PseudoRandomSequencesPy Library v 1.0.

Alejandro Arroyo Arzubi[1], Hugo Ballesteros[2], Marcelo Cipiano[1,3]

[1] Escuela Superior Técnica - IESE - C1426AAA, Buenos Aires, Argentina; [2] Instituto de Investigaciones Científicas y Técnicas para la Defensa - B1603ALO, Villa Martelli. Argentina; [3] Instituto Fátima – C1437CHD, Buenos Aires, Argentina.

Alejandro Arroyo Arzubi arroyoarzibi@iese.edu.ar, Hugo Ballesteros hballesteros@citefa.gov.ar, Marcelo Cipriano marcelocipriano@iese.edu.ar

Abstract: An open source library in Python language is developed for academic use. It can also be applied in software development. The library allows users to implement applications using relevant stream ciphers, to evaluate pseudorandom sequences and their robustness in cryptographic applications. Its use may enhance teaching techniques, improve software readability, and save encoding times at the developmental stage.

1 Introduction

With the advent of computers and telecommunications, the field of Information Security, once restricted to the military and diplomatic arena, has extended its influence to the civil and commercial scene. ATM connections, credit or debit card payments, home banking, and cellular communications are a few examples of data transmission requiring safety procedures.

Hence, teaching techniques addressing information security are strongly pervasive in academic settings [01]. The library is created, as it was its sister library NumTheoryPy [02], with a dual purpose: academic education and software development.

2 General Objectives of the Project

The main objective of the Project is to design and develop a library written in a GNU dynamic language with potential for further development. It aims to enhance direct experiences in an academic environment by merging theory and practice to foster the teaching/learning process [03] [04] [05].

In the field of software development, it aims to assist at the encoding stage by saving time and by gaining software readability.

3 Criteria for Library Design

3.1- The library must be open source.
To accomplish greater accessibility at no cost, and to enable improvements under the GNU license provisions

3.2- User control must be safeguarded.
Users utilize the software with their own objectives, methods, and styles. The library is a code cluster to be employed whenever called upon and not a final product.

3.3- Code lines must be kept at a minimum
The design reduces quantity of lines without compromising readability and algorithm comprehension. Users define catch tasks and validate required input data for each function, as well as data screen displays.

4 Programming Language Selection

Following the criteria established with the NumTheoryPy library, Python language is selected once again [09]. Created in 1991 by Guido Van Rossum, the language provides features granting the project with:

- Clarity in writing and reading the code
- Capability to work with numbers and mathematical functions
- Simplicity in the creation and implementation of functions, libraries, and modules
- Free of charge accessibility
- Solid background and potential for future development.

5 PseudoRandomSequencesPy Library

5.1 Random and pseudorandom sequences

Random sequences generate through aleatoric processes; for example at the toss of a coin, dice, or with white noise of radioactive emissions later translated into binary code. The required property to protect information with a random sequence is its unpredictability. (Given any part of a sequence, the subsequent bit cannot be predicted with probability greater than 0,5)

Pseudorandom sequences are number sequences generated with deterministic procedures or recursive formulae. They lack unpredictability by nature. However, they can be used in cryptography if they meet certain conditions [09][10] because their behavior resemble that of a random sequence.

Vernam cipher, also known as *One Time Pad,* is the only system whose security has been mathematically proven. Its encryption keys are long random or pseudorandom sequences, as lengthy as the language itself. Note that the receiver must deliver the sender the encryption key (through a different secure channel). This difficulty then, known as the *key distribution problem,* can be solved with the use of certain pseudorandom sequences if the receiver is able to reconstruct the sequence with only a small portion of information and hence avoiding transmitting the entire sequence.

5.2 Stream Cipher

A *stream cipher* is a cryptographic system which, through a XOR operation, joins text with key; thus originating the encrypted text. The key is normally a pseudorandom sequence with unpredictability properties, granting the encrypted text with the appropriate protection.

Considering these systems have high encryption speeds and a low processing capacity, cryptography and information security have focused on studying random and pseudorandom sequences and the methods to generate them.

Stream cipher methods are applied to symmetric or *private key* systems, because the same key is used for encryption and decryption – as opposed to asymmetric systems or *public key* systems. In the latter, two different keys are used: a public key for encryption and a private key for decryption.

5.3 Library General Design

In the first version 1.0, the library consists of 12 functions. Note that in this framework a function is understood in the computer science, and not mathematical, sense; i.e., a piece of software which can be executed when called upon by other software, avoiding code redundancy.

Certainly, the library does not cover the entire set of procedures and algorithms used in stream ciphers. However, this initial version 1.0 addresses the basic procedures to generate pseudorandom sequences and the tests required to satisfy the conditions for cryptographic security.

Each function has its specific output which the user shall utilize for his or her particular procedure. Although the functions are not designed to be called upon from the command line, as the output does not display information clearly to the user, it can be accomplished because Python has such features. It can also become a powerful calculator.

Users call upon the functions from software with validation capabilities for the input data and conveniently present the output values. Functions do not validate. Whenever a function requires the input of a prime number and a compound number is entered, the output could be compromised and incorrect. Validation needs to be performed at a previous stage.

6 Library Functions

Functions, valid input and output values are detailed:

6.1 Function BINARYCONV(A)

Converts a base-10 number into a binary expression

```
def BINARYCONV(A):
    """Binary conversion of A"""
    D=""
    w=0
    while w<>1:
        r=int(A%2)
        w=int(A//2)
        D=D+str(r)
        A=w
    D=D+str(w)
    return(D)
```

6.2 Function PARITYBIT(N).

Finds a bit parity d of decimal number *N* when converted to a binary number

```
def PARITYBIT(N):
    """return parity bit of n"""
    a=BINARYCONV(N)
    p=a.count("1")
    r=p%2
    return(r)
```

6.3 Function SEQUENCEOPEN(F)

This function reads a sequence stored in file F

```
def SEQUENCEOPEN(F):
    """read binary sequence stored in file A"""
    f=""
    linea=""
    f=open(F,"r")
    linea=f.read()
    return(linea)
```

6.4 Function SEQUENCESAVE(A, F)

Stores the number sequence A in file F

```
def SEQUENCESAVE(A,F):
    """save binary sequence A in file F"""
    h=""
    for i in range (len(A)):
        h=h+str(A[i])
    f=open(F,"w")
    f.write(h)
    f.close()
    return(1)
```

6.5 Function PARITYBITSEQUENCE (A)

Finds all parity bits of the number sequence A

```
def PARITYBITSEQUENCE(A):
    """RETURN PARITY BIT OF SECUENCE´S NUMBERS"""
    B=[]
    h=len(A)
    for i in range(0,h):
        B.append(PARITYBIT(A[i]))
    return(B)
```

6.6 Function BBSGENERATOR(R, N, S)

Generates a pseudorandom sequence of R numbers using the algorithm BLUM-BLUM-SHUB, module M and seed S

Note that this function calls upon another function called NTEXPMOD developed in our NumtheoryPy library. Such function performs modular exponentiation efficiently and rapidly.

```
def BBSGENERATOR(R,N,S):
    """pseudorandom number generator of BLUM-BLUM-
SHUB"""
    """for mod(M) and start on S"""
    from criptolabnumtheorypy import NTEXPMOD
    c=0
    RN=[]
    while c<R:
        h=NTEXPMOD(S,2,N)[0]
        u=PARITYBIT(h)
        RN.append(u)
        c=c+1
        S=h
    return(RN)
```

6.7 Function LCGENERATOR(S, A, B, N, R)

Finds a pseudorandom sequence of R numbers using linear congruences of the form *A*X+B (mod N).* The sequence follows a recursive procedure in *X* from the initial value (or *seed) S.*

```
def LCGENERATOR(S,A,B,N,R):
    """Sequence of R numbers generated by LINEAR
CONGRUENCES"""
    """on form x1=Ax0+B (mod N)where S=x0"""
    i=1
    s=[]
    h=(A*S+B)%N
    s.append(h)
    while i<R:
        h=(A*h+B)%N
        s.append(h)
        i=i+1
    return(s)
```

6.8 Function MONOBITTEST(F).

Performs the MONOBIT test on the sequence of bits stored in file F, yielding 1 if successful or 0 otherwise, as well as the quantity of 0 and 1 bits.

```
def MONOBITTEST(F):
    """MONOBIT TEST OVER BITS SEQUENCE SAVED ON F
FILE."""
    linea=0
    bit1=0
    b=0
    f=open(F,"r")
    linea=f.read()
    f.close()
    bit1=linea.count("1")
    if 9654<bit1<10346):
        b=1
    else:
        b=0
    return(b,t,"0="+str(20000-bit1),"1="+str(bit1))
```

6.9 Function POKERTEST(F).

Performs the POKER test on a sequence of bits stored in file F, yielding 1 if successful and 0 otherwise. It also finds the value of the statistical comparison for 4 bit combinations and a summary of their quantity.

```
def POKERTEST(F):
    """ Uniform distributed for 4 bit combinations"""
    x=0.0
    b=0
    h=0
    i=0
    g=[]
    linea=0
    m=[]
    f=open(F,"r")
    linea=f.read()
    f.close()
    g=[linea.count("0000"),linea.count("0001"),linea.co
    unt("0010"),linea.count("0011")]
    g=g+[linea.count("0100"),linea.count("0101"),linea.
    count("0110"),linea.count("0111")]
    g=g+[linea.count("1000"),linea.count("1001"),linea.
    count("1010"),linea.count("1011")]
    g=g+[linea.count("1100"),linea.count("1101"),linea.
    count("1110"),linea.count("1111")]
    m=["0000="+str(g[0]),"0001="+str(g[1]),"0010="+str(
    g[2]),"0011="+str(g[3])]
    m=m+["0100="+str(g[4]),"0101="+str(g[5]),"0110="+st
    r(g[6]),"0111="+str(g[7])]
    m=m+["1000="+str(g[8]),"1001="+str(g[9]),"1010="+st
    r(g[10]),"1011="+str(g[11])]
    m=m+["1100="+str(g[12]),"1101="+str(g[13]),"1110="+
    str(g[14]),"1111="+str(g[15])]
    for i in range (0,16):
        h=h+g[i]**2
    x=(16/5000.0)*h-5000
    if (1.03<x<57.4):
        b=1
    else:
        b=0
    return(b,x,m)
```

6.10 Function RUNSTEST(F).

Performs a set of run tests -- a run is a sequence of consecutive all ones or zeros of different lengths. It yields 1 if the general test is successful and 0 otherwise, as well as the same values for individual tests.

```
def RUNSTEST(F):
    """A Run is a maximal sequence of consecutive bits
    of either all ones or all zeros"""
    g=[]
    h=[]
    f=open(F,"r")
    linea=f.read()
    f.close()
    g=[linea.count("00"),linea.count("11")]
    g=g+[linea.count("000"),linea.count("111")]
    g=g+[linea.count("0000"),linea.count("1111")]
    g=g+[linea.count("00000"),linea.count("11111")]
    g=g+[linea.count("000000"),linea.count("111111")]
    g=g+[linea.count("0000000000000000000000000000000
    0"),linea.count("1111111111111111111111111111111
    ")]
    if (2267<=g[0]<=2733) and (2267<=g[1]<=2733):
        h=[1]
    else:
        h=[0]
    if  (1079<=g[2]<=1421) and (1079<=g[3]<=1421):
        h=h+[1]
    else:
        h=h+[0]
    if (502<=g[4]<=748) and (502<=g[5]<=748):
        h=h+[1]
    else:
        h=h+[0]
    if (223<=g[6]<=402) and (223<=g[7]<=402):
        h=h+[1]
    else:
        h=h+[0]
    if (90<=g[8]<=223) and (90<=g[9]<=223):
        h=h+[1]
    else:
        h=h+[0]
    if (90<=g[10]<=223) and (90<=g[11]<=223):
        h=h+[1]
    else:
        h=h+[0]
    return (h)
```

7 APPLICACION AND PERFORMANCE EXAMPLE

For instance, a line in Python code for a program can be written to generate a pseudorandom sequence of 80 numbers using the linear congruential generator with the formula $X_{n+1}= 3*X_n+7 \; mod(97)$, starting at seed $X_0=5$. The parity bit is found for each member of the series and the pseudorandom sequence of bits is stored in the file *b.txt”*

```
SEQUENCESAVE(PARITYBIT(LCGENERATOR(5,3,7,97,80),”btxt”)
```

8 CONCLUSIONS

Information security (IS) protects information in processing and data storing systems, as well as in communications. This field is used increasingly in software products; academic institutions have focused on its teaching as well.

PSEUDORANDOMSEQUENCESPY library presented in this work was developed for educational purposes as well as for software development use. It offers educators, students, and developers the ability to run functions commonly used in stream cipher applications such as cellular communications and other devices requiring to process great amounts of data rapidly and with limited processing capabilities.

Software developers can save encoding time and gain readability for their products.

The library was developed entirely in PYTHON language, within the GNU open source language to grant its free distribution. In addition, due to its open source nature, users can read and verify its contents, and if necessary, modify them.

9 Further Work

More functions could be added to enhance the library, such as: linear and non linear feedback shift register (LFSR and NLFSR), or use the MASSEY-BERLEKAMP algorithm to find the linear complexity of a given pseudorandom sequence. It is also possible to add examples of non linear combiners, such as Beth-Piper and Gollmann cascade, among others.

10 References

[01] Schembari Paul: "Hands-*on Crypto": Experiential Learning in Cryptography*. Proceedings in the 11th Colloquium for Information System Security Education. Boston University. Boston MA, June 2007.

[02] Benaben, Castro Lechtaler, Cipriano, Liporace: "NumTheoryPy Library for Cryptography". XV Argentine Congress of Computer Science. ISBN 978-897-24068-4. Jujuy, Argentina. October 2009.

[03] Chong, S. *Cryptographic teachings tools*. School of Computer Science and Software Engineering. June, 2003.

[03] Olejar, D; Stanek M. *Some Aspects of Cryptology Teaching*. Department of Computer Science, Comenius University, 2001.

[04] Dulal, K. *Teaching Cryptography in an Applied Computing Program.* Journal of Computing Sciences in Colleges. Volume 21 Issue 4. April 2006.

[05] Baliga, A; Boztas, S. *Cryptography in the Classroom using Maple*. Department of Mathematics. Royal Melbourne Institute of Technology University. Melbourne, 2001.

[06] Koblitz, N. Cryptography as Teaching Tool. Criptologia. Vol 21. No 4. 1991

[07] Saunders, Bonnie. and Janet Beissinger "Using Cryptography to Teach Number Theory to Future Middle School Teachers" *Paper presented at the annual meeting of the The Mathematical Association of America MathFest, TBA, Madison, Wisconsin*, Jul 28, 2008.

[08] http://www.python.org/

[09] FIPS-PUB 140-1 *Security Requirements for Cryptographic Modules.* U.S. Department of Commerce / National Institute of Standards and Technology. United States, 1994.

[10] Golomb *Shift Register Sequences*, Holden-Day, San Francisco, 1967.