# Nofish : A new stream cipher

Marius Oliver Gheorghita
331 Cotofenii Fata 207013,Dolj, Romania
e-mail: redwire05@yahoo.com

**Abstract**
The purpose of this paper is to provide a description of the Nofish cryptographic algorithm. This paper is intended to release the algorithm to the cryptographic community, for its future analysis and possible use. The name of the algorithm comes in respect to other well-known secure crypto algorithms like Blowfish and Twofish and since this one has not gained this status, I call it Nofish.

## 1. Algorithm description

The proposed algorithm is a synchronous stream cipher, more precisely a binary additive stream cipher because it using the XOR function to encrypt the plaintext. The design is based on HENKOS stream cipher (http://eprint.iacr.org/2004/080.pdf), the functions used in the internal state are kept, the initialization and mixing key part being modified with respect to its revealed weaknesses. This stream cipher uses a named key of 64 bytes (512 bits) as a secret key and no initialization vector.

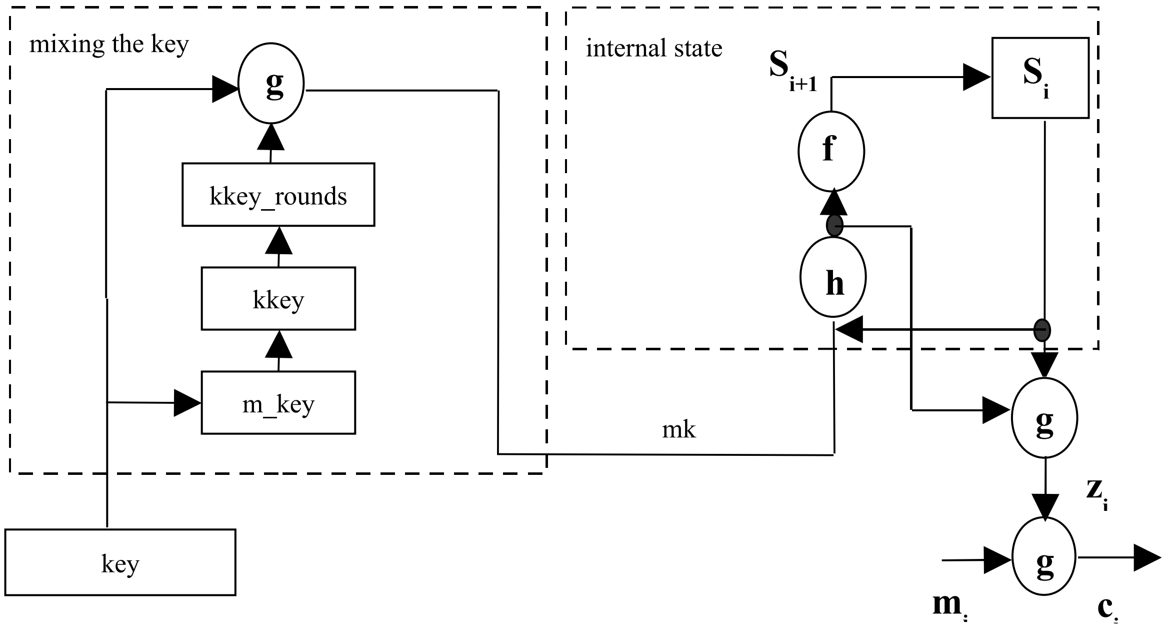

Figure 1. Structure of the Nofish stream cipher

Notations:
$m_i$ – stream of plaintext
$c_i$ – stream of ciphertext
$z_i$ – keystream
h=SW function;
f=AD function;
$S_i$ – internal state i; $S_0$ – initial state;
g=XOR;

*Key mixing*
Role of this module is to mix enough the provided key and is using as an initialization value kkey, set to 512; the ouput is a new kkey value.

m_key[i]= i^key[i] , i=0,63

There are four sets of values:
S1=[7,26,45]
S2=[5,22,39,56]
S3=[3,16,29,42,55]
S4=[2,13,24,35,46,57]
If j (for j=0,63) is in first set of values then kkey value is decreased with m_key[j], and if it is in the second set of values becomes the reminder of the division with m_key[j] (modulo operation). If it is in the third set, kkey is incresead with m_key[j] and for the last one is multiplied with m_key[j].
A number of rounds equal with kkey value is performed for mixing the m_key vector, in each round being calculated the sum of the m_key values and used to calculate the next m_key values as the reminder of the division with the value 67-i, i=0,63.

*Key initialization*
In this part of the algorithm, transform key in order to obtain a proper initialization before it can be used to generate the keystream. It is done using two major functions: one is the “switch” function (SW), which will mix the bytes of the data key as follows: every byte *j* is switched with byte *k* in the data key, where *k* is the value from the mixed key (mk) in the *i* position.
The next function is an additive function (AD) that will replace the value from each position with the sum between two near bytes, excepting the last value, which is obtained as a sum between the last byte and the first byte.
After these two transformations, obtain an intermediate data key; to initialize the data key properly, these rounds will be repeated T times (a variable number comparing with HENKOS design which uses a fixed 64 value), without producing any output, where T=64+kkey%64. After the last round a mixed key is released in order to be used for stream generation.

*Keystream generation*
To obtain the keystream, $z_i = g(S_{i+1}, t(S_i))$, where $S_i$ is the output from the last cycle of key initialization.
For generation of a keystream with predefined length, function g must be applied as long as necessary.

*Encryption/decryption*
The encryption/decryption between the plaintext/ciphertext is done using XOR:
$c_i = h(m_i, z_i)$ ; $m_i = h(c_i, z_i)$ ;
$c_i$ = ciphertext, $m_i$ = plaintext, $z_i$ = keystream;

## 2. Performance

The provided source code from the annex implementation using C language running on a Celeron 1.73 GHz reporting speed about 60MB/s at keystream generation, an optimized or assembler version could perform better.

## 3. Security analysis

*Time/Memory/Data tradeoff attacks*
This kind of attack has two phases: During pre-computation phase the attacker exploits the structure of the stream cipher and summarizes his findings in large tables. During the attack phase, the attacker uses these tables and the observed data to determine the secret key or the internal state of the stream cipher.
The size of the tables in the pre-computation stage, the required keystream, and the computational effort required to recover the secret key determine the feasibility of this attack. A simple way to provide security against this attack in stream ciphers is to increase the search space. In Nofish stream cipher the size of the internal state and the secret key space is 512 bits.

*Related key Attack*
Related key attack is attempted to find two different keys that will produce the same keystream. The cipher isn't vulnerable to this kind of attack, it was verified correlation between keystream produces from keys that differ through one bit one of another.
It can be assumed that for keys that differ through more bits, the possibility to appear correlation between produced keystreams under the same data key become negligible.

*Statistical tests analysis*
A keystream generator that exhibits basic statistical biases or detectable characteristics is weak. The output from Nofish has been tested using the statistical test packages like Ent, Diehard, Rabigete and have detected no statistical weaknesses.

## 4. Summary

The proposed Nofish algorithm is fast, in particular faster than AES in counter and CBC mode, has a small size which made it quite flexible for various platforms and implementations and it seems to provide an adequate level of security.

## References:

[1] Handbook of applied cryptography, A. Menezes, P. van Oorschot and S. Vanstone, CRC Press, 1996.
[2] Applied cryptography second edition, B. Schneier, J Wiley &Sons Inc. 1996.
[3] Analysis and design of stream ciphers, R.A Rueppel, Springer-Verlag, 1996.
[4] An introduction to new stream cipher design, Tor E. Bjorstad, University of Bergen, Norway, 2008.
[5] Turing a fast stream cipher, G. Rose, P. Hawkes, Qualcomm Australia, 2002.
[6] Primitive specification for SOBER-128, P. Hawkes, G. Rose, Qualcomm Australia, 2003.
[7] Design and primitive specification for Shannon, P.Hawkes, C. McDonald, M. Paddon, G. Rose, M. Wiggers de Vries, Qualcomm Australia, 2007.
[8] eSTREAM project Ecrypt II –European Network of excellence in Cryptology II http://www.ecrypt.eu.org/

## Appendix A

This appendix presents the ANSI C source code for Nofish.

```
/* filename nofish.c*/
/*Author: Marius Oliver Gheorghita e-mail: redwire05@yahoo.com*/
/*THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND AGAINST
INFRINGEMENT ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
/*
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <alloc.h>
#include <conio.h>
#include <io.h>

        const keylen=65,isize=65;
        FILE *fd,*fp;
        unsigned long cacheidx=0,cachesize,kkey;
        long num_blocks;
        unsigned int input[isize],m_key[isize];
        unsigned char *cache,key[isize];
        unsigned int start,stop,sump;
        unsigned int i,k=0,primul,handle;
        char file_d[128],inpath[128],outpath[128],ppath[128];
```

```
void mix_key()
{
                                kkey=512;
                                for(i=0;i<64;i++)
                                {
                                if(i%19==7) kkey-=i^key[i];
                                  else if(i%17==5) kkey%=i^key[i];
                                    else if(i%13==3) kkey+=i^key[i];
                                      else if(i%11==2) kkey*=i^key[i];
                                }
                                kkey=abs(kkey);

                                for(i=0;i<64;i++)
                                   m_key[i]=i^key[i];

                                for(int j=0;j<=kkey;j++)
                                  {
                                   sump=0;
                                   for(i=0;i<64;i++)
                                    sump+=m_key[i];
                                   for(i=0;i<64;i++)
                                    m_key[i]=sump%(67-i);
                                  }

                                for(i=0;i<64;i++)
                                 {
                                   input[i]=m_key[i]^key[i];
                                 }
}


void generare_secv(unsigned long X)
{
                register int i,x;
                cacheidx=0;
                cachesize=(coreleft()&((keylen-1)^-1));
                cache=(char *)malloc(cachesize);
                mix_key();
                while(k<(64+kkey%64))
                {
                                for(i=0;i<64;i++)
                                {
                                                x=input[m_key[i]];
                                                input[m_key[i]]=input[i];
                                                input[i]=x;
                                }
                                primul=input[0];
                                for(i=0;i<(keylen-1);i++)
                                                input[i]=((input[i]+input[i+1])%256)^(int)key[i];
                                input[keylen-1]=((input[keylen-1]+primul)%256)^(int)key[keylen-1];
                                k++;
                }
                num_blocks=X/keylen+1;
                while(num_blocks--)
                {
                                if(cacheidx>(cachesize-1))
                                {
                                                fwrite(cache,1,cacheidx,fd);
                                                cacheidx=0;
                                }
                                for(i=0;i<64;i++)
                                {
                                                x=input[m_key[i]];
                                                input[m_key[i]]=input[i];
                                                input[i]=x;
                                }
                                primul=input[0];
                                for(i=0;i<(keylen-1);i++)
                                                input[i]=((input[i]+input[i+1])%256)^(int)key[i];
                                input[keylen-1]=((input[keylen-1]+primul)%256)^(int)key[keylen-1];
```

```
				for(i=0;i<keylen-1;i++)
					cache[cacheidx++]=(char)(input[i]^((input[i]+input[i+1])%256)^key[i]);
				cache[cacheidx++]=(char)(input[keylen-1]^((input[keylen-1]+input[0])%256)^key[keylen-1]);
		}
		if(cacheidx)		fwrite(cache,1,X%cachesize,fd);
		free(cache);
}

int main(int argc, char* argv[])
{
		if(argc<4)
		{
				printf("\n usage: nofishg <key_path> <stream_number_of_bytes> <stream_path>\n",file_d);
				return -1;
		}
		if((fd=fopen(argv[1],"rb"))==NULL)
		{
				printf("Error opening file %s",file_d);
				return -1;
		}
		i=0;
		while(!feof(fd))
		{
		 key[i]=(int)fgetc(fd);
		 i++;
		}
		fclose(fd);
		num_blocks=atol(argv[2]);
		fd=fopen(argv[3],"wb");
		start=clock();
		generare_secv(num_blocks);
		stop=clock();
		fclose(fd);
		printf("\r\nTime: %f s\n",(double)(stop-start)/CLOCKS_PER_SEC);
		return 0;
}
```