**密码学原理**

**实验二：对称加密与认证**

学号： 2022113564 姓名： 张哲恺

**实验目的**：本实验旨在掌握运用密码学工具实现CPA安全加密与CCA安全加密，并采用CCA能力攻击CPA安全加密方案。

1. **使用密码学工具实现CPA安全加密算法**
2. 使用密码学工具实现CPA安全加密方案

要求：选择CPA安全的加密方案对一个图片内容进行加密和解密，密文文件可用图片浏览器打开。

加密时先使用get\_random\_bytes创建初始向量iv和密钥key，使用cv库读取图片并转化为字节流，填充后对图片数据使用AES\_CBC进行加密，再将密文与初始向量和填充进行拼接，再使用np.frombuffer转化回三维数组并存储在新的图像中。

解密时同样先读取图片数据，并分离初始向量iv和密文以及填充块，使用相同的密钥对密文进行解密得到明文。



原图 加密后 解密后

1. **采用CCA攻击分析CPA安全加密方案**
2. 利用CCA能力敌手攻击CPA安全的加密方案

要求：对上一步中CPA安全加密方案加密的图片文件进行CCA攻击：篡改密文图片，然后用解密预言机对篡改图片解密。

读取密文图片，随机选取100个像素点进行篡改，然后使用同样的解密预言机解密得到解密成果，发现篡改成功，CCA攻击成功。



篡改后的密文文件 解密结果



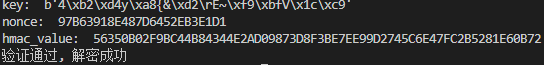
局部放大

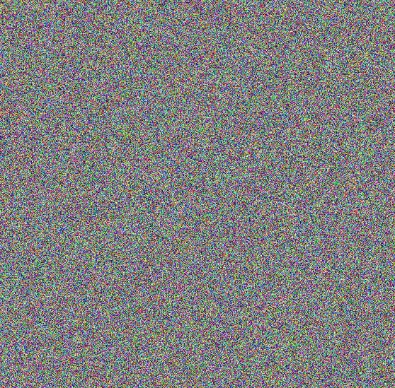
1. **使用密码学工具实现CCA安全的加密算法**
2. 使用密码学工具实现CCA安全加密方案

要求：选择CCA安全的加密方案对一个图片内容进行加密和解密，密文文件可用图片浏览器打开。

使用AES\_GCM加密方案，先随机生成长度为128的密钥，再读取bmp格式图像文件，分离文件头部分和图像数据部分。对其中的图像数据部分进行加密，并对加密结果hash得到hmac用于解密时的MAC验证。然后将nonce和hmac拼接在文件头之后，接着修改文件头，将图像数据偏移量修改为98，再将密文拼接进去，得到的整个图像数据写入密文文件中。

解密时，读取图像数据，分别获取文件头，nonce，hmac和密文，对hmac进行验证，如果密文hash结果与hmac相等，验证通过即可解密，否则不解密。



 原图 加密后 解密后

1. CCA敌手能否攻击成功CCA安全的加密方案

要求：尝试用第2步CCA攻击来攻击CCA安全加密的图片。

同样先读取密文文件，获取文件头，nonce和hmac部分 以及密文数据，随机选取100个像素点进行篡改，并使用解密预言机对篡改结果进行解密。Hmac验证不通过解密失败，CCA攻击失败。





篡改后的密文文件 解密结果

CPA.py:

1. import sys

2. import cv2

3. import numpy as np

4. from Crypto.Cipher import AES

5. from Crypto.Util.Padding import pad, unpad

6. from Crypto.Random import get\_random\_bytes

7.

8. def encrypt\_image(input\_image\_path, output\_image\_path, key , iv):

9.

10.     input\_image = cv2.imread(input\_image\_path)

11.     input\_row , input\_column , input\_depth = input\_image.shape

12.

13.     input\_image\_bytes = input\_image.tobytes()

14.

15.

16.     cipher = AES.new(key , AES.MODE\_CBC , iv)

17.

18.     image\_bytes\_padded = pad(input\_image\_bytes , 16) #填充

19.     cihpertext = cipher.encrypt(image\_bytes\_padded) #加密

20.

21.     padsize = len(image\_bytes\_padded) - len(input\_image\_bytes)

22.

23.     \_pad = input\_column \* input\_depth - 16 - padsize

24.     ciphertext\_padded = iv + cihpertext + bytes(\_pad) #拼接

25.

26.     finally\_ciphertext = np.frombuffer(ciphertext\_padded , dtype=input\_image.dtype).reshape(input\_row + 1 , input\_column , input\_depth) #转化

27.

28.     cv2.imwrite(output\_image\_path, finally\_ciphertext) #保存

29.

30. def decrypt\_image(input\_image\_path, output\_image\_path, key):

31.     encrypt\_image = cv2.imread(input\_image\_path)

32.     encrypt\_image\_bytes = encrypt\_image.tobytes()

33.

34.     encry\_row , encry\_column , encry\_depth = encrypt\_image.shape

35.

36.     iv = encrypt\_image\_bytes[:16]

37.

38.     input\_image\_size = ((encry\_row - 1) \* encry\_column \* encry\_depth)

39.     padsize = (input\_image\_size // 16 + 1) \* 16 - input\_image\_size

40.

41.     ciphertext = encrypt\_image\_bytes[16 : 16 + input\_image\_size + padsize]

42.

43.     cipher = AES.new(key , AES.MODE\_CBC , iv)

44.     plaintext = cipher.decrypt(ciphertext)

45.

46.     plaintext\_unpad = unpad(plaintext , 16)

47.

48.     plaintext\_image = np.frombuffer(plaintext\_unpad , encrypt\_image.dtype).reshape(encry\_row - 1 , encry\_column , encry\_depth)

49.

50.     cv2.imwrite(output\_image\_path, plaintext\_image)

51.

52.

53. key = get\_random\_bytes(16)

54. iv = get\_random\_bytes(16)

55. print(key)

56. # 加密图像

57. encrypt\_image('C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\input\_image.jpeg', 'C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\encrypted\_image.bmp', key , iv)

58.

59. # 解密图像

60. decrypt\_image('C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\encrypted\_image.bmp', 'C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\decrypted\_image.bmp', key)

61. #key: b'\xda\xf5\nG\xe0\xf7\tN\xfa \x10\xfe\xaa\xb4\xc6\xe1'

62.

CCA\_Attack1.py:

1. import cv2

2. import numpy as np

3. from Crypto.Cipher import AES

4. from Crypto.Util.Padding import unpad

5. from Crypto.Random import get\_random\_bytes

6. import random

7.

8. def decrypt\_image(input\_image\_path, output\_image\_path, key):

9.     encrypt\_image = cv2.imread(input\_image\_path)

10.

11.     encrypted\_image = cv2.imread(input\_image\_path)

12.     if encrypted\_image is None:

13.         print("Error: Unable to read the input\_image")

14.     else:

15.         print("Image loaded successfully")

16.

17.     encrypt\_image\_bytes = encrypt\_image.tobytes()

18.

19.     encry\_row, encry\_column, encry\_depth = encrypt\_image.shape

20.

21.     iv = encrypt\_image\_bytes[:16]

22.

23.     input\_image\_size = ((encry\_row - 1) \* encry\_column \* encry\_depth)

24.     padsize = (input\_image\_size // 16 + 1) \* 16 - input\_image\_size

25.

26.     ciphertext = encrypt\_image\_bytes[16 : 16 + input\_image\_size + padsize]

27.

28.     cipher = AES.new(key, AES.MODE\_CBC, iv)

29.     plaintext = cipher.decrypt(ciphertext)

30.

31.     plaintext\_unpad = unpad(plaintext, 16)

32.

33.     plaintext\_image = np.frombuffer(plaintext\_unpad, encrypt\_image.dtype).reshape(encry\_row - 1, encry\_column, encry\_depth)

34.

35.     cv2.imwrite(output\_image\_path, plaintext\_image)

36.

37. def perform\_CCA\_attack(input\_image\_path, output\_image\_path, key):

38.     # 加载密文图片

39.     encrypted\_image = cv2.imread(input\_image\_path)

40.

41.     # 确定篡改范围，排除初始向量部分

42.     height, width, \_ = encrypted\_image.shape

43.     tamper\_range = [(i, j) for i in range(height) for j in range(width) if not (i == 0 and j < 16)]

44.

45.     # 随机选择多个像素点进行篡改

46.     num\_pixels\_to\_tamper = min(len(tamper\_range), 100)  # 选择最多100个像素点进行篡改

47.     pixels\_to\_tamper = random.sample(tamper\_range, num\_pixels\_to\_tamper)

48.     for pixel in pixels\_to\_tamper:

49.         encrypted\_image[pixel[0], pixel[1], random.randint(0, 2)] = random.randint(0, 255)

50.

51.     print("Tampering successful")

52.

53.     # 保存篡改后的图片

54.     cv2.imwrite(output\_image\_path, encrypted\_image)

55.

56.     # 使用解密函数对篡改图片进行解密

57.     decrypt\_image(output\_image\_path, 'C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\decrypted\_image\_after\_attack.bmp', key)

58.

59. key = b'q\xe0gY\x98\xc0\xc6\xbf\x18\xfa\x1au/\xbb|\xe8'

60.

61. # 进行 CCA 攻击并尝试解密

62. perform\_CCA\_attack('C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\encrypted\_image.bmp', 'C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\tampered\_encrypted\_image.bmp', key)

63.

CCA.py:

1. from cryptography.hazmat.primitives.ciphers.aead import AESGCM

2. import numpy as np

3. from PIL import Image

4. import os

5. import hmac

6.

7. def encrypt(key) :

8.     with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\CCA\\input\_image.bmp" , 'rb') as f:

9.         input\_image = bytearray(f.read())

10.

11.     input\_image[10] = 98 #修改图片数据起始位置

12.

13.     plaintext = np.array(input\_image[55: ]) #记录图片数据

14.     #加密过程

15.     aesgcm = AESGCM(key)

16.     print("key: " ,  key)

17.     nonce = os.urandom(12)

18.     print("nonce: " , nonce.hex().upper())

19.     ciphertext = aesgcm.encrypt(nonce , plaintext.tobytes() , None)

20.     hmac\_value = hmac.new(key, ciphertext, digestmod='sha256').digest()

21.     print("hmac\_value: " , hmac\_value.hex().upper())

22.

23.     with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\CCA\\encrypted\_image.bmp" , "wb") as f :

24.         f.write(input\_image[:55]) #文件头

25.         f.write(nonce) #nonce

26.         f.write(hmac\_value) #HMAC消息

27.         f.write(ciphertext) #加密图像数据

28.

29. def decrypt(key) :

30.     aesgcm = AESGCM(key)

31.     with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\CCA\\encrypted\_image.bmp" , "rb") as f :

32.         bmphead = bytearray(f.read(55)) #获取文件头

33.         nonce = bytearray(f.read(12)) #nonce

34.         # print(nonce.hex().upper())

35.         hmac\_v = bytearray(f.read(32)) #hmac

36.         # print(hmac\_v.hex().upper())

37.         encrypted\_image = bytearray(f.read())

38.     hmac\_value = hmac.new(key , encrypted\_image , digestmod='sha256').digest()

39.     if (hmac\_v == hmac\_value) :

40.         bmphead[10] = 54 #修改图像数据起始位置

41.         ciphertext = np.array(encrypted\_image)

42.         decrypted\_img = aesgcm.decrypt(nonce , ciphertext.tobytes(), None) #解密

43.         with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\CCA\\decrypted\_image.bmp" , "wb") as f :

44.             f.write(bmphead)

45.             f.write(decrypted\_img)

46.         print("验证通过, 解密成功")

47.     else :

48.         with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\CCA\\decrypted\_image.bmp" , "wb") as f :

49.             f.write(bmphead)

50.             f.write(nonce)

51.             f.write(hmac\_v)

52.             f.write(encrypted\_image)

53.         print("验证失败，解密失败")

54.

55. key = AESGCM.generate\_key(bit\_length=128)

56. encrypt(key)

57. decrypt(key)

58.

CCA\_Attack2.py:

1. import random

2. from cryptography.hazmat.primitives.ciphers.aead import AESGCM

3. import numpy as np

4. from PIL import Image

5. import os

6. import hmac

7.

8. def modify\_image(width , height , image\_data, num\_pixels):

9.     total\_pixels = height \* width

10.

11.     pixel\_indices = random.sample(range(total\_pixels), num\_pixels)

12.     for index in pixel\_indices:

13.         row = index // width

14.         col = index % width

15.         pixel\_index = (row \* width + col) \* 3

16.

17.         red = random.randint(0, 255)

18.         green = random.randint(0, 255)

19.         blue = random.randint(0, 255)

20.

21.         image\_data[pixel\_index:pixel\_index + 3] = bytes([red, green, blue])

22.

23.         return image\_data

24.

25. def decrypt(key , input\_file , output\_file) :

26.     aesgcm = AESGCM(key)

27.     with open(input\_file , "rb") as f :

28.         bmphead = bytearray(f.read(55)) #获取文件头

29.         nonce = bytearray(f.read(12)) #nonce

30.         # print(nonce.hex().upper())

31.         hmac\_v = bytearray(f.read(32)) #hmac

32.         print("hmac: " , hmac\_v.hex().upper())

33.         encrypted\_image = bytearray(f.read())

34.     hmac\_value = hmac.new(key , encrypted\_image , digestmod='sha256').digest()

35.     print("hmac\_value: " , hmac\_value.hex().upper())

36.     if (hmac\_v == hmac\_value) :

37.         bmphead[10] = 54 #修改图像数据起始位置

38.         ciphertext = np.array(encrypted\_image)

39.         decrypted\_img = aesgcm.decrypt(nonce , ciphertext.tobytes(), None) #解密

40.         with open(output\_file , "wb") as f :

41.             f.write(bmphead)

42.             f.write(decrypted\_img)

43.         print("验证通过, 解密成功")

44.     else :

45.         with open(output\_file , "wb") as f :

46.             f.write(bmphead)

47.             f.write(nonce)

48.             f.write(hmac\_v)

49.             f.write(encrypted\_image)

50.         print("验证失败，解密失败")

51.

52. with open("C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\CCA\\encrypted\_image.bmp", "rb") as f:

53.     encrypted\_image\_data = bytearray(f.read())

54.

55. bmphead = encrypted\_image\_data[:98]

56. image\_data = encrypted\_image\_data[98:]

57.

58. image\_data = modify\_image(int.from\_bytes(bmphead[18:22], byteorder='little') , int.from\_bytes(bmphead[22:26], byteorder='little') ,image\_data, 100)

59.

60. tampered\_encrypted\_image\_data = bmphead + image\_data

61. tampered\_file = "C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\CCA\\tampered\_encrypted\_image.bmp"

62. with open(tampered\_file, "wb") as f:

63.     f.write(tampered\_encrypted\_image\_data)

64. tempered\_output\_file = "C:\\Users\\zigo\\Desktop\\crypto\\LAB\\LAB2\\CCA\\tampered\_decrypted\_image.bmp"

65.

66. key = b'4\xb2\xd4y\xa8{&\xd2\rE~\xf9\xbfV\x1c\xc9'

67. decrypt(key , tampered\_file , tempered\_output\_file)