Introdução

O algoritmo Advanced Encryption Standard, abreviado com as siglas AES, é o algoritmo mais popular usado para proteção eletrónica de dados. Desde 2006, é o algoritmo mais popular dentro dos algoritmos usados no que se chama criptografia simétrica. A criptografia simétrica também é denominada criptografia de um código, e é que o código que se usa tanto para cifrar uma mensagem como para a decifrar é o mesmo. Portanto, recetor e emissor concordam neste mesme código.

Com este artigo, pretende-se dar um esquema sobre como funciona este tipo de algoritmos, qual a matemática que o suporta, e formar uma ideia de que é o que acontece, por exemplo, quando passamos um cartão de pré-pagamento para o pagamento do bilhete do autocarro, ou como funcionam internamente os cartões pré-pagamento de energia como os que explicamos na Secção de Novidades deste número.

Em termos muito genéricos, temos um emissor e um recetor. O emissor dá uma informação (input) que é encriptada pelo algoritmo que apresentamos, o recetor recebe esta informação encriptada (output), e junto com o código secreto (pin) que é conhecido por ambos realiza o processo de desencriptação. Deste modo, foi transferida uma informação com alta segurança de que terceiros não possam ter acesso a ela. Veremos como é processada essa informação no algoritmo, como processa o código e como realiza o processo de encriptação, e em que tipo de matemática está suportado todo este jogo de encriptação. O processo de desencriptação não o veremos por ser bastante similar, mas a trabalhar de forma inversa.

1 Bytes e vetor de bytes

A unidade básica para processar no algoritmo AES é um byte: isso é uma sequência de oito bits tratado como uma unidade inteira. Um bit é um dígito do sistema binário. No sistema decimal, que é o que usualmente se usa na vida quotidiana, usamos 10 dígitos. No entanto, para o sistema binário apenas se necessitam dois dígitos, 0 e 1. A nível de placa informática, podemos imaginar o zero como desligado e o 1 como ligado. Ver um exemplo de um byte de oito bits representado na Figura 1.

Figura 1: Exemplo de um byte de 8 bits em representação animada e binária

A partir da unidade do byte podemos deﬁnir uma lista de bytes ou uma matriz de bytes como se mostra nas

Deﬁnições 1.1 e 1.2.

Deﬁnição 1.1. Uma lista de bytes é uma sequência ﬁnita de bytes que podemos denotar como se segue,

em que é um byte para

Figura 2: Exemplo de uma matriz de bytes

Deﬁnição 1.2. uma matriz de bytes é um conjunto ordenado em ﬁlas e colunas cujos elementos são bytes. Mais particularmente, diremos que A é uma matriz de m ﬁlas e n colunas e os sus elementos serão denotados como para sendo um byte.

Ver um exemplo de uma matriz quadrada de tamanho 4 na Figura 2.

1.1 Entradas(input), saídas(output) e código secreto

A entrada e a saída para o algoritmo AES consiste em um array de 16 bytes, isto perfaz um total de 128 bits. No processo interno do algoritmo, estes 128 bits serão operados como uma matriz de bytes de 4 por 4 como foi visto na Figura 2. Desta forma, o input ou entrada do algoritmo é representado como o seguinte array e o output ou saída como , em que cada e é um byte para . Internamente, a entrada e todas as modiﬁcações, que denominamos estados intermédios, que produzirá o algoritmo serão trabalhados como matrizes de bytes de tamanho 4 por 4, como se pode ver na Figura 3.

Figura 3: Entrada, saída e estados intermédios

No algoritmo os estados intermédios podem ver-se de outro ponto de vista. Os quatro bytes que formam cada coluna dão um total de 32 bits. Portanto, o estado pode ser interpretado como um vetor coluna de quatro palavras, cada uma delas formada com 32 bits. Assim, por exemplo, o estado intermédio representado na Figura 3 pode ser considerado como um vetor coluna das seguintes palavras:

O código secreto também é um array de bytes, mas temos três possibilidades: 16 bytes(128 bits), 24 bytes(192 bits) ou 32 bytes(256 bits). No caso de 128 bits, o código inicial e os posteriores processados no algoritmo serão processados como blocos de tamanho 4 por 4; no caso de 192 bits blocos de tamanho 4 por 6, e ﬁnalmente no caso de 256 bites blocos de tamanho 4 por 8.

1.2 Interpretações diferentes de um byte

Parece divertido que um byte de oito bits possa ser interpretado como um polinómio ou como um par de elementos hexadecimais. Vamos ver em que é que consiste isto.

**Notação polinomial**

Representamos um byte de oito dígitos como se segue . Este byte pode ser representado como um polinómio p(x) de grau sete como se segue:

Por exemplo, o byte representa o polinómio

**Notação hexadecimal**

O sistema hexadecimal consta de 16 dígitos: . A letra A representa 10, a letra B onze, e assim sucessivamente. Por exemplo, o número 143 em notação decimal corresponde a em notação binária e a 9E com notação hexadecimal. A representação hexadecimal que vamos fazer de um byte é realizada mediante a seguinte representação:

Padrão Caráter Padrão Caráter Padrão Caráter Padrão Caráter

Usando este padrão, se tomarmos, por exemplo, o byte corresponde em notação hexadecimal aos primeiros quatro dígitos corresponde a 9 em notação hexadecimal e os últimos quatro dígitos a E. Isto é, corresponde ao par 9E. Usando este padrão, podemos representar a matriz dada na Figura 2 em notação hexadecimal como se segue:

2 A matemática do algoritmo

Todos os bytes no algoritmo AES são interpretados como polinómios de grau 7 com coeﬁcientes 0 ou 1. De um ponto de vista mais matemático e próprio, diríamos que os bytes estão interpretados como elementos do campo ﬁnito (ver [2]). Não vamos entrar em detalhes, e vamos falar da forma mais simples, mas se quiser saber mais do que é um corpo ﬁnito, os polinómios, as estruturas algebraicas, remeta-se às referências seguintes [2, 3, 4], é um mundo amazónico!

Vamos ver as operações que podemos fazer entre os bytes mediante a representação polinomial descrita na

Equação (1.1).

Soluções

2.1 Soma

Sejam dois bytes a e b deﬁnimos a soma c entre a e b e que denotamos como a b como se segue:

**Notação de byte:**

em que denota a operação XOR binária, isto é:

**Notação polinomial:**

sendo a operação deﬁnida no anterior item.

**Exercício 2.1**. a soma do polinómio com dá como resultado o polinómio que corresponde ao byte

2.2 Multiplicação

A multiplicação que vamos explicar não é tão intuitiva como a soma anterior. De facto, vamos abrir um intervalo para falar da operação módulo.

Vamos reparar nisto: quando dividimos qualquer número natural por 2, temos duas opções de resto, 0 e 1. Esta distinção não é outra que falar de números pares e ímpares. Mais formalmente, sejam todos os números inteiros Z e realizemos a divisão entre dois. Temos uma coleção de números cujo resto é 0, e outros cujo resto é 1. Deste ponto de vista, podemos falar de Z2 (espaço cociente) que é um corpo com dois únicos elementos: a classe dos elementos pares e a classe dos elementos ímpares. Repare que a operação XOR binária definida antes tem muita relação com isto. De facto, entender a operação 1 1 = 0 é muito simples agora se pensar que a soma de dois números ímpares é um número par.

Podemos pensar em fazer algo similar, por exemplo dividindo por 5. Desta forma, o conjunto de números inteiros poderia ser descomposto em 5 classes: os divisíveis por 5 (classe [0]), os que dão resto 1 (classe [1]), os que dão resto 2 (classe [2]) , os que dão resto 3 (classe [3]) e os que dão resto 4 (classe [4]). Em que classe se encontra o número 1234?

Acha a operação modular estranha? Imagina que podemos fazer o mesmo com polinómios? Pois, também se pode. Imaginemos o polinómio e dividamo-lo pelo polinómio irredutível de grau oito: . Se dividirmos ambos os polinómios (faça-o como exercício para recordar como dividir dois polinómios) o resto que fica é o polinómio: De forma mais genérica: se tivermos um polinómio e um outro polinómio de menor ou igual grau q(x), diremos que r(x) é p(x) módulo q(x) sendo r(x) o resto de dividir p(x) entre q(x).

Dado este suposto salto ou parêntese, vamos entender o porquê disto tudo. Como se intitula esta subsecção, a multiplicação, isto é multiplicação de dois polinómios que cada um deles representa um byte. Mas se recordar como multiplicar dois polinómios, o grau do polinómio produto é a soma dos graus dos polinómios. Por exemplo, se multiplicar um polinómio de grau 3 por outro de grau 6, o resultado será um polinómio de grau 9. Mas em todo o momento dissemos que o algoritmo AES trabalhará com polinómios de grau menor ou igual que 7. Então se multiplicarmos polinómios de grau, no máximo sete, teremos na maioria dos casos polinómios de grau maior que sete. Esta é a razão pela qual a multiplicação não é a usual, mas que a multiplicação que é deﬁnida é a usual seguida de uma operação modular com o polinómio irredutível seguinte:

Ao ser q(x) um polinómio de grau 8, o resto terá sempre um grau no máximo 7, obtendo o que queríamos, trabalhar com bytes. Esta multiplicação vamos denotá-la como a(x) b(x) ou em notação de bytes como a b entre a e b.

Exercício 2.2. Sejam os polinómios e a sua multiplicação usual é e operação modular com q(x) dado na Equação (2.1) dá como resultado o polinómio . Assim,

Observemos que na operação de multiplicar os polinómios se dá a multiplicação entre elementos binários, isto é:

**2.3 Polinómios com coeﬁcientes que são bytes de oito dígitos**

Na subsecção 1.1 falamos de palavras como uns vetores colunas de 32 bits concatenados. Vamos ver como trabalhar com tudo isto e a matemática que modeliza isto. Antes falámos de polinómios de grau menor ou igual a sete com coeﬁcientes em Z2, isto é 0 ou 1. Agora vamos é deﬁnir o seguinte polinómio.

em que e são bytes de oito dígitos. Vamos ver como fizemos nas anteriores subsecções com este tipo de polinómios, como sejam as operações soma e multiplicação entre dois polinómios dados como a equação 2.2

**2.3.1 Soma**

Esta é muito intuitiva, uma vez que é suficiente somar os bytes como vimos na subsecção 2.1. Desta forma, sejam e dois polinómios como na equação (2.2), então a soma denotada como será a seguinte:

Exercício 2.3. Seja e

polinómios com coeﬁcientes bytes, é fácil conferir que

**2.3.2 Multiplicação**

Esta operação segue um raciocínio similar ao realizado na subsecção 2.2. Mas é um pouco mais complicado.

**Passo multiplicativo** Criamos o polinómio produto a partir de e dados como antes, como se segue:

em que

**Passo modular** Como o resultado na equação (2.3) não é um polinómio de quatro palavras, temos que realizar uma operação modular com um polinómio de grau 4. O algoritmo AES realiza-o com o polinómio . Esta operação vamos deixá-la como exercício para que pratique (pode ver todos os detalhes em [5]).

Finalmente, se trabalhar o que propusemos, pode ver-se que realizar o produto entre e b(x), o qual denotaremos como , é equivalente a realizar uma operação matricial em que as somas e produtos são os explicados na subsecção anterior, isto é e entre bytes. Assim,

em que

Observemos que se tomamos como polinómio o resultado do produto de a(x) b(x) é uma rotação da palavra ; isto é . Esta operação vai denotar o algoritmo AES como rotWord().

3 Um esquema do algoritmo

Antes de dar as ideias do algoritmo, queremos salientar vários pontos:

• A longitude da entrada e saída é de 128 bits representados em blocos de tamanho quatro por quatro, cada elemento um byte (usando notação hexadecimal será mais fácil representá-lo).

• A longitude do código secreto dependerá do tipo de algoritmo AES que use. Distinguem-se três AES-128, AES-192 e AES-256, os quais usam um código secreto, respetivamente de 128 bits, 192 bits e 256 bits.

• O número de voltas que se dará ao algoritmo dependerá também do tipo de AES. O AES.128 realizará 10 voltas, o AES-192 doze e o AES-256 realizará 14.

Para não complicar demais o algoritmo, apenas veremos o processo de encriptação. O processo de desencriptação é similar mas com operações inversas.

**3.1 Input do algoritmo**

O estado e o código secreto. Usando notação hexadecimal teríamos algo assim

ESTADO INICIAL CÓDIGO SECRETO

Figura 4: Input do Algoritmo AES-128

**3.2 Processo de encriptação do estado**

Distinguimos três etapas:

Roda inicial

ESTADO INICIAL CÓDIGO SECRETO ESTADO 0

Figura 5: Roda Inicial do algoritmo AES-128

9 rodas

ESTADO éssimo ESTADO éssimo

Figura 6: Roda i-éssima do algoritmo AES-128 para

Roda final

ESTADO 9 ESTADO FINAL

Figura 7: Última roda do algoritmo AES-128

Nas Figuras 5, 6 e 7 mostra-se um esquema do processo de encriptação do algoritmo AES e na. O processo de desencriptação é bastante análogo mas com operações inversas. Na secção em anexo vamos pôr claros exemplos do que fazem cada uma das operações que aparecem no esquema, estas são: SubBytes(), ShiftRows(), MixColumns(), AddRoundKey().

**3.3 Processo de código**

O código inicial irá modiﬁcando-se ao longo do processo para ser utilizado do modo que se apresenta nas ﬁguras 6 e 7. Este processo consiste em, dado o código inicial, o qual vamos ver como um array de quatro palavras, cada palavra serão os 32 bits que encontramos em cada coluna. Para gerar a primeira palavra do código ronda 1, o que se fará é aplicar à palavra 4 do atual código a operação RotWord() que vimos na subsecção 2.3.2, e depois de ter feito isto, soma-se a primeira palavra e a primeira coluna da matriz Rcom que aparece na tabela 1. A segunda palavra é calculada a partir da calculada somando a segunda do código anterior; o mesmo com a terceira e a quarta. E este seria o código volta 1. As nove restantes são calculadas de forma análoga, utilizando sempre a anterior.

Nas Figuras 5, 6 e 7 mostra-se um esquema do processo de encriptação do algoritmo AES e na. O processo de desencriptação é bastante análogo, mas com operações inversas. Na secção em anexo vamos pôr claros exemplos do que fazem cada uma das operações que aparecem no esquema, estas são: SubBytes(), ShiftRows(), MixColumns(),

AddRoundKey().

3.3 Processo de código

O código inicial irá modiﬁcando-se ao longo do processo para ser utilizado do modo que se apresenta nas ﬁguras 6 e 7. Este processo consiste em, dado o código inicial, o qual veremos como um array de quatro palavras, cada palavra serão os 32 bits que encontramos em cada coluna. Para gerar a primeira palavra do código ronda 1, o que se fará é aplicar à palavra 4 do atual código a operação RotWord() que vimos na subsecção 2.3.2, e após ter feito isto se somará a primeira palavra e a primeira coluna da matriz Rcom que aparece na tabela 1. A segunda palavra calcula-se a partir da calculada somando a segunda do código anterior; o mesmo com a terceira e a quarta. E este seria o código ronda 1. As nove restantes são calculadas de forma análoga utilizando sempre a anterior.

Tabela 1: Matriz Rcon, cada coluna representa a potência módulo para

4. Conclusões

O processo de desencriptação é muito similar ao mostrado, mas as operações que são utilizadas e que se explicam no Anexo 4 devem ser vistas de forma inversa. Todos os detalhes pode vê-los em 5. Se quiser ver exemplos, pode remeter-se a 1, em que aparece um programa e códigos das partes dos algoritmos.

Anexo: as operações do processo de encriptação

Aqui vamos representar de maneira informal e com simples exemplos cada uma das operações que aparecem no processo de encriptação do Algoritmo RSA-128. Por simplicidade vamos trabalhar com notação hexadecimal.

1. SubBytes( · ): é uma transformação não linear que substitui cada byte de um estado por um outro utilizando uma tabela de substituição. Esta tabela é denominada S-box (ver [5] para ver a origem da mesma)

Tabela 2: Substituição do byte xy (formato hexadecimal).

Desta forma, se tivermos o estado seguinte,

vamos substituir cada elemento deste bloco por um outro resultante da S-box. Por exemplo, o elemento posicionado na fila 2 e coluna 4 é 5B. Portanto, tomamos o elemento resultante da S-box da fila que corresponde ao elemento 5 e a coluna B, este novo elemento é 39. De forma análoga é fácil ver que:

2. ShiftRows: é uma transformação que a fila i-éssima mudanças cíclicas, para . Por exemplo, se é a fila três de um estado, realizam-se duas mudanças cíclicas, isto é . Desta forma podemos ver:

3. MixColumns: é uma transformação que reflete a operação matemática vista na subsecção 2.3.2. Consideramos cada coluna de um estado como os coeficientes de um polinómio da forma como na Equação 2.2 , e cada uma dela será multiplicado pelo polinómio fixo .

Assim, a operação MixColumns realizada em cada coluna de um estado é a operação definida em 2.2. Pode tentar confirmar o seguinte.

4. AddRoundkey: é uma transformação que tem dois argumentos, um estado e um código e a operação que se realiza é elemento a elemento a soma explicada na subsecção 2.1. O seguinte exemplo mostra o seguinte:

Referências