GENERATE SERIAL CPP

Introduction

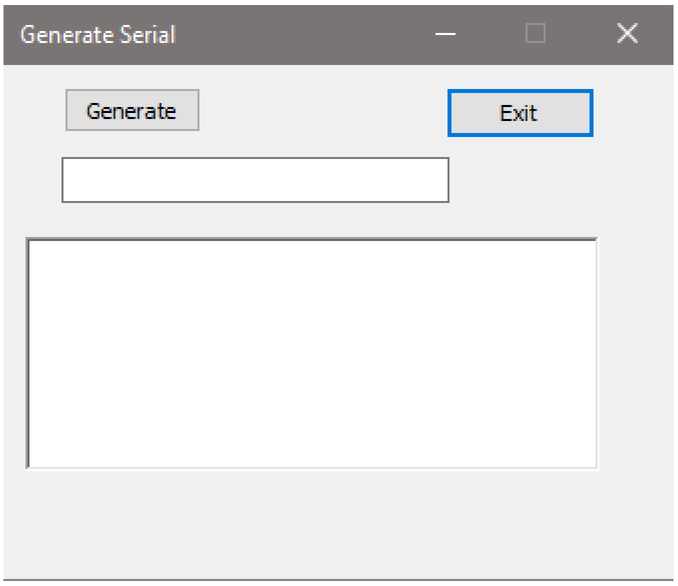
Creating a key generator for software is a sensitive issue, it has been the subject of many articles in various publications. The program presented here is based on the native Windows API. It was developed with Visual Studio 2019 and Cpp

Principe

Usually a picture board is used in which a series of characters is sought pseudo-randomly. The problem is essentially the word nickname. Then a series of more or less complicated rules in order to make decoding more or less difficult. The whole thing remains secret for as long as possible. The random generator used here is the one presented by Microsoft (Crypto API).

GUI Interface

The interface is simple, a dialog box with two buttons. A user's editing area provides the key they want to encode in the key generator. This key is between 5 and 12 characters.



For example, if you enter « HELLOBOYS ».

The generated Serial will be «EZ6SM3-3LS4P44PE-IMT6FL-KFWFYHYRL-JSBSA585O-V2X016-3VJLWO-ARPXHWSAY-489ZPBS-RDCDGXTA6 »

With each click the "Generate" button produces a different serial while respecting the key entered

Sample code

The entire program is written in C with Microsoft MFC with massive use of dynamically constructed classes and objects. It is difficult to present all the code develops this presentation and will make it soporific. This isn’t the aim.

Nota:

The key entered is converted to capital letters, the number of sections in the serial is the same as the length of the key entered. Only numbers and alphabetical characters are allowed (without spaces or special characters).

Random Function:

```
CString Generate_Serial_one ( int lg, int index)

{

HCRYPTPROV p;
          int temp [MAX_PATH];
   CString Result;
   int SUM = 0;

   if (CryptAcquireContext (&p, NULL, NULL,
      PROV_RSA_FULL, CRYPT_VERIFYCONTEXT) == FALSE)
     {
      return (_T(""));
     }
   if (CryptGenRandom (p, sizeof(temp), (BYTE*) &temp) == FALSE)
    {
      return (_T(""));
    }
   CryptReleaseContext (p, 0) ;

……

}
```

We then calculate the "checksum" of the letters generated. The number of letters generated in a section is set in a table.

```
CString  Generate_Serial(CString Key,int nb_key)
{
   int lg_Generate[] = {5,8,5,8,8,5,5,8,6,4,5,4};

   Gest_Serial_Key = Key.MakeUpper();

   for (int i = 0; i < nb_key; i++)
   {
      Result = Result + Generate_Serial_one(lg_Generate[i], i) + Separate;
   }

   Result = Result + Sum_Control;
   return (Result);……

}
```

The last section is composed of the sum of the checksums of the different sections. At the end of each section the first and then second character of the key provided by the user is inserted. Of course, this character is in little "hidden". A signature is used when generating each section to manage the overshooting of the loaded character board.

The whole thing is set in the file "stdafx.h"

```
#define _SIGNATURE     10
#define MB_MAX_SECTION 12
#define MB_MIN_SECTION  5
```

Shaping the result:

The result is displayed in a « ComboBox » and it’s store to « ClipBoard ».

Character table :

After the "n" character generation phase, the associated character is picked up in a character board. This table is dynamically loaded part of the resource section of the program (section: rcdata). If it is dumped from the executable, we will find binary data. It is a "little" encoded. It is encoded with the Huffman method.

```
void Init_Table ()
{
  CString Result = _T("");

   C_Password* pt_pass = new C_Password;
   Result = pt_pass->Get_Ressource_huff (MESSAGE_TABLE_STRING);
   delete pt_pass;
   Result.ReleaseBuffer();

…

}
```

Class Gest_Serial:

A single class generates both the creation of the "serial" as well as the decoding with two public functions

```
CString Generate_Serial (CString Key, int nb_key);

BOOL Control_serial (CString Valeur);
```

Linker Option :

The Linker instruction are used to merge the section ".text" with ".rcdata" without modify the code source only by add this option linker.

```
#pragma comment(linker,"/merge:.rdata=.text")
```

Nota:

In the project provided there are two files in the "res" sub directory:

TABLE_STRING.txt: the character table for encode and decoding

TABLE_STRING.huff: the same content but compressed.

In this project I use a table of characters included into “rcdata”. Section. If you want to change it you must proceed as follows.

1) Don't call Init_Table()

```
Void Gest_Serial()
{

…
 // Don’t call this Function
  //  Init_Table();

}
```

2) Change the statement of the character table as follows.

```
 // char      Buffer [MAX_PATH] = "";

// with

 char      Buffer [MAX_PATH] = "1234567890AZERTYUIOPQSDFMLKJHGNBVXCW";

// and modify the order
```

Of course, you'll have to hide this table of characters a little..

That's why I used the insert in the section “rcdata” section and I encoded it”.

Conclusion

Generating a "serial" key is simple enough for you to try to pay attention to certain points. I do not claim to have made an inviolable and complicated key generator but my goal was to show that we could complicate these two phases generation and decoding a "serial".