Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**ЗВІТ**

**Про виконання лабораторної роботи № 3**

**з дисципліни «Безпека програм та даних»**

**«Створення програмного засобу для забезпечення конфіденційності інформації»**

**Лектор:**

доц. кафедри ПЗ

Сенів М. М.

**Виконав:**

студ. групи ПІ-41

Павленчик М. М.

**Прийняв:**

доц. кафедри ПЗ

Сенів М. М.

«\_\_\_» \_\_\_\_\_\_ 2018 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2018

**ТЕМА РОБОТИ**: Створення програмного засобу для забезпечення конфіденційності інформації.

**МЕТА РОБОТИ**: Ознайомитись з методами криптографічного забезпечення конфіденційності інформації, навчитись створювати комплексні програмні продукти для захисту інформації з використанням алгоритмів симетричного шифрування, хешування та генераторів псевдовипадкових чисел.

### Теоретичні відомості

RC5 – це алгоритм симетричного шифрування, розроблений Роном Райвестом в середині 90-х років. При розробці RC5 ставилась задача досягнення наступних характеристик:

* Придатність для апаратної та програмної реалізації. В RC5 використовуються тільки елементарні обчислювальні операції, які зазвичай застосовуються в мікропроцесорах.
* Швидкість виконання. RC5 є простим алгоритмом, що працює з даними розміром в машинне слово. Усі основні операції передбачають також роботу з даними довжиною в слово.
* Адаптованість до процесорів з різною довжиною слова. Довжина слова в бітах є параметром RC5 – при зміні довжини слова змінюється сам алгоритм.
* Змінна кількість раундів. Кількість раундів є другим параметром RC5. Цей параметр дозволяє вибрати оптимальне співвідношення необхідної швидкості роботи і вимог до ступеня захисту.
* Змінна довжина ключа. Довжина ключа є третім параметром RC5. Як і в попередньому випадку, цей параметр дозволяє знайти прийнятний компроміс між швидкістю роботи та необхідним рівнем безпеки.
* Простота. Структура RC5 дуже проста не тільки для реалізації, але й для оцінки її криптоаналітичної стійкості.
* Низькі вимоги до пам'яті. Низькі вимоги до пам'яті роблять RC5 придатним для використання в смарт-картах та інших подібних пристроях з обмеженим об'ємом пам'яті.
* Високий ступінь захисту. RC5 покликаний забезпечити високий ступінь захисту за умови вибору відповідних значень параметрів.
* Залежність циклічних зсувів від даних. В RC5 використовуються циклічні зсуви, величина яких залежить від даних, що повинно підвищувати криптоаналітичну стійкість алгоритму.

RC5 фактично являє собою родину алгоритмів шифрування, що визначається трьома наступними параметрами: Розмір слова в біта, Кількість раундів, Кількість байтів в таємному ключі К. Таким чином, RC5 шифрує блоки відкритого тексту довжиною 32, 64 чи 128 бітів в блоки шифрованого тексту тієї самої довжини. Довжина ключа може змінюватись від 0 до 2040 бітів. Конкретна версія RC5 позначається RC5-w/r/b. Наприклад, RC5-32/12/16 використовує 32-бітові слова (64-бітові блоки відкритого і шифрованого тексту), 12 раундів шифрування і ключ довжиною 16 байтів (128 бітів). Райвест пропонує використовувати RC5-32/12/16 в якості "стандартної" версії RC5.

**Завдання для лабораторної роботи**

Згідно до варіанту, наведеного в таблиці, створити прикладну програму для шифрування інформації за алгоритмом RC5. Програма повинна отримувати від користувача парольну фразу і, на її основі, шифрувати файли довільного розміру, а результат зберігати у вигляді файлу з можливістю подальшого дешифрування (при введенні тієї самої парольної фрази). Для перетворення парольної фрази у ключ шифрування використати алгоритм MD5, реалізований в лабораторній роботі № 2 – ключем шифрування повинен бути хеш парольної фрази. Якщо згідно варіанту довжина ключа становить 64 біти, беруться молодші 64 біти хешу; якщо довжина ключа повинна бути 256 бітів, то хеш парольної фрази стає старшими 128 бітами, а молодшими є хеш від старших 128 бітів (тобто, позначивши парольну фразу через P, отримаємо K=H(H(P))||H(P)). Для забезпечення можливості роботи створеного програмного продукту з відкритим текстом довільної довжини, програмну реалізацію здійснити в режимі RC5-CBC-Pad. В якості вектора ініціалізації (IV) використати генератор псевдовипадкових чисел, реалізований в лабораторній роботі № 1. Для кожного нового шифрованого повідомлення слід генерувати новий вектор ініціалізації. Вектор ініціалізації зашифровується в режимі ECB і зберігається в першому блоці зашифрованого файлу. У звіті навести протокол роботи програми та зробити висновки про поєднання різних криптографічних примітивів для задач захисту інформації.

### Результати

Протокол роботи програми

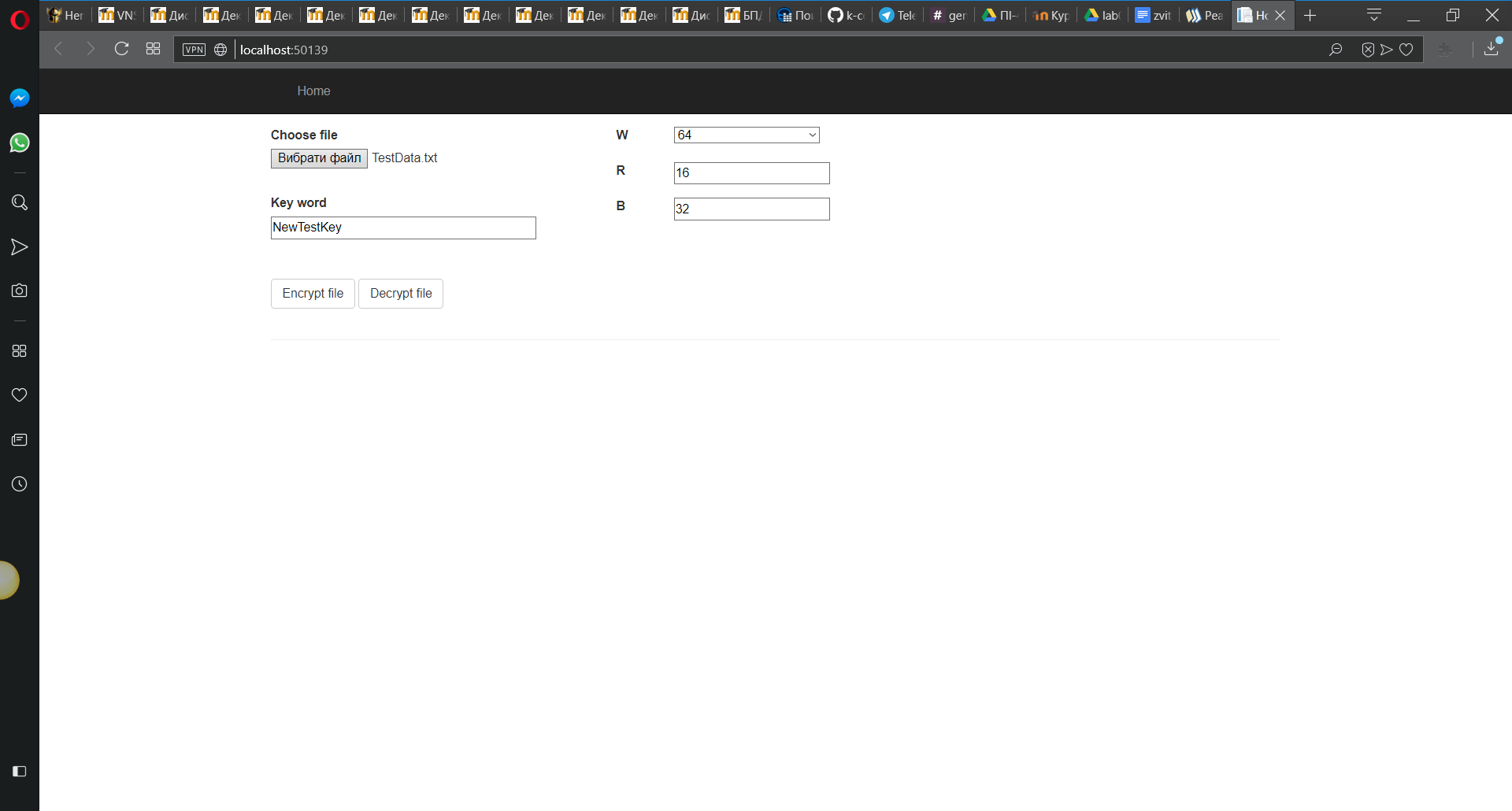


Рис. 1. Протокол роботи застосунку

Вихідний код програми

public class RC5Service

{

private static RandomNumberGenerator RandomNumberGenerator = new RandomNumberGenerator();

private HashService HashService { get; set; } = new HashService();

public EncryptingResultModel Encrypt(byte[] input, byte[] keyInput, int w, int r, int b)

{

var key = GetKey(keyInput, b);

for (int i = 0; i < keyInput.Length; i++)

{

keyInput[i] = 0;

}

var helper = new RC5Helper(w, r, key);

if (input.Length % helper.BlockSize != 0)

{

input = GetFullInput(input, helper.BlockSize);

}

var IV = GetInitializeVector(helper.BlockSize);

var resultData = new byte[input.Length];

var result = new EncryptingResultModel { IV = EncryptInitializeVector(IV, key, w, r) };

for (int i = 0; i < input.Length; i += helper.BlockSize)

{

var currentBlock = new byte[helper.BlockSize];

Array.Copy(input, i, currentBlock, 0, helper.BlockSize);

currentBlock = XOR(IV, currentBlock, helper.W);

IV = EncryptBlock(currentBlock, helper);

Array.Copy(IV, 0, resultData, i, helper.BlockSize);

}

result.EncryptedData = resultData;

return result;

}

public byte[] Decrypt(byte[] startupInput, byte[] keyInput, int w, int r, int b)

{

var key = GetKey(keyInput, b);

for (int i = 0; i < keyInput.Length; i++)

{

keyInput[i] = 0;

}

var helper = new RC5Helper(w, r, key);

var input = GetEncryptingResultModel(startupInput, helper.BlockSize);

var result = new byte[input.EncryptedData.Length];

var IV = DecryptInitializeVector(input.IV, key, w, r);

for (int i = 0; i < input.EncryptedData.Length; i += helper.BlockSize)

{

var currentBlock = new byte[helper.BlockSize];

Array.Copy(input.EncryptedData, i, currentBlock, 0, helper.BlockSize);

var decryptedBlock = DecryptBlock(currentBlock, helper);

decryptedBlock = XOR(IV, decryptedBlock, helper.W);

Array.Copy(decryptedBlock, 0, result, i, helper.BlockSize);

IV = currentBlock;

}

return result;

}

private byte[] GetInitializeVector(int blockSize)

{

switch (blockSize)

{

case 4: return BitConverter.GetBytes(RandomNumberGenerator.GetNextNumber()).Take(4).ToArray();

case 8: return BitConverter.GetBytes(RandomNumberGenerator.GetNextNumber());

case 16:

{

var part1 = BitConverter.GetBytes(RandomNumberGenerator.GetNextNumber());

var part2 = BitConverter.GetBytes(RandomNumberGenerator.GetNextNumber());

return part1.Concat(part2).ToArray();

}

default: throw new Exception("Incorrect block size.");

}

}

private byte[] EncryptInitializeVector(byte[] input, byte[] key, int w, int r)

{

var helper = new RC5Helper(w, r, key);

if (input.Length % helper.BlockSize != 0)

{

input = GetFullInput(input, helper.BlockSize);

}

var result = new byte[input.Length];

for (int i = 0; i < input.Length; i += helper.BlockSize)

{

var currentBlock = new byte[helper.BlockSize];

Array.Copy(input, i, currentBlock, 0, helper.BlockSize);

currentBlock = EncryptBlock(currentBlock, helper);

Array.Copy(currentBlock, 0, result, i, helper.BlockSize);

}

return result;

}

private byte[] DecryptInitializeVector(byte[] input, byte[] key, int w, int r)

{

var helper = new RC5Helper(w, r, key);

var result = new byte[input.Length];

for (int i = 0; i < input.Length; i += helper.BlockSize)

{

var currentBlock = new byte[helper.BlockSize];

Array.Copy(input, i, currentBlock, 0, helper.BlockSize);

currentBlock = DecryptBlock(currentBlock, helper);

Array.Copy(currentBlock, 0, result, i, helper.BlockSize);

}

return result;

}

private byte[] GetKey(byte[] keyInput, int b)

{

var keyHash = HashService.GetHashInBytes(keyInput);

if (b < 16)

{

var result = new byte[8];

Array.Copy(keyHash, 0, result, 0, result.Length);

return result;

}

else if (b < 32)

{

return keyHash;

}

else

{

var addingHash = HashService.GetHashInBytes(keyHash);

var result = new byte[32];

Array.Copy(keyHash, 0, result, 0, keyHash.Length);

Array.Copy(addingHash, 0, result, keyHash.Length, addingHash.Length);

return result;

}

}

private byte[] GetFullInput(byte[] input, int blockSize)

{

var fullInputSize = (input.Length / blockSize + 1) \* blockSize;

var appendSize = fullInputSize - input.Length;

var append = new byte[appendSize];

for (int i = 0; i < appendSize; i++)

{

append[i] = (byte) appendSize;

}

var fullInput = new byte[fullInputSize];

Array.Copy(input, 0, fullInput, 0, input.Length);

Array.Copy(append, 0, fullInput, input.Length, append.Length);

return fullInput;

}

private byte[] EncryptBlock(byte[] block, RC5Helper helper)

{

var A = GetBlockPart(block, true, helper.W);

var B = GetBlockPart(block, false, helper.W);

A += helper.S[0];

B += helper.S[1];

for (int i = 1; i <= helper.R; i++)

{

A = helper.RotateLeft((A ^ B), (int)B, helper.W \* 8) + helper.S[2 \* i];

B = helper.RotateLeft((B ^ A), (int)A, helper.W \* 8) + helper.S[2 \* i + 1];

}

var result = new byte[2 \* helper.W];

var partA = BitConverter.GetBytes(A);

var partB = BitConverter.GetBytes(B);

Array.Copy(partA, 0, result, 0, helper.W);

Array.Copy(partB, 0, result, helper.W, helper.W);

return result;

}

private byte[] DecryptBlock(byte[] block, RC5Helper helper)

{

var A = GetBlockPart(block, true, helper.W);

var B = GetBlockPart(block, false, helper.W);

for (int i = helper.R; i > 0; i--)

{

B = helper.RotateRight(B - helper.S[2 \* i + 1], (int)A, helper.W \* 8) ^ A;

A = helper.RotateRight(A - helper.S[2 \* i], (int)B, helper.W \* 8) ^ B;

}

A -= helper.S[0];

B -= helper.S[1];

var result = new byte[2 \* helper.W];

var partA = BitConverter.GetBytes(A);

var partB = BitConverter.GetBytes(B);

Array.Copy(partA, 0, result, 0, helper.W);

Array.Copy(partB, 0, result, helper.W, helper.W);

return result;

}

private ulong GetBlockPart(byte[] block, bool isFirst, int w)

{

if (w < 8)

{

var tmp = new byte[8];

Array.Copy(block, isFirst ? 0 : w, tmp, 0, w);

return BitConverter.ToUInt64(tmp, 0);

}

else

{

return BitConverter.ToUInt64(block, isFirst ? 0 : w);

}

}

private byte[] XOR(byte[] arr1, byte[] arr2, int w)

{

var A1 = GetBlockPart(arr1, true, w);

var B1 = GetBlockPart(arr1, false, w);

var A2 = GetBlockPart(arr2, true, w);

var B2 = GetBlockPart(arr2, false, w);

return BitConverter.GetBytes(A1 ^ A2).Concat(BitConverter.GetBytes(B1 ^ B2)).ToArray();

}

private EncryptingResultModel GetEncryptingResultModel(byte[] input, int blockSize)

{

var result = new EncryptingResultModel();

result.IV = new byte[blockSize];

result.EncryptedData = new byte[input.Length - blockSize];

Array.Copy(input, 0, result.IV, 0, blockSize);

Array.Copy(input, blockSize, result.EncryptedData, 0, result.EncryptedData.Length);

return result;

}

}

**Висновки**

На даній лабораторній роботі я ознайомився з методами криптографічного забезпечення конфіденційності інформації, навчився створювати комплексні програмні продукти для захисту інформації з використанням алгоритмів симетричного шифрування, хешування та генераторів псевдовипадкових чисел.