**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет**   
**імені Петра Могили**

**Факультет комп'ютерних наук**

**Кафедра Інтелектуальних інформаційних систем**

**ЗВІТ**

*З лабораторної роботи №1*

*на тему:* Створення алгоритму шифрування SHA-256

Спеціальність “122 Комп’ютерні науки та ІТ”

6.122-401.1610108

***Cтудент*** Канатьєв М.Ю.

*\_\_\_\_\_\_\_\_\_\_\_* ***\_\_\_\_\_\_\_\_\_\_\_\_***

*(підпис)*

\_\_\_\_\_\_\_\_\_\_

*(дата)*

***Викладач*** Тогоєв О. Р.

*\_\_\_\_\_\_\_\_\_\_\_* ***\_\_\_\_\_\_\_\_\_\_\_\_***

*(підпис)*

*\_\_\_\_\_\_\_\_\_\_\_*

*(дата)*

Миколаїв – 2019

# Теоретичні відомості:

**SHA-2** (англ. *Secure Hash Algorithm Version 2* — безпечний алгоритм хешування, версія 2) — збірна назва односторонніх геш-функцій SHA-224, SHA-256, SHA-384 і SHA-512. Геш-функції призначені для створення «відбитків» або «дайджестів» повідомлень довільної бітової довжини. Застосовуються в різних додатках або компонентах, пов'язаних із захистом інформації.

**Історія**

Хеш-функції *SHA-2* розроблені Агентством національної безпеки США і опубліковані Національним інститутом стандартів і технології США у федеральному стандарті обробки інформації *FIPS PUB 180-2* в серпні 2002 року. У цей стандарт також увійшла геш-функція *SHA-1*, розроблена в 1995 році. У лютому 2004 року до *FIPS PUB 180-2* була додана *SHA-224*.

У жовтні 2008 року вийшла нова редакція стандарту — *FIPS PUB 180-3*.

В липні 2006 року з'явився стандарт RFC 4634 «Безпечні геш-алгоритми США (*SHA* і *HMAC-SHA*)», що описує *SHA-1* і сімейство *SHA −2*.

Агентство національної безпеки від імені держави випустило патент на *SHA-2*під ліцензією *Royalty Free*.

**Алгоритм**

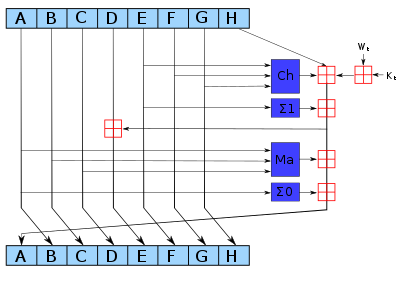
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:SHA-2.svg)

Схема однієї ітерації алгоритмів SHA-2

Геш-функції сімейства *SHA-2* побудовані на основі структури Меркла-Демґарда.

Початкове повідомлення після доповнення розбивається на блоки, кожен блок — на 16 слів. Алгоритм пропускає кожен блок повідомлення через цикл з 64-ма чи 80-ма ітераціями (раундами). На кожній ітерації 2 слова перетворюються, функцію перетворення задають інші слова. Результати обробки кожного блоку складаються, сума є значенням геш-функції.

Алгоритм використовує такі бітові операції:

ǁ — Конкатенація,

+ — Додавання,

*And* — Побітове «І»,

*Or* — Побітове «АБО»,

*Xor* — Виключне «АБО»,

*Shr* (Shift Right) — Логічний зсув вправо,

*Rotr* (Rotate Right) — Циклічний зсув вправо.

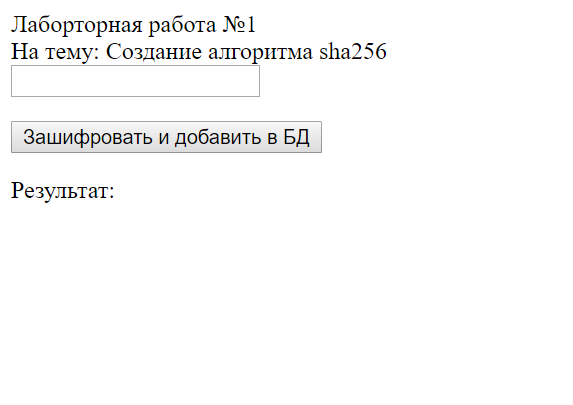
Завдання:

* На будь-які мові програмування відтворити алгоритм шифрування SHA-256.
* Результат повинен бути записаним в базу даних(на вибір студента).
* Створити репозиторій github з викладеними файлами та звітом.

**Виконання роботи:**

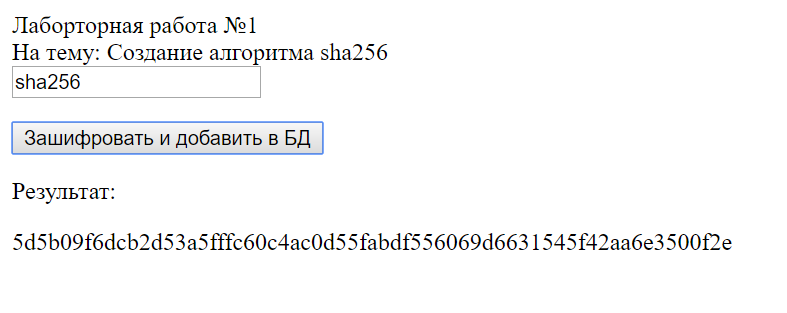
В даній лабораторну роботу було виконано на мові програмування PHP та JavaScript.

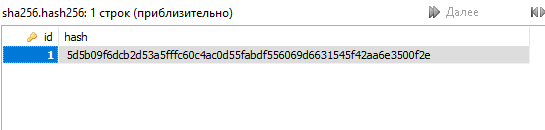
Спочатку було створено просту HTML форму вводу даних для шифрування.



Сторінка має поле ввода та кнопку.

Кнопка шифрує текст та додає його до бази даних

****

****

**Лістинг програми:**

titul.html:

<html>

<head>

<script type="text/javascript" src="jquery.js">

</script>

<script type="text/javascript">

$( document ).ready(function() {

console.log( "ready!" );

$( "#cryptButton" ).on( "click", function() {

var data = {'string': $('#textField')[0].value};

console.log('-=data=-', data);

$.ajax({

type: "POST",

data: data,

url: 'http://localhost/index.php',

success: function(xhr){

console.log('-=xhr=-', xhr);

$('#result').text(xhr);

},

});

});

});

</script>

</head>

<body>

<div>

Лаборторная работа №1

</br>На тему: Создание алгоритма sha256

</div>

<div>

<form>

<input id="textField">

</input>

</form>

<button id="cryptButton">

Зашифровать и добавить в БД

</button>

</div>

<div>

<p>

Результат:

</p>

<span id="result">

</span>

</div>

</body>

</html>

Index.php:

<?php

include './cryptClass.php';

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

$crypt = new SHA256();

$result = $crypt->hashing($string);

$link = mysqli\_connect('localhost', 'root', '', 'sha256');

if (!$link) {

die('Ошибка соединения: ' . mysql\_error());

} else {

$link->real\_query('INSERT INTO hash256 (hash) VALUES ("'.$result.'")');

}

mysqli\_close($link);

header('Access-Control-Allow-Origin: \*');

header ("Content-Type: text/html; charset=utf-8");

echo $result;

?>

CryptClass.php

<?php

class hashData

{

var $hash = null;

}

class hash

{

function hash($str = '', $mode = 'hex')

{

return false;

}

function hashChunk($str, $length, $mode = 'hex')

{

trigger\_error('hash::hashChunk() NOT IMPLEMENTED', E\_USER\_WARNING);

return false;

}

function hashFile($filename, $mode = 'hex')

{

trigger\_error('hash::hashFile() NOT IMPLEMENTED', E\_USER\_WARNING);

return false;

}

function hashChunkFile($filename, $length, $mode = 'hex')

{

trigger\_error('hash::hashChunkFile() NOT IMPLEMENTED', E\_USER\_WARNING);

return false;

}

}

class SHA256Data extends hashData

{

var $buf = array();

var $chunks = null;

function SHA256Data($str)

{

$M = strlen($str);

$L1 = ($M >> 28) & 0x0000000F;

$L2 = $M << 3;

$l = pack('N\*', $L1, $L2);

$k = $L2 + 64 + 1 + 511;

$k -= $k % 512 + $L2 + 64 + 1;

$k >>= 3; // convert to byte count

$str .= chr(0x80) . str\_repeat(chr(0), $k) . $l;

assert('strlen($str) % 64 == 0');

preg\_match\_all( '#.{64}#', $str, $this->chunks );

$this->chunks = $this->chunks[0];

$this->hash = array

(

1779033703, -1150833019,

1013904242, -1521486534,

1359893119, -1694144372,

528734635, 1541459225,

);

}

}

class SHA256 extends hash

{

static function hashing($str, $mode = 'hex')

{

static $modes = array( 'hex', 'bin', 'bit' );

$ret = false;

if(!in\_array(strtolower($mode), $modes))

{

trigger\_error('mode specified is unrecognized: ' . $mode, E\_USER\_WARNING);

}

else

{

$data = new SHA256Data($str);

SHA256::compute($data);

$func = array('SHA256', 'hash' . $mode);

if(is\_callable($func))

{

$func = 'hash' . $mode;

$ret = SHA256::$func($data);

}

else

{

trigger\_error('SHA256::hash' . $mode . '() NOT IMPLEMENTED.', E\_USER\_WARNING);

}

}

return $ret;

}

static function sum()

{

$T = 0;

for($x = 0, $y = func\_num\_args(); $x < $y; $x++)

{

$a = func\_get\_arg($x);

$c = 0;

for($i = 0; $i < 32; $i++)

{

$j = (($T >> $i) & 1) + (($a >> $i) & 1) + $c;

$c = ($j >> 1) & 1;

$j &= 1;

$T &= ~(1 << $i);

$T |= $j << $i;

}

}

return $T;

}

static function compute(&$hashData)

{

static $vars = 'abcdefgh';

static $K = null;

if($K === null)

{

$K = array (

1116352408, 1899447441, -1245643825, -373957723,

961987163, 1508970993, -1841331548, -1424204075,

-670586216, 310598401, 607225278, 1426881987,

1925078388, -2132889090, -1680079193, -1046744716,

-459576895, -272742522, 264347078, 604807628,

770255983, 1249150122, 1555081692, 1996064986,

-1740746414, -1473132947, -1341970488, -1084653625,

-958395405, -710438585, 113926993, 338241895,

666307205, 773529912, 1294757372, 1396182291,

1695183700, 1986661051, -2117940946, -1838011259,

-1564481375, -1474664885, -1035236496, -949202525,

-778901479, -694614492, -200395387, 275423344,

430227734, 506948616, 659060556, 883997877,

958139571, 1322822218, 1537002063, 1747873779,

1955562222, 2024104815, -2067236844, -1933114872,

-1866530822, -1538233109, -1090935817, -965641998,

);

}

$W = array();

for($i = 0, $numChunks = sizeof($hashData->chunks); $i < $numChunks; $i++)

{

for($j = 0; $j < 8; $j++)

${$vars{$j}} = $hashData->hash[$j];

for($j = 0; $j < 64; $j++)

{

if($j < 16)

{

$T1 = ord($hashData->chunks[$i]{$j\*4 }) & 0xFF; $T1 <<= 8;

$T1 |= ord($hashData->chunks[$i]{$j\*4+1}) & 0xFF; $T1 <<= 8;

$T1 |= ord($hashData->chunks[$i]{$j\*4+2}) & 0xFF; $T1 <<= 8;

$T1 |= ord($hashData->chunks[$i]{$j\*4+3}) & 0xFF;

$W[$j] = $T1;

}

else

{

$W[$j] = SHA256::sum(((($W[$j-2] >> 17) & 0x00007FFF) | ($W[$j-2] << 15)) ^ ((($W[$j-2] >> 19) & 0x00001FFF) | ($W[$j-2] << 13)) ^ (($W[$j-2] >> 10) & 0x003FFFFF), $W[$j-7], ((($W[$j-15] >> 7) & 0x01FFFFFF) | ($W[$j-15] << 25)) ^ ((($W[$j-15] >> 18) & 0x00003FFF) | ($W[$j-15] << 14)) ^ (($W[$j-15] >> 3) & 0x1FFFFFFF), $W[$j-16]);

}

$T1 = SHA256::sum($h, ((($e >> 6) & 0x03FFFFFF) | ($e << 26)) ^ ((($e >> 11) & 0x001FFFFF) | ($e << 21)) ^ ((($e >> 25) & 0x0000007F) | ($e << 7)), ($e & $f) ^ (~$e & $g), $K[$j], $W[$j]);

$T2 = SHA256::sum(((($a >> 2) & 0x3FFFFFFF) | ($a << 30)) ^ ((($a >> 13) & 0x0007FFFF) | ($a << 19)) ^ ((($a >> 22) & 0x000003FF) | ($a << 10)), ($a & $b) ^ ($a & $c) ^ ($b & $c));

$h = $g;

$g = $f;

$f = $e;

$e = SHA256::sum($d, $T1);

$d = $c;

$c = $b;

$b = $a;

$a = SHA256::sum($T1, $T2);

}

for($j = 0; $j < 8; $j++)

$hashData->hash[$j] = SHA256::sum(${$vars{$j}}, $hashData->hash[$j]);

}

}

static function hashHex(&$hashData)

{

$str = '';

reset($hashData->hash);

do

{

$str .= sprintf('%08x', current($hashData->hash));

}

while(next($hashData->hash));

return $str;

}

function hashBin(&$hashData)

{

$str = '';

reset($hashData->hash);

do

{

$str .= pack('N', current($hashData->hash));

}

while(next($hashData->hash));

return $str;

}

}

function hexerize($str)

{

$n = 0;

$b = 0;

if(is\_array($str))

{

reset($str);

$o = 'array(' . sizeof($str) . ')::' . "\n\n";

while($s = current($str))

{

$o .= hexerize($s);

next($str);

}

$o .= 'end array;'."\n";

}

else

{

if(is\_integer($str) || is\_float($str))

$str = pack('N',$str);

$o = 'string(' . strlen($str) . ')' . "::\n";

for($i = 0, $j = strlen($str); $i < $j; $i++, $b = $i % 4)

{

$o .= sprintf('%02X', ord($str{$i}));

if($i != 0 && $b == 3)

{

if($n == 3)

$o .= "\n";

else

$o .= ' ';

++$n;

$n %= 4;

}

}

}

return $o . "\n";

}

?>

**Висновок:** в ході лабораторної роботи було освоєно алгоритм шифрування SHA256. Отриманні знання про важливість захисту даних. Покращено навички роботи з мовами програмування PHP, JS,JQuery.