

8-11-1-Angewandte_Kryptographie_Notes

Autor: Manuel Fellner

Version: 05.03.2024

#SYT

#Labor

#4BHIT

#4-SoSe

#Angewandte-Kryptographie

1. Einführung

Vorgeschichte:

Bei einem Ransomware-Angriff auf deinen Rechner wurden einige wichtige Dateien verschlüsselt. Aus online verfügbaren Analysen weißt du, dass die Täter dabei wie folgt vorgehen (Diese Darstellung ist ein Wenig vereinfacht);

1. Angreifer generieren zunächst für jedes Opfer ein RSA 2048bit-Schlüsselpaar (public & private key)
2. Der public key wird zum Opfer transferiert
3. Der private key bleibt unter der Kontrolle der Angreifer
4. Für jede zu verschlüsselnde Datei wird ein AES-128 Schlüssel generiert
5. Die Datei wird danach mit diesem im AES-CBC-Modus verschlüsselt
6. Der Schlüssel selbst wird mit dem RSA public key des Opfers verschlüsselt und an die verschlüsselte Datei angehängt

Zum Glück wurde auf der Plattform `nomoreransom.org` bekanntgegeben, wer die Ransomware betreibt - Dadurch haben wir den vom Angreifer verwendeten RSA key `key.pem`

Aufgabenstellung:

- Rekonstruiere die Inhalte der verschlüsselten Datei `wichtig.enc`:
 - Extrahiere den verschlüsselten AES-Key aus der Datei
 - Entschlüssele diesen Key
 - Verwende den entschlüsselten Key zum Wiederherstellen der Daten
 - Erstelle ein Programm, das deine Wiederherstellungsschritte automatisiert und eine gegebene verschlüsselte Datei automatisch entschlüsseln kann
- Sind die verwendeten Algorithmen und Schlüssellängen (AES und RSA) zur Zeit als sicher eingestuft? - Begründe deine Antwort

Hinweis:

Der Initialisierungsvektor beim CBC-Modus ist nur relevant, wenn der selbe Key mehrfach verwendet wird. Ist dies nicht der Fall, so wird häufig 0.....000 verwendet.

2. Durchführung

2.1 Manuelle Durchführung

- Der AES-128 Key ist hinten an der Datei angehängt
 - Wir müssen also die letzten Bytes vom `wichtig.enc` file nehmen
 - Wie viele Bytes? Dafür müssen wir uns die Verschlüsselungsalgorithmen genauer anschauen: Das `wichtig.enc` ist mittels RSA **2048bit** Verschlüsselt, was bedeutet, dass die Blockgröße von `wichtig.enc` = 2048bit = 256 Bytes ist.
 - Das bedeutet, dass wir einfach den letzten Block, also die letzten 256 Bytes von `wichtig.enc` nehmen und diese entschlüsseln müssen.
- Um uns die letzten Bytes eines Files zu holen, können wir `tail` verwenden.

```
TAIL(1)                                     User Commands                                     TAIL(1)

NAME
    tail - output the last part of files

SYNOPSIS
    tail [OPTION]... [FILE]...

DESCRIPTION
    Print the last 10 lines of each FILE to standard output.  With more than one FILE, precede each with a
    header giving the file name.

    With no FILE, or when FILE is -, read standard input.

    Mandatory arguments to long options are mandatory for short options too.

    -c, --bytes=[+]NUM
        output the last NUM bytes; or use -c +NUM to output starting with byte NUM of each file
```

- Nun holen wir uns die letzten 256 Bytes des `wichtig.enc` files:
 - `cat wichtig.enc | tail -c 256 >> aes-key`
- Als nächstes Entschlüsseln wir den key, da dieser ja immer noch mittels RSA 2048bit verschlüsselt ist:
 - `openssl rsautl -decrypt -in aes-key.enc -out aes-key -inkey key.pem`
 - `aes-key : 172abe01891111000deadbeef0000101 (Hex)`
- Nun müssen wir die Nachricht an sich entschlüsseln:
 - Dafür müssen wir aber die letzten 256 Bytes von `wichtig.env` NICHT beachten, da in dem Block ja der AES-128 key drangehängt ist

- Dafür können wir head verwenden:

```
HEAD(1)
NAME
    head - output the first part of files

SYNOPSIS
    head [OPTION]... [FILE]...

DESCRIPTION
    Print the first 10 lines of each FILE to standard output.  With more than one FILE, precede each with a header giving the file name.

    With no FILE, or when FILE is -, read standard input.

    Mandatory arguments to long options are mandatory for short options too.

    -c, --bytes=[-]NUM
        print the first NUM bytes of each file; with the leading '-', print all but the last NUM bytes of each file
```

- Wir entfernen die letzten 256 Bytes: `cat wichtig.enc | head -c 1016320 >> wichtig-correct-size.enc`

Als nächstes gehen wir auf <https://gchq.github.io> und entschlüsseln den Text mit folgender Eingabe:

entschlüsselter key

IV (initialisierungsvektor), hier einfach 0

decrypted file

encrypted file mit minus 256 bytes am ende

- Key : 172abe01891111000deadbeef0000101
- IV : 00000000000000000000000000000000
- Mode : CBC (weil aes-128-cbc)
- Input : Raw
- Output : Raw
- Der Inhalt ist ein "THE LORD OF THE RINGS" Buch

2.2 Automatisierte Durchführung

Nun automatisieren wir diesen Prozess des entschlüsseln.

Steps, die das Programm ausführen muss:

- Die letzten 256 Byte des Files entfernen, in ein extra file speichern
- Die gespeicherten 256 Bytes mit RSA entschlüsseln mit key.pem, speichern

3. Das file decrypten: `openssl enc -aes-128-cbc -nosalt -d -in wichtig-correct-size.enc -K '172abe01891111000deadbeef0000101' -iv '00000000000000000000000000000000' >> output`

Umgesetzt schaut das Python script dann folgendermaßen aus:

```
import os

from Crypto.PublicKey import RSA
from Crypto.Cipher import AES, PKCS1_v1_5
from Crypto.Random import get_random_bytes

# step 1: remove the last 256 Bytes and store them

with open('wichtig.enc', 'rb') as f_wichtig:
    f_wichtig.seek(-256, os.SEEK_END)
    last_256_bytes = f_wichtig.read()
with open('aes-key.bin', 'wb') as aes_key_file:
    aes_key_file.write(last_256_bytes)

# step 2: decrypt the aes_key_file with RSA and the key.pem file

rsa_private_key = RSA.import_key(open('key.pem').read())

with open('aes-key.bin', 'rb') as aes_key_file_read:
    enc_session_key =
aes_key_file_read.read(rsa_private_key.size_in_bytes())
    ciphertext = aes_key_file_read.read()

sentinel = get_random_bytes(16)
cipher_rsa = PKCS1_v1_5.new(rsa_private_key)
aes_key = cipher_rsa.decrypt(enc_session_key, sentinel)

# remove the last byte (new line)
aes_key = aes_key[:-1]
aes_key = bytes.fromhex(aes_key.decode())

# in our case, the iv is 0
iv = bytes(16)
cipher_aes = AES.new(aes_key, AES.MODE_CBC, iv)
decrypted_data = cipher_aes.decrypt(ciphertext)

# step 3: decrypt the entire data file

with open('wichtig.enc', 'rb') as f_wichtig_correct_size:
    cipher_text = f_wichtig_correct_size.read()

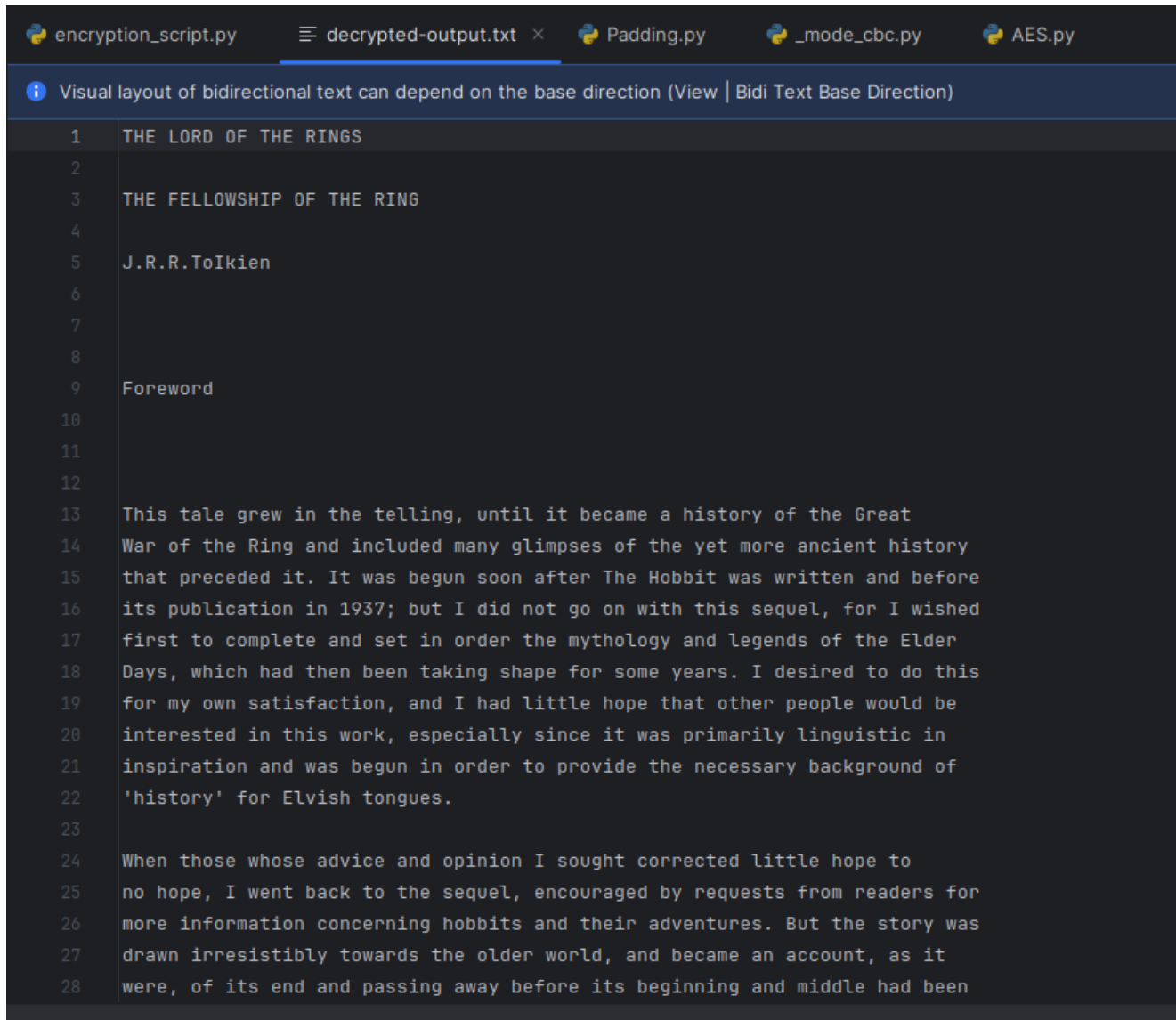
cipher = AES.new(aes_key, AES.MODE_CBC, iv)
```

```
message = cipher.decrypt(cipher_text)

with open('decrypted-output.txt', 'w') as decrypted_output:
    decrypted_output.write(message.decode('utf-8', 'replace'))
```

- Hier wird unten aber noch der Datenschrott (AES-128 Key) drangehängt

Wenn wir das Script ausführe, erhalten wir folgendes `decrypted_output.txt` file:



The screenshot shows a code editor with several tabs: `encryption_script.py`, `decrypted-output.txt` (active), `Padding.py`, `_mode_cbc.py`, and `AES.py`. A message bar at the top states: "Visual layout of bidirectional text can depend on the base direction (View | Bidi Text Base Direction)". The active tab displays the decrypted text, which is the first chapter of 'The Lord of the Rings' by J.R.R. Tolkien. The text is as follows:

```
1 THE LORD OF THE RINGS
2
3 THE FELLOWSHIP OF THE RING
4
5 J.R.R.ToIkien
6
7
8
9 Foreword
10
11
12
13 This tale grew in the telling, until it became a history of the Great
14 War of the Ring and included many glimpses of the yet more ancient history
15 that preceded it. It was begun soon after The Hobbit was written and before
16 its publication in 1937; but I did not go on with this sequel, for I wished
17 first to complete and set in order the mythology and legends of the Elder
18 Days, which had then been taking shape for some years. I desired to do this
19 for my own satisfaction, and I had little hope that other people would be
20 interested in this work, especially since it was primarily linguistic in
21 inspiration and was begun in order to provide the necessary background of
22 'history' for Elvish tongues.
23
24 When those whose advice and opinion I sought corrected little hope to
25 no hope, I went back to the sequel, encouraged by requests from readers for
26 more information concerning hobbits and their adventures. But the story was
27 drawn irresistibly towards the older world, and became an account, as it
28 were, of its end and passing away before its beginning and middle had been
```

Frage:

Sind die verwendeten Algorithmen und Schlüssellaengen (AES und RSA) zur Zeit als sicher eingestuft?

- Ja, beide Verschlüsselungsalgorithmen sind (Stand: 07.03.2024) noch sicher. Ebenso sind die Schlüssellängen von 128bit bzw. 2048bit immer noch sicher, auch wenn 128bit für die Zukunft vielleicht nicht mehr ausreichen wird.

- AES-128 hat eine Schlüssellänge von $2^{128} = 340282366920938463463374607431768211456$, diese Schlüssel sind also definitiv noch sicher.