

# به نام خداوند جان و خرد

## پروژه تحویلی اول درس مبانی امنیت اطلاعات

امیرفاضل کوزه گر کالجی

9931099

## فاز اول

بخش اول:

در این بخش برنامه ای توسعه داده شد تا میزان قدرت یک رمز را بسنجد. معیار های ارزیابی رمز در این برنامه به شکل زیر است:

- طول رمز
- وجود کاراکتر های lowercase
- وجود کاراکتر های uppercase
- وجود کاراکتر های خاص
- وجود رقم

ارضا شدن هر یک از 5 معیار نام برده یک امتیاز مثبت برای رمز ما خواهد داشت. به همین سبب، در نهایت بر اساس هر امتیازی که رمز کسب کرده است، عنوانی برای نشان دادن میزان قدرت آن نشان می دهیم.

```
def strength_score(password):  
    score = 0  
    warning_notes = []  
  
    if len(password) >= 8:  
        score += 1  
    else:  
        warning_notes.append("choose a password with length of 8 or more")  
    if any(char.isupper() for char in password):  
        score += 1  
    else:  
        warning_notes.append("better use at least an uppercase char in your password")  
    if any(char.islower() for char in password):  
        score += 1  
    else:  
        warning_notes.append("better use at least a lowercase char in your password")  
    if any(char.isdigit() for char in password):  
        score += 1  
    else:  
        warning_notes.append("better use at least a digit in your password")  
    if any(char in r'!@#$%^&*()-_+=[]{}|/;<.>?/~\`' for char in password):  
        score += 1  
    else:  
        warning_notes.append("better use at least a special character in your password")  
  
    return score, warning_notes
```

```
class Password_Strength:
    UNACCEPTABLE = 'unacceptable'
    WEAK = "weak"
    MODERATE = "moderate"
    ACCEPTABLE = 'acceptable'
    STRONG = 'strong'
```

در شکل زیر نتیجه برنامه را برای برخی پسورد های امتحانی نشان می‌دهیم:

```
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase1\part1> python .\password_qualify.py -pass amir
your password is: Weak

here are few advices to make stronger passwords:
>choose a password with length of 8 or more
>better use at least an uppercase char in your password
>better use at least a digit in your password
>better use at least a special character in your password
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase1\part1> python .\password_qualify.py -pass A@mir_23
your password is: Strong

PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase1\part1> python .\password_qualify.py -pass Ali
your password is: Moderate

here are few advices to make stronger passwords:
>choose a password with length of 8 or more
>better use at least a digit in your password
>better use at least a special character in your password
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase1\part1> █
```

همچنین رمز هایی که استفاده می‌شوند همراه با سطح خود، در یک فایل JSON ذخیره می‌شوند زیرا که اگر بعداً همین رمز وارد شد سریعتر به مقدار آن دسترسی داشته باشیم و از محاسبات اضافی بپرهیزیم:

```
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase1\part1> python .\password_qualify.py -pass P@ssword
your password is: Acceptable

here are few advices to make stronger passwords:
>better use at least a digit in your password
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase1\part1> python .\password_qualify.py -pass P@ssword
password already checked, The strength level is: Acceptable
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase1\part1> █
```

بخش دوم:

در این بخش، طبق گفته های دستورکار یک ابزار برای حدس رمز طراحی و توسعه داده شده است که دارای مود ها و فضای جست و جو های متفاوتی می باشد.

برای توسعه این برنامه از کتابخانه `itertools` استفاده شده است تا برای ما لیستی از تمام ترکیبات ممکن را فراهم آورد.

رفتار برنامه بر اساس مود های ذکر شده، به شکل زیر تعیین می شود:

```
def guess_password(password_value, mode_value, search_space_value, additional_args_value):
    password_len = len(password_value)
    passwords = []

    start_time = time.time()

    if mode_value == 1:
        passwords = brute_force_generator(password_len, search_space_value)
    elif mode_value == 2:
        first_char = password_value[0]
        passwords = brute_force_generator(password_len - 1, search_space_value)
        passwords = [first_char + p for p in passwords]
    elif mode_value == 3:
        first_k_chars = password_value[:additional_args_value]
        new_len = password_len - additional_args_value
        passwords = brute_force_generator(new_len, search_space_value)
        passwords = [first_k_chars + p for p in passwords]
    else:
        print("Unknown mode")

    result = find_match(password_value, passwords)

    if result:
        end_time = time.time()
        interval = end_time - start_time
        return result, interval
    else:
        return None
```

رفتار برنامه بر اساس محیط های جست و جو ذکر شده نیز، به شکل زیر تعیین می شود:

```
def brute_force_generator(length, search_space):
    iterable = []

    if search_space == 'NO':
        iterable = string.digits
    elif search_space == 'NL':
        iterable = string.digits + string.ascii_lowercase
    elif search_space == 'LO':
        iterable = string.ascii_lowercase
    elif search_space == 'NLU':
        iterable = string.digits + string.ascii_uppercase + string.ascii_lowercase

    return [''.join(p) for p in itertools.product(iterable, repeat=length)]
```

در این میان، هنگام اجرای برنامه و جست و جو به دنبال رمز داده شده، میزان تلاش ها و زمان سپری شده نیز ضبط و در انتها اعلام می شود:

```
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase1\part2> python.exe .\pass_guess.py -p home -m 1 -s LO
It took 132813 attempts and 0.07668066024780273 seconds to crack the desired password
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase1\part2> python.exe .\pass_guess.py -p home -m 1 -s NL
It took 825063 attempts and 0.29393982887268066 seconds to crack the desired password
```

```
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase1\part2> python.exe .\pass_guess.py -p alli432 -m 3 -s NL -k 4
It took 5295 attempts and 0.010010004043579102 seconds to crack the desired password
```

```
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase1\part2> python.exe .\pass_guess.py -p 12we -m 2 -s NL
It took 3759 attempts and 0.035068511962890625 seconds to crack the desired password
```

تصاویر فوق، نمونه هایی از اجرای برنامه مدنظر می باشند.

## فاز دوم

در این فاز از ما خواسته شده تا یک برنامه بنویسیم که محتوای فایل ها را رمزگذاری و رمزنگاری کند.

این برنامه با زبان پایتون و با در نظر گرفتن text بودن فایل های دریافتی توسعه داده شده است.

در این برنامه یه روش رمزنگاری AES، DES، و RSA پیاده سازی شده است. که غیر از RSA باقی الگوریتم ها متقارن محسوب می شوند.

تمامی الگوریتم ها از توابع یکسانی برخوردارند پس همه آنها از یک کلاس ابستراکت ارث بری می کنند:

```
from abc import ABC, abstractmethod
import os

class EncryptionAlgorithm(ABC):
    @abstractmethod
    def generate_key(self):
        pass

    @abstractmethod
    def encrypt(self, data, key):
        pass

    @abstractmethod
    def decrypt(self, encrypted_data, key):
        pass

    @abstractmethod
    def pad(self, data):
        pass

    @abstractmethod
    def unpad(self, data):
        pass
```

## الگوریتم ها

### AES

در پیاده سازی این الگوریتم از کتابخانه Crypto استفاده شده است تا به صورت واضح تمامی قابلیت های الگوریتم AES را در اختیارمان قرار دهد.

```
from Crypto.Cipher import AES as AES_ALGORITHM
from Crypto.Random import get_random_bytes
from algorithms.EncryptionAlgorithm import EncryptionAlgorithm

class AES(EncryptionAlgorithm):

    def __init__(self):
        self.nonce = 0
        self.iv = 0

    def generate_key(self):
        return get_random_bytes(AES_ALGORITHM.block_size)

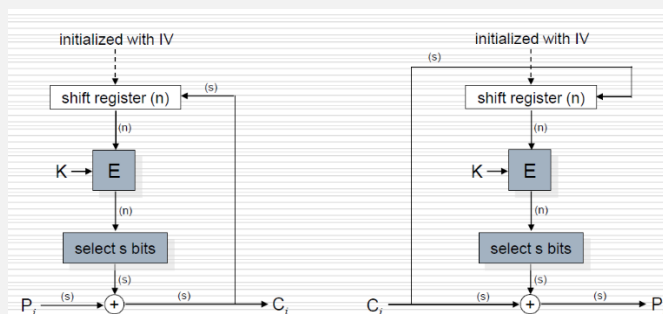
    def encrypt(self, data, key):
        cipher = AES_ALGORITHM.new(key, AES_ALGORITHM.MODE_CFB)
        ciphertext = cipher.encrypt(data.encode('utf8'))
        self.iv = cipher.iv
        return ciphertext

    def decrypt(self, encrypted_data, key):
        cipher_dec = AES_ALGORITHM.new(key, AES_ALGORITHM.MODE_CFB, self.iv)
        plaintext = cipher_dec.decrypt(encrypted_data)
        return plaintext.decode()

    def pad(self, data):
        pass

    def unpad(self, data):
        pass
```

طبق قطعه کد فوق، می توان دریافت که این الگوریتم از مد CFB یا Cipher feedback استفاده می کند. اساس کار CFB به شکل زیر می باشد:



همانند AES، در پیاده سازی این الگوریتم نیز از کتابخانه Crypto استفاده شده است.

```
from Crypto.Cipher import DES as DES_ALGORITHM
from Crypto.Random import get_random_bytes
from algorithms.EncryptionAlgorithm import EncryptionAlgorithm

class DES(EncryptionAlgorithm):

    def generate_key(self):
        return get_random_bytes(DES_ALGORITHM.block_size)

    def encrypt(self, data, key):
        cipher = DES_ALGORITHM.new(key, DES_ALGORITHM.MODE_ECB)
        ciphertext = cipher.encrypt(self.pad(data.encode('utf8')))
        return ciphertext

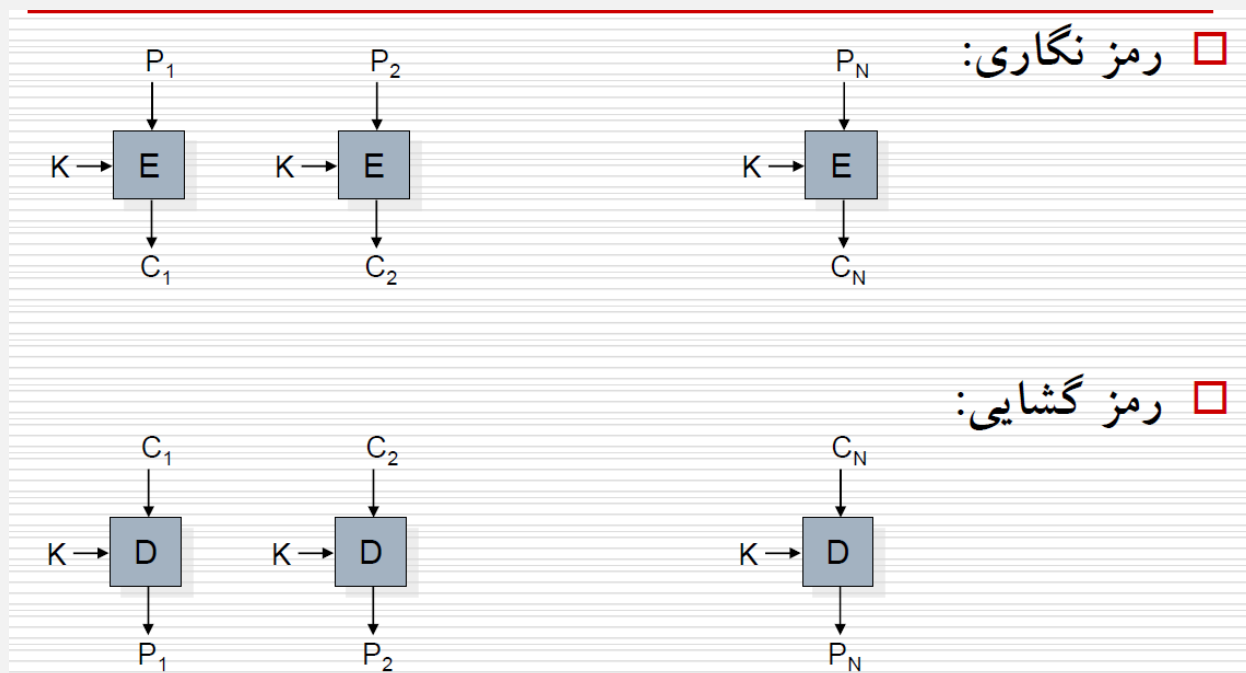
    def decrypt(self, encrypted_data, key):
        cipher = DES_ALGORITHM.new(key, DES_ALGORITHM.MODE_ECB)
        decrypted_data = cipher.decrypt(encrypted_data)
        return self.unpad(decrypted_data).decode()

    def pad(self, data):
        block_size = DES_ALGORITHM.block_size
        padding_length = block_size - (len(data) % block_size)
        padding = bytes([padding_length]) * padding_length
        return data + padding

    def unpad(self, data):
        padding_length = data[-1]
        return data[:-padding_length]
```



همانطور که طبق کد فوق قابل مشاهده است، الگوریتم DES با مد ECB، طراحی یافته است:



دو متد pad و unpad مناسب با نوع عملکرد DES رفتار می کنند و بیت هایی را به داده اصلی الحاق یا از آن جدا می کنند.

## RSA

این الگوریتم با دو کلید انجام می‌پذیرد. یکی کلید عمومی و دیگری کلید خصوصی. پس باید هنگام کار با آن دو کلید تولید کنیم. با کلید عمومی داده را رمز کنیم، و با کلید خصوصی داده را رمزگشایی کنیم.

با توجه به توضیحات بیان شده، بهتر است نگاهی به قطعه کد نوشته شده داشته باشیم:

```
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding
from algorithms.EncryptionAlgorithm import EncryptionAlgorithm

class RSA(EncryptionAlgorithm):
    def generate_key(self):
        private_key = rsa.generate_private_key(...)
        public_key = private_key.public_key

        return private_key, public_key

    def encrypt(self, data, key):
        _, public_key = key
        print('pub: ', public_key())
        cipher = public_key().encrypt(...)

        return cipher

    def decrypt(self, data, key):
        private, _ = key
        plaintext = private.decrypt(...)
        return plaintext.decode()

    def pad(self, data):
        return data

    def unpad(self, data):
        return data
```

همانطور که قابل مشاهده است این بار از کتابخانه cryptography استفاده کردیم زیرا که به ما امکان تولید کلید های عمومی و خصوصی را می‌داد. در ادامه منطق توضیح داده در ابتدا را می‌توان به صورت کد، مشاهده کرد (عملیات رمز کردن و رمزنگاری کردن).

حال نتیجه برنامه را در هر 3 حالت مشاهده خواهیم کرد:

فایل example.txt فایلی است که باید رمز شود:

```
example.txt X
code > phase2 > data > example.txt
1 Your mouth produces about one litre of saliva each day!
```

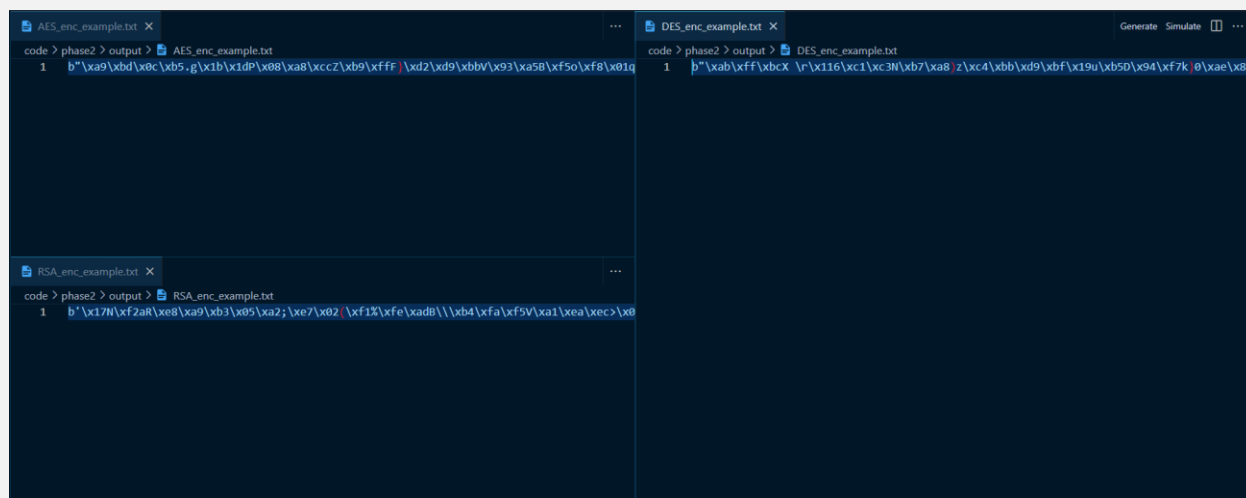
برنامه را به ازای هر سه الگوریتم اجرا می‌کنیم:

```
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase2> python.exe .\app.py -a AES -f example.txt
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase2> python.exe .\app.py -a DES -f example.txt
PS E:\University\Semester-7\Information Security\projects\1-eval-crack-encrypt\code\phase2> python.exe .\app.py -a RSA -f example.txt
```

در پوشه output که برای قرار دادن خروجی‌ها، تعبیه شده فایل‌های زیر ظاهر خواهند شد:

```
output
AES_dec_example.txt
AES_enc_example.txt
DES_dec_example.txt
DES_enc_example.txt
RSA_dec_example.txt
RSA_enc_example.txt
```

محتوای رمز شده به ازای هر الگوریتم را می‌توانید از شکل زیر مشاهده کنید:



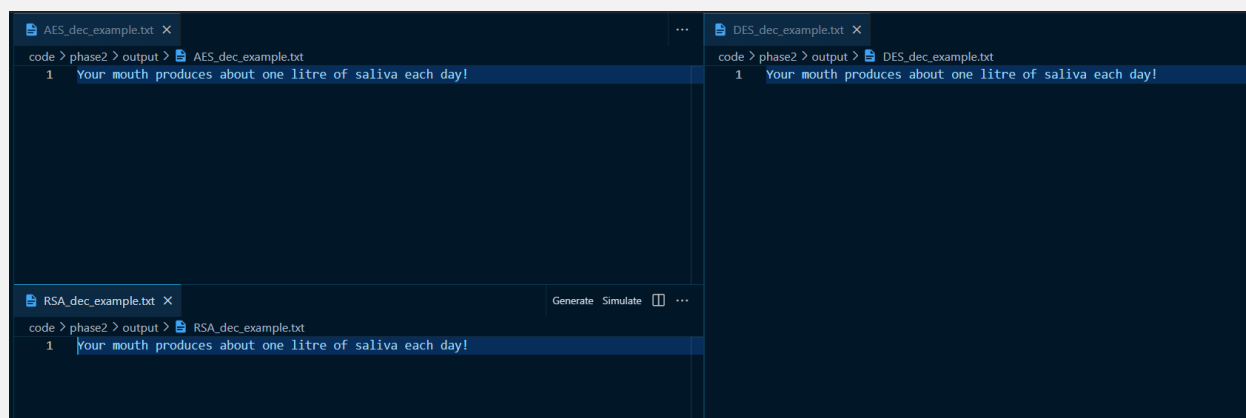
```

AES_enc_example.txt X
code > phase2 > output > AES_enc_example.txt
1 b"\xa9\xbd\xac\xb5.g\x1b\x1dP\x08\xccZ\xb9\xff\x02\xd9\xbbv\x93\xa58\xf50\xf8\x01g

DES_enc_example.txt X
code > phase2 > output > DES_enc_example.txt
1 b"\xab\xff\xbcX \r\x116\xc1xc3M\xb7\xa8 z\xc4\xbb\xd9\xbf\x19u\xb50\x94\xf7k \0\xae\x8b\

RSA_enc_example.txt X
code > phase2 > output > RSA_enc_example.txt
1 b"\x17N\xf2aR\xea\x9\x05\xa2;\xe7\x02 \xf1\xfe\xad0\\\xb4\xfa\xf5v\xa1\xea\xec>\x0
  
```

و در نهایت فایل رمزگشایی شده به ازای هر یک را، می‌توان در فایل‌های مربوطه مشاهده کرد:



```

AES_dec_example.txt X
code > phase2 > output > AES_dec_example.txt
1 Your mouth produces about one litre of saliva each day!

DES_dec_example.txt X
code > phase2 > output > DES_dec_example.txt
1 Your mouth produces about one litre of saliva each day!

RSA_dec_example.txt X
code > phase2 > output > RSA_dec_example.txt
1 Your mouth produces about one litre of saliva each day!
  
```