O Avançado Padrão de Criptografia (AES)

Rafael Campos Nunes[1], Rafael Henrique Nogalha de Lima[2]
19/0098295[1] 19/0036966[2]

Contents

1 O AES

O algoritmo AES (Advanced Encryption Standard) é uma especificação de criptografia criado por Rijndael[1], sendo uma cifra de bloco de chave simétrica. Este algoritmo, ao contrário do DES (Data Encryption Standard), não utiliza uma rede Feistel e é rápido tanto em hardware quanto em software.

A ideia geral do algoritmo é separar a mensagem em blocos e cifrá-las individualmente. A depender do modo utilizado para cifragem dos blocos algumas particularidades são adicionadas aos blocos. É importante ressaltar que esses blocos podem ter tamanhos variados de 128, 192, 256 bits.

A cifragem em blocos do AES implica em uma característica interessante sobre a execução do algoritmo, ele pode ser paralelizado ou executado em concorrência já que, em alguns modos, cada bloco é cifrado independentemente do outro.

1.1 O modo ECB

Esse é o modo mais simples do AES. Basicamente, o *plaintext* é dividido em blocos, e no caso da implementação do trabalho, ele tem comprimento de 128 bits. E a cada rodada da criptografia, os dados são preenchidos até que fiquem com o mesmo comprimento do bloco. Assim, cada bloco será criptografado com a mesma chave e o mesmo algoritmo. Logo, a vulnerabilidade do algoritmo está nessa última característica apresentada, pois se criptografarmos o mesmo *plaintext*, obteremos o mesmo *cyphertext*, tendo assim a possibilidade de criptografar e descriptografar em paralelo.

1.1.1 Cifração

O modo ECB é inicializado com /key/(chave) e /round/(número de rodadas). O primeiro método a ser chamado é o **__expand_key**, nele é que ocorre o processo de expansão da chave, ele tem como entrada uma chave de 16 *bytes*, permitindo assim a construção de uma matriz linear de 176 *bytes*, o suficiente para fornecer uma chave de 16 *bytes* para o estágio **__add_round_key** no início da criptografia e cada uma das 10 rodadas seguintes.

O próximo estágio do modo é o *bytes2matrix*, nele os *bytes* são convertidos em uma matriz *4x4*. Logo depois o loop, de acordo com o número de rodadas especificado, é executado sendo que dentro dele há 4 funções. Sendo elas, *sub_bytes* em que os 16 *bytes* de entrada são substituídos de acordo com a constante *rijndael_box* fornecida pelo algoritmo; resultando assim em uma matriz *4x4*. A segunda função é **shift_rows**, nela cada uma das quatro linhas da matriz é deslocada para a esquerda, caso haja *leak* dos elementos da matriz eles serão inseridos no lado direito da linha. A mudança ocorre da seguinte forma: a primeira linha não é deslocada; a segunda linha é deslocada um (byte) para a esquerda; a terceira linha é deslocada duas posições para a esquerda; a quarta linha é deslocada três posições para a esquerda; resultando assim em uma matriz de 16 *bytes*, mas deslocados em relação um ao outro. A terceira função é **mix_columns**, nela cada coluna de 4 *bytes* é transformada usando uma função que toma como entrada os 4 *bytes* de uma coluna e gera outros 4 novos *bytes*, substituindo assim a coluna original; resultando em uma nova matriz de 16 *bytes*; essa é a única função que não é utilizada na última rodada. A última função do laço é a **add_round_key**, nela os 16 *bytes* são considerados 128 *bits* e se for a última rodada, a saída será o texto cifrado, caso contrário, o resultado de 128 *bits* são interpretados como 16 *bytes* e é novamente iniciada uma nova rodada. Após o laço, as funções de **sub_bytes**, **shift_rows** e **add_round_key** são executadas novamente e por fim a função de *encrypt* no modo ECB retorna um conjunto de bytes da imagem cifrada.

[1] Composição de nome para os dois autores originais Vincent Rijmen e Joan Daemen

1.1.2 Decifração

O processo de decifração no modo ECB é semelhante ao processo de cifração, porém, é realizado na ordem inversa. Cada rodada consiste em 4 processos conduzidos na ordem inversa: **add_round_key**, **inv_mix_columns**, **inv_shift_rows**, **inv_sub_bytes**. O retorno da função é a imagem em *bytes* decriptada.

1.2 O modo CTR

Esse modo é um dos mais complexos e completos do AES. Nele é utilizado um contador que tem o mesmo tamanho do bloco usado. No CTR, a operação XOR com o *plaintext* é realizada no bloco de saída do criptografador. Todos os blocos de criptografia usam a mesma chave de criptografia. Portanto, para garantir a maior segurança é necessário que a chave seja alterada a cada $2^{\frac{n}{2}}$ blocos de criptografia.

1.2.1 Cifração

O modo CTR possui algumas funções em comum com o modo ECB, assim como com os demais modos do modelo AES. Por isso, comentaremos somente as diferentes funções do modo ECB. Esse modo é inicializado com um parâmetro (*iv* - vetor de inicialização) a mais em relação ao modo ECB, esse vetor é o bloco "falso" para o processo de cifragem nesse modo que gera mais aleatoriedade na função. Além disso, o incremento é realizado pela função **text__inc_bytes**. Uma diferença importante entre o modo ECB e CTR é o uso de XOR, nno modo CTR, em cada bloco gerado com o auxílio do vetor de inicialização para gerar mais aleatoriedade na cifragem. Ao final é retornado um vetor de *bytes* da imagem encriptada.

1.2.2 Decifração

O processo de decifração no modo CTR é semelhante ao processo de cifração, porém, é realizado na ordem inversa. O retorno da função é a imagem em *bytes* decriptada.

2 Arquitetura do Projeto

O projeto foi escrito na linguagem Python utilizando-se de módulos para organização do código. O AES, neste projeto, é um módulo contendo dois arquivos que implementam o modo ECB e CTR. Além disso, há abstrações para manipular imagens no formato bitmap para que seja possível visualizar o resultado da cifragem e decifragem.

Para implementar a o registro da imagem em modo bitmap corretamente, é necessário extrair o cabeçalho e o corpo da imagem, cifrar o corpo e inserir o cabeçalho no corpo cifrado. Deste modo, a imagem é corretamente interpretada.

3 Resultados

Para obtenção dos resultados foi necessário utilizar cinco ciclos de criptografia e o comando para inicialização do algoritmo pode ser visto abaixo.

```
python3 src/main.py -i assets/medium.bmp -k "r@f@el" -c 5 -m ecb
python3 src/main.py -i assets/medium.bmp -k "r@f@el" -c 5 -m ctr
```

Ao executar o algoritmo no modo ECB, observou-se os cinco ciclos de cifragem e decifragem da imagem. As imagens estão organizadas abaixo, sendo incluído somente 4 para simplificação.

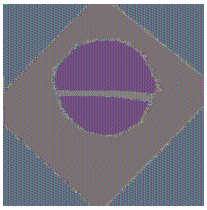

Figure 1: Ciclo 1 de AES (enc)

Figure 2: Ciclo 1 de AES (dec)

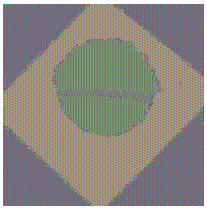

Figure 3: Ciclo 2 de AES (enc)

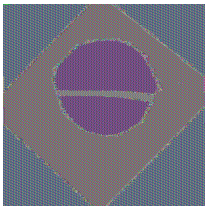

Figure 4: Ciclo 2 de AES (dec)

As imagens mosram, claramente, como os ciclos de cifragem alteram a imagem. Isto é, a imagem decifrada do ciclo dois (imagem 4) corresponde exatamente com a figura cifrada do ciclo 1 (imagem 1).

A execução do modo CTR foi realizada de modo análogo, com uma ressalva, foi possível indicar que a entropia da imagem é maior do que o modo ECB.

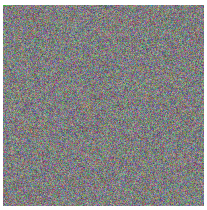

Figure 5: Imagem cifrada com CTR

Figure 6: Imagem decifrada com CTR

4 Problemas

O fator limitante na execução do algoritmo em grandes imagens foi a escolha da linguagem além de que não foi implementado qualquer tipo de mecanismo de concorrência ou paralelismo afim de aumentar a velocidade do processamento em computadores com mais núcleos.

O material na internet divergia bastante e foi necessário fazer um grande filtro e vários testes para verificar o que funcionava ou não, com respeito a manipulação de imagens.

A manipulação de imagens no formato png ou jpeg/jpg é mais complexa pois dados da imagem permeiam os dados de cor, dificultando a edição dos dados tal qual é feito de maneira simples utilizando bitmaps.

5 Configuração de Ambiente

O ambiente utilizado para construção e teste do trabalho é o GNU/Linux, com o python na versão 3.6.9. No Windows o python3 é instalado com o nome python. certifique-se de que está utilizando a versão correta com `python --version`.

Se for desejo do corretor, o trabalho também contém testes que podem ser executados afim de demonstrar a corretude do algoritmo. Para isso basta instalar a única dependência necessária (Crypto) através do pip com o seguinte comando:

```
$ pip install -r requirements.txt
```

Após instalar a dependência, execute o script `src/(ctr|ecb)_test.py`. O CTR e o ECB vão falhar de imediato pois a implementação desses na biblioteca Crypto é diferente da realizada por nós. Isso porquê existem algumas diferenças nas APIs de construção da cifra.

A título de exemplo o algoritmo de CTR é inicializado, com a biblioteca Crypto, utilizando uma função interna de contagem.

5.1 Execução e testes do programa

O programa na sua configuração atual suporta diversos tipos de parâmetros, todos listados e documentos ao executar o comando `python3 src/main.py --help`. Exemplos de utilização são listados abaixo.

```
$ python3 src/main.py -i assets/github_profile.png -k "sua chave"
$ python3 src/main.py -i assets/github_profile.png -k "sua chave"
  -m ctr
$ python3 src/main.py -i assets/github_profile.png -k "sua chave"
  -m ctr -c 5
```

Dois dos argumentos que podem confundir o usuário é o `-c` e o `-r`. O primeiro diz respeito ao número de ciclos de cifração e decifração que será aplicado ao arquivo e o segundo é o número de rodadas que o algoritmo irá executar para cifrar o arquivo.

Por fim, a execução do programa resulta em arquivos com sufixos **dec** ou **enc**, juntamente ao índice do ciclo do algoritmo na pasta em que o programa python foi iniciado.

6 Referências

1. https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-gu
   documents/aes-development/rijndael-ammended.pdf

2. https://pillow.readthedocs.io/en/stable/reference/Image.html

3. https://xilinx.github.io/Vitis_Libraries/security/2020.1/guide_L1/
   internals/ecb.html

4. https://www.intechopen.com/chapters/67728

5. https://www.gta.ufrj.br/grad/10_1/aes/index_files/Page588.htm

6. https://asecuritysite.com/subjects/chapter88

7. https://www.intechopen.com/chapters/67728