בס"ד

הסבר

בתיקייה זו ישנן 4 תמונות:

Picture1 and picture2: שתי התמונות שמהן נרצה לייצר את הקובץ המשותף.

Final: התמונה שנעלה לאתר

Final_encrypted: התמונה לאחר ההצפנה.

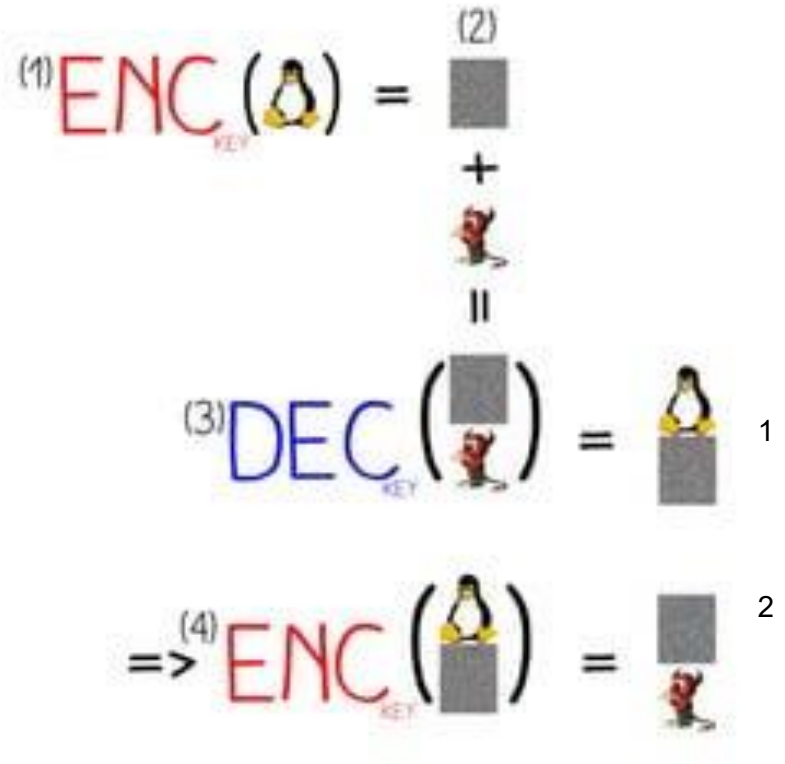
כמו כן, יש פה קובץ encrypt_aes, שתפקידו להצפין תמונה אחת ב AES-CBC.

בנוסף, יש פה את הקובץ החשוב שמטרתו היא ליצור את התמונה Final, תמונה שמייצגת תמונה מסויימת ולאחר הצפנה תביא לנו תמונה אחרת.

הסבר על התהליך:

מה שקורה בעצם זה שאתה מקבל תמונה אחת, מצפין אותה -> ויוצאת תמונה אחרת.. בדר"כ, זה לא אמור לקרות. אז מה בעצם קורה?

זה מבוסס על משהו שלמדנו בקורס הקודם, יסודות באבטחת תוכנה.



זה בעצם הטריק.

בעצם, מה שאני צריך למצוא פה זה IV כזה ככה שב 16 הבתים הראשונים, הוא ייתן לי header מתאים שיגרום לי להתעלם מהאורך של הקובץ השגוי (האורך של tux), ואז הוא יציג לי את התמונה של השטן.

מה שהמשתמש מקבל בהתחלה, זה (1), ובסוף, לאחר ההצפנה מתקבל אצלו (2).

קטע 1:

בס"ד

פה בעצם מקבלים את שני הקבצים, ובודקים שהגודל של הקובץ הראשון הוא קטן מ 0xffff, אחרת זה לא יעבוד.

```
def main():
    KEY = b'Secr3tKeyForAES!'
    key = bytes(KEY)

    input_file_path1 = r"picture1.jpg"
    input_file_path2 = r"picture2.jpg"

    output_file_path = r"final.jpg"

    with open(input_file_path1, 'rb') as input_file1:
        content1 = input_file1.read()
        header1 = content1[:16]
    with open(input_file_path2, 'rb') as input_file2:
        content2 = input_file2.read()

    output_file = open(output_file_path, 'wb')

    original_size = len(content1)
    if original_size>=0xffff:
        print(f"ERROR, program won't work, first image is too long,
{hex(original_size)} is bigger than 0xffff")
        return
```

זה ה header שאנחנו רוצים שיתקבל בשלב 2, בתחילת הקובץ, אחרי שנעשה encrypt ל tux עם ה IV.

```
 # create the header:
    # FFD8FFFE + size_of_chunk + padding to 16 bytes
    cipher_block1 = b'\xff\xd8\xff\xfe' + original_size.to_bytes(2,
byteorder='big') + b'\x00' * 10

    cipher = Cipher(algorithms.AES(key), modes.CBC(b'\x00'*16)) # at the
beginning, iv = 0
```

זה ה header שאנחנו רוצים שיתקבל בשלב 2, בתחילת הקובץ, אחרי שנעשה encrypt ל tux עם ה IV.

לכן, אנחנו עושים את התהליך ההפוך. בתור התחלה עושים decrypt, ואז עושים xor לכל דבר עם ה header המקורי, כדי לגלות מה ה IV שבאמצעותו בתהליך ההצפנה יתקבל ה header שניסינו ליצור.

```
# decrypt first cipher_block, to get the new iv
    decryptor = cipher.decryptor()
    decrypted_cipher_block1 = decryptor.update(cipher_block1)

    iv = bytes([ decrypted_cipher_block1[i] ^ header1[i] for i in range(16) ])

    print(f"input file1 is: {input_file1.name}\ninput file2 is:
{input_file2.name}")
```

בס"ד

```
    print(f"IV is: {list(iv)}")
```

לבסוף, אנחנו עושים decrypt להכל ביחד, וזה התוכן שהמשתמש מקבל

```
cipher = Cipher(algorithms.AES(key), modes.CBC(iv))

    encryptor = cipher.encryptor()
    content1_encrypted = encryptor.update(content1)

    res = content1_encrypted + content2 # becuase i remove the signature,
which is: 0xFFD8FFFE, (it works also without the removing of the target.
interesting. ah, the size of the junk chunk removes the 4 bytes of the
signature, genius.)

    decryptor = cipher.decryptor()

    file_decrypted = decryptor.update(res)

    output_file.write(file_decrypted)

    print(f"output was writted to {output_file.name}")
```

הטריק הזה לקוח מהמקור הבא:

https://www.slideshare.net/slideshow/when-aes-a-cryptobinary-magic-trick/33101365#43