



Universidade de Aveiro  
Engenharia de Computadores e Telemática  
Linguagens Formais e Autómatos  
2019–2020, 2º Semestre

# Criptografia

Grupo 03,

Autores:

- Diogo Miguel Rocha Amaral: 93228
- Guilherme Amaral Ribeiro Pereira: 93134
- José Luís Rodrigues Costa: 92996
- João Tiago Lacerda Rainho: 92984

14 junho de 2020

# Índice

|                           |    |
|---------------------------|----|
| Índice                    | 1  |
| Introdução                | 2  |
| Documentação              |    |
| • 1º Gramática            | 3  |
| • 2º Gramática            | 7  |
| Como usar                 | 8  |
| Exemplo das gramáticas    |    |
| • 1º Gramática            | 9  |
| • 2º Gramática            | 10 |
| Comparação de resultados  | 11 |
| Contribuições dos autores | 12 |

# Introdução

Este projeto foi elaborado no âmbito da unidade curricular de Linguagens Formais e autómatos da Universidade de Aveiro do ano letivo 2019/2020. Neste projeto foi elaborado uma linguagem para criptografia com o objetivo de produzir uma linguagem que fosse mais simples a construção de cifras mais complexas.

Numa fase inicial, começamos por produzir a nossa gramática de encriptação (principal), discutir como iria ser, os objetivos de cada função, o que implementar, etc... Ao fim de definirmos uma gramática já bem constituída fizemos então a análise semântica, o compilador (String Template), alguns testes com o objetivos de replicar cifras reais (AES) e por fim a gestão de erros.

Na fase final, ao fim de concluirmos a primeira parte produzimos uma segunda gramática (secundária), com a finalidade de estipular modos de cifra como por exemplo CBC, ECB, CFB, OFB, PCBC. Ao fim de a concluirmos, elaboramos novamente os passos descritos anteriormente.

# Documentação

## 1º Gramática (Principal)

### Tipos de variáveis

| Tipos de variável | Nome |
|-------------------|------|
| Byte              | byte |
| Inteiro           | num  |
| Bit               | bit  |

| Estruturas | Exemplo                               |
|------------|---------------------------------------|
| Matrizes   | byte[2,2] nome <- 0xff,0xff 0xff,0xff |

- **Breve descrição:** Uma linguagem para descrever a operação de um algoritmo de cifra. A linguagem suportar a definição de operandos com dimensão, de forma a permitir uma validação semântica e limitar os tipos necessários na geração de código.
- **Cabeçalho:**
  1. **File Name:** Inicialização do ficheiro, contém o nome do ficheiro a gerar e o tamanho da *Data* e da *Key* em bytes, respetivamente.  
**p.ex:** Cypher Nome1 -> num1, num2;

- **Definições de variáveis:**

1. **Declare:** Declarar uma variável que nunca tenha sido criada, esta pode ser de três tipos (bit, byte, num), e matrizes de todos eles.

**p.ex:** num var <- 153;  
byte var <- 0x1F; A linguagem suportar a defini  
bit var <- 0b0;

2. **Assignment:** Atribuir a uma variável já criada um novo valor.

**p.ex:** var <- 153;  
var <- 0x1F;  
var <- 0b0;

- **Funções que não devolvem o valor:**

1. **Sub Bytes:** De acordo com o seu valor da matriz(obrigatoriamente de bytes), troca esse valor por outro, conforme uma matriz pré-feita de substituição (e.g: Rijndael S-box).

**p.ex:** Sub Bytes[id1, id2];  
Sub Bytes[id1, Get\_rijndael\_s\_box()];

2. **Shift Row:** Desloca os valores da linha para a esquerda o número de posições indicado.

**p.ex:** Shift Row[id1, 1, 2];  
Shift Row[id2, 5, 5];

3. **Shift Col:** Desloca os valores da coluna para baixo o número de posições indicado.

**p.ex:** Shift Col[id1, 1, 2];  
Shift Col[id2, 5, 5];

4. **Set Value:** Definir um valor (2ºarg), numa posição (3/4ºarg) da matrix (1ºarg).

**p.ex:** Set Value[id1, id2, 0, 0];  
Set Value[id1, 4, 5];  
Set Value[id1, Get Value from[id2, 2, 2], 6, 1];

5. **Set Matrix:** Definir uma parte da matriz (3/4º Ponto Inicial).

**p.ex:** Set Matrix[id1, id2, 2, 3];  
Set Matrix[id1, id2, id3, id4];

6. **Set Row:** Definir uma linha de uma matriz.

**p.ex:** Set Row[id1, id2, id3];  
Set Row[id1, Get Row from[id1, 4], 4];

7. **Set Col:** Definir uma coluna de uma matriz.

**p.ex:** Set Col[id1, id2, id3];  
Set Col[id1, Get Row from[id1, 4], 4];

- **Funções que devolvem um valor:**

1. **Get Value:** Obter um valor de uma matrix com 1 ponto.  
**p.ex:** Get Value from[id2, 2, 2];  
Get Value from[get\_rijndael\_s\_box(), 5, 1];  
Get Value from[Get Value from[id2, 2, 2], id1, 3];
2. **Get Matrix:** Obter uma parte de uma matriz(1ºarg) com 2 pontos.  
**p.ex:** Get Matrix from[id1, 2, 2, 4, 5];  
Get Matrix from[Get\_rcon(), id1, 2, 4, Get Value from[id2, 2, 2]];
3. **Get Row:** Obter a linha de matriz.  
**p.ex:** Get Row from[id1, 4];  
Get Row from[Get\_L(), id2];
4. **Get Col:** Obter a coluna de uma matriz.  
**p.ex:** Get Col from[id1, 4];  
Get Col from[Get\_L(), id2];
5. **Zeros:** Cria uma matriz nula, ou seja, com os valores todos a zero.  
**p.ex:** byte var[4,4] <- Zero[byte, 4, 4];  
num var[4,4] <- Zero[num, 4, 4];
6. **To Line:** Transforma a matriz passada no parâmetro de entrada para uma linha.  
**p.ex:** To Line[id1];  
To Line[Get\_rijndael\_s\_box()];
7. **To Matrix:** Transforma o tipo que está no parâmetro de entrada para uma matriz.  
**p.ex:** To Matrix[id1, 2, 1];  
To Matrix[Get\_rijndael\_s\_box(), id1, 3];
8. **To Num:** Transforma o tipo que está no parâmetro de entrada para um número.  
**p.ex:** To Num[id1];  
To Num[0xFF];  
To Num[0b1];  
To Num[Get Value from[id2, 2, 2]];
9. **To Byte:** Transforma o tipo que está no parâmetro de entrada para um byte.  
**p.ex:** To Byte[id1];  
To Byte[4];  
To Byte[0b1];  
To Byte[Get Value from[id2, 2, 2]];

10. **To Bit:** Transforma o tipo que está no parâmetro de entrada para um bit.

**p.ex:** To Bit[id1];  
To Bit[0xFF];  
To Bit[66];  
To Bit[Get Value from[id2, 2, 2]];

- **Funcionalidades básicas:** Estas funcionalidades têm o mesmo objetivo do que outras linguagens de programação.

1. **For**

**p.ex:** for(num i <- 2:5)

\$for

2. **While**

3. **p.ex:** while( i < 5)

\$while

4. **If**

**p.ex:** if( i < 5)

\$if

5. **Functions:** Nos parâmetros das funções e no valor de return, se as variáveis forem matrizes, é necessário por “-M” depois do tipo de variável. Outro aspecto importante é que se não quisermos retornar nenhum valor temos a palavra “nothing” para esse efeito.

**p.ex:** function out <- tipoDeDados\_de\_return nomeDaFuncao(parâmetros)

out <- var;

\$func

function out <- nothing funcao1(byte p1, num-M p2)

\$func

## 2º Gramática (Secundária)

### Tipos de variáveis

| Tipos de variável | Nome |
|-------------------|------|
| Byte              | byte |
| Inteiro           | num  |

| Estruturas | Exemplo                                |
|------------|--|
| Matrizes   | byte[2,2] nome <- 0xff,0xff  0xff,0xff |

| Padding usado |
|---------------|
| PKCS7         |

- **Breve descrição:** Modos de cifra, que são formas genéricas de aplicação de cifras a volumes de dados arbitrários.
- **Cabeçalho:**
  1. **File Name:** Inicialização do ficheiro, contém o nome do ficheiro a gerar e o tamanho da *Data* e da *Key* em bytes, respetivamente.  
**p.ex:** CypherMode Nome1 -> num1, num2;
  2. **Use:** Definir que algoritmo de encriptação usar  
**p.ex:** use cypher nomeCripto;
- **Definições de variáveis:**
  1. **Declare:** Declarar uma variável que nunca tenha sido criada, esta pode ser de três tipos (byte, num, matrix) e matrizes de todos estes.  
**p.ex:** num var <- 153;  
byte var <- 0x1F;



2. **Assignment**: Atribuir a uma variável já criada um novo valor.

**p.ex:** var <- 153;  
var <- 0x1F;

### ● Funcionalidades básicas:

1. **For**: Ciclo específico para a gramática, em que a variável incrementa a cada bloco de dados.

**p.ex:** increment var forEach data

\$forEach

2. **IfNotEquals**: Verifica se os valores são diferentes.

**p.ex:** ifNotEquals

\$ifNotEquals

3. **CypherFunction**: Chamar um algoritmo de encriptação, passando um argumento.

**p.ex:** num var = AES(var);

### ● Operações:

1. **xor**: Operação de xor com dois valores.

**p.ex:** 2 xor 4  
0x13 xor 0xF1

2. **Adição**: Operação de adição de dois valores.

**p.ex:** 1 + 2  
0x12 + 0xFF

## Como usar:

Para utilizar a primeira gramática é necessário a utilização da segunda gramática, de forma a correr o algoritmo com o modo de cifra associado.

Primeiro devemos compilar a primeira gramática que se encontra na pasta *Primeira Gramática*, através do antlr4-run (nome do ficheiro), e.g : antlr4-run AES.txt , este vai gerar um ficheiro python com o código gerado. Depois de gerado este código, este precisa de ser colocado na pasta da *Segunda Gramática*, gerar o código da segunda gramática que é executável através de ficheiro.py arg1 arg2 (arg3 opcional), sendo o *arg1* o nome do ficheiro a encriptar e o *arg2* o ficheiro que contém a key (no caso de AES de 16 bytes (128 bits)) e o *arg3* o nome do ficheiro a colocar a informação encriptada. Depois de executar o ficheiro python, este vai imprimir na consola cada bloco encriptado, guardando num ficheiro chamado result.txt caso não seja fornecido o terceiro argumento.

Vídeo exemplificativo com AES: <https://youtu.be/uWbZ7IPILDM>

**Nota:** A linguagem destino do compilador é o python3 e faz uso da library numpy

# Exemplo da primeira gramática:

```
Cypher AES -> 16 , 16;

func out <- byte-M keySchedule(byte-M keyMatrix,num it)
  byte newKey[4,4] <- Zeros[byte,4,4];
  byte col1[4,1] <- Get Col from [keyMatrix,3];
  Shift Col[col1,0,3];
  Sub Bytes[col1, Get_rijndael_s_box()];
  col1 <- Get Col from[keyMatrix,0] xor col1 xor Get Col from [Get_rcon(),it];
  Set Col[newKey,col1,0];
  for(num z <- 1 : 4 )
    col1 <- col1 xor Get Col from [keyMatrix,z];
    Set Col[newKey,col1,z];
  $for
  out <- newKey;
$func

func out <- byte-M shiftRows(byte-M data)
  Shift Row[data, 1, 1];
  Shift Row[data, 2, 2];
  Shift Row[data, 3, 3];
  out <- data;
$func

func out <- byte mul_GF8(byte s,byte t)
  byte ans <- 0x0;
  num ind1 <- To Num [Get Value from [Get_L(),0,To Num[s]]];
  num ind2 <- To Num [Get Value from [Get_L(),0,To Num[t]]];
  num ind <- ind1 + ind2;

  if(ind > 255)
    ind <- ind - 255;
  $if

  ans <- Get Value from [Get_E(),0,ind];

  if((s = 0x0) || (t = 0x0))
    ans <- 0x0;
  $if
  if(s = 0x01)
    ans <- t;
  $if
  if(t = 0x01)
    ans <- s;
  $if
  out <- ans;
$func
```

```

func out <- byte-M mixColumns(byte-M data,byte-M matrix)
  byte result[4,1] <- Zeros[byte,4,1];
  for(num k <- 0:4)
    result <- Zeros[byte,4,1];
    for(num s <- 0:4)
      byte colData[4,1] <- Get Col from [data,k];
      for(num t <- 0:4)
        byte new <- mul_GF8(Get Value from [matrix,s,t],Get Value from[colData,t,0]);
        byte temp <- Get Value from [result,s,0] xor new;
        Set Value[result,temp,s,0];
      $for
    $for
    Set Col[data,result,k];
  $for
  out <- data;
$func

data <- To Matrix [data,4,4];

key <- To Matrix [key,4,4];

byte matrix[4,4] <- 0x2,0x3,0x1,0x1|0x1,0x2,0x3,0x1|0x1,0x1,0x2,0x3|0x3,0x1,0x1,0x2;
data <- data xor key;
for(num i <- 0 : 9)
  Sub Bytes[data, Get_rijndael_s_box()];
  data <- shiftRows(data);
  data <- mixColumns(data,matrix);
  key <- keySchedule(key,i);
  data <- data xor key;
$for

Sub Bytes[data, Get_rijndael_s_box()];
data <- shiftRows(data);
key <- keySchedule(key,9);
data <- data xor key;

```

## Exemplo da segunda gramática:

```

CypherMode ECB -> 16 , 16;

use cypher AES;
increment p forEach data
  out[0,p] <- AES(data[0,p]);
$forEach

```

# Comparação de resultados

Para a comparação e verificação dos nossos resultados obtidos foi utilizado uma calculadora de cifras (<http://aes.online-domain-tools.com/>) .

Resultados do site:

## AES – Symmetric Ciphers Online

Input type: Text

Input text: (plain) Two One Nine Two1234567890123456

☒ Plaintext ☐ Hex Autodetect: **ON** | OFF

Function: AES

Mode: ECB (electronic codebook)

Key: (plain) Thats my Kung Fu

☒ Plaintext ☐ Hex

> Encrypt! > Decrypt!

Encrypted text:

|          |   |                               |
|----------|---|-------------------------------|
| 00000000 | 29 c3 50 5f 57 14 20 f6 40 22 99 b3 1a 02 d7 3a | ) Ã P _ W . ö @ " . ³ . . x : |
| 00000010 | 73 8a f9 90 4d 56 62 19 46 09 ac 4f 8b 4f a8 a6 | s . ù M V b . F . ~ 0 0 " !   |

Resultados do nosso programa:

```
luiscosta@luiscosta-Predator:~/CypherGenerator/Entrega/SegundaGramaticas$ python3 test.py ../ficheiros\ para\ teste\ExampleData.txt ../ficheiros\ para\ teste\ExampleKey.txt
[[0x29 0x57 0x40 0x1a]
[0xc3 0x14 0x22 0x2]
[0x50 0x20 0x99 0xd7]
[0x5f 0x16 0xb3 0x3a]]
[[0x73 0x4d 0x46 0x8b]
[0x8a 0x56 0x9 0x4f]
[0xf9 0x62 0xac 0xa8]
[0x90 0x19 0x4f 0xa6]]
[[0xb3 0xba 0xc1 0x9a]
[0xe4 0x8d 0x87 0x89]
[0xbf 0x2b 0x69 0xe8]
[0x11 0x97 0x44 0x68]]
luiscosta@luiscosta-Predator:~/CypherGenerator/Entrega/SegundaGramaticas$
```

Podemos verificar que os valores estão iguais, o que mostra um bom funcionamento do nosso programa.

# **Contribuições dos autores**

Para a conclusão deste trabalho todos os membros do grupo concordaram em estar sempre a comunicar na realização do trabalho o que resultou numa contribuição similar por parte dos membros do grupo (25% a cada).