אבטחת תקשורת – תרגיל 1

עידן סימאי 206821258.
עידו אהרן 319024600.

שלב 1:
נתקין את הספריות הדרושות(בצורה הזו אנו לא מסתמכים על הגרסאות של הבודק, שכן בכל מקרה הוא יעבוד עם הגרסאות שכתבנו כאן).

```
try:
    from Cryptodome.Cipher import DES
    from Cryptodome.Util.Padding import pad,unpad
except ImportError:
    import subprocess
    import sys

    subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'pycryptodome'])
    from Cryptodome.Cipher import DES
    from Cryptodome.Util.Padding import pad,unpad
```

שלב 2:
נרפד את המחרוזת בצורה הבאה:

```
#padd the 'Hello World' string to be 16 long
padded_data = pad(b'Hello World', DES.block_size)
print(padded_data)
```

ונראה שקיבלנו את הדרוש:

```
[Running] python -u "c:\Users\idans\OneDrive\מסמכים\GitHub\CommunicationSecurityEx1\ex1.py"
b'Hello World\x05\x05\x05\x05\x05'
```

וכמו שמצופה בDES, מס' הבתים שריפדנו הוא 5, ולכן 5 הבתים האחרונים הם 5.
(Hello World זה 11 בתים, וצריך כפולה שלמה של 8).

שלב 3:
ניצור את המחרוזת המוצפנת בעזרת מופע של המחלקה DES, עם הפרמטרים המופיעים בקוד(המפתח 'poaisfun', אופן החלוקה לבלוקים, הפעלת ההצפנה, הפענוח – CBC ומחרוזת האתחול להצפנה הראשונית – IV היא מחרוזת שכולה 0).
נזכור שגודל בלוק בDES הוא 8 בתים ולכן 8 פעמים x00.
לאחר מכן הצפנו את הטקסט ממקודם('b'Hello World\x05\x05\x05\x05\x05'), והדפסנו בפורמט הקסה.

```
#The reuslt should be 0x33 0xaa 0xa3 0x1 0x7e 0x45 0x33 0x7b 0xd3 0x63 0x42 0xb3 0x92 0xb 0xe6 0x56.
des = DES.new(b'poaisfun', DES.MODE_CBC, b'\x00'*8)
ciphertext = des.encrypt(padded_data)
for i in ciphertext:
    print(hex(i))
```

נראה שקיבלנו את הדרוש:

```
0x33
0xaa
0xa3
0x1
0x7e
0x45
0x33
0x7b
0xd3
0x63
0x42
0xb3
0x92
0xb
0xe6
0x56
```

שלב 4:

נבטל את ההצפנה של המחרוזת מקודם.

לאחר מכן נוריד את הריפוד ממקודם(של 5 הבתים האחרונים).

```
#Decrypt the ciphertext from the previous question and print the plaintext, the reuslt should be b'Hello World'.
des = DES.new(b'poaisfun', DES.MODE_CBC, b'\x00'*8)
plaintext = des.decrypt(ciphertext)
plaintext = unpad(plaintext, DES.block_size)
print(plaintext)
```

נראה שקיבלנו את הדרוש:

```
b'Hello World'
```

שלב 5:
ניצור את פונקציית הXOR הבאה:
נשים לב שאנחנו מחזירים את התוצאה בפורמט של בתים, כדי לקבל את הדרוש.

```
#Write a xor fucntion that takes 3 bits and returns the result of the xor operation on them
def xor(a, b, c):
    return bytes([a ^ b ^ c])
```

נראה שקיבלנו את הדרוש:

```
print(xor(0,0,0))
print(xor(0,0,1))
print(xor(0,1,0))
print(xor(0,1,1))
print(xor(1,0,0))
print(xor(1,0,1))
print(xor(1,1,0))
print(xor(1,1,1))
```

נקבל:

```
b'\x00'
b'\x01'
b'\x01'
b'\x00'
b'\x01'
b'\x00'
b'\x00'
b'\x01'
```

שלב 6:
נבנה את הפונק' הבאה:
שזה בדיוק אותו דבר כמו הפענוח, רק שכאן לא בטוח שהוא יצליח כמו שצריך, ולכן אנחנו עוטפים בtry except.
אם לא הצלחנו(כלומר במס' הבתים שרופדו בסוף לא קיבלנו בדיוק את אותו המס' בplaintext_tag,
לדוגמה במקרה שלנו לעיל ריפדנו ב5 – אז אם לאחר הפענוח אצלנו לא קיבלנו בplaintext_tag
בסוף 5 בתים של '\x05') האורקל יחזיר Fales.
אם כן קיבלנו, נחזיר True ובעצם נוכל להתקדם לשלב הבא בהתקפה.

```python
#This Oracle function checks whether the ciphertext is a valid DES ciphertext or not.
def Oracle(ciphertext, key, iv):
    try:
        des = DES.new(key, DES.MODE_CBC, iv)
        plaintext_tag = des.decrypt(ciphertext)
        print(plaintext_tag)
        plaintext_tag = unpad(plaintext_tag, DES.block_size)
        return True
    except:
        return False
```

נראה שקיבלנו את הדרוש:

```python
def test_oracle(ciphertext):
    original_bytes = ciphertext
    index_to_change = 5 #index
    new_byte = b'\x50'  # New byte value
    modified_bytes = original_bytes[:index_to_change] + new_byte + original_bytes[index_to_change + 1:]
    for i in modified_bytes:
        print(hex(i), end=' ')
    print()
    print(Oracle(modified_bytes, b'poaisfun', b'\x00'*8))
```

ניסינו לשנות את הבית באינדקס ה5 בchipertext.
ולאחר מכן הכנסנו את המקורית לOracle ונראה מה נקבל.

```python
test_oracle(ciphertext)
print(Oracle(ciphertext, b'poaisfun', b'\x00'*8))
```

נראה מה קיבלנו:

```
0x33 0xaa 0xa3 0x1 0x7e 0x50 0x33 0x7b 0xd3 0x63 0x42 0xb3 0x92 0xb 0xe6 0x56
b'\xa8\x8a\xc5\xbeI\xbb\xd6\x8crld\x05\x05\x10\x05\x05'
False
b'Hello World\x05\x05\x05\x05\x05'
True
```

השורה הראשונה היא הcipher_text לאחר ששמנו את הבית x50 באינדקס ה5(כלומר בעצם modified_bytes).
ניתן לראות שלא כל 5 הבתים האחרונים הם \x05, ולכן קיבלנו False.
לאחר מכן, כשהכנסנו לOracle(לא לtest_oracle), קיבלנו True, כנדרש.
***נשים לב, שיכול להיות שנשנה משהו בchipertext ובכל זאת נקבל True, אם רצף הבתים בסוף נשאר כרצף מרופד תקין, שזה בעצם כיצד ממשיכים את ההתקפה.**

שלב 7:
ניצור את המשתנה c בצורה הבאה:

```python
#c is a concatenation of a block of 0 and the second block of the ciphertext
c = b'\x00'*8 + ciphertext[8:16]
for i in c:
    print(hex(i))
```

כלומר שרשור של בית של \x00 ואז את הבלוק השני של ciphertext.
נראה שקיבלנו את הדרוש:

```
0x0
0x0
0x0
0x0
0x0
0x0
0x0
0x0
0xd3
0x63
0x42
0xb3
0x92
0xb
0xe6
0x56
```

שלב 8:

ניצור את הפונק' הבאה:

```python
c = create_c()
#Increase c's eight byte by one till we get True.
def run_till_true(c):
    while not Oracle(c, b'poaisfun', b'\x00'*8):
        c = c[:7] + bytes([c[7] + 1]) + c[8:]
    return c

c = run_till_true(c)
print_c_in_hexa(c)
```

נעבור בלולאה, כל עוד האורקל לא החזיר True על c.

(נעזרנו בפונק' create_c שהיא בעצם הפונק' מהסעיף הקודם, שיוצרת את c):

```python
def create_c():
    c = b'\x00'*8 + ciphertext[8:16]
    print_c_in_hexa(c)
    return c
```

הפונק' print_c_in_hex:

```python
def print_c_in_hexa(c):
    for i in c:
        print(hex(i), end=' ')
    print()
```

לאחר שעצרנו בלולאה בrun_till_true, נבדוק מה קיבלנו:

```
0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x7f 0xd3 0x63 0x42 0xb3 0x92 0xb 0xe6 0x56
```

כלומר הבית ה8 השתנה ל0x7f.

לעומת ההדפסה של c שהייתה לפני:

```
0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0xd3 0x63 0x42 0xb3 0x92 0xb 0xe6 0x56
```

שלב 9:

כיוון שאנחנו רוצים לקבל את הבית האחרון בבלוק השני, לפי הנוסחה:

$$P_i[x] = P_2'[x] \oplus C_{i-1}[x] \oplus X_j[x]$$

אנחנו יודעים כי $P_2'[x] = P_2'[7] = 0x01$, כיוון שאנו רוצים שהאורקל יחזיר אמת(כלומר, שהשרשור בין X_J לC_2 בבית האחרון יחזיר אמת, ולפי DES אם יש בית אחד שרוצים לרפד שמים בו $0x01$, אם שניים שמים פעמיים $0x02$ וכו').
את $C_1[7]$ אנחנו גם יודעים(זהו בעצם הבית באינדקס ה7 ב ciphertext המקורי), ועבור $X_j[7]$ מצאנו אותו בסעיף הקודם, שכן זהו הבית שמחזיר true באורקל.

```
#Using the equation from class and the xor function, lets find the second block's last byte.
#The answer Needs to be 0x05.
p = xor(0x01, ciphertext[7], c[7])
print(hex(p[0]))
```

כעת נראה את הערך שקיבלנו:

```
0x5
```

כדרוש.

שלב 10:
כיוון שאנחנו רוצים לקבל את הבית האחרון בבלוק השני, לפי הנוסחה:

$$P_i[x] = P_2'[x] \oplus C_{i-1}[x] \oplus X_j[x]$$

אנחנו רוצים שיתקיים כי $P_2'[x] = P_2'[7] = 0x02$, את $C_1[7]$ אנחנו גם יודעים(זהו בעצם הבית באינדקס ה7 ב ciphertext המקורי), וקיבלנו בסעיף הקודם ש$P_i[7] = 0x05$.
כעת, נחלץ את $X_j[7]$.

```
#Using the equation from class and the xor function, let's find what c'[7] should be so P'_2[7] = 0x02, we already know P_2[7].
c_tag = xor(0x02, ciphertext[7], p[0])
print(hex(c_tag[0]))
```

נקבל:

```
0x7c
```

כלומר הבית באינדקס ה7 בc יהיה 0x7c.

```
0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x7c 0xd3 0x63 0x42 0xb3 0x92 0xb 0xe6 0x56
```

כדרוש.

שלב 11:
נשנה את הקוד בצורה הבאה:
נחליף את הפונק' שרצה על האורקל שתהיה עם אינדקס כללי, במקום 7 קבוע שהיה מקודם.
כעת, בפונק' הכללית למציאת הבלוק, אנו מאתחלים את X_j לבלוק של 0, ונחזיר לבסוף את התוצאה בplain_text_second_block.
אנו מתחילים לרוץ על כל האינדקסים, על מנת לחשב מה הערכים של X_j צריכים להיות, לפי הערכים בplaintext המקורי.
(באיטרציה הראשונה איננו נכנסים לשם, כיוון שאנו יודעים שזה אמור להיות 0x01 ב'P, אז משיגים את התו האחרון בplaintext המקורי).
לאחר מכן, אחרי שיש לנו את הערכים שאנו צריכים בX_j לפי הערכים הקודמים שיצאו לplaintext, אנו מייצרים את בלוק השרשור בין הבלוק השני המקורי של הciphertext לבין הX_j, ואת זה שולחים לבדיקה באורקל על מנת לדעת מה הבית הבא בX_j.
לאחר שקיבלנו את השרשור המעודכן, נחשב את הבית הבא בטקסט המקורי.
ככה נרוץ על כל האינדקסים(0 – 7).

```python
def find_second_block():
    Xj = bytes([0x00] * 8)
    plaintext_second_block = bytes([0x00] * 8)
    c = create_c_second_block()
    for i in range(7, -1, -1):
        Xj = bytes([0x00] * 8)
        for j in range(7, i, -1):
            p = plaintext_second_block[j]
            xj = xor(0x01 + 7 - i, ciphertext[j], p)[0]
            Xj = Xj[:j] + bytes([xj]) + Xj[j + 1:]
        c = Xj + ciphertext[8:16]
        print()
        print_c_in_hexa(c)
        print()
        c = run_till_true_general(c, i)
        p = xor(0x01 + 7 - i, ciphertext[i], c[i])[0]
        plaintext_second_block = plaintext_second_block[:i] + bytes([p]) + plaintext_second_block[i + 1:]

    print_c_in_hexa(plaintext_second_block)
find_second_block()
```

לבסוף קיבלנו:

```
0x72 0x6c 0x64 0x5 0x5 0x5 0x5 0x5
```

שזה בעצם:

'rld\x05\x05\x05\x05\x05'

שלב 12:

נשנה את הפונק' שנעזרנו בהן מקודם לכלליות:

```python
def run_till_found(c, i, key, iv):
    while not Oracle(c, key, iv):
        c = c[:i] + bytes([c[i] + 1]) + c[i + 1:]
    return c

def create_c_block(i):
    c = b'\x00'*8 + ciphertext[i:i + 8]
    return c
```

הראשונה היא כללית למציאת הבית שהאורקל מחזיר עליו אמת, השנייה היא ליצירת מחרוזת השרשור בין X_j לCiphertext שאותה בעצם שולחים לאורקל.

```python
def find_block(k, ciphertext, key, iv):
    Xj = bytes([0x00] * 8)
    plaintext_block = bytes([0x00] * 8)
    start_index = (k - 1) * 8
    c = create_c_block(start_index)
    for i in range(7, -1, -1):
        Xj = bytes([0x00] * 8)
        for j in range(7, i, -1):
            p = plaintext_block[j]
            if start_index != 0:
                xj = xor(0x01 + 7 - i, ciphertext[start_index - 8 + j], p)[0]
            elif start_index == 0:
                xj = xor(0x01 + 7 - i, iv[j], p)[0]
            Xj = Xj[:j] + bytes([xj]) + Xj[j + 1:]
        c = Xj + ciphertext[start_index: start_index + 8]
        c = run_till_found(c, i, key, iv)
        if start_index != 0:
            p = xor(0x01 + 7 - i, ciphertext[start_index - 8  + i], c[i])[0]
        elif start_index == 0:
            p = xor (0x01 + 7 - i, iv[i], c[i])[0]
        plaintext_block = plaintext_block[:i] + bytes([p]) + plaintext_block[i + 1:]
    return plaintext_block

def find_all_blocks(ciphertext, key, iv):
    blocks = []
    k = (int((len(ciphertext)) // 8))
    for i in range(k, 0, -1):
        blocks.append(find_block(i, ciphertext, key, iv))
    blocks.reverse()
    return blocks
```

הפונק' find_block היא זהה לפונק' מהסעיף הקודם, רק שכאן זה עבור בלוק כלשהו ולא בהכרח השני(וניתן לראות זאת לפי המשתנים), יש לשים לב שכאשר אנחנו מנסים לפענח את הבלוק הראשון אנחנו עושים זאת בעזרת הIV ולא בעזרת הCiphertext (שכן לפי הנוסחה אנחנו צריכים את בלוק הCiphertext הקודם, אך אין קודם במקרה הזה, ולכן אנחנו משתמשים בIV), וניתן לראות את התנאים בקוד שבודקים זאת.
בפונק' השנייה אנחנו בעצם עוברים על כל הבלוקים(מהסוף להתחלה), ומכניסים את כולם למבנה הנתונים שהוא רשימה של בלוקים, ואז עושים reverse לרשימה.

```python
def print_plaintext(ciphertext, key, iv):
    #convert ciphertext from string to bytes
    ciphertext = bytes.fromhex(ciphertext)
    #convert key from string to bytes
    key = bytes(key, 'utf-8')
    #convert iv from string to bytes
    iv = bytes.fromhex(iv)

    plaintext = find_all_blocks(ciphertext, key, iv)
    # for p in plaintext:
    #     print_c_in_hexa(p)
    text = bytes()
    for block in plaintext:
        text += block
    text = unpad(text, DES.block_size)
    text = text.decode()
    print(text)

print_plaintext(sys.argv[1], sys.argv[2], sys.argv[3])
```

כאן, כיוון שאנו מקבלים את הקלט ciphertext בהקסה, את המפתח בstring ואת הiv בהקסה, אנחנו ממירים, שולחים לפונק' לעיל, עושים לבסוף unpad אחרי ששרשרנו, ולבסוף ממירים חזקה לstring ומדפיסים למסך, כפי שנדרש בתרגיל.

אנו קוראים לפונק' הזו עם הארגומנטים מהמשתמש.

ניתן לראות שעבור הקלט מהמודל:

83e10d51e6d122ca3faf089c7a924a7b mydeskey 0000000000000000

נקבל:

```
PS C:\Users\idans\OneDrive\מסמכים\GitHub\CommunicationSecurityEx1> python ./ex1.py 83e10d51e6d122ca3faf089c7a924a7b myde
skey 0000000000000000
Hello World
PS C:\Users\idans\OneDrive\מסמכים\GitHub\CommunicationSecurityEx1>
```

כדרוש.