INTRODUCTION

An open source cryptographically secure pseudo random number generator on values distributed between zero and 2**56. It's safe, fast, with a large period and good statistical randomness quality. Algorithm, randomness, performance, cryptography and other issues, are explained forward.

ALGORITHM

Description

a, b, c, d and n as (16 bit)

I and m as (64 bit)

m=2**56

n = 2**16-1

a = 0

b = 0

c = 0

d = 2**13

X[0 to n] = Initialzed with a 56 bit values

Key-scheduling algorithm

for i from 0 to n

  a = (a + c + X[i] mod n + i) mod (n + 1)

  b = (b + a) mod (n + 1)

  c = (c + b) mod (n + 1)

  X[i] = (X[i] + a * b * c * d + a + b + c) mod m

endfor

Pseudo-random generation algorithm

i = 0

while GeneratingOutput:

  X[a] = (X[b] + X[c]) mod m

a = (a + b + i) mod (n + 1)

Output X[a] + X[b]

c = (c + a) mod (n + 1)

Output X[b] + X[c]

b = (b + c) mod (n + 1)

Output X[c] + X[a]

i = i + 1

endwhile

Bellow is the code in JavaScript, that describe the algorithm, generating 10000 pseudo random integer numbers.

```
a = 0
b = 0
c = 0
d = Math.pow(2,13)-1;
m = Math.pow(2,56);
n = Math.pow(2,16)-1;
x = new Array();

for ( i = 0 ;  i < = n  ; i + + )  {
        x[i] = new Date();
}

for ( i = 0 ;  i < = n  ; i + + )  {
        a = ( a + b + x[i] % n + i ) % ( n + 1 )
        c = ( b + a ) % ( n + 1 )
        b = ( c + a ) % ( n + 1 )
        x[i] = ( a * b * c * d + a ) % m
```

```
}

for ( i = 0 ; i < 3333 ;  i + + )  {
        x[a] = ( x[b] + x[c] ) % m
        a = ( a + b + i ) % ( n + 1 )
        document.write( ( x[a] + x[b] ).toString().substring(1,17)+'<br>');
        c= ( c + a ) % ( n + 1 )
        document.write( ( x[c] + x[a] ).toString().substring(1,17)+'<br>');
        b = ( b + c ) % ( n + 1 )
        document.write( ( x[b] + x[c] ).toString().substring(1,17)+'<br>');
}
```

RANDOMNESS

Several statistical tests done with George Marsaglia's Diehard battery of tests, was not find any 0.00000 or 1.00000, p-values. Comparative tests done with binary files containing true random numbers generated by physical sources, such as atmospheric noise, atomic radiation, and chaotic source, given good results.

PERFORMANCE

Running the above code on Windows 10/PowerBasic Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz 2.40 GHz processor, the performance rounds the 725 MBytes/second at 1 cycle/Byte, to process the (2**27) 134217728 Bytes.

APPLICATION

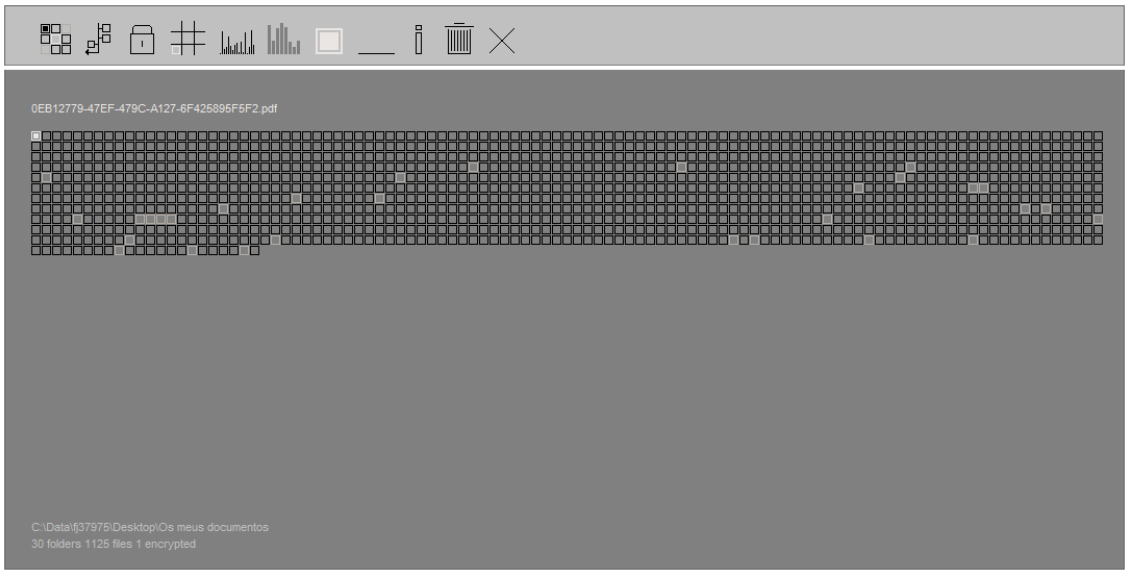
Generate 10000 numbers on:

http://www.number.com.pt/randomAdhcifar.html

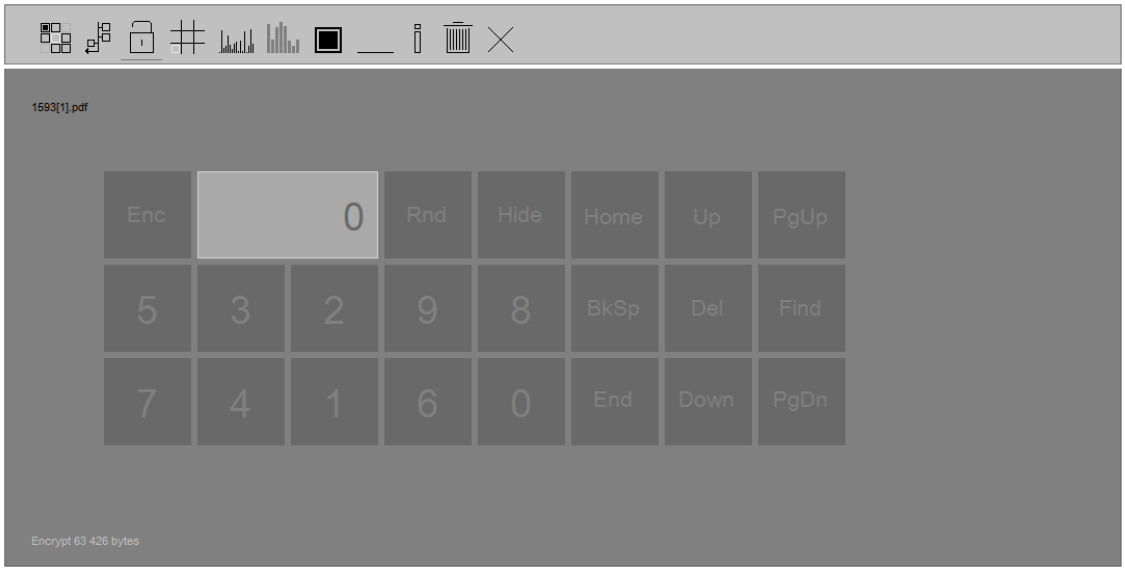
Open source application can be downloaded in:

http://www.number.com.pt/Melgo.zip

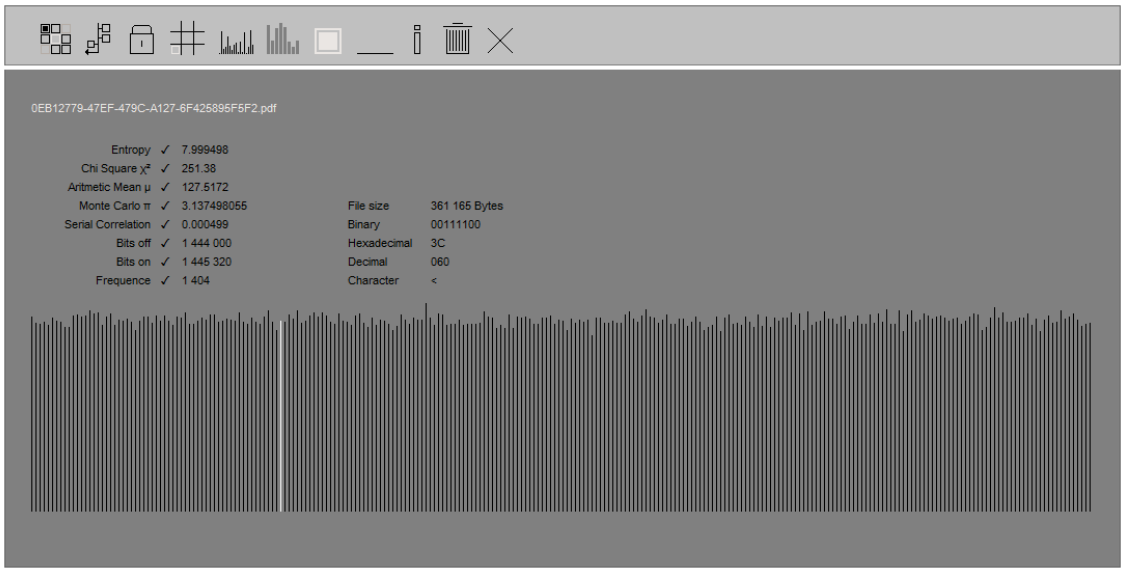
The bar menu.



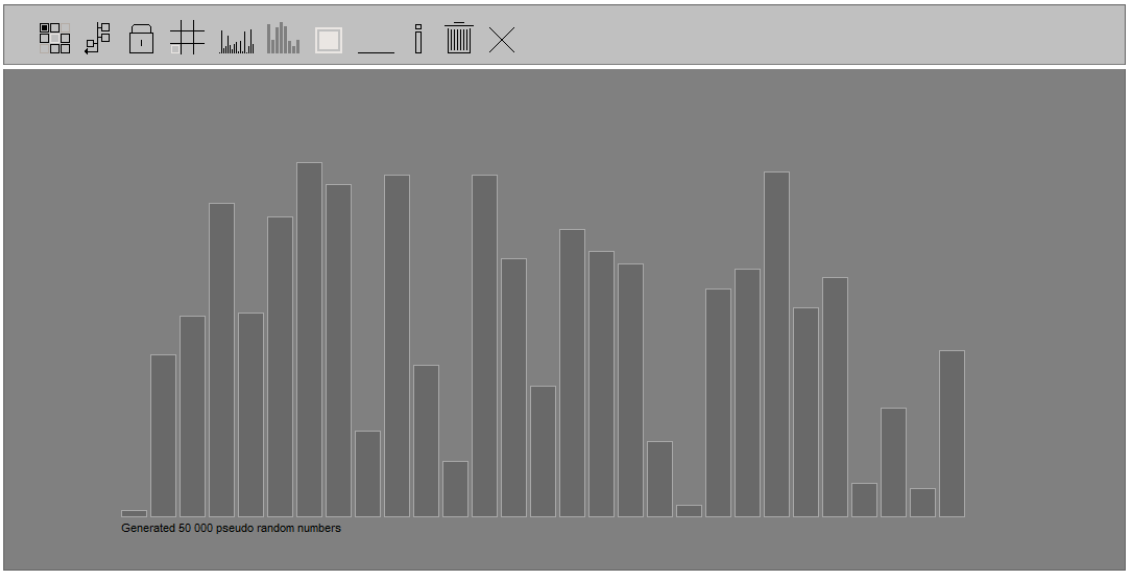
The file browser.



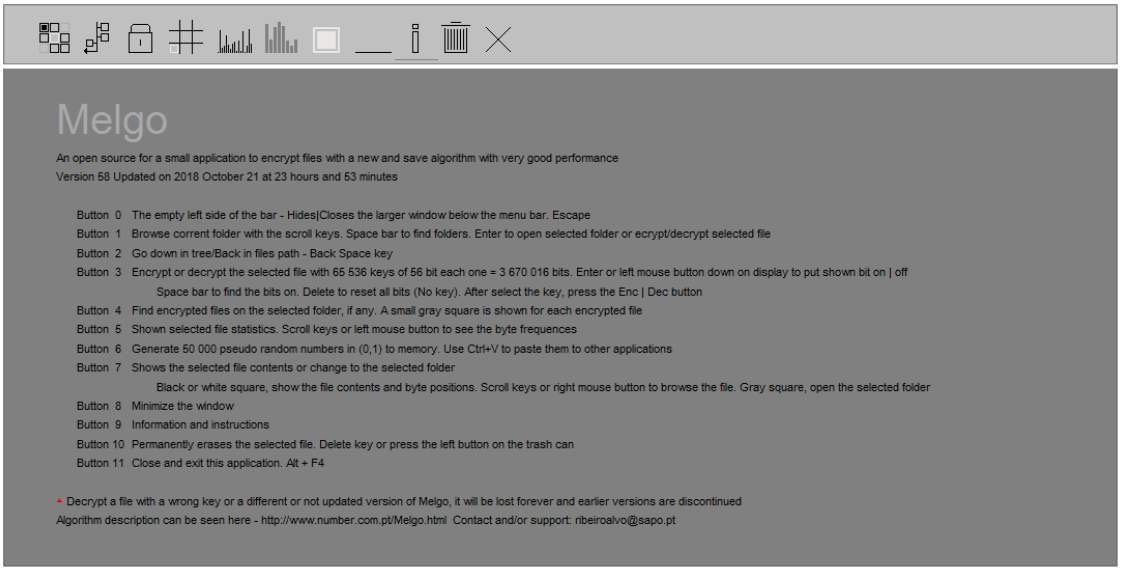
Module to encrypt/decrypt



Example of an encrypted file statistics.



Window of 50 000 pseudo random numbers generation



Help window

Module to erase permanently the selected file

CONCLUSION

Simple and elegant, tiny and fast this algorithm offers no difficulties on implementation, in any system/platform or programming language. I believe it can be used with various applications, for its statistical quality and encryption safety. Criticism or suggestions, contact me in ribeiroalvo@sapo.pt