключа: коли кожен підключ може бути обчислений незалежно від іншого підключа. Високоякісні реалізації могли б заздалегідь обчислювати підключі для вищої швидкості, але реалізації з низькими вимогами можуть обчислювати підключі, коли це необхідно.

2.4. Шифр RSA

Відомим прикладом криптосистеми з відкритим ключом (асиметричної криптосистеми) є криптосистема RSA, розроблена в 1977 році і отримала назву в честь її творців: Рівеста, Шаміра й Ейдельмана [62].

RSA став першим алгоритмом придатним і для шифрування і для цифрового підпису.

Безпека алгоритму RSA побудована на принципі складності факторизації [42]. Алгоритм використовує два ключі — відкритий (public) і секретний (private), разом відкритий і відповідний йому секретний ключі утворюють пари ключів (keypair). Відкритий ключ не потрібно зберігати в таємниці, він використовується для шифрування даних. Якщо повідомлення було зашифровано відкритим ключом, то розшифрувати його можна тільки відповідним секретним ключом.

Для того, щоб згенерувати пари ключів виконуються такі дії:

1. Вибираються два великих простих числа p і q.

Для знаходження двох великих простих чисел p і q, при генерації ключа, звичайно використовуються імовірнісні тести чисел на простоту, які дозволяють швидко виявити й відкинути складні числа.

Для генерації p і q необхідно використовувати криптографічно надійний генератор випадкових чисел. У порушника не має бути можливості одержати будь-яку інформацію про значення цих чисел.

p і q не повинні бути занадто близькими одне до одного, інакше можна буде знайти їх використовуючи метод факторизації Ферма [52]. Крім того, необхідно вибирати «сильні» прості числа, щоб не можна було скористатися p-1 алгоритмом Поларда.

2. Обчислюється їх добуток $n = pq$.

Число n повинно мати розмір не менше 512 біт. З 2007 року система шифрування на основі RSA вважається надійною, починаючи з величини n в 1024 біта.

3. Обчислюється Функція Ейлера $\varphi(n) = (p-1)(q-1)$.

4. Вибирається ціле e таке, що $1 < e < \varphi(n)$ та e взаємно просте з $\varphi(n)$.

5. За допомогою розширеного алгоритму Евкліда знаходиться число d таке, що $ed \equiv 1(\mathrm{mod}\varphi(n))$.

Число n називається модулем, а числа e і d — відкритою й секретною експонентами, відповідно. Пари чисел (n,e) є відкритою частиною ключа, а (n,d) — секретною. Числа p і q після генерації пари ключів можуть бути знищені, але в жодному разі не повинні бути розкриті [10].

Вибір значень відкритого та секретного показників.

RSA працює значно повільніше симетричних алгоритмів. Для підвищення швидкості шифрування відкритий показник e вибирається невеликим, звичайно 3, 17 або 65537. Ці числа у двійковому вигляді містять тільки по дві одиниці, що зменшує число необхідних операцій множення при піднесенні до степеня. Наприклад, для піднесення числа m до степеня 17 потрібно виконати тільки 5 операцій множення:

$$m^2 = m * m;\ m^4 = m^2 * m^2;\ m^8 = m^4 * m^4;\ m^{16} = m^8 * m^8;\ m^{17} = m^{16} * m.$$

Вибір малого значення відкритого показника може призвести до розкриття повідомлення, якщо воно відправляється відразу декільком одержувачам, але ця проблема вирішується за рахунок доповнення повідомлень.

Значення секретного показника d повинно бути досить великим. У 1990 році Міхаель Вінер (Michael J. Wiener) показав, що якщо $q < p < 2q_i$ і $d < \frac{n^{\frac{1}{4}}}{3}$, то є ефективний спосіб обчислити d по n і e. Однак, якщо значення e вибирається невеликим, то d виявляється досить великим і проблеми не виникає [42].

Шифрування й дешифрування.

Для того, щоб зашифрувати повідомлення $m < n$ обчислюється

$c = m^e \bmod n$.

Число c і використовується в якості шифротексту. Для дешифрування потрібно обчислити

$c = m^d \bmod n$.

Неважко переконатися, що при розшифруванні ми відновимо вихідне повідомлення:

$c^d \equiv (m^e)^d \equiv m^{ed} (\bmod n)$

З умови
$ed \equiv 1(\bmod \varphi(n))$
виходить, що
$ed = k\varphi(n) + 1$ для деякого цілого k, отже
$m^{ed} \equiv m^{k\varphi(n)+1} (\bmod n)$.

Згідно теореми Ейлера:

Теорема Ейлера:

Нехай, а та n – натуральні числа. Тоді

$НОД(a,n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1(\bmod n),$

Де функція Ейлера $\varphi(n)$ підраховує кількість натуральних чисел від 1 до n, взаємно простих з n. Функцію φ можна визначити з рівності:

$$\varphi(n) = n\prod_{p|n,\, p\ просте}(1-\frac{1}{p})$$ [18].

$m^{\varphi(n)} \equiv 1(\bmod n)$, тому $\begin{matrix} m^{k\varphi(n)+1} \equiv m(\bmod n) \\ c^d \equiv (\bmod n) \end{matrix}$.

При практичному використанні необхідно деяким чином доповнювати повідомлення. Відсутність доповнень може призвести до деяких проблем:

- значення $m = 0$ і $m = 1$ дадуть при шифруванні шифротексти 0 і 1 при будь-яких значеннях e і n.
- при малому значенні відкритого показника ($e = 3$, наприклад) можлива ситуація, коли виявиться, що $m^e < n$. Тоді $c = m^e \bmod n = m^e$, і зловмисник легко зможе відновити вихідне повідомлення обчисливши корінь ступеня e з c.
- оскільки RSA є детермінованим алгоритмом, тобто не використовує випадкових значень у процесі роботи, то зловмисник може використати атаку з обраним відкритим текстом.

Для розв'язання цих проблем повідомлення доповнюються перед кожним шифруванням деяким випадковим значенням. Доповнення виконується таким чином, щоб гарантувати, що $m \neq 0$, $m \neq 1$ і $m^e > n$. Крім того, оскільки повідомлення доповнюється випадковими даними, то зашифровуючи той самий відкритий текст ми щораз будемо одержувати інше значення шифротексту, що робить атаку з обраним відкритим текстом неможливою.

Цифровий підпис.

RSA може використовуватися не тільки для шифрування, але й для цифрового підпису[15]. Підпис s повідомлення m обчислюється з використанням секретного ключа за формулою:

$s = m^d \bmod n$.

Для перевірки правильності підпису потрібно переконатися, що виконується рівність

$m = s^e \bmod n$.

Стійкість даної системи основана на складності зворотності степеневої функції в кільці вирахувань цілих чисел по зіставному модулю n (при належному виборі модуля) [45].

Система RSA використовується для захисту програмного забезпечення й у схемах цифрового підпису. Також вона використовується у відкритій системі шифрування PGP.

Через низьку швидкість шифрування (близько 30 кбіт/сек при 512 бітному ключі на процесорі 2 ГГц), повідомлення звичайно шифрують за допомогою більш продуктивних симетричних алгоритмів з випадковим ключем (*сеансовий ключ*), а за допомогою RSA шифрують лише цей ключ.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ

3.1. Опис алгоритму реалізації системи

Алгоритм реалізації системи захисту інформації можна умовно розділити на декілька етапів:

- ✓ Вибір сервера баз даних (БД) і організація даних про користувачів системи;
- ✓ Налаштування програмного середовища для роботи із БД;
- ✓ Організація відправки електронних листів користувачам системи, що включають шифрований файл (ключ);
- ✓ Організація технології обміну ключами, перевірка цифрового підпису ключа на основі шифру RSA;
- ✓ Організація методів потокової взаємодії із файлами довільного типу;
- ✓ Організація процесу кодування даних шифром DES;
- ✓ Організація процесу кодування даних шифром ГОСТ;
- ✓ Організація процесу кодування даних шифром Blowfish;

На початковому етапі розробки системи було сформовано БД. В якості сервера БД обрано Microsoft Access 2003, який є найбільш поширеним і зручним при використанні для користувачів.

У базі даних зберігається інформація про користувачів системи захисту інформації. У БД зберігається поля із наступною інформацією:

- ➢ прізвище;
- ➢ ім’я;
- ➢ по батькові;
- ➢ email;
- ➢ відкритий ключ n;
- ➢ відкритикй ключ e;

Заповнена таблиця зберігає (рис.3.1) необхідну інформацію про користувачів у системі захисту інформації. Поле у якому міститься електронна адреса кожного із користувачів призначене для автоматизованої відправки електронних листів на їх адреси. Ключі, що зберігаються у БД призначені для кодування файлу, що додається до листа.

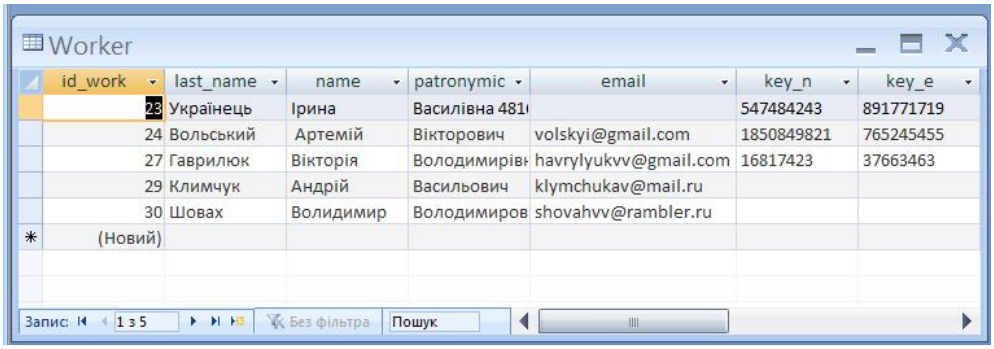

Рис. 3.1. Користувачі у базі даних

Загалом файл, що прикріпляється до листа — це зашифрований файл за допомогою шифру RSA. У файл можна записувати довільну текстову інформацію, що реалізується у системі захисту інформації. Даний файл, що надсилається можна дешифрувати тільки закритим ключом кожного із користувачів системи, для якого цей файл відправлено. Ключ дешифрується закритим ключом кожного користувача.

Така технологія організації відправки листів дозволяє організувати обмін ключами для шифрів DES, ГОСТ та Blowfish. Для даних шифрів ключ повинен знаходитися в таємниці. Для даних шифрів ключ, що використовується для шифрування та дешифрування інформації. У шифрі RSA є відкриті ключі для шифрування та закритий для дешифрування. Надійність шифру вважається найбільш криптостійкою, тому обмін ключами реалізується на основі даного шифру захисту інформації.

Відправка листа реалізована за допомогою smtp сервера електронної пошти. Для розсилки повідомлень користувачам БД у системі потрібно вказати наступні налаштування електронного ящика:

- ✓ smtp сервер;
- ✓ smtp порт;
- ✓ email;
- ✓ пароль.

Тобто потрібно вказати налаштування власної електронної пошти кожного із користувачів системи, щоб можна було проводити відправку листів. Відправка листа на мові програмування C# має наступний вигляд:

```
private void SendMail(string toAdd, string temaEmail, string textEmail)
{
    string smtpServer = "";
    string smtpPort = "";
    string email = "";
    string password = "";
    //Читаємо параметри із файлу налаштувань електронної пошти
    try
    {
        FileStream myFileStream = new FileStream("email.ini", FileMode.Open,
            FileAccess.Read);
        BinaryReader binReade = new BinaryReader(myFileStream);
        smtpServer = CryptionClass.Decrypt(binReade.ReadString(), "34msrti98iew");
        smtpPort = CryptionClass.Decrypt(binReade.ReadString(), "34msrti98iew");
        email = CryptionClass.Decrypt(binReade.ReadString(), "34msrti98iew");
        password = CryptionClass.Decrypt(binReade.ReadString(), "34msrti98iew");
        binReade.Close();
        myFileStream.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Помилка при відкритті файлу, що містить параметри підключення до
email " + ex.ToString(), "Помилка!");
    }

    SmtpClient Smtpotpravka = new SmtpClient();
    try
    {
        Smtpotpravka = new SmtpClient(Convert.ToString(smtpServer), int.Parse(smtpPort));
        Smtpotpravka.Credentials = new NetworkCredential(email, password);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Проблема при підключенні до електронної пошти. " +
ex.Message.ToString());
    }
    if (toAdd == "")
        return;
    //Відпрвка пошти із електронного адресу
    try
    {
        MailMessage Email = new MailMessage();
        Email.From = new MailAddress(email);
        Email.To.Add(new MailAddress(toAdd));
        Email.Subject = temaEmail;
        Email.Body = textEmail;
        Attachment attachData = new Attachment("Ключ.key");
        Email.Attachments.Add(attachData);
        Smtpotpravka.Send(Email); // Отправляем сообщения
    }
    catch (Exception ex)
    {
        MessageBox.Show("Проблема при відправці листа. " + ex.Message.ToString(),
temaEmail);
    }

}
```

Наступною задачою алгоритму є організація кодування інформації довільного типу за допомогою симетричних шифрів DES, ГОСТ та Blowfish. Для читання і запису файлів довільного типу використовуються потоки для роботи із двійковими даними. При цьому розуміється читання і запис

послідовності бітів, де файл не має значення. Це реалізовано наступним програмним кодом на мові програмування C#:

```
myFileCode = dlgCode.FileCode;
myFileDecode = dlgCode.FileDecode;
myKey = dlgCode.Key;
int temp = 0;
int leng = 8;
byte[] key = new byte[leng];

FileStream readKey = null;
BinaryReader binKey = null;
try
{
    readKey = new FileStream(myKey, FileMode.Open, FileAccess.Read);
    binKey = new BinaryReader(readKey);
    for (temp = 0; temp < leng; temp++)
    {
        key[temp] = binKey.ReadByte();
    }
    binKey.Close();
}
catch (IOException errorKey)
{
    MessageBox.Show("Помилка при відкриті файлу, який містить ключ!!!\n" +
errorKey.Message, "Помикла!");
    is_good = false;
}
finally
{
    if (readKey != null)
        readKey.Close();
}
if (temp != leng && is_good)
{
    MessageBox.Show("Розмір файлу, який містить ключ є надто малим!!!", "Помилка!");
    is_good = false;
}
FileStream stream = null;         //Файл, який потрібно читати
FileStream streamWrite = null;    //Файл в який потрібно записувати
BinaryReader reader = null;
BinaryWriter writer = null;
if (is_good)
{
    try
    {
        stream = new FileStream(myFileCode, FileMode.Open, FileAccess.Read);
        streamWrite = new FileStream(myFileDecode, FileMode.Create, FileAccess.Write);
        reader = new BinaryReader(stream);
        writer = new BinaryWriter(streamWrite);
        byte[] data = new byte[8];
        DES my_des = new DES();
        my_des.des_key(ref key);
        long processed = 0;
        long fileSize = stream.Length;
        long lushok = fileSize % 8;
        fileSize -= lushok;
        int i = 0;
        while (processed < fileSize)
        {
            for (i = 0; i < 8; i++)
                data[i] = reader.ReadByte();
            my_des.des_enc(ref data);
```

```
            for (i = 0; i < 8; i++)
                writer.Write(data[i]);
                            processed += 8;    //Переміщаємося на 8 біт
        }
        for (i = 0; i < lushok; i++)
            writer.Write(reader.ReadByte());
    }
    catch (IOException expt)
    {
        MessageBox.Show("Помилка при відкриті файлу, який потрібно кодувати!!!\n" +
expt.Message, "Помикла!");
        is_good = false;
    }
    finally
    {
        if (stream != null)
        {
            stream.Close();
        }
        if (streamWrite != null)
            streamWrite.Close();
        if (is_good)
        {
            //Фуксуємо, що кодування проведено шифром DES
            action = 1;           //Кодування файлу проведенно успішно
        }
        timeEnd = DateTime.Now;
        this.Invalidate();
    }
}
```

Показано приклад реалізації для шифру DES. Для шифрування використовується спеціальний об’єкт `DES my_des = new DES()`. У якому реалізовано основний механізм шифрування та дешифрування інформації даним шифром.

Аналогічно обробку файлів реалізовано за допомогою шифрів ГОСТ та Blowfish. У даному випадку різницю має спосіб шифрування даних та розмір відповідних ключів для даних шифрів.

Тами чином, на основі організації такого алгоритму вдалося досягти великої гнучкості системи захисту інформації, що передбачає мережеву взаємодію із надійністю та зручністю обміну інформацією із забезпеченням відповідної потрібної надійності усієї системи.

3.2. Середовище розробки

Розробка програм проводилася у середовищі Visual Studio 2010, що дозволило розробити ефективну систему програмного захисту інформації. Представимо короткий опис основних можливостей та переваг середовища

розробки Visual Studio [48]. Вкажемо також процес створення та розгортки системи захисту інформації.

Visual Studio - це набір інструментів розробки, заснованих на використанні компонентів, і інших технологій для створення потужних, продуктивних додатків. Крім того, середа Visual Studio оптимізована для спільного проектування, розробки та розгортання корпоративних рішень.

Середовище розробки Visual Studio представляє собою повний набір засобів розробки для створення веб-додатків ASP.NET, XML (веб-служби), настільних та мобільних додатків [46]. Visual Basic, Visual C# і Visual C++ використовують єдине інтегроване середовище розробки (IDE), яка дозволяє спільно використовувати засоби і спрощує створення рішень на базі декількох мов. Крім того, в цих мовах використовуються функціональні можливості платформи. NET Framework [29], яка дозволяє отримати доступ до ключових технологій, що спрощує розробку веб-додатків ASP і XML (веб-служби).

Програма розроблена на мові програмування C#. Синтаксис C # дуже виразний, але простий у вивченні. Усі, хто знайомий з мовами C, C + або Java з легкістю впізнають синтаксис з фігурними дужками, характерний для мови C#. Розробники, які знають будь-який з цих мов, як правило, зможуть добитися ефективної роботи з мовою C# за дуже короткий час. Синтаксис C# робить простіше те, що було складно в C++, і забезпечує потужні можливості, такі як типи значень Nullable, перерахування, делегати, лямбда-виразу і прямий доступ до пам'яті, чого немає в Java.

Програма на мові C# виконується в середовищі. NET Framework - інтегрованому компоненті Windows, що містить віртуальну систему виконання (середовище CLR) і уніфікований набір бібліотек класів [41]. Середовище CLR представляє собою комерційну реалізацію Майкрософт інфраструктури CLI (common language infrastructure), міжнародного стандарту, основи середовищ виконання і розробки з тісною взаємодією мов і бібліотек.

Опишемо процес створення програми за допомогою середовища Visual Studio 2010 на основі Windows Forms, оскільки наша програма представлена у вигляді стандартної програми Windows.

Проект програми Windows Forms є основою більшості рішень, що включають Windows Forms. Такий проект просто створити в інтегрованому середовищі розробки (IDE) [46].

Створення проекту програми Windows Forms:

- ✓ Запустіть Visual Studio 2010.
- ✓ У меню Файл виберіть команду **Создать** та виберіть **Проект**.
- ✓ Відкриється діалогове вікно **Новый проект**.
- ✓ На панелі Установленные шаблоны розгорніть Visual C#, оберіть **Windows**.
- ✓ Над середньою областю в списку потрібну версію NET Framework.
- ✓ У середній області (рис. 3.2) виберіть шаблон **Приложение Windows Forms**.
- ✓ У текстовому полі **Имя** задайте ім'я проекту.
- ✓ У текстовому полі **Расположение** вкажіть каталог, в якому потрібно зберегти проект.
- ✓ Натисніть кнопку **ОК**.

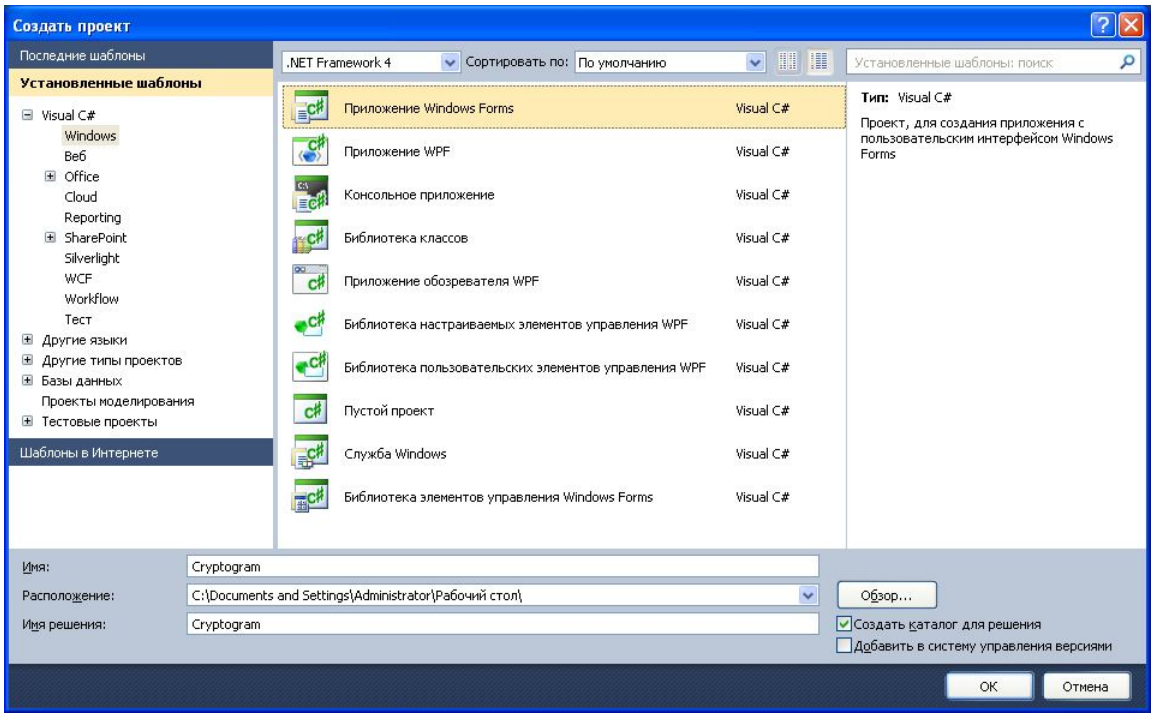

Рис. 3.2. Створення програми Windows Forms

Таким чином створюється пустий проект, який містить у собі форму, яка запускається автоматично, тобто є головним вікном програми (рис 3.3).

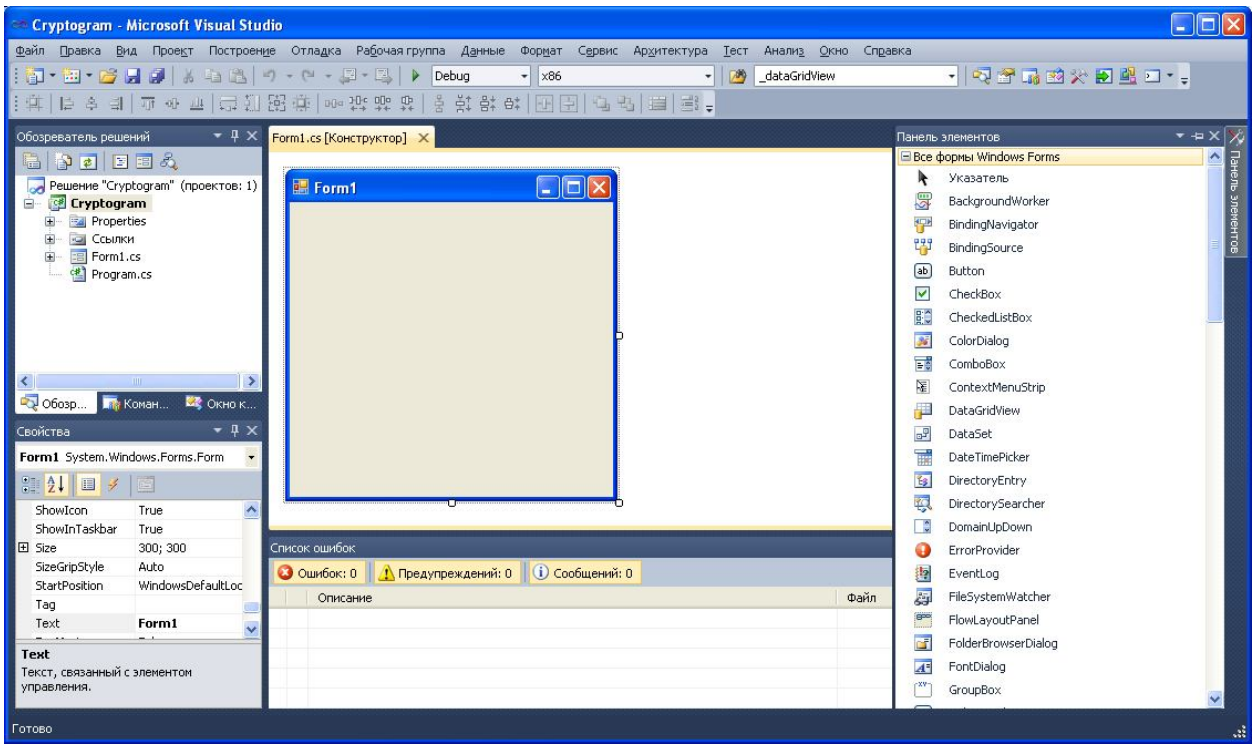

Рис. 3.3. Новостворений проект рішення

Після того, як створено проект варто звернути увагу на можливості роботи і організації інтерфейсу користувача, який має великий набір властивостей та інтерактивних компонентів.

Windows Forms є технологією інтелектуальних клієнтів для NET Framework; це набір керованих бібліотек, що забезпечують поширені завдання програм, наприклад читання і запис у файлову систему. За допомогою середовища розробки типу Visual Studio можна створювати додатки Windows Forms, які відображають інформацію, запитують введення від користувачів і обмінюються даними з віддаленими комп'ютерами по мережі.

У Windows Forms форма є видимою поверхнею, на якій відображається інформація для користувача. Зазвичай додаток Windows Forms будується шляхом поміщення елементів керування на форму і написанням коду для реагування на дії користувача, такі як клацання миші або натиснення клавіш.

Елемент управління - це окремий елемент користувацького інтерфейсу, призначений для відображення або вводу даних.

При виконанні користувачем будь-якої дії з формою або одним з її елементів управління, створюється подія. Додаток реагує на ці події за допомогою коду і обробляє події при їх виникненні.

Windows Forms включає широкий набір елементів керування, які можна додавати на форми: текстові поля, кнопки, розкриваємі списки, перемикачі і навіть веб-сторінки. Список всіх елементів керування, які можна використовувати у формі. Якщо існуючий елемент управління не задовольняє потребам, в Windows Forms можна створити власні настроювані елементи керування за допомогою класу UserControl.

До складу Windows Forms входять елементи призначеного для користувача інтерфейсу з розширеними функціями, відповідними можливостями потужних додатків, таких як Microsoft Office. Використовуючи елементи управління ToolStrip і MenuStrip, можна створювати панелі інструментів і меню, що містять текст і малюнки, що відображають підменю та містять в собі інші елементи керування, такі як текстові поля і поля з випадним списком.

За допомогою конструктора Windows Forms Visual Studio, що підтримує перетягування, можна легко створювати додатки Windows Forms: Досить виділити елемент керування курсором і помістити його на потрібне місце на формі. Конструктор надає такі засоби, як лінії сітки і "прив'язка ліній" для подолання труднощів вирівнювання елементів управління. І в разі використання Visual Studio або компіляції з командного рядка можна використовувати елементи керування FlowLayoutPanel, TableLayoutPanel і SplitContainer для створення просунутих розміток форми за мінімальний час і з мінімальними зусиллями [48].

Після розробки усієї системи проект був наповнений (рис. 3.4) відповідними вікнами, класами, елементами керування та ін..

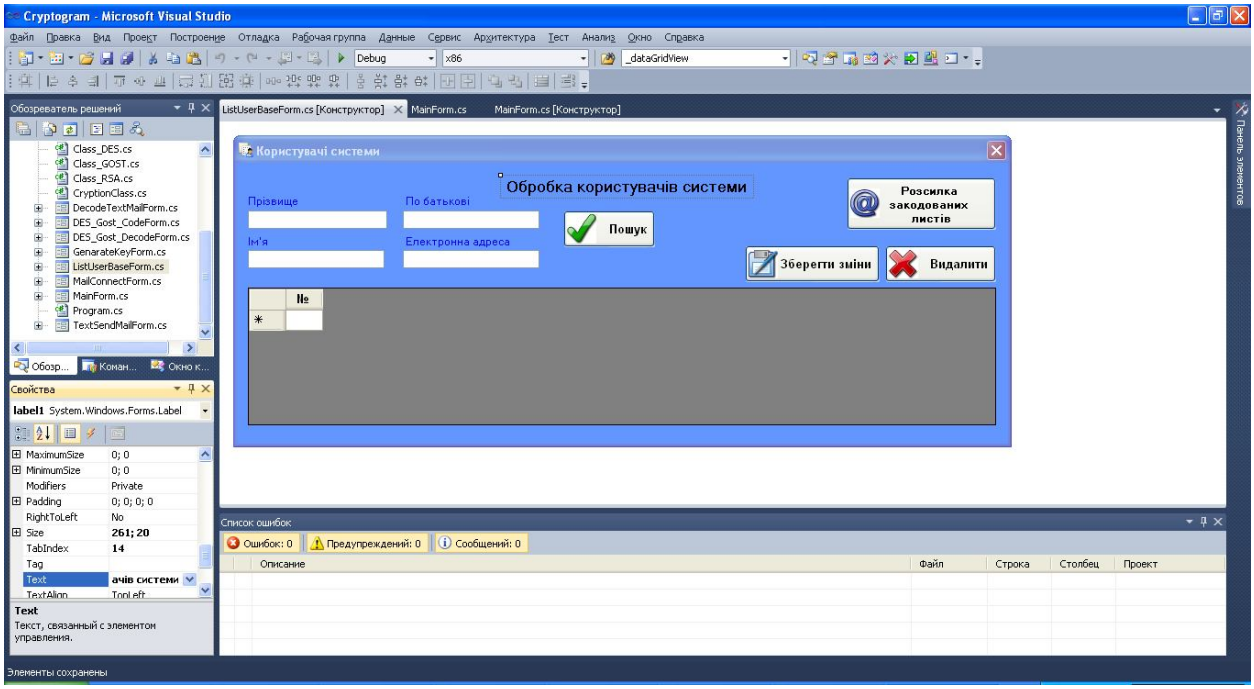

Рис. 3.4. Розробка системи захисту інформації

Виходячи з вище зазначеного, розроблено програмну систему захисту інформації у середовищі Visual Studio NET. Ефективна система інформування та допомоги візуального представлення елементів надзвичайно сильно допомагає розробляти та реалізувати програмні системи різної складності та гнучкості функціонування.

3.3. Інструкція користувачеві по використанню

Програмну систему захисту інформації розроблено на базі додатків Windows Forms. Середовищем розробки є Visual Studio 2010. Мова програмування C#.

Головне вікно програми (рис. 3.5) представляє собою стандартний додаток Windows. Головне вікно складається із заголовка, верхнього меню, робочої області та стрічки стану.

Верхнє меню містить у собі основні команди роботи програми. Верхнє меню включає в себе меню: Файл, Кодування, Налаштування.

Для повної функціональності програми потрібно провести налаштування системи захисту інформації. Для цього призначене (Рис. 3.6) меню **Налаштування**.

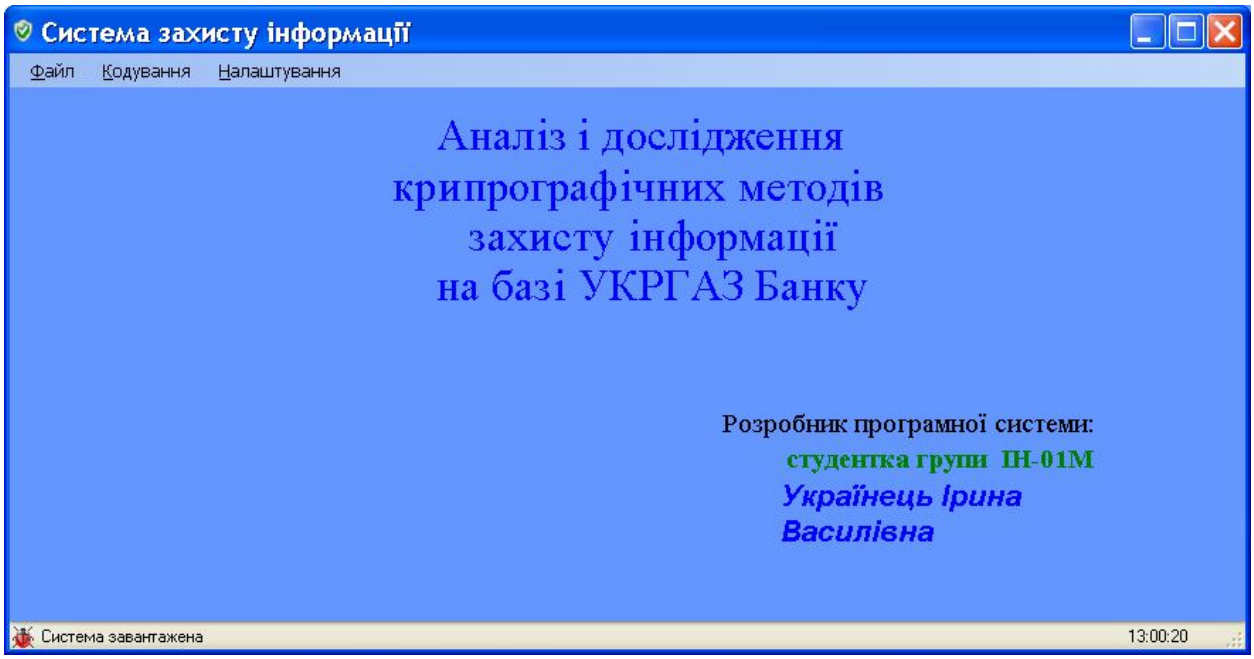

Рис. 3.5. Головне вікно програми

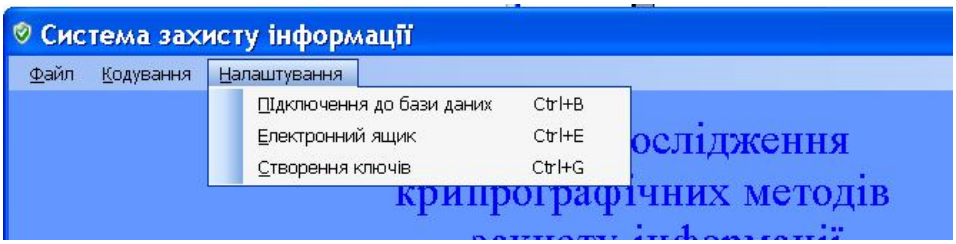

Рис. 3.6. Вміст меню **Налаштування**

На початковій стадії роботи із програмою потрібно вказати розміщення бази даних. Для цього потрібно виконати команду **Налаштування** → **Підключення до бази даних**. Після цього буде відображено (рис. 3.7) вікно **Підключення до бази даних**.

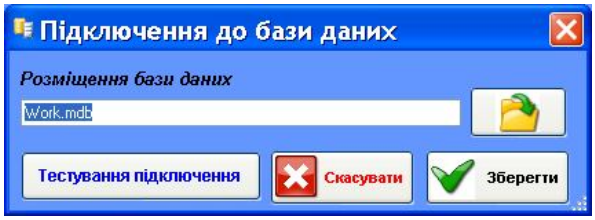

Рис. 3.7. Налаштування підключення до БД

У полі *Розміщення бази даних* потрібно вказати файл бази даних, якщо шлях до файлу не вказано, то база даних буде шукатися у каталозі програми. Кнопка  дозволяє поводити пошук файлу (рис. 3.8.) у операційні системи за допомогою стандартного вікна для Windows відкриття файлу.

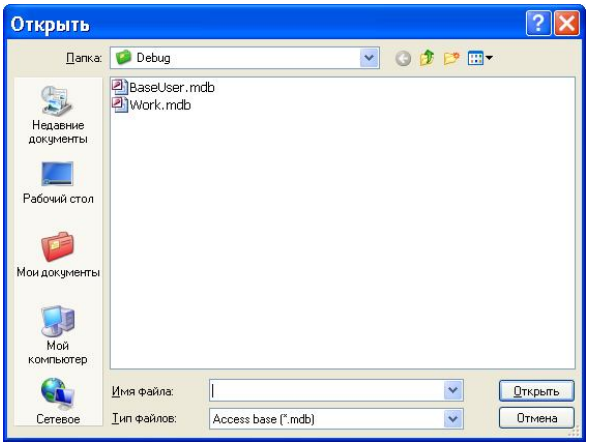

Рис. 3.8. Пошук бази даних для системи захисту інформації

У вікні **Підключення до бази даних** кнопки мають наступне призначення:

- *Тестування підключення* — перевірка можливості підключення.
- *Скасувати* — закрити вікно і не зберігати зроблені зміни.
- *Зберегти* — закрити вікно і зберегти зміни.

Налаштування підключення до БД зберігаються у файлі ***sql.ini***.

Для налаштування підключення до електронної пошти із якої буде відправлятися лист потрібно виконати команду **Налаштування → Електронний ящик**. Після цього буде відображено (рис. 3.9) вікно **Підключення до електронного ящика**.

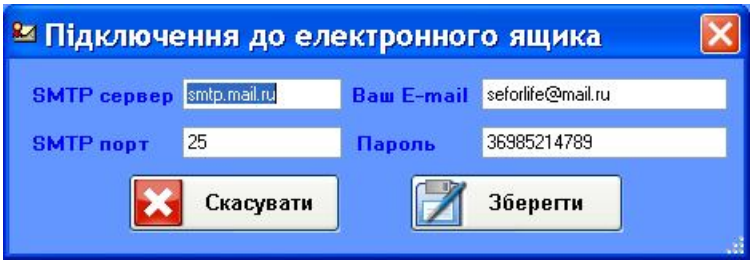

Рис. 3.9. Параметри підключення до БД

Створення ключів відкритих ключів та закритого проводиться за допомогою їх створення на основі правили алгоритму RSA. Для створення ключів потрібно викликати команду **Налаштування → Створення ключів.** Перед користувачем з’явиться (рис. 3.10) діалогове вікно **Створення ключів для шифру RSA.**

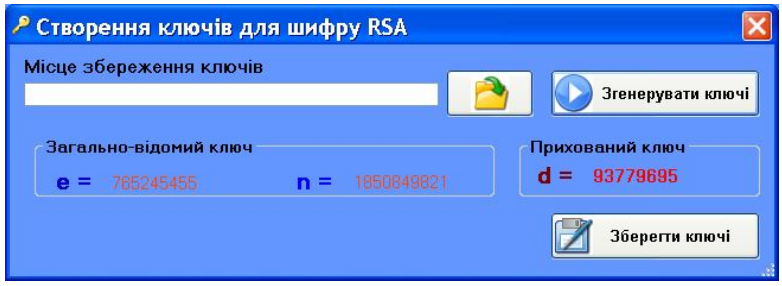

Рис. 3.10. Створення та збереження ключів

По-замовчувані програма проводить читання ключів із файлу, де було попередньо їх збережено. При натиску на кнопку ***Генерування ключів*** програма випадковим чином створює ключі і користувачеві їх відображає. При натиску на кнопку ***Зберегти ключі*** програма проводить збереження ключів і відображає відповідне повідомлення (рис. 3.11) про успішність виконання операції.

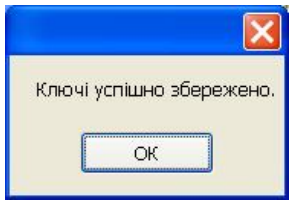

Рис. 3.11. Успішне збереження ключів

Після проведення усіх необхідних налаштувань програми можна приступати до заповнення бази даних користувачів програми. Для цього потрібно виконати команду **Кодування → Розсилка повідомлень**. Після цього (рис 3.12) відображається вікно **Користувачі системи**.

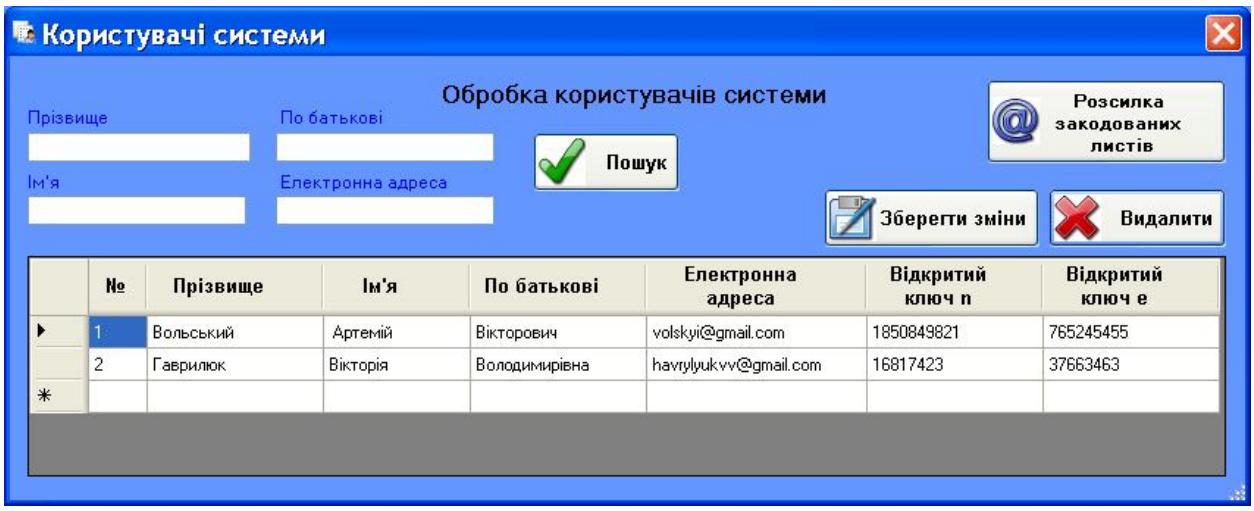

Рис. 3.12. Роботи із користувачами системи

У даному вікні є усі можливості для добавлення нових користувачів до системи, редагування користувачів та видалення потрібних користувачів. Після зроблених змін потрібно натиснути кнопку ***Зберегти зміни***. Також є можливість пошуку користувачів, якщо потрібно відібрати окремих користувачів. Для цього потрібно заповнити поля для пошуку і натиснути кнопку ***Пошук***. Якщо в параметрах пошуку нічого не вказувати, то буде відображено увесь список користувачів.

Необхідними полями для заповнення є електронна пошта користувача, відкритий ключ **n** та **e**. Після відбору користувачів є можливість відправляти їх повідомлення, при цьому необхідно натиснути кнопку ***Розсилка закодованих лисів***. Після цього (рис. 3.13) відображається вікно **Відправка листа**. На малюнку показано приклад відправки повідомлення із вкладеним ключом для кодування даних шифрами DES, ГОСТ та BlowFish. Оскільки у шифрі Blowfiish використовується ключ найбільшої довжини. Відповідний даний ключ підходить для шифру DES і ГОСТ.

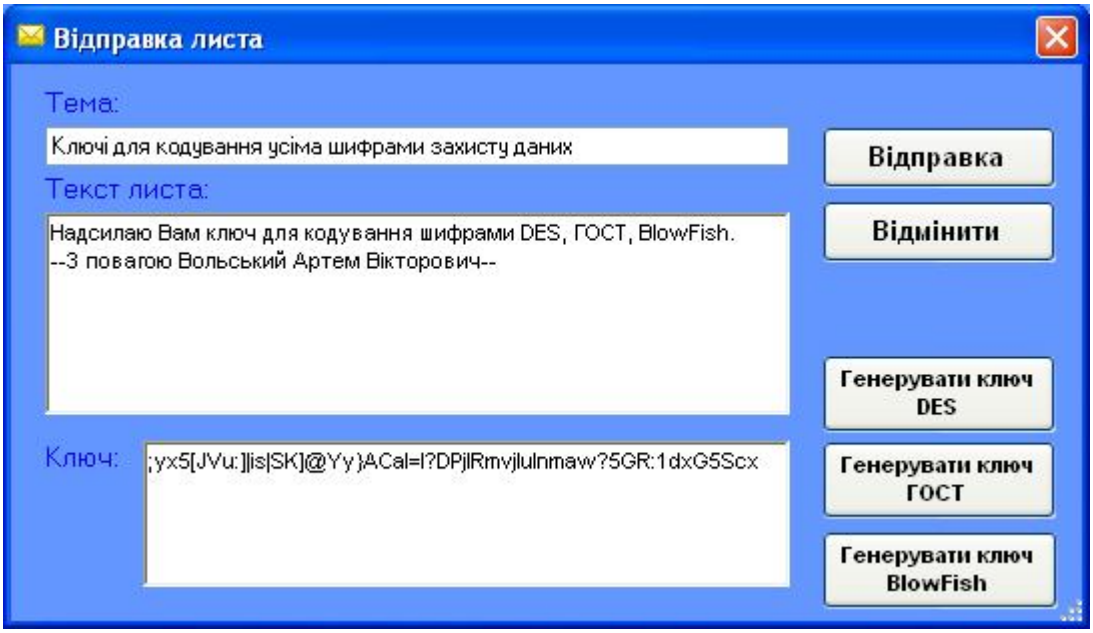

Рис. 3.13. Відправка повідомлень

Після натиску на кнопку **Відправити** лист із прикріпленим файлом ключа надсилається на електронну пошту користувачів системи. Ключ при цьому кодується відкритими ключами користувачів, що вказані для них.

Користувачі отримують лист читається його вміст і проводять загрузку файлу із ключем на ПК.

Для розшифрування ключа потрібно виконати команду **Кодування → Дешифрування ключів**. При цьому відображається вікно (рис. 3.14) **Декодування повідомлень**.

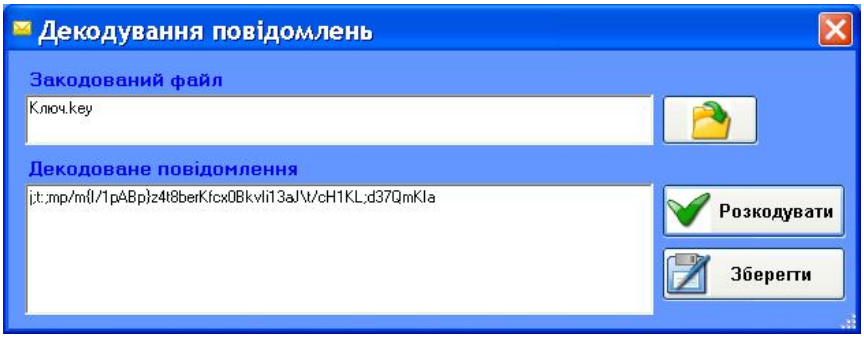

Рис. 3.14. Процес декодування повідомлень

У полі З*акодований файл* необхідно вказати файл, що потрібно розкодувати. У полі *Декодоване повідомлення* відображається результат дешифрування повідомлення. Для розшифрування повідомлення потрібно натиснути кнопку **Розкодувати**. Для збереження результатів необхідно натиснути кнопку **Зберегти**. Після цього відображається вікно (рис. 3.15) збереження результату, що є стандартним для даної операційної системи.

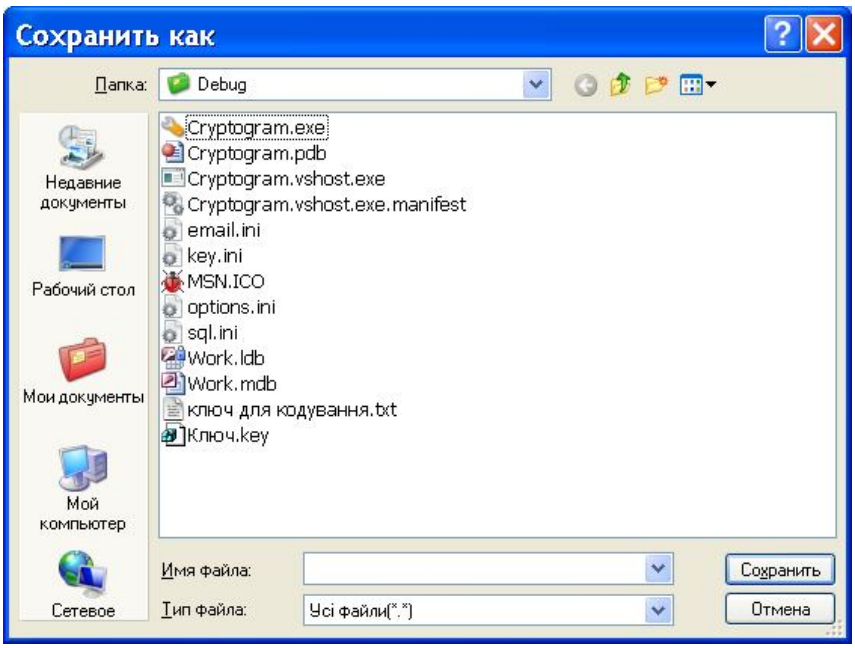

Рис. 3.15. Збереження результатів у файл

Після збереження результатів можна проводити шифрування файлів симетричними шифрами. Нами було виокремлено шифри DES, ГОСТ та BlowFish. Для шифрування інформації шифром DES потрібно виконати команду **Кодування → Des → Шифрування**. Після чого з'являється діалогове вікно (рис. 3.16) кодування інформації шифром DES.

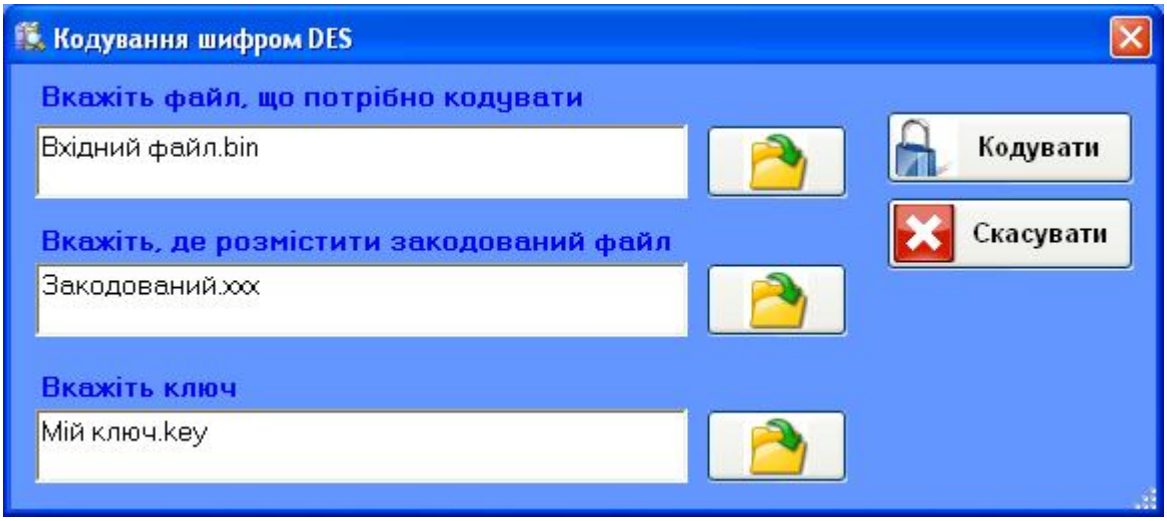

Рис. 3.16. Кодування інформації шифром DES

При вказані закодованого (рис. 3.17) файлу, можна вживати довільне ім'я і вказувати його місце розташування.

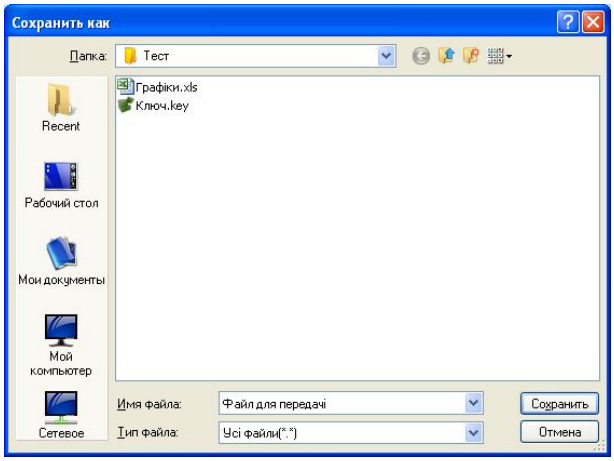

Рис. 3.17. Збереження закодованого файлу

Приклад налаштування параметрів кодування інформації шифром DES може мати вигляд наведений на рис. 3.18. Варто зазначити, що шифрування інформації можна проводити довільного типу. Розмір вхідного файлу при цьому значення не має. Швидкість кодування інформації має лінійну закономірність.

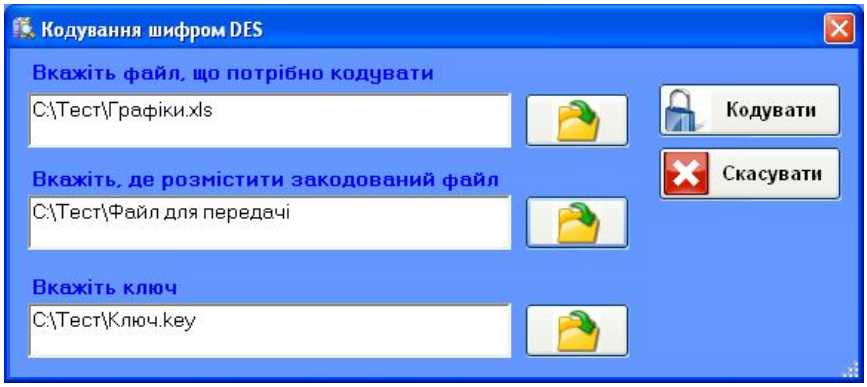

Рис. 3.18. Налаштування параметрів кодування

Після натиску на кнопку **Кодувати** виконується шифрування інформації згідно налаштувань користувача. При успішному результаті виконання операції шифрування (рис. 3.19) виводиться результат у робочу область.

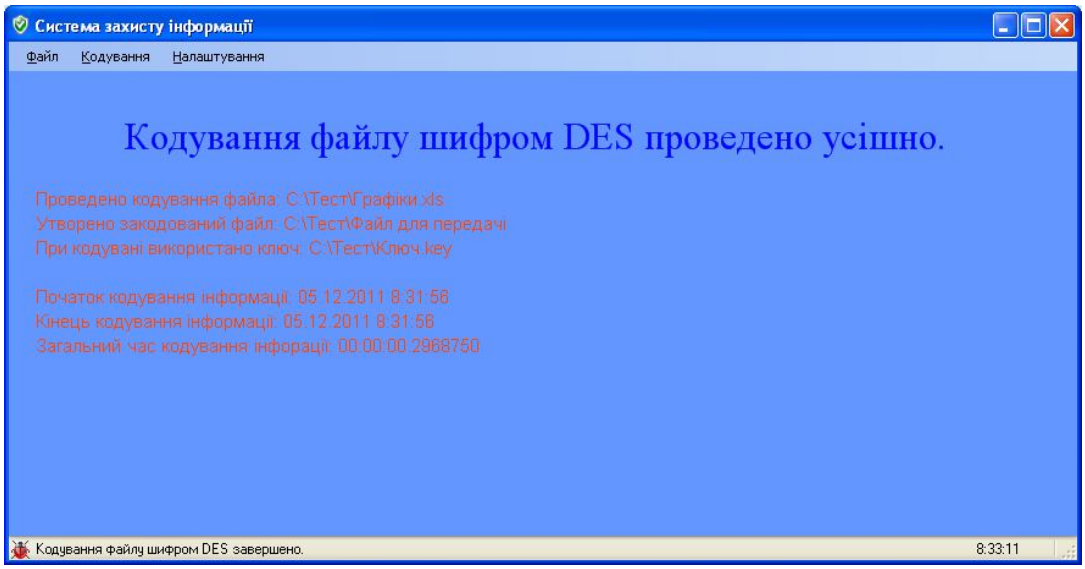

Рис. 3.19. Звіт по процесу кодування інформації шифром DES

Для виконання зворотного процесу потрібно виконати команду **Кодування → Des → Дешифрування**. Після цього відображається (рис. 3.20) діалогове вікно, де потрібно вказати параметри налаштування процесу декодування. Детально описувати процес дешифрування не будемо, оскільки параметри налаштування є схожими до процесу шифрування. Після налаштування параметрів дешифрування потрібно натиснути кнопку **Декодувати.**

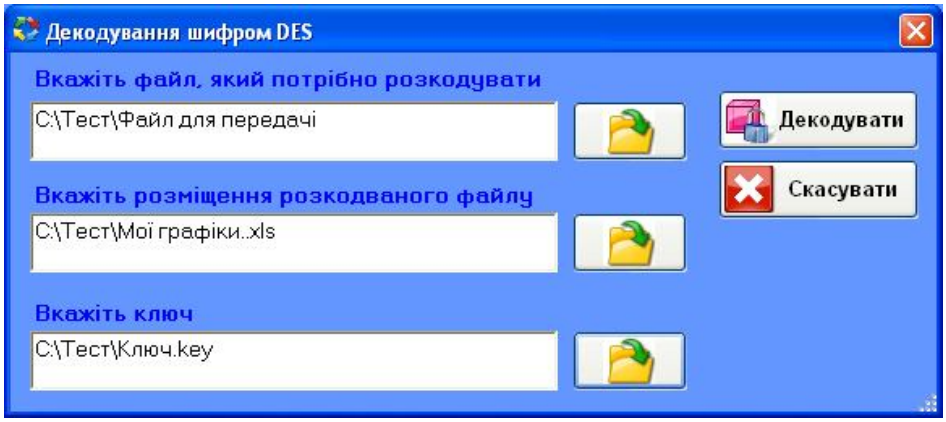

Рис. 3.20. Декодування інформації шифром DES

Після дешифрування вказаного файлу результат виводиться (рис. 3.21) у робочу область вікна.

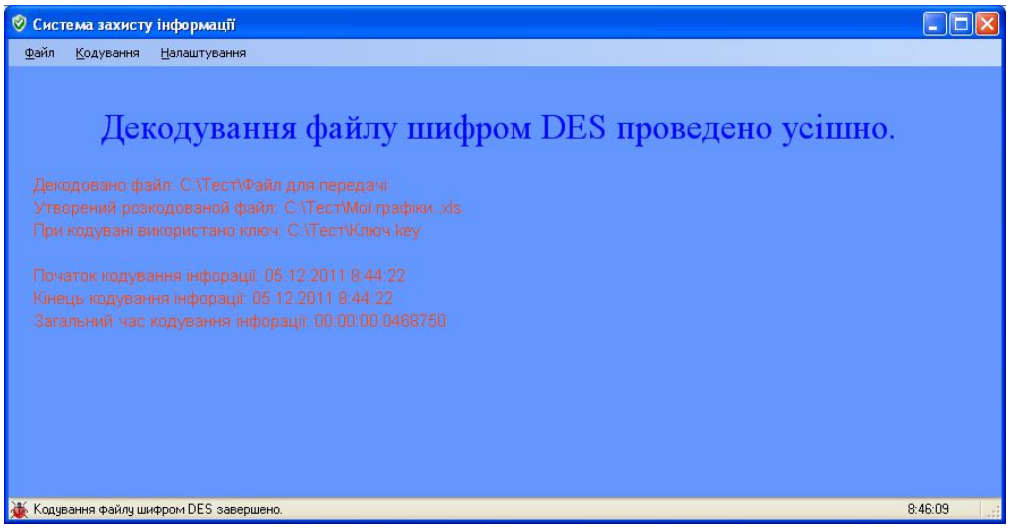

Рис. 3.21. Результат успішного декодування інформації

Для шифрування інформації шифром ГОСТ потрібно виконати команду **Кодування → ГОСТ → Шифрування**. У результаті відображається діалогове вікно **Кодування шифром ГОСТ**. Дане вікно є аналогічним рис. 3.16, тому актуалізувати увагу на шифруванні інформації шифром ГОСТ не будемо, аналогічно виконується процес дешифрування. При цьому потрібно виконати команду **Кодування → ГОСТ → Дешифрування**. Зазначимо лише, про шифруванні інформації шифром ГОСТ ключ повинен бути розміром не меншим 256 біт, а при шифруванні шифром DES не меншим 64 біт.

Для шифрування BlowFish ключ має змінну довжину, яка може досягати довжини 448 біт. Для кодування інформації шифром BlowFish

потрібно виконати команду **Кодування → BlowFish → Шифрування**. При цьому відображається (рис. 3.22) вікно **Кодування шифром BlowFish**.

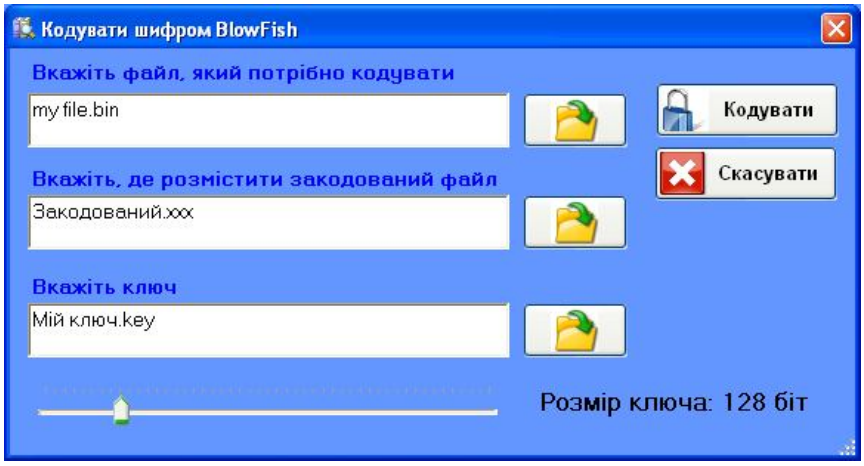

Рис. 3.22. Кодування інформації шифром BlowFish

Важливим параметром при шифруванні інформації шифром BlowFish є розмір ключа. Для зміни розміру ключа використовується повзунок у низу діалогового вікна. Розмір ключа безпосередньо впливає на надійність шифруванні інформації. Після налаштування параметрів шифрування потрібно натиснути кнопку **Кодувати**. Програма виконає шифрування вказаної інформації і результат (Рис. 3.23) буде відображено у робочій області вікна.

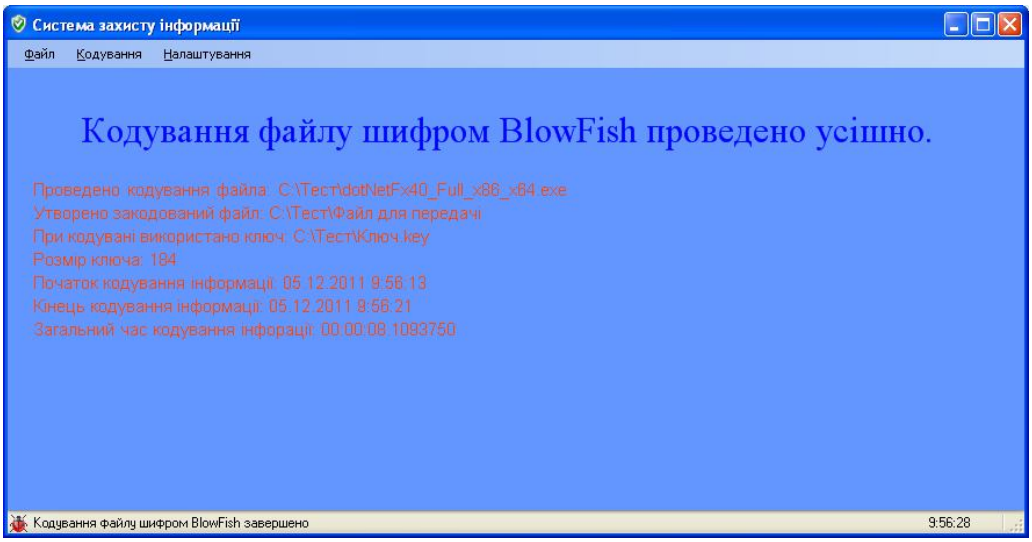

Рис. 3.23. Результат шифрування даних шифром BlowFish

У даному прикладі ми використали ключ довжиною 184 біти і провели кодування файлу розміром 48.1 Мб. Для виконання зворотного процесу

потрібно виконати команду **Кодування → BlowFish → Дешифрування**. Після цього буде відображено (рис. 3.24) вікно **Декодування шифром BlowFish**.

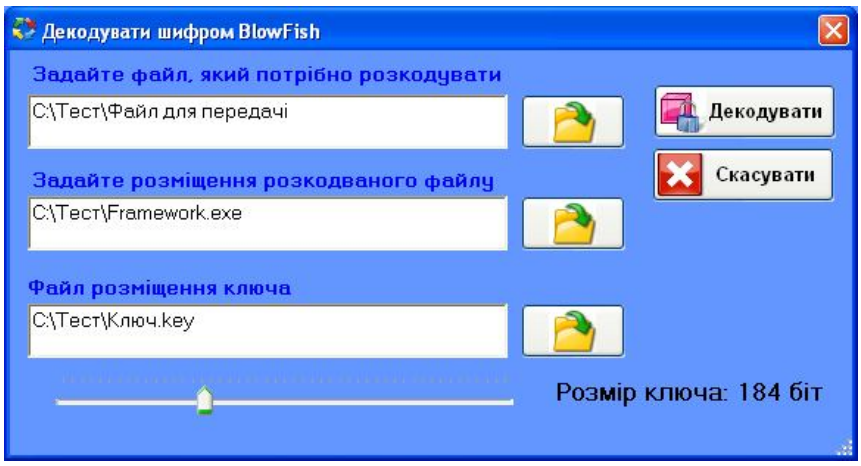

Рис. 3.24. Налаштування процесу дешифрування шифром BlowFish

Після проведення усіх необхідних налаштувань потрібно натиснути кнопку **Декодувати**. Результат буде виведено (рис. 3.25) у робочій області головного вінка.

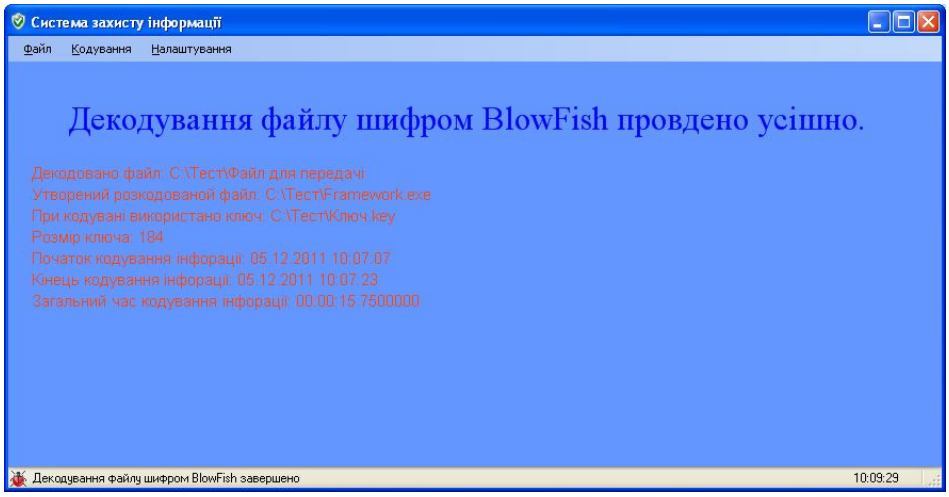

Рис. 3.25. Результат дешифрування шифром BlowFish

Таким чином нами розроблено ефективну системи захисту інформації, що представлена у вигляді додатку Windows. Програма включає в себе усі можливості для шифрування та дешифрування інформації відомими світовими стандартами захисту інформації. Програма була протестована у операційній системи Windows XP Profissional із пакетом поновлення Service Pack 3.

ВИСНОВКИ

У магістерській роботі проаналізовано та досліджено криптографічні методи захисту інформації. Криптографічні методи діляться на симетричні та асиметричні. У роботі практично реалізовано як симетричні так і асиметричні алгоритми захисту інформації. Серед симетричних алгоритмів виокремлено алгоритми блочного типу. Оскільки вони є зручними у реалізації та мають великий спектр для застосування. Симетричні шифри блочного типу характеризуються високою швидкістю роботи.

Загалом серед асиметричних алгоритмів захисту інформації основну увагу приділено шифру RSA. Даний шифр використовується у банківських системах, оскільки він дає найбільшу криптостійкість інформації, що захищається. Одними із недоліків шифру RSA є низька швидкість кодування інформації і складна апаратна реалізація.

Нами було реалізовано та простестовано наступні шифри захисту інформації:

- ✓ DES;
- ✓ ГОСТ;
- ✓ Blowfish;
- ✓ RSA.

Кожен із досліджених та реалізованих шифрів є досить надійним, що визначає його популярность у всьому світі. Проектування та реалізація даних шифрів була проведена із урахуванням їх практичного застосування у цифрових системах. Велику увагу у роботі приділено сліпому цифровому підпису. Сліпий цифровий підпис реалізовано за допомогою алгоритму RSA. Найбільш широке застосування протокол сліпих підписів знайшов у сфері цифрових грошей.

На основі розглянутих шифрів було розроблено систему захисту інформації. При виконанні операцій шифрування та дешифрування кожним із шифрів показується швидкість та ефективність їх роботи, проте із

збільшенням об’єму інформації час роботи алгоритмів відповідно збільшується. Найбільш швидко працюють шифри блочного типу.

Програмна система захисту інформації включає в себе можливості відправки електронних повідомлень, кодування довільного типу інформації симетричними шифрами, організація роботи із базою даних користувачів системи.

Система розроблена на мові програмування C#. Середовищем розробки обрано Visual Studio 2010.

Таким чином, нам вдалося організувати систему захисту інформації, що дозволяє забезпечити необхідний ступінь захищеності для користувачів, які використовують дану систему. Система ефективно впроваджена у використання на базі УКРГАЗ Банку. Головною вимогою до розробленої системи є правильність її налаштування для досягнення використання усіх її можливостей.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Алекс Макки Введение в .NET 4.0 и Visual Studio 2010 для профессионалов. – М.: "Вильямс", 2010. – 416 с.

2. Алферов А.П., Зубов А.Ю., Кузьмин А.С., Черемушкин А.В. Основы криптографии. – М.: “Гелиос АРВ”, 2001. – 480 с.

3. Асосков А.В., Иванов М.А., Мирский А.А., Рузин А.В., Славин А.В., Тютвин А.Н. Поточные шифры.- М.: КУДИЦ – ОБРАЗ, 2003. – 336 с.

4. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. - М.: Мир, 1979. – 536с.

5. Бабаш А.В., Шанкин Г.П. Криптография. – М.: Соломон – Р, 2002. – 512 с.

6. Батурин Ю.М., Жодзишский А.М. Компьютерная преступность и компьютерная безопасность. – М.: Юридическая литература, 1991. – 158 с.

7. Биркгоф Г., Барти Т. Современная прикладная алгебра. – М. Мир, 1976.

8. Брикэлл Э.Ф., Одлижко Э.М. Криптоанализ: Обзор новейших результатов. // ТИИЭР. – 1988. – Т.76, №5. – С. 75-94.

9. Бичков О., Турбал Ю. Основи сучасного програмування Навч. посібник для студ. техніч. спец. вищ. навч. закладів М-во освіти і науки України. КНУ ім. Т.Шевченка. МУ "РЕГІ" ім. акад. С. Дем'янчука. – Рівне: РВЦ "ТЕТІС", 2004. – 448с.

10. Вербіцький О.В. Вступ до криптології. – Львів: Науково-техн. літ., 1998. – 248с.

11. Гайкович В., Першин А. Безопасность электронных банковских систем. – М.: Единая Европа, 1994. – 564с.

12. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2001. — 368 с.

13. Герасименко В.А. Защита информации в автоматизированных системах обработки данных. В 2-х кн. – М.: Энергоатомиздат, 1994. – Кн.1 – 401с., кн.2 – 176 с.

14. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. – М.: Мир, 1982. – 416с.

15. Деднев М.А., Дыльнов Д.В., Иванов М.А. Защита информации в банковском деле и электронном бизнесе. – М.: КУДИЦ-ОБРАЗ, 2004. – 512с.

16. Диффи У., Хеллман М. Защищенность и имитостойкость. // ТИИЭР. – 1979. – Т.67, №3. – с. 71-109.

17. Жарков В.А. Visual C#.NET в науке и технике. — М.: Жарков Пресс, 2002.

18. Жарков В.А. Самоучитель Жаркова по Visual Studio .Net: Visul Basic .NET, Visual C# .NET, Visual C++ .NET, Visual J# .NET. — М.: Жарков Пресс, 2002.

19. Жарков В.А. Visual C++ на практике. — М.: Лаборатория базовых знаний, 2002.

20. Завадская Л.А., Фаль А.М. Криптографически сильные генераторы псевдослучайных последовательностей. // Безопасность информации. – 1997. - №1. – С. 7-11.

21. Задірака В., Олексик О. Комп'ютерна криптологія. – Київ, 2002. – 505 с.

22. Задірака В.К. Олексюк О.С. Методи захисту фінансової інформації. – К.: Вища школа, 2000. – 460 с.

23. Зегжда Д.П., Ивашко А.М. Как построить защищенную информационную систему. – СПб.: НПО "Мир и Семья-95", ООО "Интерлайн", 1998. – 256с.

24. Кейт Грегори Использование Microsoft Visual C++.NET. – Киев, 2002. – 780 с.

25. Климов Л. П. C#. Советы программистам. – СПб.: БХВ-Петербург, 2008. – 544 с.

26. Кнут Д. Искусство программирования на ЭВМ. Т2. – М.: Мир, 1976. – 724с.

27. Колесник В.Д., Полтырев Г.Ш. Курс теории информации. – М.: Наука, 1982. – 416с.

28. Кудин А.М. Методы тестирования чисел на простоту и построение простых чисел. // Безопасность информации. – 1996. - №3. – С.23-32.

29. Кристиан Нейгел, Билл Ивьен, Джей Глинн, Карли Уотсон, Морган Скиннер Visual C# 2008. Базовый курс. – М.: "Вильямс", 2009. – 1210 с.

30. Лабор В.В. Си Шарп. Создание приложений для Windows — Мн.: Харвест, 2003. — 384 с.

31. Мафтик С. Механизмы защиты в сетях ЭВМ.–М.: Мир, 1993. – 216 с.

32. Месси Дж.Л. Введение в современную криптологию. // ТИИЭР. – 1988. – Т.76, №5. – С.24-42.

33. Нечаев В.И. Элементы криптографии (основы защиты информации). – Москва: Высшая школа, 1999. – 109с.

34. Нікольський Ю.В., Пасічник В.В., Щебрина Ю.М. Дискретна математика Підручник для студентів вищ. навч. закладів За ред.: акад. НАН України М.З.Згуровського . – К.:Видавнича група BHV, 2007. – 368с.

35. Олифер В.Г., Олифер Н.А. Компьютерные сети: принципы, технологии, протоколы Учеб. пособие для студентов высш. учеб. заведений .- 2-е изд. – М.:ПИТЕР, 2005. – 864с.

36. Павловская Т.А., Щупак Ю.А. С++. Объективно-ориентированное программирование: Практикум.. Навч. посібник для студ. изучающих С++ . – СПБ.: Питер, 2005 . – М.:Питер,2005 . – 264с.

37. Патрис Пелланд, Паскаль Паре, Кен Хайнс Переход к Microsoft Visual Studio 2010. – Корпорация Microsoft, 2011. – 256 с.

38. Петров А.А. Компьютерная безопасность. Криптографические методы защиты. – М.: ДМК, 2000. – 448 с.

39. Петраков А.В. Основы практической защиты информации Учеб. пособие по спец. "Защищенные системы связи". – 4-е изд., доп. – М.: СОЛОН-Пресс, 2005 . – 384 с.

40. Прата С. Язык программирования С++. Лекции и упражнения The Waite Group's C++ Primer Plus, Third Edition . – К.: ДиаСофт, 2001 . – 656с.

41. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 2.0 на языке C#. Мастер-класс. / Пер. с англ. – М.: Издательство «Русская Редакция»; СПб.: Питер, 2007. – 656 с.

42. Саломаа А. Криптография с открытым ключом. – М.: Мир, 1996. – 318 с.

43. Себеста, Роберт, Основные концепции языков программирования . – 5-е изд. – М.: "Вильямс", 2001. – 672 с.

44. Симмонс Г.Дж. Обзор методов аутентификации информации // ТИИЭР – 1988. – Т.76.№5 – с.105-125.

45. Смарт Н. Криптография. – М.: Техносфера, 2005. – 528 с.

46. Троелсен Эндрю Язык программирования C# 2010 и платформа .NET 4.0, 5-е издательство. – М.: "Вильямс", 2011. – 1392 с.

47. Троэлсен Э. C# и платформа .NET. Библиотека программиста. – СПб.: Питер, 2004. – 796 с.

48. Федоров А. Microsoft Visual Studio 2010: первое знакомство. – Microsoft, 2009. – 42 с.

49. Хорошко В.А., Чекатков А.А. Методы и средства защиты информации Под ред. Ю.С.Ковтанюка . – К.:ЮНИОР, 2003. – 504 с.

50. Хоффман Л.Дж. Современные методы защиты информации. – М.: Сов. радио, 1980. – 264с.

51. Цирлер Н. Линейные возвратные последовательности. // В кн.: Кибернетический сборник. Вып.6 – М.: ИЛ, 1963. – С.55-79.

52. Чмора А. Современная прикладная криптография. – М.: “Гелиос АРВ”, 2001. – 256 с.

53. Шаханова, М.В. Современные технологии информационной безопасности: учеб. пособие / М.В .Шаханова; Дальневосточный государственный технический университет. - Владивосток: Изд-во ДВГТУ, 2007. - 217 с.

54. Шеннон К.Э. Теория связи в секретных системах. // В кн.: Шеннон К.Э. Работы по теории информации и кибернетике. – М.: ИЛ, 1963. – с. 243-332

55. Шилд Г. C# Учебный курс. — М.: Питер 2003. — 471 с.

56. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке СИ. – М.: ТРИУМФ, 2003. – 816с.

57. Яблонский С.В. Введение в дискретную математику. – М.: Наука, 1979. – 272 с.

58. Ященко В.В. Введение в криптографию.-М.: МЦНМО: ЧеРо, 1999.– 272 с.

59. Needham R M, Schroeder M D Using encryption for authentication in large networks of computers // Communications of the ACM. – 1978. – Vol. 21. – P. 993 – 999.

60. Schneier Bruce. Applied cryptography (Second edition). Protocols, Algorithms and suorce Code in C.- New York, Toronto: John Wiley & Sons, 1996.

61. Stinson D.R. Criptography: theory and practice. – Boca Ruton, New York, London, Tokyo: CRC Press, 1995 – 434 p.

62. http://secureinfo.ru/category/education/ – Безопасные технологи. Все, что связанно с компьютерной безопасностью.

ДОДАТКИ

ДОДАТОК 1. РЕАЛІЗАЦІЯ ПОДІЙ ТА МЕТОДІВ ГОЛОВНОЇ ФОРМИ ВІКНА

```
 using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;


namespace Cryptogram
{
    public partial class MainForm : Form
    {
        string myFileCode;     //Файл який потрібно кодувати
        string myFileDecode;   //Файл в який потрібно декодувати
        string myKey;          //Файл розмідення ключа
        int mySizeKey;      //розмфр ключа для кодування
        int action;           //Яка операція була виконана
        DateTime timeBegin;   //Час початку кодування інфомрації
        DateTime timeEnd;     //Час закінчення кодування
        private const int CS_DROPSHADOW = 0x00020000;
        //Создаем обьекты для всей строки состояния и ее отдельных панелей
        private StatusBar statusBar = new StatusBar();
        private StatusBarPanel sbPn1Prompt = new StatusBarPanel();
        private StatusBarPanel sbPn1Time = new StatusBarPanel();
        //Показує поточний час на панелі
        private Timer timer1 = new Timer();
        public MainForm()
        {
            //Настраиваем обьект Timer
            timer1.Interval = 1000;
            timer1.Enabled = true;
            timer1.Tick += new EventHandler(timer1_Tick);
            InitializeComponent();
            BuildStatBar();  //метод который создает нам  строку состояния и настраивает
ее нужным образом
            CenterToScreen();
            action = 0;

        }
// Переопределяем свойства CreateParams  Показую тінь вікна
        protected override CreateParams CreateParams
        {
            get
            {
                CreateParams cp = base.CreateParams;
                cp.ClassStyle |= CS_DROPSHADOW;
                return cp;
            }
        }
        //Вспомагательная функция показує строку стану
        private void BuildStatBar()
        {
```

```
            //Настраиваем строку состояния
            statusBar.ShowPanels = true;
            statusBar.Size = new System.Drawing.Size(312, 20);
            statusBar.Location = new System.Drawing.Point(0, 216);

            //Панель добавляется в строку состояния при помощи метода AddRange()
            statusBar.Panels.AddRange(new StatusBarPanel[] { sbPn1Prompt, sbPn1Time });
            //Настраиваем левую панель
            sbPn1Prompt.BorderStyle = StatusBarPanelBorderStyle.None;
            sbPn1Prompt.AutoSize = StatusBarPanelAutoSize.Spring;
            sbPn1Prompt.Width = 62;
            sbPn1Prompt.Text = "Система завантажена";

            //Настраиваем правую панель
            sbPn1Time.Alignment = HorizontalAlignment.Left;
            sbPn1Time.Width = 70;
            //Добавляем значок
            try
            {
                //Значок должен находится в поточном каталоге
                Icon i = new Icon("MSN.ICO");
                sbPn1Prompt.Icon = i;
            }
            catch (Exception e)
            {
                MessageBox.Show(e.Message);
            }
            //Теперь добавляем панель управления в колекцию Controls для формы
            this.Controls.Add(statusBar);
        }
        //Этот метод будет вызыватся примерно каждую секунду
        private void timer1_Tick(object sender, EventArgs e)
        {
            DateTime t = DateTime.Now;
            string s = t.ToLongTimeString();
            //Выводим полученоке строковое значение на правую панель
            sbPn1Time.Text = s;
        }

        private void menFile_Exit_Click(object sender, EventArgs e)
        {
            this.Close();
        }
        //Підключення до бази даних
        private void menOptionsBaseConection_Click(object sender, EventArgs e)
        {
            BaseConForm dlg=new BaseConForm();
            if (dlg.ShowDialog() == DialogResult.Abort)
                return;
        }
        //Попередження при виході із програми
        private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (MessageBox.Show("Ви дійсно бажаєте вийти із програми?", "Попередження!",
MessageBoxButtons.YesNo) == DialogResult.No)
                e.Cancel = true;
        }
        //Розсилка ключів користувачам програми
        private void menCodeSendKey_Click(object sender, EventArgs e)
        {
            ListUserBaseForm dlg = new ListUserBaseForm();
            if (dlg.ShowDialog() == DialogResult.Abort)
                return;
        }
```

```
        //Підключення до електронного ящика
        private void menOptionMailConnec_Click(object sender, EventArgs e)
        {
            MailConnectForm dlg = new MailConnectForm();
            if (dlg.ShowDialog() == DialogResult.OK)
                return;
        }
        //генерування ключів
        private void generateKeyToolStripMenuItem_Click(object sender, EventArgs e)
        {
            GenarateKeyForm dlg = new GenarateKeyForm();
            if (dlg.ShowDialog() == DialogResult.OK)
                return;
        }

        private void mnuCodeDecodeTextMail_Click(object sender, EventArgs e)
        {
            DecodeTextMailForm dlg = new DecodeTextMailForm();
            if (dlg.ShowDialog() == DialogResult.Abort)
            {
                return;
            }
        }
        //Кодування шифром DES
        private void menCodeDesCode_Click(object sender, EventArgs e)
        {
            bool is_good = true;
            //Тот же диалог для шифра DES
            Gost_Code dlgCode = new Gost_Code();
            dlgCode.FileCode = "Вхідний файл.bin";
            dlgCode.FileDecode = "Закодований.xxx";
            dlgCode.Key = "Мій ключ.key";
            dlgCode.Text = "Кодування шифром DES";
            if (dlgCode.ShowDialog(this) == DialogResult.OK)
            {
                timeBegin = DateTime.Now;
                myFileCode = dlgCode.FileCode;
                myFileDecode = dlgCode.FileDecode;
                myKey = dlgCode.Key;
                int temp = 0;
                int leng = 8;
                byte[] key = new byte[leng];

                FileStream readKey = null;
                BinaryReader binKey = null;
                try
                {
                    readKey = new FileStream(myKey, FileMode.Open, FileAccess.Read);
                    binKey = new BinaryReader(readKey);
                    for (temp = 0; temp < leng; temp++)
                    {
                        key[temp] = binKey.ReadByte();
                    }
                    binKey.Close();
                }
                catch (IOException errorKey)
                {
                    MessageBox.Show("Помилка при відкриті файлу, який містить ключ!!!\n" +
errorKey.Message, "Помикла!");
                    is_good = false;
                }
                finally
                {
                    if (readKey != null)
```

```
                    readKey.Close();
                }
                if (temp != leng && is_good)
                {
                    MessageBox.Show("Розмір файлу, який містить ключ є надто малим!!!",
"Помилка!");
                    is_good = false;
                }
                FileStream stream = null;          //Файл, який потрібно читати
                FileStream streamWrite = null;     //Файл в який потрібно записувати
                BinaryReader reader = null;
                BinaryWriter writer = null;
                if (is_good)
                {
                    try
                    {
                        stream = new FileStream(myFileCode, FileMode.Open,
FileAccess.Read);
                        streamWrite = new FileStream(myFileDecode, FileMode.Create,
FileAccess.Write);
                        reader = new BinaryReader(stream);
                        writer = new BinaryWriter(streamWrite);
                        byte[] data = new byte[8];
                        DES my_des = new DES();
                        my_des.des_key(ref key);
                        long processed = 0;
                        long fileSize = stream.Length;
                        long lushok = fileSize % 8;
                        fileSize -= lushok;
                        int i = 0;
                        while (processed < fileSize)
                        {
                            for (i = 0; i < 8; i++)
                                data[i] = reader.ReadByte();
                            my_des.des_enc(ref data);
                            for (i = 0; i < 8; i++)
                                writer.Write(data[i]);
                            processed += 8;    //Переміщаємося на 8 біт
                        }
                        for (i = 0; i < lushok; i++)
                            writer.Write(reader.ReadByte());
                    }
                    catch (IOException expt)
                    {
                        MessageBox.Show("Помилка при відкриті файлу, який потрібно
кодувати!!!\n" + expt.Message, "Помикла!");
                        is_good = false;
                    }
                    finally
                    {
                        if (stream != null)
                        {
                            stream.Close();
                        }
                        if (streamWrite != null)
                            streamWrite.Close();
                        if (is_good)
                        {
                            //Фуксуємо, що кодування проведено шифром DES
                            action = 1;           //Кодування файлу проведенно успішно
                        }
                        timeEnd = DateTime.Now;
                        this.Invalidate();
                    }
```

```
                }
            }
        }
        //Відображення інформації у вікні
        private void MainForm_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            //размер шрифта будет между 12 та 62 в зависимоти от swellValue
            Font theFont = new Font("Times New Roman", 23);
            switch (action)
            {
                case 1:   //Кодування шифром DES
                    {
                        string messageCode = "Кодування файлу шифром DES проведено
усішно.";
                        //Вивід по центру форми
                        float windowsCenter = this.DisplayRectangle.Width / 2;
                        SizeF stringSize = g.MeasureString(messageCode, theFont);
                        float startPos = windowsCenter - (stringSize.Width / 2);
                        g.DrawString(messageCode, theFont,
                            new SolidBrush(Color.Blue), startPos, 60);
                        g.DrawString("Проведено кодування файла: " +
myFileCode.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
120);
                        g.DrawString("Утворено закодований файл: " +
myFileDecode.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
140);
                        g.DrawString("При кодувані використано ключ: " + myKey.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
160);
                        g.DrawString("Початок кодування інформації: " +
timeBegin.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
200);
                        g.DrawString("Кінець кодування інформації: " + timeEnd.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
220);
                        g.DrawString("Загальний час кодування інфорації: " + (timeEnd-
timeBegin).ToString() + "",
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
240);
                        sbPn1Prompt.Text = "Кодування файлу шифром DES завершено.";
                        break;
                    }
                case 2: //Декодування шифром DES
                    {
                        string messageCode = "Декодування файлу шифром DES проведено
усішно.";
                        //Вивід по центру форми
                        float windowsCenter = this.DisplayRectangle.Width / 2;
                        SizeF stringSize = g.MeasureString(messageCode, theFont);
                        float startPos = windowsCenter - (stringSize.Width / 2);
                        g.DrawString(messageCode, theFont,
                            new SolidBrush(Color.Blue), startPos, 60);
                        g.DrawString("Декодовано файл: " + myFileDecode.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
120);
                        g.DrawString("Утворений розкодованой файл: " +
myFileCode.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
140);
                        g.DrawString("При кодувані використано ключ: " + myKey.ToString(),
```

```
                            new Font("Arial", 11), new
SolidBrush(Color.OrangeRed), 20, 160);
                        g.DrawString("Початок кодування інфорації: " +
timeBegin.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
200);
                        g.DrawString("Кінець кодування інфорації: " + timeEnd.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
220);
                        g.DrawString("Загальний час кодування інфорації: " + (timeEnd -
timeBegin).ToString() + "",
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
240);
                        sbPn1Prompt.Text = "Кодування файлу шифром DES завершено.";
                        break;
                    }
                case 3:  //Кодування шифром ГОСТ
                    {
                        string messageCode = "Кодування файлу шифром ГОСТ проведено
усішно.";
                        //Вивід по центру форми
                        float windowsCenter = this.DisplayRectangle.Width / 2;
                        SizeF stringSize = g.MeasureString(messageCode, theFont);
                        float startPos = windowsCenter - (stringSize.Width / 2);
                        g.DrawString(messageCode, theFont,
                            new SolidBrush(Color.Blue), startPos, 60);
                        g.DrawString("Проведено кодування файла: " +
myFileCode.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
120);
                        g.DrawString("Утворено закодований файл: " +
myFileDecode.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
140);
                        g.DrawString("При кодувані використано ключ: " + myKey.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
160);
                        g.DrawString("Початок кодування інформації: " +
timeBegin.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
200);
                        g.DrawString("Кінець кодування інформації: " + timeEnd.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
220);
                        g.DrawString("Загальний час кодування інфорації: " + (timeEnd -
timeBegin).ToString() + "",
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
240);
                        sbPn1Prompt.Text = "Кодування файлу шифром ГОСТ завершено.";
                        break;
                    }
                case 4:  //Декодування шифром ГОСТ
                    {
                        string messageCode = "Декодування файлу шифром ГОСТ проведено
усішно.";
                        //Вивід по центру форми
                        float windowsCenter = this.DisplayRectangle.Width / 2;
                        SizeF stringSize = g.MeasureString(messageCode, theFont);
                        float startPos = windowsCenter - (stringSize.Width / 2);
                        g.DrawString(messageCode, theFont,
                            new SolidBrush(Color.Blue), startPos, 60);
                        g.DrawString("Декодовано файл: " + myFileDecode.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
120);
```

```
                        g.DrawString("Утворений розкодованой файл: " +
myFileCode.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
140);
                        g.DrawString("При кодувані використано ключ: " + myKey.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
160);
                        g.DrawString("Початок кодування інфорації: " +
timeBegin.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
200);
                        g.DrawString("Кінець кодування інфорації: " + timeEnd.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
220);
                        g.DrawString("Загальний час кодування інфорації: " + (timeEnd -
timeBegin).ToString() + "",
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
240);
                        sbPn1Prompt.Text = "Кодування файлу шифром ГОСТ завершено.";
                        break;
                    }
                case 5:
                    {
                        string messageCode = "Кодування файлу шифром BlowFish проведено
усішно.";
                        //Вивід по центру форми
                        float windowsCenter = this.DisplayRectangle.Width / 2;
                        SizeF stringSize = g.MeasureString(messageCode, theFont);
                        float startPos = windowsCenter - (stringSize.Width / 2);
                        g.DrawString(messageCode, theFont,
                            new SolidBrush(Color.Blue), startPos, 60);
                        g.DrawString("Проведено кодування файла: " +
myFileCode.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
120);
                        g.DrawString("Утворено закодований файл: " +
myFileDecode.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
140);
                        g.DrawString("При кодувані використано ключ: " + myKey.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
160);
                        g.DrawString("Розмір ключа: " + mySizeKey.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
180);
                        g.DrawString("Початок кодування інформації: " +
timeBegin.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
200);
                        g.DrawString("Кінець кодування інформації: " + timeEnd.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
220);
                        g.DrawString("Загальний час кодування інфорації: " + (timeEnd -
timeBegin).ToString() + "",
                             new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
240);
                        sbPn1Prompt.Text = "Кодування файлу шифром BlowFish завершено";
                        break;
                    }
                case 6:
                    {
                        string messageCode = "Декодування файлу шифром BlowFish провдено
усішно.";
                        //Вивід по центру форми
```

```
                        float windowsCenter = this.DisplayRectangle.Width / 2;
                        SizeF stringSize = g.MeasureString(messageCode, theFont);
                        float startPos = windowsCenter - (stringSize.Width / 2);
                        g.DrawString(messageCode, theFont,
                            new SolidBrush(Color.Blue), startPos, 60);
                        g.DrawString("Декодовано файл: " + myFileDecode.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
120);
                        g.DrawString("Утворений розкодованой файл: " +
myFileCode.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
140);
                        g.DrawString("При кодувані використано ключ: " + myKey.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
160);
                        g.DrawString("Розмір ключа: " + mySizeKey.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
180);
                        g.DrawString("Початок кодування інфорації: " +
timeBegin.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
200);
                        g.DrawString("Кінець кодування інфорації: " + timeEnd.ToString(),
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
220);
                        g.DrawString("Загальний час кодування інфорації: " + (timeEnd -
timeBegin).ToString() + "",
                            new Font("Arial", 11), new SolidBrush(Color.OrangeRed), 20,
240);
                        sbPn1Prompt.Text = "Декодування файлу шифром BlowFish завершено";
                        break;
                    }
                default:
                    {
                        sbPn1Prompt.Text = "Система завантажена";
                        string z = "      Аналіз і дослідження \n";
                        z+="  крипрографічних методів \n";
                        z+="         захисту інформації\n";
                        z += "      на базі УКРГАЗ Банку\n";
                        //Вивід по центру форми
                        float windowsCenter = this.DisplayRectangle.Width / 2;
                        SizeF stringSize = g.MeasureString(z, theFont);
                        float startPos = windowsCenter - (stringSize.Width / 2);
                        g.DrawString(z, theFont, new SolidBrush(Color.Blue), startPos,
40);
                        g.DrawString("Розробник програмної системи:", new Font("Times New
Roman", 14), new SolidBrush(Color.Black), 500, 250);
                        g.DrawString("студентка групи  ІН-01М", new Font("Times New
Roman", 14,FontStyle.Bold), new SolidBrush(Color.Green), 545, 275);
                        g.DrawString("Українець Ірина", new Font("Arial", 16,
FontStyle.Italic|FontStyle.Bold), new SolidBrush(Color.Blue), 540, 300);
                        g.DrawString("Василівна", new Font("Arial", 16, FontStyle.Italic |
FontStyle.Bold), new SolidBrush(Color.Blue), 540, 325);
                        g.DrawString("2011 рік", new Font("Times New Roman", 16,
FontStyle.Bold), new SolidBrush(Color.Red), 650, 420);
                        break;
                    }
            }
        }
        //Декодування шифром DES
        private void menCodeDesDecode_Click(object sender, EventArgs e)
        {
            bool is_good = true;
            DES_Gost_DecodeForm dlgDeCode = new DES_Gost_DecodeForm();
```

```
            dlgDeCode.Text = "Декодування шифром DES";
            dlgDeCode.FileDecode = "Закодований.xxx";
            dlgDeCode.FileCode = "Розкодований.my";
            dlgDeCode.Key = "Мій ключ.key";
            if (dlgDeCode.ShowDialog(this) == DialogResult.OK)
            {
                sbPn1Prompt.Text = "Розпочато декодування файлу шифром DES";
                timeBegin = DateTime.Now;         //Now Time
                myFileDecode = dlgDeCode.FileDecode;
                myFileCode = dlgDeCode.FileCode;
                myKey = dlgDeCode.Key;
                int temp = 0;
                int leng = 8;
                byte[] key = new byte[leng];
                FileStream readKey = null;
                BinaryReader binKey = null;
                try
                {
                    readKey = new FileStream(myKey, FileMode.Open, FileAccess.Read);
                    binKey = new BinaryReader(readKey);
                    for (temp = 0; temp < leng; temp++)
                    {
                        key[temp] = binKey.ReadByte();
                    }
                    binKey.Close();
                }
                catch (IOException errorKey)
                {
                    MessageBox.Show("Помилка при відкриті файлу, який містить ключ!!!\n" +
errorKey.Message, "Помикла!");
                    is_good = false;
                }
                finally
                {
                    if (readKey != null)
                        readKey.Close();
                }
                if (temp != leng && is_good)
                {
                    MessageBox.Show("Розмір файлу, який містить ключ є надто малим!!!",
"Помилка!");
                    is_good = false;
                }


                FileStream stream = null;         //Файл, який потрібно читати
                FileStream streamWrite = null;    //Файл в який потрібно записувати
                BinaryReader reader = null;
                BinaryWriter writer = null;
                if (is_good)
                {
                    try
                    {
                        stream = new FileStream(myFileDecode, FileMode.Open,
FileAccess.Read);
                        streamWrite = new FileStream(myFileCode, FileMode.Create,
FileAccess.Write);
                        reader = new BinaryReader(stream);
                        writer = new BinaryWriter(streamWrite);
                        byte[] data = new byte[8];
                        DES my_des = new DES();
                        my_des.des_key(ref key);
                        long processed = 0;
                        long fileSize = stream.Length;
```

```
                        long lushok = fileSize % 8;
                        fileSize -= lushok;
                        int i = 0;
                        while (processed < fileSize)
                        {
                            for (i = 0; i < 8; i++)
                                data[i] = reader.ReadByte();
                            my_des.des_dec(ref data);
                            for (i = 0; i < 8; i++)
                                writer.Write(data[i]);
                            processed += 8;    //Переміщаємося на 8 біт
                        }
                        for (i = 0; i < lushok; i++)
                            writer.Write(reader.ReadByte());
                    }
                    catch (IOException expt)
                    {
                        MessageBox.Show("Помилка при відкриті файлу, який потрібно
кодувати!!!\n" + expt.Message, "Помикла!");
                        is_good = false;
                    }
                    finally
                    {
                        if (stream != null)
                        {
                            stream.Close();
                        }
                        if (streamWrite != null)
                            streamWrite.Close();
                        if (is_good)
                        {
                            action = 2;           //Декодування шифром DES
                            timeEnd = DateTime.Now;
                        }
                        this.Invalidate();
                    }
                }
            }
        }
        //кодування шифром ГОСТ
        private void menCodeGostCode_Click(object sender, EventArgs e)
        {
            bool is_good = true;
            //Тот же диалог для шифра DES
            Gost_Code dlgCode = new Gost_Code();
            dlgCode.FileCode = "Вхідний файл.bin";
            dlgCode.FileDecode = "Закодований.xxx";
            dlgCode.Key = "Мій ключ.key";
            dlgCode.Text = "Кодування шифром ГОСТ";
            if (dlgCode.ShowDialog(this) == DialogResult.OK)
            {
                timeBegin = DateTime.Now;
                myFileCode = dlgCode.FileCode;
                myFileDecode = dlgCode.FileDecode;
                myKey = dlgCode.Key;
                int temp = 0;
                int leng = 8;
                uint[] key = new uint[leng];
                FileStream readKey = null;
                BinaryReader binKey = null;
                try
                {
                    readKey = new FileStream(myKey, FileMode.Open, FileAccess.Read);
                    binKey = new BinaryReader(readKey);
```

```
                    for (temp = 0; temp < leng; temp++)
                    {
                        key[temp] = binKey.ReadUInt32();
                    }
                    binKey.Close();
                }
                catch (IOException errorKey)
                {
                    MessageBox.Show("Помилка при відкриті файлу, який містить ключ!!!\n" +
errorKey.Message, "Помикла!");
                    is_good = false;
                }
                finally
                {
                    if (readKey != null)
                        readKey.Close();
                }
                if (temp != leng && is_good)
                {
                    MessageBox.Show("Розмір файлу, який містить ключ є надто малим!!!",
"Помилка!");
                    is_good = false;
                }
                FileStream stream = null;          //Файл, який потрібно читати
                FileStream streamWrite = null;     //Файл в який потрібно записувати
                BinaryReader reader = null;
                BinaryWriter writer = null;
                if (is_good)
                {
                    try
                    {
                        stream = new FileStream(myFileCode, FileMode.Open,
FileAccess.Read);
                        streamWrite = new FileStream(myFileDecode, FileMode.Create,
FileAccess.Write);
                        reader = new BinaryReader(stream);
                        writer = new BinaryWriter(streamWrite);
                        uint[] data = new uint[2];
                        GOST my_gost = new GOST();
                        my_gost.gost_key(key);
                        long processed = 0;
                        long fileSize = stream.Length;
                        long lushok = fileSize % 8;
                        fileSize -= lushok;
                        while (processed < fileSize)
                        {
                            data[0] = reader.ReadUInt32();
                            data[1] = reader.ReadUInt32();
                            my_gost.gost_enc(ref data, 1);
                            writer.Write(data[0]);
                            writer.Write(data[1]);
                            processed += 8;    //Переміщаємося на 8 біт
                        }
                        for (int i = 0; i < lushok; i++)
                            writer.Write(reader.ReadByte());
                    }
                    catch (IOException expt)
                    {
                        MessageBox.Show("Помилка при відкриті файлу, який потрібно
кодувати!!!\n" + expt.Message, "Помикла!");
                        is_good = false;
                    }
                    finally
                    {
```

```
                        if (stream != null)
                        {
                            stream.Close();
                        }
                        if (streamWrite != null)
                            streamWrite.Close();
                        if (is_good)
                        {
                            //Фуксуємо, що кодування проведено шифром ГОСТ
                            action = 3;           //Кодування файлу проведенно успішно
                            timeEnd = DateTime.Now;
                        }
                        this.Invalidate();
                    }
                }
            }
        }
        //Декодвання шифром ГОСТ
        private void menCodeGostDecode_Click(object sender, EventArgs e)
        {
            bool is_good = true;
            DES_Gost_DecodeForm dlgDeCode = new DES_Gost_DecodeForm();
            dlgDeCode.Text = "Декодування шифром ГОСТ";
            dlgDeCode.FileDecode = "Закодований.xxx";
            dlgDeCode.FileCode = "Розкодований.my";
            dlgDeCode.Key = "Мій ключ.key";
            if (dlgDeCode.ShowDialog(this) == DialogResult.OK)
            {
                sbPn1Prompt.Text = "Розпочато декодування файлу шифром ГОСТ";
                timeBegin = DateTime.Now;        //Now Time
                myFileDecode = dlgDeCode.FileDecode;
                myFileCode = dlgDeCode.FileCode;
                myKey = dlgDeCode.Key;
                int temp = 0;
                int leng = 8;
                uint[] key = new uint[leng];
                FileStream readKey = null;
                BinaryReader binKey = null;
                try
                {
                    readKey = new FileStream(myKey, FileMode.Open, FileAccess.Read);
                    binKey = new BinaryReader(readKey);
                    for (temp = 0; temp < leng; temp++)
                    {
                        key[temp] = binKey.ReadUInt32();
                    }
                    binKey.Close();
                }
                catch (IOException errorKey)
                {
                    MessageBox.Show("Помилка при відкриті файлу, який містить ключ!!!\n" +
errorKey.Message, "Помикла!");
                    is_good = false;
                }
                finally
                {
                    if (readKey != null)
                        readKey.Close();
                }
                if (temp != leng && is_good)
                {
                    MessageBox.Show("Розмір файлу, який містить ключ є надто малим!!!",
"Помилка!");
                    is_good = false;
```

```
                }
                FileStream stream = null;         //Файл, який потрібно читати
                FileStream streamWrite = null;    //Файл в який потрібно записувати
                BinaryReader reader = null;
                BinaryWriter writer = null;
                if (is_good)
                {
                    try
                    {
                        stream = new FileStream(myFileDecode, FileMode.Open,
FileAccess.Read);
                        streamWrite = new FileStream(myFileCode, FileMode.Create,
FileAccess.Write);
                        reader = new BinaryReader(stream);
                        writer = new BinaryWriter(streamWrite);
                        uint[] data = new uint[2];
                        GOST my_gost = new GOST();
                        my_gost.gost_key(key);
                        long processed = 0;
                        //size byte
                        long fileSize = stream.Length;
                        long lushok = fileSize % 8;
                        fileSize -= lushok;
                        while (processed < fileSize)
                        {
                            data[0] = reader.ReadUInt32();
                            data[1] = reader.ReadUInt32();
                            my_gost.gost_dec(ref data, 1);
                            writer.Write(data[0]);
                            writer.Write(data[1]);
                            processed += 8;    //Переміщаємося на 8 біт
                        }
                        for (int i = 0; i < lushok; i++)
                            writer.Write(reader.ReadByte());
                    }
                    catch (IOException expt)
                    {
                        MessageBox.Show("Помилка при відкриті файлу, який потрібно
кодувати!!!\n" + expt.Message, "Помикла!");
                        is_good = false;
                    }
                    finally
                    {
                        if (stream != null)
                        {
                            stream.Close();
                        }
                        if (streamWrite != null)
                            streamWrite.Close();
                        if (is_good)
                        {
                            action = 4;           //Декодування шифром ГОст
                            timeEnd = DateTime.Now;
                        }
                        this.Invalidate();
                    }
                }
            }
        }
        //Кодування шифром Blowfish
        private void mneCodeBlowfishCode_Click(object sender, EventArgs e)
        {
            bool is_good = true;
            Blow_Code dlgCode = new Blow_Code();
```

```
            dlgCode.FileCode = "my file.bin";
            dlgCode.FileDecode = "Закодований.xxx";
            dlgCode.Key = "Мій ключ.key";
            dlgCode.KeySize = 128;
            if (dlgCode.ShowDialog(this) == DialogResult.OK)
            {
                sbPn1Prompt.Text = "Розпочато кодування файлу шифром BlowFish";
                timeBegin = DateTime.Now;         //Now Time
                myFileCode = dlgCode.FileCode;
                myFileDecode = dlgCode.FileDecode;
                myKey = dlgCode.Key;
                mySizeKey = dlgCode.KeySize;
                mySizeKey -= mySizeKey % 8;
                int leng = mySizeKey;//розмір ключа
                leng /= 8;//size byte
                int temp = 0;
                byte[] key = new byte[leng];
                FileStream readKey = null;
                BinaryReader binKey = null;
                try
                {
                    readKey = new FileStream(myKey, FileMode.Open, FileAccess.Read);
                    binKey = new BinaryReader(readKey);
                    for (temp = 0; temp < leng; temp++)
                    {
                        key[temp] = binKey.ReadByte();
                    }
                    binKey.Close();
                }
                catch (IOException errorKey)
                {
                    MessageBox.Show("Помилка при відкриті файлу, який містить ключ!!!\n" +
errorKey.Message, "Помикла!");
                    is_good = false;
                }
                finally
                {
                    if (readKey != null)
                        readKey.Close();
                }
                if (temp != leng)
                {
                    MessageBox.Show("Розмір файлу, який містить ключ є надто малим!!!",
"Помилка!");
                    is_good = false;
                }
                FileStream stream = null;          //Файл, який потрібно читати
                FileStream streamWrite = null;     //Файл в який потрібно записувати
                BinaryReader reader = null;
                BinaryWriter writer = null;
                try
                {
                    stream = new FileStream(myFileCode, FileMode.Open, FileAccess.Read);
                    streamWrite = new FileStream(myFileDecode, FileMode.Create,
FileAccess.Write);
                    reader = new BinaryReader(stream);
                    writer = new BinaryWriter(streamWrite);
                    uint[] data = new uint[2];
                    BlowFish bl = new BlowFish();
                    bl.blf_key(key, (short)temp);
                    long processed = 0;
                    long fileSize = stream.Length;
                    long lushok = fileSize % 8;
                    fileSize -= lushok;
```

```
                    while (processed < fileSize)
                    {
                        data[0] = reader.ReadUInt32();
                        data[1] = reader.ReadUInt32();
                        bl.dlf_enc(ref data, 1);
                        writer.Write(data[0]);
                        writer.Write(data[1]);
                        processed += 8;    //Переміщаємося на 8 біт
                    }
                    for (int i = 0; i < lushok; i++)
                        writer.Write(reader.ReadByte());
                }
                catch (IOException expt)
                {
                    MessageBox.Show("Помилка при відкриті файлу, який потрібно
кодувати!!!\n" + expt.Message, "Помикла!");
                    is_good = false;
                }
                finally
                {
                    if (stream != null)
                    {
                        stream.Close();
                    }
                    if (streamWrite != null)
                        streamWrite.Close();
                    if (is_good)
                    {
                        action = 5;           //Кодування файлу BlowFish
                        timeEnd = DateTime.Now;
                    }
                    this.Invalidate();
                }
            }
        }
        //Декодування шифром BlowFish
        private void mneCodeBlowfishDecode_Click(object sender, EventArgs e)
        {
            bool is_good = true;
            BlowFishDecodeForm dlgDeCode = new BlowFishDecodeForm();
            dlgDeCode.FileDecode = "Закодований.xxx";
            dlgDeCode.FileCode = "Розкодований.my";
            dlgDeCode.Key = "Мій ключ.key";
            dlgDeCode.KeySize = 128;
            if (dlgDeCode.ShowDialog(this) == DialogResult.OK)
            {
                sbPn1Prompt.Text = "Розпочато декодування файлу шифром BlowFish";
                timeBegin = DateTime.Now;        //Now Time
                myFileDecode = dlgDeCode.FileDecode;
                myFileCode = dlgDeCode.FileCode;
                myKey = dlgDeCode.Key;
                mySizeKey = dlgDeCode.KeySize;
                mySizeKey -= mySizeKey % 8;
                int leng = mySizeKey;//розмір ключа
                leng /= 8;//size byte
                int temp = 0;
                byte[] key = new byte[leng];
                FileStream readKey = null;
                BinaryReader binKey = null;
                try
                {
                    readKey = new FileStream(myKey, FileMode.Open, FileAccess.Read);
                    binKey = new BinaryReader(readKey);
                    for (temp = 0; temp < leng; temp++)
```

```
                    {
                        key[temp] = binKey.ReadByte();
                    }
                    binKey.Close();
                }
                catch (IOException errorKey)
                {
                    MessageBox.Show("Помилка при відкриті файлу, який містить ключ!!!\n" +
errorKey.Message, "Помикла!");
                    is_good = false;
                }
                finally
                {
                    if (readKey != null)
                        readKey.Close();
                }
                if (temp != leng)
                {
                    MessageBox.Show("Розмір файлу, який містить ключ є надто малим!!!",
"Помилка!");
                    is_good = false;
                }
                FileStream stream = null;          //Файл, який потрібно читати
                FileStream streamWrite = null;     //Файл в який потрібно записувати
                BinaryReader reader = null;
                BinaryWriter writer = null;
                try
                {
                    stream = new FileStream(myFileDecode, FileMode.Open, FileAccess.Read);
                    streamWrite = new FileStream(myFileCode, FileMode.Create,
FileAccess.Write);
                    reader = new BinaryReader(stream);
                    writer = new BinaryWriter(streamWrite);
                    uint[] data = new uint[2];
                    BlowFish bl = new BlowFish();
                    bl.blf_key(key, (short)temp);
                    long processed = 0;
                    long fileSize = stream.Length;
                    long lushok = fileSize % 8;
                    fileSize -= lushok;
                    while (processed < fileSize)
                    {
                        data[0] = reader.ReadUInt32();
                        data[1] = reader.ReadUInt32();
                        bl.dlf_dec(ref data, 1);
                        writer.Write(data[0]);
                        writer.Write(data[1]);
                        processed += 8;    //Переміщаємося на 8 біт
                    }
                    for (int i = 0; i < lushok; i++)
                        writer.Write(reader.ReadByte());
                }
                catch (IOException expt)
                {
                    MessageBox.Show("Помилка при відкриті файлу, який потрібно
кодувати!!!\n" + expt.Message, "Помикла!");
                    is_good = false;
                }
                finally
                {
                    if (stream != null)
                    {
                        stream.Close();
                    }
```

```
                    if (streamWrite != null)
                        streamWrite.Close();
                    if (is_good)
                    {
                        action = 6;           //Кодування файлу проведенно успішно
                        timeEnd = DateTime.Now;
                    }
                    this.Invalidate();
                }
            }
        }
        //Прехід до початкового вікна
        private void menFilae_New_Click(object sender, EventArgs e)
        {
            action = 0;
            this.Invalidate();
        }
    }
}
```

ДОДАТОК 2. РЕАЛІЗАЦІЯ ОБ'ЄКТІВ ШИФРУВАННЯ ДАНИХ

```
    struct des_ctx
    {
        public uint[] ek;
        public uint[] dk;
    };

class DES
    {
        public DES()
        {
            KnL = new uint[32];
            KnR = new uint[32];
            Kn3 = new uint[32];
            Df_Key = new byte[24];
            byte[] xDf_Key ={
       0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef,
       0xfe,0xdc,0xba,0x98,0x76,0x54, 0x32, 0x10,
       0x89,0xab,0xcd,0xef,0x01,0x23, 0x45, 0x67
};
            Df_Key = xDf_Key;
            bytebit = new ushort[8];
            ushort[] xbytebit ={
0200, 0100, 040, 020, 010, 04, 02, 01 };
            bytebit = xbytebit;
            bigbyte = new uint[24];
            uint[] xbigbyte = {
(uint)0x800000L,    (uint)0x400000L,        (uint)0x200000L,    (uint)0x100000L,
(uint)0x80000L,     (uint)0x40000L,         (uint)0x20000L,     (uint)0x10000L,
(uint)0x8000L,      (uint)0x4000L,          (uint)0x2000L,      (uint)0x1000L,
(uint)0x800L,       (uint)0x400L,           (uint)0x200L,       (uint)0x100L,
(uint)0x80L,        (uint)0x40L,            (uint)0x20L,        (uint)0x10L,
(uint)0x8L,         (uint)0x4L,             (uint)0x2L,         (uint)0x1L   };
            bigbyte = xbigbyte;
            pc1 = new byte[56];
            byte[] xpc1 ={
56, 48, 40, 32, 24, 16, 8, 0, 57, 49, 41, 33, 25, 17,
9, 1, 58, 50, 42, 34, 26, 18, 10, 2, 59, 51, 43, 35,
62,54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37, 29, 21,
13, 5, 60, 52, 44, 36, 28, 20, 12, 4, 27, 19, 11, 3 };
            pc1 = xpc1;
            totrot = new byte[16];
```

```
            byte[] xtotrot ={
1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,28 };
            totrot = xtotrot;
            pc2 = new byte[48];
            byte[] xpc2 ={
13, 16, 10, 23,  0,  4, 2, 27, 14, 5, 20, 9,
22, 18, 11,  3, 25,  7, 15, 6, 26, 19, 12, 1,
40, 51, 30, 36, 46, 54, 29, 39, 50, 44, 32, 47,
43, 48, 38, 55, 33, 52, 45, 41, 49, 35, 28, 31 };
            pc2 = xpc2;
            ENO = 0;
            DEI = 1;
            uint[] xSP1 ={
       (uint)0x01010400L,   (uint)0x00000000L,   (uint)0x00010000L,   (uint)0x01010404L,
       (uint)0x01010004L,   (uint)0x00010404L,   (uint)0x00000004L,   (uint)0x00010000L,
       (uint)0x00000400L,   (uint)0x01010400L,   (uint)0x01010404L,   (uint)0x00000400L,
       (uint)0x01000404L,   (uint)0x01010004L,   (uint)0x01000000L,   (uint)0x00000004L,
       (uint)0x00000404L,   (uint)0x01000400L,   (uint)0x01000400L,   (uint)0x00010400L,
       (uint)0x00010400L,   (uint)0x01010000L,   (uint)0x01010000L,   (uint)0x01000404L,
       (uint)0x00010004L,   (uint)0x01000004L,   (uint)0x01000004L,   (uint)0x00010004L,
       (uint)0x00000000L,   (uint)0x00000404L,   (uint)0x00010404L,   (uint)0x01000000L,
       (uint)0x00010000L,   (uint)0x01010404L,   (uint)0x00000004L,   (uint)0x01010000L,
       (uint)0x01010400L,   (uint)0x01000000L,   (uint)0x01000000L,   (uint)0x00000400L,
       (uint)0x01010004L,   (uint)0x00010000L,   (uint)0x00010400L,   (uint)0x01000004L,
       (uint)0x00000400L,   (uint)0x00000004L,   (uint)0x01000404L,   (uint)0x00010404L,
       (uint)0x01010404L,   (uint)0x00010004L,   (uint)0x01010000L,   (uint)0x01000404L,
       (uint)0x01000004L,   (uint)0x00000404L,   (uint)0x00010404L,   (uint)0x01010400L,
       (uint)0x00000404L,   (uint)0x01000400L,   (uint)0x01000400L,   (uint)0x00000000L,
       (uint)0x00010004L,   (uint)0x00010400L,   (uint)0x00000000L,   (uint)0x01010004L };
            SP1 = xSP1;
            uint[] xSP2 ={
       (uint)0x80108020L,   (uint)0x80008000L,   (uint)0x00008000L,   (uint)0x00108020L,
       (uint)0x00100000L,   (uint)0x00000020L,   (uint)0x80100020L,   (uint)0x80008020L,
       (uint)0x80000020L,   (uint)0x80108020L,   (uint)0x80108000L,   (uint)0x80000000L,
       (uint)0x80008000L,   (uint)0x00100000L,   (uint)0x00000020L,   (uint)0x80100020L,
       (uint)0x00108000L,   (uint)0x00100020L,   (uint)0x80008020L,   (uint)0x00000000L,
       (uint)0x80000000L,   (uint)0x00008000L,   (uint)0x00108020L,   (uint)0x80100000L,
       (uint)0x00100020L,   (uint)0x80000020L,   (uint)0x00000000L,   (uint)0x00108000L,
       (uint)0x00008020L,   (uint)0x80108000L,   (uint)0x80100000L,   (uint)0x00008020L,
       (uint)0x00000000L,   (uint)0x00108020L,   (uint)0x80100020L,   (uint)0x00100000L,
       (uint)0x80008020L,   (uint)0x80100000L,   (uint)0x80108000L,   (uint)0x00008000L,
       (uint)0x80100000L,   (uint)0x80008000L,   (uint)0x00000020L,   (uint)0x80108020L,
       (uint)0x00108020L,   (uint)0x00000020L,   (uint)0x00008000L,   (uint)0x80000000L,
       (uint)0x00008020L,   (uint)0x80108000L,   (uint)0x00100000L,   (uint)0x80000020L,
       (uint)0x00100020L,   (uint)0x80008020L,   (uint)0x80000020L,   (uint)0x00100020L,
       (uint)0x00108000L,   (uint)0x00000000L,   (uint)0x80008000L,   (uint)0x00008020L,
       (uint)0x80000000L,   (uint)0x80100020L,   (uint)0x80108020L,   (uint)0x00108000L };
            SP2 = xSP2;
            uint[] xSP3 ={
       (uint)0x00000208L,   (uint)0x08020200L,   (uint)0x00000000L,   (uint)0x08020008L,
       (uint)0x08000200L,   (uint)0x00000000L,   (uint)0x00020208L,   (uint)0x08000200L,
       (uint)0x00020008L,   (uint)0x08000008L,   (uint)0x08000008L,   (uint)0x00020000L,
       (uint)0x08020208L,   (uint)0x00020008L,   (uint)0x08020000L,   (uint)0x00000208L,
       (uint)0x08000000L,   (uint)0x00000008L,   (uint)0x08020200L,   (uint)0x00000200L,
       (uint)0x00020200L,   (uint)0x08020000L,   (uint)0x08020008L,   (uint)0x00020208L,
       (uint)0x08000208L,   (uint)0x00020200L,   (uint)0x00020000L,   (uint)0x08000208L,
       (uint)0x00000008L,   (uint)0x08020208L,   (uint)0x00000200L,   (uint)0x08000000L,
       (uint)0x08020200L,   (uint)0x08000000L,   (uint)0x00020008L,   (uint)0x00000208L,
       (uint)0x00020000L,   (uint)0x08020200L,   (uint)0x08000200L,   (uint)0x00000000L,
       (uint)0x00000200L,   (uint)0x00020008L,   (uint)0x08020208L,   (uint)0x08000200L,
       (uint)0x08000008L,   (uint)0x00000200L,   (uint)0x00000000L,   (uint)0x08020008L,
       (uint)0x08000208L,   (uint)0x00020000L,   (uint)0x08000000L,   (uint)0x08020208L,
       (uint)0x00000008L,   (uint)0x00020208L,   (uint)0x00020200L,   (uint)0x08000008L,
       (uint)0x08020000L,   (uint)0x08000208L,   (uint)0x00000208L,   (uint)0x08020000L,
```

```
        (uint)0x00020208L,   (uint)0x00000008L,   (uint)0x08020008L,
(uint)0x00020200L };
             SP3 = xSP3;
             uint[] xSP4 ={
        (uint)0x00802001L,   (uint)0x00002081L,   (uint)0x00002081L,   (uint)0x00000080L,
        (uint)0x00802080L,   (uint)0x00800081L,   (uint)0x00800001L,   (uint)0x00002001L,
        (uint)0x00000000L,   (uint)0x00802000L,   (uint)0x00802000L,   (uint)0x00802081L,
        (uint)0x00000081L,   (uint)0x00000000L,   (uint)0x00800080L,   (uint)0x00800001L,
        (uint)0x00000001L,   (uint)0x00002000L,   (uint)0x00800000L,   (uint)0x00802001L,
        (uint)0x00000080L,   (uint)0x00800000L,   (uint)0x00002001L,   (uint)0x00002080L,
        (uint)0x00800081L,   (uint)0x00000001L,   (uint)0x00002080L,   (uint)0x00800080L,
        (uint)0x00002000L,   (uint)0x00802080L,   (uint)0x00802081L,   (uint)0x00000081L,
        (uint)0x00800080L,   (uint)0x00800001L,   (uint)0x00802000L,   (uint)0x00802081L,
        (uint)0x00000081L,   (uint)0x00000000L,   (uint)0x00000000L,   (uint)0x00802000L,
        (uint)0x00002080L,   (uint)0x00800080L,   (uint)0x00800081L,   (uint)0x00000001L,
        (uint)0x00802001L,   (uint)0x00002081L,   (uint)0x00002081L,   (uint)0x00000080L,
        (uint)0x00802081L,   (uint)0x00000081L,   (uint)0x00000001L,   (uint)0x00002000L,
        (uint)0x00800001L,   (uint)0x00002001L,   (uint)0x00802080L,   (uint)0x00800081L,
        (uint)0x00002001L,   (uint)0x00002080L,   (uint)0x00800000L,   (uint)0x00802001L,
        (uint)0x00000080L,   (uint)0x00800000L,   (uint)0x00002000L,   (uint)0x00802080L };
             SP4 = xSP4;
             uint[] xSP5 ={
        (uint)0x00000100L,   (uint)0x02080100L,   (uint)0x02080000L,   (uint)0x42000100L,
        (uint)0x00080000L,   (uint)0x00000100L,   (uint)0x40000000L,   (uint)0x02080000L,
        (uint)0x40080100L,   (uint)0x00080000L,   (uint)0x02000100L,   (uint)0x40080100L,
        (uint)0x42000100L,   (uint)0x42080000L,   (uint)0x00080100L,   (uint)0x40000000L,
        (uint)0x02000000L,   (uint)0x40080000L,   (uint)0x40080000L,   (uint)0x00000000L,
        (uint)0x40000100L,   (uint)0x42080100L,   (uint)0x42080100L,   (uint)0x02000100L,
        (uint)0x42080000L,   (uint)0x40000100L,   (uint)0x00000000L,   (uint)0x42000000L,
        (uint)0x02080100L,   (uint)0x02000000L,   (uint)0x42000000L,   (uint)0x00080100L,
        (uint)0x00080000L,   (uint)0x42000100L,   (uint)0x00000100L,   (uint)0x02000000L,
        (uint)0x40000000L,   (uint)0x02080000L,   (uint)0x42000100L,   (uint)0x40080100L,
        (uint)0x02000100L,   (uint)0x40000000L,   (uint)0x42080000L,   (uint)0x02080100L,
        (uint)0x40080100L,   (uint)0x00000100L,   (uint)0x02000000L,   (uint)0x42080000L,
        (uint)0x42080100L,   (uint)0x00080100L,   (uint)0x42000000L,   (uint)0x42080100L,
        (uint)0x02080000L,   (uint)0x00000000L,   (uint)0x40080000L,   (uint)0x42000000L,
        (uint)0x00080100L,   (uint)0x02000100L,   (uint)0x40000100L,   (uint)0x00080000L,
        (uint)0x00000000L,   (uint)0x40080000L,   (uint)0x02080100L,   (uint)0x40000100L };
             SP5 = xSP5;
             uint[] xSP6 ={
        (uint)0x20000010L,   (uint)0x20400000L,   (uint)0x00004000L,   (uint)0x20404010L,
        (uint)0x20400000L,   (uint)0x00000010L,   (uint)0x20404010L,   (uint)0x00400000L,
        (uint)0x20004000L,   (uint)0x00404010L,   (uint)0x00400000L,   (uint)0x20000010L,
        (uint)0x00400010L,   (uint)0x20004000L,   (uint)0x20000000L,   (uint)0x00004010L,
        (uint)0x00000000L,   (uint)0x00400010L,   (uint)0x20004010L,   (uint)0x00004000L,
        (uint)0x00404000L,   (uint)0x20004010L,   (uint)0x00000010L,   (uint)0x20400010L,
        (uint)0x20400010L,   (uint)0x00000000L,   (uint)0x00404010L,   (uint)0x20404000L,
        (uint)0x00004010L,   (uint)0x00404000L,   (uint)0x20404000L,   (uint)0x20000000L,
        (uint)0x20004000L,   (uint)0x00000010L,   (uint)0x20400010L,   (uint)0x00404000L,
        (uint)0x20404010L,   (uint)0x00400000L,   (uint)0x00004010L,   (uint)0x20000010L,
        (uint)0x00400000L,   (uint)0x20004000L,   (uint)0x20000000L,   (uint)0x00004010L,
        (uint)0x20000010L,   (uint)0x20404010L,   (uint)0x00404000L,   (uint)0x20400000L,
        (uint)0x00404010L,   (uint)0x20404000L,   (uint)0x00000000L,   (uint)0x20400010L,
        (uint)0x00000010L,   (uint)0x00004000L,   (uint)0x20400000L,   (uint)0x00404010L,
        (uint)0x00004000L,   (uint)0x00400010L,   (uint)0x20004010L,   (uint)0x00000000L,
        (uint)0x20404000L,   (uint)0x20000000L,   (uint)0x00400010L,   (uint)0x20004010L };
             SP6 = xSP6;
             uint[] xSP7 ={
        (uint)0x00200000L,   (uint)0x04200002L,   (uint)0x04000802L,   (uint)0x00000000L,
        (uint)0x00000800L,   (uint)0x04000802L,   (uint)0x00200802L,   (uint)0x04200800L,
        (uint)0x04200802L,   (uint)0x00200000L,   (uint)0x00000000L,   (uint)0x04000002L,
        (uint)0x00000002L,   (uint)0x04000000L,   (uint)0x04200002L,   (uint)0x00000802L,
        (uint)0x04000800L,   (uint)0x00200802L,   (uint)0x00200002L,   (uint)0x04000800L,
        (uint)0x04000002L,   (uint)0x04200000L,   (uint)0x04200800L,   (uint)0x00200002L,
```

```
        (uint)0x04200000L,   (uint)0x00000800L,   (uint)0x00000802L,
(uint)0x04200802L,
        (uint)0x00200800L,   (uint)0x00000002L,   (uint)0x04000000L,   (uint)0x00200800L,
        (uint)0x04000000L,   (uint)0x00200800L,   (uint)0x00200000L,   (uint)0x04000802L,
        (uint)0x04000802L,   (uint)0x04200002L,   (uint)0x04200002L,   (uint)0x00000002L,
        (uint)0x00200002L,   (uint)0x04000000L,   (uint)0x04000800L,   (uint)0x00200000L,
        (uint)0x04200800L,   (uint)0x00000802L,   (uint)0x00200802L,   (uint)0x04200800L,
        (uint)0x00000802L,   (uint)0x04000002L,   (uint)0x04200802L,   (uint)0x04200000L,
        (uint)0x00200800L,   (uint)0x00000000L,   (uint)0x00000002L,   (uint)0x04200802L,
        (uint)0x00000000L,   (uint)0x00200802L,   (uint)0x04200000L,   (uint)0x00000800L,
        (uint)0x04000002L,   (uint)0x04000800L,   (uint)0x00000800L,   (uint)0x00200002L };
             SP7 = xSP7;
             uint[] xSP8 ={
        (uint)0x10001040L,   (uint)0x00001000L,   (uint)0x00040000L,   (uint)0x10041040L,
        (uint)0x10000000L,   (uint)0x10001040L,   (uint)0x00000040L,   (uint)0x10000000L,
        (uint)0x00040040L,   (uint)0x10040000L,   (uint)0x10041040L,   (uint)0x00041000L,
        (uint)0x10041000L,   (uint)0x00041040L,   (uint)0x00001000L,   (uint)0x00000040L,
        (uint)0x10040000L,   (uint)0x10000040L,   (uint)0x10001000L,   (uint)0x00001040L,
        (uint)0x00041000L,   (uint)0x00040040L,   (uint)0x10040040L,   (uint)0x10041000L,
        (uint)0x00001040L,   (uint)0x00000000L,   (uint)0x00000000L,   (uint)0x10040040L,
        (uint)0x10000040L,   (uint)0x10001000L,   (uint)0x00041040L,   (uint)0x00040000L,
        (uint)0x00041040L,   (uint)0x00040000L,   (uint)0x10041000L,   (uint)0x00001000L,
        (uint)0x00000040L,   (uint)0x10040040L,   (uint)0x00001000L,   (uint)0x00041040L,
        (uint)0x10001000L,   (uint)0x00000040L,   (uint)0x10000040L,   (uint)0x10040000L,
        (uint)0x10040040L,   (uint)0x10000000L,   (uint)0x00040000L,   (uint)0x10001040L,
        (uint)0x00000000L,   (uint)0x10041040L,   (uint)0x00040040L,   (uint)0x10000040L,
        (uint)0x10040000L,   (uint)0x10001000L,   (uint)0x10001040L,   (uint)0x00000000L,
        (uint)0x10041040L,   (uint)0x00041000L,   (uint)0x00041000L,   (uint)0x00001040L,
        (uint)0x00001040L,   (uint)0x00040040L,   (uint)0x10000000L,   (uint)0x10041000L };
             SP8 = xSP8;
             dc = new des_ctx();
             dc.dk = new uint[32];
             dc.ek = new uint[32];
         }
         private des_ctx dc;
         private uint[] KnL;
         private uint[] KnR;
         private uint[] Kn3;
         private byte[] Df_Key;
         private ushort[] bytebit;
         private uint[] bigbyte;
         private byte[] pc1;
         private byte[] totrot;
         private byte[] pc2;
         private short ENO;
         private short DEI;
         private uint[] SP1;
         private uint[] SP2;
         private uint[] SP3;
         private uint[] SP4;
         private uint[] SP5;
         private uint[] SP6;
         private uint[] SP7;
         private uint[] SP8;
         //Перестановка со зжатием
         private void deskey(ref byte[] key, short edf)  //Thanks to James Gillogly & Phil
Karn!
         {
             int i, j, l, m, n;
             byte[] pclm = new byte[56];
             byte[] pcr = new byte[56];
             uint[] kn = new uint[32];
             for (j = 0; j < 56; j++)
             {
```

```
            l = pc1[j];
            m = l & 07;
            if ((key[l >> 3] & bytebit[m]) >= 0)
                pclm[j] = 1;
            else
                pclm[j] = 0;
            //pclm[j] = (key[l >> 3] & bytebit[m]) ? 1 : 0;
        }
        for (i = 0; i < 16; i++)
        {
            if (edf == DEI)
                m = (15 - i) << 1;
            else m = i << 1;
            n = m + 1;
            kn[m] = kn[n] = 0;
            for (j = 0; j < 28; j++)
            {
                l = j + totrot[i];
                if (l < 28)
                    pcr[j] = pclm[l];
                else
                    pcr[j] = pclm[l - 28];
            }
            for (j = 28; j < 56; j++)
            {
                l = j + totrot[i];
                if (l < 56)
                    pcr[j] = pclm[l];
                else pcr[j] = pclm[l - 28];
            }
            for (j = 0; j < 24; j++)
            {
                if (pcr[pc2[j]] >= 1)
                    kn[m] |= bigbyte[j];
                if (pcr[pc2[j + 24]] >= 1)
                    kn[n] |= bigbyte[j];
            }
        }
        cookey(ref kn);
        return;
    }
    private void cookey(ref uint[] rawl)
    {
        uint[] cook;
        uint raw0;
        uint[] dough = new uint[32];
        int i;
        int j = 0;
        cook = dough;
        for (i = 0; i < 16; i++)
        {
            raw0 = rawl[j];
            cook[j] = (raw0 & (uint)0x0fc0000L) << 6;
            cook[j] |= (raw0 & (uint)0x00000fc0L) << 10;
            cook[j] |= (rawl[j + 1] & (uint)0x00fc0000L) >> 10;
            cook[j] |= (rawl[j + 1] & (uint)0x00000fc0L) >> 6;
            cook[j + 1] = (raw0 & (uint)0x0003f000L) << 12;
            cook[j + 1] |= (raw0 & (uint)0x0000003fL) << 16;
            cook[j + 1] |= (rawl[j + 1] & (uint)0x0003f000L) >> 4;
            cook[j + 1] |= (rawl[j + 1] & (uint)0x0000003fL);
            j += 2;
        }
        usekey(ref dough);
        return;
```

```
}
private void cpkey(ref uint[] into)
{
    for (int i = 0; i < 32; i++)
        into[i] = KnL[i];
}
private void usekey(ref uint[] from)
{
    for (int i = 0; i < 32; i++)
        from[i] = KnL[i];
}
private void des(ref byte[] inblock, ref byte[] outblock)
{
    uint[] work = new uint[2];
    scrunch(ref inblock, ref work);
    //   desfunc(work, KnL);
    unscrun(ref work, ref outblock);
    return;
}
private void scrunch(ref byte[] outof, ref uint[] into)
{
    int j = 0;
    into[0] = (outof[j++] & (uint)0xffL) << 24;
    into[0] |= (outof[j++] & (uint)0xffL) << 16;
    into[0] |= (outof[j++] & (uint)0xffL) << 8;
    into[0] |= (outof[j++] & (uint)0xffL);
    into[1] = (outof[j++] & (uint)0xffL) << 24;
    into[1] |= (outof[j++] & (uint)0xffL) << 16;
    into[1] |= (outof[j++] & (uint)0xffL) << 8;
    into[1] |= (outof[j] & (uint)0xffL);
    return;
}
private void unscrun(ref uint[] outof, ref byte[] into)
{
    int j = 0;
    into[j++] = (byte)((outof[0] >> 24) & (uint)0xffL);
    into[j++] = (byte)((outof[0] >> 16) & (uint)0xffL);
    into[j++] = (byte)((outof[0] >> 8) & (uint)0xffL);
    into[j++] = (byte)(outof[0] & (uint)0xffL);
    into[j++] = (byte)((outof[1] >> 24) & (uint)0xffL);
    into[j++] = (byte)((outof[1] >> 16) & (uint)0xffL);
    into[j++] = (byte)((outof[1] >> 8) & (uint)0xffL);
    into[j] = (byte)(outof[1] & (uint)0xffL);
    return;
}
private void desfunc(ref uint[] block, ref uint[] keys)
{
    uint fval, work, right, leftt;
    int round;
    leftt = block[0]; right = block[1];
    work = ((leftt >> 4) ^ right) & (uint)0x0f0f0f0fL;
    right ^= work;
    leftt ^= (work << 4);
    work = ((leftt >> 16) ^ right) & (uint)0x0000ffffL;
    right ^= work;
    leftt ^= (work << 16);
    work = ((right >> 2) ^ leftt) & (uint)0x33333333L;
    leftt ^= work;
    right ^= (work << 2);
    work = ((right >> 8) ^ leftt) & (uint)0x00ff00ffL;
    leftt ^= work;
    right ^= (work << 8);
    right = ((right << 1) | ((right >> 31) & (uint)1L)) & (uint)0xffffffffL;
    work = (leftt ^ right) & (uint)0xaaaaaaaaL;
```

```
            leftt ^= work;
            right ^= work;
            leftt = ((leftt << 1) | ((leftt >> 31) & (uint)1L)) & (uint)0xffffffffL;
            int j = 0;
            for (round = 0; round < 8; round++)
            {
                work = (right << 28) | (right >> 4);
                work ^= keys[j++];
                fval = SP7[work & (uint)0x3fL];
                fval |= SP5[(work >> 8) & (uint)0x3fL];
                fval |= SP3[(work >> 16) & (uint)0x3fL];
                fval |= SP1[(work >> 24) & (uint)0x3fL];
                work = right ^ keys[j++];
                fval |= SP8[work & (uint)0x3fL];
                fval |= SP6[(work >> 8) & (uint)0x3fL];
                fval |= SP4[(work >> 16) & (uint)0x3fL];
                fval |= SP2[(work >> 24) & (uint)0x3fL];
                leftt ^= fval;
                work = (leftt << 28) | (leftt >> 4);
                work ^= keys[j++];
                fval = SP7[work & (uint)0x3fL];
                fval |= SP5[(work >> 8) & (uint)0x3fL];
                fval |= SP3[(work >> 16) & (uint)0x3fL];
                fval |= SP1[(work >> 24) & (uint)0x3fL];
                work = leftt ^ keys[j++];
                fval |= SP8[work & (uint)0x3fL];
                fval |= SP6[(work >> 8) & (uint)0x3fL];
                fval |= SP4[(work >> 16) & (uint)0x3fL];
                fval |= SP2[(work >> 24) & (uint)0x3fL];
                right ^= fval;
            }
            right = (right << 31) | (right >> 1);
            work = (leftt ^ right) & (uint)0xaaaaaaaaL;
            leftt ^= work;
            right ^= work;
            leftt = (leftt << 31) | (leftt >> 1);
            work = ((leftt >> 8) ^ right) & (uint)0x00ff00ffL;
            right ^= work;
            leftt ^= (work << 8);
            work = ((leftt >> 2) ^ right) & (uint)0x33333333L;
            right ^= work;
            leftt ^= (work << 2);
            work = ((right >> 16) ^ leftt) & (uint)0x0000ffffL;
            leftt ^= work;
            right ^= (work << 16);
            work = ((right >> 4) ^ leftt) & (uint)0x0f0f0f0fL;
            leftt ^= work;
            right ^= (work << 4);
            block[0] = right;
            block[1] = leftt;
            return;
        }
        /*   Validation sets:
* Single-length key,   single-length plaintext -
* Key    : 0123 4567   89ab cdef
* Plain  : 0123 4567   89ab cde7
* Cipher : c957 4425   6a5e d31d *
********************************************/
        public void des_key(ref byte[] key)
        {
            deskey(ref key, ENO);
            cpkey(ref dc.ek);
            deskey(ref key, DEI);
            cpkey(ref dc.dk);
```

```
        }
        /*
ECB режим ризначений для шифрування в банківських системах
*/
        /* Encrypt several blocks in ECB mode.  Caller is responsible for
        short blocks. */
        public void des_enc(ref byte[] data)
        {
            uint[] work = new uint[2];
            byte[] cp;
            cp = data;
            scrunch(ref cp, ref work);
            desfunc(ref work, ref dc.ek);
            unscrun(ref work, ref cp);
        }
        public void des_dec(ref byte[] data)
        {
            uint[] work = new uint[2];
            byte[] cp;
            cp = data;
            scrunch(ref cp, ref work);
            desfunc(ref work, ref dc.dk);
            unscrun(ref work, ref cp);

        }

    }
// u4 unsigned long u4; in C# uint
    struct gost_ctx
    {
        public uint[] k;
        /* Constant s-boxes -- set up in gost_init(). */
        public byte[] k87;
        public byte[] k65;
        public byte[] k43;
        public byte[] k21;
    };
    class GOST
    {
        private gost_ctx c;
        public GOST()
        {
            this.c.k = new uint[8];
            this.c.k87 = new byte[256];
            this.c.k65 = new byte[256];
            this.c.k43 = new byte[256];
            this.c.k21 = new byte[256];
            this.kboxinit();
        }
        private void kboxinit()
        {
            int i;
            byte[] k8 = {14,      4,   13,      1,     2, 15, 11,  8,  3, 10,  6,
              12,   5,      9,      0,      7 };
            byte[] k7 = {15,      1,      8,   14,      6, 11,  3,  4,  9,  7,  2,
              13,  12,      0,      5,   10 };
            byte[] k6 = {10,      0,      9,   14,      6,  3, 15,  5,  1, 13, 12,
              7,   11,      4,      2,      8 };
            byte[] k5 = { 7,   13,   14,      3,      0,  6,  9, 10,  1,  2,  8,
              5,   11,   12,      4,   15 };
            byte[] k4 = { 2,   12,      4,      1,      7, 10, 11,  6,  8,  5,  3,
              15,   13,      0,   14,      9 };
            byte[] k3 = {12,      1,   10,   15,      9,  2,  6,  8,  0, 13,  3,
              4,   14, 7,      5,   11 };
```

```
            byte[] k2 =   {   4,   11,      2,   14,   15,      0,      8,   13,
3,   12,      9,
              7,      5,   10,      6,      1   };
            byte[] kl =   {13,      2,      8,      4,      6,   15,   11,      1,   10,
9,      3,
              14,      5,      0,     12,      7   };
            for (i = 0; i < 256; i++)
            {
                c.k87[i] = (byte)(k8[i >> 4] << 4 | k7[i & 15]);
                c.k65[i] = (byte)(k6[i >> 4] << 4 | k5[i & 15]);
                c.k43[i] = (byte)(k4[i >> 4] << 4 | k3[i & 15]);
                c.k21[i] = (byte)(k2[i >> 4] << 4 | kl[i & 15]);
            }
        }
        private uint f(uint x)
        {
            x = (uint)(c.k87[x >> 24 & 255] << 24 | c.k65[x >> 16 & 255] << 16 | c.k43[x
>> 8 & 255] << 8 | c.k21[x & 255]);
            /* Rotate left 11 bits */
            return x << 11 | x >> (32 - 11);
        }
        public void gostcrypt(ref uint d, ref uint d1)
        {
            uint nl, n2; /* As named in the GOST */
            nl = d;
            n2 = d1;
            /*   Instead  of   swapping  halves,   swap  names   each  round  */
            n2 ^= f(nl + c.k[0]); nl ^= f(n2 + c.k[1]);
            n2 ^= f(nl + c.k[2]); nl ^= f(n2 + c.k[3]);
            n2 ^= f(nl + c.k[4]); nl ^= f(n2 + c.k[5]);
            n2 ^= f(nl + c.k[6]); nl ^= f(n2 + c.k[7]);
            n2 ^= f(nl + c.k[0]); nl ^= f(n2 + c.k[1]);
            n2 ^= f(nl + c.k[2]); nl ^= f(n2 + c.k[3]);
            n2 ^= f(nl + c.k[4]); nl ^= f(n2 + c.k[5]);
            n2 ^= f(nl + c.k[6]); nl ^= f(n2 + c.k[7]);
            n2 ^= f(nl + c.k[0]); nl ^= f(n2 + c.k[1]);
            n2 ^= f(nl + c.k[2]); nl ^= f(n2 + c.k[3]);
            n2 ^= f(nl + c.k[4]); nl ^= f(n2 + c.k[5]);
            n2 ^= f(nl + c.k[6]); nl ^= f(n2 + c.k[7]);
            n2 ^= f(nl + c.k[7]); nl ^= f(n2 + c.k[6]);
            n2 ^= f(nl + c.k[5]); nl ^= f(n2 + c.k[4]);
            n2 ^= f(nl + c.k[3]); nl ^= f(n2 + c.k[2]);
            n2 ^= f(nl + c.k[1]); nl ^= f(n2 + c.k[0]);
            d = n2;
            d1 = nl;
        }
        private void gostdecrypt(ref uint d, ref uint d1)
        {
            uint nl, n2; /* As named in the GOST */
            nl = d; n2 = d1;
            n2 ^= f(nl + c.k[0]); nl ^= f(n2 + c.k[1]);
            n2 ^= f(nl + c.k[2]); nl ^= f(n2 + c.k[3]);
            n2 ^= f(nl + c.k[4]); nl ^= f(n2 + c.k[5]);
            n2 ^= f(nl + c.k[6]); nl ^= f(n2 + c.k[7]);
            n2 ^= f(nl + c.k[7]); nl ^= f(n2 + c.k[6]);
            n2 ^= f(nl + c.k[5]); nl ^= f(n2 + c.k[4]);
            n2 ^= f(nl + c.k[3]); nl ^= f(n2 + c.k[2]);
            n2 ^= f(nl + c.k[1]); nl ^= f(n2 + c.k[0]);
            n2 ^= f(nl + c.k[7]); nl ^= f(n2 + c.k[6]);
            n2 ^= f(nl + c.k[5]); nl ^= f(n2 + c.k[4]);
            n2 ^= f(nl + c.k[3]); nl ^= f(n2 + c.k[2]);
            n2 ^= f(nl + c.k[1]); nl ^= f(n2 + c.k[0]);
            n2 ^= f(nl + c.k[7]); nl ^= f(n2 + c.k[6]);
            n2 ^= f(nl + c.k[5]); nl ^= f(n2 + c.k[4]);
```

```
            n2 ^= f(nl + c.k[3]); nl ^= f(n2 + c.k[2]);
            n2 ^= f(nl + c.k[1]); nl ^= f(n2 + c.k[0]);
            d = n2; d1 = nl;
        }
        public void gost_enc(ref uint[] d, int blocks)
        {
            int i;
            int j = 0;
            for (i = 0; i < blocks; i++)
            {
                gostcrypt(ref d[j], ref d[j + 1]);
                j += 2;
            }
        }
        public void gost_dec(ref uint[] d, int blocks)
        {
            int i;
            int j = 0;
            for (i = 0; i < blocks; i++)
            {
                gostdecrypt(ref d[j], ref d[j + 1]);
                j += 2;
            }
        }
        public void gost_key(uint[] k)
        {
            int i;
            for (i = 0; i < 8; i++)
                c.k[i] = k[i];
        }
        public void gost_destroy()
        {
            int i;
            for (i = 0; i < 8; i++)
                c.k[i] = 0;
        }
    }


class RSA
    {
        //Піднесення до степеня n по модулю
        public long qe2(long x, int y, int n)
        {
            long s, t, u;
            s = 1; t = x; u = (long)y;
            while (u > 0)
            {
                if ((u & 1) > 0)        //якщо виконується побітова операція і
                    s = (s * t) % (long)n;
                u = u >> 1;//зміщення u на один біт вправо
                t = (t * t) % (long)n;
            }
            return (s);
        }
        //Найбільший спільний дільник двох чисел
        public int nsd(int x, int y)
        {
            int g = 0;
            if (x < 0)
                x = -x;
            if (y < 0)
                y = -y;
            if ((x + y) == 0)
```

```
            return -1;
        g = y;
        while (x > 0)
        {
            g = x;
            x = y % x;
            y = g;
        }
        return g;
    }
    //Функція Ойлера
    public int fi(int p, int q)
    {
        return ((p - 1) * (q - 1));
    }
    //Допоміжні функції
    private void swap(ref int x, ref int y)  //обмін значенми
    {
        x ^= y; y ^= x; x ^= y;
    }
    private void swaptype(ref int x, ref int y)  //обмін значенми
    {
        x ^= y; y ^= x; x ^= y;
    }
    private bool isEvent(int x)  //Перевірка числа
    {
        if ((x & 0x01) == 0)
            return true;
        else
            return false;
    }
    private bool isOdd(int x)
    {
        if ((x & 0x01) > 0)
            return true;
        return false;

    }
    public void ExtBinEuclid(ref int u, ref int v, ref int u1, ref int u2, ref int u3)
    {
        int k, t1, t2, t3;
        if (u < v)
            swaptype(ref u, ref v);
        for (k = 0; (isEvent(u) && isEvent(v)); k++)
        {
            u >>= 1;
            v >>= 1;
        }
        u1 = 1; u2 = 0; u3 = u; t1 = v; t2 = u - 1; t3 = v;
        do
        {
            do
            {
                if (isEvent(u3))
                {
                    if (isOdd(u1) || isOdd(u2))
                    {
                        u1 += v;
                        u2 += u;
                    }
                    u1 >>= 1; u2 >>= 1; u3 >>= 1;
                }
                if (isEvent(t3) || u3 < t3)
                {
```

```
                swaptype(ref u1, ref t1);
                swaptype(ref u2, ref t2);
                swaptype(ref u3, ref t3);
            }
        }
        while (isEvent(u3));
        while (u1 < t1 || u2 < t2)
        {
            u1 += v;
            u2 += u;
        }
        u1 -= t1;
        u2 -= t2;
        u3 -= t3;
    }
    while (t3 > 0);
    while (u1 >= v && u2 >= u)
    {
        u1 -= v; u2 -= u;
    }
    u <<= k;
    v <<= k;
    u3 <<= k;
}
//генерація ряду простих чисел
private void GetSimples(List<int> list, int max)
{
    list.Clear();
    list.Add(1);
    list.Add(2);
    list.Add(3);
    for (int n = 5; list.Count < max; n += 2)
    {
        bool b = true;
        for (int i = 1; i < list.Count; i++)
        {
            if ((n % list[i]) == 0)
            {
                b = false;
                break;
            }
        }
        if (b)
            list.Add(n);
    }
}
//Повертає просте число
private int simple(int n)
{
    //Список простих чисел діапазону
    List<int> simples = new List<int>();
    //генеруємо прості числа
    GetSimples(simples, n);
    return simples[n - 1];
}
//Шукаю взаємно просте число з n
private int GetExp(int n)
{
    Random r = new Random();
    int p = 0;
    bool z = true;
    while (z)
    {
        p = r.Next(n);
```

```
                if (nsd(p, n) == 1)
                    z = false;
            }
            return p;
        }
        public void GenKey(ref int n, ref int e, ref int d)
        {
            Random r = new Random();
            int z = r.Next(56, 4793);
            int p = simple(z);
            z = r.Next(56, 4793);
            int q = simple(z);
            n = p * q;
            int Oyl = fi(p, q);
            e = GetExp(Oyl);
            int a = 0, b = 0, gcd = 0;
            int emod = e;
            int Oylmod = Oyl;
            ExtBinEuclid(ref emod, ref Oylmod, ref a, ref b, ref gcd);
            if (gcd == 1)
                d = emod - b;
        }
    }
```

ДОДАТОК 3. ПРОГРАМУВАННЯ ДОДАТКОВИХ ІНТЕРФЕЙСІВ

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Windows.Forms;
using System.Data.OleDb;
using System.IO;

using System.Web;
using System.Net;
using System.Net.Mail;
using System.Net.Mime;

namespace Cryptogram
{
    public partial class ListUserBaseForm : Form
    {
        //Підключення до БД
        private OleDbConnection cn; //= new
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" + @"data source=bread.mdb");
        private System.Data.OleDb.OleDbDataAdapter dAdapt;
        private System.Data.OleDb.OleDbCommandBuilder supBuilder;
        private System.Data.DataSet myDS;
        private bool is_change = false;
        private string strConection;
        public ListUserBaseForm()
        {
            InitializeComponent();

        }
        //проводимо підключення до БД
```

```
        private void ListUserBaseForm_Load(object sender, EventArgs e)
        {
            strConection = "Provider=Microsoft.Jet.OLEDB.4.0;" + @"data source = ";
            //Читаю параметри підключення
            try
            {
                FileStream myFileStream = new FileStream("sql.ini", FileMode.Open,
                        FileAccess.Read);
                BinaryReader binReade = new BinaryReader(myFileStream);
                strConection += binReade.ReadString();
                binReade.Close();
                myFileStream.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка при відкритті файлу, що містить параметри
підключення до БД " + ex.ToString(), "Помилка!");
                return;
            }
            try
            {
                //Підключаємося до БД
                cn = new OleDbConnection();
                cn.ConnectionString = strConection;//"Provider=Microsoft.Jet.OLEDB.4.0;" +
@"data source=Work.mdb"; //strConection;//"Provider=Microsoft.Jet.OLEDB.4.0;" + @"data
source=bread.mdb";
                cn.Open();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка при відкриті бази даних. " + ex.ToString(),
"Помилка!");
                return;
            }
            try
            {
                dAdapt = new OleDbDataAdapter("SELECT * FROM Worker ORDER BY last_name,
name", cn);
                supBuilder = new OleDbCommandBuilder(dAdapt);
                myDS = new DataSet();
                dAdapt.Fill(myDS, "Worker");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка при читані даних із бази. " + ex.ToString(),
"Помилка!");
                return;
            }
            ViewDataGrid();
        }
        //Зберегти зміни
        private void btnSaveEdit_Click(object sender, EventArgs e)
        {
            try
            {
                UserDataGrid.Refresh();
                dAdapt.Update(myDS, "Worker");
                myDS.Clear();
                UserDataGrid.Refresh();
                dAdapt.Fill(myDS, "Worker");
                ViewDataGrid();
                is_change = false;
            }
            catch (Exception ex)
```

```
            {
                MessageBox.Show(ex.ToString(), "Помилка!");
            }
        }
        //Видалити запит
        private void btnDel_Click(object sender, EventArgs e)
        {
            if (is_change)
            {
                MessageBox.Show("Перед видаленям потрібно зберегти зроблені зміни!",
"Збереження внесених змін!");
                return;
            }
            try
            {
                UserDataGrid.Refresh();
                dAdapt.Update(myDS, "Worker");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString(), "Помилка!");
            }
            cn.Close();
            cn.Open();
            string strSQL = "DELETE FROM Worker WHERE id_work=" +
myDS.Tables["Worker"].Rows[UserDataGrid.CurrentRow.Index].ItemArray[0].ToString() + ";";
            System.Data.OleDb.OleDbCommand myComand = new OleDbCommand(strSQL, cn);
            System.Data.OleDb.OleDbDataReader myReader;
            myReader = myComand.ExecuteReader();
            myReader.Close();
            cn.Close();
            myDS.Clear();
            UserDataGrid.Refresh();
            dAdapt.Fill(myDS, "Worker");
            ViewDataGrid();

        }
        //налаштування вигляду для таблиці dataGrid і передача таблиці
        private void ViewDataGrid()
        {
            //передача структури і таблиці у dataGridView1
            UserDataGrid.DataSource = myDS.Tables["Worker"].DefaultView;
            //Налаштування стовпців
            UserDataGrid.Columns[1].HeaderText = "Код";
            UserDataGrid.Columns[1].Width = 40;
            UserDataGrid.Columns[1].Visible = false;
            UserDataGrid.Columns[2].HeaderText = "Прізвище";
            UserDataGrid.Columns[2].Width = 120;
            UserDataGrid.Columns[3].HeaderText = "Ім'я";
            UserDataGrid.Columns[3].Width = 100;
            UserDataGrid.Columns[4].HeaderText = "По батькові";
            UserDataGrid.Columns[4].Width = 120;
            UserDataGrid.Columns[5].HeaderText = "Електронна адреса";
            UserDataGrid.Columns[5].Width = 150;
            UserDataGrid.Columns[6].HeaderText = "Відкритий ключ n";
            UserDataGrid.Columns[6].Width = 120;
            UserDataGrid.Columns[7].HeaderText = "Відкритий ключ e";
            UserDataGrid.Columns[7].Width = 120;
            //Номерація рядків
            for (int i = 0; i < UserDataGrid.RowCount-1; i++)
            {
                UserDataGrid.Rows[i].Cells[0].Value = (i + 1).ToString();
            }
        }
```

```
        //Коли закакінчено редагування стовпця
        private void UserDataGrid_CellEndEdit(object sender, DataGridViewCellEventArgs e)
        {
            //встановлюємо флажок редагування таблиці
            is_change = true;
            //Перевіряємо поля на коретність вводу значень
            int rowIndex = e.RowIndex;
            int cellIndex = e.ColumnIndex;
            //Правильність вводу електроної пошти
            if (cellIndex == 5)
            {
                //Перевіряємо коректність вводу електронної адреси
                if
(!isValid(UserDataGrid.Rows[rowIndex].Cells[cellIndex].Value.ToString()))
                {
                    MessageBox.Show("Електрону пошту вказано не вірно!", "Помилка!");
                    UserDataGrid.Rows[rowIndex].Cells[cellIndex].Value = "";
                }
            }
            if (cellIndex == 6)
            {
                try
                {

Convert.ToInt32(UserDataGrid.Rows[rowIndex].Cells[cellIndex].Value.ToString());
                }
                catch
                {
                    MessageBox.Show("Потрібно вказати цілочисельне числове
значення!","Помилка вводу!");
                    UserDataGrid.Rows[rowIndex].Cells[cellIndex].Value = "";
                }
            }
            if (cellIndex == 7)
            {
                try
                {

Convert.ToInt32(UserDataGrid.Rows[rowIndex].Cells[cellIndex].Value.ToString());
                }
                catch
                {
                    MessageBox.Show("Потрібно вказати цілочисельне числове значення!",
"Помилка вводу!");
                    UserDataGrid.Rows[rowIndex].Cells[cellIndex].Value = "";
                }
            }
        }
        //Функція перевірки вводу email
        private bool isValid(string email)
        {
            string pattern = "[.\\-_a-z0-9]+@([a-z0-9][\\-a-z0-9]+\\.)+[a-z]{2,6}";
            Match isMatch = Regex.Match(email, pattern, RegexOptions.IgnoreCase);
            return isMatch.Success;
        }
        //формую запит для пошуку користувачів
        private void SQLZaput()
        {
            bool is_sql = true; //не вказано умову WHERE
            string strSQL = "SELECT * FROM Worker";
            if (is_sql && txtLastName.Text != "")
            {
                strSQL+=" WHERE last_name LIKE '" + txtLastName.Text + "%'";
                is_sql = false;
```

```
            }
            if (is_sql && txtName.Text != "")
            {
                strSQL += " WHERE name LIKE '" + txtName.Text + "%'";
                is_sql = false;
            }
            if (!is_sql && txtName.Text != "")
                strSQL += " AND name LIKE '" + txtName.Text + "%'";
            if (is_sql && txtPatronymic.Text != "")
            {
                strSQL += " WHERE patronymic LIKE '" + txtPatronymic.Text + "%'";
                is_sql = false;
            }
            if (!is_sql && txtName.Text != "")
                strSQL += " AND patronymic LIKE '" + txtPatronymic.Text + "%'";
            if (is_sql && txtEmail.Text != "")
            {
                strSQL += " WHERE email LIKE '" + txtEmail.Text + "%'";
                is_sql = false;
            }
            if (!is_sql && txtName.Text != "")
                strSQL += " AND email LIKE '" + txtEmail.Text + "%'";
            strSQL += " ORDER BY last_name, name";
            dAdapt = new OleDbDataAdapter(strSQL, cn);
        }
        //Пошук користувачів системи
        private void btnSelectUser_Click(object sender, EventArgs e)
        {
            if (is_change)
            {
                MessageBox.Show("Перед пошуком потрібно зберегти зроблені зміни!",
"Збереження внесених змін!");
                return;
            }
            SQLZaput();
            supBuilder = new OleDbCommandBuilder(dAdapt);
            myDS = new DataSet();
            dAdapt.Fill(myDS, "Worker");
            ViewDataGrid();
        }
        //Метод отправки письма
        private void SendMail(string toAdd, string temaEmail, string textEmail)
        {
            string smtpServer = "";
            string smtpPort = "";
            string email = "";
            string password = "";
            //Читаємо параметри із файлу налаштувань електронної пошти
            try
            {
                FileStream myFileStream = new FileStream("email.ini", FileMode.Open,
                    FileAccess.Read);
                BinaryReader binReade = new BinaryReader(myFileStream);
                smtpServer = CryptionClass.Decrypt(binReade.ReadString(), "34msrti98iew");
                smtpPort = CryptionClass.Decrypt(binReade.ReadString(), "34msrti98iew");
                email = CryptionClass.Decrypt(binReade.ReadString(), "34msrti98iew");
                password = CryptionClass.Decrypt(binReade.ReadString(), "34msrti98iew");
                binReade.Close();
                myFileStream.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка при відкритті файлу, що містить параметри
підключення до email " + ex.ToString(), "Помилка!");
```

```
            }
            //Підключаюся до пошти
            SmtpClient Smtpotpravka = new SmtpClient();
            try
            {
                Smtpotpravka = new SmtpClient(Convert.ToString(smtpServer),
int.Parse(smtpPort)); // Создаем Smtpotpravka типа SmtpClient, подключаем текст с текст-
бокса smtpServer и smtpPort
                Smtpotpravka.Credentials = new NetworkCredential(email, password); //
Авторизируемся для отправки почты
            }
            catch (Exception ex)
            {
                MessageBox.Show("Проблема при підключенні до електронної пошти. " +
ex.Message.ToString());
            }
            if (toAdd == "")
                return;
            //Відпрвка пошти із електронного адресу
            try
            {
                MailMessage Email = new MailMessage(); // Создаем объект Email типа
MailMessage

                Email.From = new MailAddress(email); // Подключаем поле "От кого"
(email.Text) - должен совпадать с ящиком отправителя
                Email.To.Add(new MailAddress(toAdd)); // Подключаем поле "Для кого"
(toAdd.Text), ящик получателя
                Email.Subject = temaEmail; // Подключаем тему для письма (temaEmail.Text)
                Email.Body = textEmail; // Подключаем текст сообщения (textEmail.Text)
                Attachment attachData = new Attachment("Ключ.key");
                Email.Attachments.Add(attachData);
                Smtpotpravka.Send(Email); // Отправляем сообщения
            }
            catch (Exception ex)
            {
                MessageBox.Show("Проблема при відправці листа. " + ex.Message.ToString(),
temaEmail);
            }

        }
        //Кодування тексту шифром RSA
        private void CodeTextEmail(string textEmail, int open_key_n, int open_key_e)
        {
            int n = textEmail.Length; //довжина тексту повідомлення
            RSA myRSA = new RSA();
            short data = 0;
            FileStream myFileStream = null;
            BinaryWriter binWrite = null;
            try
            {
                myFileStream = new FileStream("Ключ.key", FileMode.OpenOrCreate,
FileAccess.Write);
                binWrite = new BinaryWriter(myFileStream);
                for (int i = 0; i < n; i++) //кодую кожну літеру повідомлення
                {
                    data = Convert.ToInt16(textEmail[i]);
                    int z = (int)data;
                    z = (int)myRSA.qe2((long)z, open_key_e, open_key_n);
                    binWrite.Write(z);
                }
            }
            catch
            {
```

```
                Console.WriteLine("Помилка при збережені закодованого ключа!");
            }
            binWrite.Close();
            myFileStream.Close();
        }
        //Розсилка закодованих листів
        private void btnSendMail_Click(object sender, EventArgs e)
        {
            //Перевіряємо кількість записів у DataGridView
            if (UserDataGrid.Rows.Count <= 1)
            {
                MessageBox.Show("Не відібрано жодного користувача!", "Відсутність записів
у таблиці");
                return;
            }
            if (is_change)
            {
                MessageBox.Show("Збережіть зроблені зміни!", "Збереження внесених змін!");
                return;
            }
            TextSendMailForm dlg = new TextSendMailForm();
            dlg.Tema = "";
            dlg.TextList = "";
            if (dlg.ShowDialog() == DialogResult.OK)
            {
                string temaEmail = dlg.Tema;  //Тема
                string textEmail = dlg.TextList;  //Текст листа
                //текст ключа
                string textKey = dlg.TextKey;

                //читаю список записів dataGrid
                for (int i = 0; i < UserDataGrid.RowCount - 1; i++)
                {
                    string toAdd = UserDataGrid.Rows[i].Cells[5].Value.ToString();
//електронна адреса
                    //Ключі
                    int open_key_n =
Convert.ToInt32(UserDataGrid.Rows[i].Cells[6].Value.ToString());
                    int open_key_e =
Convert.ToInt32(UserDataGrid.Rows[i].Cells[7].Value.ToString());
                    CodeTextEmail(textKey, open_key_n, open_key_e);
                    //Відправка закодованого повідомлення на електронну пошту
                    SendMail(toAdd, temaEmail, textEmail); //відпрвка листа
                    //string textDecode = DeCodeTextEmail("ppp", 150524983, open_key_n);
//DeCodeTextEmail(textCode, 150524983, open_key_n);
                    //MessageBox.Show("Розкодований текст " + textDecode);
                }
            }
        }
        //Повідомлення при закриті форми
        private void ListUserBaseForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (is_change)
            {
                MessageBox.Show("Перед закриттям форми потрібно зберегти зроблені зміни!",
"Збереження внесених змін!");
                e.Cancel=true;
            }
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace Cryptogram
{
    public partial class GenarateKeyForm : Form
    {
        RSA RSA_gen_key;
        public GenarateKeyForm()
        {
            InitializeComponent();
            RSA_gen_key = new RSA();
        }
        //генерування ключів
        private void button_gen_key_Click(object sender, EventArgs e)
        {
            int n_my = 0, e_my = 0, d_my = 0;
            RSA_gen_key.GenKey(ref n_my, ref e_my, ref d_my);
            Gen_key_n.Text = n_my.ToString();
            Gen_key_e.Text = e_my.ToString();
            Gen_key_d.Text = d_my.ToString();
        }
        //збереженні ключів
        private void btnSaveKey_Click(object sender, EventArgs e)
        {
            string fileName = txtDirSave.Text + "key.ini";
            FileStream streamWrite = null;    //Файл в який потрібно записувати ключі
            BinaryWriter writer = null;
            bool is_good = true;
            try
            {
                streamWrite = new FileStream(fileName, FileMode.OpenOrCreate,
FileAccess.Write);
                writer = new BinaryWriter(streamWrite);
                writer.Write(Gen_key_e.Text);
                writer.Write(Gen_key_n.Text);
                writer.Write(Gen_key_d.Text);
            }
            catch (IOException expt)
            {
                MessageBox.Show("Помилка при створені файлу!!!\n" + expt.Message,
"Помилка!");
                is_good = false;
            }
            finally
            {
                if (writer != null)
                {
                    writer.Close();
                }
                if (streamWrite != null)
                    streamWrite.Close();
                if (is_good)
                {
                    MessageBox.Show("Ключі успішно збережено.");
                    this.DialogResult = DialogResult.OK;
                }
```

```
            }
            //збергаю шляд до файлу, де зберігаються ключі
            fileName="options.ini";
            streamWrite = null;    //Файл в який потрібно записувати ключі
            writer = null;
            is_good = true;
            try
            {
                streamWrite = new FileStream(fileName, FileMode.OpenOrCreate,
FileAccess.Write);
                writer = new BinaryWriter(streamWrite);
                writer.Write(txtDirSave.Text);
            }
            catch (IOException expt)
            {
                MessageBox.Show("Помилка при створені файлу конфігурації!!!\n" +
expt.Message, "Помилка!");
                is_good = false;
            }
            finally
            {
                if (writer != null)
                {
                    writer.Close();
                }
                if (streamWrite != null)
                    streamWrite.Close();
                if (is_good)
                {
                    //MessageBox.Show("КоКлючі успішно збережено.");
                    this.DialogResult = DialogResult.OK;
                }
            }

        }
        //Відриття каталогу зберігання ключів
        private void btnFindDir_Click(object sender, EventArgs e)
        {
            FolderBrowserDialog folderBrowserDialog1 = new FolderBrowserDialog();
            folderBrowserDialog1.Description = "Виберіть каталог для зберігання ключів";
            if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
            {
                txtDirSave.Text = folderBrowserDialog1.SelectedPath;
                string v = txtDirSave.Text;
                if (txtDirSave.Text[v.Length - 1] != '\\')
                    txtDirSave.Text += @"\";
            }
        }
        //читаю параметри конфігурацій
        private void GenarateKeyForm_Load(object sender, EventArgs e)
        {
            //читаю, де розміщується файл із ключами
            string txtDirFile = "";
            FileStream stream = null;         //Файл, який потрібно читати
            BinaryReader reader = null;
            //Читаю ключі із файлу

            //Читаю шлях до файлу конфігурації
            try
            {
                stream = new FileStream("options.ini", FileMode.Open, FileAccess.Read);
                reader = new BinaryReader(stream);
                txtDirFile = reader.ReadString();
            }
```

```
            catch(Exception expt)
            {
                MessageBox.Show("Помилка при читані конфігураційного файлу!!!\n" +
expt.Message, "Помикла!");
            }
            finally
            {
                if (reader != null)
                {
                    reader.Close();
                }
                if (stream != null)
                    stream.Close();
            }
            txtDirFile += "key.ini";
            //якщо шлях прочитано, тоді починаю читати ключі
            stream = null;         //Файл, який потрібно читати
            reader = null;
            try
            {
                stream = new FileStream(txtDirFile, FileMode.Open, FileAccess.Read);
                reader = new BinaryReader(stream);
                Gen_key_e.Text = reader.ReadString();
                Gen_key_n.Text = reader.ReadString();
                Gen_key_d.Text = reader.ReadString();
            }
            catch (IOException expt)
            {
                MessageBox.Show("Помилка при читані файлу, що містить ключ!!!\n" +
expt.Message, "Помикла!");
            }
            finally
            {
                if (reader != null)
                {
                    reader.Close();
                }
                if (stream != null)
                    stream.Close();
            }
        }
    }
}



using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace Cryptogram
{
    public partial class Gost_Code : Form
    {
        public Gost_Code()
        {
            InitializeComponent();
        }
        public string FileCode
```

```
{
    get { return txtFileCode.Text; }
    //Эта функция позволяет настроить исходное значение
    //для текстовго поля в диалговом окне
    set
    {
        txtFileCode.Text = value;
    }
}
public string FileDecode
{
    get { return txtFileDecode.Text; }
    set
    {
        txtFileDecode.Text = value;
    }
}
public string Key
{
    get { return txtKey.Text; }
    set { txtKey.Text = value; }
}
private void btmGostCode_Click(object sender, EventArgs e)
{
    //Настраиваем свойства диалогового окна для открытия файлов
    OpenFileDialog myOpenFileDialog = new OpenFileDialog();
    myOpenFileDialog.InitialDirectory = "*.*";
    myOpenFileDialog.Filter = "Усі файли(*.*)|*.*";
    myOpenFileDialog.FilterIndex = 1;
    myOpenFileDialog.RestoreDirectory = true;
    if ((myOpenFileDialog.ShowDialog()) == DialogResult.OK)
    {
        Stream myStream = null;
        if ((myStream = myOpenFileDialog.OpenFile()) != null)
        {
            txtFileCode.Text = myOpenFileDialog.FileName.ToString();
            myStream.Close();
        }
        else
            MessageBox.Show("Помилка при відкриті файлу", "Error");
    }
}
private void btnGostDecode_Click(object sender, EventArgs e)
{
    //Настраиваем свойства диалогового окна для открытия файлов
    SaveFileDialog myOpenFileDialog = new SaveFileDialog();
    myOpenFileDialog.InitialDirectory = "*.*";
    myOpenFileDialog.Filter = "Усі файли(*.*)|*.*";
    myOpenFileDialog.FilterIndex = 1;
    myOpenFileDialog.RestoreDirectory = true;
    if ((myOpenFileDialog.ShowDialog()) == DialogResult.OK)
    {
        Stream myStream = null;
        if ((myStream = myOpenFileDialog.OpenFile()) != null)
        {
            txtFileDecode.Text = myOpenFileDialog.FileName.ToString();
            myStream.Close();
        }
        else
            MessageBox.Show("Помилка при відкриті файлу", "Error");
    }
}
private void btnGostKey_Click(object sender, EventArgs e)
{
```

```
            //Настраиваем свойства диалогового окна для открытия файлов
            OpenFileDialog myOpenFileDialog = new OpenFileDialog();
            myOpenFileDialog.InitialDirectory = "*.*";
            myOpenFileDialog.Filter = "Усі файли(*.*)|*.*";
            myOpenFileDialog.FilterIndex = 1;
            myOpenFileDialog.RestoreDirectory = true;
            if ((myOpenFileDialog.ShowDialog()) == DialogResult.OK)
            {
                Stream myStream = null;
                if ((myStream = myOpenFileDialog.OpenFile()) != null)
                {
                    txtKey.Text = myOpenFileDialog.FileName.ToString();
                    myStream.Close();
                }
                else
                    MessageBox.Show("Помилка при відкриті файлу", "Error");
            }
        }

        private void btnCancel_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Ірина Василівна Українець

Науковий керівник Р.М.Літнарович

АНАЛІЗ І ДОСЛІДЖЕННЯ КРИПТОГРАФІЧНИХ ЗАСОБІВ ЗАХИСТУ ІНФОРМАЦІЇ НА БАЗІ «УКРГАЗБАНК»

ФАКУЛЬТЕТ КІБЕРНЕТИКИ

КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ

Комп'ютерний набір, в редакторі Microsoft ® Offise ® Word 2003 І.В.Українець.

Редагування, верстка, макетування та дизайн Р.М.Літнарович

Відповідальний редактор Й.В. Джунь

Підп.до друку 11. 12. 2010 р.

Формат 60x84/16. Папір офсетн.№1.

Гарнітура Times New Roman.

Друк різограф. Тираж 300 пр.

Редакційно-видавничий центр «Тетіс»

Міжнародного економіко-гуманітарного університету

Імені академіка Степана Дем'янчука

33027 Рівне , Україна

Вул..С.Дем'янчука, 4, корпус 1

Телефон : (+00380) 362 23 – 73 – 09

Факс :(+00380) 362 23 – 01 – 86

E-mail:mail@regi.rovno.ua