Coursework 3 - The AES Cipher

Rafael Rost

***Abstract*—In this report I have presented how to use an AES cipher library to encrypt and decrypt messages. I will provide a quickly overview about the AES cipher and then present how I implemented a tool using Ruby and the OpenSSL library.**

***Keywords*—AES, Cipher, Ruby.**

I. AES Overview

The AES (Advanced Encryption Standard) cipher is a block cipher that was created between 1997 and 2001 in order to substitute the predecessor DES cipher. The cipher passed through an evaluate period, that was started with a NIST request for proposals. The cipher had to handle 128-bit block and operate with keys size 128, 192 and 256. NIST received 15 submissions and after some conferences, they selected five finalists in 1999. Finally, after more conferences and cryptanalysis sessions in October of 2000, the Rijndael cipher was selected as the AES cipher. The big difference of this new cipher, is that instead of using the classical Feistel Network, this cipher now uses Substitution and Permutation Network.

II. AES Algorithm

AES is a cipher thats basically iterates several times based on the key size, and each round applies a simple round function. This function is composed by 3 steps: ByteSub, ShiftRow and MixColumns. For example, an AES-128 cipher iterates 10 rounds, in each round the round function is applied and the result is xored with the round key. The round output will be the input of the next round, and this is repeated until the last round. On the last round, the function that is applied is a little bit different, and omit the MixColumns step. All these steps are invertible, that means that its easy to encrypt and decrypt.

A. Round function

The input of round function is a 128-bit block that is splitted into a 4x4 array. Each of the 16 cells have 8 bits. All the steps performed inside the round function are against this 4x4 array.

- SubBytes: Substitution step. Each byte is replaced with another according to Rijndael Substitution Box.
- ShiftRows: Transposition step. Each byte in each matrix row is left cyclically shifted. The shift size change to each matrix row.
- MixColumns: Mixing step. Each matrix column is transformed based on a fixed matrix.

B. Key Expansion

Since the input of each round function is a 128-bit round key, AES cipher must expand the cipher key. This is done by the Rijndaels key schedule, that will derivate the cipher key into many round keys as the AES key sizes needs.

III. Ruby AES Implementation

Since we are implementing a tool with Ruby, lets understand how to handle cipher in the language. Ruby has a specific gem (ruby package) called OpenSSL that adds the capability of encrypt/decrypt in the most common ciphers. On a ruby environment, just add the package and you can check all the supported ciphers by performing a OpenSSL::Cipher.ciphers. You will see that AES is one of the supported algorithms. We can then instantiate a new AES class and perform some encryptions. There are many ways of instantiating Cipher class, but basically we have to specify which cipher we want and also the key size and operation mode.

- cipher = OpenSSL::Cipher::AES.new(128, :CBC)
- cipher = OpenSSL::Cipher.new('AES-128-CBC')
- cipher = OpenSSL::Cipher::AES128.new(:CBC)

All those three options will gave to us a class that is capable of encrypt and decrypt using an AES 128 cipher with CBC operation mode. After choosing the encryption algorithm we must specify which operation we want to perform by calling either cipher.encrypt or cipher.decrypt. Next step its to define a security key and IV, for both, OpenSLL has methods that generate secure random values, and we can take advantage of that.

- cipher.random_key
- cipher.random_iv

Those methods will generate and also assign the values to the class. If you want a key generated from a password you must use the OpenSSL::PKCS5.pbkdf2_hmac_sha1 or OpenSSL::PKCS5.pbkdf2_hmac methods.

At this point, we have everything configured to encrypt a plain text and after that decrypt the same text.

```
data = 'testing␣openssl'

ciph = OpenSSL::Cipher::AES.new(128, :CBC)
ciph.encrypt
key = ciph.random_key
iv = ciph.random_iv

encrypted = ciph.update(data) +
            ciph.final

dec = OpenSSL::Cipher::AES.new(128, :CBC)
dec.decrypt
dec.key = key
dec.iv = iv

plain = dec.update(encrypted) +
        dec.final
puts data == plain
```

We always must add the final decrypted/encrypted block to the text (by calling the cipher.final). This happens because the block ciphers require a finalization step that handles some type of padding with the last block being encrypted/decrypted.

IV. AES Encrypt and Decrypt

A. *Decrypt*

The main goal of this report is to present how do encrypt and decrypt using the AES cipher. Since there are no limitations around the language, I chosen Ruby and explained how Ruby handle ciphers. So, we are able to start decrypting and encrypting the tasks received as part of this coursework. In all the tasks, an AES 128 cipher was used to encrypt and the IV was prepended to the cipher text.

From tasks one to four, we received the key and cipher texts that was encoded using the CBC or CTR operation modes. So, I implemented a decrypt method that first normalize the data, by converting from hex encode to binary, that its the type of data that the cipher algorithm understand. After that, I just instantiate a cipher, filled the key and IV, and got the plaintexts. The only thing that its different from our previous ruby cipher examples, is that the IV was got from the first 16 positions of the cipher text. The code implementation looks like this:

```
def decrypt(hex_key, hex_ciphertext, mode)
  ciphertext =  hex_to_bin(hex_ciphertext)
  decipher = OpenSSL::Cipher::AES.new(128,
                                      mode)
  decipher.decrypt
  decipher.key = hex_to_bin(hex_key)
  decipher.iv = ciphertext[0..15]
  decipher.update(
  ciphertext[16..ciphertext.size])
 + decipher.final
end
```

Executing this method to all tasks will gave us the following plaintexts:

- Basic CBC mode encryption needs padding.
- Our implementation uses rand. IV
- CTR mode lets you build a stream cipher from a block cipher.
- Always avoid the two time pad!

B. *Encrypt*

Now, from the tasks 5 to 6, we received plaintexts and key, and the goal was to encrypt the messages and after decrypt them for testing our encryption implementation. The encrypt method looks very similar to the decrypt method.

```
def encrypt(hex_key, hex_plaintext, mode)
  plaintext = hex_to_bin(hex_plaintext)
  cipher = OpenSSL::Cipher::AES.new(128,
                                    mode)
  cipher.encrypt
  iv = cipher.random_iv
  cipher.iv = iv
  cipher.key = hex_to_bin(hex_key)
  encrypted = cipher.update(plaintext) +
              cipher.final
  # prependes the iv to the ciphertext
  prepended = iv + encrypted
  # hex enconde the result
  hex = bin_to_hex(prepended)
  hex.scan(/../).pack('A2' * prepended.size)
end
```

Again, data was normalized (from hex to binary), a random IV was generated and prepended to cipher text, and the output was encoded to hex again. Running the encryption for tasks 5 and 6 we ended up with two cipher texts.

```
3c7249df9e19892c92123704b88191677ef70e58f2
45f2fcb4dc783a7197bf85d4ead41e1976b3a57684
f76f20a13ab16b14be242589d210a2a5a3c21150f2
82c99c91017225d6624e1f

15ca74cce2af88217834239fe03253eadbab27c576
ff316adae246253d7575f2cb122654e2c4aae6bb31
9e9e619c684067f6d34d2664d7c6039ceb1a3c2f7b
ca1d9ad2c5ff806ebc22dad5096e111ff0ba56a5e3
82d3b3a555b3951fdcdf1fc4d0d64f2fbba7f19373
7c63786b06bf9290d9cc78162549daa8f21208038a
4e9144eccc32eb3825c7e8592eda57bee5ed
```

Now, just to make sure that our encryption is currently working, we should run the decrypt method again, passing our new generated cipher texts. By doing that we expect to have the plaintexts back.

- This is a sentence to be encrypted using AES and CTR mode.
- Next Thursday one of the best teams in the world will face a big challenge in the Libertadores da America Championship.

Although the last message is really a big lie, the encrypt method worked!

V. Conclusion

In this coursework I had the chance to understand and exercise encryption and decryption using an existing cipher library. It was a really god exercise and I took the advantage of the rich documentation that the Ruby community provides about the OpenSSL library.

Rafael Rost Software engineer at HP Inc. GitHub account: github.com/rafarost