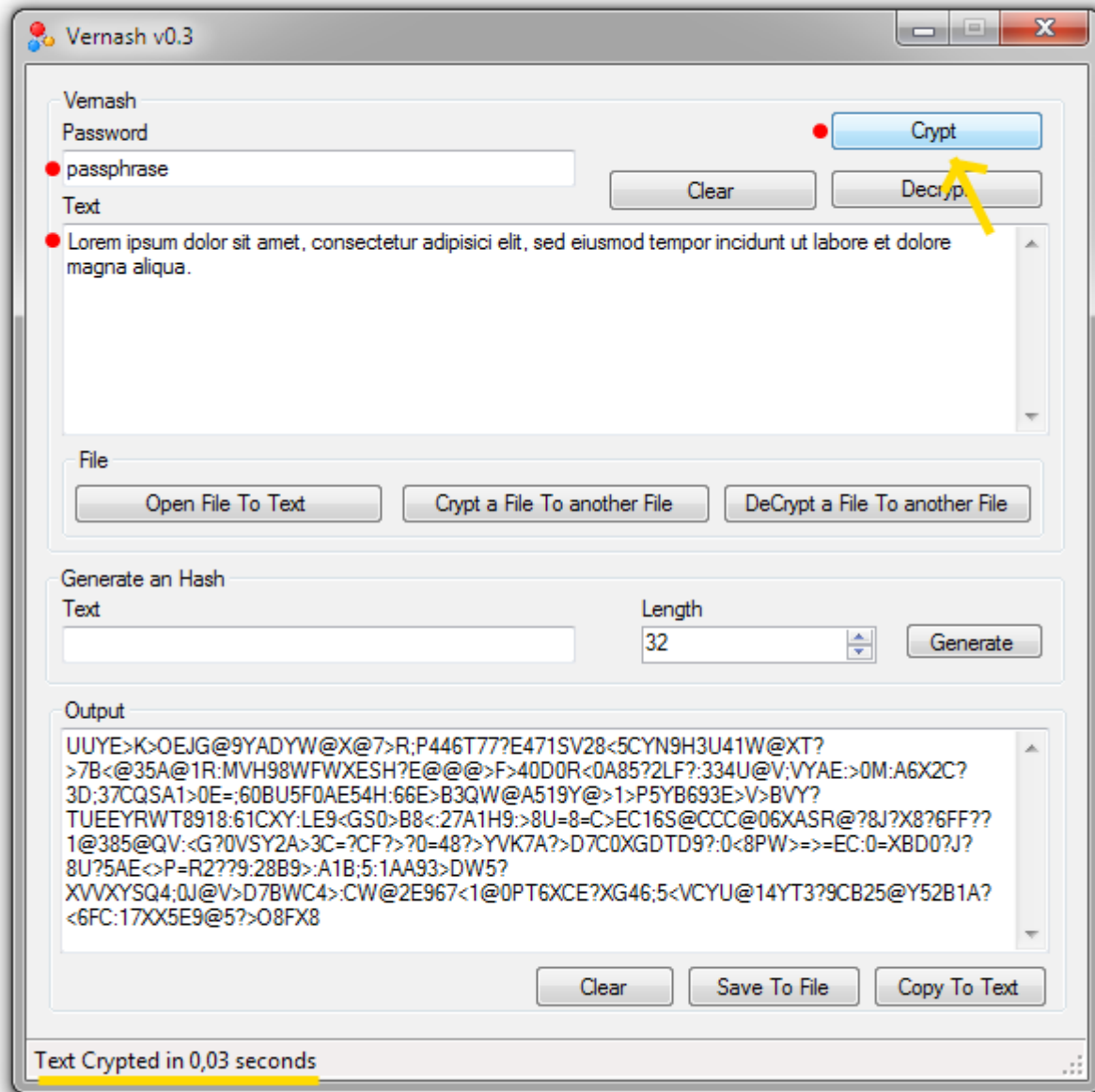


Vernash

Encryption (Step By Step)



vernashCrypt()

`public string vernashCrypt(byte[] text, string key, bool saveToFile, string filename)`

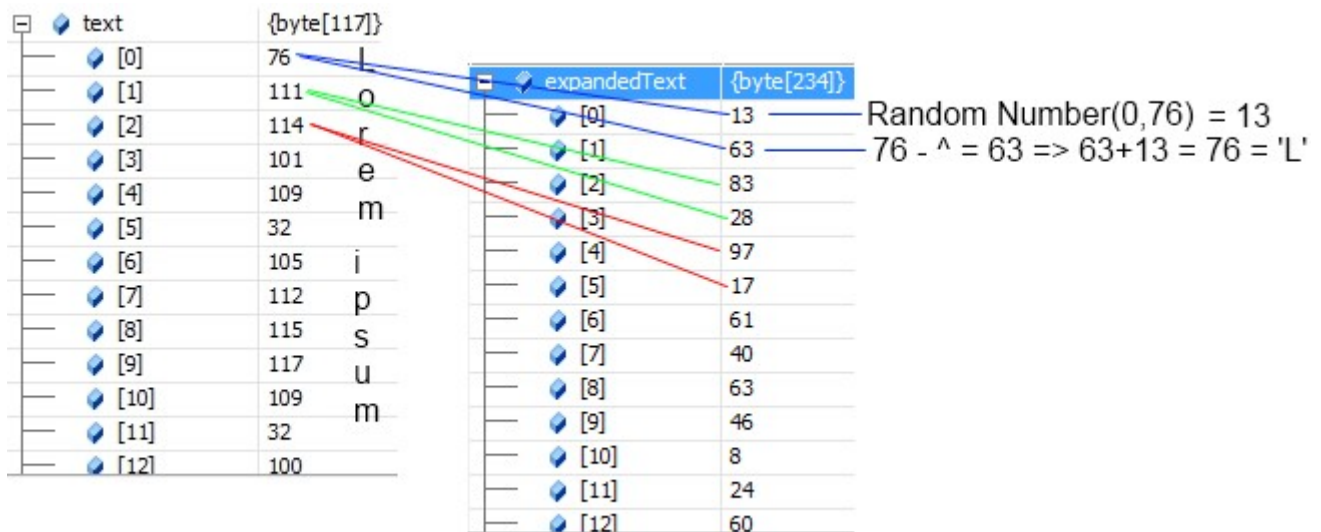
Vernash("Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incididunt ut labore et dolore magna aliqua.", "passphrase") ==> "UUYE>K>OEJG@9YADYW@X@7>R;P446T77?E471SV28<5CYN9H3U41W@XT?>7B<@35A@1R:MVH98WFWXESH?E@@@>F>40D0R<0A85?2LF?:334U@V;VYAE:>0M:A6X2C?3D;37CQSA1>0E=;60BU5F0AE54H:66E>B3QW@A519Y@>1>P5YB693E>V>...."

Step 1: randomExpansion(byte[] text)

```
/* randomExpansion
 *   Expand 1 byte to 2 bytes:
 *   1# byte is a random number from 0 to byte value
 *   2# is the rest.
 *
 * Example: randomExpansion('a') =>
 *   random_number = random number from 0 to 'a'
 *   randomExpansion = random_number & ('a' - random_number)
 *   ==> random_number + 'a' - random_number = 'a'
 */
byte[] randomExpansion(byte[] text)
{
    byte[] expandedText = new byte[text.Length * 2];
    int i = 0;
    Random random = new Random();

    foreach (byte c in text)
    {
        expandedText[i++] = (byte)random.Next(0, ((int)c));
        expandedText[i++] = (byte)((int)c - expandedText[i - 2]);
    }
    return expandedText;
}
```

randomExpansion("Loerm ipsum...") =>



Example: randomExpansion("ABC") =>

```
/* randomExpansion
 *   Expand 1 byte to 2 bytes:
 *   1# byte is a random number from 0 to byte value
 *   2# is the rest.
 *
 * Example: randomExpansion('a') =>
 *   random_number = random number from 0 to 'a'
 *   randomExpansion = random_number & ('a' - random_number)
 *   ==> random_number + 'a' - random_number = 'a'
 */
byte[] randomExpansion(byte[] text)
{
    byte[] expandedText = new byte[text.Length * 2];
    int i = 0;
    Random random = new Random();

    foreach (byte c in text)
    {
        expandedText[i++] = (byte)random.Next(0, ((int)c));
        expandedText[i++] = (byte)((int)c - expandedText[i - 2]);
    }
    return expandedText;
}
```

Espressione di controllo

Nome	Valore
text	{byte[3]}
[0]	65 'A'
[1]	66 'B'
[2]	67 'C'
expandedText	{byte[6]}
[0]	57 57 + 8 = 65
[1]	8
[2]	26 26 + 40 = 66
[3]	40
[4]	51 51 + 16 = 67
[5]	16

Step 2: expandText(byte[] text)

```
/* expandText
 *   expandText("ABC") ==> "414243"
 */
string expandText(byte[] text)
{
    string expandedText = string.Empty;

    foreach (byte c in text)
    {
        expandedText += String.Format("{0:X" + EXPANSION + "}", (int)c);
    }
    return expandedText;
}
```

expandText() =>

expandedText	"0D3F531C61113D283F2E08183C2D0070492A2E471F4E021E23411B542F3D5718155D031D076C5910195B1D03124F650
text	{byte[0x000000ea]}
[0x00000000]	0x0d
[0x00000001]	0x3f
[0x00000002]	0x53
[0x00000003]	0x1c
[0x00000004]	0x61
[0x00000005]	0x11
[0x00000006]	0x3d
[0x00000007]	0x28
[0x00000008]	0x3f
[0x00000009]	0x2e
[0x0000000a]	0x08
[0x0000000b]	0x18

text	{byte[234]}
[9]	13
[1]	63
[2]	83
[3]	28
[4]	97
[5]	17
[6]	61
[7]	40
[8]	63
[9]	46
[10]	8
[11]	24

expandedText =

"0D3F531C61113D283F2E08183C2D0070492A2E471F4E021E23411B542F3D5718155D031D076C5910195B1D03124F6508610461131616041C52112F40006E086B6302330621263023A3A70052D4509175B06283C42270F612C3D640F363343202346091744211C50482140340E1E081840335D08412317094C192C3D1461175C2B421758085C1A062D472D381F4E19574D22571B1D0355141D51055E3534095B264F5A145E16110F571E551F0C142E3E124F382A2E4142301F46150B372E3B3907192B39521D175556194C261C491C04006D2B363A2D640A3E231010006137352B3E3F322F4659081C12"

Step 3: scrambleText(string text, string key)

```
/* scrambleText
 *   Shuffles a string.
 *
 * Example:
 *   scrambleText ("12345", "password"); ==> "53241"
 *   unScrambleText("53241", "password"); ==> "12345"
 *
 *   scrambleText("abcdefghilmnopq1234567", "passphrase"); == "chan43mlopqdif6bg127e5"
 */
public string scrambleText(string text, string key)
{
    char[] scrambled = new char[text.Length];

    int[] iPos = genKeyPos(text.Length, key);

    for (int x = 0; x < text.Length; x++)
    {
        // swap
        scrambled[x] = text[iPos[x]];
    }
    return new string(scrambled);
}
```

**scrambleText("0D3F531C61113D283F2E08183C2D0070492A2E471F4E02...",
"passphrase") =>**

Before Shuffle

text	"0D3F531C61113D283F2E08183C2D0070492A2E471F4E021E23411B542F3D5718155D031D"
key	"passphrase"

After Shuffle

text	"0B1D314B3E6D11305C11354D00244F072B28391D180F072B6D33081F495F13201E0D274E02"
key	"passphrase"

Step 3,5: genKeyPos(int nPos, string key)

```
/* genKeyPos
 *   Generates pseudorandom order for the string to be shuffled.
 */
int[] genKeyPos(int nPos, string key)
{
    int[] iPos = new int[nPos];
    int newPos;
    int curW = 0;
```

genKeyPos(468, "passphrase") =>

Before		After	
nPos	468	nPos	468
key	"passphrase"	key	"passphrase"
iPos	{int[468]}	iPos	{int[468]}
[0]	0	[0]	216
[1]	1	[1]	389
[2]	2	[2]	296
[3]	3	[3]	59
[4]	4	[4]	278
[5]	5	[5]	11
[6]	6	[6]	118
[7]	7	[7]	53
[8]	8	[8]	350
[9]	9	[9]	219
[10]	10	[10]	8

Step 4: hashum(string sWord, int iTextLength)

```
/* hashum
 * Variable-Length Hashes Generator
 */
public string hashum(string sWord, int iTextLength)
{
    char[] hash = new char[iTextLength];

    if (sWord.Length < 1)
        return string.Empty;
}
```

hashum(expandText("passphrase"), textLen) =>

```
if (lastLength != textLen)
{
    hashKey = hashum(exKey, textLen);
    lastLength = textLen;
}
```

Espressione di controllo

Nome	Valore
key	"passphrase"
exKey	"70617373706872617365"
textLen	468
hashKey	"CYFTTNTUWNYFOCUGEEARLTWOWJHLANIVHIJFVDCPKMXETLWJYKHCEAVWDKQ!

Variable Length HASH =

"CYFTTNTUWNYFOCUGEEARLTWOWJHLANIVHIJFVDCPKMXETLWJYKHCEAVWDKQSDFKWEUMPDYPGDIBBWXYPXXRTVTKHHDXPMSMKQHPYBPBIJBWCPBBXUQB...."

Step 5: `char RotCharUp(int value, char lBound, char rBound)`

```
/* RotCharUp
 */
public char RotCharUp(int value, char lBound, char rBound)
{
    value += (lBound*2);
    return (char) (lBound + ((value - lBound) % (rBound - lBound)));
}
```

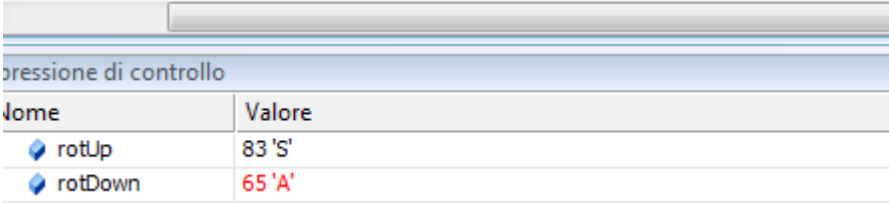
Final: Rotating chars

Now we have a pseudorandom, expanded and shuffled text and an Hash of the same length (468 bytes): we can rotate them!

Example:

1. Rotate UP('A' + 'X' in the range A-Z) => 'S'
2. Rotate DOWN('S' - 'X' in the range A-Z) => we should get: 'A'

```
vernash v = new vernash();
char rotUp = v.RotCharUp('A' + 'X', 'A', 'Z');
char rotDown = v.RotCharDown(rotUp - 'X', 'A', 'Z');
```



The screenshot shows a debugger window with a table of variables. The table has two columns: 'Nome' and 'Valore'. The first row shows 'rotUp' with a value of '83 'S''. The second row shows 'rotDown' with a value of '65 'A''.

Nome	Valore
rotUp	83 'S'
rotDown	65 'A'

We get: 'A'! Everything works!!!

Author:

Stefano 'Shen139' Alimonti
{ shen139 [at] gmail (dot) com }

Website:

<http://lab.openwebspider.org/vernash/>