Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа №3**

по дисциплине: «Информационная безопасность и защита информации».

Выполнил:

студент 4 курса, гр. ИВТАПбд-41

Кондратьев Павел Сергеевич.

Проверил:

преподаватель кафедры ВТ

Мартынов Антон Иванович.

г. Ульяновск, 2019

**1) Задание:**

Реализовать приложение с графическим интерфейсом, позволяющее выполнять следующие действия:

1. Шифровать и дешифровать текстовые и двоичные файлы с помощью потокового шифрования и генератора псевдослучайных чисел, разработанного в предыдущей лабораторной работе
2. Сохранять зашифрованные /де шифрованные данные в файл
3. Хешировать текстовый пароль, который используется при шифровании для инициализации генератора псевдослучайных чисел с помощью функции хеширования, указанной в варианте

**2) Краткие теоретические сведения:**

**SHA-1 (Secure Hash Algorithm 1)**

**Secure Hash Algorithm 1** — алгоритм криптографического хеширования. Описан в [RFC 3174](https://tools.ietf.org/html/rfc3174). Для входного сообщения произвольной длины (максимум 264−1{\displaystyle 2^{64}-1} бит, что равно 2 эксабайта алгоритм генерирует 160-битное хеш-значение, называемое также дайджестом сообщения. Используется во многих криптографических приложениях и протоколах. Также рекомендован в качестве основного для государственных учреждений в США. Принципы, положенные в основу SHA-1, аналогичны тем, которые использовались Рональдом Ривестом при проектировании MD4.

**Инициализация**

Исходное сообщение разбивается на блоки по 512 бит в каждом. Последний блок дополняется до длины, кратной 512 бит. Сначала добавляется 1 а потом нули, чтобы длина блока стала равной (512 - 64 = 448) бит. В оставшиеся 64 бита записывается длина исходного сообщения в битах. Если последний блок имеет длину более 448, но менее 512 бит, дополнение выполняется следующим образом: сначала добавляется 1, затем нули вплоть до конца 512-битного блока; после этого создается ещё один 512-битный блок, который заполняется вплоть до 448 бит нулями, после чего в оставшиеся 64 бита записывается длина исходного сообщения в битах. Дополнение последнего блока осуществляется всегда, даже если сообщение уже имеет нужную длину.

Инициализируются пять 32-битовых переменных.

A = a = 0x67452301

B = b = 0xEFCDAB89

C = c = 0x98BADCFE

D = d = 0x10325476

E = e = 0xC3D2E1F0

Определяются четыре нелинейные операции и четыре константы.

|  |  |  |
| --- | --- | --- |
| Ft(m,l,k)=(m∧l)∨(¬m∧k){\displaystyle F\_t(m, l, k) = (m \wedge l) \vee (\neg{m} \wedge k)} | Kt{\displaystyle K\_t} = 0x5A827999 | 0≤t≤19 |
| Ft(m,l,k)=m⊕l⊕k{\displaystyle F\_t(m, l, k) = m \oplus l \oplus k } | Kt{\displaystyle K\_t} = 0x6ED9EBA1 | 20≤t≤39 |
| Ft(m,l,k)=(m∧l)∨(m∧k)∨(l∧k){\displaystyle F\_t(m, l, k) = (m \wedge l) \vee (m \wedge k) \vee (l \wedge k)} | Kt{\displaystyle K\_t} = 0x8F1BBCDC | 40≤t≤59 |
| Ft(m,l,k)=m⊕l⊕k{\displaystyle F\_t(m, l, k) = m \oplus l \oplus k } | Kt{\displaystyle K\_t} = 0xCA62C1D6 | 60≤t≤79 |

**3) Порядок выполнения работы:**

1. Программа должна быть оформлена в виде удобной утилиты с интерактивным интерфейсом пользователя
2. Текст программы оформляется прилично (удобочитаемо, с описанием ВСЕХ функций, переменных и критических мест).
3. В процессе работы программа ОБЯЗАТЕЛЬНО выдает информацию о состоянии процесса генерации / тестирования (если процесс занимает длительное время)
4. Интерфейс программы может быть произвольным, но удобным и понятным (разрешается использование библиотек GUI)
5. Среда разработки и язык программирования могут быть произвольными

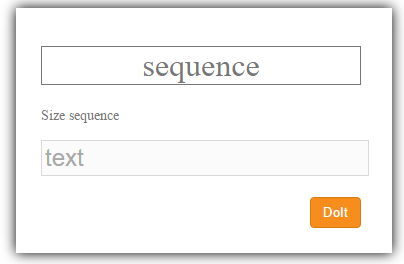


Рис1. Форма для задания начального текса



Рис2. Страница для ввода пароля (сравнимая с паролем на сайте с помощью хеширования SHA-1)

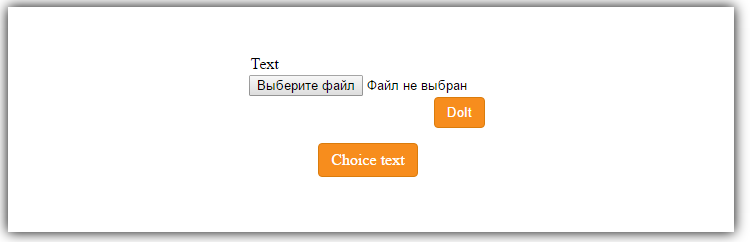


Рис3. Результат программы после ввода пароля (выбор текстового файла)

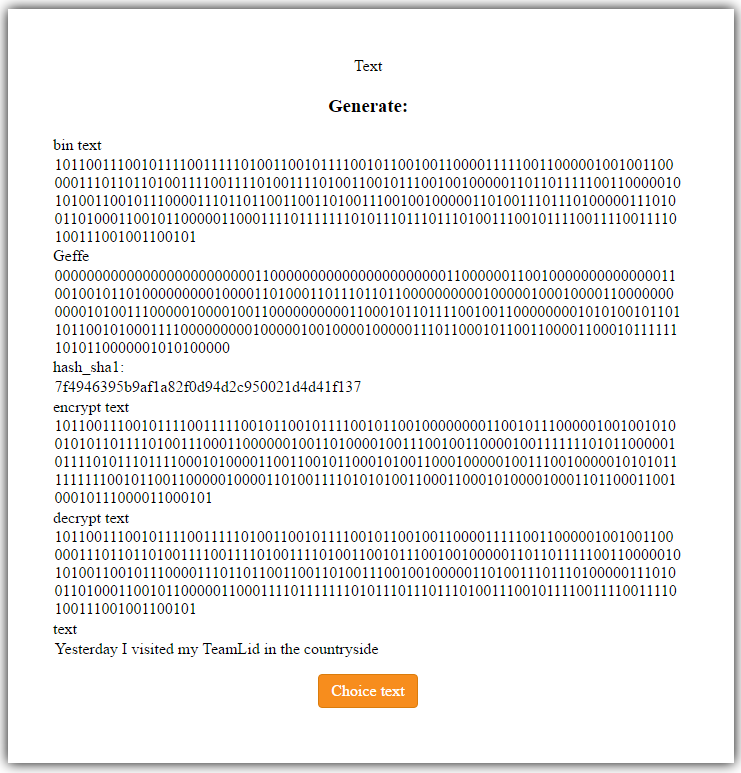


Рис4. Результат программы

**Вывод:**

В ходе выполнения данной лабораторной работы, мне удалось освоить базовую разработку графического интерфейса. Я научиться реализовывать потоковый алгоритм шифрования с использованием генераторов псевдослучайных чисел, а также получить опыт реализации собственных функций хеширования текстовых паролей.

**Список литературы:**

* Руководство по PHP https://www.php.net/manual/ru/index.php (Дата обращения 2.12.19).
* Лабораторная работа Мартынов Антон Иванович - Информационная безопасность и защита информации.
* Криптографические генераторы. Поточные шифры и их криптоанализ http://cryptowiki.net/index.php? (Дата обращения 8.12.19).
* sha1 https://www.php.net/manual/ru/function.sha1.php (Дата обращения 15.12.19).

**Приложение 1**

<?php

function BinString($bin\_string) {

$mybitseq = "";

$hex\_seq = $bin\_string[0];

$end = strlen($bin\_string);

for($i = 1 ; $i <= $end; $i++){

if(strlen($hex\_seq) % 7 == 0) {

$char = chr(bindec($hex\_seq));

$hex\_seq = "";

$mybitseq .= $char;

}

$hex\_seq .= $bin\_string[$i];

if ($hex\_seq === '100000') {

$hex\_seq = "";

$mybitseq .= ' ';

continue;

}

}

return $mybitseq;

}

function StringBin($mystring) {

$mybitseq = "";

$end = strlen($mystring);

for($i = 0 ; $i < $end; $i++){

$bin = decbin(ord($mystring[$i]));

$mybitseq .= $bin;

}

return $mybitseq;

}

if($\_POST['command-name'] == "debug") {

$text = $\_POST['text'];

$bin = StringBin($text);

// //делается попытка создать его

$fp = fopen("file.txt", "w");

// записываем в файл текст

fwrite($fp, $bin);

// закрываем

fclose($fp);

}

function preProcess($message){

/\*

Pre-processing:

append the bit '1' to the message i.e. by adding 0x80 if characters are 8 bits.

append 0 ≤ k < 512 bits '0', thus the resulting message length (in bits)

is congruent to 448 (mod 512)

append ml, in a 64-bit big-endian integer. So now the message length is a multiple of 512 bits.

\*/

$originalSize = strlen($message) \* 8;

$message .= chr(128);

while (((strlen($message) + 8) % 64) !== 0) {

$message .= chr(0);

}

foreach (str\_split(sprintf('%064b', $originalSize), 8) as $bin) {

$message .= chr(bindec($bin));

}

return $message;

}

function rotl($x, $n) {

return ($x << $n) | ($x >> (32 - $n));

}

function SHAfunction($step, $b, $c, $d)

{

switch ($step) {

case 0;

return ($b & $c) ^ (~$b & $d);

case 1;

case 3;

return $b ^ $c ^ $d;

case 2;

return ($b & $c) ^ ($b & $d) ^ ($c & $d);

}

}

function hash\_sha1($input) {

$h0 = 0x67452301;

$h1 = 0xEFCDAB89;

$h2 = 0x98BADCFE;

$h3 = 0x10325476;

$h4 = 0xC3D2E1F0;

$K = array(0x5a827999, 0x6ed9eba1, 0x8f1bbcdc, 0xca62c1d6);

$message = preProcess($input);

// Process the message in successive 512-bit chunks:

// break message into 512-bit chunks

$chunks = str\_split($message, 64);

foreach ($chunks as $chunk) {

// break chunk into sixteen 32-bit big-endian words w[i], 0 ≤ i ≤ 15

$words = str\_split($chunk, 4);

foreach ($words as $i => $chrs) {

$chrs = str\_split($chrs);

$word = '';

foreach ($chrs as $chr) {

$word .= sprintf('%08b', ord($chr));

}

$words[$i] = bindec($word);

}

// Extend the sixteen 32-bit words into eighty 32-bit words:

for ($i = 16; $i < 80; $i++) {

// for i from 16 to 79

// w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]) leftrotate 1

$words[$i] = rotl($words[$i-3] ^ $words[$i-8] ^ $words[$i-14] ^ $words[$i-16], 1) & 0xffffffff;

}

// Initialize hash value for this chunk:

$a = $h0; $b = $h1; $c = $h2; $d = $h3; $e = $h4;

// Main loop:[39]

foreach ($words as $i => $word) {

$s = floor($i / 20);

$f = SHAfunction($s, $b, $c, $d);

$temp = rotl($a, 5) + $f + $e + $K[$s] + ($word) & 0xffffffff;

$e = $d;

$d = $c;

$c = rotl($b, 30);

$b = $a;

$a = $temp;

}

// Add this chunk's hash to result so far:

$h0 = ($h0 + $a) & 0xffffffff;

$h1 = ($h1 + $b) & 0xffffffff;

$h2 = ($h2 + $c) & 0xffffffff;

$h3 = ($h3 + $d) & 0xffffffff;

$h4 = ($h4 + $e) & 0xffffffff;

}

return sprintf('%08x%08x%08x%08x%08x', $h0, $h1, $h2, $h3, $h4);;

}

function Geffe($L1, $L2, $L3, $n) {

$holder = "";

for($i = 0; $i < $n; $i++) {

$L1 = ($L1 << 1) | ((($L1 >> 29)^ ($L1 >> 28) ^ ($L1 >> 25) ^ ($L1 >> 23)) & 1);

$L2 = ($L2 << 1) | ((($L2 >> 30)^ ($L2 >> 27)) & 1);

$L3 = ($L3 << 1) | ((($L3 >> 31)^ ($L3 >> 30) ^ ($L3 >> 29) ^ ($L3 >> 28) ^ ($L3 >> 26) ^ ($L3 >> 24)) & 1 );

$holder .= (((($L3 >> 32) & 1 )\*(($L1 >> 30) & 1)) ^ (((($L3 >> 32) & 1) ^ 1) \* (($L2 >> 31) & 1)) );

// $holder .= ' ';

}

return $holder;

}

?>

<?php

if($\_POST['file']) {

echo "<script>

$('#check').css(\"display\", \"none\");

$('#password').css(\"display\", \"none\");

</script>";

$text = file\_get\_contents($\_POST['file']);

$len = strlen($text);

$l1 = round(rand(1, 29)); // 29

$l2 = round(rand(1, 30)); // 30

$l3 = round(rand(1, 31)); // 31

$sequence = Geffe($l1, $l2, $l1, $len);

echo '<h3>Generate: </h3>';

echo '<div class=\'sequence\'>bin text<p>' . $text. '</p>Geffe<p>' . $sequence . '</p>';

echo 'hash\_sha1:<p>' . hash\_sha1($text). '</p>';

$new\_arr = '';

for($i = 0; $i < $len; $i++) {

$new\_arr .= '' . (int)$text[$i] ^ (int)$sequence[$i];

}

echo 'encrypt text<p>' . $new\_arr . '</p>decrypt text<p>';

$bin\_text = '';

for($i = 0; $i < $len; $i++) {

$bin\_text .= (int)$new\_arr[$i] ^ (int)$sequence[$i];

}

echo $bin\_text;

echo '</p>text<p>' . BinString($bin\_text). '</p></div>';

}

if($\_POST['password']) {

if(hash\_sha1($\_POST['password']) === '322f89ebdf649e7f850e07c1fd0bdf0d134ce6b9') {

echo "<script>

$('#file').css(\"display\", \"block\");

$('#cls').css(\"display\", \"block\");

$('#check').css(\"display\", \"none\");

$('#password').css(\"display\", \"none\");

</script>";

}

}

?>