Lab 03 - Cryptography

Objectives

- Basic Cryptography
- Block Cipher Modes
- RSA

Resources

- Block Cipher Mode of Operation [https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation]
- Chinese remainder theorem - Broadcast attack [http://www.di-mgt.com.au/crt.html]
- Fermat factorisation [https://en.wikipedia.org/wiki/Fermat%27s_factorization_method]
- Factor DB [http://factordb.com/]

Overview

Symmetric Key Encryption

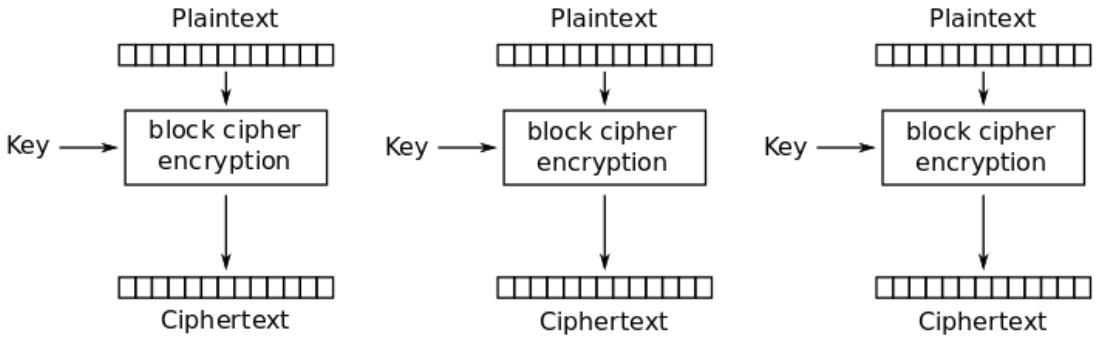
Symmetric-key encryption is based on either stream ciphers or block ciphers.

- **Stream ciphers** consume the message one bit at a time
  - *Examples*: the legendary one-time pad (though unfeasible for most applications); RC4 (deprecated); most block ciphers can be turned into stream ciphers using special modes of operation (read on).
- **Block ciphers** operate on fixed-length blocks
  - *Examples*: **Data Encryption Standard** (DES) - deprecated, **AES (Advanced Encryption Standard)** - designed to be efficient in both hardware and software, and supports a block length of 128 bits and key lengths of 128, 192, and 256 bits.

Block ciphers have one or more block size(s) but, during transformation, the block size is always fixed. Block cipher modes operate on whole blocks and require that the last part of the data be padded to a full block if it is smaller than the current block size. There are, however, modes that do not require padding because they effectively use a block cipher as a stream cipher; such ciphers are capable of encrypting arbitrarily long sequences of bytes or bits.

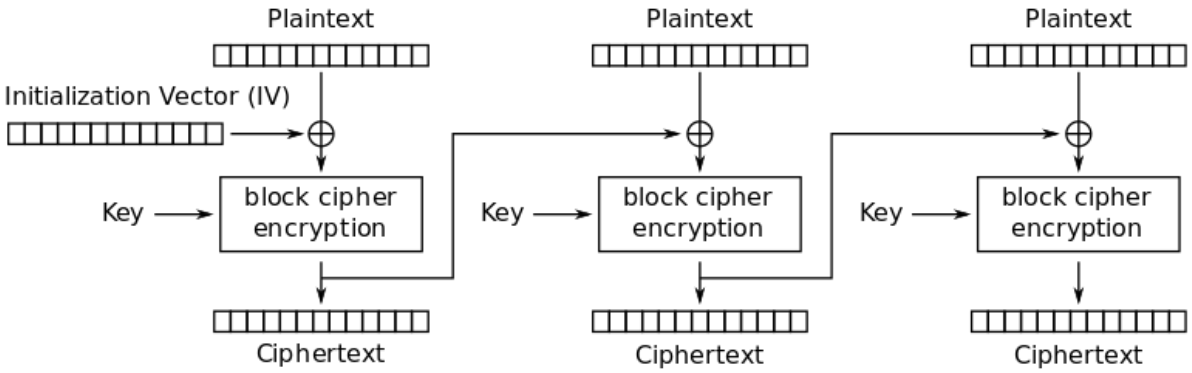
Most block cipher modes require a unique binary sequence, often called an initialization vector (IV), for each encryption operation. The IV has to be non-repeating and, for some modes, random as well. The initialization vector is used to ensure distinct ciphertexts are produced even when the same plaintext is encrypted multiple times independently with the same key.

The simplest of the encryption modes is the ECB (Electronic Codebook) mode. The message is divided into blocks, and each block is encrypted separately (with the same key and no IV). The disadvantage of this method is that identical plaintext blocks are encrypted into identical ciphertext blocks, so statistical analysis can reveal the protected message:



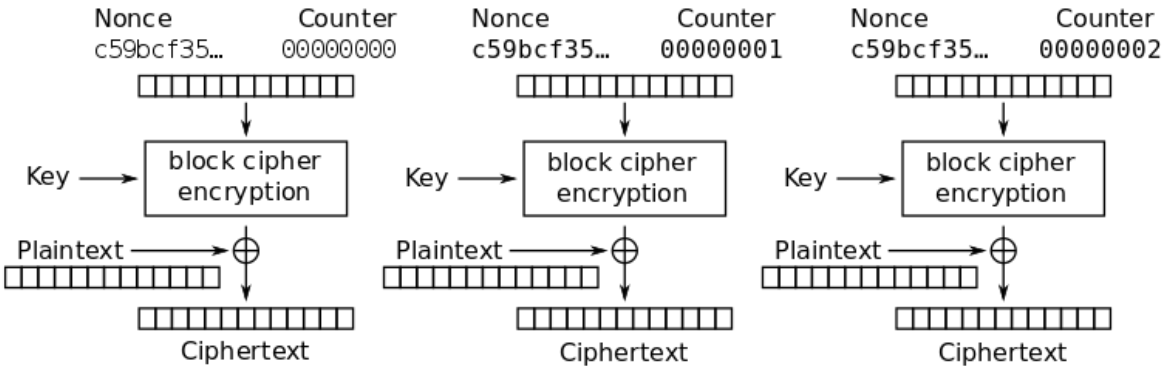
Electronic Codebook (ECB) mode encryption

In CBC (Cipher Block Chaining) mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, a random initialization vector must be used in for first block.



Cipher Block Chaining (CBC) mode encryption

There is also a CTR (Counter) mode that specifically turns the block algorithm into a stream cipher: a keystream is generated by encrypting a random IV (i.e. nonce) with the key for the first block and, for successive fragments, the IV is incremented (any deterministic function can be used) and encrypted with the same key. The plaintext is XOR-ed with the keystream (basic stream cipher operation) thus the ciphertext is obtained.



Counter (CTR) mode encryption

Asymmetric Encryption, RSA

Public-key cryptography uses two separate keys, one for encryption (the public key) and one for decryption (the private key). Anyone with the public key can compute an encrypted message that only the owner of the private key can read.

RSA was the first algorithm that demonstrated this concept. Its security assumptions are based on complexity theory: computing the product of two prime numbers is easy (polynomial time), but there is no efficient algorithm for factoring them back (so far, all factorization methods are in the non-polynomial class).

The keys for the RSA algorithm are generated the following way:

1. Choose two different large random prime numbers ***p*** and ***q***
2. Calculate ***n = pq***
   - ***n*** is the modulus for the public key and the private keys
3. Calculate the totient: ***phi (n)=(p-1)(q-1)***.
4. Choose an integer ***e*** such that 1 < ***e*** < ***phi(n)***, and ***e*** is coprime to ***phi(n)***
   - ***e*** is released as the public key exponent
5. Compute ***d*** to satisfy the congruence relation ***d*e ≡ 1 (mod(phi(n)))***
   - ***d*** is kept as the private key exponent

The public key is made of the modulus ***n*** and the public (or encryption) exponent ***e***.

The private key is made of the modulus ***n*** and the private (or decryption) exponent ***d*** which must be kept secret.

Encrypting a message: ***c = m ^ e (mod n)***

Decrypting a message: ***m = c ^ d (mod n)***

Example:

```
p = 29, q = 31
n = p * q = 29 * 31 = 899
phi = (p -1) * (q – 1) = (29 – 1) * (31 – 1) = 840
e = 11
d * e ≡ 1 mod phi => (d * 11) / phi will give us a remainder of one.
(611 * 11) = 6721 and 6721 / 840 = 8 with remainder 1 => d = 611
C = M ^ e mod n
C = 119 ^ 11 mod 899 = 595
M = C ^ d mod n
M = 595 ^ 611 mod 899 = 119
```

In this case "^" means exponentiation (raise to power), **not XOR**

Recording

Please record your screen using asciinema [https://www.mankier.com/1/asciinema] while working.

```
# install asciinema
$ sudo apt update -y && sudo apt install -y asciinema

# start recording
# NOTE: use --append if you pause your work and resume it later
#       it helps us if you upload a single .cast
$ asciinema rec [--append] lab03_${LDAP_ID}.cast

# IMPORTANT: before you start, echo your name in the terminal
$ echo "Andrei Popescu"

# work...

# stop recording
$ exit

# upload recording
$ ASCIINEMA_API_URL=https://asciinema.cs.pub.ro asciinema upload lab03_${LDAP_ID}.cast
```

When you finish your work, submit the details on the form [https://forms.office.com/r/GZzRJVqQuy]. Double check the form responses [https://ctipub-my.sharepoint.com/:x:/g/personal/mihai_chiroiu_upb_ro/Ee4pZApRKA5Iq9JgR2r652QB0FVL4J9EFtTBtva3jX-1Lw?e=ylQ7lu] so there aren't any surprises.

Exercises

0. [5p] AES ECB (Warmup)

It is recommended NOT to encrypt more than one block with AES in ECB mode, but in order to understand why, an image with the following header was encrypted. The encrypted photo cand be found here [https://drive.google.com/open?id=0B_sFoQMHYxA4TUFNY1JkQzZsdFU]. Is it possible to figure out what the initial image was?

```
42 4D 66 CA D7 00 00 00 00 00 36 00 00 00 28 00 00 00 D0 07 00 00 35 09 00 00 01 00 18 00 00 00 00 00 30 CA D7 00 74 12 00 00 74 12 00 00 00 00 00 00 00 00 00 00
```

- **Hint:** The image seems corrupted. Why? The header is encrypted, of course! Fix this and you can see how an encrypted image looks like!
- You can use your favorite hex editor for modifying the binary file (bless. Try Python [http://www.devdungeon.com/content/working-binary-data-python] or Basic shell scripting [http://stackoverflow.com/questions/4411014/how-to-get-only-the-first-ten-bytes-of-a-binary-file] if you don't have any!
- You can use bless editor

  ```
  sudo apt-get install bless
  ```

1. [20p] AES

This file [https://drive.google.com/open?id=0B_sFoQMHYxA4NzVCMEFUM3JxMXc] was encrypted using the following code. Can you decrypt it?

```
from Crypto.Cipher import AES
from Crypto import Random

BLOCK_SIZE = 32
PADDING = '#'
iv = "\x00" * 16

def encrypt(key, iv, data):
    aes = AES.new(key, AES.MODE_CBC, iv)
    data = aes.encrypt(data)
    return data

def pad(s):
    return s + (BLOCK_SIZE - len(s) % BLOCK_SIZE) * PADDING


key = Random.new().read(BLOCK_SIZE)
```

```
with open('plain.jpg', 'rb') as f:
    data = f.read()

enc = encrypt(key, iv, pad(data))

f_out = open("secret.enc", 'wb')
f_out.write(key)
f_out.write(enc)
f_out.close()
```

2. [20p] RSA - Known factorisation

In order to decrypt the ciphertext, you need to factorize n into p and q, compute phi and find d.

```
c = 288223652035779295361840391258706384406923161007725836578179393490515464731 85
n = 707360252392652399763150886901745940216466548816264214610090894808706334009 73
e = 3
```

- **Hint:** Check out FactorDB [http://factordb.com]!
- Use Python to do the math: try the gmpy2 [https://gmpy2.readthedocs.io/en/latest/overview.html] module:

```
sudo apt-get install python3-gmpy2  # or python-gmpy2, depending on your Python version
```

- **Note**: the result is a decimal number, you need to convert it to an ASCII text using the snippet below:

```
print(hex(message)[2:].decode("hex"))  # python 2
print(bytearray.fromhex(hex(message)[2:]))  # python 3
```

Useful gmpy2 functions:

- `invert(x, m)` - returns y such that x * y == 1 modulo m, or 0 if no such y exists.

Please download Task Archive [https://drive.google.com/file/d/1suqUuzJbz10Ane7icWWftILnatQ2d4Ww/view?fbclid=IwAR3bQ4dqqU8iHk_L9yNYdzDH1qSwlf10SoWxQWWNWFlzSNwwyBDM5-4AXd8] for the following tasks.

03 [15p] Is this even OTP?

- Someone applied one time pad [https://en.wikipedia.org/wiki/One-time_pad] on a text.
- However, he failed to understand that you should not use the same byte for the entire key.
    - **Hint:** bruteforce

04 [20p] Many Time Pad

- This time, he did use a proper key. Unfortunately for him, he used it for multiple encryptions.
- Knowing that the key starts with "ALEXCTF{", can you determine the rest?
    - **Hint:** take a close look at the folder for this task.

05 [20p] We want Nudes instead of Nukes

- Donald has gone completely crazy. To prevent world chaos, you kidnapped him. Right before the kidnapping he tried to send one encrypted message to his wife Melania. Luckily you intercepted the message. Donald admits that he used AES256-CBC encryption - a block cipher operating with a block length of 16 bytes.
- The IV that he used is "7ec00bc6fd663984c1b6c6fd95ceeef1" (hex encoded). After torturing him by stealing his hairpiece, he tells you the plain text of the message is: "FIRE_NUKES_MELA!".
- As a passionate hacker you of course try to take advantage of this message. To get the flag alter the IV such that Melania will read: "SEND_NUDES_MELA!".
    - **Hint 1:** The encrypted message and the key are not relevant. You will not break AES today. Look at the IV and the plaintext.
    - **Hint 2:** How does CBC [https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Block_Chaining_(CBC)] work exactly? Take a look at the decryption process and remember that the message is only one block in length.
    - **Hint 3:** Run the given oracle with the altered IV (hex encoded) to check that the message was modified correctly.

6. [10p] Feedback

Please take a minute to fill in the feedback form [https://docs.google.com/forms/d/e/1FAIpQLSeMrKoWY6UKe1N_BASUARA-HixTuvSfrEnx_FKstT-RW464NQ/viewform] for this lab.

isc/labs/03.txt · Last modified: 2021/03/23 14:28 by ionut_mugurel.anghel