**ЛЕКЦИЯ. ШИФРОВАНИЕ ФАЙЛОВ**

Для шифрования нужно получить криптопровайдер. Криптопровайдер – это объект, который обеспечивает методы шифрования и дешифрования. Мы получаем его таким образом:

if (!CryptAcquireContext(&hProv, NULL, NULL,

PROV\_RSA\_FULL, CRYPT\_VERIFYCONTEXT))

Здесь первый аргумент получает ссылку на криптопровайдер. Указывается, что он обеспечивает шифрование по методу RSA. Детали этого метода шифрования мы приводим в конце этой лекции. Для шифрования нужно получить ключ. Ключ генерирует система таким образом:

CryptGenKey(hProv, CALG\_RC4,

CRYPT\_EXPORTABLE, &hSessionKey);

Ключ определяется в адресной переменной hSessionKey.

Шифрование выполняется по команде

if (!CryptEncrypt(hSessionKey, 0, true, 0, (BYTE\*)string,

&count, strlen(string)))

Шифруется строка string с помощью подготовленного ключа. Количество шифруемых байтов задается переменной count.

Программа расшифровки вызывается таким образом

if(CryptDecrypt(

hSessionKey,

0,

TRUE,

0,

pbBuffer,

&count))

Расшифрованная строка помещается в буфер pbBuffer.

Полный текст программы имеет следующий вид:

// CRYPTO.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <stdio.h>

#include <windows.h>

#include <wincrypt.h>

#include <cryptuiapi.h>

#include <iostream>

#include <tchar.h>

#define KEYLENGTH 0x00800000

#define ENCRYPT\_BLOCK\_SIZE 8

#pragma comment (lib, "crypt32.lib")

#pragma comment (lib, "cryptui.lib")

int \_tmain(int argc, \_TCHAR\* argv[])

{

HCRYPTPROV hProv;

HCRYPTKEY hSessionKey;

// Получение контекста криптопровайдера

if (!CryptAcquireContext(&hProv, NULL, NULL,

PROV\_RSA\_FULL, CRYPT\_VERIFYCONTEXT))

{

std::cout << "Error in Crypto1" << std::endl;

return 1;

}

std::cout << "Cryptographic provider initialized" << std::endl;

// Генерация сессионного ключа

CryptGenKey(hProv, CALG\_RC4,

CRYPT\_EXPORTABLE, &hSessionKey);

//if (!CryptGenKey(hProv, CALG\_RC4,

// CRYPT\_ENCRYPT | CRYPT\_DECRYPT, &hSessionKey))

//{

// Error("CryptGenKey");

// return;

//}

std::cout << "Session key generated" << std::endl;

// Данные для шифрования

char string[]="Test";

DWORD count=strlen(string);

DWORD delen = strlen(string);

PBYTE pbBuffer=NULL;

DWORD dwBufferLen = count + ENCRYPT\_BLOCK\_SIZE - (count % ENCRYPT\_BLOCK\_SIZE);

// Allocate memory

pbBuffer = (BYTE \*)malloc(dwBufferLen);

// Шифрование данных

if (!CryptEncrypt(hSessionKey, 0, true, 0, (BYTE\*)string,

&count, strlen(string)))

{

std::cout << "Error in Crypto2" << std::endl;

return 1;

}

std::cout << "Encryption completed" << std::endl;

// Тестовый вывод на экран

std::cout << "Encrypted string: " << string << std::endl;

////+++++++++++++

memcpy(pbBuffer, string, delen);

if(CryptDecrypt(

hSessionKey,

0,

TRUE,

0,

pbBuffer,

&count))

{

printf("After decryption: %s\n", pbBuffer);

//std::cout << "\nDecrypted text = ";

//std::cout<<pbBuffer<<std::endl;

}

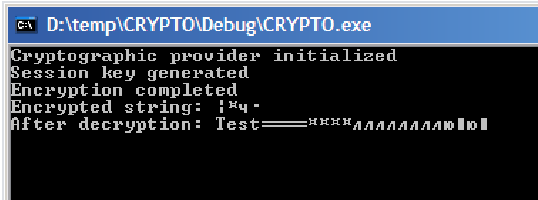
//------

getchar();

return 0;

}

Скриншот такой:



Важным механизмом шифрования является хэширование. Особенную роль хеширование играет в системе защиты данных компьютера. Имеются разные алгоритмы хэширования. Ниже приведен код для реализации алгоритма MD5. Сначала нужно получить криптопровайдер:

if (!CryptAcquireContext(&hProv,

NULL,

NULL,

PROV\_RSA\_FULL,

CRYPT\_VERIFYCONTEXT))

{

dwStatus = GetLastError();

printf("CryptAcquireContext failed: %d\n", dwStatus);

CloseHandle(hFile);

return dwStatus;

}

Объект криптопровайдера есть hProv. Ниже по тексту получаем объект hHash, выполняющий хеширование:

if (!CryptCreateHash(hProv, CALG\_MD5, 0, 0, &hHash))

Собственно хеширование реализуется в методе

if (!CryptHashData(hHash, rgbFile, cbRead, 0))

Результат хеширования помещается в файл rgbFileю

#include <stdio.h>

#include <windows.h>

#include <Wincrypt.h>

#define BUFSIZE 1024

#define MD5LEN 16

DWORD main()

{

DWORD dwStatus = 0;

BOOL bResult = FALSE;

HCRYPTPROV hProv = 0;

HCRYPTHASH hHash = 0;

HANDLE hFile = NULL;

BYTE rgbFile[BUFSIZE];

DWORD cbRead = 0;

BYTE rgbHash[MD5LEN];

DWORD cbHash = 0;

CHAR rgbDigits[] = "0123456789abcdef";

LPCWSTR filename=L"filename.txt";

// Logic to check usage goes here.

hFile = CreateFile(filename,

GENERIC\_READ,

FILE\_SHARE\_READ,

NULL,

OPEN\_EXISTING,

FILE\_FLAG\_SEQUENTIAL\_SCAN,

NULL);

if (INVALID\_HANDLE\_VALUE == hFile)

{

dwStatus = GetLastError();

printf("Error opening file %s\nError: %d\n", filename,

dwStatus);

return dwStatus;

}

// Get handle to the crypto provider

if (!CryptAcquireContext(&hProv,

NULL,

NULL,

PROV\_RSA\_FULL,

CRYPT\_VERIFYCONTEXT))

{

dwStatus = GetLastError();

printf("CryptAcquireContext failed: %d\n", dwStatus);

CloseHandle(hFile);

return dwStatus;

}

if (!CryptCreateHash(hProv, CALG\_MD5, 0, 0, &hHash))

{

dwStatus = GetLastError();

printf("CryptAcquireContext failed: %d\n", dwStatus);

CloseHandle(hFile);

CryptReleaseContext(hProv, 0);

return dwStatus;

}

while (bResult = ReadFile(hFile, rgbFile, BUFSIZE,

&cbRead, NULL))

{

if (0 == cbRead)

{

break;

}

if (!CryptHashData(hHash, rgbFile, cbRead, 0))

{

dwStatus = GetLastError();

printf("CryptHashData failed: %d\n", dwStatus);

CryptReleaseContext(hProv, 0);

CryptDestroyHash(hHash);

CloseHandle(hFile);

return dwStatus;

}

}

if (!bResult)

{

dwStatus = GetLastError();

printf("ReadFile failed: %d\n", dwStatus);

CryptReleaseContext(hProv, 0);

CryptDestroyHash(hHash);

CloseHandle(hFile);

return dwStatus;

}

cbHash = MD5LEN;

if (CryptGetHashParam(hHash, HP\_HASHVAL, rgbHash, &cbHash, 0))

{

printf("MD5 hash of file %s is: ", filename);

for (DWORD i = 0; i < cbHash; i++)

{

printf("%c%c", rgbDigits[rgbHash[i] >> 4],

rgbDigits[rgbHash[i] & 0xf]);

}

printf("\n");

}

else

{

dwStatus = GetLastError();

printf("CryptGetHashParam failed: %d\n", dwStatus);

}

CryptDestroyHash(hHash);

CryptReleaseContext(hProv, 0);

CloseHandle(hFile);

return dwStatus;

}

Алгоритм RSA

Пусть необходимо передать по линии связи числа *x* (рассмотрим здесь только целые положительные числа). Вместо числа *x* передают число *y*, вычисляемое по формуле

|  |  |
| --- | --- |
| , | (1) |

где *e* и *m* являются открытыми числами (известны всем абонентам сети).

Требуем, чтобы *e* и *m* были взаимно простыми числами (т.е. не числами общих делителей, кроме 1, причем ).

Оказывается, что зная *y*, *e* и *m*, найти *x* – сложнейшая математическая задача[[1]](#footnote-2). Пока же продемонстрируем, как найти *y* по *x*, *e*, *m*.

Операция

|  |  |
| --- | --- |
|  | (2) |

находит целочисленный остаток *a* от деления *b* на *m*. Например,

2 = 17 mod 5

или

1 = 41 mod 8.

Но пусть требуется найти

630 mod 18 = ?

Это сделать посложнее. Мно записать

630 = 2\*315 = 2\*5\*63 = 2\*5\*7\*9 = 63\*10.

Теперь можно использовать правило разложения на множители

.

В самом деле, пусть

,

,

.

Тогда

.

Последняя сумма дает остаток от деления на *m*, равный . Но , . Поэтому . Теперь нетрудно это правило применить, скажем, к

713 mod 8 = ?

Запишем . Имеем . Поэтому .

Обратимся теперь к формуле (2.1).

Пусть , , .

Найдем

.

Итак, . Это значение и будет передано по сети вместо *x*.

Теперь рассмотрим, как восстановить *x* по *y*, *m*, *e*. Для этой цели нужно найти число *d*, удовлетворяющее условию

|  |  |
| --- | --- |
| , | (3) |

где  – значение функции Эйлера от числа *m*. Функция Эйлера вычисляется сравнительно просто. Так,

|  |  |
| --- | --- |
| . | (4) |

Если *p* простое число и *r* – целое, то

|  |  |
| --- | --- |
| . | (5) |

Формул (2.4) и (2.5) достаточно для того, чтобы найти функцию Эйлера для любого целого положительного числа. В нашем случае получаем:

.

Для любознательных читателей отметим, что значение  равно числу целых чисел на отрезке 1..*m*, взаимно простых с *m*. Отыскание значения функции Эйлера для больших целых чисел является вычислительной задачей очень большой сложности.

Пример. . Все четыре числа: 1, 2, 3, 4 взаимно просты с *m*.

Теперь обратимся к уравнению (2.3). В этом уравнении *d* играет роль секретного ключа. Решить уравнение (2.3) путем перебора значений *d* можно, но если в числе *m*, например, 100 цифр, то на вычисление *d* уйдет достаточно много времени. Для небольших значений, таких как в нашем примере, можно воспользоваться алгоритмом решения уравнений в целых числах, который мы и приведем.

Итак, в нашем примере уравнение такое:

|  |  |
| --- | --- |
| . | (.6) |

Уравнение (2.6) можно переписать следующим известным образом:

|  |  |
| --- | --- |
| . | (7) |

В (7) r и d неизвестные целые числа. Представим (7) в виде системы двух линейных неравенств.

,

,

или, что эквивалентно:

|  |  |
| --- | --- |
| ,  . | (8) |

В неравенстве с положительной правой частью выделим член с минимальным по модулю коэффициентом и разрешим неравенство относительно этого члена:

,

.

Отсюда легко получить отсекающее неравенство:

|  |  |
| --- | --- |
| (a) ,  (b) ,  (c) . | (9) |

Здесь *z* – новая целочисленная переменная. Заметим, что переход от (a) к (b) в (9) правомерен, так как *r* , *d*, *z* – целочислены.

Выполним подстановку (2.9) в систему (2.8). Получим новую систему:

|  |  |
| --- | --- |
| ,  . | (10) |

Обратим внимание на следующее принципиальное обстоятельство. В сравнении с (2.8) значение минимального коэффициента понизилось. Этот факт можно строго обосновать. Следовательно, весь процесс должен закончиться рано или поздно одним из двух результатов:

* 1. минимальный коэффициент по модулю станет равным 1 (как в (2.10));
  2. будет получена система вида

,

,

где *a* и *b* – взаимно просты (в этом случае нет решения в целых числах).

В первом случае процесс решения завершен. Получаем из (10) подстановку для *d:*

|  |  |
| --- | --- |
| . | (11) |

Тогда из (9) найдем:

|  |  |
| --- | --- |
| . | (12) |

Итак, формулы (11) и (12) и дают нам итоговые подстановки для *d* и *r* из (7). Например, пусть . Тогда , . Возьмем именно это значение для минимального *d*.

Итак, мы подошли к решающему моменту: наш секретный ключ . Получили число , , .Восстанавливаем *x* по формуле:

|  |  |
| --- | --- |
| . | (13) |

Итак, .

Все сошлось.

1. [↑](#footnote-ref-2)