Cryptology I (spring 2020) Dominique Unruh

Exercise Sheet 3

Out: 2020-03-11 Due: 2020-03-22

Problem 1: "Inverse" CBC

Consider the following mode of operation (which I call "inverse CBC"):

To encrypt a message m consisting of blocks $m_1, \ldots, m_n$ with key k, pick a random initialization vector iv and then compute $c_1 := E_0(k, m_1) \oplus iv$ and $c_i := E_0(k, m_i) \oplus m_{i-1}$ for $i = 2, \ldots, n$. Here E_0 is the block cipher. And $E(k, m) := iv \| c_1 \| \ldots \| c_n$.

The adversary has intercepted a ciphertext $c = E(k, m)$. He happens to know the last block m_n of m (e.g., because that one is prescribed by the protocol).

(a) Explain how the adversary can completely decrypt m. He can make chosen plaintext queries (i.e., he can ask for encryptions of arbitrary message m'). He cannot make decryption queries.

Hint: First think how you can, e.g., find out $E_0(k, m_n)$ by performing an encryption query $E(k, m_n)$.

(b) Suggest how to fix the mode of operation so that it becomes secure at least against this attack (and simple modifications thereof). You do not need to prove security.

Problem 2: Breaking ECB

In the lecture we have seen that encrypting a file with ECB mode is not very secure. For example, if an uncompressed image file is encrypted, the result may still reveal much of the picture to the naked eye.

In this exercise, we consider the task of distinguishing the encryption of two given messages m_0, m_1 automatically. That is, assume that two messages m_0, m_1 (English texts) of the same length are given and known to the adversary. Furthermore, the adversary learns c, which is the ECB encryption of m_0 or m_1 (using a random and unknown key k). The adversary is now supposed to guess which message was encrypted. (I.e., we have a known plaintext attack, not a chosen plaintext attack.)

(a) Describe an algorithm that finds out (given m_0, m_1, c) whether m_0 or m_1 was encrypted. It should work on "typical" text files. (That is, it should not require, e.g., one of the text files to contain only spaces or similar.)

Example of "typical" text files are `ecb-distinguish-1.txt` and `ecb-distinguish-2.txt` from the lecture webpage.

(b) (Bonus points) Implement the algorithm. That is, fill in the missing code for the function `adv` in the code below (also available on the lecture webpage):

```
#!/usr/bin/python3

# "Crypto" might need "pip install pycrypto" if it's not installed

import Crypto, random
from Crypto.Cipher import AES

def int_to_bytes(i,len): # Not optimized
    res = []
    for j in range(len):
        res.append(i%256)
        i = i>>8
    return bytes(res)

def aes_ecb_enc(k,m):
    from Crypto import Random
    assert isinstance(m,bytes)
    assert len(m)%AES.block_size == 0, len(m)%AES.block_size
    k = int_to_bytes(k,AES.block_size)
    cipher = AES.new(k, AES.MODE_ECB)
    return cipher.encrypt(m)

def aes_ecb_dec(k,m):
    from Crypto import Random
    assert isinstance(m,bytes)
    k = int_to_bytes(k,AES.block_size)
    cipher = AES.new(k, AES.MODE_ECB)
    return cipher.decrypt(m)

# Just a test
assert aes_ecb_dec(2123414234,aes_ecb_enc(2123414234,b'hello there test')) == b'hello t


# The game: it gets a prg and an adversary as arguments,
# as well as the messages to be distinguished
def guessing_game(adv,m0,m1):
    b = random.randint(0,1) # Random bit
    k = random.getrandbits(256) # Random AES key
    seed = random.randint(0,2**32-1) # Random seed
    rand = [random.randint(0,2**32-1) for i in range(10)] # Truly random output
```

```
    msg = (m0,m1)[b]
    ciph = aes_ecb_enc(k,msg)
    badv = adv(ciph)
    return b==badv


def adv(ciph):
    # blocks contains the ciphertext as a list of blocks
    blocks = [ciph[i*AES.block_size:(i+1)*AES.block_size] for i in range(len(ciph)//AES

    ???
    return ??? # return 0 or 1


def test_adv(adv):
    num_true = 0
    num_tries = 3000
    m0 = open("ecb-distinguish-1.txt","rb").read()
    m1 = open("ecb-distinguish-2.txt","rb").read()
    for i in range(num_tries):
        #if i%100==0: print(str(i)+"...")
        if guessing_game(adv,m0,m1): num_true += 1
    ratio = float(num_true)/num_tries
    print(ratio)

# An output near 0.5 means no attack
# An output neat 0.0 or 1.0 means a successful attack
test_adv(adv)
```