

문자열(String) 사용 하기

- ▶ String concatenation with + operator
- ▶ String replication with * operator
- ▶ Indexing with []
 - ▶ positive, negative index: [n] [-n]
- ▶ Slicing with [:] (range)
 - ▶ two indexes: [n:m]
 - ▶ blank index: [:m] [n:]
- ▶ Repeated index [:][:]

```
# 문자열 합치기  
print('Hello, ' + 'Python')
```

```
# 문자열 인덱싱  
print('Hello'[0])  
print('Hello'[-1])  
print('Hello'[-2])  
print('Hello'[2])
```

```
# 문자열 슬라이싱  
print('Hello, world'[0:4])  
print('Hello, world'[-5:-1])  
print('Hello, world'[-5:])
```

입출력 함수

- ▶ 출력 함수 print()
- ▶ 입력 함수 input()
- ▶ 주석(comments) #

```
# 사용자 입력 받기  
print("What is your name?")  
myName = input()  
print("Nice to meet you, " + myName + ".")
```

Sample Code: Reverse Cipher

- ▶ 반복문 while
- ▶ 문자열 길이: len()

```
# 문자열 거꾸로 만들기
```

```
message = 'This is a sample text.'  
translated = ''  
i = len(message)-1  
while i >= 0:  
    translated = translated + message[i]  
    print('translated message = ', translated)  
    i = i - 1  
print('\n Final Result = ', translated)
```

문자열 검색 함수: find()

- ▶ 문자열 찾기: find()
 - ▶ 찾는 문자열의 인덱스(0~)
 - ▶ 찾지 못하면 -1
- ▶ 나머지 연산자 %
 - ▶ 연산의 우선 순위에 주의

```
#-- find()
msg = "abcdefghijklmnopqrstuvwxyz"
print(msg.find("def"))
print(msg.find("aa"))
print(msg[msg.find("j"):])
```

```
#--- % 연산자
print(5 % 3)
print( (10 + 10) % 3)
print( 10 + 10 % 3)
```

Caesar Cipher (Encryption)

```
UpAlphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
LowAlphabet = 'abcdefghijklmnopqrstuvwxyz'

plain_msg = 'This is a plaintext message to be encrypted.'
key = 3 # 암호키: select from (0-25)

##--- 암호화 과정
cipher_msg = ''
for symbol in plain_msg :
    if symbol in UpAlphabet:
        symbol_idx = UpAlphabet.find(symbol)
        trans_idx = (symbol_idx + key) % len(UpAlphabet)
        cipher_msg = cipher_msg + UpAlphabet[trans_idx]
    elif symbol in LowAlphabet:
        symbol_idx = LowAlphabet.find(symbol)
        trans_idx = (symbol_idx + key) % len(LowAlphabet)
        cipher_msg = cipher_msg + LowAlphabet[trans_idx]
    else:
        cipher_msg = cipher_msg + symbol

print('PLAINTEXT = ', plain_msg)
print('CIPHERTEXT = ', cipher_msg)
```

Caesar Cipher (Decryption)

```
key = 3 # select from (0-25)

UpAlphabet    = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
LowerAlphabet = 'abcdefghijklmnopqrstuvwxyz'

#--- 복호화 과정
ciphertext = "wklv lv d sodlqwhaw phvvdjh wr eh hqfubswhg."
recovered_msg = ""

for symbol in ciphertext :
    if symbol in UpAlphabet:
        symbol_idx = UpAlphabet.find(symbol)
        trans_idx = (symbol_idx - key) % len(UpAlphabet)
        recovered_msg = recovered_msg + UpAlphabet[trans_idx]
    elif symbol in LowerAlphabet:
        symbol_idx = LowerAlphabet.find(symbol)
        trans_idx = (symbol_idx - key) % len(LowerAlphabet)
        recovered_msg = recovered_msg + LowerAlphabet[trans_idx]
    else:
        recovered_msg = recovered_msg + symbol
print('CIPHERTEXT = ', cipher_msg)
print('PLAINTEXT  = ', recovered_msg)
```

사용자 정의 함수

▶ 함수 정의 기본

```
def my_func(x,y):  
    z = x+y  
    return z
```

결과 값
(return value)

```
a = 1  
b = 2  
print(my_func(a,b))
```

```
def my_double(x,y):  
    return (2*x, 2*y)
```

```
a2, b2 = my_double(a, b)  
print(a2, b2)
```

파라미터

함수 파라미터 전달 방식

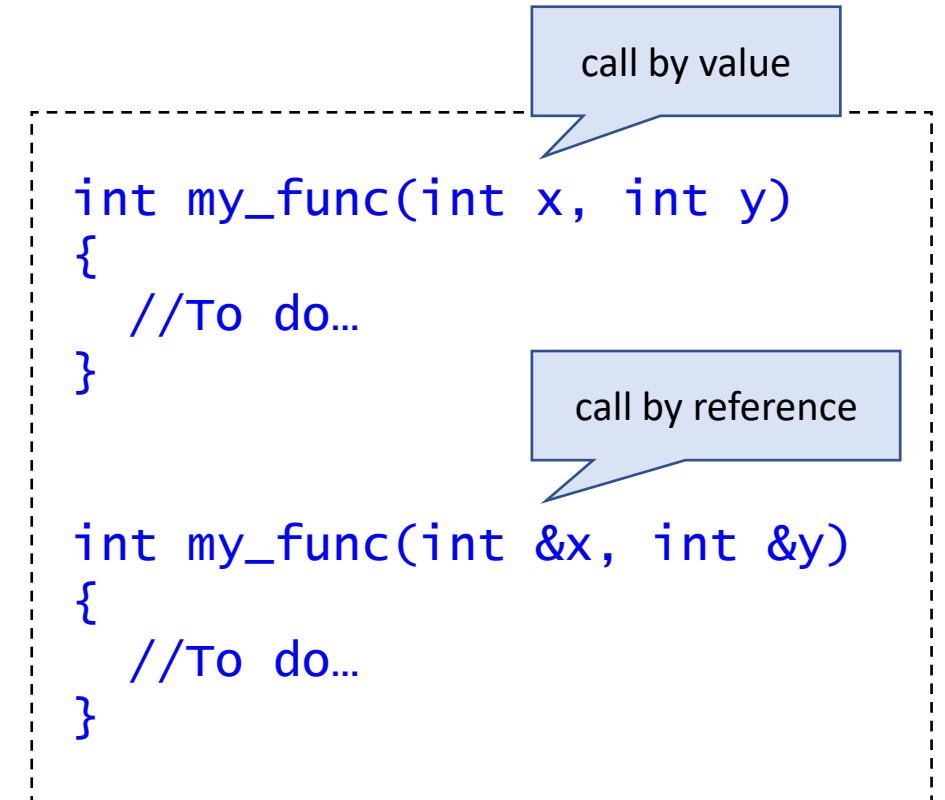
▶ C/C++의 파라미터 전달 방식

- ▶ Call by value
- ▶ Call by reference

▶ Python의 파라미터 전달 방식

- ▶ Call by object
(=Call by object reference, =Call by sharing)

Python initially behaves like call-by-reference, but as soon as we are changing the value of such a variable, i.e., as soon as we assign a new object to it, Python "switches" to call-by-value.



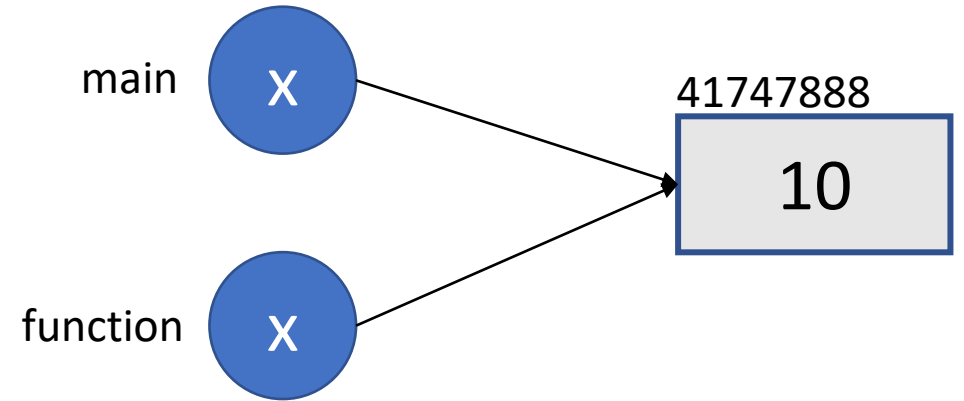
Call by object

▶ 필요한 경우에만 복사본을 만든다.

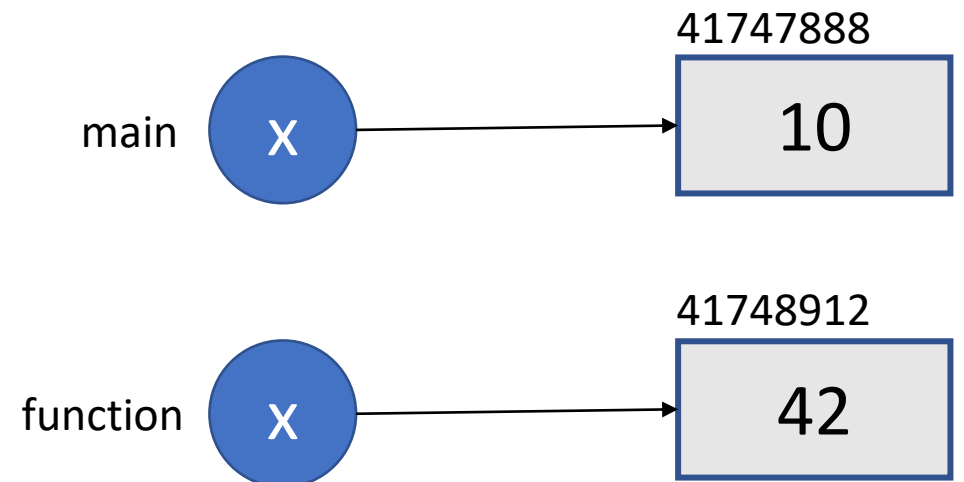
```
def ref_demo(x):  
    print("x=",x," id=",id(x))  
    x=42  
    print("x=",x," id=",id(x))
```

```
x = 10  
ref_demo(x)  
print("x=",x," id=",id(x))
```

x=42 이면?



x=42 실행 후



In-place operation

▶ 함수로 전달된 파라미터의 값은 바뀔 수 있는가?

▶ Immutable variable vs Mutable variable

▶ In-place operation

immutable variable: 정수, 실수
mutable variable: 리스트, 배열

```
def no_side_effects(cities):  
    print("cities=", cities, " id=", id(cities))  
    cities = cities + ["Birmingham", "Bradford"]  
    print("cities=", cities, " id=", id(cities))
```

```
locations = ["London", "Leeds", "Glasgow", "Sheffield"]  
no_side_effects(locations)  
print(locations)
```

```
def side_effects(cities):  
    print("cities=", cities, " id=", id(cities))  
    cities += ["Birmingham", "Bradford"] # in-place operation  
    print("cities=", cities, " id=", id(cities))
```

```
locations = ["London", "Leeds", "Glasgow", "Sheffield"]  
side_effects(locations)  
print(locations)
```

새로 메모리를
할당하지 않고
기존 데이터를
업데이트 함

```
cities= ['London', 'Leeds', 'Glasgow', 'Sheffield'] id= 2631971113800  
cities= ['London', 'Leeds', 'Glasgow', 'Sheffield', 'Birmingham', 'Bradford'] id= 2631959055432  
['London', 'Leeds', 'Glasgow', 'Sheffield']  
cities= ['London', 'Leeds', 'Glasgow', 'Sheffield'] id= 2631959055432  
cities= ['London', 'Leeds', 'Glasgow', 'Sheffield', 'Birmingham', 'Bradford'] id= 2631959055432  
['London', 'Leeds', 'Glasgow', 'Sheffield', 'Birmingham', 'Bradford']
```

SWAP 함수

- ▶ swap() 두 변수의 값을 서로 바꾸는 함수

```
def my_swap(a,b):  
    print('a = ', a, 'b = ', b)  
    temp = a  
    a= b  
    b = temp  
    print('a = ', a, 'b = ', b)  
    return a,b
```

C언어 스타일로
구현하면...

```
a, b = b, a
```

사실은
swap() 함수를
만들 필요 없음

함수를 이용한 Caesar Cipher 구현

▶ 함수로 정의된 암호화/복호화 사용

```
UpAlphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'  
LowAlphabet = 'abcdefghijklmnopqrstuvwxyz'
```

전역변수

```
plain_msg = 'This is a plaintext message to be encrypted.'  
my_key = 13 # select from (0-25)  
cipher_msg = caesar_encrypt(my_key, plain_msg)  
print('PLAINTEXT = ', plain_msg)  
print('CIPHERTEXT = ', cipher_msg, '\n')
```

Caesar 암호를 함수로 정의하고
필요할 때 호출하여 사용함

```
recovered_msg = caesar_decrypt(my_key, cipher_msg)  
print('CIPHERTEXT = ', cipher_msg)  
print('PLAINTEXT = ', recovered_msg)
```

함수를 이용한 Caesar Cipher 구현

▶ 암호화 함수

```
def caesar_encrypt(key, plain_msg):  
    cipher_msg = ''  
    for symbol in plain_msg :  
        if symbol in UpAlphabet:  
            symbol_idx = UpAlphabet.find(symbol)  
            trans_idx = (symbol_idx + key) % len(UpAlphabet)  
            cipher_msg = cipher_msg + UpAlphabet[trans_idx]  
        elif symbol in LowAlphabet:  
            symbol_idx = LowAlphabet.find(symbol)  
            trans_idx = (symbol_idx + key) % len(LowAlphabet)  
            cipher_msg = cipher_msg + LowAlphabet[trans_idx]  
        else:  
            cipher_msg = cipher_msg + symbol  
    return cipher_msg
```

함수를 이용한 Caesar Cipher 구현

▶ 복호화 함수

```
def caesar_decrypt(key, cipher_msg):  
    recovered_msg = ''  
    for symbol in cipher_msg :  
        if symbol in UpAlphabet:  
            symbol_idx = UpAlphabet.find(symbol)  
            trans_idx = (symbol_idx - key) % len(UpAlphabet)  
            recovered_msg = recovered_msg + UpAlphabet[trans_idx]  
        elif symbol in LowAlphabet:  
            symbol_idx = LowAlphabet.find(symbol)  
            trans_idx = (symbol_idx - key) % len(LowAlphabet)  
            recovered_msg = recovered_msg + LowAlphabet[trans_idx]  
        else:  
            recovered_msg = recovered_msg + symbol  
    return recovered_msg
```

출력 함수 print()의 포매팅

- ▶ C언어와 유사한 방법으로 print() 함수의 포맷이 가능함

```
n = 100
print('I trust you %s%%!' %(n))

Old_Name = 'Rijndael'
New_Name = 'AES'
print('%s is the old name of %s algorithm in 1990.' %(Old_Name, New_Name))
```

%를 출력할 땐 %%

주의!
coma(,) 없음

파일 다루기

▶ 파일에서 읽어오기

```
import os, sys # 파일을 다루기 위한 라이브러리

in_file = 'my_text.txt'

if not os.path.exists(in_file):
    print('File %s does not exist.' %(in_file))
    sys.exit() # 프로그램 종료

#-- 입력파일에서 텍스트 읽기
InFileObj = open(in_file)
my_content = InFileObj.read()
InFileObj.close()

print(my_content)
```

```
# 작업 폴더 확인
print('Working directory : ',os.getcwd())

# 작업 폴더 변경
os.chdir('folder_name')
```


파일 다루기

▶ 파일에 쓰기

```
import os, sys # 파일을 다루기 위한 라이브러리

out_file = 'my_out.txt'

#-- 출력파일이 존재하면 덮어쓸지 물어보기
if os.path.exists(out_file):
    print('This will overwrite the file %s. (C)ontinue or (Q)uit' % (out_file))
    response = input('> ') # 사용자 입력 기다리기
    if not response.lower().startswith('c'):
        sys.exit()

outFileObj = open(out_file, 'w')
outFileObj.write(my_content)
outFileObj.close()
```

리스트 다루기

▶ 리스트

- ▶ (서로 다른 타입의) 데이터를 순서에 따라 모은 것

▶ 리스트 만들기, 인덱싱

```
animals = ['cat', 'dog', 'lion', 'tiger', 'hippo', 'snake', 'bird']
```

```
#-- 리스트 인덱싱
```

```
print(animals[0])
```

```
print(animals[1:])
```

```
print(animals.index('man'))
```

리스트에 없으면
오류가 발생한다!

리스트 다루기

- ▶ 원소 추가 방법
- ▶ 원소 여부를 확인하기

```
animals += 'man'
```



```
animals += list('man')
```

```
list('man') # ['m', 'a', 'n']
```

```
animals = ['cat', 'dog', 'lion', 'tiger', 'hippo', 'snake', 'bird']
```

```
#-- Append (리스트, 문자열, 메소드)
```

```
animals += ['man']
```

```
animals += 'man'
```

```
animals.append('woman')
```

```
print(animals)
```

```
if 'man' in animals:
```

```
    print('A man is an animal.')
```

3가지 방법은 모두 같은 결과?

원소 확인

리스트 다루기

- ▶ 리스트를 이용한 반복문
- ▶ 리스트를 문자열로

```
animals = ['cat', 'dog', 'lion', 'tiger', 'hippo', 'snake', 'bird']
```

```
for pet in animals:  
    print('I have a %s.' % (pet))
```

```
#-- 리스트 -> 문자열 (join)  
print (''.join(animals))
```

리스트를 문자열로 변환

```
print(3*[1,2,3]+[9])
```

이 결과는?

Dictionary 다루기

▶ Dictionary 데이터 타입

- ▶ Dictionary: (key, value) 의 모임
- ▶ 숫자 인덱스를 사용하지 않고 키(key)에 대응되는 값(value)를 저장함
- ▶ 해시함수를 이용하여 원소(key, value)에 빠른 접근이 가능(순차적 검색아님)

```
myDic1 = { 'us' : 'AES', 'kr' : 'LEA', 'jp' : 'MISTY' }  
print(myDic1['kr'])
```

```
#-- copy  
myDic2 = myDic1  
myDic2['ru'] = 'GOST'  
print(myDic1)  
print(myDic2)
```

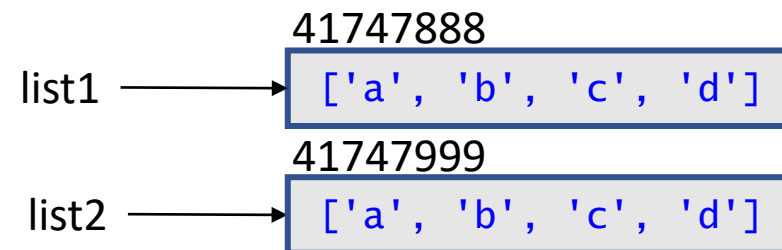
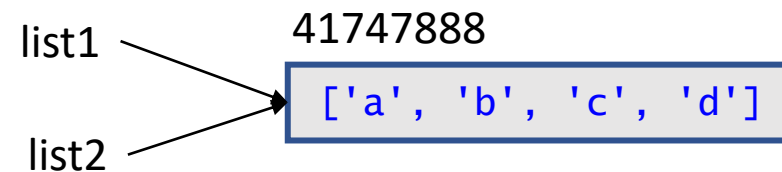
myDic2[0]과 같은
접근은 가능하지 않음!

두 Dictionary가 같아진다!

Shallow and Deep Copy

▶ 복잡한 구조의 복사 방법

- ▶ Shallow copy: 같은 메모리를 참조
- ▶ Deep copy: 새로운 메모리 할당



```
#shallow copy
```

```
list1 = ['a', 'b', 'c', 'd']  
list2 = list1 #shallow copy  
list2[0] = 'A'  
print('list1 = ', list1)  
print('list2 = ', list2)
```

```
#deep copy (copy lib)
```

```
import copy  
list1 = ['a', 'b', 'c', 'd']  
list2 = copy.deepcopy(list1) #deep  
copy  
list2[0] = 'A'  
print('list1 = ', list1)  
print('list2 = ', list2)
```

deepcopy를 위한
라이브러리

Deep Copy and Slice Operator

▶ 리스트의 깊은 복사

- ▶ Slice operator를 이용한 깊은 복사: 1단계까지만 가능 (아래 예제를 확인!!!)
- ▶ Deep copy(copy library): 모든 단계의 깊은 복사

```
#deep copy (copy lib)
```

```
list1 = ['a', 'b', ['c', 'd']]
list2 = copy.deepcopy(list1)
list2[2] = 'CD'
list3 = copy.deepcopy(list1)
list3[2][0] = 'C'
print('list1 = ', list1)
print('list2 = ', list2)
print('list3 = ', list3)
```

```
#deep copy (slice operator)
```

```
list1 = ['a', 'b', ['c', 'd']]
list2 = list1[:] #slice operator
list2[2] = 'CD'
list3 = list1[:]
list3[2][0] = 'C'
print('list1 = ', list1)
print('list2 = ', list2)
print('list3 = ', list3)
```

Split과 Join

- ▶ split: 문자열을 나누어 리스트로
- ▶ join: 리스트를 문자열로

```
msg = 'This is a sample text'
list_msg = msg.split()
print('msg = ', msg)
print('list = ', list_msg)
```

공백을 기준으로 나눈다.

```
joined_msg = ''.join(list_msg)
print('joined = ', joined_msg)
```

공백없는 문자열로 합쳐진다.

```
for k in range(len(list_msg)-1):
    list_msg[k] += ' '
joined_msg2 = ''.join(list_msg)
print('joined2 = ', joined_msg2)
```

원래 문자열과 같은 결과

영어 사전 활용하기

▶ 영어단어사전 활용함수

- ▶ dictionary.txt: 영어단어로 된 파일
 - ▶ 영어 단어로된 dictionary 데이터 만들기

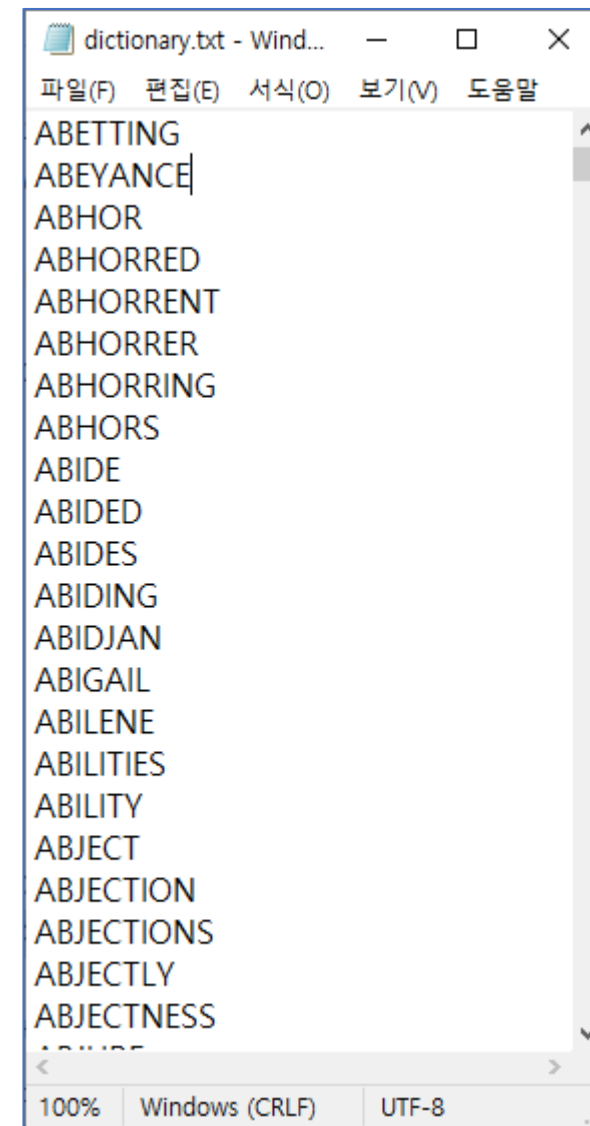
```
def loadDictionary():  
    dictionary_file = open('dictionary.txt')  
    EnglishWords = {}  
    for word in dictionary_file.read().split('\n'):  
        EnglishWords[word] = None  
    dictionary_file.close()  
    return EnglishWords
```

-- 전역변수

```
EnglishWords = loadDictionary()
```

(key, value) → (word, None)
예: { 'this': None, 'is' : None }

- ▶ 주어진 텍스트가 영어인지 판정하는 함수 만들기
 - ▶ isEnglish('This is a sample') → True



영어단어 다루기

- ▶ removeNonLetters()
 - ▶ 문자열에서 영문자, 공백만 남기기

```
letters_and_space = UpAlphabet + UpAlphabet.lower() + '\t\n'
```

```
#---- 특수문자, 숫자 지우기
def removeNonLetters(message):
    letters_only = []
    for ch in message:
        if ch in letters_and_space:
            letters_only.append(ch)
    return ''.join(letters_only)
```

```
#-- 전역변수
EnglishWords = loadDictionary()
```

- percentEnglishWords()
 - 영어사전에 있는 단어의 비율

```
#---- 올바른 영어단어의 비율
def percentEnglishWords(message):
    message = message.upper()
    message = removeNonLetters(message)
    possible_words = message.split()

    if possible_words == []:
        return 0.0
    count_words = 0
    for word in possible_words:
        if word in EnglishWords:
            count_words += 1
    return
    float(count_words)/len(possible_words)
```

0으로 나누는
오류발생을 방지

영어사전에 있는
단어인지?

영어 판정 함수: isEnglish()

- ▶ 주어진 문자열이 영어로 된 것인지 판정하는 함수
 - ▶ 복호화 분장이 바르게 되었는지 판정할 때 사용

```
#--- 영어인지 판정하기
def isEnglish(message, wordPercentage=20, letterPercentage=80):
    wordsMatch = percentEnglishWords(message)*100 >= wordPercentage

    numLetters = len(removeNonLetters(message))
    messageLettersPercentage = float(numLetters) / len(message) * 100
    lettersMatch = messageLettersPercentage >= letterPercentage

    return wordsMatch and lettersMatch
```

영어 단어의 비율이
충분함

영문자의 비율이
충분함

Caesar Cipher Attack 2

▶ 영어의 특성을 이용한 Caesar Cipher 공격법

```
import CaesarCipher_lib
import EngDic_lib
import os, sys
```

```
ciphertext = 'Znoy oy g ygsvrk'
print('CIPHERTEXT = ', ciphertext)
```

```
for key in range(0,26):
    recovered_msg = CaesarCipher_lib.caesar_decrypt(key, ciphertext)
    PercentEngWords = EngDic_lib.percentEnglishWords(recovered_msg)*100
    print('key # %2s : %s (English word: %5.1f%%)' % (key, recovered_msg, PercentEngWords) )
```

키 전수조사

이 값이 가장 큰 경우가
올바른 암호키 !!!

```
CIPHERTEXT = Znoy oy g ygsvrk
key # 0 : Znoy oy g ygsvrk (English word: 0.0%)
key # 1 : Ymnx nx f xfruqj (English word: 0.0%)
key # 2 : Xlmw mw e weqtpi (English word: 0.0%)
key # 3 : Wklv lv d vdpsoh (English word: 0.0%)
key # 4 : Vjku ku c ucorng (English word: 0.0%)
key # 5 : Uijt jt b thngmf (English word: 0.0%)
key # 6 : This is a sample (English word: 75.0%)
key # 7 : Sghr hr z rlekd (English word: 0.0%)
key # 8 : Rfgq gq y qyknjc (English word: 0.0%)
key # 9 : Qefp fp x pxjmib (English word: 0.0%)
key #10 : Pdeo eo w owilha (English word: 0.0%)
key #11 : Ocdn dn v nvhkgz (English word: 0.0%)
key #12 : Nbcm cm u mugjfy (English word: 0.0%)
key #13 : Mabl bl t ltfiex (English word: 0.0%)
key #14 : Lzak ak s ksehdw (English word: 0.0%)
key #15 : Kyzj zj r jrdgcv (English word: 0.0%)
key #16 : Jxyi yi q iqcfbu (English word: 0.0%)
key #17 : Iwxh xh p hpbeat (English word: 0.0%)
key #18 : Hvwg wg o goadz (English word: 0.0%)
key #19 : Guvf vf n fnzcyr (English word: 0.0%)
key #20 : Ftue ue m emybxq (English word: 0.0%)
key #21 : Estd td l dlxawp (English word: 0.0%)
key #22 : Drsc sc k ckwzvo (English word: 0.0%)
key #23 : Cqrb rb j bjvyun (English word: 0.0%)
key #24 : Bpqa qa i aiuxtm (English word: 0.0%)
key #25 : Aopz pz h zhtwsl (English word: 0.0%)
```

목차

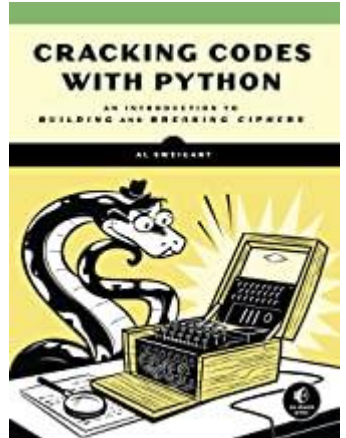
1. 치환암호 Subst Cipher

**Cracking codes with Python:
An Introduction to Building and Breaking Ciphers
by Al Sweigart (2018)**

2. 문자열의 빈도

3. 난수생성 및 활용

4. 단어사전을 이용한 치환암호 Subst Cipher 해독



치환암호 Subst Cipher

- 암호화 함수

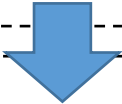
```
Alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def subst_encrypt(key, msg):
    result = ''
    InSet = Alphabet
    OutSet = key

    for ch in msg:
        if ch.upper() in InSet:
            idx = InSet.find(ch.upper())
            if ch.isupper():
                result += OutSet[idx].upper()
            else:
                result += OutSet[idx].lower()
        else:
            result += ch
    return result
```

```
my_key = 'VWXABCDEIJKFGHLMQRSNOPTUYZ'
ciphertext = subst_encrypt(my_key, message)
print(ciphertext)
```

message =
'Cryptography is the practice and study of techniques for secure communication in the presence of third parties (called adversaries). More generally, it is about constructing and analyzing protocols that overcome the influence of adversaries and which are related to various aspects in information security such as data confidentiality, data integrity, and authentication.'



ciphertext =
'Xrymnlrdvmey is neb mrvxnixb vha snoay lc nbxehiqobs clr sbxorb xlggoxixvnilh ih neb mrbsbhxb lc neira mvrnibs (xvffba vapbrsvribs). Glrb dbhbrvffy, in is vwlon xlhsnroxnihd vha vhfvyzihd mrlnlxlfv nevn lpbrxlgb neb ihcfobhxb lc vapbrsvribs vha teixe vrb rbfvnb nl pvrilos vsmbxns ih ihclrgvnilh sbxoriny soxe vs avnv xlhciabhnivfny, avnv ihnbdriny, vha vonebhnixvnilh.'

치환암호 Subst Cipher

- 복호화 함수

```
def subst_decrypt(key, msg):  
    result = ''  
    InSet = key  
    OutSet = Alphabet
```

암호화와 복호화는 서로
InSet 과 OutSet의 역할만 바뀐다.

```
    for ch in msg:  
        if ch.upper() in InSet:  
            idx = InSet.find(ch.upper())  
            if ch.isupper():  
                result += OutSet[idx].upper()  
            else:  
                result += OutSet[idx].lower()  
        else:  
            result += ch  
  
    return result
```

```
my_key = 'VWXABCDEIJKFGHLMQRSNOPTUYZ'  
message = subst_decrypt(my_key, ciphertext)  
print(message)
```

```
ciphertext =  
'Xrymnlrdvmey is neb mrvxnixb vha snoay lc nbxehiqobs clr sbxorb  
xlggohixvnilh ih neb mrbsbhxb lc neira mvrnibs (xvffba vapbrsvribs).  
Glr dbhbrvffy, in is vwlon xlsnroxnihd vha vhfyzihd mrlnlxlfv nevn  
lpbrxlgv neb ihcfobhxb lc vapbrsvribs vha teixe vrb rbfvnb nl pvrilos  
vsmbxns ih ihclrgvnilh sbxoriny soxe vs avnv xlhciabhnivfiny, avnv  
ihnbdriny, vha vonebhnixvnilh.'
```



```
message =  
'Cryptography is the practice and study of techniques for secure  
communication in the presence of third parties (called adversaries).  
More generally, it is about constructing and analyzing protocols that  
overcome the influence of adversaries and which are related to  
various aspects in information security such as data confidentiality,  
data integrity, and authentication.'
```

빈도 분석 - 빈도 사전 만들기

- 문자열에서 각 알파벳의 출현 횟수를 계산하는 함수

```
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
def getLetterCount(message):  
    letterCount = {'A':0, 'B':0, 'C':0, 'D':0, 'E':0, 'F':0, 'G':0, 'H':0,  
                  'I':0, 'J':0, 'K':0, 'L':0, 'M':0, 'N':0, 'O':0, 'P':0,  
                  'Q':0, 'R':0, 'S':0, 'T':0, 'U':0, 'V':0, 'W':0, 'X':0,  
                  'Y':0, 'Z':0}
```

```
    for char in message.upper():  
        if char in LETTERS:  
            letterCount[char] += 1
```

```
    return letterCount
```

(복습)

사전(dictionary) 데이터 타입

Dic = { key1 : value1, key2 : value2, }

참조 방식: Dic[key1] = value1

함수의 리턴 값으로
'사전'도 가능하다

```
{'A': 166, 'B': 24, 'C': 115, 'D': 76, 'E': 270, 'F': 43, 'G': 43,  
 'H': 87, 'I': 174, 'J': 0, 'K': 10, 'L': 73, 'M': 69, 'N': 160,  
 'O': 160, 'P': 63, 'Q': 5, 'R': 151, 'S': 125, 'T': 204, 'U': 58,  
 'V': 22, 'W': 19, 'X': 3, 'Y': 54, 'Z': 2}
```


사전 반대로 만들기

영한사전 → 한영사전

- 사전의 key와 value의 역할 서로 바꾸기

- Dic1 = { key1 : value1, key2 : value2,}
- Dic2 = {value1 : key1, value2 : key2,}

키, 값의 역할 바꾸기

```
letter2freq = getLetterCount(message)
freq2letter = {}
```

```
for char in LETTERS:
```

처음 나오는
빈도(key)인가?

```
    if letter2freq[char] not in freq2letter:
```

```
        freq2letter[letter2freq[char]] = [char]
```

```
    else:
```

```
        freq2letter[letter2freq[char]].append(char)
```

이미 있는 값에 추가하기
예: 43: ['F'] → 43: ['F', 'G']

letter2freq

{'A': 166, 'B': 24, 'C': 115, 'D': 76, 'E': 270, 'F': 43, 'G': 43,
'H': 87, 'I': 174, 'J': 0, 'K': 10, 'L': 73, 'M': 69, 'N': 160,
'O': 160, 'P': 63, 'Q': 5, 'R': 151, 'S': 125, 'T': 204, 'U': 58,
'V': 22, 'W': 19, 'X': 3, 'Y': 54, 'Z': 2}

(key: value) = (빈도:문자)
예: 166: ['A']
추가하기

freq2letter

{166: ['A'], 24: ['B'], 115: ['C'], 76: ['D'], 270: ['E'], 43: ['F', 'G'],
87: ['H'], 174: ['I'], 0: ['J'], 10: ['K'], 73: ['L'], 69: ['M'], 160: ['N', 'O'],
63: ['P'], 5: ['Q'], 151: ['R'], 125: ['S'], 204: ['T'], 58: ['U'], 22: ['V'], 19:
['W'], 3: ['X'], 54: ['Y'], 2: ['Z']}

리스트 정렬하기 (sort)

- 리스트 정렬 함수 – sort()
 - key: 정렬의 기준이 될 값을 계산하는 **함수** 이름
 - reverse: 순방향(True)/역방향(False) 선택

```
# 영문자 빈도순서
ETAOIN = 'ETAOINSHRDLCLUMWFGYPBVKJXQZ'

list1 = [ 'F', 'G', 'N', 'O' ]
list1.sort(key= ETAOIN.find, reverse=True)
print(list1)
```

['G', 'F', 'N', 'O']

큰 값을 앞쪽에

```
print(''.join(list1))
```

'GFNO'

리스트를 문자열로

ETAOIN.find

ETAOIN.find('F') = 15
ETAOIN.find('G') = 16
ETAOIN.find('N') = 5
ETAOIN.find('O') = 3

빈도사전의 정렬(1) - 리스트(값) 정렬

- 각 값(문자의 리스트) 정렬
 - 동일한 출현 빈도(key)인 문자들(동점자들)을 정렬하는 방법
 - 일반적인 영문빈도의 순서로 정렬하고 문자열로 변경

```
freq2letter = {166: ['A'], 24: ['B'], 115: ['C'], 76: ['D'], 270: ['E'],  
43: ['F', 'G'], 87: ['H'], 174: ['I'], 0: ['J'], 10: ['K'], 73: ['L'],  
69: ['M'], 160: ['N', 'O'], 63: ['P'], 5: ['Q'], 151: ['R'], 125: ['S'],  
204: ['T'], 58: ['U'], 22: ['V'], 19: ['W'], 3: ['X'], 54: ['Y'], 2: ['Z']}
```

```
for freq in freq2letter:
```

작은 값을 앞쪽에

```
    freq2letter[freq].sort(key=ETA0IN.find, reverse=False)
```

```
    freq2letter[freq] = ''.join(freq2letter[freq])
```

동일한 빈도(key)에
두개 이상의 문자가 대응된
경우만 정렬된다.

```
print(freq2letter)
```

```
{166: 'A', 24: 'B', 115: 'C', 76: 'D', 270: 'E', 43: 'FG', 87: 'H', 174: 'I', 0: 'J',  
10: 'K', 73: 'L', 69: 'M', 160: 'ON', 63: 'P', 5: 'Q', 151: 'R', 125: 'S', 204: 'T',  
58: 'U', 22: 'V', 19: 'W', 3: 'X', 54: 'Y', 2: 'Z'}
```

빈도사전의 정렬(2) - 사전 정렬

- 빈도사전의 문자를 빈도순으로 정렬하고 문자열로 변환

사전을 리스트로
변환

```
freq2letter = {166: 'A', 24: 'B', 115: 'C', 76: 'D', 270: 'E',  
43: 'FG', 87: 'H', 174: 'I', 0: 'J', 10: 'K', 73: 'L', 69: 'M',  
160: 'ON', 63: 'P', 5: 'Q', 151: 'R', 125: 'S', 204: 'T', 58: 'U',  
22: 'V', 19: 'W', 3: 'X', 54: 'Y', 2: 'Z'}
```

```
freqPairs = list(freq2letter.items())  
print('FreqPairs = ', freqPairs)
```

배열의 첫번째 값을
기준으로 정렬함

```
freqPairs.sort(key=getItemAtIndexZero, reverse=True)  
print('FreqPairs = ', freqPairs)
```

```
FreqPairs = [(166, 'A'), (24, 'B'), (115, 'C'), (76, 'D'), (270, 'E'), (43, 'FG'), (87, 'H'),  
(174, 'I'), (0, 'J'), (10, 'K'), (73, 'L'), (69, 'M'), (160, 'ON'), (63, 'P'), (5, 'Q'),  
(151, 'R'), (125, 'S'), (204, 'T'), (58, 'U'), (22, 'V'), (19, 'W'), (3, 'X'), (54, 'Y'), (2, 'Z')]
```

```
--- 배열의 첫번째 원소를 주는 함수  
def getItemAtIndexZero(items):  
    return items[0]
```

```
FreqPairs = [(270, 'E'), (204, 'T'), (174, 'I'), (166, 'A'), (160, 'ON'), (151, 'R'), (125, 'S'),  
(115, 'C'), (87, 'H'), (76, 'D'), (73, 'L'), (69, 'M'), (63, 'P'), (58, 'U'), (54, 'Y'), (43, 'FG'),  
(24, 'B'), (22, 'V'), (19, 'W'), (10, 'K'), (5, 'Q'), (3, 'X'), (2, 'Z'), (0, 'J')]
```

빈도순서를 문자열로

- 사전정렬 결과를 이용하여 빈도순서를 문자열로 만들기

```
FreqPairs = [(270, 'E'), (204, 'T'), (174, 'I'), (166, 'A'), (160, 'ON'),  
             (151, 'R'), (125, 'S'), (115, 'C'), (87, 'H'), (76, 'D'), (73, 'L'),  
             (69, 'M'), (63, 'P'), (58, 'U'), (54, 'Y'), (43, 'FG'), (24, 'B'),  
             (22, 'V'), (19, 'W'), (10, 'K'), (5, 'Q'), (3, 'X'), (2, 'Z'), (0, 'J')]
```

```
freqOrder = []
```

```
for freq_pair in freqPairs:  
    freqOrder.append(freq_pair[1])
```

```
['E', 'T', 'I', 'A', 'ON', 'R', 'S', 'C', 'H', 'D', 'L', 'M', 'P', 'U', 'Y', 'FG', 'B', 'V', 'W', 'K', 'Q', 'X', 'Z', 'J']
```

```
freq_order_str = ''.join(freqOrder)  
print('freq_order_str = ', freq_order_str)
```

```
freq_order_str = ETIAONRSCHDLMPUYFGBVWKQXZJ
```

이제 이 정도는
쉬워야 함!

알파벳 출현 빈도 - 종합

```
def getFreqOrder(message):
```

```
    #- (letter, freq) 사전 만들기: { 'A': 999, ... }
    letter2freq = getLetterCount(message)
    #- (freq, letter) 사전 만들기: { 999: ['A'], ... }
    freq2letter = {}
    for char in LETTERS:
        if letter2freq[char] not in freq2letter:
            freq2letter[letter2freq[char]] = [char]
        else:
            freq2letter[letter2freq[char]].append(char)

    for freq in freq2letter:
        freq2letter[freq].sort(key=ETAOIN.find, reverse=False)
        freq2letter[freq] = ''.join(freq2letter[freq])

    freqPairs = list(freq2letter.items())
    freqPairs.sort(key=getItemAtIndexZero, reverse=True)

    freqOrder = []
    for freq_pair in freqPairs:
        freqOrder.append(freq_pair[1])

    return ''.join(freqOrder)
```

```
message =
    'Cryptography is the practice and study of techniques for secure
    communication in the presence of third parties (called adversaries).
    More generally, it is about constructing and analyzing protocols that
    overcome the influence of adversaries and which are related to
    various aspects in information security such as data confidentiality,
    data integrity, and authentication.'
```

freq_order = getFreqOrder(message)

```
freq_order_str = ETIAONRSCHDLMPUYFGBVWKQXZJ
```

알파벳 빈도를 이용한 키 예측

- 키 예측 전략

- 암호문의 알파벳 빈도를 계산한다.
- 알파벳의 빈도와 영문 빈도순을 비교한다.
- 대응되는 문자가 암호화된 것으로 판단하여 암호키를 예측한다.

print()를 활용하여
계산과정을
파악해보자!

```
def Freq2Key(freq_order):  
    temp_dict = {}  
    i=0  
    for char in freq_order:  
        temp_dict[ETAOIN[i]] = char  
        i += 1  
    temp_list = list(temp_dict.items())  
    #print('temp_list =', temp_list)  
  
    temp_list.sort(key=getItemAtIndexZero)  
    #print('temp_list =', temp_list)  
  
    temp_key_list = []  
    for item in temp_list:  
        temp_key_list.append(item[1])  
  
    return ''.join(temp_key_list)
```

기본 정보

ETAOIN = 'ETAOINSHRDLCLUMWFGYPBVKJXQZ' # 영문 빈도 순서
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

암호문의 빈도

FREQ = 'ETIAONRSCHDLMPUYFGBVWKQXZJ'

암호키 예측

KEY = 'IVLHEYFSOQKDPNABZCRTMWUXGJ'

난수 생성하기

- 랜덤한 숫자 만들기

- 의사난수 생성기: 초기값(seed)로부터 결정되는 난수를 생성하는 알고리즘
- 특징 통계적으로 난수성이 우수한 출력이 얻어지나
같은 초기값을 사용하면 똑같은 난수가 재연됨

```
import random
```

```
# seed 설정 (고정값)  
random.seed(2020)
```

반복 실행시
동일한 난수 출력

```
for i in range(10):  
    print(random.randint(1,100))
```

```
import random
```

```
# seed 설정 (현재 시각)  
random.seed(time.time())
```

매번 다른
초기값으로
설정되어 실행할
때마다 다른 난수가
출력

```
for i in range(10):  
    print(random.randint(1,100))
```

1,2,..., 100
중에서 랜덤하게 출력

난수를 이용한 무작위 섞기(shuffling)

- 난수를 활용한 예제들

```
#-- 알파벳으로 랜덤 길이 랜덤 메시지 만들기
Alphabet      = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
alphabet_msg  = Alphabet * random.randint(5,10)
alphabet_list_msg = list(alphabet_msg)
random.shuffle(alphabet_list_msg)
shuffled_msg = ''.join(alphabet_list_msg)
print(shuffled_msg)
print('Random Message: "%s..."' % (shuffled_msg[:50])) # 간결한 출력
```

알파벳
알파벳을 반복한 문자열
문자열 --> 리스트
리스트 랜덤셔플
리스트 --> 문자열
랜덤 메시지 출력

난수를 이용하여 랜덤한
암호키를 생성해보자!

사전을 이용한 Subst Cipher 해독

- * 프로그래밍의 난이도가 조금 높으니 어려우면 아이디어만 이해하도록 합시다.
- * 시험범위에는 포함하지 않겠습니다.

단어의 패턴 분석

- 단어를 패턴으로 변환

```
def getWordPattern(word):  
    # Returns a string of the pattern form of the given word.  
    # e.g. '0.1.2.3.4.1.2.3.5.6' for 'DUSTBUSTER'  
    word = word.upper()  
    nextNum = 0  
    letterNums = {}  
    wordPattern = []  
  
    for letter in word:  
        if letter not in letterNums:  
            letterNums[letter] = str(nextNum)  
            nextNum += 1  
        wordPattern.append(letterNums[letter])  
    return '.'.join(wordPattern)
```

```
word_list = [ 'apple', 'connection', 'birthday', 'happy', 'mommy', 'cloud' ]  
for word in word_list:  
    print(getWordPattern(word))
```

'apple'	0.1.1.2.3
'connection'	0.1.2.2.3.0.4.5.1.2
'birthday'	0.1.2.3.4.5.6.7
'happy'	0.1.2.2.3
'mommy'	0.1.0.0.2
'cloud'	0.1.2.3.4

사전의 패턴 분석

- 사전파일('dictionary.txt') 내 모든 단어의 패턴 분석

```
import pprint
allPatterns = {}
```

사전

단어가 Enter('\n')로 구분된
사전파일을 읽는다.

```
fo = open('dictionary.txt')
wordList = fo.read().split('\n')
fo.close()
```

```
for word in wordList:
    # Get the pattern for each string in wordList:
    pattern = getWordPattern(word)
```

```
    if pattern not in allPatterns:
        allPatterns[pattern] = [word]
```

새로운 패턴을 만들고
단어를 추가

```
    else:
        allPatterns[pattern].append(word)
```

이미 존재하는 패턴에
단어를 추가

```
# This is code that writes code. The wordPatterns.py file contains
# one very, very large assignment statement:
fo = open('wordPatterns.py', 'w')
fo.write('allPatterns = ')
fo.write(pprint.pformat(allPatterns))
fo.close()
```

```
allPatterns = {'0.0.1': ['EEL'],
'0.0.1.2': ['EELS', 'OOZE'],
'0.0.1.2.0': ['EERIE'],
'0.0.1.2.3': ['AARON', 'LLOYD', 'OOZED'],
'0.0.1.2.3.4': ['AARHUS', 'EERILY'],
'0.0.1.2.3.4.5.5': ['EELGRASS'],
'0.1.0': ['ADA',
'BIB',
'BOB',
'DAD',
'DID',
'DUD',
'EKE',
'ERE',
'EVE',
'EWE',
'EYE',
'GAG',
'GIG',
'HUH',
'NAN',
'NON',
'NUN',
'PEP',
'PIP',
'POP',
'PUP',
'SUS',
'TIT'],
'0.1.0.0': ['DODD', 'LULL'],
'0.1.0.0.1': ['MAMMA'],
'0.1.0.0.1.2': ['MAMMAL', 'MAMMAS', 'PAPPAS', 'PEPPER'],
'0.1.0.0.1.2.1.3': ['PEPPERED'],
'0.1.0.0.1.2.3': ['BABBAGE',
'BIBBING',
'LILLIAN',
'MAMMALS'],
```

암호문 단어의 패턴 분석

- 예: 암호문 단어의 패턴 분석
 - 암호문 단어: 'XYX' → 패턴: 0.1.0
 - 사전 분석으로 얻는 패턴파일에서 'ADA', 'BIB', 'BOB', 'DAD', ..., 'TIT' 중 하나임
 - 패턴으로부터 x, y의 가능한 후보를 만들 수 있음

```
{'A': [], 'B': [], 'C': [], 'D': [], 'E': [], 'F': [], 'G': [], 'H': [], 'I': [], 'J': [],  
'K': [], 'L': [], 'M': [], 'N': [], 'O': [], 'P': [], 'Q': [], 'R': [], 'S': [], 'T': [],  
'U': [], 'V': [], 'W': [],  
'X': ['A', 'B', 'D', 'E', 'G', 'H', 'N', 'P', 'S', 'T'],  
'Y': ['D', 'I', 'O', 'A', 'U', 'K', 'R', 'V', 'W', 'Y', 'E'],  
'Z': []}
```

```
allPatterns = {'0.0.1': ['EEL'],  
'0.0.1.2': ['EELS', 'OOZE'],  
'0.0.1.2.0': ['EERIE'],  
'0.0.1.2.3': ['AARON', 'LLOYD', 'OOZED'],  
'0.0.1.2.3.4': ['AARHUS', 'EERILY'],  
'0.0.1.2.3.4.5': ['EELGRASS'],  
'0.1.0': ['ADA',  
          'BIB',  
          'BOB',  
          'DAD',  
          'DID',  
          'DUD',  
          'EKE',  
          'ERE',  
          'EVE',  
          'EWE',  
          'EYE',  
          'GAG',  
          'GIG',  
          'HUH',  
          'NAN',  
          'NON',  
          'NUN',  
          'PEP',  
          'PIP',  
          'POP',  
          'PUP',  
          'SUS',  
          'TIT'],  
'0.1.0.0': ['DODD', 'LULL'],  
'0.1.0.0.1': ['MAMMA'],  
'0.1.0.0.1.2': ['MAMMAL', 'MAMMAS', 'PAPPAS', 'PEPPER'],  
'0.1.0.0.1.2.1.3': ['PEPPERED'],  
'0.1.0.0.1.2.3': ['BABBAGE',  
                  'BIBBING',  
                  'LILLIAN',  
                  'MAMMALS']
```

암호문 단어 'XYX'
패턴 0.1.0

암호문 단어의 패턴 분석

```
import random, re, copy # re: regular expression
import wordPatterns, makeWordPatterns # 교재의 라이브러리
import SubstCipher_lib

nonLettersOrSpacePattern = re.compile('[^A-Z\s]')
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

#--- 분석용 Mapping 리스트 초기화
def getBlankMapping():
    return { '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':[]}

#--- 암호문의 단어 cipher_word의 후보단어 candidate의 정보를 Mapping에 반영
#--- 예: Mapping (빈 리스트), cipher_word='XTTPY', candidate='APPLE'
#       Mapping = { 'X': ['A'], 'T': ['P'], 'P': ['L'], 'Y': ['E'], ...}
def addLettersToMapping(Mapping, cipher_word, candidate):
    for i in range(len(cipher_word)):
        if candidate[i] not in Mapping[cipher_word[i]]:
            Mapping[cipher_word[i]].append(candidate[i])
```

```
word_map = getBlankMapping()
word = 'XYXXYZ'
addLettersToMapping(word_map, word, 'MAMMAL')
print(word_map)
```

```
word_map = {'A': [], 'B': [], 'C': [], 'D': [], 'E': [], 'F': [], 'G': [], 'H': [], 'I': [],
            'J': [], 'K': [], 'L': [], 'M': [], 'N': [], 'O': [], 'P': [], 'Q': [], 'R': [], 'S': [], 'T': [],
            'U': [], 'V': [], 'W': [], 'X': ['M'], 'Y': ['A'], 'Z': ['L']}
```

```
word_candidate = ['MAMMAL', 'MAMMAS', 'PAPPAS', 'PEPPER']
for candidate in word_candidate:
    addLettersToMapping(word_map, word, candidate)
print(word_map)
```

```
word_map = {'A': [], 'B': [], 'C': [], 'D': [], 'E': [], 'F': [], 'G': [], 'H': [], 'I': [], 'J': [],
            'K': [], 'L': [], 'M': [], 'N': [], 'O': [], 'P': [], 'Q': [], 'R': [], 'S': [], 'T': [], 'U': [], 'V': [],
            'W': [], 'X': ['M', 'P'], 'Y': ['A', 'E'], 'Z': ['L', 'S', 'R']}
```

후보 알파벳의 교집합

```
#--- 두 개의 Mapping의 교집합을 계산
# 주의: 어느 한쪽의 Mapping에만 candidate가 있는 경우는 그것을 사용함
# 공통 후보가 있으면 교집합으로 후보를 업데이트
def intersectMapping(mapA, mapB):
    intersectedMapping = getBlankMapping()
    for letter in LETTERS:
        if mapA[letter] == []:
            intersectedMapping[letter] = copy.deepcopy(mapB[letter])
        elif mapB[letter] == []:
            intersectedMapping[letter] = copy.deepcopy(mapA[letter])
        else:
            for mapped_letter in mapA[letter]:
                if mapped_letter in mapB[letter]:
                    intersectedMapping[letter].append(mapped_letter)

    return intersectedMapping
```

후보 알파벳이 다수인
문자는 교집합으로

```
map_A = { 'A': ['C', 'D', 'E'], 'B': [], 'C': ['B', 'A'], 'D': ['E'], 'E': ['C', 'D'], ... }
map_B = { 'A': ['D', 'E'], 'B': ['B', 'E'], 'C': ['A'], 'D': [], 'E': ['C'], ... }
```

```
map_C = intersectMapping(map_A, map_B)
print(map_C)
```

후보 알파벳이 한쪽에
있으면 모두 후보로 유지

```
map_C =
{ 'A': ['D', 'E'], 'B': ['B', 'E'], 'C': ['A'], 'D': ['E'], 'E': ['C'], 'F': [], 'G': [], 'H': [], 'I': [], 'J': [], 'K': [], 'L': [], 'M': [],
  'N': [], 'O': [], 'P': [], 'Q': [], 'R': [], 'S': [], 'T': [], 'U': [], 'V': [], 'W': [], 'X': [], 'Y': [], 'Z': [] }
```

후보 알파벳의 정리

```
#--- 후보가 하나만 남아 확정된 문자들을 정리
def removeSolvedfromMapping(Mapping):
    loop_flag = True
    while loop_flag:
        loop_flag = False
        solved_letters = [] # 중간계산에 필요한 리스트 (출력에 반영안됨)
        for cipher_letter in LETTERS:
            #- 한개의 후보만 남은 경우 => 확정
            if len(Mapping[cipher_letter]) == 1:
                solved_letters.append(Mapping[cipher_letter][0])

        # 남은 문자의 후보에서 이미 확정된 것을 제외
        for cipher_letter in LETTERS:
            for s in solved_letters: # 이미 확정된 문자에 대하여
                # 다른 문자에서 확정된 문자가 후보에 있으면 제외
                if len(Mapping[cipher_letter]) != 1 and s in Mapping[cipher_letter]:
                    Mapping[cipher_letter].remove(s)
                    if len(Mapping[cipher_letter]) == 1: # 이제 후보가 한개만 남으면
                        loop_flag = True # while 반복에서 후보 확정으로 이어짐

        # 확정된 문자들은 후보가 한개만 남아 있게 된다.
    return Mapping
```

대응되는 후보 알파벳이 한개이면 확정

확정된 알파벳은 다른 알파벳의 후보에서 삭제

확정된 알파벳 삭제후 후보가 한 개만 남으면

암호문으로부터 후보키 맵을 만든다

```
--- 사전에 저장된 패턴을 이용하여 후보키를 만든
def hacksubst(message):
    intersectedMap = getBlankMapping()
    #- message를 대문자로 바꾸고, 영문자만 남기고, 단어로 나누어 리스트 만들기
    cipherword_list = nonLettersOrSpacePattern.sub('', message.upper()).split()

    # message의 각 단어에 대하여
    for cipher_word in cipherword_list:
        candidate_map = getBlankMapping()
        wordPattern = makeWordPatterns.getWordPattern(cipher_word) # 단어의 패턴을 만들기
        if wordPattern not in wordPatterns.allPatterns: # 영어단어 패턴에 속하지 않으면
            continue # 사전에서 발견되지 않는 단어면 통과

        # 사전에 있는 단어이면 해당 패턴으로 후보를 업데이트 함
        for candidate in wordPatterns.allPatterns[wordPattern]:
            addLettersToMapping(candidate_map, cipher_word, candidate)
        intersectedMap = intersectMapping(intersectedMap, candidate_map)

    return removeSolvedfromMapping(intersectedMap)
```

암호문을 단어 단위로 나누어 리스트로 만든다.
(특수문자, 숫자는 삭제한다)

각 단어에 대하여 패턴을 분석하여
후보 알파벳 정보를 만든다.

기존 후보 알파벳 정보와 합친다.

후보키 맵을 이용한 복호화

```
def decryptWithMapping(ciphertext, letterMap):
```

key = 26개의 'x'를 갖는 리스트
(타이핑을 덜하기 위한 것일 뿐)

```
    key = ['x']*len(LETTERS)
```

```
    for cipher_char in LETTERS:
```

후보키 맵의 각 문자에 대하여

```
        if len(letterMap[cipher_char]) == 1:
```

문자에 대응되는 후보키가 한 개이면 확정

```
            key_index = LETTERS.find(letterMap[cipher_char][0])
```

```
            key[key_index] = cipher_char
```

```
        else:
```

```
            ciphertext = ciphertext.replace(cipher_char.lower(), '_')
```

```
            ciphertext = ciphertext.replace(cipher_char.upper(), '_')
```

문자에 대응되는 후보키가 여러 개이면
문자를 '_'로 바꾸어 복호화에 반영되지
않도록 한다.

확정된 문자가 아닌 것은
'x'를 그대로 둔다.

```
    key = ''.join(key)
```

```
    return SubstCipher_lib.subst_decrypt(key, ciphertext)
```

복호화 예제

- 복호화 전략

- 암호문 분석으로 얻는 후보키 맵으로부터 암호키를 만든다.
- 암호키로 암호문을 복호화한다.

```
cipher_file_name = 'c0401-1.txt'
decrypt_file_name = 'dict_attack0401-1.txt'
print('Ciphertext File Name = ', cipher_file_name)
print('Decrypted File Name = ', decrypt_file_name)

ciphertext = SubstCipher_lib.ReadFile(cipher_file_name)

letterMapping = hackSubst(ciphertext)
print('Mapping: ', letterMapping)

decrypt_data = decryptwithMapping(ciphertext, letterMapping)

SubstCipher_lib.WriteFile(decrypt_file_name, decrypt_data)
```

알파벳 후보 맵

Mapping: {'A': ['D'], 'B': ['E'], 'C': ['F'], 'D': ['G'], 'E': ['H'], 'F': ['L'], 'G': ['M'], 'H': ['N'], 'I': ['I'], 'J': ['J'], 'K': ['K'], 'L': ['O'], 'M': ['P'], 'N': ['T'], 'O': ['U'], 'P': ['V'], 'Q': [], 'R': ['R'], 'S': ['S'], 'T': ['W'], 'U': ['X'], 'V': ['A'], 'W': ['B'], 'X': ['C'], 'Y': ['Y'], 'Z': ['Z']}

복호화한 결과
(해결하지 못한 '가 보임')

Since Bitcoin arrived in 2009, there has been growing economic and ideological interest in perfecting money that is native to the internet, and its popularity has sparked thousands of iterations. Blockchain-based digital money was indeed supposed to make our financial systems more free, fair and transparent. But where visionaries saw improvements over traditional finance, criminal enterprises saw opportunities. Sure, the traditional financial system is regularly used by traffickers and terrorists, but cryptocurrencies have some attributes that are especially attractive for illicit activity. Unlike payments through banks, blockchain-based transactions settle in a matter of minutes. There's no central authority to handle disputes, and transfers are irreversible. One currency can be rapidly traded for another. Security company CipherTrace notes that while only a fraction of Bitcoin transactions is used for crime, it's also true that "nearly all dark market commerce is transacted in cryptocurrencies."

평문

Cryptanalysis

(암호분석)

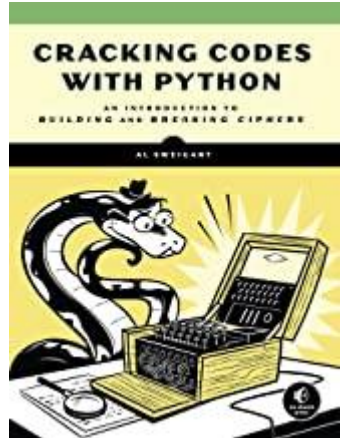
Vigenere Cipher Attack

2023. 3

목차

1. Vigenere Cipher
2. IC(Index of Coincidence) 구현
3. Vigenere Cipher의 해독

**Cracking codes with Python:
An Introduction to Building and Breaking Ciphers
by Al Sweigart (2018)**



Vigenere 암호 개요

- Vigenere 암호
 - Vigenere 테이블: 공개 정보
 - 각 행은 Caesar 암호 치환법 (key=0,1,2,..., 25)
 - 비밀번호(암호키): 비공개 정보
 - 비밀번호에 따라 선택된 행의 치환방식을 적용
 - 암호문에는 평문의 통계적 특성이 나타나지 않음
- 예
 - 암호키: ABC ABCABCABCABC
 - 평문: kookmin math 012012012012
 - 암호문: kpqknkn oauj kookmin math kpqknkn oauj

Vigenere Tableau

	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
a	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
b	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	a
c	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
d	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
e	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
f	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
g	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
h	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
i	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
j	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
k	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
l	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
m	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
n	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
o	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
p	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
q	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
r	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
s	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
t	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
u	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
v	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
w	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
x	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
y	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
z	z	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

Vignere 암호화

• 암호화 함수

```
Alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
#-- Vignere 암호화
def vignere_encrypt(key, msg):
    result = ''
    #암호키(단어)를 리스트로 만든다. (유효성 확인도 추가하자)
    key_list = list(key.upper())
    key_pos= 0
    for ch in msg:
        if ch.upper() in Alphabet:
            key_ch = Alphabet.find(key_list[key_pos])
            idx = Alphabet.find(ch.upper())
            if ch.isupper():
                result += Alphabet[(idx+key_ch)%26].upper()
            else:
                result += Alphabet[(idx+key_ch)%26].lower()
        else:
            result += ch
        key_pos = (key_pos + 1) % len(key)
    return result
```

암호키를 순환하여
적용한다
예: ABCDABCDABCD....

```
my_key = 'ABCD'
msg1 = "What is Caesar cipher? Is it secure?"
CT = vignere_encrypt(my_key, msg1)
print('msg =', msg1)
print('key =', my_key)
print('ciphertext =', CT)
```



msg = What is Caesar cipher? Is it secure?
key = ABCD
ciphertext = Wicw ju Cbgvas fiqjhr? Ls kw tgfusg?

한글자 암호화에
Caesar 암호화 같은
원리를 사용한다

Vignere 복호화

- 복호화 함수

```
#!/usr/bin/perl
#-- vignere 복호화
def vignere_decrypt(key, msg):
    result = ''
    #암호키(단어)를 리스트로 만든다. (유효성 확인도 추가하자)
    key_list = list(key.upper())
    key_pos= 0
    for ch in msg:
        if ch.upper() in Alphabet:
            key_ch = Alphabet.find(key_list[key_pos])
            idx = Alphabet.find(ch.upper())
            if ch.isupper():
                result += Alphabet[(idx-key_ch)%26].upper()
            else:
                result += Alphabet[(idx-key_ch)%26].lower()
        else:
            result += ch
        key_pos = (key_pos + 1) % len(key)
    return result
```

```
my_key = 'ABCD'
msg1 = "What is Caesar cipher? Is it secure?"
CT = vignere_encrypt(my_key, msg1)
print('msg =', msg1)
print('key =', my_key)
print('ciphertext =', CT)
recPT = vignere_decrypt(my_key, CT)
print('recPT =', recPT)
```



```
msg = What is Caesar cipher? Is it secure?
key = ABCD
ciphertext = Wicw ju Cbgvas fiqjhr? Ls kw tgfusg?
recPT = What is Caesar cipher? Is it secure?
```

암호화와 반대로
키를 빼주면 된다.

Vigenerere 암호 – 파일 암호화

• 평문 파일 읽기

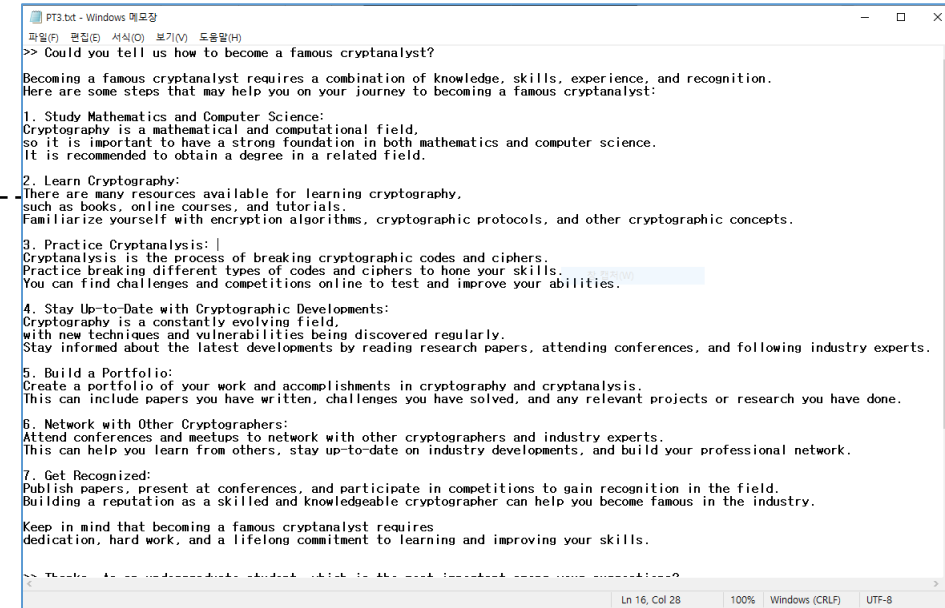
```
import vigenerereLib
import os, sys

in_file = 'PT3.txt'

#== 예외처리를 여기에 놓자! if not os.path.exists(in_file): ...

#-- 입력파일에서 읽기
InFileObj = open(in_file)
PT = InFileObj.read()
InFileObj.close()

print('input file (%s) : ' %(in_file), end = '')
print(PT[:10], "...", PT[-10:])
```



input file (PT3.txt) : >> Could y ... I ChatGPT)

Vignere 암호 - 파일 암호화

- 암호화하고 파일에 쓰기

```
key = 'ABCD'
CT = VigenereLib.vigenere_encrypt(key, PT)

out_file = 'CT3.txt'

#-- 덮어쓸지 물어보기 if os.path.exists(out_file):...

OutFileObj = open(out_file, 'w')
OutFileObj.write(CT)
OutFileObj.close()

print('Output file (%s) : ' %(out_file), end='')
print(CT[:10], '...', CT[-10:])
```

Output file (recPT3.txt) : >> Could y ... I ChatGPT)



```
CT3.txt - Windows  메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
>> Fovng zax ugoi wv iaz uq bfermf d gcpovu csastbpdzluz?

Bfermip b ianax eyuqvadnbnsu uerlwrf a ermckaukrn qi lprwmggf, sklolt, eyhrhgacq, afr fergokwipp.
Bovt drf vonga sussx vkau paz kemr ypw oo bovt ipwnfa tp eedapioi a hdmvpu dtpbuscaamvat:

1. Vtvfb Ncwhftodtjev bpg Dappvhr Ufifpfe:
Csastpuaqib ju a odtigpaufkam dne fonrxtbvloco skhle,
tu iu ls kppotwaov tp kawg a uwrppj gaxncwipp io eouj mbvkencwidu aof cposuug teleoh.
Lt kv safonohnegu uq ocvdio d egjrf io d sgaougg skhle.

2. Lfcun Euyavrgscshz:
Ujhrf drf paa rfrurehs cyaindbmg fpt lfcunipj dtpuqairbrky,
svk bo bogns, rnmkce erusuhs, dne wuuauibv.
Iankobltzf bovtvnh wjvk fprzrwipp amirrvkat, csastpuaqile ruouafomu, bpg pykes frzwohtdopkf daqacfrws.

3. Srbewida Csastbpdzluz:
Csastbpdzluz kv ujh atrcfv ph bsadkpij dtpuqairbrkid foegv bpg dkshtfv.
Srbewida bsadkpij ekiffthnu wvavv ph cpfhs cad elipius vr iage aru vkjnos.
Ypw bpe cfps djdmgafu aof cposuokwippv ppoio tp wetv aof inruus ypw bdllivlet.

4. Suvb Yr-to-Gaug wjvk Dtpuqairbrkid Gwsgooahnuu:
Euyavrgscshz ls c cpvotbwlz hvenvio figod,
wjvk oaz uafoktufu aof vnoesceakwifu bfpag flsdqayegv sgjumculz.
Tvdv kapttee dpbwv ujh mcwety dfxhlpreevv ca rfcgioi rfuhasck acesu, bwveoflnh foohhrffpet, aof fpmooxkg kadvuwrz hxagutt.

5. Bvkod C Pwtfnio:
Csastd d qauksoip rf aru zsoh aof adermanlfnh io frzwohtdolu aof csastpuaqib.
Tikv dca jptlflv acesu ypw hbmh xtlitga, ekannhnv zax icye urlage, cad cax thlxidnu srplhcu oo uetgdrid ypw hbhx eage.

6. Qeuvrri ziuj Qujhr Euyavrgscshftv:
Dtudt ensgueoehs cad cawess vr oawpnt xkwh qwhft csastpuaqihrt dne Inewtsa eyhruu.
Wkit fao kemr ypw lfcun huon rtigus, viba uq wo'fdrtf rn kadvuwrz gwsgooahnuu, bpg cwille bovt psaietulooco oawpntn.

7. Iht Thopiaiaag:
Suncisi sagaus, srfuhnu dt ensgueoehs, dne sasvlcirdf In eragwukrnt wo idio uedainjvlooo In vke hleamf.
Dxiatfnd d ssguucippe at d tmlimg bpg lprwmggfcelf frzwohtdolu dca isop aru dchop gcpovu io whf Inewtsa.

Mheg In olne whbv bfermip b ianax eyuqvadnbnsu uerlwrf
eggidwipp, icud yrri, aof a niffnrnh fonoltnqat vr mgdrokag cad kppsayioi ypwu tmlimu.

> Windows - C:\Program Files\Windows NT\Winlogon\...
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

Vigenere 암호 - 파일 복호화

• 복호화하고 파일에 쓰기

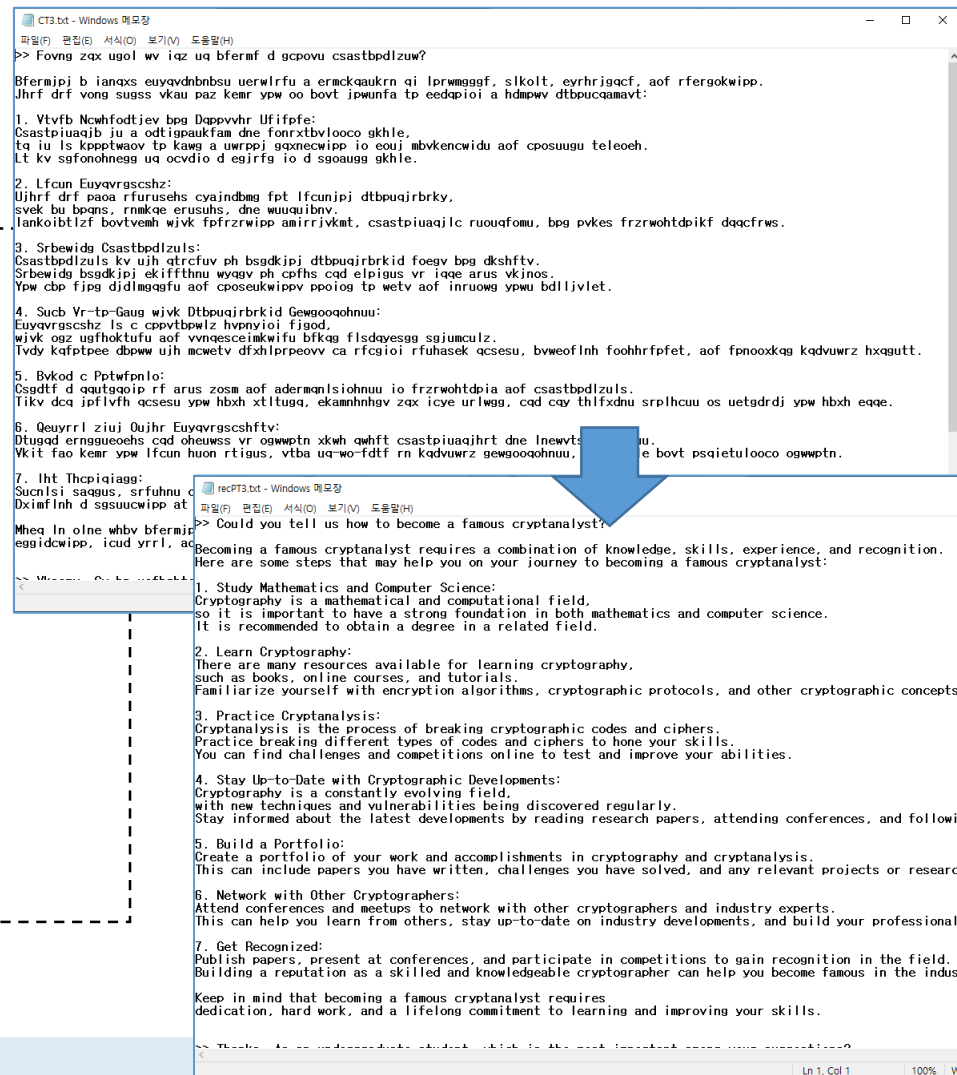
```
key = 'ABCD'
recPT = VigenereLib.vigenere_decrypt(key, CT)

out_file = 'recPT3.txt'

#-- 덮어쓰지 물어보기 if os.path.exists(out_file):...

outFileObj = open(out_file, 'w')
outFileObj.write(recPT)
outFileObj.close()

print('Output file (%s) : ' %(out_file), end='')
print(recPT[:10], '...', recPT[-10:])
```



```
CT3.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움(H)
>> Fovng zqx ugoi wv iaz uq bfermf d gcpovu csastbpdizuw?

Bfermipj b ianqxs euyqvdbnbsu uerwlrfa a erackaukrn qi lprwmgggf, slkolt, eyhrigqcf, aof rfergokwipp.
Jhrf drf vong sugss vkau paz kear ypw oo bovt ipwunfa tp eedapioi a hdmwv dtbpucaamavt:

1. Vtvtb Nwhfodtjev bpg Dappvvr Ufifpfe:
Csastpiaqib iu a odtispaufam dne fonxtbvlooco gkhle.
tq iu ls kppptaov tp kawg a uwrppj gqxnecwipp io eouj mbvkenwidu aof cposuugu teleoeh.
Lt kv sgfonohneg uq ocvdio d egjrfq io d sgougg gkhle.

2. Lfcun Euyavrgscshz:
Ujhrf drf paa rfurusehs cyajndbm fpt lfcunipj dtbpuqirbrky.
svek bu bpans, rnmkqe erusuhs, dne wuquibnv.
lankoibtlzf bovtvnmh wjvk ffrzrwipp amirrkmt, csastpiaqibc ruouqfomu, bpg pvkes frzrwohtdpikf dqacfrws.

3. Srbewidg Csastbpdizuls:
Csastbpdizuls kv ujh qtrcfuv ph bsadkipj dtbpuqirbrkid foegv bpg dkshtfv.
Srbewidg bsadkipj ekiffthnu wyqav ph cpfhs cad elpigs vr iqge arus vkims.
Ypw cbp figp djdmgagfu aof cpousekwippv ppoio tp wetv aof inruowg ypwu bdlijvlet.

4. Subc Vr-to-Gaug wjvk Dtbpuqirbrkid Gwagooahnuu:
Euyavrgscshz ls c cpvtpwlvz hvnyioi figod.
wjvk ogz ughoktufu aof vnaqscemkwifu bfgq fisdqyesg sgjumculz.
Tvdv kqfptee dbpww ujh mcwetv dfxhprpeovv ca rfciogi rfuasek qcesu, bwecoflnh foohrpfet, aof fonooxqg kadvurz hxagutt.

5. Bvkod c Pptwfpno:
Csastbpdizuls rf arus zosm aof adernanlsiohnuu io frzrwohtdpia aof csastbpdizuls.
Tikv dca jpflyth qcesu ypw hbxh xltuga, ekamnnhgv zqx icye urlwgg, cad cay thifxdu srplhcuu os uetgdrj ypw hbxh eqqe.

6. Qeuyrrl ziuq Ouhr Euyavrgscshftv:
Dtuqod erngusehs cad oheussr ogawpbn xkwh qwhft csastpiaqihrt dne Inewts...
Vkit fao kear ypw lfcun huon rtigus, vtba uq wo-fdfr in kadvurz gwagooahnuu, ... bovt psqietuloooc ogawpbn.

7. Iht Thcpiaqag:
Sucnisi saquss, srfuhnu d
Dximflnh d sgauucwipp at
Mheq ln olne whbv bfermipj
egsidcwipp, icud yrrl, ad

>> Could you tell us how to become a famous cryptanalyst?
Becoming a famous cryptanalyst requires a combination of knowledge, skills, experience, and recognition.
Here are some steps that may help you on your journey to becoming a famous cryptanalyst:

1. Study Mathematics and Computer Science:
Cryptography is a mathematical and computational field,
so it is important to have a strong foundation in both mathematics and computer science.
It is recommended to obtain a degree in a related field.

2. Learn Cryptography:
There are many resources available for learning cryptography,
such as books, online courses, and tutorials.
Familiarize yourself with encryption algorithms, cryptographic protocols, and other cryptographic concepts.

3. Practice Cryptanalysis:
Cryptanalysis is the process of breaking cryptographic codes and ciphers.
Practice breaking different types of codes and ciphers to hone your skills.
You can find challenges and competitions online to test and improve your abilities.

4. Stay Up-to-Date with Cryptographic Developments:
Cryptography is a constantly evolving field,
with new techniques and vulnerabilities being discovered regularly.
Stay informed about the latest developments by reading research papers, attending conferences, and following industry experts.

5. Build a Portfolio:
Create a portfolio of your work and accomplishments in cryptography and cryptanalysis.
This can include papers you have written, challenges you have solved, and any relevant projects or research you have done.

6. Network with Other Cryptographers:
Attend conferences and meetups to network with other cryptographers and industry experts.
This can help you learn from others, stay up-to-date on industry developments, and build your professional network.

7. Get Recognized:
Publish papers, present at conferences, and participate in competitions to gain recognition in the field.
Building a reputation as a skilled and knowledgeable cryptographer can help you become famous in the industry.

Keep in mind that becoming a famous cryptanalyst requires
dedication, hard work, and a lifelong commitment to learning and improving your skills.

>> Thanks to an underground student, which is the most important reason your success?

Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

IC – Index of Coincidence

- Index of Coincidence의 정의

- 주어진 메시지에서 두 글자를 비복원 추출할 때 같은 글자가 나올 확률

$$I_c = \sum_{c \in \text{alphabet}} \frac{n_c(n_c - 1)}{N(N - 1)} = \frac{n_a(n_a - 1)}{N(N - 1)} + \frac{n_b(n_b - 1)}{N(N - 1)} + \dots + \frac{n_z(n_z - 1)}{N(N - 1)}$$

- 예:

- 영문의 경우: $I_c \approx 0.0639$
- 랜덤한 알파벳 나열: $I_c \approx 0.038$

IC – Index of Coincidence

- Index of Coincidence 함수

```
UpAlphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
def IC(msg):
    AlphaDic = {}
    for ch in UpAlphabet:
        AlphaDic[ch] = 0

    num_alpha = 0
    for ch in msg:
        if ch.upper() in UpAlphabet:
            AlphaDic[ch.upper()] += 1
            num_alpha += 1

    ic = 0
    for ch in UpAlphabet:
        ic += AlphaDic[ch]*(AlphaDic[ch]-1)
    ic /= num_alpha*(num_alpha-1)
    return ic
```

```
msg1 = "what is Caesar cipher? Is it secure?"
msg2 = "Zkdw lv Fdhvdu flskhu? Lv lw vhfuxh?"
print('msg1 = ', msg1[:30], '...')
print('msg2 = ', msg2[:30], '...')
print('Index of Coincidence, IC(msg1) = %6.4f' %(IC(msg1)))
print('Index of Coincidence, IC(msg2) = %6.4f' %(IC(msg2)))
```



```
msg1 = What is Caesar cipher? Is it s ...
msg2 = Zkdw lv Fdhvdu flskhu? Lv lw v ...
Index of Coincidence, IC(msg1) = 0.0767
Index of Coincidence, IC(msg2) = 0.0767
```

Vigenere 암호 공격 (1) 키 길이 찾기

```
in_file = 'CT3.txt'
InFileObj = open(in_file)
CT = InFileObj.read()
InFileObj.close()
MAX_KEY_LENGTH = 8 # 암호키의 최대길이 설정(가정)
keylen_candidate = 0
max_ic = 0.0
for key_len in range(1, MAX_KEY_LENGTH+1):
    sub_msg = ''
    idx = 0
    while idx < len(CT):
        sub_msg += CT[idx]
        idx += key_len
    sub_ic = EngDicLib.IC(sub_msg)
    if max_ic < sub_ic:
        max_ic = sub_ic
        keylen_candidate = key_len
    print('key_len =', key_len, ':', end='')
    print('sub_msg', sub_msg[:10], "...", '( length =', len(sub_msg), ')\t', end='')
    print('IC(sub_msg)= %6.4f' %(sub_ic))
```

```
input file (CT3.txt) : >> Fovng z ... K CicwGQV) ( length = 3072 )
key_len = 1 :sub_msg >> Fovng z ... ( length = 3072 ) IC(sub_msg)= 0.0435
key_len = 2 :sub_msg > on q glw ... ( length = 1536 ) IC(sub_msg)= 0.0493
key_len = 3 :sub_msg >Fnz owi ... ( length = 1024 ) IC(sub_msg)= 0.0428
key_len = 4 :sub_msg >o l bm ... ( length = 768 ) IC(sub_msg)= 0.0598
key_len = 5 :sub_msg >vqo uedos ... ( length = 615 ) IC(sub_msg)= 0.0424
key_len = 6 :sub_msg >n w e utu ... ( length = 512 ) IC(sub_msg)= 0.0477
key_len = 7 :sub_msg >ggibdub?r ... ( length = 439 ) IC(sub_msg)= 0.0444
key_len = 8 :sub_msg >l mot?ma ... ( length = 384 ) IC(sub_msg)= 0.0597
key length = 4
```

Vigenere 암호 공격 - 키 찾기 아이디어

- 암호키의 첫글자와 두번째 글자의 차이를 찾는 방법

```
key_list = [0]*keylen_candidate
key_ch_candidate = 0
max_ic = 0.0
for key_ch in range(0,26):
    sub_msg = ''
    idx = 0
    while idx < len(CT):
        sub_msg += CT[idx]
        if (idx+1) < len(CT):
            sub_msg += CaesarLib.caesar_decrypt(key_ch, CT[idx+1])
        idx += keylen_candidate
    sub_ic = EngDicLib.IC(sub_msg)
    if max_ic < sub_ic:
        max_ic = sub_ic
        key_ch_candidate = key_ch
    print('key_ch =', key_ch, ':', end='')
    print('sub_msg', sub_msg[:10], "...", '( length =', len(sub_msg), ')\t', end='')
    print('IC(sub_msg)= %6.4f' %(sub_ic))
    print('key_ch_candidate =', key_ch_candidate)
```

앞의 키 길이 찾기에서
얻은 값
예: 암호키=ABCD 이면, 4

key_ch = 0 :sub_msg >>ov z ul ... (length = 1536)	IC(sub_msg)= 0.0492
key_ch = 1 :sub_msg >>ou y tl ... (length = 1536)	IC(sub_msg)= 0.0613
key_ch = 2 :sub_msg >>ot x sl ... (length = 1536)	IC(sub_msg)= 0.0507
key_ch = 3 :sub_msg >>os w rl ... (length = 1536)	IC(sub_msg)= 0.0514
(중간 생략)	
key_ch = 23 :sub_msg >>oy c xl ... (length = 1536)	IC(sub_msg)= 0.0523
key_ch = 24 :sub_msg >>ox b wl ... (length = 1536)	IC(sub_msg)= 0.0471
key_ch = 25 :sub_msg >>ow a vl ... (length = 1536)	IC(sub_msg)= 0.0490
key_ch_candidate = 1	

Vigenere 암호 공격 (2) 키 찾기 (상대적)

- 암호키의 첫글자를 기준으로 다른 위치의 키를 나타내기
 - 예: 암호키가 ABCD이면, [0,1,2,3]

```
key_list = [0]*keylen_candidate
for key_pos in range(1,keylen_candidate):
    key_ch_candidate = 0
    max_ic = 0.0
    for key_ch in range(0,26):
        sub_msg = ''
        idx = 0
        while idx < len(CT):
            sub_msg += CT[idx]
            if (idx+key_pos) < len(CT):
                sub_msg += CaesarLib.caesar_decrypt(key_ch, CT[idx+key_pos])
            idx += keylen_candidate
        sub_ic = EngDicLib.IC(sub_msg)
        if max_ic < sub_ic:
            max_ic = sub_ic
            key_ch_candidate = key_ch
    key_list[key_pos] = key_ch_candidate
    print('key[%d] : key_ch_candidate = %d' %(key_pos, key_ch_candidate))
```

앞의 키 길이 찾기에서
얻은 값
예: 암호키=ABCD 이면, 4

key[1] : key_ch_candidate = 1
key[2] : key_ch_candidate = 2
key[3] : key_ch_candidate = 3

Vigenere 암호 공격 (3) 암호키 찾기

```
for key_ch in range(0,26):
    dec_msg = ''
    key_pos= 0
    for ch in CT:
        key_now = (key_ch + key_list[key_pos]) % 26
        dec_msg += CaesarLib.caesar_decrypt(key_now, ch)
        key_pos = (key_pos + 1) % keylen_candidate
```

```
eng_percent = EngDicLib.percentEnglishWords(dec_msg)
```

```
print('key_ch =', key_ch, ':', end='')
print('dec_msg', dec_msg[:10], "...", '( length =', len(dec_msg), ')\t', end='')
print('Eng(dec_msg)= %5.2f %%' %(eng_percent*100))
```

```
if EngDicLib.isEnglish(dec_msg):
    key_0_candidate, rightPT = key_ch, dec_msg
```

```
if key_0_candidate >= 0:
    rightkey = ''
    for idx in key_list:
        rightkey += VigenereLib.Alphabet[(key_0_candidate + idx) % 26]
```

```
print('right key =', rightkey)
print('PT = ', rightPT[:20], '...', rightPT[-10:])
```

```
key_ch = 0 :dec_msg >> Could y ... ( length = 3072 ) Eng(dec_msg)= 59.35 %
key_ch = 1 :dec_msg >> Bntkc x ... ( length = 3072 ) Eng(dec_msg)= 0.00 %
key_ch = 2 :dec_msg >> Amsjb w ... ( length = 3072 ) Eng(dec_msg)= 0.00 %
```

(중간 생략)

```
key_ch = 24 :dec_msg >> Eqwnf a ... ( length = 3072 ) Eng(dec_msg)= 0.00 %
key_ch = 25 :dec_msg >> Dpvme z ... ( length = 3072 ) Eng(dec_msg)= 3.50 %
```

right key = ABCD

PT = >> Could you tell us ... I ChatGPT)

영어단어의
비율을
확인해본다.

이 줄의 의미를
생각해 보세요!

Cryptanalysis (암호분석)

암호공격과 안전성 개념

2023.3

Cryptanalysis - 암호 분석(공격)

- 암호분석의 목표
 - 실용적 목표: 암호문과 관련된 **암호키**, **평문의 정보**를 획득
 - 이론적 목표: 암호 설계자의 주장에서 모순을 발견
- 안전성 분석의 고려사항
 - 공격 조건 (공격 시나리오, 모델)
 - 가용 자원 (공격자의 능력)
 - (가능한 경우만) 사용 환경의 특이 사항

공격자의 자원(resource)

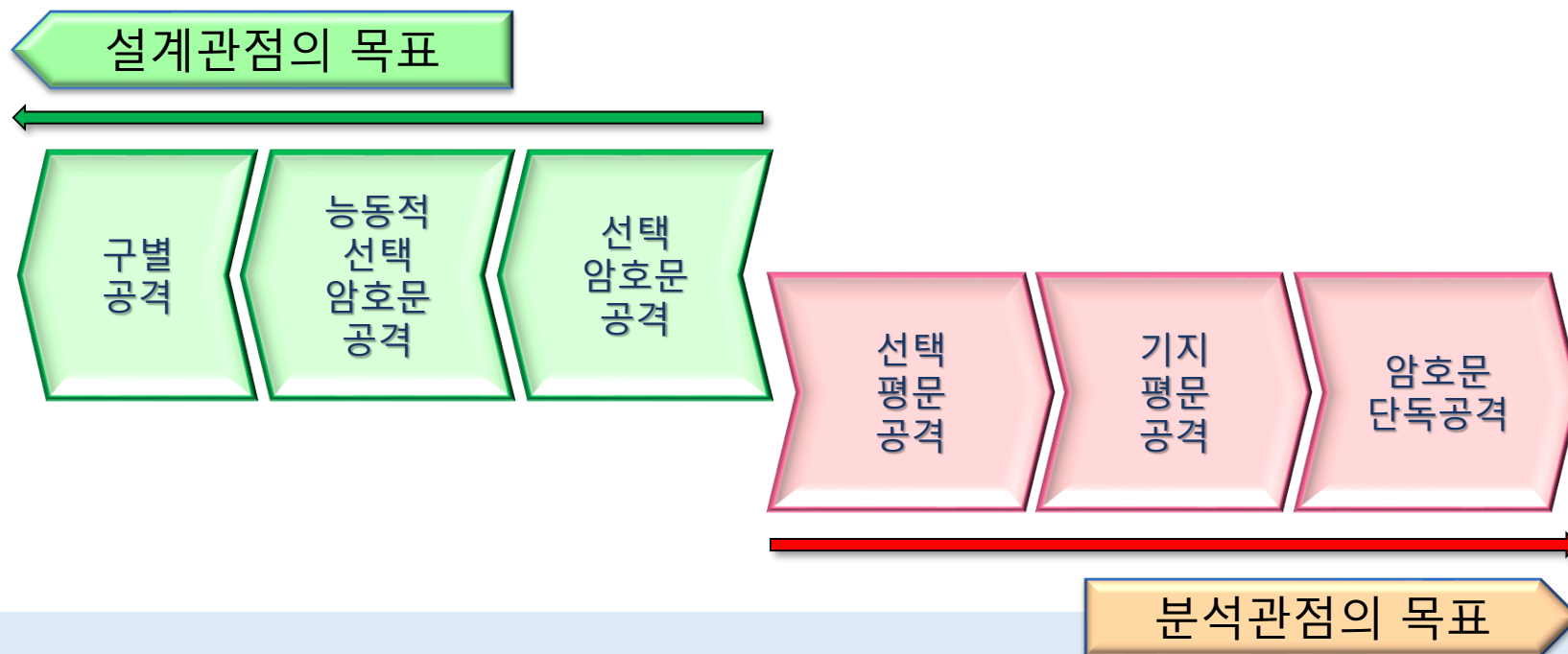
- 공격에 필요한 기본 자원들
 - Computing Power
 - Memory
 - Data(Plaintext/Ciphertext/Key)
- 부가정보(구현정보)
 - Blackbox Cryptography
 - Greybox Cryptography
 - Whitebox Cryptography
- 새로운 자원의 도입
 - DNA/Molecular Computing
 - Quantum Computing

공격 모델

- Ciphertext Only Attack(암호문 단독 공격)
 - 암호문만으로 공격하는 방법 (+ 평문 정보 예상)
 - 예: 도청 등으로 수집한 암호문의 해독
- Known Plaintext Attack (기지평문 공격)
 - 획득한 평문과 암호문 쌍을 이용한 공격
 - 예: 헤더가 예측되는 암호문의 해독
- Chosen Plaintext Attack (선택평문 공격)
 - 공격자가 원하는 평문, 암호문 쌍을 얻을 수 있는 환경의 공격
 - 예: 획득한 암호장비를 이용한 암호문의 해독
- Chosen Ciphertext Attack (선택암호문 공격)
 - 공격자가 복호화 능력까지 갖는 환경에서의 공격
 - 예: 공격자가 암호문을 만들어 복호기에 넣어볼 수 있는 환경

공격과 방어

- 설계자 관점
예상 공격 모델에 대한 안전성 보장을 목표로
- 공격자 관점
가능한 적은 자원으로 공격하는 방법을 목표로

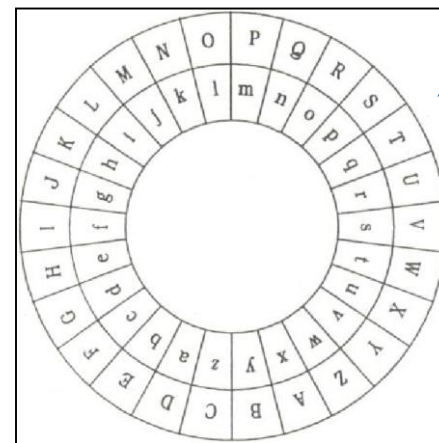
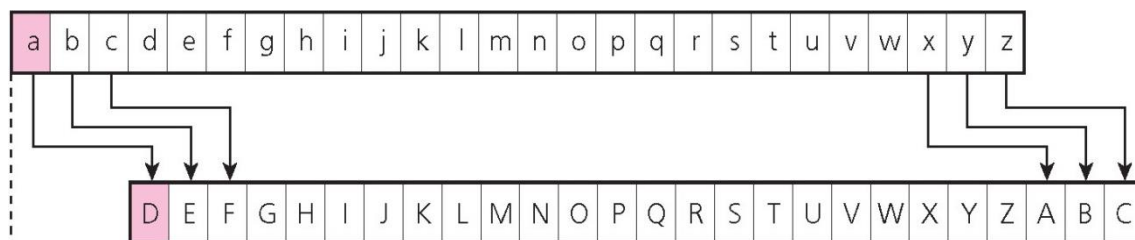


안전한 암호의 조건(1) - 암호키 공간

- 씨저 암호(Caesar Cipher)
 - 평문(P), 암호문(C), 암호키(K)의 관계

$$C = E_K(M) = M + K \bmod 26$$

K=3 인 경우

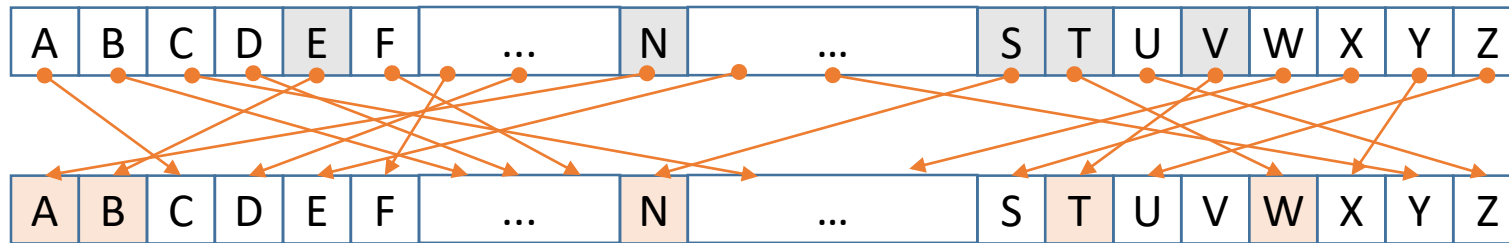
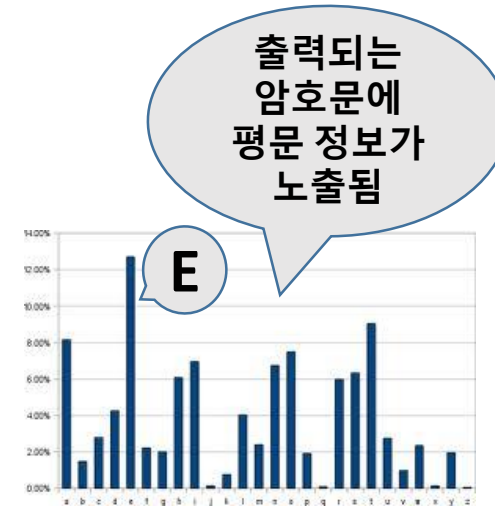


암호키의
종류(26가지)가
너무 적어
안전하지 않음

안전한 암호의 조건(2) - 평문/암호문의 관계

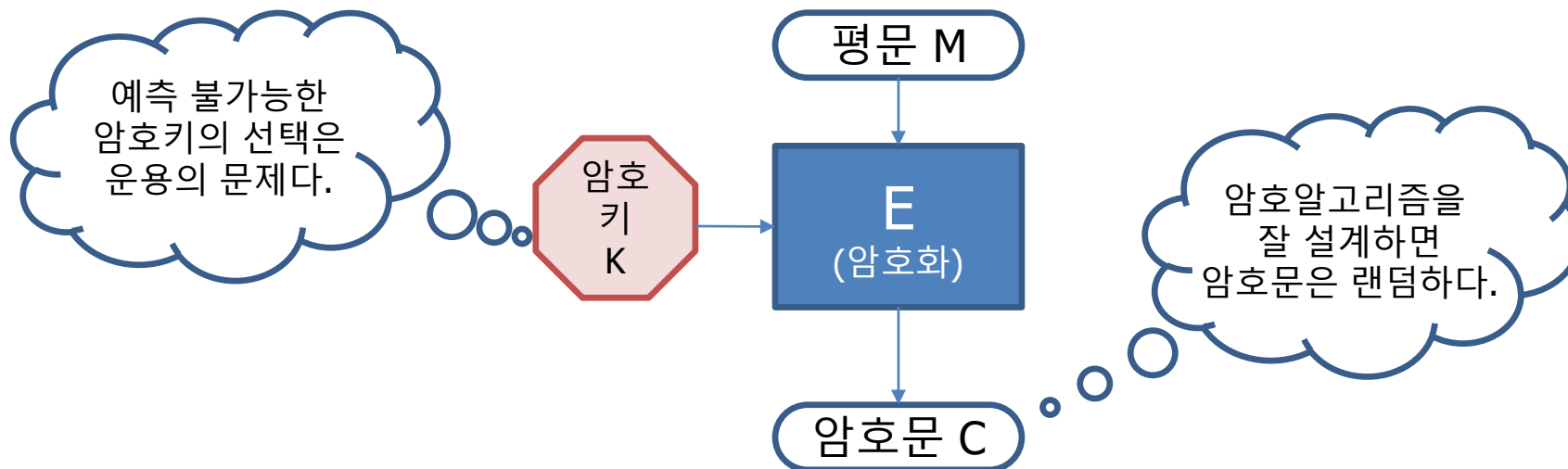
- 단순치환 암호
 - 암호키(=치환표): A~Z를 무작위로 섞는 방법
 - 암호키의 종류는 $26! \approx 4 \times 10^{26}$ 로 충분히 큼

E(seventeen) = nbtbawbba



안전한 비밀키 암호의 조건

- 안전한 암호화의 조건
 - 암호문은 **난수**와 구별할 수 없어야 한다.
(평문을 추측할 수 있는 정보를 제공하지 않아야 한다)
 - 암호키는 공격자가 예측할 수 없어야 한다.
(키 공간이 충분히 크고, **랜덤하게 선택**되어, 공격자가 키를 맞출 확률이 낮아야 한다)



Cryptanalysis (암호분석)

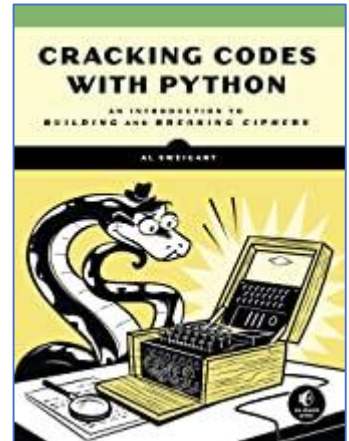
Python – Data Conversion
Toy Cipher TC20

2023.3

Contents

- ▶ Data type conversion
 - ▶ ASCII 문자
 - ▶ 정수
 - ▶ 바이트
- ▶ Block Ciphers
 - ▶ Design Rationale
- ▶ TC20 – Toy Cipher

**Cracking codes with Python:
An Introduction to Building and Breaking Ciphers
by Al Sweigart (2018)**



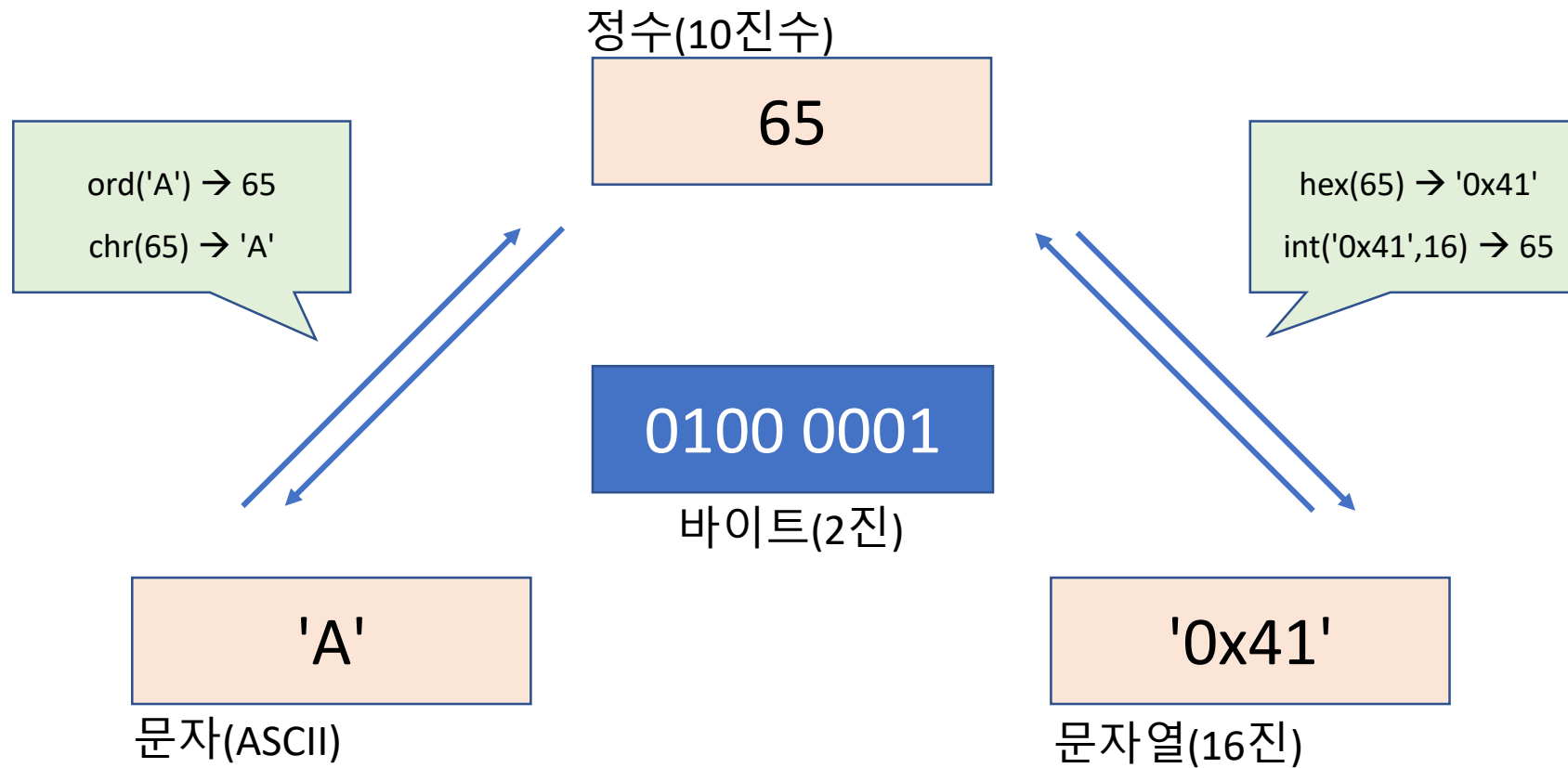
바이트 표현

- ▶ 1바이트 데이터의 다양한 표현 방법
 - ▶ 1바이트는 2^8 가지 데이터를 나타낼 수 있음
 - ▶ 10진수: 0, 1, 2, ..., 255
 - ▶ 16진수: 00, 01, 02, ..., FE, FF
 - ▶ ASCII 문자: 영문자, 숫자, 특수기호를 7비트로 표현한 것
- ▶ 예 (동일한 데이터의 여러가지 표현)
 - ▶ '1'(문자) \leftrightarrow 49(10진수) \leftrightarrow 31(16진수)
 - ▶ 'A'(문자) \leftrightarrow 65(정수) \leftrightarrow 41(16진수)
 - ▶ 'z'(문자) \leftrightarrow 122(정수) \leftrightarrow 7A(16진수)

ASCII

ASCII control characters			ASCII printable characters			Extended ASCII characters		
00	NULL	(Null character)	32	space	64	@	96	`
01	SOH	(Start of Header)	33	!	65	A	97	a
02	STX	(Start of Text)	34	"	66	B	98	b
03	ETX	(End of Text)	35	#	67	C	99	c
04	EOT	(End of Trans.)	36	\$	68	D	100	d
05	ENQ	(Enquiry)	37	%	69	E	101	e
06	ACK	(Acknowledgement)	38	&	70	F	102	f
07	BEL	(Bell)	39	'	71	G	103	g
08	BS	(Backspace)	40	(72	H	104	h
09	HT	(Horizontal Tab)	41)	73	I	105	i
10	LF	(Line feed)	42	*	74	J	106	j
11	VT	(Vertical Tab)	43	+	75	K	107	k
12	FF	(Form feed)	44	,	76	L	108	l
13	CR	(Carriage return)	45	-	77	M	109	m
14	SO	(Shift Out)	46	.	78	N	110	n
15	SI	(Shift In)	47	/	79	O	111	o
16	DLE	(Data link escape)	48	0	80	P	112	p
17	DC1	(Device control 1)	49	1	81	Q	113	q
18	DC2	(Device control 2)	50	2	82	R	114	r
19	DC3	(Device control 3)	51	3	83	S	115	s
20	DC4	(Device control 4)	52	4	84	T	116	t
21	NAK	(Negative acknowl.)	53	5	85	U	117	u
22	SYN	(Synchronous idle)	54	6	86	V	118	v
23	ETB	(End of trans. block)	55	7	87	W	119	w
24	CAN	(Cancel)	56	8	88	X	120	x
25	EM	(End of medium)	57	9	89	Y	121	y
26	SUB	(Substitute)	58	:	90	Z	122	z
27	ESC	(Escape)	59	;	91	[123	{
28	FS	(File separator)	60	<	92	\	124	
29	GS	(Group separator)	61	=	93]	125	}
30	RS	(Record separator)	62	>	94	^	126	~
31	US	(Unit separator)	63	?	95	_		
127	DEL	(Delete)						
128	Ç		160	á	192	Ł	224	Ó
129	ü		161	í	193	ł	225	ô
130	é		162	ó	194	Ł	226	õ
131	â		163	ú	195	ł	227	ö
132	ä		164	ñ	196	—	228	ø
133	à		165	Ñ	197	†	229	Õ
134	á		166	ª	198	ä	230	µ
135	ç		167	º	199	Å	231	þ
136	ê		168	¿	200	Ł	232	þ
137	ë		169	®	201	Œ	233	ú
138	è		170	¬	202	Œ	234	û
139	ï		171	½	203	Œ	235	ü
140	î		172	¼	204	Œ	236	ý
141	ì		173	ı	205	=	237	ÿ
142	Ä		174	«	206	†	238	—
143	Å		175	»	207	‡	239	·
144	É		176	„	208	ø	240	≡
145	æ		177	„	209	Ð	241	±
146	Æ		178	„	210	Ê	242	≡
147	ô		179	„	211	Ë	243	¾
148	ö		180	„	212	È	244	¶
149	ò		181	„	213	Ì	245	§
150	û		182	„	214	Í	246	÷
151	ù		183	„	215	Î	247	°
152	ÿ		184	„	216	Ï	248	°
153	Ö		185	„	217	„	249	°
154	Ü		186	„	218	„	250	°
155	ø		187	„	219	„	251	°
156	£		188	„	220	„	252	°
157	Ø		189	„	221	„	253	°
158	×		190	„	222	„	254	°
159	f		191	„	223	„	255	nbsp

바이트 데이터 변환함수



바이트 데이터 변환함수

▶ 변환함수의 활용

▶ ord(), chr(), hex(), int()

문자(ASCII)로 시작

```
ch1 = 'A'
num1 = ord(ch1)
hex1 = hex(num1)

list1 = []
list1.append(ch1)
list1.append(num1)
list1.append(hex1)

print(list1)
print(len(hex1))
```

['A', 65, '0x41']

정수(10진)로 시작

```
num2 = 66
ch2 = chr(num2)
hex2 = hex(num2)

list2 = []
list2.append(ch2)
list2.append(num2)
list2.append(hex2)

print(list2)
```

['B', 66, '0x42']

16진 문자열로 시작

```
hex3 = '0x43'
num3 = int(hex3,16)
ch3 = chr(num3)

list3 = []
list3.append(ch3)
list3.append(num3)
list3.append(hex3)

print(list3)
```

['C', 67, '0x43']

문자열 데이터

▶ 문자열을 리스트로

- ▶ 문자열을 각 문자로 나누어 리스트로 만든다.
- ▶ 향후 만들 블록암호의 입력으로 리스트를 사용한다.

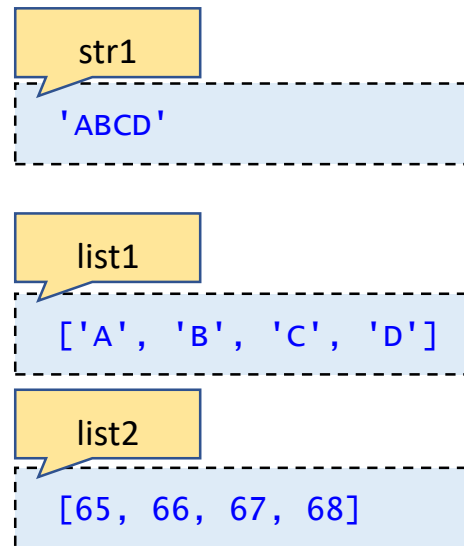
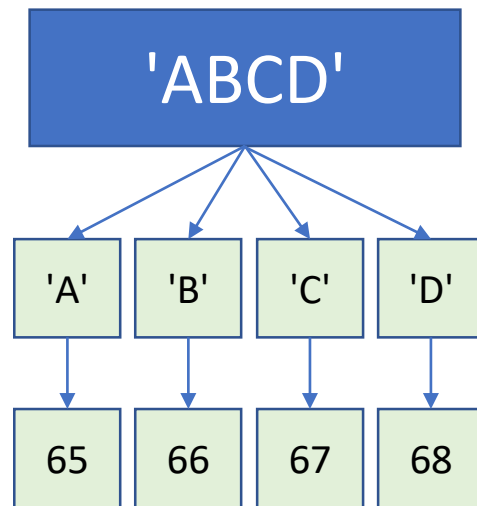
```
str1 = 'ABCD'
```

```
list1 = [ch for ch in str1]  
print(list1)
```

```
list2 = [ord(ch) for ch in str1]  
print(list2)
```

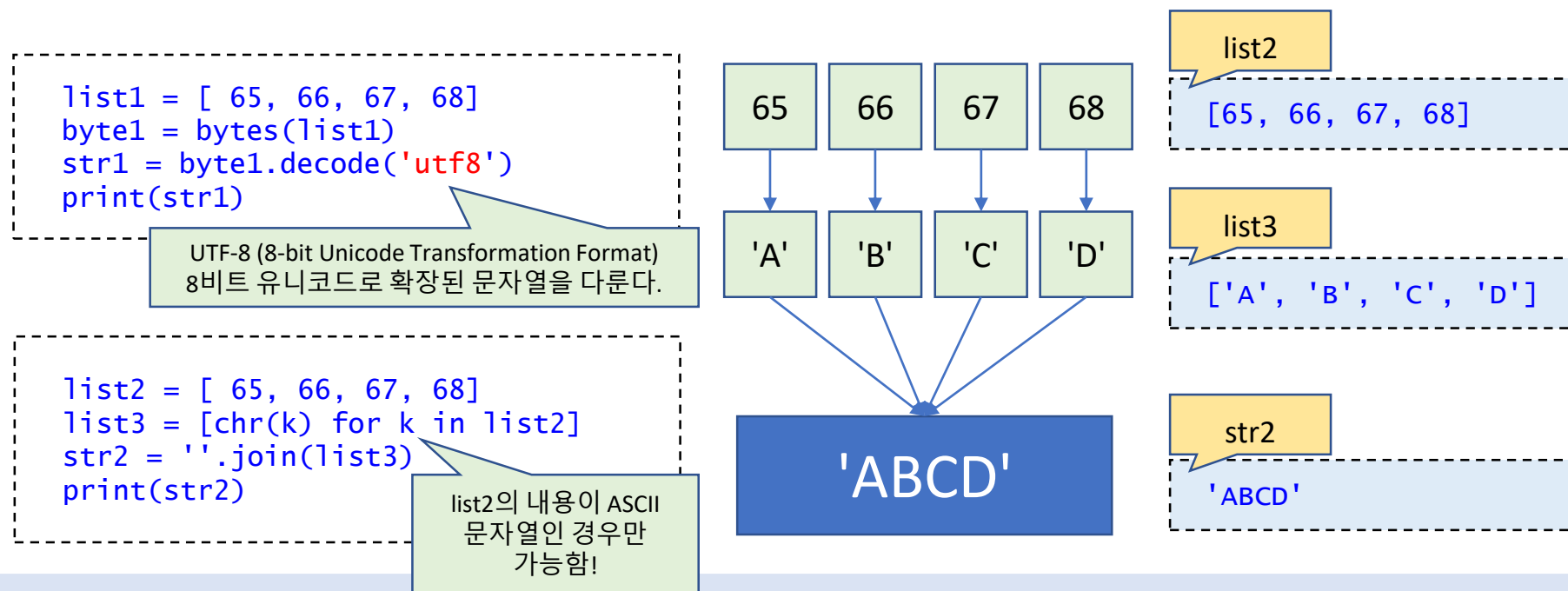
```
list3 = list(str1)
```

list1과 list3는
같은 결과



문자열 데이터

- ▶ 리스트를 바이트열/문자열로
 - ▶ 리스트의 내용을 합쳐 바이트열 또는 문자열로 만든다.
 - ▶ 향후 만들 블록암호의 출력인 리스트를 문자열로 변환한다.



바이트열

▶ 바이트열 vs 문자열

- ▶ 문자열(string): ASCII 문자로 구성된 배열
- ▶ 바이트열(bytes): 바이트(0~255)로 구성된 배열

▶ 변환 함수

- ▶ encode(): 문자열 → 바이트열
- ▶ decode(): 바이트열 → 문자열

▶ 리스트와 문자열/바이트열 변환

- ▶ list(), join(): 리스트 → 문자열
- ▶ list(), bytes(): 리스트 → 바이트열

바이트열 예제

▶ 문자열, 바이트열, 리스트 변환 예제

```
list1 = [65, 66, 67, 68]
byte1 = bytes(list1)
str1 = byte1.decode('utf8')
print(str1)
```

ABCD

```
str2 = 'ABCDE'
bytes2 = bytes(str2, 'utf8')
print(bytes2)
```

b'ABCDE'

```
str3 = 'ABCDE'
bytes3 = str3.encode('utf8')
list_s3 = list(str3)
list_b3 = list(bytes3)
print(list_s3)
print(list_b3)
```

['A', 'B', 'C', 'D', 'E']
[65, 66, 67, 68, 69]

bytes(str3, 'utf8')
와 동일한 결과

파일 입출력

- ▶ 데이터를 문자열 또는 바이트열로 변환 후 파일에 출력
 - ▶ 문자열 출력: ASCII 문자를 텍스트(text) 파일로 출력
 - ▶ 바이트열 출력: 바이트열을 이진(binary) 파일로 출력

```
list1 = [1, 2, 3, 4, 65, 66, 67, 68]
byte1 = bytes(list1)
```

```
f = open('data1.txt', 'w+b')
f.write(byte1)
f.close()
```

이진파일에는
'b'를 추가해야 한다

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	01	02	03	04	41	42	43	44									...ABCD

```
list2 = ['A', 'B', 'C', 'D']
str2 = ''.join(list2)
```

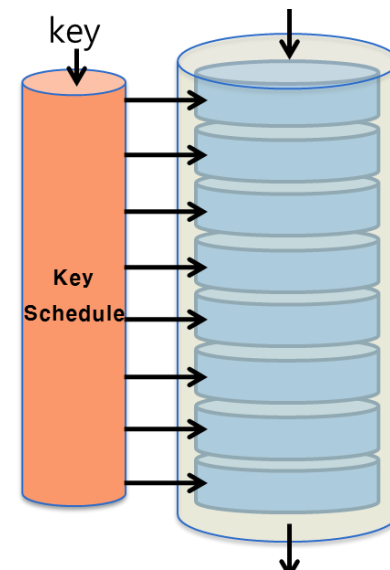
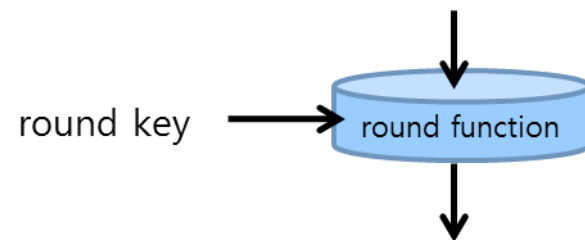
```
f = open('data2.txt', 'w+')
f.write(str2)
f.close()
```

Hex Editor
프로그램으로
확인한 파일저장 결과

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	41	42	43	44													ABCD

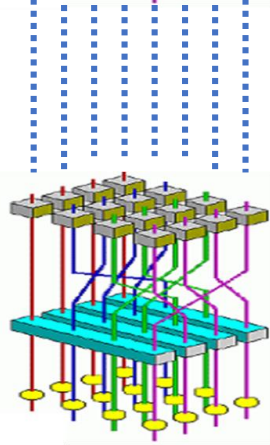
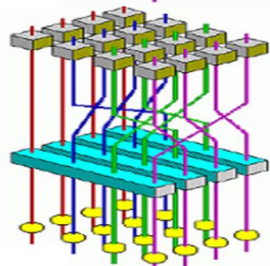
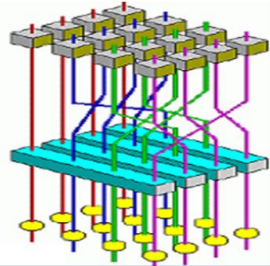
블록암호 개요

- ▶ Building block (구성요소)
 - ▶ Round function (라운드 함수)
 - ▶ Key schedule (키 스케줄)
- ▶ Design consideration
 - ▶ Security and Efficiency
 - ▶ Determine the number of rounds with security margin

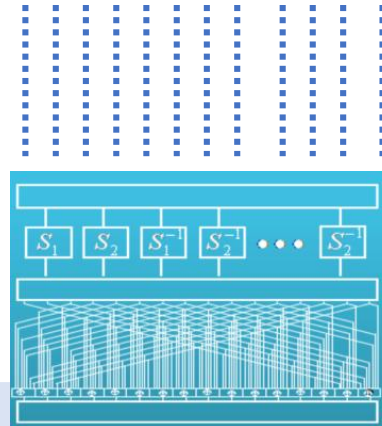
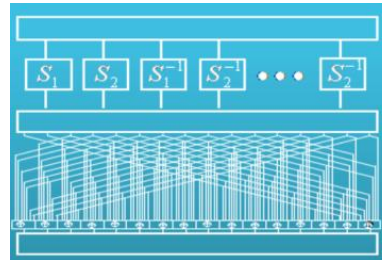
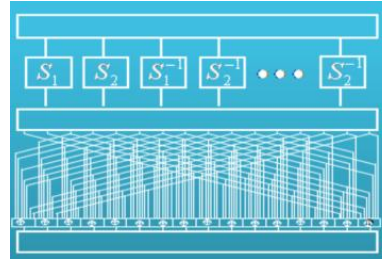


블록암호의 구조

미국 표준 AES

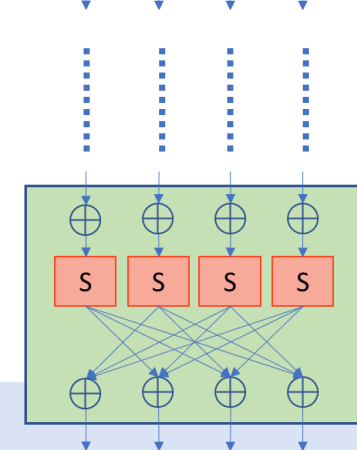
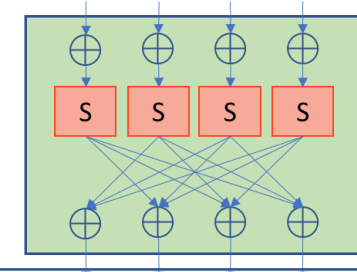
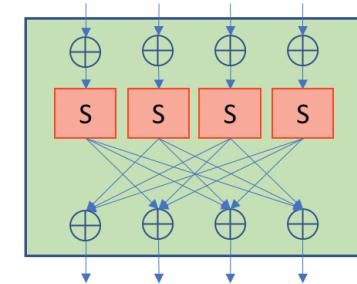


Round
function



한국 KS표준 ARIA

연습용 Toy Cipher
TC20



TC20 – AR (라운드키 적용)

▶ 라운드키 RK의 적용 방법

- ▶ 라운드키: $RK = (rk_0, rk_1, rk_2, rk_3)$
- ▶ XOR(Exclusive or)를 이용한 라운드 키 덧셈

$$(y_0, y_1, y_2, y_3) = (x_0 \oplus rk_0, x_1 \oplus rk_1, x_2 \oplus rk_2, x_3 \oplus rk_3)$$

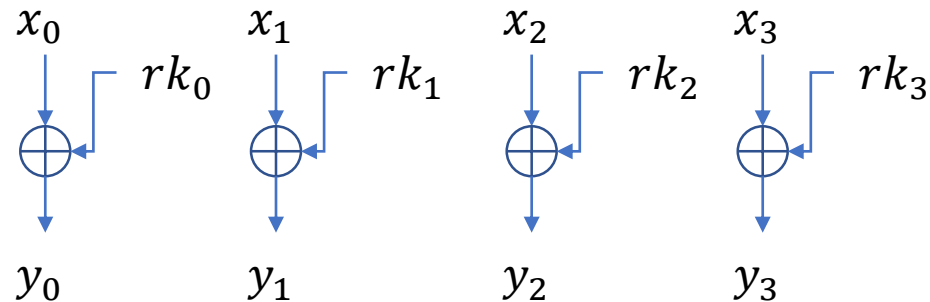
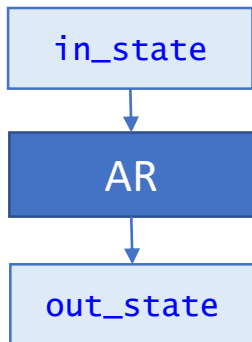
▶ 키 스케줄

- ▶ 단순한 구조를 위해 키 스케줄을 사용하지 않음
- ▶ 라운드키를 암호키 32비트와 동일한 값으로 사용함

TC20 – AR (라운드키 적용)

▶ AR (Add Roundkey) 연산

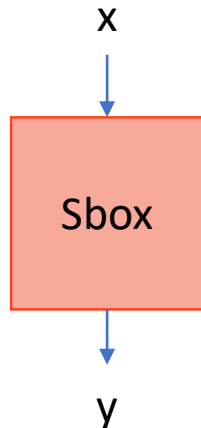
```
##-- Add Roundkey 함수  
def AR(in_state, rkey):  
    out_state = [0,0,0,0]  
    for i in range(len(in_state)):  
        out_state[i] = in_state[i] ^ rkey[i]  
  
    return out_state
```



TC20 – Sbox (비선형 변환)

▶ Sbox 구조

- ▶ AES와 동일한 Sbox 사용 (8비트 → 8비트)
- ▶ 각 라운드마다 4개의 Sbox를 각 바이트에 적용

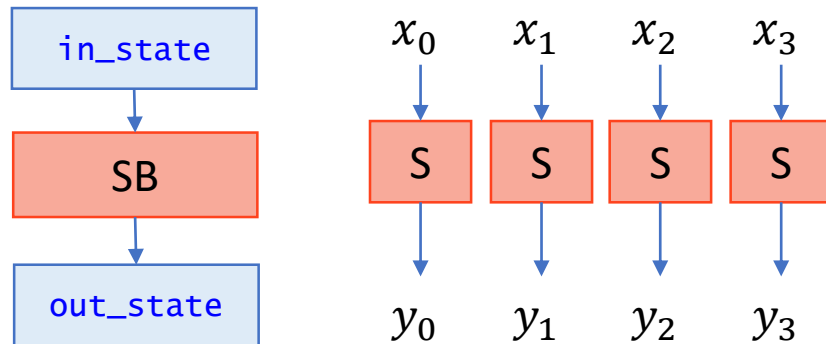


```
Sbox = [  
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,  
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,  
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,  
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,  
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,  
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,  
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,  
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,  
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,  
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,  
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,  
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,  
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,  
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,  
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,  
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16  
]
```

TC20 – Sbox (비선형 변환)

▶ SB (Sbox) 연산 – 바이트 단위의 비선형 변환

```
##-- Sbox layer 함수  
def SB(in_state):  
    out_state = [0,0,0,0]  
    for i in range(len(in_state)):  
        out_state[i] = Sbox[in_state[i]]  
  
    return out_state
```



TC20 – LM (선형변환)

▶ 선형함수 LM(Linear Map)

- ▶ 4바이트에 대한 바이트 단위의 선형변환
- ▶ 4x4 이진(binary)행렬 A 로 표현되는 함수: $Y = AX$

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 \oplus x_2 \oplus x_3 \\ x_0 \oplus x_2 \oplus x_3 \\ x_0 \oplus x_1 \oplus x_3 \\ x_0 \oplus x_1 \oplus x_2 \end{pmatrix}$$

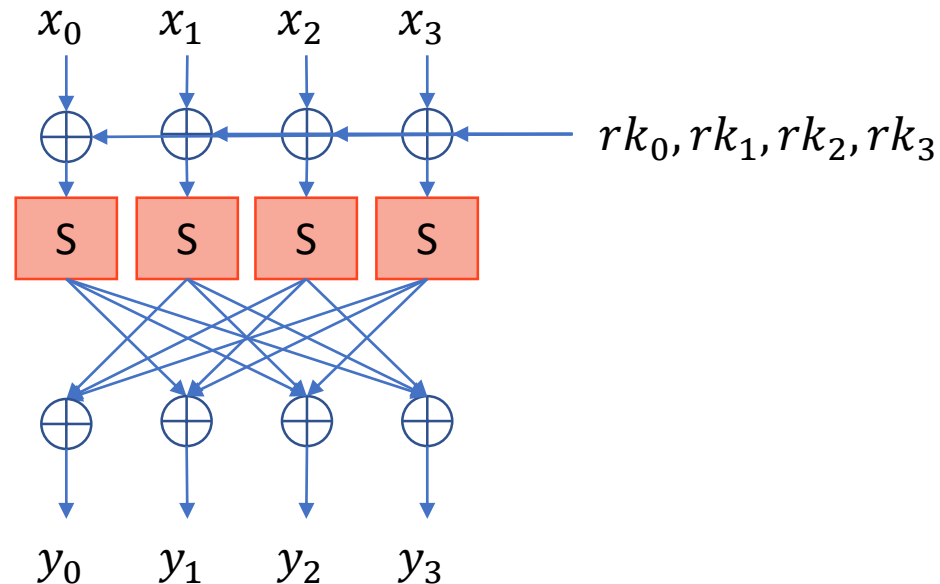
▶ 행렬의 특징

- ▶ Involution ($A^2 = I$) : $A^{-1} = A$
- ▶ 암호화, 복호화에 동일한 선형함수를 사용함

TC20 - 라운드 함수

▶ 라운드 구조

- ▶ AR: 라운드키 적용
- ▶ Sbox: 각 바이트에 비선형 Sbox 적용
- ▶ LM: 선형변환

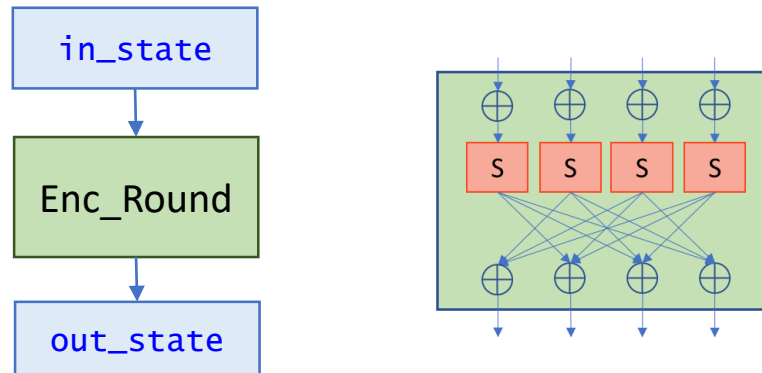


TC20 - 라운드 함수

▶ 라운드 함수

```
##-- 라운드 암호화 함수
def Enc_Round(in_state, rkey):
    out_state = [0,0,0,0]
    out_state = AR(in_state, rkey)
    in_state = SB(out_state)
    out_state = LM(in_state)

    return out_state
```



Toy Cipher – TC20

- ▶ TC20의 특징 – 극단적인 단순화
 - ▶ 작은 블록, 키 크기: 32비트
 - ▶ 키 스케줄 없음: 라운드키=암호키
 - ▶ 출력의 난수성: 통계적으로는 우수함 (10라운드)
 - ▶ SPN구조: 복호화를 위해 모든 단계의 역연산을 만들어야 함 (단, 선형함수 LM은 역함수가 자기자신과 같다)
- ▶ TC20 설계의 문제점
 - ▶ 32비트 암호키의 전수조사 공격(brute force attack) 가능
 - ▶ 마지막 AR이후 연산은 사실상 불필요함 (why?)

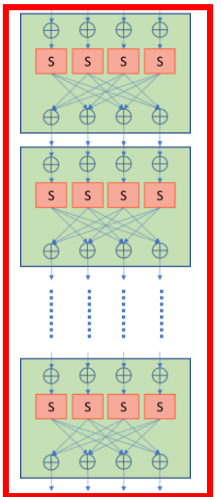
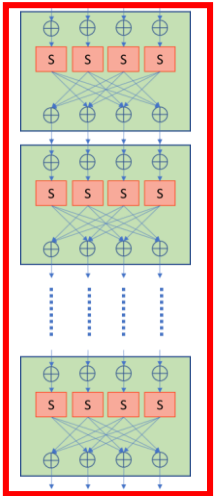
Cryptanalysis (암호분석)

Exhaustive Key Search
Meet In The Middle(MITM) Attack

2023.4

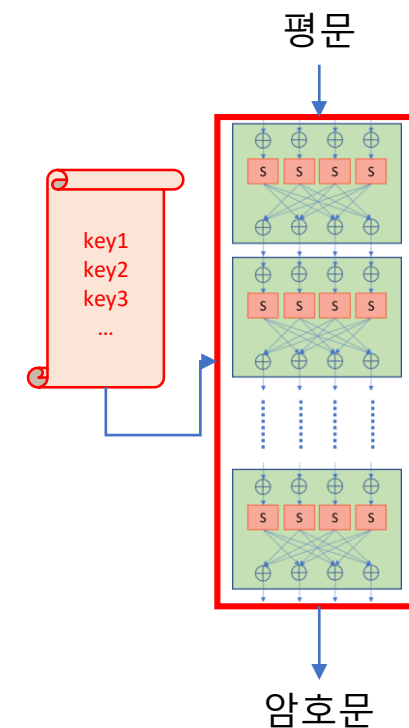
Contents

- ▶ Integer \leftrightarrow List
- ▶ Pickle dump (variable save/load)
- ▶ Brute force attack: Exhaustive key search
- ▶ Meet-in-the-Middle Attack



Brute Force Attack (1)

- ▶ 암호키 전수조사 (Exhaustive key search)
 - ▶ 모든 키를 시도해보는 방법으로 공격
 - ▶ 암호키의 크기가 작은 경우만 가능
- ▶ 예: TC20의 공격 (기지평문 공격)
 - ▶ TC20: 블록크기=키크기=32비트
 - ▶ 주어진 평문, 암호문으로부터 암호키를 찾는 공격
 - ▶ 공격 알고리즘
 - ▶ 2^{32} 가지 암호키 모두를 대입하여 평문을 암호화
 - ▶ 주어진 암호문과 같은 것이 나오면 올바른 암호키 후보로 선택
 - ▶ False alarm 확률: $1/2^{32}$ (잘못된 키가 암호키 후보가 될 확률)



Int2list() - 정수를 리스트로

0x12345678: 16진수표현

`print(0x12345678) → 305419896`
`print(hex(305419896)) → 0x12345678`

```
#--- int(4bytes) to list  
#--- Example: 0x12345678 -> [ 0x12, 0x34, 0x56, 0x78 ]
```

```
def int2list(n):  
    out_list = []  
    out_list.append( (n >> 24) & 0xff )  
    out_list.append( (n >> 16) & 0xff )  
    out_list.append( (n >> 8) & 0xff )  
    out_list.append( (n ) & 0xff )  
  
    return out_list
```

비트 쉬프트 (오른쪽으로 밀기)

`0x12345678 >> 8 = 0x00123456`
`0x12345678 >> 16 = 0x00001234`
`0x12345678 >> 24 = 0x00000012`

비트 마스크 (선택 비트만 추출)

`0x00123456 & 0xff = 56`
`0x00001234 & 0xff = 34`
`0x00000012 & 0xff = 12`

`0xff = 0x000000ff`
`& : 비트연산자 AND`

list2int() - 리스트를 정수로

```
#--- list to int  
#--- Example: [ 0x12, 0x34, 0x56, 0x78 ] -> 0x12345678
```

```
def list2int(l):  
    n = 0  
    num_bytes = len(l)  
    for idx in range(len(l)):  
        n += l[idx] << 8*(num_bytes - idx - 1)  
  
    return n
```

예제: 함수 입력 l = [0x12, 0x34, 0x56, 0x78]

num_bytes = len(l) = 4

정수로 만들기

```
n = 0  
n = 0x00000000 + (0x12 << 24) = 0x12000000  
n = 0x12000000 + (0x34 << 16) = 0x12340000  
n = 0x12340000 + (0x56 << 8) = 0x12345600  
n = 0x12345600 + (0x78 << 0) = 0x12345678
```

비트 쉬프트 (왼쪽으로 밀기)

Shift = 8*(num_bytes - idx - 1)

```
idx = 0 → Shift = 8*(4 - 0 - 1) = 24  
idx = 1 → Shift = 8*(4 - 1 - 1) = 16  
idx = 2 → Shift = 8*(4 - 2 - 1) = 8  
idx = 3 → Shift = 8*(4 - 3 - 1) = 0
```

평문-암호문 만들기

- ▶ 키 전수조사 공격을 시험하기 위한 예제 만들기
 - ▶ 평문(pt), 암호키(key)를 설정하고 암호문(ct)을 만든다.
 - ▶ 키 전수조사 공격 프로그램에는 평문(pt), 암호문(ct)만을 입력으로 하여, 암호키를 찾아내도록 한다.

```
import TC20_Enc_lib
```

```
given_pt = [0, 1, 2, 3]  
key = [0, 20, 20, 4]
```

암호는 Toy Cipher TC20을 사용한다.

```
ct = TC20_lib.TC20_Enc(given_pt, key)
```

```
print('pt  =', given_pt)  
print('key =', key)  
print('ct  =', ct)
```

(출력)

```
pt  = [0, 1, 2, 3]  
key = [0, 20, 20, 4]  
ct  = [192, 126, 66, 83]
```

Exhaustive Key Search

```
#key = [0, 20, 20, 4]
```

```
given_pt = [0, 1, 2, 3]  
given_ct = [192, 126, 66, 83]
```

```
Flag = False  
KeySize = 1<<24 # 24-bit key
```

Flag : for-loop 종료시 암호키를 찾았는지 확인하는 변수
KeySize : 키 전수조사 범위 설정용 ($1 \ll 24 = 2^{24}$)
(전체로 하면 너무 오래 걸림)

```
print('key Searching', end=' ')  
for idx in range(0, KeySize):  
    key_guess = int2list(idx)  
    pt = TC20_lib.TC20_Dec(given_ct, key_guess)  
    if pt == given_pt:  
        Flag = True  
        break
```

```
    if (idx % (1<<16)) == 0:  
        print('.', end='')
```

```
print('\n')  
if Flag:  
    print('key found!')  
    print('key =', key_guess)  
else:  
    print('key Not found!')
```

평문-암호문 쌍을 생성하는 암호키를
찾으면 for-loop를 나간다.

Exhaustive Key Search

▶ 생각해 볼 문제

- ▶ $ct = E(pt, key)$ 를 만족하는 다른 키가 우연히 나올 확률은?
- ▶ 키 후보가 나오면 모두 키 후보 리스트에 넣어 두는 방식을 구현하려면?
- ▶ 여러 개의 키 후보가 발견되는 경우 올바른 키를 얻기 위해서는 어떤 방법을 추가로 사용할까?

Double Encryption

▶ 블록암호 안전성 문제

- ▶ 암호키 전수조사가 가능하다면?

→ 암호키 크기를 늘이면 됨

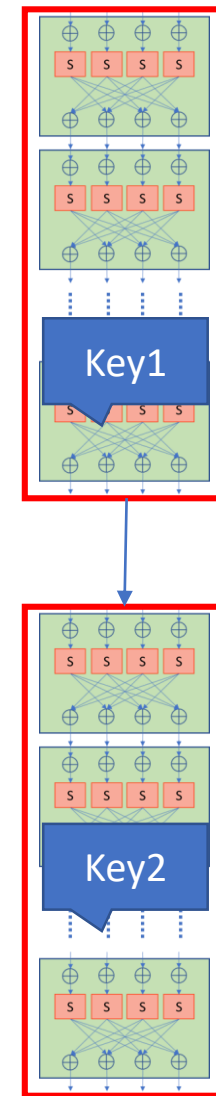
DES의 56비트 암호키가 작다면,
2배로 늘이면 된다.

56비트 전수조사에 1초가 걸린다면,
112비트 전수조사에는 23억년 걸린다.

▶ Double Encryption

- ▶ 암호 알고리즘: $C = E(P, \text{key})$
- ▶ 강화된 알고리즘: $C = E(E(P, \text{key1}), \text{key2})$
- ▶ 블록 크기는 그대로, 암호키 크기는 2배로 강화

→ 키 전수조사 공격에 안전할까?



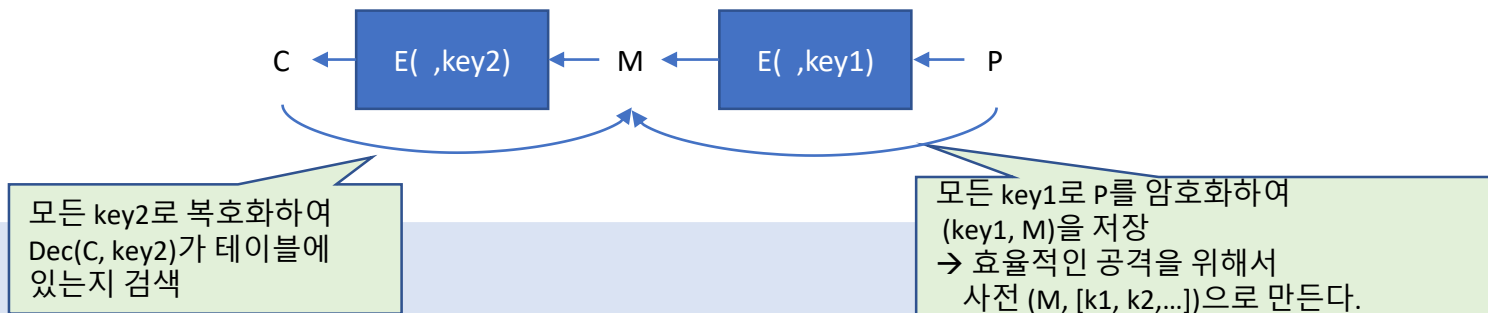
Brute Force Attack (2)

▶ Double Encryption의 안전성

- ▶ $C = E(P, \text{key})$ 키 크기: $2^k \rightarrow$ 공격량: 2^k 암호화 계산
- ▶ $C = E(E(P, \text{key1}), \text{key2})$ 키 크기: $2^{2k} \rightarrow$ 공격량: 2^{2k} 을 기대하지만
→ MITM attack을 이용하면 2^{k+1} 로 공격이 가능함
- ▶ 결론적으로 double encryption의 안전성은 강화되지 않음

▶ MITM(Meet-in-the-Middle) Attack 알고리즘

- ▶ 주어진 평문(P)와 암호문(C)에 대하여, 암호키 key1, key2를 찾는 공격
- ▶ 평문(P)를 가능한 모든 key1으로 암호화하여 테이블에 저장한다.
- ▶ 암호문 (C)를 가능한 모든 key2로 복호화하여 저장해둔 테이블에서 찾는다.



Double Encryption 구현

#- 암호키 (공격에서 찾아야 할 키)

```
key1 = [0, 20, 20, 4]  
key2 = [0, 1, 2, 3]
```

공격 프로그램에 찾아야 할
암호키: key1, key2

(예제 수행시간을 단축하기 위해
첫 바이트를 0으로 고정한다)

#- 평문1-암호문1

```
pt1 = [0, 1, 2, 3]  
mid1 = TC20_lib.TC20_Enc(pt1, key1)  
ct1 = TC20_lib.TC20_Enc(mid1, key2)
```

```
print('pt1  =', pt1)  
print('mid1 =', mid1)  
print('ct1  =', ct1)
```

#- 평문2-암호문2

```
pt2 = [4, 5, 6, 7]  
mid2 = TC20_lib.TC20_Enc(pt2, key1)  
ct2 = TC20_lib.TC20_Enc(mid2, key2)
```

```
print('pt2  =', pt2)  
print('mid2 =', mid2)  
print('ct2  =', ct2)
```

두 개의 (평문, 암호문) 쌍
(pt1, ct1), (pt2, ct2)
를 입력으로 암호키 key1, key2를
찾는 공격 예제를 만들어 본다.

(출력)

```
pt1  = [0, 1, 2, 3]  
mid1 = [192, 126, 66, 83]  
ct1  = [30, 18, 28, 114]
```

```
pt2  = [4, 5, 6, 7]  
mid2 = [63, 233, 23, 246]  
ct2  = [215, 173, 154, 71]
```

(키-암호문) 테이블 만들기

#- 주어진 평문을 모든 후보키로 암호한 결과를 파일로 저장한다.

```
def make_enc_table(pt):
    f = open('TC20EncTable.txt', 'w+')

    print('Encryption Table File', end=' ')
    # 전수조사 범위를 24비트 한정한다 key=[0,*, *, *]
    N = 1<<24
```

```
for idx in range(0, N):
    key = int2list(idx)
    mid = TC20_lib.TC20_Enc(pt, key)
    int_key = list2int(key) # = idx
    int_mid = list2int(mid)
```

```
str = format('%10d %10d \n' %(int_key, int_mid))
f.write(str)
```

```
if (idx % (1<<18)) == 0:
    print('.', end='')
```

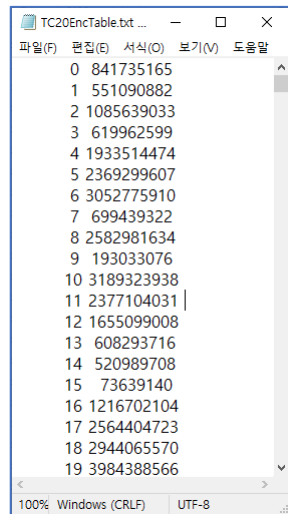
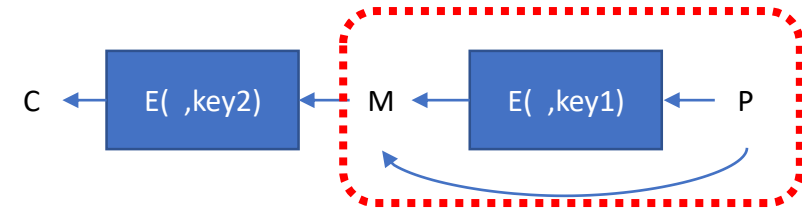
```
f.close()
```

```
pt1 = [0, 1, 2, 3]
make_enc_table(pt1)
(출력)
Encryption Table File
.....
```

(주의) 이 함수는
많은 시간이 소요된다!

'키, 암호문' 쌍을
각각 정수로 파일에 저장한다.

수행 단계를 '.'으로 표시하며
모두 64개가 찍히면 끝난다.



테이블 파일 → 사전 만들기

#- 파일에서 테이블을 읽어 사전으로 만든다.

```
def make_dic_from_enc_table(table_file):  
    MaxItems = 1<<24 # 파일의 일부만 읽을때 필요 (최대 2^24까지만)  
    dic = {} # (mid, [k1,k2,,])  
    f = open(table_file, 'r')  
    cnt = 0  
    print('Read Encryption Table', end=' ')  
    while True:  
        line = f.readline().split()  
        if not line:  
            break  
        if cnt > MaxItems :  
            break  
        cnt += 1
```

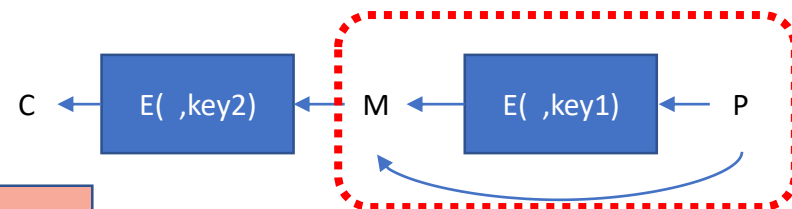
```
        key1_guess, mid = int(line[0]), int(line[1])  
        if mid in dic :  
            dic[mid].append(key1_guess)  
        else:  
            dic[mid] = [key1_guess]
```

```
        if (cnt % (1<<19)) == 0:  
            print('.', end='')  
    f.close()  
    print('\n')
```

```
    return dic
```

(주의) 고정된 pt에 대하여
mid = E(pt, key) 를 만들면
하나의 mid에 여러 개의 key가 대응될 수
있다.

처음 나오는 mid 값이면
리스트를 만들고,
아니면 기존 리스트에
추가(append)한다.



변수를 파일에 저장하기

▶ pickle: 변수를 파일로 저장

- ▶ 만드는데 오래 걸리는 변수는 파일에 저장해두고 재사용할 수 있다.
- ▶ 이 과정을 피클링이라고 한다.

```
import pickle

#--- 변수를 파일에 저장하기
def save_var_to_file(var, filename):
    f = open(filename, 'w+b')
    pickle.dump(var, f)
    f.close()

#--- 파일에서 변수를 가져오기
def load_var_from_file(filename):
    f = open(filename, 'rb')
    var = pickle.load(f)
    f.close()
    return var
```

```
#--- 사전을 파일로 저장(dump)하기
#save_var_to_file(mid_dic, 'TC20Enc_Dic.p')

#-- 파일에서 사전 가져오기
mid_dic2 = load_var_from_file('TC20Enc_Dic.p')
```

(암호문-키) 사전 만들기

```
def make_enc_dic(pt):  
    dic = {}  
    print('Making Encryption Dictionary', end=' ')  
    # 전수조사 범위를 24비트 한정한다 key=[0,*, *, *]  
    N = 1<<24
```

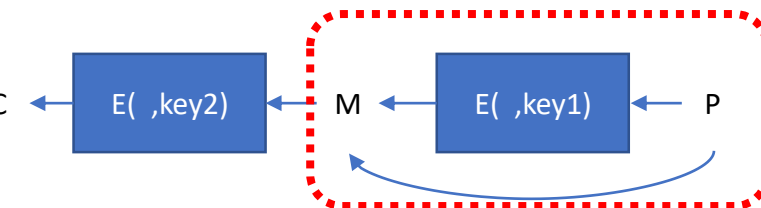
```
    for idx in range(0, N):  
        key = int2list(idx)  
        mid = TC20_lib.TC20_Enc(pt, key)  
        int_key = list2int(key)  
        int_mid = list2int(mid)
```

```
        if int_mid in dic :  
            dic[int_mid].append(int_key)  
        else:  
            dic[int_mid] = [int_key]
```

```
    if (idx % (1<<18)) == 0:  
        print('.', end='')  
    return dic
```

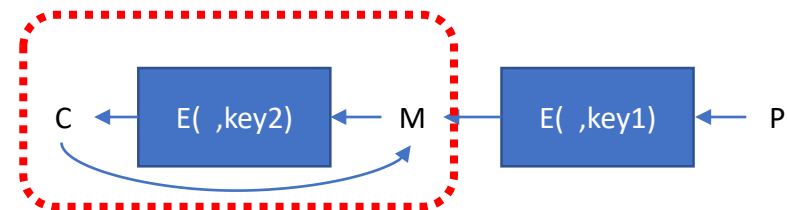
```
pt1 = [0, 1, 2, 3]  
mid_dic = make_enc_dic(pt1)  
(출력)  
Making Encryption Dictionary  
.....
```

(주의) 이 함수는
많은 시간이 소요된다!



사전에
(key, value) = (암호문,
[키리스트])
형식으로 저장한다.

MITM-Attack



▶ MITM(Meet-In-The-Middle) Attack

- ▶ 미리 만든 사전을 이용하여 key2의 가능한 모든 값을 대입한다.
- ▶ key2로 암호문(ct1)을 복호한 결과를 사전에서 찾는다.
- ▶ 사전에 있는 후보키가 새로운 (평문, 암호문)쌍 (pt2, ct2)에 대하여 올바른 결과를 주면 암호 키를 찾은 것으로 간주한다.

```
print('Meet in the middle attack', end='')
N = 1<<24 # key2 크기
for idx in range(0, N):
    key2 = int2list(idx)
    mid2 = TC20_lib.TC20_Dec(ct1, key2)
    int_mid2 = list2int(mid2)
```

```
    if int_mid2 in mid_dic:
        list_key_candidate = mid_dic[int_mid2]
        if len(list_key_candidate) > 0:
            verify_key_candidate_list(list_key_candidate, key2, pt2, ct2)
```

```
    if (idx % (1<<18)) == 0:
        print('.', end='')

```

```
print('\n key search completed!')
```

mid_dic에 (mid, [key list])
가 저장되어 있으며,
키 리스트의 길이는 대부분
1로 예상된다. (왜?)

(출력)

Meet in the middle attack.

key1 = [0, 20, 20, 4] key2 = [0, 1, 2, 3]

.....
key search completed!

키 후보 리스트

▶ 키 후보 리스트 탐색

- ▶ 키(key1) 후보를 **정수로** 저장한 리스트에 올바른 키가 있는지 두번째 평문 암호문쌍으로 확인한다.

#- 암호키 (공격에서 찾아야 할 키)

```
def verify_key_candidate_list(key1_list, key2, pt2, ct2):  
    flag = False  
    for key1 in key1_list:  
        key1_state = int2list(key1)
```

```
        mid1 = TC20_lib.TC20_Enc(pt2, key1_state)  
        ct_comp = TC20_lib.TC20_Enc(mid1, key2)
```

```
        if ct_comp == ct2:  
            print('\n key1 =', key1_state, ' key2 =', key2)  
            flag = True
```

```
    return flag
```

(pt2 → ct2)로 암호화하는 경우
올바른 키로 간주하고 출력

```
candidate = [66051, 504503410, 67438087, 3618478663, 1315844, 66051]  
verify_key_candidate_list(candidate, key2, pt2, ct2)
```

(출력)

```
key1 = [0, 20, 20, 4] key2 = [0, 1, 2, 3]
```

Triple DES and Double Enc.

- ▶ 1990년대 DES의 암호키(56비트)가 충분한 안전성을 주지 못하게 되어 새로운 대안을 모색
- ▶ Double DES(암호키 2개로 두번 암호화)는 안전성을 57비트 밖에 주지 못하게 되어 Triple DES를 대안으로 채택함
 - ▶ 두 개의 키를 $key1 - key2 - key1$ 의 순서로 사용하거나
 - ▶ 3개의 키를 사용하는 방식이 채택됨
- ▶ 현재는 모두 AES로 대체되고, 이전 자료의 복호화에만 권장됨

Cryptanalysis (암호분석)

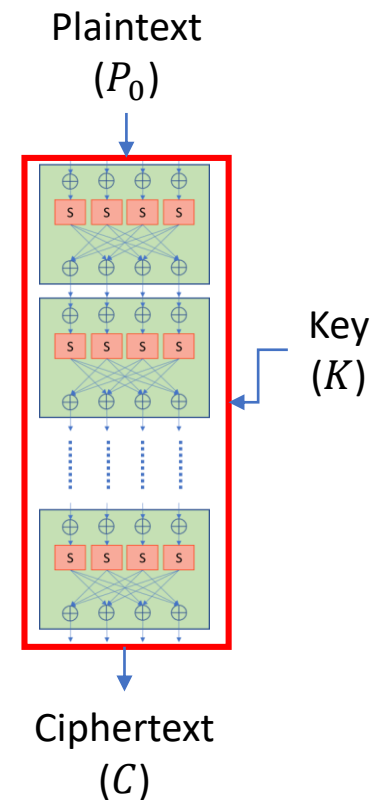
Time Memory Trade Off (TMTO) Attack

2023.4

Hellman, A Cryptanalytic Time-Memory Trade-Off, IEEE Trans. Information Theory (1980)

키 전수조사 공격 유형

- ▶ 공격 조건: 선택평문공격(Chosen Plaintext Attack)
 - ▶ 선택평문(공격자의 선택): P_0
 - ▶ 암호문: $C = E(P_0, K)$
 - ➔ 사용된 암호키 (K)를 찾는 공격 ($K \in \{0,1\}^n$)
- ▶ 계산 능력에 의존하는 경우(Exhaustive key search)
 - ▶ 공격방법: 가능한 2^n 가지의 모든 키를 다 조사해본다.
 - ▶ 공격에 사용한 자원: 2^n 계산량
- ▶ 저장 능력에 의존하는 경우(Pre-computation)
 - ▶ 공격방법: 모든 키에 대하여 $(K, C) = (K, E(P_0, K))$ 를 테이블에 저장한 후, 공격대상 암호문(C)을 테이블에서 찾는다.
 - ▶ 공격에 사용한 자원: 2^n 메모리



Trade-Off

- 테이블의 크기는 사전계산량보다 작을 수 있다.
- Table을 만드는 사전계산(Pre-computation) 시간은 공격량에 포함하지 않는다.

▶ 두 공격방법의 비교

	Time(T)	Memory(M)	Attack Complexity
1. Exhaustive Key Search	2^n	1	2^n
2. Table Size	1	2^n	2^n
3. TMTO	▲	■	▲ + ■

$< 2^n$

- ▶ 암호 키 공간 $\{0,1\}^n$ 이 충분히 크면, 계산시간, 또는 메모리 모두 2^n 능력을 갖기는 어렵다.
- ▶ 계산시간, 메모리 자원의 적절한 Trade-Off로 실현 가능한 공격기법이 있을까?
- ▶ 56비트 키를 사용하는 DES의 경우,
▲ = ■ = 2^{40} 의 공격법이 있다면, 2^{41} 의 공격량으로 해독됨

Encryption Chain 구성

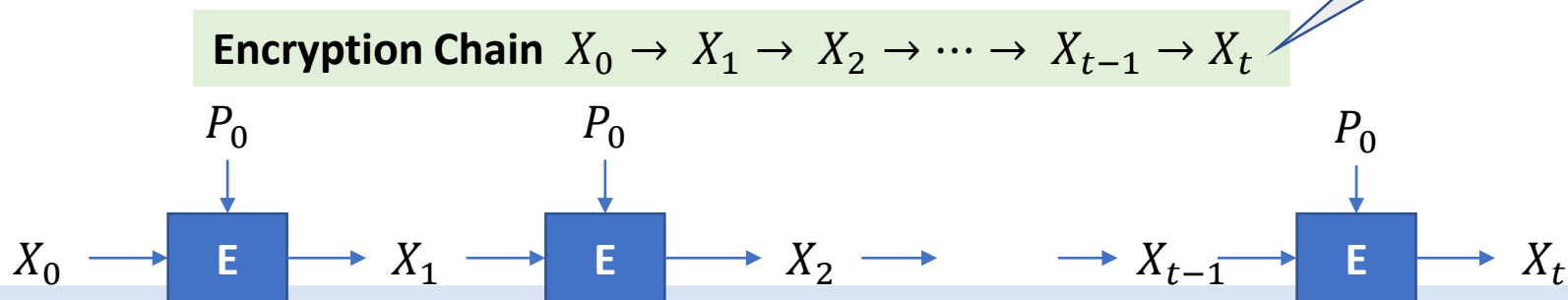
▶ 암호 키 체인의 구성

- ▶ 고정된 선택평문 (P_0) 에 대하여
- ▶ 시작점(X_0): 랜덤하게 선택한 암호 키 $X_0 = K_0$
- ▶ 체인의 계산 규칙: $X_{i+1} = f(X_i)$

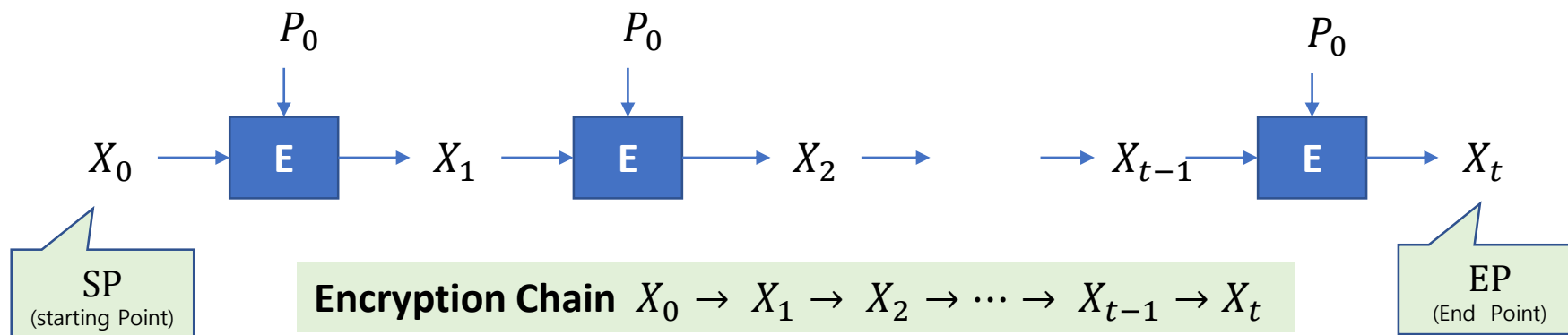
$$f(X_i) = R(E(P_0, X_i))$$

- ▶ $E(P_0, X_i)$: 암호 키 X_i 로 평문 P_0 를 암호화한 결과(암호문)
- ▶ $X_{i+1} = R(C_i)$: 암호문을 암호 키로 변환하는 랜덤 함수

(랜덤한?)
암호키들의
체인



Encryption Chain의 활용(1)



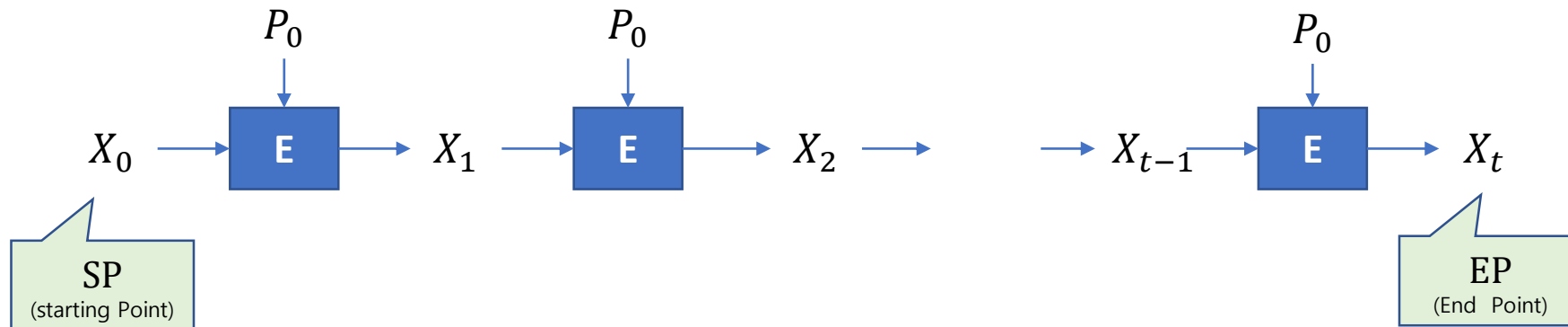
▶ 체인의 활용 (공격)

$$C = E(P_0, K)$$

- ▶ 암호문(C) 생성에 사용된 암호 키(K)를 찾는 공격
- ▶ 암호문(C)에 대하여, $R(C) = X_k$ 를 만족한다면,
 $R(\mathbf{C}) = X_k = R(\mathbf{E(P_0, X_{k-1})})$
- ▶ 암호 키를 X_{k-1} 로 판정한다.

암호문과 암호키의 크기가 같다면, $R()$ 을 사용하지 않아도 된다.

Encryption Chain의 활용(2)



- ▶ 체인의 시작점(SP)과 끝점(EP)만 저장한다면
 - ▶ 암호문(C)에 대하여, $R(C) = X_k$ 를 만족하는 k 를 찾기 위해
$$R(\textcolor{red}{C}) = X_k = f(X_{k-1}) = R(\textcolor{red}{E}(P_0, X_{k-1}))$$
 - ▶ 체인 길이(t) 만큼의 시도가 필요하다. ($1 \leq k \leq t$)

Encryption Chain $X_0 \rightarrow X_1 \rightarrow X_2 \rightarrow \cdots \textcolor{red}{X}_{k-1} \rightarrow \textcolor{red}{X}_k \rightarrow \cdots \rightarrow X_{t-1} \rightarrow X_t$

Hellman 테이블 구성

$$\begin{aligned} \text{SP}_1 &= X_{1,0} \rightarrow X_{1,1} \rightarrow X_{1,2} \rightarrow \cdots \rightarrow X_{1,t-1} \rightarrow X_{1,t} = \text{EP}_1 \\ \text{SP}_2 &= X_{2,0} \rightarrow X_{2,1} \rightarrow X_{2,2} \rightarrow \cdots \rightarrow X_{2,t-1} \rightarrow X_{2,t} = \text{EP}_2 \\ \text{SP}_i &= X_{i,0} \rightarrow X_{i,1} \rightarrow X_{i,2} \rightarrow \cdots \rightarrow X_{i,t-1} \rightarrow X_{i,t} = \text{EP}_i \\ \text{SP}_m &= X_{m,0} \rightarrow X_{m,1} \rightarrow X_{m,2} \rightarrow \cdots \rightarrow X_{m,t-1} \rightarrow X_{m,t} = \text{EP}_m \end{aligned}$$

계산량: $m \times t$
메모리: m

▶ 랜덤하게 m 개의 시작점을 선택한다. $\{\text{SP}_1, \text{SP}_2, \dots, \text{SP}_m\}$

▶ 각 시작점으로 체인을 만든다.

$$\text{SP}_i = X_{i,0} \rightarrow X_{i,1} \rightarrow X_{i,2} \rightarrow \cdots \rightarrow X_{i,t-1} \rightarrow X_{i,t} = \text{EP}_i$$

▶ 시작점과 끝점만 저장한다.

$$\{(\text{SP}_1, \text{EP}_1), (\text{SP}_2, \text{EP}_2), \dots, (\text{SP}_m, \text{EP}_m)\}$$

▶ 끝점을 기준으로 정렬(sort)한다.

Hellman 테이블과 키 탐색(1)

$$\begin{array}{l} \text{SP}_1 = X_{1,0} \rightarrow X_{1,1} \rightarrow X_{1,2} \rightarrow \cdots \rightarrow X_{1,t-1} \rightarrow X_{1,t} = \text{EP}_1 \\ \text{SP}_2 = X_{2,0} \rightarrow X_{2,1} \rightarrow X_{2,2} \rightarrow \cdots \rightarrow X_{2,t-1} \rightarrow X_{2,t} = \text{EP}_2 \\ \text{SP}_i = X_{i,0} \rightarrow X_{i,1} \rightarrow X_{i,2} \rightarrow \cdots \rightarrow X_{i,t-1} \rightarrow \mathbf{X_{i,t}} = \text{EP}_i \\ \text{SP}_m = X_{m,0} \rightarrow X_{m,1} \rightarrow X_{m,2} \rightarrow \cdots \rightarrow X_{m,t-1} \rightarrow X_{m,t} = \text{EP}_m \end{array}$$

▶ 암호 키 탐색

- ▶ 암호문(C)이 $R(C) = \text{EP}_i$ 를 만족한다면,
$$R(\mathbf{C}) = X_t = R(\mathbf{E(P_0, X_{t-1})})$$
- ▶ 암호 키를 X_{t-1} 로 판정한다.

Hellman 테이블과 키 탐색(2)

SP_1	$= X_{1,0} \rightarrow X_{1,1} \rightarrow \dots$	$X_{1,s} \rightarrow X_{1,s+1} \rightarrow \dots \rightarrow X_{1,t-1} \rightarrow X_{1,t} = EP_1$
SP_2	$= X_{2,0} \rightarrow X_{2,1} \rightarrow \dots$	$X_{2,s} \rightarrow X_{2,s+1} \rightarrow \dots \rightarrow X_{2,t-1} \rightarrow X_{2,t} = EP_2$
SP_i	$= X_{i,0} \rightarrow X_{i,1} \rightarrow \dots$	$\mathbf{X_{i,s}} \rightarrow \mathbf{X_{i,s+1}} \rightarrow \dots \rightarrow \mathbf{X_{i,t-1}} \rightarrow \mathbf{X_{i,t}} = EP_i$
SP_m	$= X_{m,0} \rightarrow X_{m,1} \rightarrow \dots$	$X_{m,s} \rightarrow X_{m,s+1} \rightarrow \dots \rightarrow X_{m,t-1} \rightarrow X_{m,t} = EP_m$

▶ 암호 키 탐색

- ▶ 암호문(C)이 $R(C) = EP_i$ 를 만족한다면, 암호 키를 $X_{i,t-1}$ 로 판정한다.

$$R(\mathbf{C}) = X_{i,t} = f(X_{i,t-1}) = R\left(\mathbf{E(P_0, X_{i,t-1})}\right)$$

- ▶ 같은 원리로 암호문이 다음을 만족하면 암호키를 $X_{i,s-1}$ 로 판정한다.

$$f^{t-s}(R(C)) = EP_i$$

Hellman 테이블과 키 탐색(3)

$$SP_1 = X_{1,0} \rightarrow X_{1,1} \rightarrow X_{1,2} \rightarrow \cdots \rightarrow X_{1,t-1} \rightarrow X_{1,t} = EP_1$$

$$SP_2 = X_{2,0} \rightarrow X_{2,1} \rightarrow X_{2,2} \rightarrow \cdots \rightarrow X_{2,t-1} \rightarrow X_{2,t} = EP_2$$

$$SP_i = X_{i,0} \rightarrow X_{i,1} \rightarrow X_{i,2} \rightarrow \cdots \rightarrow X_{i,t-1} \rightarrow X_{i,t} = EP_i$$

$$SP_m = X_{m,0} \rightarrow X_{m,1} \rightarrow X_{m,2} \rightarrow \cdots \rightarrow X_{m,t-1} \rightarrow X_{m,t} = EP_m$$

$m \times t$ 행렬
(각 원소에
하나의 키
후보가 대응됨)

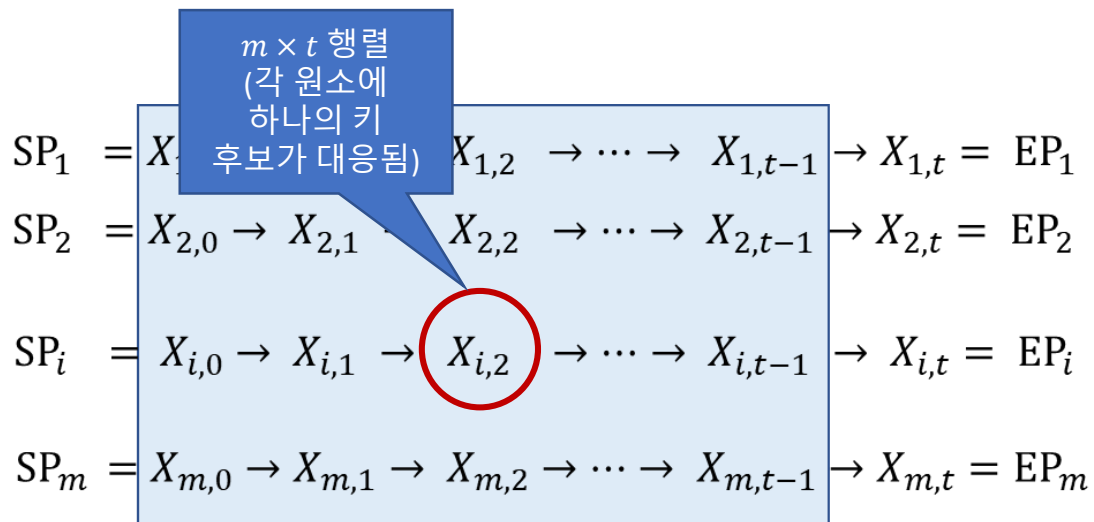
▶ (매우 낙관적인) 공격 방법

- ▶ 암호문(C)에 대하여 $R(C) = EP_i$ 를 만족하는 끝점이 있는지 찾는다.
- ▶ 없으면 $f(R(C)) = EP_i$ 를 만족하는 끝점을 찾는다.
- ▶ 반복하여 $f^2(R(C)), f^3(R(C)), \dots$ 중 EP_i 와 같은 것이 있는지 계속한다.

➔ 각 점은 $R(E(P_0, X_{i,j}))$ 암호키 $X_{i,j}$ 로 고정된 평문 P_0 를 암호화한 것으로,
키 공간 만큼 큰 테이블이 모든 암호키를 포함하고 있으면 공격이 가능하다.

키 탐색 공격의 문제점(1)

- ▶ 탐색공간 문제: 테이블의 포함된 암호키가 충분한가?
 - ▶ 체인 구성에 사용된 함수 f 가 랜덤하다고 가정하자.
 - ▶ 테이블의 크기 ($m \times t$)를 어떻게 하는 것이 적절한가?

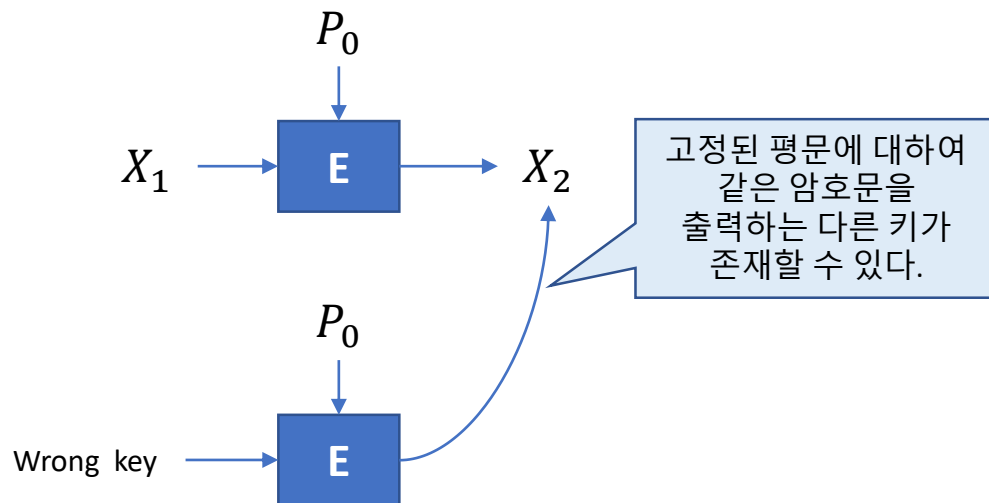


- ▶ 테이블의 모든 값이 다르다면 각각 다른 키를 추천하므로,
 $mt = 2^n$ 이면 항상 키 복구가 가능할 것으로 기대됨

→ 확률계산을 통해 분석해보면, 불가능한 시나리오 임 !!!

키 탐색 공격의 문제점(2)

- ▶ False Alarm 문제: 잘못된 키가 추천될 가능성과 이를 걸러내는 방법은?
 - ▶ 판별식 $f^{t-s}(R(C)) = EP_i$ 에서 다음 식을 우연히 만족하는 $X_{i,s-1}$ 가 추천되었을 가능성은?
$$R(\mathbf{C}) = X_{i,s} = f(X_{i,s-1}) = R\left(E(P_0, X_{i,s-1})\right)$$



→ 다른 평문에 대한 암호문을 비교하는 방법으로 wrong key를 걸러내면 된다

테이블 분석

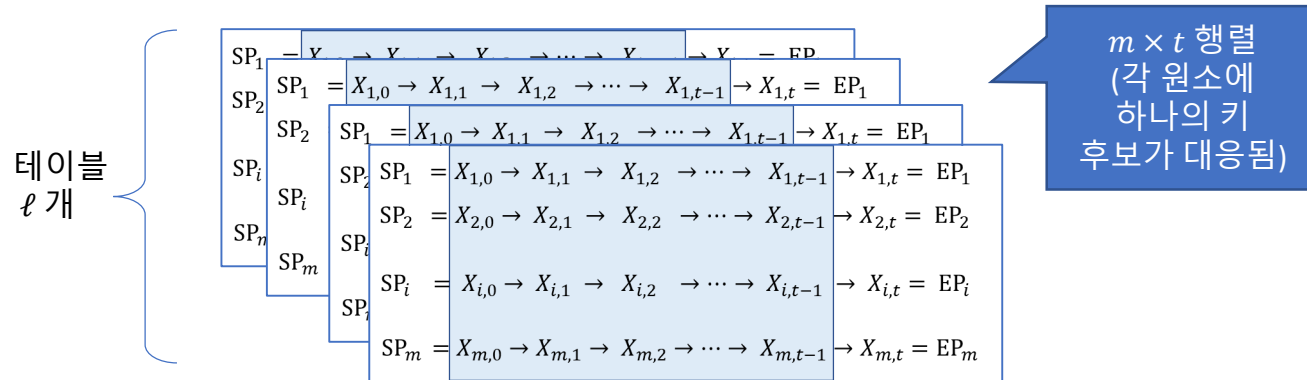
▶ ECR: Expected Coverage Rate

- ▶ 암호 알고리즘이 랜덤하다고 가정하자.
- ▶ 테이블 구성에 사용된 $X_{i,j}$ 가 모두 랜덤하게 선택된다고 생각할 수 있다.
- ▶ $ECR(N, m, t) = \frac{\bar{H}}{mt}$, $\bar{H} = \{X_{i,j} \mid 1 \leq i \leq m, 0 \leq j < t\}$

$$\begin{aligned} SP_1 &= X_{1,0} \rightarrow X_{1,1} \rightarrow X_{1,2} \rightarrow \cdots \rightarrow X_{1,t-1} \rightarrow X_{1,t} = EP_1 \\ SP_2 &= X_{2,0} \rightarrow X_{2,1} \rightarrow X_{2,2} \rightarrow \cdots \rightarrow X_{2,t-1} \rightarrow X_{2,t} = EP_2 \\ SP_i &= X_{i,0} \rightarrow X_{i,1} \rightarrow X_{i,2} \rightarrow \cdots \rightarrow X_{i,t-1} \rightarrow X_{i,t} = EP_i \\ SP_m &= X_{m,0} \rightarrow X_{m,1} \rightarrow X_{m,2} \rightarrow \cdots \rightarrow X_{m,t-1} \rightarrow X_{m,t} = EP_m \end{aligned}$$

$\bar{H} = mt$ 라면 $ECR = 1$.
(모두 다른 값)
하지만, 실제로 ECR은
경우에 따라 다르며
크지 않다.

TMTO 공격 성공 확률



- ▶ Hellman 테이블 ℓ 개를 이용한 TMTO 공격의 성공 확률
(암호 키 공간: $N = 2^n$)

$$P(S) = 1 - \left(1 - \text{ECR} \frac{mt}{N}\right)^\ell \approx 1 - \exp\left(-\frac{\ell mt \text{ ECR}}{N}\right)$$

하나의 테이블에서
암호 키가 발견될 확률

모든 테이블에서
암호 키가 발견되지
않을 확률