Symmetric Key Encryption Standards (AES) and Modes of Operation

A revamped version of the AES Encryption Experiment hosted by IIIT-H for VLABS (a Govt. of India Initiative)

Objective

The objective is to learn about Symmetric Key Encryption Standards by encrypting long messages using different modes of operation wherein a block cipher (in this case, AES) is provided.

Theory

- ➔ Encryption
  - ◆ Encryption can be understood as a function which takes a value and transforms it to a different value. Hence an encryption function can be understood as : $y = f(x, k)$, where y is the encrypted answer, x is the text entered for encryption and k is the key. This is the basic working of any encryption

- ➔ Modes of Encryption
  - ◆ Electronic Code Block (ECB) Mode
    - ECB mode takes the encrypted value of all the plain text given and concatenates the outputs and sends them

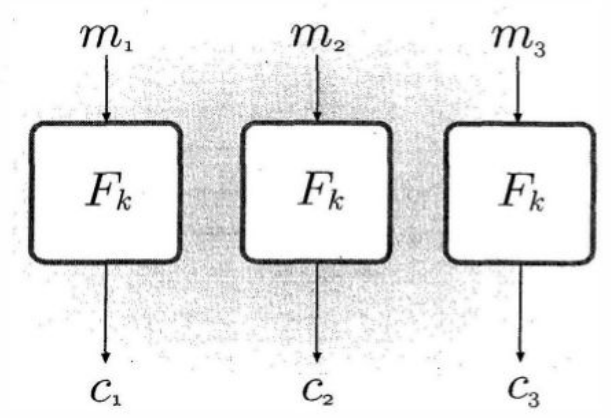

Image : ECB mode

- This mode is not very secure as the data sent can easily be manipulated without the end user knowing about it

◆ Cipher Block Chaining (CBC) Mode

- In CBC mode a random string IV is generated which creates a new cypher every time even with same key and plain text

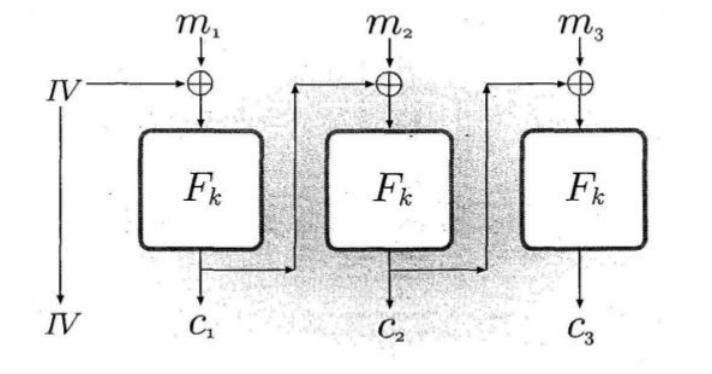

Image : CBC mode

- In CBC the input is XORed initially with the string and and then with the encrypted text of the previous data string

◆ Counter (CTR) Mode

- In Counter mode instead of the plain text being encrypted, a new random string is being encrypted and XORed with the plain text to get the result

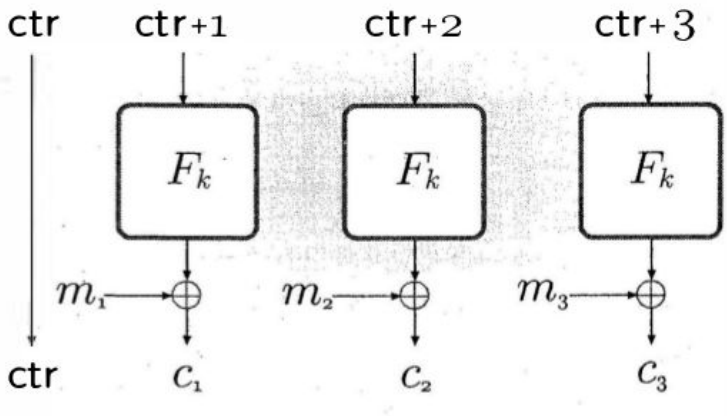

Image: Contour mode

- For i^{th} plain text block ctr + i will be encrypted and then all the strings will be concatenated

◆ Output Feedback

- In this the IV is encrypted with the output then fed back to the next encryption block , with the encrypted data XORed with the plain text

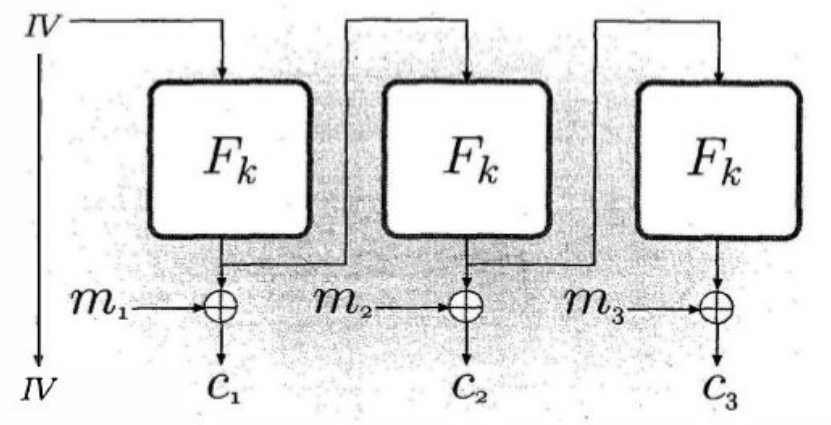

Image: Output feedback mode

- After the entire algorithm, a single block of data is output to the user to be sent

Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

Prerequisites

The following are required on your system in order to test the application

1. Python
2. Flask
3. Linux System is preferred for testing

Installation

1. Python

sudo apt-get update sudo apt-get install python3.6 Check the version of python using python3 -v. Python3.6 + is recommended.

1. Flask

pip3 install flask

Testing

The setup for a flask application is pretty simple. Just run the following command:
Python3 views.py

This should start a web server at localhost:5000/ Type this link into your browser's URL field in order to view the application.

Understand the code

The **directory structure** is pretty simple.

Overview:

Static folder contains all the CSS, JavaScript, images, and vendor applications
Template folder contains all the HTML files
The script to run the program is **views.py** - all pages are being rendered using flask. It is also responsible for sending information to the server.
It calls the functions of reqISS.py
aesLib.py is where the actual generation of strings, encryption, decryption etc takes place.
testCases.py contains unit test cases. It should be run to check the correctness of the experiment.
Quiz.db is the database which stores all responses of the user attempting the quiz

Working:

All pages have a **common header and footer** which is rendered using the Header.html and Footer.html files invokes in the other files as {% include "Header.html" %} & {% include "Footer.html" %} respectively. Only the content in the center differs which is unique to every page.

JavaScript has been written for the Experiment page :
GET requests are in **getReq.js**

Whenever a button such as key, plaintext etc is pressed, a flask.js function is invoked
An example is:

```
function nextKey() {
        $.ajax({
         type: "GET",
         url:"/experiment/nextkey",

   success: function(result){
         $('#keyarea').text(result.key);
         console.log(result.key)
         }
   });
}
```

This makes a GET request from the server and displays the result in the Key textbox.

POST requests are in **postReq.js**
An example is :

```
function XOR() {
        item ={}
        item["one"] = document.getElementById('num1').value;
        item["two"] = document.getElementById('num2').value;
        console.log(item);

        $.ajax({
```

```
        type: "POST",
        url:"/experiment/answer",
        data: JSON.stringify(item),
        contentType: 'application/json;charset=UTF-8',

        success: function(result){
        $('#xor').text(result);
        console.log(result);
        }
        });
}
```

This gets the two values entered by the user in the XOR boxes and makes a dictionary of them called ‘item’. It then makes a POST request and sends this to the server.

In views.py:

```
@app.route("/experiment/answer", methods=['GET','POST'])
def answer():
        data = request.get_json()
        one = str(data.get('one'))
        two = str(data.get('two'))

        #If the text has some spaces it won’t be taken into consideration
        oneEdit = one.split(" ")
        one = ""
        for i in oneEdit:
        one+=i
        twoEdit = two.split(" ")
        two = ""
        for i in twoEdit:
        two+=i

        xor_value = printReadable(xor(one,two),8)
        print(xor_value)
        return jsonify(xor_value)
```

This requests for the data sent by the XOR() function and performs computations on the string to return the XOR value in JSON form.

The following code in the JavaScript:

```
success: function(result){
        $('#xor').text(result);
```

Retrieves the result and displays it in the XOR answer textbox.

GET and POST requests are handled in this manner and the answers are computed and displayed on the webpage.

The actual functions that perform the encryption are in aesLib.py which are invoked in views.py

The file itself contains a class which does all the encryption and also two independent functions.

The functions are :

- XOR function
    - This functions does XOR of two HEX strings and give back corresponding output

- Print Readable
    - This functions separates the HEX string into parts of length of smaller size to make it easier to read

The methods of the class are :

- Initialization
    - This function initializes an object of the class and gives it a unique key, plaintext, an iv and a ctr

- Generate
    - They are four functions which generate a random value of the key, plaintext, an iv and a ctr by giving the appropriate method.

    - Example $genKey()$ generates a new random key

- Print
    - They are four functions which return a string of the data required

- Encrypt
    - This function takes the plaintext and encrypt it according the method given

Quiz

To get a better understanding of the experiment, try out the Quiz.
Evaluation for the quiz has been done in **quizVal.js** using JavaScript.
A database **Quiz.db** is also maintained with all the responses of the quiz. It has been written in SQLAlchemy.

Collaborators :

Shradha Sehgal : 2018101071
Kalp Shah : 2018113003