

# P R A T I K U M

## KEAMANAN SISTEM INFORMASI DAN JARINGAN

### ONE TIME PAD (OTP) CIPHER

Dimas Oktavian Prasetyo

S1 Sistem Informasi | Universitas Respati Yogyakarta |  Email : [dimasoktavianprasetyo@gmail.com](mailto:dimasoktavianprasetyo@gmail.com)  
 Github : [www.github.com/dimasjuniorseven](https://www.github.com/dimasjuniorseven)

## PENGANTAR

OTP Cipher, atau **One-Time Pad Cipher**, adalah salah satu metode enkripsi tertua dan paling aman yang pernah ditemukan. Teknik ini bekerja dengan cara menggabungkan pesan asli (plaintext) dengan kunci acak yang panjangnya sama persis dengan pesan tersebut. Setiap karakter dalam pesan dikodekan dengan karakter dari kunci menggunakan operasi matematika sederhana, seperti XOR atau MOD. Karena kunci hanya digunakan satu kali dan bersifat acak sepenuhnya, OTP menjanjikan keamanan mutlak bahkan tidak bisa dipecahkan oleh komputer tercepat sekalipun, **asal digunakan dengan benar**.

Namun, keunggulan keamanan OTP juga datang dengan tantangan besar. Kunci yang digunakan harus benar-benar acak, tidak boleh dipakai ulang, dan harus dirahasiakan serta didistribusikan secara aman kepada penerima. Inilah yang membuat OTP kurang praktis digunakan dalam kehidupan sehari-hari atau komunikasi modern berskala besar. Meski begitu, OTP tetap menjadi simbol dari keamanan sempurna dalam dunia kriptografi, dan sering dijadikan referensi atau inspirasi dalam pengembangan algoritma enkripsi lainnya.

## IMPLEMENTASI OTP CIPHER DENGAN PYTHON

1. Disajikan script dan hasil run berikut ini :

```
[4]: # Contoh dari Modul LMS

import random
import string

def generate_key(length):
    """Generate a random key of uppercase letters."""
    return ''.join(random.choice(string.ascii_uppercase) for _ in range(length))

def text_to_numbers(text):
    """Convert A-Z to 0-25"""
    return [ord(c) - ord('A') for c in text]

def numbers_to_text(numbers):
    """Convert 0-25 to A-Z"""
    return ''.join(chr(n + ord('A')) for n in numbers)

def otp_encrypt(plaintext, key):
    plain_nums = text_to_numbers(plaintext)
    key_nums = text_to_numbers(key)
    cipher_nums = [(p + k) % 26 for p, k in zip(plain_nums, key_nums)]
    return numbers_to_text(cipher_nums)

def otp_decrypt(ciphertext, key):
    cipher_nums = text_to_numbers(ciphertext)
    key_nums = text_to_numbers(key)
    plain_nums = [(c - k + 26) % 26 for c, k in zip(cipher_nums, key_nums)]
    return numbers_to_text(plain_nums)

def clean_input(text):
    """Uppercase and remove non-letter characters."""
    return ''.join(filter(str.isalpha, text.upper()))

if __name__ == "__main__":
    print("== One-Time Pad Cipher ===")
    mode = input("Mode (encrypt/decrypt): ").strip().lower()

    if mode == "encrypt":
        plaintext = clean_input(input("Enter plaintext: "))
        key = generate_key(len(plaintext))
        ciphertext = otp_encrypt(plaintext, key)
        print("\n--- Encryption Result ---")
        print("Plaintext : ", plaintext)
        print("Key      : ", key)
        print("Ciphertext: ", ciphertext)

    elif mode == "decrypt":
        ciphertext = clean_input(input("Enter ciphertext: "))
        key = clean_input(input("Enter key (same length): "))
        if len(ciphertext) != len(key):
            print("Error: Key length must match ciphertext length.")
        else:
            plaintext = otp_decrypt(ciphertext, key)
            print("\n--- Decryption Result ---")
            print("Ciphertext: ", ciphertext)
            print("Key      : ", key)
            print("Plaintext : ", plaintext)

    else:
        print("Invalid mode. Use 'encrypt' or 'decrypt'.")
```

## Hasil Run :

```
== One-Time Pad Cipher ===
Mode (encrypt/decrypt): encrypt
Enter plaintext: DIMASOKTAVIANPRASETYO

--- Encryption Result ---
Plaintext : DIMASOKTAVIANPRASETYO
Key      : DAHOIZUKINBDAJMPMMNIF
Ciphertext: GITOANEDEIIJDNYDPEQGGT
```

## Penjelasan Source Code OTP Cipher

Program ini menerapkan **algoritma OTP (One-Time Pad)** untuk melakukan enkripsi dan dekripsi pesan menggunakan huruf A-Z. Prosesnya dimulai dengan **membersihkan input** (mengubah huruf kecil ke besar dan menghapus karakter non-huruf), lalu **mengubah huruf menjadi angka (0-25)** agar bisa diproses secara matematis. Untuk **enkripsi**, tiap huruf plaintext dijumlahkan dengan huruf pada kunci, sedangkan **dekripsi** dilakukan dengan mengurangkan huruf ciphertext dengan kunci. Operasi ini dibungkus dengan modulus 26 agar hasilnya tetap dalam rentang A-Z. Program tersebut menyediakan dua mode:

- **Encrypt (enkripsi)**: pengguna memasukkan teks asli, lalu program akan membuat kunci acak sepanjang teks tersebut, dan menghasilkan ciphertext.
- **Decrypt (dekripsi)**: pengguna memasukkan ciphertext dan kunci (dengan panjang yang sama), lalu program mengembalikan plaintext aslinya.

Dengan kata lain, ini adalah implementasi sederhana OTP Cipher berbasis teks dengan Python.

# P R A T I K U M

## KEAMANAN SISTEM INFORMASI DAN JARINGAN

### ONE TIME PAD (OTP) CIPHER

#### TUGAS !

DIKETAHUI ARRAY ALPHABET BERIKUT INI :

<b>A = 0</b>	<b>G = 6</b>	<b>M = 12</b>	<b>S = 18</b>	<b>Y = 24</b>
<b>B = 1</b>	<b>H = 7</b>	<b>N = 13</b>	<b>T = 19</b>	<b>Z = 25</b>
<b>C = 2</b>	<b>I = 8</b>	<b>O = 14</b>	<b>U = 20</b>	
<b>D = 3</b>	<b>J = 9</b>	<b>P = 15</b>	<b>V = 21</b>	
<b>E = 4</b>	<b>K = 10</b>	<b>Q = 16</b>	<b>W = 22</b>	
<b>F = 5</b>	<b>L = 11</b>	<b>R = 17</b>	<b>X = 23</b>	

Plaintext : **GUNUNGKIDUL**

Key : **WYFXBGLIRAL**

Ciphertext : ???

**Maka OTP Ciphernya adalah ?**

\*Saya buat program python saja untuk mendemonstrasikan *step-by-step* daripada perhitungan/operasi manual yang bikin ribet dan rawan akan *human error*.

```

1 # Data
2 plaintext = ["G", "U", "N", "U", "N", "G", "K", "I", "D", "U", "L"]
3 key = ["W", "Y", "F", "X", "B", "G", "L", "I", "R", "A", "L"]
4
5 # Hasil cipher akan disimpan di list ini, di MSIB kemarin saya lebih suka memakai "Result = []"
6 ciphertext = []
7
8 # Proses enkripsi >> (ord() dipakai buat ngonversi karakter (huruf) jadi nilai angka Unicode-nya,
9 for huruf_plain, huruf_key in zip(plaintext, key): # Nilai dibambil dari variabel plaintext dan key dengan metode ZIP
10    nilai_plain = ord(huruf_plain) - ord('A') # Konversi huruf ke angka 0-25 >> ord('A') - ord('A') = maka 65 - 65 = 0
11    nilai_key = ord(huruf_key) - ord('A') # Konversi huruf ke angka 0-25 >> ord('A') - ord('A') = maka 65 - 65 = 0
12    nilai_cipher = (nilai_plain + nilai_key) % 26 # Penjumlahan mod 26
13    huruf_cipher = chr(nilai_cipher + ord('A')) # Konversi angka balik ke huruf
14    ciphertext.append(huruf_cipher) # Ini akan menyimpan huruf hasil enkripsi
15
16 # Tampilkan proses per langkah
17 print(f"\n{huruf_plain}+{huruf_key} = ({nilai_plain}+{nilai_key}) % 26 = {nilai_cipher} = {huruf_cipher}")
18
19 # Tampilkan hasil akhir
20 print(f"\nCiphertext: {''.join(ciphertext)}")
21

```

Maka didapat step-by-stepnya adalah, jadi ini meminimalisir debat dengan teman seperti di kelas praktikum kemarin, karena perhitungan komputer tidak akan salah asal codenya dibuat dengan benar :

```

PS C:\Users\ASUS>
PS C:\Users\ASUS> & C:/Users/ASUS/AppData/Local/Programs/Python/Python33/python.exe e:/Downloads/demo_otp.py
G+W = (6+22) % 26 = 2 = C
U+Y = (20+24) % 26 = 18 = S
N+F = (13+5) % 26 = 18 = S
U+X = (20+23) % 26 = 17 = R
N+B = (13+1) % 26 = 14 = O
G+G = (6+6) % 26 = 12 = M
K+L = (10+11) % 26 = 21 = V
I+I = (8+8) % 26 = 16 = Q
D+R = (3+17) % 26 = 20 = U
U+A = (20+0) % 26 = 20 = U
L+L = (11+11) % 26 = 22 = W
Ciphertext: CSSROMVQUUW
PS C:\Users\ASUS>

```

Step-by-step perhitungannya adalah sebagai berikut :

$$\begin{aligned}
 G+W &= (6+22) \% 26 = 2 = C \\
 U+Y &= (20+24) \% 26 = 18 = S \\
 N+F &= (13+5) \% 26 = 18 = S \\
 U+X &= (20+23) \% 26 = 17 = R \\
 N+B &= (13+1) \% 26 = 14 = O \\
 G+G &= (6+6) \% 26 = 12 = M \\
 K+L &= (10+11) \% 26 = 21 = V \\
 I+I &= (8+8) \% 26 = 16 = Q \\
 D+R &= (3+17) \% 26 = 20 = U \\
 U+A &= (20+0) \% 26 = 20 = U \\
 L+L &= (11+11) \% 26 = 22 = W
 \end{aligned}$$

Ciphertext: **CSSROMVQUUW**

Untuk membandingkan hasil, saya coba aplikasikan ke program yang saya buat di Jupyter Notebook dengan fitur kita bisa memasukkan plaintext dengan key apakah random atau kita yang *create manual*, berikut pengaplikasiannya :

```

[*]: import random
[*]: import string
# CopyrightDimas - Fungsi untuk membersihkan input (hapus spasi, tanda baca, ubah ke huruf besar)
def bersihkan_input(teks):
    return ''.join(filter(str.isalpha, teks.upper()))

# CopyrightDimas - Fungsi untuk generate key secara acak
def buat_key_random(panjang):
    return ''.join(random.choice(string.ascii_uppercase) for _ in range(panjang))

# CopyrightDimas - Fungsi konversi karakter ke angka 0-25
def huruf_ke_angka(huruf):
    return ord(huruf) - ord('A')

# CopyrightDimas - Fungsi konversi angka 0-25 ke karakter A-Z
def angka_ke_huruf(angka):
    return chr(angka + ord('A'))

# Mulai program
print("== Program One-Time Pad (OTP) Cipher ==\n")
print("Keamanan Sistem Informasi dan Jaringan\n")

# Input plaintext dari user
plaintext = input("Masukkan plaintext (hanya huruf): ")
plaintext = bersihkan_input(plaintext)

# Pilih mode key: random atau manual
mode_key = input("Pilih jenis key (ketik 'random' atau 'manual'): ").strip().lower()

if mode_key == "random":
    key = buat_key_random(len(plaintext))
elif mode_key == "manual":
    key = input("Masukkan key manual (panjang harus sama dengan plaintext): ")
    key = bersihkan_input(key)
    if len(key) != len(plaintext):
        print("X Yahh.. Panjang key harus sama dengan panjang plaintext dong!")
        exit()
else:
    print("X Mode key tidak dikenali bos!")

# Proses enkripsi
ciphertext = []
print("\n--- Proses Enkripsi ---")
for huruf_p, huruf_k in zip(plaintext, key):
    nilai_p = huruf_ke_angka(huruf_p)
    nilai_k = huruf_ke_angka(huruf_k)
    nilai_c = (nilai_p + nilai_k) % 26
    huruf_c = angka_ke_huruf(nilai_c)
    ciphertext.append(huruf_c)

# Tampilkan detail proses
print(f"\n{huruf_p}+{huruf_k} = ({nilai_p}+{nilai_k}) % 26 = {nilai_c} = {huruf_c}")

# Tampilkan hasil akhir
print("\n==== Hasil Akhir ===")
print("Plaintext : ", plaintext)
print("Key : ", key)
print("Ciphertext: ", ''.join(ciphertext))

```

**Hasilnya ?**

BOOM!

== Program One-Time Pad (OTP) Cipher ==

Keamanan Sistem Informasi dan Jaringan

Masukkan plaintext (hanya huruf): **GUNUNGKIDUL**  
Pilih jenis key (ketik 'random' atau 'manual'): **manual**  
Masukkan key manual (panjang harus sama dengan plaintext): **WYFXBGLIRAL**

--- Proses Enkripsi ---  
G+W = (6+22) % 26 = 2 = C  
U+Y = (20+24) % 26 = 18 = S  
N+F = (13+5) % 26 = 18 = S  
U+X = (20+23) % 26 = 17 = R  
N+B = (13+1) % 26 = 14 = O  
G+G = (6+6) % 26 = 12 = M  
K+L = (10+11) % 26 = 21 = V  
I+I = (8+8) % 26 = 16 = Q  
D+R = (3+17) % 26 = 20 = U  
U+A = (20+0) % 26 = 20 = U  
L+L = (11+11) % 26 = 22 = W

== Hasil Akhir ==  
Plaintext : **GUNUNGKIDUL**  
Key : **WYFXBGLIRAL**  
Ciphertext: **CSSROMVQUUW**

Kita akan lanjut ke case study selanjutnya yang lebih panjang !

# P R A T I K U M

## KEAMANAN SISTEM INFORMASI DAN JARINGAN

### ONE TIME PAD (OTP) CIPHER

Plaintext : AGUS MEMANCING DI SUNGAI

Key : CMRI FNPUJXLCA ZH XLFTUZ

Ciphertext : ???

Langkah pertama adalah kita hilangkan spasi. **Mengapa??**  
*Spasi, tanda baca, angka, dan karakter lain* biasanya dihilangkan sebelum enkripsi karena OTP hanya bekerja dengan alfabet (26 huruf). Kemudian tidak ada nilai numerik untuk spasi dalam skema standar  $A=0$ ,  $B=1$ , ...,  $Z=25$  seperti pada tabel yang saya buat di halaman kedua. Alasan lainnya adalah menambahkan spasi ke perhitungan justru bisa bikin hasil jadi tidak valid atau susah didekripsi balik. Jadi disini hanya huruf A-Z yang digunakan dalam proses perhitungan.

Jadi Plaintext nya adalah :

**AGUSMEMANCINGDI SUNGAI**

Serta Key nya adalah :

**CMRIFNPUJXLCAZHXLFTUZ**

Oke kita ke perhitungan :

A+C =  $(0+2) \% 26 = 2 = \mathbf{C}$   
G+M =  $(6+12) \% 26 = 18 = \mathbf{S}$   
U+R =  $(20+17) \% 26 = 11 = \mathbf{L}$   
S+I =  $(18+8) \% 26 = 0 = \mathbf{A}$   
M+F =  $(12+5) \% 26 = 17 = \mathbf{R}$   
E+N =  $(4+13) \% 26 = 17 = \mathbf{R}$   
M+P =  $(12+15) \% 26 = 1 = \mathbf{B}$   
A+U =  $(0+20) \% 26 = 20 = \mathbf{U}$   
N+J =  $(13+9) \% 26 = 22 = \mathbf{W}$   
C+X =  $(2+23) \% 26 = 25 = \mathbf{Z}$   
I+L =  $(8+11) \% 26 = 19 = \mathbf{T}$   
N+C =  $(13+2) \% 26 = 15 = \mathbf{P}$   
G+A =  $(6+0) \% 26 = 6 = \mathbf{G}$   
D+Z =  $(3+25) \% 26 = 2 = \mathbf{C}$   
I+H =  $(8+7) \% 26 = 15 = \mathbf{P}$   
S+X =  $(18+23) \% 26 = 15 = \mathbf{P}$   
U+L =  $(20+11) \% 26 = 5 = \mathbf{F}$   
N+F =  $(13+5) \% 26 = 18 = \mathbf{S}$   
G+T =  $(6+19) \% 26 = 25 = \mathbf{Z}$   
A+U =  $(0+20) \% 26 = 20 = \mathbf{U}$   
I+Z =  $(8+25) \% 26 = 7 = \mathbf{H}$

Ciphertext:

**CSLARRBUWZTPGCPPFSZUH**

--- Program One-Time Pad (OTP) Cipher ---

Keamanan Sistem Informasi dan Jaringan

Masukkan plaintext (hanya huruf): AGUS MEMANCING DI SUNGAI  
Pilih jenis key (ketik 'random' atau 'manual'): manual  
Masukkan key manual (panjang harus sama dengan plaintext): CMRI FNPUJXLCA ZH XLFTUZ

--- Proses Enkripsi ---  
A+C =  $(0+2) \% 26 = 2 = \mathbf{C}$   
G+M =  $(6+12) \% 26 = 18 = \mathbf{S}$   
U+R =  $(20+17) \% 26 = 11 = \mathbf{L}$   
S+I =  $(18+8) \% 26 = 0 = \mathbf{A}$   
M+F =  $(12+5) \% 26 = 17 = \mathbf{R}$   
E+N =  $(4+13) \% 26 = 17 = \mathbf{R}$   
M+P =  $(12+15) \% 26 = 1 = \mathbf{B}$   
A+U =  $(0+20) \% 26 = 20 = \mathbf{U}$   
N+J =  $(13+9) \% 26 = 22 = \mathbf{W}$   
C+X =  $(2+23) \% 26 = 25 = \mathbf{Z}$   
I+L =  $(8+11) \% 26 = 19 = \mathbf{T}$   
N+C =  $(13+2) \% 26 = 15 = \mathbf{P}$   
G+A =  $(6+0) \% 26 = 6 = \mathbf{G}$   
D+Z =  $(3+25) \% 26 = 2 = \mathbf{C}$   
I+H =  $(8+7) \% 26 = 15 = \mathbf{P}$   
S+X =  $(18+23) \% 26 = 15 = \mathbf{P}$   
U+L =  $(20+11) \% 26 = 5 = \mathbf{F}$   
N+F =  $(13+5) \% 26 = 18 = \mathbf{S}$   
G+T =  $(6+19) \% 26 = 25 = \mathbf{Z}$   
A+U =  $(0+20) \% 26 = 20 = \mathbf{U}$   
I+Z =  $(8+25) \% 26 = 7 = \mathbf{H}$

--- Hasil Akhir ---  
Plaintext : AGUSMEMANCINGDISUNGAI  
Key : CMRIFNPUJXLCAZHXLFTUZ  
Ciphertext: CSLARRBUWZTPGCPPFSZUH

#### Kesimpulan

One-Time Pad (OTP) Cipher adalah metode enkripsi paling aman secara teori karena menggunakan kunci acak yang panjangnya **sama persis** dengan pesan asli (plaintext). Setiap karakter dari plaintext dijumlahkan secara modular (mod 26) dengan karakter dari kunci untuk menghasilkan ciphertext. Kunci ini hanya boleh digunakan sekali (one-time), dan **tidak boleh dibagikan secara sembarangan**. Jika kunci benar-benar acak, tidak diulang, dan dirahasiakan dengan baik, maka OTP tidak bisa dipecahkan bahkan dengan kekuatan komputasi tak terbatas.

File dapat dilihat pada Repositori GitHub :  
[https://github.com/dimasjuniorseven/OTP\\_Cipher](https://github.com/dimasjuniorseven/OTP_Cipher)



Yogjakarta, 23 April 2025 17:45 WIB

**Dimas Oktavian Prasetyo**

Junior Software Engineer and Full-Stack Developer, Digital Forensics Audit Investigator & Cybersecurity Specialist.

Untuk menguji saya buat program sama seperti pada studi kasus 1 dan hasilnya :