



Information Security

Lab 3

Héctor Núñez Fernández

In this lab, I will make an implementation of AACS (Advanced Access Content System). The source code is located in <https://github.com/hectornunfer/SI-Munics/tree/main/Lab3>. Before all, you need to install the libraries from **requirements.txt**, to do that, just run `pip install requirements.txt`.

The code is composed only by 1 file:

1- lab3-image.py

In this file resides all the functions and the implementation of the AACS, encrypting and decrypting an image. The code is commented with the explanation of each operation.

```
1 import os
2 from io import BytesIO
3 from cryptography.hazmat.backends import default_backend
4 from cryptography.hazmat.primitives import padding
5 from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
6 from PIL import Image
7 import PIL
8
9 n = 16
10
11 key_dictionary = {}
12
13 for i in range(1, n):
14     key = os.urandom(n)
15     index = f'{i}'
16     key_dictionary[index] = key
17
18 def encrypt(key, plaintext):
19     # Generate a random 128-bit IV.
20     iv = os.urandom(16)
21
22     # Construct an AES-128-CBC Cipher object with the given key and a
23     # randomly generated IV.
24     encryptor = Cipher(
25         algorithms.AES(key),
26         modes.CBC(iv),
27         backend=default_backend()
28     ).encryptor()
29
30     # Encrypt the plaintext and get the associated ciphertext.
31     ciphertext = encryptor.update(plaintext) + encryptor.finalize()
32
33     return (iv, ciphertext)
34
35 def decrypt(key, iv, ciphertext):
36     # Construct a Cipher object, with the key, iv
37     decryptor = Cipher(
38         algorithms.AES(key),
39         modes.CBC(iv),
40         backend=default_backend()
41     ).decryptor()
42
```

```

43     # Decryption gets us the plaintext.
44     return decryptor.update(ciphertext) + decryptor.finalize()
45
46 # Function to generate the cover set given a compromised node.
47 def generate_cover_set(node, n):
48     cover_set = []
49     while node > 1:
50         sibling = node + 1 if node % 2 == 0 else node - 1
51         cover_set.append(sibling)
52         node = node // 2
53     return cover_set
54
55 # Encrypt excluding the revoked devices
56 def encrypt_with_revocation(message, revoked_devices):
57     # Generate a random key k.
58     k = os.urandom(16)
59     # Generate cover set for revoked devices.
60     cover_set_duplicates = []
61     # Iterate over all the revoked devices.
62     for device in revoked_devices:
63         values_coverset = generate_cover_set(device, n)
64         for value in values_coverset:
65             cover_set_duplicates.append(value)
66     cover_set = []
67     # Remove duplicated nodes.
68     for node in cover_set_duplicates:
69         if node not in cover_set:
70             cover_set.append(node)
71     # Encrypt key k with the devices keys of the cover set.
72     key_encryptions = []
73     for node in cover_set:
74         c = encrypt(key_dictionary[str(node)], k)
75         key_encryptions.append(c)
76     # Encrypt the content with k.
77     content_encryption = encrypt(k, message)
78
79     return (key_encryptions, content_encryption)
80
81 def decrypt_with_revocation(encrypted_keys, content_encryption, device):
82     # Iterate over the cipher keys to find one that matches with the device's key path.
83     for k_iv, k_ciphertext in encrypted_keys:
84         # It's used to calculate if some key of the parents of the node were used to
85         # encrypt.
86         i = device
87         key_root = None
88         # If i = 0 means that there is no more parents nodes. It's used to try to
89         # decrypt with the parents keys of the node, iterating over them.
90         while i > 0:
91             # The decrypt can throw an error if the key is incorrect, so I catch this
92             # errors.
93             try:
94                 # Decrypt the key
95                 key_root = decrypt(key_dictionary[str(i)], k_iv, k_ciphertext)
96                 iv, ciphertext = content_encryption
97                 # Decrypt the message
98                 message = decrypt(key_root, iv, ciphertext)
99                 try:
100                     # Try to open an image with the decrypted bytes.
101                     Image.open(BytesIO(message))
102                     except PIL.UnidentifiedImageError as e:
103                         key_root = None
104                         print(f"Device cannot decrypt the content with key {i}.")
105                     except Exception as e:
106                         print(f"Decryption attempt failed with key: {i}. Error: {e}")
107                     # If key found, break
108                     if key_root is not None:
109                         break
110                     # If the key wasn't found, it proves again with node's parent.
111                     i = i // 2
112             # If key found, break
113             if key_root is not None:
114                 break

```

```

112     return message
113
114
115 device = 3
116 with open('image.jpg', 'rb') as f:
117     data = f.read()
118
119 # Pad the plaintext to a multiple of 16 bytes
120 padder = padding.PKCS7(128).padder()
121 padded_plaintext = padder.update(data) + padder.finalize()
122 revoked_devices = {10}
123 encrypted_result = encrypt_with_revocation(padded_plaintext, revoked_devices)
124
125 # Simulate device attempting to decrypt the content.
126 decrypted_result = decrypt_with_revocation(*encrypted_result, device)
127 try:
128     im = Image.open(BytesIO(decrypted_result))
129     # Display image
130     im.show()
131 except PIL.UnidentifiedImageError as e:
132     print(f"Device {device} cannot decrypt the content because the device key was
    compromised.")

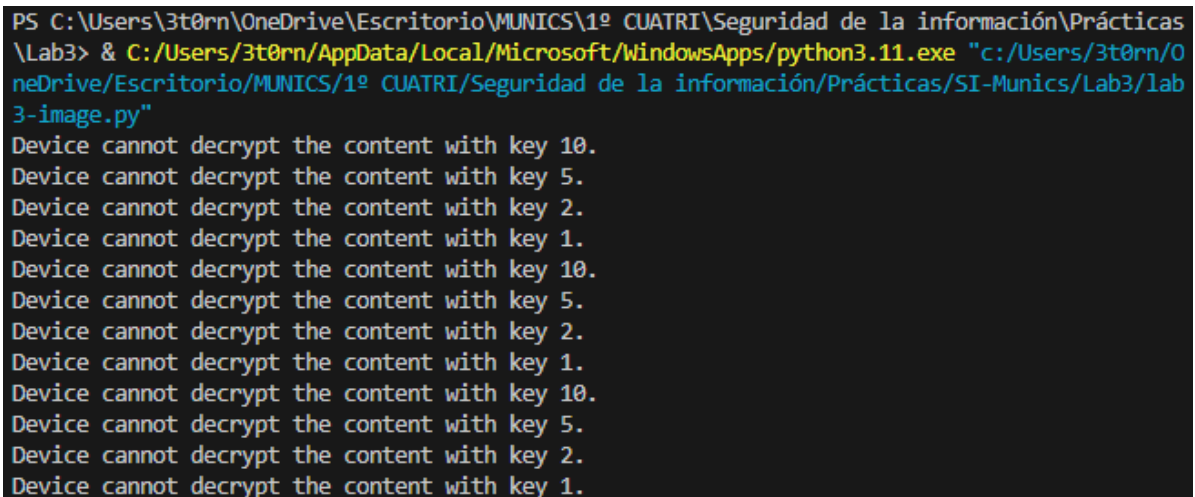
```

Listing 1: functions.py

Tests.

Note: If you want to perform different test, you only have to modify the "device" and "revoked-devices" variables, the program works always with 15 nodes, just like the statement.

Firstable, I will test the same example as in the statement, note that in this implementation the root node is 1 not 15. I know that the device 10 (3 in the statement) was compromised, so, after encrypting and calculating the cover set, the devices 10, 5, 2 and 1 will not able to open the image. I'll only test with the device 10 because in the implementation, it iterates over his parents nodes to determine if they can decrypt the key. With each key, it proves if it can decrypt one of the 3 ciphers that return the output of the encryption, each one corresponding to a device on the cover set [1](#).



```

PS C:\Users\3t0rn\OneDrive\Escritorio\MUNICS\1º CUATRI\Seguridad de la información\Prácticas\Lab3> & C:/Users/3t0rn/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/3t0rn/OneDrive/Escritorio/MUNICS/1º CUATRI/Seguridad de la información/Prácticas/SI-Munics/Lab3/lab3-image.py"
Device cannot decrypt the content with key 10.
Device cannot decrypt the content with key 5.
Device cannot decrypt the content with key 2.
Device cannot decrypt the content with key 1.
Device cannot decrypt the content with key 10.
Device cannot decrypt the content with key 5.
Device cannot decrypt the content with key 2.
Device cannot decrypt the content with key 1.
Device cannot decrypt the content with key 10.
Device cannot decrypt the content with key 5.
Device cannot decrypt the content with key 2.
Device cannot decrypt the content with key 1.

```

Figure 1: Test revoked.

Now, I'll prove with no-attacked nodes, like 14, 11 and 8 for example. Obviously, they can open the image [2](#).

```

PS C:\Users\3t0rn\OneDrive\Escritorio\MUNICS\1º CUATRI\Seguridad de la información\Prácticas
\Lab3> & C:/Users/3t0rn/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/3t0rn/O
neDrive/Escritorio/MUNICS/1º CUATRI/Seguridad de la información/Prácticas/SI-Munics/Lab3/lab
3-image.py"
Device cannot decrypt the content with key 14.
Device cannot decrypt the content with key 7.
Device cannot decrypt the content with key 3.
Device cannot decrypt the content with key 1.
Device cannot decrypt the content with key 14.
Device cannot decrypt the content with key 7.
Device cannot decrypt the content with key 3.
Device cannot decrypt the content with key 1.
Device cannot decrypt the content with key 14.
Device cannot decrypt the content with key 7.
PS C:\Users\3t0rn\OneDrive\Escritorio\MUNICS\1º CUATRI\Seguridad de la información\Prácticas
\Lab3> & C:/Users/3t0rn/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/3t0rn/O
neDrive/Escritorio/MUNICS/1º CUATRI/Seguridad de la información/Prácticas/SI-Munics/Lab3/lab
3-image.py"
Case device 11, key found in the first cipher with the key of the device 11.
PS C:\Users\3t0rn\OneDrive\Escritorio\MUNICS\1º CUATRI\Seguridad de la información\Prácticas
\Lab3> & C:/Users/3t0rn/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/3t0rn/O
neDrive/Escritorio/MUNICS/1º CUATRI/Seguridad de la información/Prácticas/SI-Munics/Lab3/lab
3-image.py"
Device cannot decrypt the content with key 8.
Device cannot decrypt the content with key 4.
Device cannot decrypt the content with key 2.
Device cannot decrypt the content with key 1.
Device cannot decrypt the content with key 8.
Case device 8, it founds the key in
the 2nd cipher and with the key of
the device 4
PS C:\Users\3t0rn\OneDrive\Escritorio\MUNICS\1º CUATRI\Seguridad de la información\Prácticas
\Lab3> 

```

Figure 2: Test successfull.

I'll make a final test just to prove another case, but you can verify that every case works successfully. Revoked= 2 and Device = 8, 10, 13, 15. With this, the whole left part of the tree will be revoked, so the 2 first cases will not work and the other 2 will open de image properly [3](#).

```

PS C:\Users\3t0rn\OneDrive\Escritorio\MUNICS\1º CUATRI\Seguridad de la información\Prácticas
\Lab3> & C:/Users/3t0rn/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/3t0rn/O
neDrive/Escritorio/MUNICS/1º CUATRI/Seguridad de la información/Prácticas/SI-Munics/Lab3/lab
3-image.py"
Device cannot decrypt the content with key 8.
Device cannot decrypt the content with key 4.
Device cannot decrypt the content with key 2.
Device cannot decrypt the content with key 1.
Device 8 cannot decrypt the content because the device key was compromised.
PS C:\Users\3t0rn\OneDrive\Escritorio\MUNICS\1º CUATRI\Seguridad de la información\Prácticas
\Lab3> & C:/Users/3t0rn/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/3t0rn/O
neDrive/Escritorio/MUNICS/1º CUATRI/Seguridad de la información/Prácticas/SI-Munics/Lab3/lab
3-image.py"
Device cannot decrypt the content with key 10.
Device cannot decrypt the content with key 5.
Device cannot decrypt the content with key 2.
Device cannot decrypt the content with key 1.
Device 10 cannot decrypt the content because the device key was compromised.
PS C:\Users\3t0rn\OneDrive\Escritorio\MUNICS\1º CUATRI\Seguridad de la información\Prácticas
\Lab3> & C:/Users/3t0rn/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/3t0rn/O
neDrive/Escritorio/MUNICS/1º CUATRI/Seguridad de la información/Prácticas/SI-Munics/Lab3/lab
3-image.py"
Device cannot decrypt the content with key 13.
Device cannot decrypt the content with key 6.
PS C:\Users\3t0rn\OneDrive\Escritorio\MUNICS\1º CUATRI\Seguridad de la información\Prácticas
\Lab3> & C:/Users/3t0rn/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/3t0rn/O
neDrive/Escritorio/MUNICS/1º CUATRI/Seguridad de la información/Prácticas/SI-Munics/Lab3/lab
3-image.py"
Device cannot decrypt the content with key 15.
Device cannot decrypt the content with key 7.

```

Figure 3: Other test