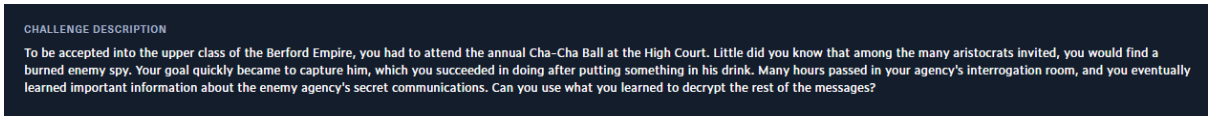
Erel Regev

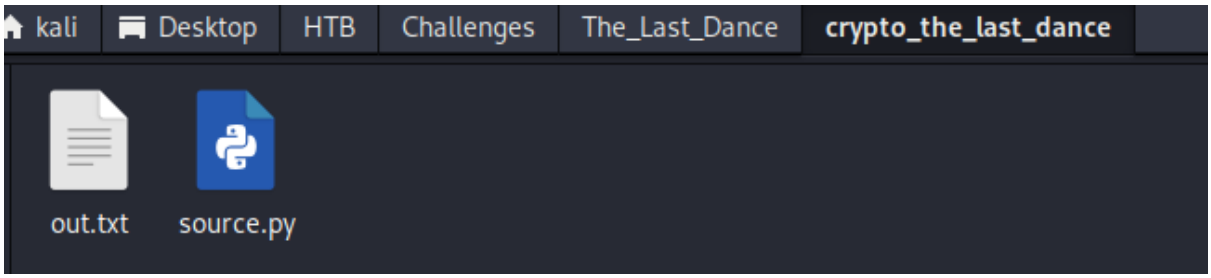
Table of Contents

Intro

Challenge description:



Received files:



Viewing out.txt:

```
1  c4a66edfe80227b4fa24d431
2  7aa34395a258f5893e3db1822139b8c1f04cfab9d757b9b9cca57e1df33d093f07c7f06e06bb6293676f9060a838ea138b6bc9f20b08afeb73120506e2ce7b9b9dcd9e4a421584cfaba2481132dfbdf4216e98e3facec9ba199ca3a97641e9c
3  7d8273ceb459e4d4386df4e32e1aecc1aa7aaafda50cb982f6c62623cf6b29693d86b15457aa76ac7e2eef6cf814ae3a8d39c7
```

Viewing source.py:

Erel Regev

```python
from Crypto.Cipher import ChaCha20
from secret import FLAG
import os


def encryptMessage(message, key, nonce):
    cipher = ChaCha20.new(key=key, nonce=iv)
    ciphertext = cipher.encrypt(message)
    return ciphertext


def writeData(data):
    with open("out.txt", "w") as f:
        f.write(data)


if __name__ == "__main__":
    message = b"Our counter agencies have intercepted your messages and a lot "
    message += b"of your agent's identities have been exposed. In a matter of "
    message += b"days all of them will be captured"

    key, iv = os.urandom(32), os.urandom(12)

    encrypted_message = encryptMessage(message, key, iv)
    encrypted_flag = encryptMessage(FLAG, key, iv)

    data = iv.hex() + "\n" + encrypted_message.hex() + "\n" + encrypted_flag.hex()
    writeData(data)
```

Source.py

Source.py is a Python script that demonstrates the encryption of a message and a flag using the ChaCha20 stream cipher. The script generates a random key and nonce (also referred to as IV, Initialization Vector) for encryption. The encrypted message and flag are then written to an output file named "out.txt" in hexadecimal format.

The encryptMessage function takes three parameters:

- The message to be encrypted
- The encryption key
- The nonce iv (Initialization Vector).

Inside the encryptMessage function:

- It creates a ChaCha20 cipher object with the given key and nonce (iv).
- It then encrypts the input message using the cipher.
- The encrypted ciphertext is returned.
- The writeData function takes data as a parameter and writes the provided data to an output file named "out.txt".

In the if __name__ == "__main__": block:

- A sample message is created by concatenating several b-prefixed bytes objects.
- A random key and iv (nonce) are generated using the os.urandom function.
- The encryptMessage function is called twice: once to encrypt the sample message and once to encrypt the FLAG variable.

Erel Regev

- The encrypted iv, encrypted message, and encrypted FLAG are all converted to hexadecimal representation and concatenated into a single string.
- The writeData function is called to write this concatenated data to an output file named "out.txt".

Based on the structure of the code I assume that:

- The first line is the IV (nonce) value in hexadecimal format.
- The second line is the encrypted message in hexadecimal format.
- The third line is the encrypted flag in hexadecimal format.

So obviously it needs to be decrypted.

dec.py

I wrote the following script:

```
known_plaintext = b"Our counter agencies have intercepted your messages and a lot "
known_ciphertext = bytes.fromhex("7aa34395a258f5893e3db1822139b8c1f04cfab9d757b9b9cca57e1df33d093f07c7f06e06bb6293676f9060a838ea138b6bc9f20b08afeb73120506e2ce7b9b9dcd9e4a421584cfaba2481132df

# Calculate the keystream by XORing known plaintext with known ciphertext
keystream = bytes([p ^ c for p, c in zip(known_plaintext, known_ciphertext)])

# Encrypted Flag
encrypted_flag = bytes.fromhex("7d8273ceb459e4d4386df4e32e1aecc1aa7aaafda50cb982f6c62623cf6b29693d86b15457aa76ac7e2eef6cf814ae3a8d39c7")

# Decrypt the flag by XORing encrypted flag with the keystream
decrypted_flag = bytes([k ^ c for k, c in zip(keystream, encrypted_flag)])

print("Decrypted Flag:", decrypted_flag)
```

known_plaintext:

This is the known plaintext message provided in the challenge source code: "Our counter agencies have intercepted your messages and a lot "

known_ciphertext:

This is the corresponding ciphertext of the known plaintext. The challenge has provided this ciphertext as a hex string:
"7aa34395a258f5893e3db1822139b8c1f04cfab9d757b9b9cca57e1df33d093f07c7f06e06bb6293676f9060a838ea138b6bc9f20b08afeb73120506e2ce7b9b9dcd9e4a421584cfaba2481132dfbdf4216e98e3facec9ba199ca3a97641e9ca9782868d0222a1d7c0d3119b867edaf2e72e2a6f7d344df39a14edc39cb6f960944ddac2aaef324827c36cba67dcb76b22119b43881a3f1262752990"

Calculate the Keystream:

The keystream is calculated by XORing each byte of the known_plaintext with the corresponding byte of the known_ciphertext. This is based on the property of the XOR operation that allows you to recover one of the operands if the other is known. This keystream will be used to decrypt the flag later.

HTB Challenge: The Last Dance - Subject: Crypto - Difficulty: Easy

Erel Regev

encrypted_flag:

This is the encrypted flag provided in the challenge source code as a hex string.

Decrypt the Flag:

To decrypt the flag, the script uses the calculated keystream and XORs each byte of the encrypted_flag with the corresponding byte of the keystream. This operation cancels out the encryption, revealing the original flag in bytes.

XORing (exclusive OR) is a fundamental operation in cryptography, and it's used in various encryption algorithms, including ChaCha20. XORing is a bitwise operation that combines two binary values. In cryptography, XORing is often used for several purposes:

Keystream Generation (Stream Ciphers like ChaCha20):

ChaCha20, like many stream ciphers, uses XORing as a key step in generating a keystream. The keystream is a sequence of random-looking bits that are generated based on a secret key and an initialization vector (IV). The keystream is then combined with the plaintext (or ciphertext) using XOR to produce the ciphertext (or plaintext).

One-Time Pads:

In a one-time pad encryption scheme, XORing the plaintext with a random key (the one-time pad) creates the ciphertext. Decryption is achieved by XORing the ciphertext with the same random key.

Error Detection and Correction:

XORing is used in error detection and correction codes. For example, in parity checking, a parity bit is XORed with the data bits to create a parity check bit. XORing the data and the check bit can help detect errors.

Data Manipulation:

XORing can be used to manipulate data, such as flipping specific bits to modify data without affecting other bits.

In the context of ChaCha20, XORing is part of the process used to generate the keystream, which is then used to encrypt or decrypt data.

HTB Challenge: The Last Dance - Subject: Crypto - Difficulty: Easy

Erel Regev

```
┌──(kali㉿kali)-[~/…/HTB/Challenges/The_Last_Dance/crypto_the_last_dance]
└─$ python3 dec.py
Decrypted Flag: b'HTB{u                                                  7}'
```