View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Servicio de Difusión de la Creación Intelectual

NumTheoryPy Library v.1: for Cryptography Education and Software Design

Alejandro Benaben[1], Antonio Ricardo Castro Lechtaler[1], Marcelo Cipiano[2], Julio César Liporace[2].

[1] Escuela Superior Técnica - IESE / Universidad Tecnológoca Nacional, C1426AAA, Buenos Aires, Argentina; [2] Escuela Superior Técnica - IESE C1426AAA, Buenos Aires, Argentina.
{Antonio Castro Lechtaler, acastro@iese.edu.ar, Marcelo Cipriano marcelocipriano@iese.edu.ar
Alejandro Benaben abenaben@gmail.com, jcliporace@arnet.com.ar}

Abstract. This work develops an open source library in Python with applications in academic settings, for educational purposes. It can also be used for general software development. It allows users to implement Number Theory applications commonly employed in Cryptography and Information System Security. Improvements in teaching quality, user software readability, and time savings at the encoding stage could be expected.

Keywords. Library, Python, Education, Cryptography, Number Theory.

1 Introduction

Information Security (IS) is the field focusing on granting integrity, confidentiality and availability of information in processing and communications systems. Its requirements increase over time. Hence, a greater number of applications are required as well as the need for their study in academic settings [01].

A significant number of its techniques and procedures are drawn from other fields, such as Number Theory. Therefore, teaching Cryptography, a fundamental component in information confidentiality, requires the study of particular mathematical functions and procedures.

The NumTheoryPy library was created with a twofold purpose: education and software development. In other words, its primary targets are the academic community and software developers.

Developed in Python, this tool provides applications for educational settings, achieving a positive synergy between theory and practice while fostering the learning process [02] [03] [04].

In the software development sector, developers can add this tool to their professional portfolio to save time at the encoding stage, achieving improved readability in their software.

2 General Criteria for Library Design

Three strategic criteria prevailed in the development of this project:

1- The library must be open source.
Greater accessibility at no cost.

2- User control must be safeguarded.
They use the software for their own objectives, methods, and styles. The library is a code cluster to be employed whenever called upon and not a final product.

3- Code lines must be kept at a minimum
No superfluous code lines. Users define catch tasks and validate required input data for each function, as well as data screen display.

3 ¿Why Do We Use Number Theory?

We focus on Number Theory (NT) because, among other disciplines, it constitutes the theoretical foundation of Cryptography and System Security. Increasingly, academic courses in the applied sciences have incorporated Number Theory in their subject matter [05], considered until relatively recently in the history of science as one of the "purest" disciplines. With its use, a greater and better understanding of the information and communications protection techniques can be achieved.

Some authors, such as Neal Koblitz [06], have suggested an inverse path: to use Cryptography and its applications to Information Systems and Communications (internet, e-mail confidentiality, digital certification, home banking, and others) as a teaching resource for Number Theory. Similarly, other authors encourage its use as a requirement in the subject matter for teacher education at the secondary level [06].

3.1 Number Theory or Arithmetic

In general terms, NT is a field of Mathematics dealing with the study of the set of integers, their operations and properties. Known also by the name of Arithmetic, it might have developed along with human beings at the moment they created the notion of quantity, i.e., the number. Regardless whether their representation consists of fingers, notch cuts on a bone, or any other method, numbers lie at the root of civilization.

Euclid (IV c. B.C.) used the notion of prime numbers and even conceived an elegant proof of their infinity.[1] Eratosthenes (III B.C.) continued working along the same line.

[1] Consider a finite quantity of prime numbers. Let r be the product of all of them. Then r+1 is not divisible by any of those primes. Thus, r+1 is a prime not included on the list or is

Twenty-three centuries later, massive compound primes protect our communications and information as they form part of a cryptographic system which so far has shown no apparent downfalls. Such system is called RSA after Rivest, Shamir and Adleman in 1977. The strength of RSA is based on what is known as the *factoring problem*

Another well-known problem, *the discrete logarithm problem*[2] originated the Diffie and Hellman method to exchange keys,[3] operating efficiently since its discovery to present times.

4 Language Selection

The selection of the appropriate language was of utmost importance considering its impact on the library and its goals for the Project

4.1 Requirements

Considering the library was conceived as an open resource, the chosen language had to be aligned to the general philosophy and objectives. The requirements were:

- Clarity in writing and reading the code. Considering its use in education, the code had to be easy to read, so students can rapidly grasp its contents.
- Capability to work with large integers. Cryptography uses integers with a large amount of binary digits (e.g., 2^{1024} o 2^{2048} bits). Consequently, the language had to be able to handle these values.
- Simplicity in the creation and implementation of functions, libraries, and modules.
- Free of charge accessibility. In addition, preference was given to GNU-GNP - General Public License- languages. In this manner, installation and user costs could be avoided.
- Background and long term lifespan. Mature languages were considered with widespread use among the software community and not reserved only for experts. A long lifespan was significant to ensure its suitability over time.

4.2 Python Language

The language chosen for the code was Python, created in 1991 by Guido Van Rossum, as it complied with the detailed requirements [08].

In addition, it provided:

- Multi-paradigm: allows the programmer to choose among the “structured,” “object-oriented,” or “functional programming” modes. It can also add “extensions.”

---

divisible by some other prime which is not on the list. Therefore, the quantity of prime numbers cannot be finite.

[2] Diffie-Hellman key exchange is based on this problem, as well as the El Gamal cryptosystem.

[3] Bailey Whitfield "Whit" Diffie, creator of Asymmetric Cryptography.

- Memory use through dynamic data types and reference counting.
- Dynamic name resolution, linking a method and a variable name during software execution.
- Expansion capabilities by accepting modules written in C and C++

5 Numtheorypy Library

5.1 Issues addressed

Along with language choice, library functions were selected. For this version 1.0, certain mathematical procedures with application to Number Theory and Cryptography were included.

Later, the procedures were encoded to create the software of the library. Multiple tests were carried out on each segment to detect and correct errors. After depurating all errors found, the final library was obtained, ready for implementation.

5.2 Library general concepts

Version 1.0 of the library consists of 12 functions -- the latter term used 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.

Each function has, in addition to its specific output, related output displays for educational use. One of them indicates the number of iterations used in the procedure, and execution time measured in seconds. Both results allow users to compare and carry out benchmark studies on procedure behavior. It also admits verification when contrasting theoretical and implementation results.[4]

Although these functions were not designed to be called upon from the command line, as the output does not display information clearly to the user, it can be achieved as Pyhon has such features, and it can also be used as a powerful calculator.

The user calls upon these 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:

[4] NTFERMATFACT and NTRHOPOLLARD or NTMONTECARLO functions obtain some factors of compound numbers using different methods. They cannot be used when searching for correlations between the size of the input number (measured in bits) and execution times or number of necessary iterations, showing both methods as NP type.

6.1 Function *NTGCD(a,b)* "Greatest Common Divisor"

Calculates the Greatest Common Divisor (GCD) between two positive integers *a* and *b*, using Euclid's algorithm. The output displays the GCD, number of iterations and execution time.

```
def NTGCD(a,b):
      r=0
      ti=0
      t=0
      tf=0
      i=0
      if a<b:
          s=a
          a=b
          b=s
      ti=clock()
      while b<>0:
          r=a%b
          a=b
          b=r
      i+=1
      tf=clock()
      return (a,i,tf-ti)
```

6.2 Function *NTLCM(a,b)* "Least Common Multiple"

Calculates the least common multiple (lcm) between two positive integers *a* and *b*. GCD properties are used: the quotient between the product *a* and *b* and their GDC.

The output displays the lcm, number of iterations and execution time. The latter is calculated as a function of the GCD process.

```
def NTLCM(a,b):
      D=NTGCD(a,b)
      m=(a*b)/D[0]
      return (m,D[1],D[2])
```

6.3 Function *NTINVERMOD(b,n)* (modular inverse)

Calculates the inverse of a number *b* (mod n), using *Euclid's Extended Algorithm*; i.e., finds *b* such that:

$$bd \equiv 1(n)\,(01) \qquad (01)$$

The condition for *d* to exist is *mcd(b,n) =1* and needs to be validated by the user.

The output displays the integer *d*, the number of iterations in the process and execution time

```
def NTINVERMOD(b,n):
      a=n
      f=0
      g=1L
      A=0L
      B=1L
      C=0L
      D=1L
      q=0L
      r=0L
      x=0L
      y=0L
      t=0
      ti=0
      tf=0
      i=0
      ti=clock()
      while b>0:
          q=a/b
          r=a-q*b
          x=B-q*A
          y=C-q*D
          a=b
          b=r
          B=A
          A=x
          C=D
          D=y
          i+=1
      d=a
      x=B
      y=C
      if y<0:
          y=(y+n)%n
      tf=clock()
      return(y,i,tf-ti)
```

6.4 Functions *NTRHOPOLLARD(n) and NTMONTECARLO(n)*

Given that this factoring method is known by both names, two input forms were defined for the same function. They find a non trivial compound factor of *n*.

The polynomial Z[x] is used, starting at x_0=2:

$$P(x) = x^2 + 1 \tag{02}$$

The output displays a factor of *n* (not necessarily prime), number of iterations and execution time.

Note: it is possible that the algorithm may yield factor 1 as output. In such case, the number entered is prime or P(x) and x_0=2 are not appropriate to find the factors of n.

```
def NTRHOPOLLARD(n):
    t=0
    ti=0
    tf=0
    i=0
    x=2L
    y=2L
    d=1L
    r=0L
    ti=clock()
    while d==1:
        x=(x**2+1)%n
        y=((y**2+1)**2+1)%n
        r=abs(x-y)
        i+=1
        d=NTGCD(r,n)
    tf=clock()
    if d==n:
        return(-1,i,tf-ti)
    else:
        return(d,i,tf-ti)

def NTMONTECARLO(n):
      return NTRHOPOLLARD(n)
```

6.5 Function *NTPI(n)*

Finds the approximate quantity of primes less or equal to a positive integer *n,* using the Theorem of Tchebycheff and evaluating the function Phi(x).

```
def NTPI(n):
      from math import log
      z=n/log(n)
      return(z)
```

6.6 Function *NTFERMATFACT(n)*

It finds a non trivial factor of *n,* if it is a compound number, using Fermat's factoring method. In the case the output are the trivial factors, *n* is prime.

```
def NTFERMATFACT(n):
    from math import floor
    a=0
    b=2
    p=0
    q=0
    i=0
    ti=0
    tf=0
    ti=clock()
    a=long(floor(n**0.5))
    while (b**0.5-long(b**0.5))<>0:
        a+=1
        b=long(a*a%n)
        i+=1
    p=long(a+(b**0.5))
    q=n/p
    tf=clock()
    return(p,q,i,tf-ti)
```

6.7 Function *NTLEGENDRE(a,p)*

This functions yields 1 if the positive integer *a* is a quadratic residue (mod *p*) and -1 if it is not.

The function could give a wrong answer if p is not prime. The method used is the traditional formula:

$$a^{\frac{(p-1)}{2}} \equiv 1(p) \tag{05}$$

```
def NTLEGENDRE(a,p):
    if a%p==0:
        return(0,0,0)
    k=(p-1)/2
    L=NTEXPMOD(a,k,p)
    if L[0]==1:
        return(1,L[1],L[2])
    if L[0]==p-1:
        return(-1,L[1],L[2])
```

6.8 Function *NTBINARYCONV(a)* "Binary conversion"

It yields the binary conversion of a positive integer *a*. It accelerates the procedure for the exponential functions in modular form of paragraph (6.9)

```
def NTBINARYCONV(a):
    D=[]
    w=0
    while w<>1:
        r=a%2
        w=long(int(a/2))
        D.append(r)
        a=w
    D.append(w)
    return(D)
```

6.9 Function *NTEXPMOD(b,e,n))*

Finds the value of the modular exponential with base *b*, power *e,* and module *n*; i.e.:

$$b^e \bmod(n) \tag{05}$$

It uses the accelerated process for modular exponentiation, employing the binary expression of the exponent to increase its effectiveness.

The output displays the value of the modular exponential, number of iterations, and execution time.

```
def NTEXPMOD(b,e,n):
    p=1
    ti=0
    tf=0
    i=0
    if e==0:
        return(1)
    E=NTBINARYCONV(e)
    A=b
    ti=clock()
    if E[0]==1:
        p=b
    for t in range(1,len(E)):
        A=(A*A)%n
        if E[t]==1:
            p=(A*p)%n
        i+=1
    tf=clock()
```

```
    return(p,i,tf-ti)
```

6.10 Function *NTQR(a,n) "Quadratic Residues"*

Finds whether the positive integer *a* is a quadratic residue module *n* or not. As opposed to the case of the function *NTLEGENDRE*, it is not necessary for *n* to be a positive prime as it tracks all possible squares mod *n*, through sums of odd numbers. It should be noted that it could be solved by the use of the *Jacobi* process, but it requires the factoring of *n* for all to be prime and applying the product of Legendre symbols for each prime factor. However, if *n* needs no factoring, this function is more convenient.

```
def NTQR(a,n):
      i=0
      ti=0
      tf=0
      r=0
      if a%n==0:
          return(0,0)
      ti=clock()
      if a<0:
          a=a+n
    while i<(n-1)/2 and r<>a:
        i+=1
        r=(r+2*i-1)%n
        if r==a:
              tf=clock()
              return(1,i,tf-ti)
              break
      if r<>a:
          tf=clock()
          return(-1,i,tf-ti)
```

6.11 Functions *NTCTR(a1,m1,a2,m2)* "Chinese Theorem Residues"

Finds number *s* and number *m* (the product of m_1 and m_2) such that it complies with the two linear equations of congruence: *s* is congruent to a_1 module m_1 and congruent to a_2 module m_2.

It should be noted that the Chinese Remainder Theorem can solve a quantity r of equations as long as m_r are co primes among each other. In order not to determine beforehand a number *r* to apply in the library, though less efficient, the system of equations is solved in pairs. The user should perform repeated calls to this function to complete the full system of modular linear equations, hence finding *s* and *m*.

```
def NTCTR(a1,m1,a2,m2):
    m=0
```

```
M1=0
M2=0
N1=0
N2=0
s=0
ti=clock()
m=m1*m2
M1=m/m1
M2=m/m2
N1=NTINVERMOD(M1,m1)
N2=NTINVERMOD(M2,m2)
s=(a1*M1*N1[0]+a2*M2*N2[0])%m
tf=clock()
return(s,m,tf-ti)
```

7 CONCLUSIONS

Information security (IS) protects information in storage and processing systems, as well as in communications. It offers confidentiality, availability, and integrity. Hence, the academic field is focusing increasingly on this discipline. Progressively, graduate programs have included courses dealing with this subject matter.

NUMTHEORYPY library was created for application in educational settings and software development. It offers educators, students and developers the ability to run functions commonly used in Number Theory with direct application to cryptographic techniques.

Regarding educational applications, each function provides, besides its resulting value, additional information; for example, number of iterations and execution times used in finding the value. Such information provides the means for efficiency comparisons among different methods and algorithms, fostering improved teaching techniques.

Software developers can benefit from this library by saving time and gaining program readability.

The library was developed completely in PYTHON language and within GNU license of open source software, thus providing free access. In addition, due to this open source nature, users can read and verify content as well as change the code if desired.

8 Further Work

Additional functions could be added in an improved version: other factoring methods, primality test, and discrete algorithm, among others.

The idea is to provide a starting tool for evaluation in the quality of teaching. Drawing from the learning process of the educator using the tool, later versions could include additional functions.

Similarly, it can be evaluated by software developers in the creation of new IS software and later include the additional functions they may require.

9 References

1. 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.
2 Chong, S. Cryptographic teachings tools. School of Computer Science and Software Engineering. June, 2003.
3 Olejar, D; Stanek M. Some Aspect of Cryptology Teaching. Department of Computer Science, Comenius University, 2001.
4 Dulal, K. Teaching Cryptography in an applied computing program. Journal of Computing Sciences in Colleges. Volume 21 Issue 4. April 2006.
5 Baliga, A; Boztas, S. Cryptography in the classroom using Maple. Department of Mathematics. Royal Melbourne Institute of Technology University. Melbourne, 2001.
6 Koblitz, N. Cryptography as Teaching Tool. Criptologia. Vol 21. No 4. 1991
7 Saunders, Bonnie. and Beissinger, Janet. "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.
8 http://www.python.org/