Python for Security

Cryptography

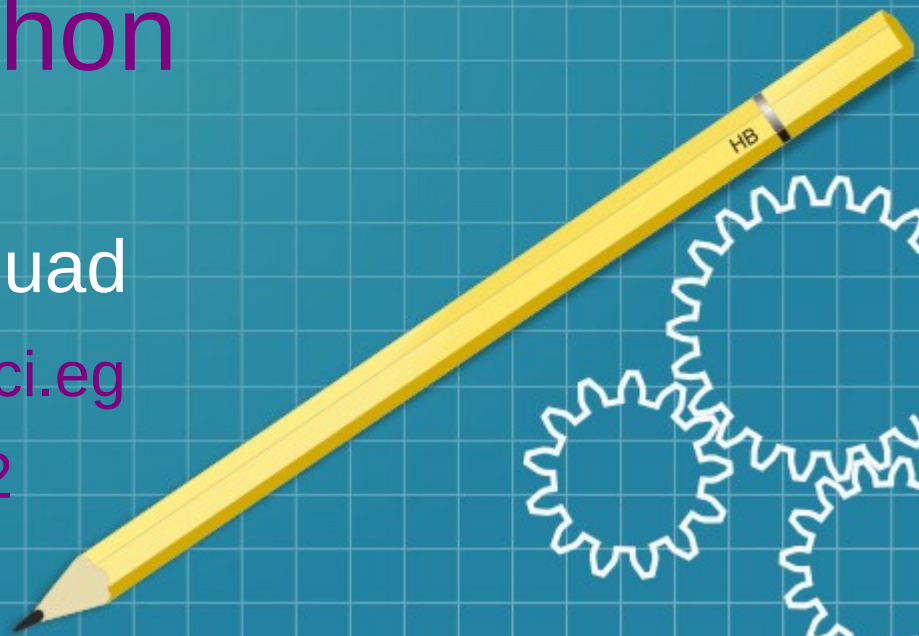
Using Python

by

Dr. Mona Fouad

mfouad@nti.sci.eg

2021-2022



Course Outline

- ✔ First Day—Python Crash Course
  - ✔ Programming Overview
  - ✔ Python Installations
  - ✔ Python Programming Concepts and Hands-on

Course Outline (cont.)

- Second Day—Cryptography using Python
  - Cryptography Overview
  - Caesar Ciphering
  - Modular Arithmetic
    - Modular Inverse
    - Greatest Common Divisor
  - Affine Ciphering
  - Attacking Encrypted Messages

Course Outline (cont.)

- Third Day—Advanced Topics
  - Vigenère and one-time pad ciphers
  - Base64 CODEC
  - Hashing and password verification
  - DES and AES Cryptography
  - Public Key and Digital Signature
- References

Review

- Solve the Google Assessment you have received in your email.

Cryptography

- Cryptography—it is the process of converting a readable document to a non-readable document, to hide the content of the original document while transferring it over shared communication channel(s).
- A secrete key—it is the key that the cryptography algorithm use to encrypt or decrypt a message.
- Cryptography algorithm—it is the procedure followed to apply the secrete key to a plain-text (or encrypted-text) to transform it into an encrypted-text (or decrypted plaint-ext).

Cryptography Algorithm Categories

- Symmetric—same key for encryption and decryption processes.
  - Stream cipher (Ceasar, Affine, ...)
  - Block cipher (DES, 3DES, AES, ...)
- Asymmetric—different keys for encryption and decryption processes. (Public key cryptography)
- Protocol-Based Cryptography

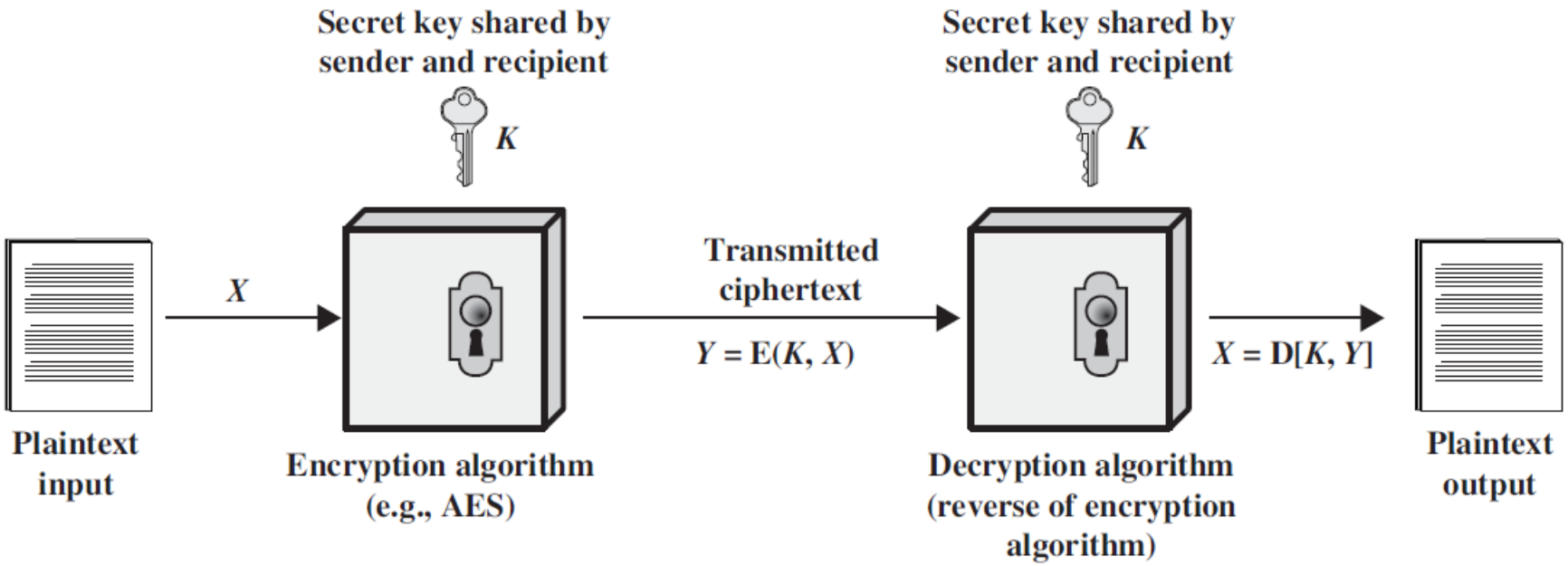
Cryptography (cont.)

- Attacks strategies—there are two strategies for attacking encrypted messages.
  - The first is to try all possible keys (Brute-Force attack).
  - The second is to analyze the encrypted message and detect patterns to be the guide for penetration (Cryptanalysis).

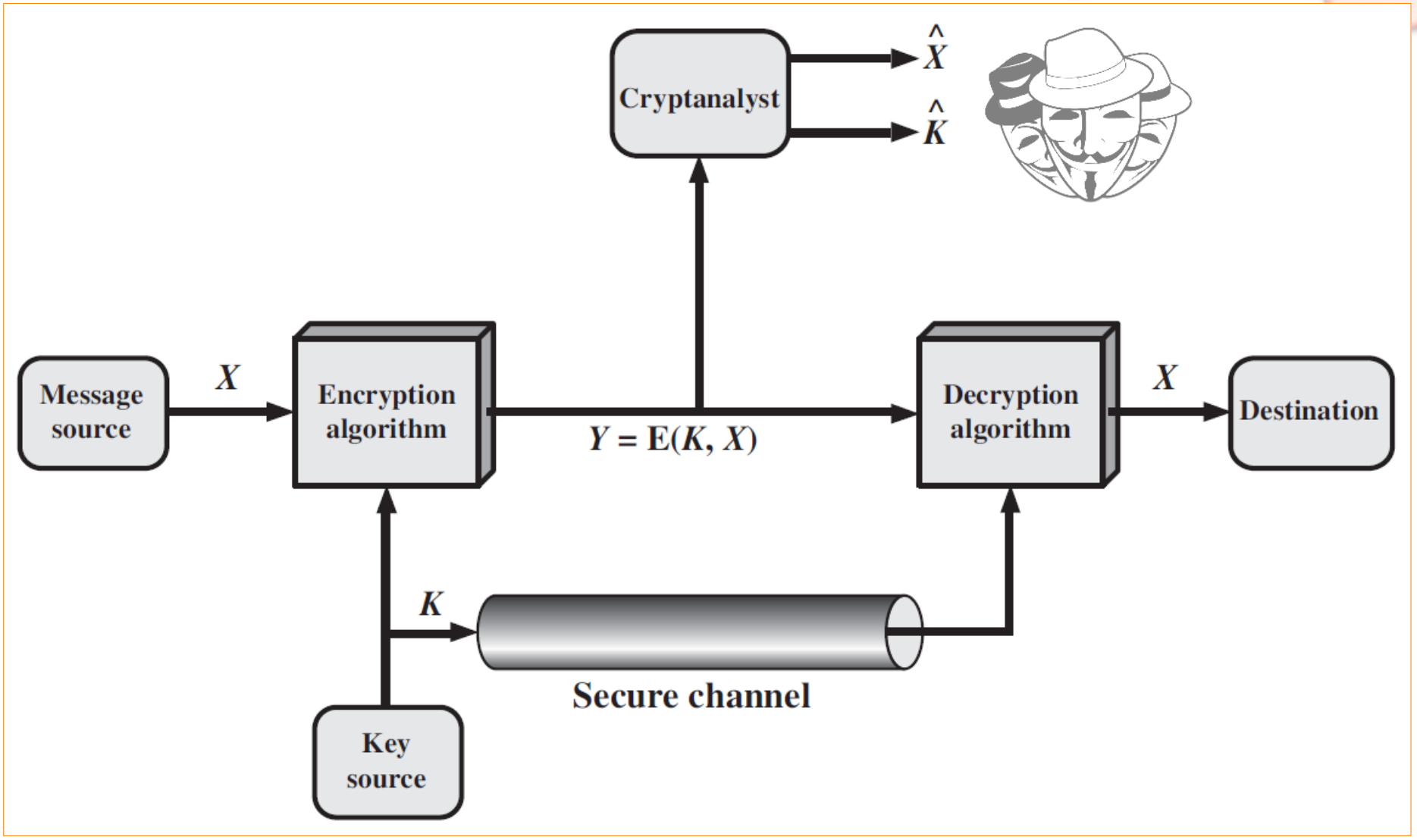
Steganography

- It is another form of cryptography
- Its philosophy depend on hiding a secret message within a larger one in such a way that others cannot discern the presence of the contents of the hidden message.
- Example of steganogrphy—hiding secrete messages into images (or video!!)

Symmetric Cryptography



Symmetric Cryptography Scenario



Ciphering

- Ciphering—is a simple symmetric cryptography technique that depends on substituting each character of a plain-text by another one to encrypt the original message.
- The reverse cipher—is the simplest ciphering way to encrypt/decrypt a text message by just reversing it...
- **Caesar Cipher** (**additive cipher**)—works by substituting each letter of a message with a new letter after shifting the alphabet over.
  - The same key is used for both encryption and decryption processes...so it is symmetric/asymmetric cryptography?
  - The key is limited: 0 > key < length of the symbol set

Lab—Saesar Ciphering

- Additive Cipher
  - see “caesar.ipynb”

    Notice the use of the following:
    - import statement
    - constants and variables
    - for loops
    - if , else , and elif statements
    - “in” and “not in” operators
    - find() string method
    - wraparound the symbol set

Pyperclip is a cross-platform Python module for copy and paste clipboard functions. It works with Python 2 and 3.

Assignments

- Using *caesarCipher.py*, encrypt the following sentences with the given keys:
  - '"You can show black as white by argument," said Filby, "but you will never convince me."' with keys 8 and 23, -15
  - Try encrypting the message with key 77

    Why ?????

    Suggest a solution ...

Multiplicative Cipher

- It is similar to the Caesar cipher but encrypts using multiplication rather than addition.
- For multiplicative cipher, if the key is equal 11, here's the mapping you would end up with:

'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890 !?.'

'ALWhs4ALWhs4ALWhs4ALWhs4ALWhs4ALWhs4ALWhs4ALWhs4ALWhs4ALWhs4ALWhs4'

- Notice that the symbols A, G, and M all encrypt to the same letter, A.

Multiplicative Cipher (cont.)

- The multiplicative cipher has two concepts to be concerned: the **Modular inverse** and the **Greatest Common Divisor (GCD)** of the proper key.
- Each of the two concepts uses the **modular operator** ***%***
- For the multiplicative cipher, the key and the size of the symbol set must be ***relatively prime*** to each other.
- ***Two numbers are relatively prime (or co-prime) if their Greatest Common Divisor is 1.***

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
Key = 2	2	4	6	8	10	12	14	16	18	20	22	24	26	2	4	6	8	10	12	14	16	18	20	22	24	26
	B	**D**	**F**	**H**	**J**	**L**	**N**	**P**	**R**	**T**	**V**	**X**	**Z**	**B**	**D**	**F**	**H**	**J**	**L**	**N**	**P**	**R**	**T**	**V**	**X**	**Z**
Key = 3	3	6	9	12	15	18	21	24	1	4	7	10	13	16	19	22	25	2	5	8	11	14	17	20	23	26
	C	**F**	**I**	**L**	**O**	**R**	**U**	**X**	**A**	**D**	**G**	**J**	**M**	**P**	**S**	**V**	**Y**	**B**	**E**	**H**	**K**	**N**	**Q**	**T**	**W**	**Z**
Gcd(3,26)	1																									

Greatest Common Divisor (GCD)

- GCD—is the greatest common factor among positive numbers.
- Factorization is used to find the GCD of two numbers.
- Example:
  - gcd(24, 30) = 6
    - Factors of 24 are: 1, 2, 3, 4, **6** ,8, 12, 24
    - Factors of 30 are: 1, 2, 3, 5, **6**, 10, 15, 30

Greatest Common Divisor (cont.)

- Euclid's Algorithm for Finding the GCD
  - **Given two integers *0 < b < a*, make *a* repeated division that eventually terminate with a zero remainder:** $\frac{a}{b} = q + \frac{r}{b}$
    - Where q is the quotient, and the remainder ***r*** satisfies ***0 ≤ r < b***. If we clear fractions, this is the equation ***a = bq + r***.

gcd(24, 30):
a = 30
b = 24

30 = 24 (1) + 6
24 = 6 (4) + 0
gcd(24, 30) = 6

$$a = bq_1 + r_1, 0 < r_1 < b,$$

$$b = r_1q_2 + r_2, 0 < r_2 < r_1,$$

$$r_1 = r_2q_3 + r_3, 0 < r_3 < r_2,$$

$$\cdots$$

$$r_{j-2} = r_{j-1}q_j + r_j, 0 < r_j < r_{j-1}$$

$$r_{j-1} = r_jq_{j+1}.$$

- **The greatest common divisor *gcd(a,b)* of *a* and *b* is r_j , the last nonzero remainder in the division process.**

Greatest Common Divisor (cont.)

- Example 1. Find the gcd of 42823 and 6409.

  42823 = 6409(6) + 4369

  6409 = 4369(1) + 2040

  4369 = 2040(2) + 289

  2040 = 289(7) + 17

  289 = 17(17) + 0

  Therefore gcd(42823, 6409) = 17.

- See the “gcd.py”

```
def gcd(a, b):
    while a != 0:
        a, b = b % a, a
    return b
```

Notice that: 0 < a < b

Quiz

- What is the GCD of 10 and 15 ? using factorization
- The GCD of 17 and 31 is 1. Are 17 and 31 relatively prime?
- Why aren't 6 and 8 relatively prime?
- Exercise: Find the gcd's of the following pairs of numbers using Euclid's Algorithm.
  - 7469 and 2464
  - 2689 and 4001

Modulo Operator

- Mathematical operation based on modulo operator is called modular arithmetic or clock arithmetic.
- The modulo operator ***%*** is used in cryptography to wraparound the symbol set.
  - Example: if the clock is ***15***, where is the arrow on the clock?

15 % 12 = 3

Modular Inverse

- *If **modular operator** is used for **encryption**, **modular inverse** should be used for **decryption**.*
- Modular inverse ***i*** for two numbers ***a***, ***m*** is

$$(a * i) \% m = 1$$

- Modular inverse is a brute-force process…
- The brute-force approach starts by testing the integer 1, and then 2, and then 3, and so on.

Modular Inverse (cont.)

- Example—The modular inverse of 5 mod 7 would be some number i where (5 * i) % 7 is equal to 1.
- You can brute-force this calculation like this:

  1 isn't the modular inverse of 5 mod 7, because (5 * 1) % 7 = 5.

  2 isn't the modular inverse of 5 mod 7, because (5 * 2) % 7 = 3.

  3 is the modular inverse of 5 mod 7, because (5 * 3) % 7 = 1.
- This process is time-consuming for large keys.
- Instead, use Euclid's extended algorithm to find the GCD of two numbers.

Euclid's Extended Algorithm

- Given two integers 0 < b < a, gcd(a, b) = r_j

a = bq + r
r = a - bq

$$r_1 = a - bq_1,$$
$$r_2 = b - r_1 q_2,$$
$$r_3 = r_1 - r_2 q_3,$$
$$\dots$$
$$r_{j-1} = r_{j-3} - r_{j-2} q_{j-1}$$
$$r_j = r_{j-2} - r_{j-1} q_j.$$

- Then, in the last of these equations , replace r_{j-1} with its expression in terms of r_{j-3} and r_{j-2} from the equation immediately above it. Continue this process successively, replacing r_{j-2} , r_{j-3} , ... until you obtain the final equation $r_j = ax + by$,
- In the special case that gcd(a, b) = 1, the integer equation reads 1 = ax + by. Therefore we deduce 1 ≡ by mod a, so that (the residue of) y is the multiplicative inverse of b, mod a.

Euclid’s Extended Algorithm (cont.)

- Example: Find the multiplicative inverse of 8 mod 11, using the Euclid’s extended algorithm.

$$(8 * ?) \% 11 = 1$$

Euclid's Extended Algorithm (cont.)

- Solution. We'll organize our work carefully. We'll do the Euclidean Algorithm in the left column It will verify that gcd(11, 8) = 1. Then we'll solve for the remainders in the right column, before back solving:

$$11 = 8(1) + 3 \mid 3 = 11 - 8(1)$$

$$8 = 3(2) + 2 \mid 2 = 8 - 3(2)$$

$$3 = 2(1) + 1 \mid 1 = 3 - 2(1)$$

$$2 = 1(2) \quad \mid$$

- Now reverse the process using the equations on the right.

$$1 = 3 - 2(1)$$

$$1 = 3 - (8 - 3(2))(1) = 3 - (8 - (3(2)) = 3(3) - 8$$

$$1 = (11 - 8(1))(3) - 8 = 11(3) - 8(4) = 11(3) + 8(-4)$$

- Therefore $1 \equiv 8(-4) \bmod 11$, or if we prefer a residue value for the multiplicative inverse, $1 \equiv 8(7) \bmod 11$.

Euclid's Extended Algorithm (cont.)

Let us examine the previous example. Find the modular inverse of 5 mod 7.

Find the gcd(7,5)	Modular inverse
7 = 5(1) + 2	2 = 7 - 5(1)
5 = 2(2) + 1	1 = 5 - 2(2)
2 = 1(2) + 0	Therefore: 1 = 5 – 2 (7 – 5(1)) 1 = 5 + 7(2) + 5(2) 1 = 5(3) + 7(2) Then 3 is the modular inverse of 5%7
gsd(5,7) = 1	
5, 7 are co-prime	

Euclid’s Extended Algorithm (cont.)

- As long as the two numbers ***a*** and ***m*** you pass to Euclid’s function are relatively prime, you will receive the modular inverse of a***:*** (a*i)%m=1.
- see “findmodinverse.ipynb”.
  - “gcd.py” should be in the same folder
- Examine the program by yourself.
- For further explanation of the Euclid’s algorithm, see “Euclid.pdf”.

Euclid's Extended Algorithm (cont.)

In [201]:
```
import gcd
def findModInverse(a, m):
    if gcd.gcd(a, m) != 1:
        return None # No mod inverse if a & m aren't relatively prime.
    u1, u2, u3 = 1, 0, a
    v1, v2, v3 = 0, 1, m
    while v3 != 0:
        q = u3 // v3 # Note that // is the integer division operator.
        #print(u3, ' = ', v3, '(', q, ')', ' + ', u3%v3)
        print(u3%v3, ' = ', u3, ' - ', v3, '(', q, ')')
        v1, v2, v3, u1, u2, u3 = (u1 - q * v1), (u2 - q * v2),\
                                 (u3 - q * v3), v1, v2, v3
    return u1%m
```

In [202]:
```
print(findModInverse(7, 5))
```

```
2  =  7  -  5 ( 1 )
1  =  5  -  2 ( 2 )
0  =  2  -  1 ( 2 )
3
```

Assignment

- Apply *findModInverse()* to display the following

```
The steps to find the Modular inverse of 56 , 15 are:
11  =  56  -  15 ( 3 )
4  =  15  -  11 ( 1 )
3  =  11  -  4 ( 2 )
1  =  4  -  3 ( 1 )
0  =  3  -  1 ( 3 )

The Modular inverse is:  11
```

Summary and Simple Explanation

- if ***n*** is the length of the character set.
- and ***e*** is the encryption key (a random integer).
- ***e, n*** should be co-prime numbers satisfying the rule that ***gcd(e,n) = 1*** following the Euclid's Algorithm.
- The decryption key is a **brute-force process** based on the rule ***(e * d) % n = 1***
- Instead Euclid's extended Algorithm is used to estimate the modular inverse.

Quiz

- Is gcd(a,m) equivalent to gcd(m,a)?

  How?

  Why?
- Based on the previous question, is there a difference between the gcd() implementation based on Euclid's algorithm and the traditional factorization method?

Affine Cipher

- Affine cipher is actually the multiplicative cipher combined with the Caesar cipher.
- It needs two keys: one for the multiplicative cipher and another for the Caesar cipher.
- How many available keys can Affine cipher has?

Affine Cipher (cont.)

- How Many Keys Can the Affine Cipher Have?
  - Each of the additive and multiplicative key is limited to the size of the symbol set.
  - When you multiply 66 possible Key A keys by 66 possible Key B keys,the result is 4356 possible combinations.
  - Then when you subtract the integers that can't be used for Key A because they're not relatively prime with 66, the total number of possible key combinations for the Affine cipher drops to 1320.

Lab—Affine Cipher

- see “affinecipher.ipynb”
- simple run

```
Key:  2894
Key A and Key B:  43 56
Encrypted text:
"5QG9ol3La6QI93!xQxaia6faQL9QdaQG1!!axQARLa!!A
-5!1RQP36ARu
Full encrypted text copied to clipboard.
```

- There is an unused function. Which one?
- **Assignment**: randomize the selection of the additive and the multiplicative keys. Loop the selection until the keys are valid.

Quizzes

- The affine cipher is the combination of which two other ciphers? _____ and _____
- What is a tuple? How is a tuple different from a list?
- If Key A is 1, why does it make the affine cipher weak?
- If Key B is 0, why does it make the affine cipher weak?
- Does the main() function of affineCipher.py get called if another program runs import affineCipher?

Hacking Encrypted Messages

- Hacking the Caesar cipher using brute-force technique.
  - See the “CaesarHacker.ipynb”
    - the brute-force technique
    - the range() function
    - string formatting (string interpolation)

Detect English

- Hacking encrypted messages using the CryptAnalysis strategy by detecting English phrase
- It is also used to validate the decrypted “English” message.
  - See the “detectEnglish.ipynb”
    - use of dictionary { }
  - Evaluate the program

Lab—Hacking the Affine Algorithm

- Because Affine cipher is limited to only a few thousand keys. It is easily possible to perform a brute-force attack against it.
- See “affineHacker.ipynb”
- Sample Run

```
Tried Key 2890... ("okCOMPUTERkWOULDkDESERVEkTOkBEkCALLEDkI)
Tried Key 2891... ("!8Zljmrqbo8tlria8abpbosb8ql8Yb8ZXiiba8f)
Tried Key 2892... ("UQw970A.y!QC9A6xQxy?y!ByQ.9QvyQwu66yxQ3)
Tried Key 2893... ("rnFRPSXWHUnZRXOGnGHVHUYHnWRnEHnFDOOHGnL)
Tried Key 2894... ("A computer would deserve to be called i)

Possible encryption hack:
Key: 2894
Decrypted message: "A computer would deserve to be called intellige
that it was human." -Alan Turing

Enter D if done, anything else to continue hacking:

> d
key is:  2894
Hacking time is:  8.77  seconds
Copying hacked message to clipboard:
"A computer would deserve to be called intelligent if it could dece
n." -Alan Turing
```

Lab—Hacking the Affine Algorithm (cont.)

- Hacking process could be stopped at any time: hackAffine(message)
- The Total Number of Possible Keys is equal to (length of the symbol set) ** 2
- run the program by yourself...

Programming Tasks

Wrap-around the developed Caesar and Affine ciphering programs (encoding/decoding) and do perform your touch to them...

Thank You