Python based implementation of a chaos based cryptosystem using external key

Madhu Sharma
Assistant Professor
DIT, Dehradun

Outline of the presentation

- *My* introduction to Python
- Introduction to Cryptography and Chaos
- The paper's core
- Results
- Conclusions
- References

My Introduction to Python

- Internet
- Need for a commonly used scripting language
- Open source/free
- Lots of documentation
- Well connected
- Mature

Introduction to Cryptography

- Ever increasing trend towards virtualization
- Various valuable transactables like information, education & money
- Continuous research towards newer cryptographic implementations
- Rapid progress in Hardware
- Rapid progress in Software

Introduction to Chaos

- Pseudo-random number generation $X_{n+1} = f(X_n)$
- For e.g., the logistic map: $X_{n+1} = pX_n(1 - X_n), p = 3.99$
- High sensitivity to initial conditions & system parameters
- Wide variety of chaotic maps
- Multidimensional chaotic maps
- Discrete/continuous maps
- Applications to cryptography [1-3]

Introduction to the Chaos based Cryptosystem - overview

- Two skew-tent chaotic maps [2]:

$$t(x) = x/p, when\ 0 \leqslant x < p$$
$$t(x) = (1-x)/(1-p), when p \leqslant x \leqslant 1$$

- The two maps differ in terms of the parameter *p* and the number of iterations for each new random number
- A 128-bit key is used to initialize the chaotic maps and to specify other operational parameters

Introduction to the Chaos based Cryptosystem – contd.

- Plaintext is broken into 8-byte blocks as:

$$P=P_1 P_2 P_3 \ldots\ldots P_m$$

The last block may be smaller

- The resulting ciphertext is similarly obtained in the form of 8-byte blocks:

$$C=C_1 C_2 C_3 \ldots\ldots C_m$$

- The two skew-tent maps have p = 0.79 & 0.39 resp.
- The first map has a variable no. of iterations T_0 and the second map has only one iteration per step

Introduction to the Chaos based Cryptosystem – contd.

- The 128-bit key is broken into four 32-bit parts:

$$K = A_1 A_2 B_1 B_2$$

- The initial value of T_0 is an 8-bit integer, obtained as,

$$T_0 = (lowest\ 4\ bits of A_2)(highest four bits of B_1)$$

- The initial values X_0 and Y_0 are:

$$X_0 = 0.((A_1 A_2)\ XOR\ (B_1 B_2))$$

$$Y_0 = 0.(A_1 B_2)$$

Introduction to the Chaos based Cryptosystem – contd.

- Using these initial values (X_0, Y_0 and T_0), the two tent-maps are run to the updated - X_b and Y_b
- The next-to-decimal 64 bits of X_b and Y_b are used as random integers X_{bz} and Y_{bz} in the further operations
- Block-wise encryption and decryption are defined as:

$$C_b = s\left(P_b\right) XOR C_{b-1} XOR X_{bz} XOR Y_{bz}$$

$$P_b = s^{-1}\left(C_b XOR C_{b-1} XOR X_{bz} XOR Y_{bz}\right)$$

Introduction to the Chaos based Cryptosystem – contd.

- Here, $C_0 = A_2 B_1$
- The *s*(..) operator consists of the following 2 steps:

1. Each byte of the block P_b is rotated to a new position in the block P_b' – the rotation number being the byte-wise sum of the block modulo the block-size
2. The bits of of the resulting block P_b' are swapped left-right – the swap location being the 1-bit population count of $A_1 B_2$

Introduction to the Chaos based Cryptosystem – contd.

- The tent-maps' iterations are re-initialized as follows: $X_{bz} = C_{b-1} XOR X_{(b-1)z} XOR B_2 A_1$

$$Y_{bz} = C_{b-1} XOR Y_{(b-1)z}$$

$$T_b = z(P_{b-1}) XOR T_{(b-1)}$$

- The $z(..)$ operator being the byte-wise XOR operator
- The above steps are repeated until the complete plaintext/ciphertext is exhausted

Application of gmpy

- GMP – GNU Multi-precision library
- gmpy – the *friendly* Python interface to the gmp
- Example code:

```
import random, gmpy
mykey1 = gmpy.mpz(random.randrange(0, 2**128 – 1))
myB = mykey1.lowbits(64)
myA = mykey1 - myB
myA = myA.__rshift__(64)
myT0 = myA.lowbits(4)
myT0 = myT0.__lshift__(4)
myT0 = myT0 + myB.__rshift__(60)
```

The commands

The encryption:

```
./myxiang08v2.py e inaroundcampus.pdf goodpw1
```

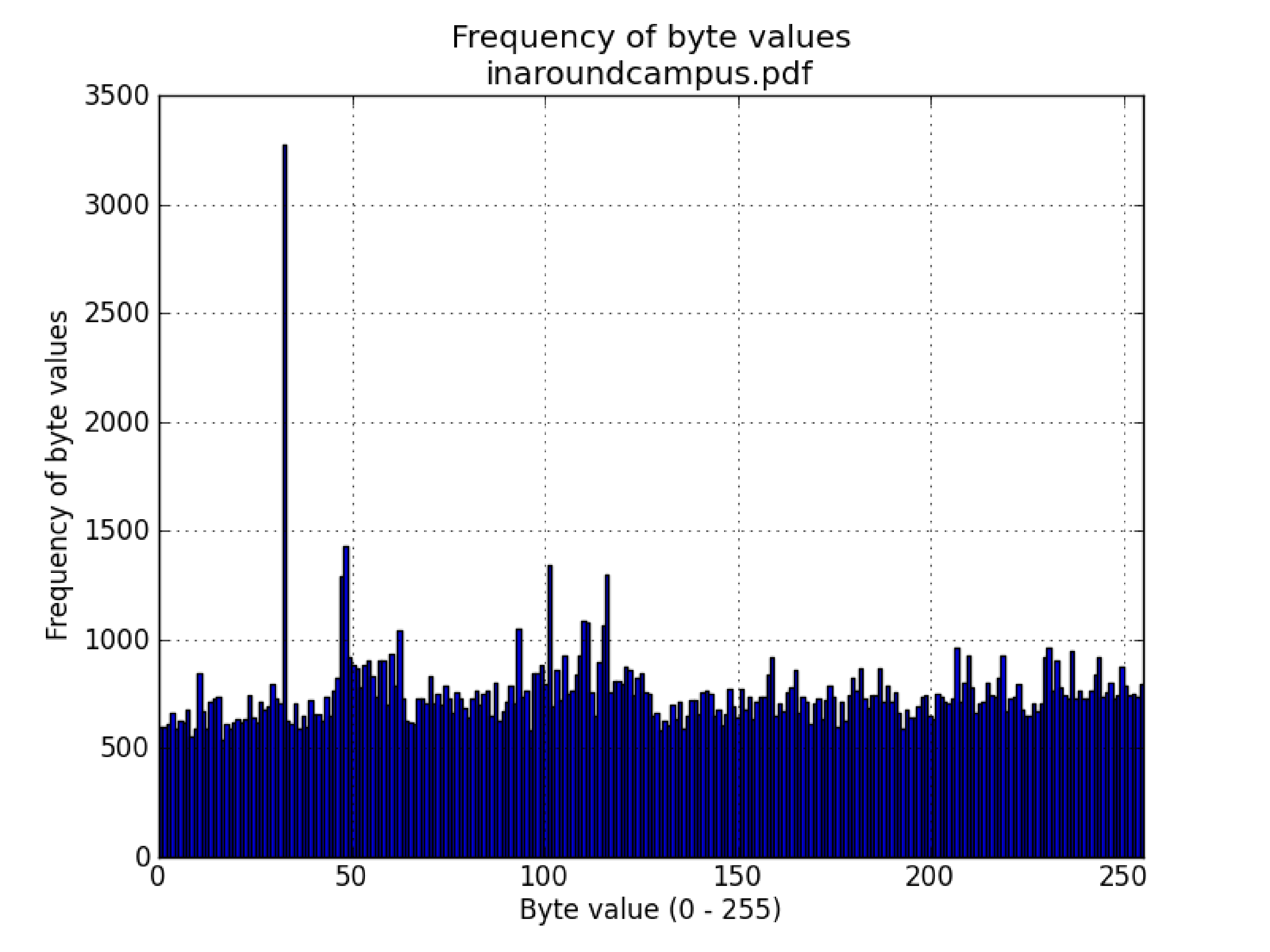
The decryption:

```
./myxiang08v2.py d einaroundcampus.pdf goodpw1
```

The histogram:

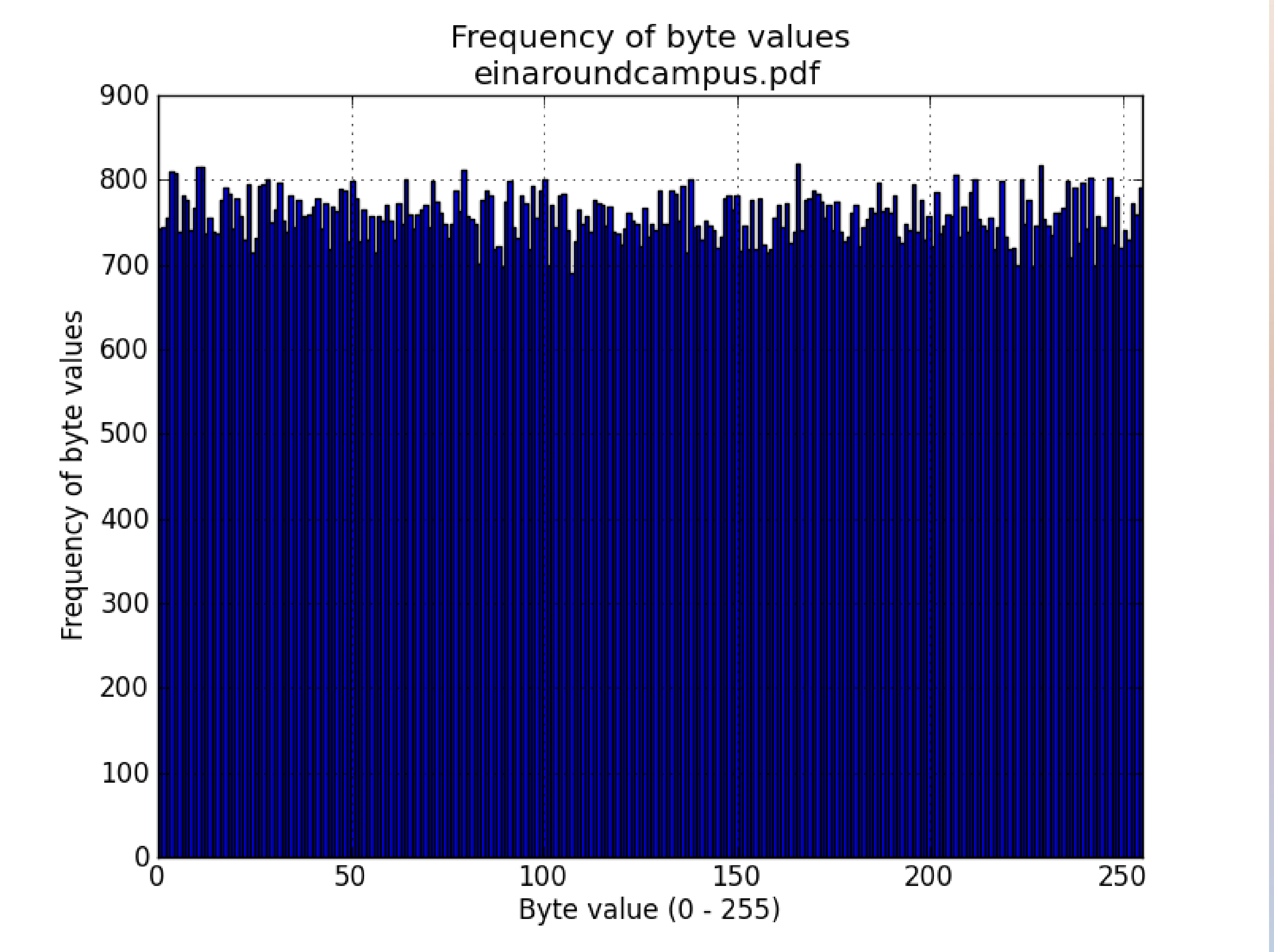
```
./myhistogram.py inaroundcampus.pdf
./myhistogram.py einaroundcampus.pdf
```

Results



Frequency of byte values
inaroundcampus.pdf
Frequency of byte values
Byte value (0 - 255)
0
500
1000
1500
2000
2500
3000
3500
0
50
100
150
200
250

Results – contd.



Conclusions

- Python, as often stated, was found to be extremely simple and user-friendly
- gmpy – provided a friendly interface to an otherwise complicated C-library
- The actual Python code was extremely short in comparison to the C-code
- The code was easily understandable and reusable
- Several straightforward extensions are possible – biometrics, audio/video encryption etc.

References

[1] Muhammad Usama , Muhammad Khurram Khan , Khaled Alghathbar and Changhoon Lee, *Chaos-based secure satellite imagery cryptosystem*, Computers and Mathematics with Applications 60 (2010) 326–337.

[2] Tao Xiang, Kwok-wo Wong and Xiaofeng Liao, *An improved chaotic cryptosystem with external key*, Communications in Nonlinear Science and Numerical Simulation 13 (2008) 1879-87.

[3] Pareek NK, Patidar V, Sud KK. *Cryptography using multiple one-dimensional chaotic maps*, Communications in Nonlinear Sciences and Numerical Simulations, 10 (2005), 715-23.

THANKS!

Have a nice day!