Minilab: Block Cipher Modes of Operation

This lab is due on **February 14** at **11:59PM**, following the submission checklist below. Late submissions will be penalized according to course policy. Your writeup MUST include the following information:

1. List of collaborators (on all parts of the project, not just the writeup)
2. List of references used (online material, course nodes, textbooks, wikipedia,...)
3. Number of late days used on this assignment
4. Total number of late days used thus far in the entire semester

If any of this information is missing, at least 20% of the points for the assignment will automatically be deducted from your assignment. See also discussion on plagiarism and the collaboration policy on the course syllabus.

While we provide you with the tools to run this lab at any platform (Windows, Linux and Mac OS), I strongly recommend working at a Linux or Mac OS machine as it will make things considerably easier for you. The instructions given for the rest of the description of the lab are therefore explicitly for this case (unless otherwise noted).

Administration: This lab will be administered by Sophia Yakoubov.

Sources: The figures in this lab come from `https://www.utdallas.edu/~muratk/courses/crypto09s_files/modes.pdf`.

Objectives:

In this lab, you will investigate the strengths and weaknesses of various block cipher modes of operation.

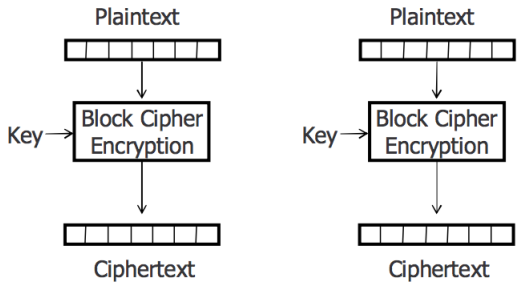

Figure 1: Electronic Code Book (ECB) Mode

1 Getting Familiar with Modes of Operation

A mode of operation is a way of using *block ciphers* like AES, which only encrypt a fixed number of bits, to encrypt arbitrary-length messages.
You can experiment with AES and modes of operation in Python.

```
THIS_MODE = AES.MODE_ECB
KEY_SIZE = AES.key_size[0]
key = Random.new().read(KEY_SIZE)
iv = Random.new().read(AES.block_size)
message = 'hello world'
cipher = AES.new(key, THIS_MODE, iv)
ciphertext = cipher.encrypt(pad(message))
```

We suggest you use the following pad and unpad functions to pad your message with 0's to the appropriate length before encrypting, and remove the zeros upon decryption:

```
def pad(s):
    """Takes in a string, and returns that string with zeros appended.
    Padding should be performed prior to encryption.
    """
    return s + b"\0" * (AES.block_size - len(s) % AES.block_size)

def unpad(s):
    """Removes trailing zeros from a string.
    Unpadding should be performed after decryption.
    """
    return s.rstrip(b"\0")
```

2 Electronic Code Book (ECB) Mode

Electronic Code Book (ECB) mode is illustrated in Figure 1. ECB mode essentially encrypts each block of the ciphertext independently.

In `tux6.ppm`, you will find an image. (PPM binary format is a simple image format where the pixels were represented as uncompressed bytes.) Encrypt that image using Electronic Code Book (ECB) mode. Encrypt the same image using Cipher Block Chaining (CBC) mode (described in Section 3). (When encrypting, remove the three header lines from `tux6.ppm`, and prepend them unencrypted to the ciphertext.)
Recall the definition of CPA security for a symmetric encryption scheme. When playing the following game with a Challenger, no efficient Adversary should be able to win with probability much more than half:

- A key k for the encryption scheme is chosen uniformly at random by the Challenger.
- The Adversary, which does not know k, is given access to an oracle that computes $Enc_k(\cdot)$ on a message m of the adversary's choice.
- The Adversary chooses two messages of equal length, m_0 and m_1, and sends them to the Challenger.
- The Challenger chooses a random bit b, produces the challenge ciphertext $c^* = Enc_k(m_b)$, and sends c^* to the adversary.
- The Adversary may continue to send any message of its choice to the the oracle that computes $Enc_k(\cdot)$.
- The Adversary outputs $b' = 0$ if it thinks that c^* was the encryption of m_0, and $b' = 1$ otherwise.
- The Adversary wins if $b' = b$.

Prove that AES in ECB mode cannot satisfy the definition of CPA-secure encryption. In other words, present an algorithm for an Adversary that wins the game described above.

What to submit

1. A file called `ECB_image.ppm`, with the image encrypted using Electronic Code Book mode.
2. A file called `CBC_image.ppm`, with the image encrypted using Cipher Block Chaining mode.
3. A file called `ECB_writeup.txt`, with a proof that AES in ECB mode is not CPA-secure.

3 Cipher Block Chaining (CBC) Mode

Cipher Block Chaining mode is illustrated in Figure 2. It XORs each ciphertext block with the next plaintext before encryption. For the first block, a random initialization vector (IV) is used instead. Note that the IV needs to be included as part of the ciphertext in order for decryption to be possible. It has been proven that CBC mode, unlike ECB mode, is CPA-secure if the block cipher encryption function is a pseudorandom permutation.

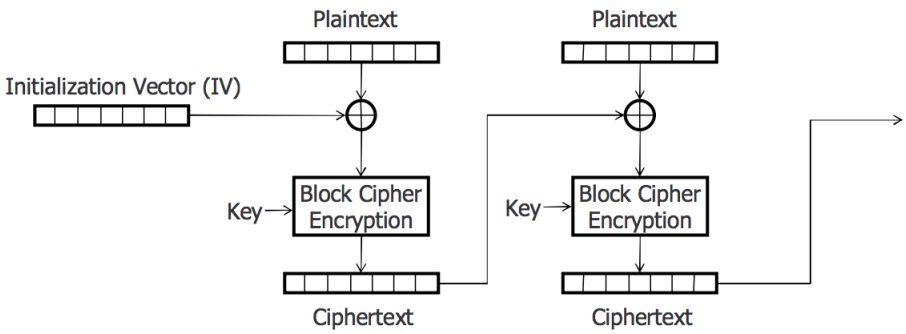

Figure 2: Cipher Block Chaining (CBC) Mode

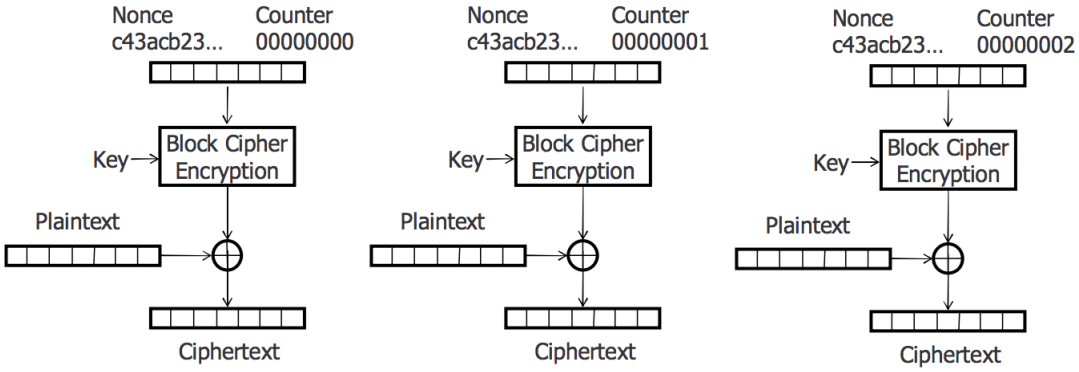

Figure 3: Counter (CTR) Mode

However, if the IVs are incremented like a counter instead of being chosen at random, this breaks down, as you will show in this section.
In `CBC_ciphertext` you will find a ciphertext. You know that it was produced using the initialization vector in `CBC_iv1`, and that the ciphertext encrypts the message in `CBC_message1`: "turn to page 01". The sender isn't using CBC mode correctly; they should be choosing random initialization vectors, but instead, they are incrementing the initialization vector like a counter, adding 1 to it for every message sent. A few messages later, they will be using the IV in `CBC_iv2`. What two messages can you specify as m_0 and m_1 in the CPA security game when it is time to use `CBC_iv2`, such that you will be able to tell the difference between the two encryptions?

What to submit

1. A file called `CBC_writeup.txt` describing m_0 and m_1, and explaining why this choice of message would allow you to break CPA-security.

4 Counter (CTR) Mode

Counter mode is illustrated in Figure 3. A random IV is chosen (as in CBC mode), and IV, $IV+1$, $IV+2$, etc are fed through the block cipher encryption function to produce a stream of random-looking bits to XOR the message with. It has been proven that counter mode is CPA-secure if the block cipher encryption function is a pseudorandom permutation. However, it is *malleable*, meaning that an adversary who sees a ciphertext can produce a different ciphertext encrypting a related message!

In `CTR_ciphertext` and `CTR_iv`, find an AES in CTR mode ciphertext and the corresponding initialization vector. You know that the ciphertext encrypts a four digit number n that is a multiple of 10. Find a ciphertext (using the same IV) encrypting $n+5$.

Next, recall the definition of CCA2 security for a symmetric encryption scheme. When playing the following game with a Challenger, no efficient Adversary should be able to win with probability much more than half:

- A key k for the encryption scheme is chosen uniformly at random by the Challenger.
- The Adversary, which does not know k, is given access to an oracle that computes $Enc_k(\cdot)$ on a message m of the Adversary's choice. The Adversary also has access to an oracle that computes $Dec_k(\cdot)$ on a ciphertext c of the Adversary's choice.
- The Adversary chooses two messages of equal length, m_0 and m_1, and sends them to the Challenger.
- The Challenger chooses a random bit b, produces the challenge ciphertext $c^* = Enc_k(m_b)$, and sends c^* to the Adversary.
- The Adversary may continue to send any message of its choice to the the oracle that computes $Enc_k(\cdot)$. It may also continue to send any ciphertext **other than** c^* to the the oracle that computes $Dec_k(\cdot)$.
- The Adversary outputs $b' = 0$ if it thinks that c^* was the encryption of m_0, and $b' = 1$ otherwise.
- The Adversary wins if $b' = b$.

Prove that AES in CTR mode cannot satisfy the definition of CCA2-secure encryption. In other words, present an algorithm for an Adversary that wins the game described above.

What to submit

1. A Python 2.x script named `CTR_solution.py` that:
   (a) Takes in two parameters:
      i. The path to a file with a ciphertext, and

ii. The path to a file with an initialization vector.

(b) Prints the ciphertext encrypting the message in the original ciphertext +5, assuming that the original ciphertext encrypts a four-digit multiple of 10.

This script should be callable as follows:

```
goldbe$ python CTR_solution.py CTR_ciphertext CTR_iv
The new ciphertext is 941b3b25eda87cac89af30f78e4cd32e.
```

2. A file called `CTR_writeup.txt` that proves that AES in CTR mode cannot satisfy the definition of CCA2-secure encryption.

5 Galois Counter Mode (GCM)

Galois counter mode is a mode of operation that offers both confidentiality and integrity; that is, it is not vulnerable to attacks like the one you executed in `CTR_solution.py`. You can read more about Galois counter mode on Wikipedia (`https://en.wikipedia.org/wiki/Galois/Counter_Mode`). You can also read a blog post explaining why GCM has recently become very popular (`https://blog.cloudflare.com/padding-oracles-and-the-decline-of-cbc-mode-ciphersuites`

6 Modes of Encryption in the Wild

In Google chrome, you can see details about the security of a website by clicking on the lock icon to the left of the URL. If the site is secure, it should say "Secure Connection - Your information (for example, passwords or credit card numbers) is private when it is sent to this site." If you click on "Details" right under that statement, you will see some additional information, like what mode of operation is being used in TLS.
Choose 3 websites that offer secure connections and use the AES block cipher. What mode of operation are they using?

What to submit

1. A file called `websites.txt` naming the websites and the mode of encryption they use.

Submission Checklist

- Section 2:
    1. `ECB_image.ppm`
    2. `CBC_image.ppm`
    3. `ECB_writeup.txt`

- Section 3:
    1. `CBC_writeup.txt`
- Section 4:
    1. `CTR_solution.py`
    2. `CTR_writeup.txt`
- Section 6:
    1. `websites.txt`