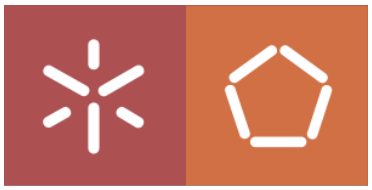


Universidade do Minho
Escola de Engenharia

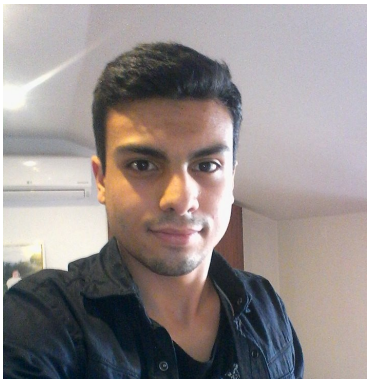
Universidade do Minho
Mestrado Integrado em Engenharia Informática

Tecnologia Criptográfica

Trabalho prático III

20 Novembro 2020

Ana Margarida Campos
(A85166)



Nuno Pereira
(PG42846)

Contents

List of Figures

Introdução

1.1 Contextualização

O presente relatório foi elaborado no âmbito do terceiro Trabalho Prático da Unidade Curricular de Tecnologia Criptográfica, que se insere no 1º semestre do 4º ano do Mestrado Integrado em Engenharia Informática (1ºano MEI).

1.2 Objetivos e Trabalho Proposto

Para este trabalho prático foi proposto a cifragem de uma imagem utilizando o AES no modo **ECB** e posteriormente no modo **CBC** ou **CTR** de maneira a incidir sobre o facto de a segurança poder ser quebrada se uma cifra for utilizada em conjunto com um modo inseguro.

O objetivo principal centra-se em demonstrar que o modo **ECB** é mais inseguro que os dois outros modos através da visualização da imagem cifrada.

1.3 Estrutura do Relatório

Inicialmente é apresentada uma breve descrição dos modos **ECB** e **CBC** seguida de uma análise ao código realizado em *python* e a visualização das imagens cifradas. Depois é feita uma conclusão sobre o trabalho desenvolvido. Por último, nos anexos encontra-se todo o código realizado.

Modos De Operação *Block-Cipher*

Os modos de operação *Block-Cipher* fornecem uma maneira de cifrar mensagens de tamanho arbitrário usando textos cifrados pequenos. Trabalham com blocos de tamanho fixo em que todas as mensagens a serem cifradas têm de ter comprimento múltiplo do tamanho do bloco. As mensagens que não são múltiplas do tamanho do bloco, podem sempre ser preenchidas de forma inequívoca para ter comprimento múltiplo.

De seguida são apresentados dois destes modos de operação.

2.1 Electronic CodeBook (ECB)

No modo **ECB** os blocos são cifrados independentemente dos outros blocos, o que significa que se ocorrer um erro na operação de cifrar um determinado bloco esse erro apenas afeta a decifragem do próprio bloco. O texto cifrado é obtido através da aplicação direta da *block-cipher* a cada bloco de texto-puro. A operação de decifrar é inversa a esta e utiliza o mesmo algoritmo deterministico. Este modo não tem indistinguabilidade uma vez que para a mesma mensagem obtemos sempre o mesmo texto cifrado (repete-se texto-puro, repete-se criptograma). Por esta razão e pela sua fácil decifragem, este algoritmo, apesar de ser o mais rápido, não deve ser utilizado.

A figura seguinte representa o esquema deste modo, onde *m1*, *m2* e *m3* são mensagens a serem cifradas, *Fk* é o bloco com tamanho fixo e chave k e *c1*, *c2* e *c3* representam os criptogramas das mensagens.

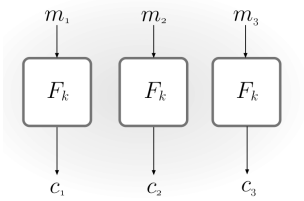


Figure 2.1: Esquema modo ECB

De modo a atingir o principal objetivo do trabalho, foi desenvolvido um programa em *python*, utilizando como auxílio a biblioteca *cryptography*, para encriptar uma imagem no modo **ECB**.

Este programa, que se encontra disponível nos anexos, inicia-se com a geração de uma chave aletória K. Após esta geração, é chamado um método da biblioteca utilizada que combina o algoritmo AES com o modo ECB. De maneira a cifrar a imagem, é necessária a sua conversão para para *bytes*. Para tal é usado *b64encode*. De forma a que o tamanho da imagem seja múltiplo do tamanho do bloco, acrescentou-se os 8 primeiros *bytes* da imagem aos *bytes* da

imagem original. Esta quantidade de *bytes* acrescentados não precisava de ser necessariamente *bytes* pertencentes à imagem original, poderiam ser por exemplo bytes nulos. A seguir é feita a conversão dos dados para hexadecimal para depois ser possível a codificação no formato *utf8*. Por fim, ocorre a descodificação dos dados utilizando *b64decode* que verifica se o *padding* está feito corretamente e, posteriormente, a escrita da imagem no formato pretendido (neste caso *bmp*). Utilizou-se o comando do enunciado para adicionar o *header*.

De seguida é possível ver a imagem original e a imagem cifrada no modo **ECB**. Como é notório este modo não é o melhor modo de cifragem uma vez que se consegue visualizar partes do formato da imagem. Isto acontece pois como foi explicado anteriormente quando se repete o texto puro, ou neste caso, um grupo de pixéis, o criptograma também é repetido.

Figure 2.2: Original

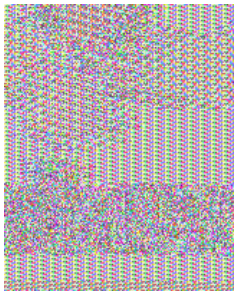


Figure 2.3: Cifrada no modo CBC

2.2 Cipher Block Chaining (CBC)

No modo **CBC** inicialmente é escolhido ou gerado aleatoriamente um vetor de incialização (IV) que é aplicado ao primeiro bloco e vai permitir que cada mensagem tenha um criptograma único. Os restantes blocos de texto cifrado são gerados aplicando a função *XOR* a cada bloco de texto puro com o bloco que foi cifrado anteriormente. Isto implica que cada bloco cifrado é dependente de todos os blocos de texto puro processados até ao momento. É um modo probabilistico o que implica ser mais difícil de decifrar. A sua principal desvantagem é a cifragem ter de ser feita sequencialmente.

A figura seguinte representa o esquema deste modo, onde *m1*, *m2* e *m3* são mensagens a serem cifradas, *Fk* é o bloco com tamanho fixo e chave *k*, *IV* é o vetor de inicialização e *c1*, *c2* e *c3* representam os criptogramas das mensagens.

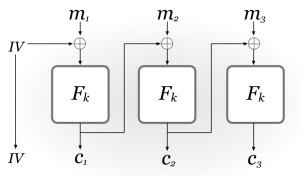


Figure 2.4: Esquema modo CBC

Aplicando os conhecimentos mencionados anteriormente, foi desenvolvido o código em *python* de maneira a cifrar a imagem no modo **CBC**.

Tal como no modo **ECB**, este programa é iniciado com a geração de uma chave aleatória *k*. A principal diferença é que neste caso também é gerado aleatoriamente o vetor de inicialização

(IV). Depois é chamada novamente a função da biblioteca *cryptography* que neste caso vai combinar o algoritmo AES com o modo CBC e o respetivo vetor de inicialização. Foi novamente necessária a passagem da imagem para *bytes* e a adição dos 8 primeiros bytes aos *bytes* da imagem original de maneira a que o tamanho fosse múltiplo do tamanho do bloco. Tal como no modo anterior, é feita a conversão dos dados para hexadecimal para depois ser possível a codificação no formato *utf8* e, posteriormente, a descodificação usando *b64decode* que verifica se o *padding* foi feito corretamente. Por fim, é escrita a imagem cifrada no modo especificado.

De seguida é possível ver a imagem original e a imagem cifrada no modo **CBC**. Como é possível verificar este modo é um bom modo de cifragem uma vez que é impossível descobir a imagem que foi cifrada olhando apenas para o criptograma.

Figure 2.5: Original

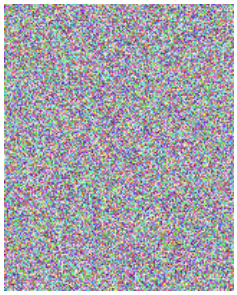


Figure 2.6: Cifrada no modo CBC

Conclusão

Com o desenvolvimento deste trabalho foram colocados em prática vários conhecimentos lecionados nas aulas, nomeadamente os modos de operação *block-cipher* **ECB** e **CBC**. Concluímos que o modo mais seguro e pelo qual se deve optar é o modo CBC uma vez que a imagem cifrada nada tem a ver com a imagem original. É completamente impossível determinar o conteúdo da imagem.

Código do Programa

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
import base64
import os


def ECB():

    key = os.urandom(32) #geração de uma key aleatório com tamanho 32 bytes
    cipher = Cipher(algorithms.AES(key), modes.ECB())  #combina algoritmo AES com modo ECB
    encryptor = cipher.encryptor()
    #passagem da imagem para bytes:
    with open("ubuntu.bmp", "rb") as imageFile:
        datas = base64.b64encode(imageFile.read())
        imageFile.close()
    #adicionados bytes para que os bytes da imagem sejam múltiplos do bloco
    #neste caso os bytes adicionados foram os 8 primeiros bytes da imagem
    ct = encryptor.update(datas+datas[:8]) + encryptor.finalize()
    #codificação em utf8
    newdatas = ct.hex().encode('utf8')
    m = base64.b64decode(newdatas)
    #escrever a imagem codificada em ecb.bmp
    fh = open("ecb.bmp", "wb")
    fh.write(m)
    fh.close()
    #adcionar o 54 bytes ao header
    os.system("dd if=ubuntu.bmp of=ecb.bmp bs=1 count=54 conv=notrunc")


def CBC():

    key = os.urandom(32) #geração de uma key aleatório com tamanho 32 bytes
    iv = os.urandom(16)  #geração de um vetor de incialização aleatório com tamnho 16 bytes
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv)) #combina algoritmo AES com modo CBC
    encryptor = cipher.encryptor()
    # passagem da imagem para bytes:
    with open("ubunto.bmp", "rb") as imageFile:
        datas = base64.b64encode(imageFile.read())
        imageFile.close()
    # adicionados bytes para que os bytes da imagem sejam múltiplos do bloco
    # neste caso os bytes adicionados foram os 8 primeiros bytes da imagem
    ct = encryptor.update(datas+datas[:8]) + encryptor.finalize()
```

```
    newdatas = ct.hex().encode('utf8')
    m = base64.b64decode(newdatas)
    fh = open("cbc.bmp", "wb")
    fh.write(m)
    fh.close()
    # adcionar o 54 bytes ao header
    os.system("dd if=ubuntu.bmp of=cbc.bmp bs=1 count=54 conv=notrunc")


if __name__ == '__main__':
    ECB()
    CBC()
```