Questionário SHA-3 (Keccak)

Lucas Tonussi, matr.: *12106577*

3 de julho de 2015

1 Questões

1. Definição de função de hash criptográfica.

   - Função de hash criptográfica é um tipo de função hash necessária para aplicações de segurança. É um algoritmo do qual é um problema intratável de se achar uma informação que mapeia para um resumo (unilateralidade), ou duas informações que mapeiam para um mesmo resumo (livre de colisões) (Stallings, pg 314).

   - Para **funções hash**, a entrada é chamada de mensagem m, e a saída é chamada a mensagem moída ou um valor hash. O tamanho da mensagem pode varias é claro; mas o tamanho da mensagem moída é fixo. Uma função de hash criptográfico é uma função hash que é desenhada para prover propriedades especiais, incluindo resistência a colisão e resistência a bilateralidade (pre-imagem), que são importantíssimas para muitas aplicações envolvendo segurança da informação.

2. Propriedades.

   Requerimentos e Segurança para funções de resumos criptográfico, existem alguns porém a questão pede apenas 3 requisitos. (Stallings Página 323 Capítulo 11.3, Tabela 11.1, Linhas 4, 5, 6).

   - Preimage resistant (one-way property): $h = H(m),\ h \rightarrow m$ **mas** **nunca** $h \leftarrow m$.

- Second preimage resistant (weak collision resistant): $len(H_1(m_1)) = len(H_2(m_2)) = \ldots = len(H_n(m_n))$ **deve ser** $len(H(m)) \leq len(m)$.

- Collision resistant (strong collision resistant): $h_1 = H(m_1),\ h_2 = H(m_2),\ \ldots,\ h_n = H(m_n) \mid h_1 \neq\ h_2 \neq,\ \ldots,\ \neq\ h_n$.

3. Explicar o **SHA3**.

   É um acrônimo para Secure Hash Algorithm-3 (DRAFT FIPS 202, pg 4). É também uma família de algoritmos Hash-Criptográfico. Ele suplementa as famílias já depreciadas, SHA1 e SHA2. Cada função do SHA-3 é baseado no algoritmo KECCAK [6] (NIST Instituto Nacional de Padrões e Tecnologia). O padrão também inclui o KECCAK-P que implementa permutações para o mesmo (p é de permutar). Esse padrão SHA3 é aplicado em todos os departamentos federais e agencias de proteção a informações secretas (DRAFT FIPS 202, pg 5). Quem mantêm os código e chaves é o NIST, o documento de especificação do SHA3 é o FIPS 202. Quem aprova as implementação é a Secretaria do Comércio (Estadunidense). A família SHA3 contém 6 implementações, 4 delas são funções de hash criptográfico:

   - SHA3-224
   - SHA3-256
   - SHA3-384
   - SHA3-512

   e as outras duas são do tipo Funções de Vazão-Estendida[1] são eles:

   - SHAKE128
   - SHAKE256 [2]

nota: Esses dois XOFs acima são modelados para resistir a colisões, e ataques à primeira e segunda pre-imagem (DRAFT FIPS 202, pg 31, pg 23 A.1.1 Security Summary).

[1] Extendable-Output Functions ou XOFs
[2] SHAKE, Secure Hash Algorithm Keccak

Para essas variantes acima 3, que foram propostas pelo campeonato do SHA-3, o valor do parâmetros c é igual ao comprimento do resumo multiplicado por 2. Por exemplo, o candidato SHA-3 com 512-bit de comprimento é o Keccak com c = 1024 e r = 576 (r + c = 1600). Variantes são então, Keccak-224, Keccak-256, e Keccak-512 (O número no final é o comprimento do resumo criptográfico gerado por essas funções (DRAFT FIPS 202, pg 20, 21).

O número r de mensagem em bits processadas por bloco de permutação depende da vazão do comprimento do resumo criptográfico. A rate r pode ser 1152, 1088, 832, ou 576 que bate respectivamente com os tamanhos de resumos criptográficos 224, 256, 384, e 512 bits (DRAFT FIPS 202, pg 20, 21).

Para assegurar que a mensagem pode ser dividida corretamente em blocos de tamanho igual a r-bits, e será enxertado (padded) com valor 1 binário, zero ou mais bits 0 (zero) binário e ao final novamente com o valor 1 binário (por isso que o nome da função se chama pad10star1 (DRAFT FIPS 202, pg 19).

Finalmente o estado (state) pode ser visualizado como um arranjo de 5x5 linhas (lanes) onde cada linha é uma palavra de 64-bit (64-bit word). O estado inicial de 1600-bit é totalmente preenchido com zeros (DRAFT FIPS 202, pg 17).

4. Responder as seguintes questões relacionadas a referência [3] que apresenta o SHA3 padronizado pelo NIST.

   (a) O que é e para que serve o *State Array* (figura 1)?

   O estado para a permutação `Keccak-p[b, nr]` é composta de **b bits**. A especificação nessa Padronização contem duas variáveis novas relacionadas à quantidade b de bits; são elas $w = b//25$ e $l = \log_2(b//25)$ [3]. Os sete possíveis valores das variáveis que são definidas para as permutações `Keccak-p` estão abaixo na tabela.

   Esses valores são convenientes em representar os estados de entrada e saída da permutação como strings de b-bits, e também para representar a entrada e saída para cada mapeamento como arranjos de $5 \times 5 \times w$ bits. Podemos definir S agora, denotando a string de cada Mapeamento que representa o estado atual, então, serão os bits indexados de 0 até b-1, serão portanto $b - 1$ estados:

[3] Símbolo mtm.: // significa divisão inteira.

b	25	50	100	200	400	800	1600
w	1	2	4	8	16	32	64
l	0	1	2	3	4	5	6

Tabela 1: Valores de $b,\ w,\ l$ relacionados para a permutação KECCAK-p

$$S = S[0] \parallel S[1] \parallel S[2] \parallel \ldots \parallel S[b-1]$$[4]

Agora interpretando a figura 1 da página 8, temos **A** denotando o nosso arranjo $5 \times 5 \times w$ de bits, e os seus índices são as 3-tuplas $(x,\ y,\ z) \mid 0 \leq x < 5$ **e** $0 \leq y < 5$ **e** $0 \leq z < w$. O bit que corresponde à coordenada $(x,\ y,\ z)$ é denotado por $A[x,\ y,\ z]$. O arranjo estado [5] é uma mera representação do estado em um arranjo tri-dimensional. Cada peça da figura 1 está exposta a seguir (Referência Keccak, sec. 1.6, pg 10, Partes do estado [7]):

i. estado (state) - conjunto de $5 \times 5 \times w$ bits
ii. plano (plane) - conjunto de 25 bits (coord. y constante)
iii. fatia (slice) - conjunto de 25 bits (coord. z constante)
iv. folha (sheet) - conjunto de $5w$ bits (coord. x constante)
v. linha (row) - conjunto 5 bits (coord. y,z constante)
vi. coluna (column) - conjunto de 5 bits (coord. x,z constante)
vii. faixa (lane) - conjunto de w bits (coord. x, y constante)
viii. pedaço (bit) - 1 bit

(b) Como é feita a conversão de strings para *State Arrays*? Mostrar também um exemplo de vocês, diferente do NIST.

Tome S como a **string** de b bits (b só pode ser 25, 50, 100, 200, 400, 800 ou 1600) que representa o estado para a permutação `Keccak-p[b, nr]`. O arranjo de estados correspondente, denotado por A', é definido como segue: $\forall\ (x,\ y,\ z) \mid 0 \leq x < 5, 0 \leq y < 5$ **e** $0 \leq z < w$ **faça**:

Fórmula: $A[x, y, z] = S[w(5y + x) + z]$

Seja S = 0xbbbb agora precisamos setar as configurações de $b = 200;\ w = b\ //\ 25 = 8$. String: $1011101110111011010000000000000 \ldots 000000000000000$; onde Tamanho da String:

[4] Símbolo mtm.: $\parallel$ denota a concatenação usual de string bytes.
[5] Para o permutador Keccak-p, é um cubo binário $5 \times 5 \times w$ dimensões.

$200bits$ são completados com zeros.

$A[0,0,0] = S[8*(0+0)+0] = S[0] = 1$
$A[0,0,1] = S[8*(0+0)+1] = S[1] = 0$
$A[0,0,2] = S[8*(0+0)+2] = S[2] = 1$
$A[0,0,3] = S[8*(0+0)+3] = S[3] = 1$
$A[0,0,4] = S[8*(0+0)+4] = S[4] = 1$
$A[0,0,5] = S[8*(0+0)+5] = S[5] = 0$
$A[0,0,6] = S[8*(0+0)+6] = S[6] = 1$
$A[0,0,7] = S[8*(0+0)+7] = S[7] = 1$
$A[0,0,8] = S[8*(0+0)+8] = S[8] = 1$
$A[0,0,9] = S[8*(0+0)+9] = S[9] = 0$
$A[0,0,10] = S[8*(0+0)+10] = S[10] = 1$
$A[0,0,11] = S[8*(0+0)+11] = S[11] = 1$
$A[0,0,12] = S[8*(0+0)+12] = S[12] = 1$
$A[0,0,13] = S[8*(0+0)+13] = S[13] = 0$
$A[0,0,14] = S[8*(0+0)+14] = S[14] = 1$
$A[0,0,15] = S[8*(0+0)+15] = S[15] = 1$
$A[0,0,16] = S[8*(0+0)+16] = S[16] = 0$
$A[0,0,17] = S[8*(0+0)+17] = S[17] = 1$
$A[0,0,18] = S[8*(0+0)+18] = S[18] = 0$
$A[0,0,19] = S[8*(0+0)+19] = S[19] = 0$
$A[0,0,20] = S[8*(0+0)+20] = S[20] = 0$
$A[0,0,21] = S[8*(0+0)+21] = S[21] = 0$

...

$A[3,2,0] = S[8*(2+3)+0] = S[19] = 0$
$A[3,2,1] = S[8*(2+3)+1] = S[20] = 0$
$A[3,2,2] = S[8*(2+3)+2] = S[21] = 0$
$A[3,2,3] = S[8*(2+3)+3] = S[22] = 0$
$A[3,2,4] = S[8*(2+3)+4] = S[23] = 0$
$A[3,2,5] = S[8*(2+3)+5] = S[24] = 0$
$A[3,2,6] = S[8*(2+3)+6] = S[25] = 0$
$A[3,2,7] = S[8*(2+3)+7] = S[26] = 0$
$A[3,2,8] = S[8*(2+3)+8] = S[27] = 0$
$A[3,2,9] = S[8*(2+3)+9] = S[28] = 0$

...

$A[4,0,181] = S[8*(0+4)+181] = S[185] = 0$
$A[4,0,182] = S[8*(0+4)+182] = S[186] = 0$
$A[4,0,183] = S[8*(0+4)+183] = S[187] = 0$
$A[4,0,184] = S[8*(0+4)+184] = S[188] = 0$
$A[4,0,185] = S[8*(0+4)+185] = S[189] = 0$
$A[4,0,186] = S[8*(0+4)+186] = S[190] = 0$
$A[4,0,187] = S[8*(0+4)+187] = S[191] = 0$
$A[4,0,188] = S[8*(0+4)+188] = S[192] = 0$
$A[4,0,189] = S[8*(0+4)+189] = S[193] = 0$
$A[4,0,190] = S[8*(0+4)+190] = S[194] = 0$
$A[4,0,191] = S[8*(0+4)+191] = S[195] = 0$
$A[4,0,192] = S[8*(0+4)+192] = S[196] = 0$
$A[4,0,193] = S[8*(0+4)+193] = S[197] = 0$
$A[4,0,194] = S[8*(0+4)+194] = S[198] = 0$
$A[4,0,195] = S[8*(0+4)+195] = S[199] = 0$
$A[4,0,196] = S[8*(0+4)+196] = S[200] = 0$

(c) Como é feita a conversão de *State Array* para Strings? Incluir um exemplo de vocês.

Seja **A** o state array. A string correspondente, denotada por **S**, pode ser construído pegando as Linhas (ou Trilhas) de **A**, da seguinte forma, cada par de inteiros $(i,j) \mid 0 \leq i < 5$ **e** $0 \leq j < 5$ definem a string Linha(i,j) por: $Linha(i,j) = A[i,j,0] \parallel A[i,j,1] \parallel A[i,j,2] \parallel A[i,j,0] \parallel A[i,j,w-2] \parallel A[i,j,w-1]$. Um exemplo:

$Trilha(0,0) = A[0,0,0] \parallel A[0,0,1] \parallel A[0,0,2] \parallel \ldots \parallel A[0,0,62] \parallel A[0,0,63]$ corresponde a: $Trilha(0,0) = 1 \parallel 0 \parallel 1 \parallel \ldots \parallel 0 \parallel 0$

$A[1,0,0] = [8(5*1+0)+0] = S[40] = 0$
$A[1,0,2] = [8(5*1+0)+1] = S[41] = 0$
$A[1,0,3] = [8(5*1+0)+2] = S[42] = 0$

$\ldots$

$A[1,0,4] = [8(5*1+0)+62] = S[102] = 0$
$A[1,0,5] = [8(5*1+0)+63] = S[103] = 0$

$Trilha(1,0) = A[1,0,0] \parallel A[1,0,1] \parallel A[1,0,2] \parallel \ldots \parallel A[1,0,62] \parallel A[1,0,63]$ corresponde a: $Trilha(1,0) = 0 \parallel 0 \parallel 0 \parallel \ldots \parallel 0 \parallel 0$. E assim sucessivamente. O que se faz é uma

transformação linear $R^3 \rightarrow R^2$ das coordenadas da dimensão do arranjo. O mesmo para fazer o contrário String para Linha.

(d) Explicar os cinco Passos de Mapeamento (NIST, pg 19, 3.2 Mapeamentos de passo). Explicar os algoritmos envolvidos e cada uma das figuras apresentadas (figuras 3 a 6, páginas 19 até 23).

A maneira mais fácil de entender o θ é ver o arranjo estado em duas dimensões (mais precisamente: 5×5). Notar que cada módulo (os cinco passos denotados pelas letras gregas $\theta, \rho, \pi, \chi, \iota$) são invocados pelas 24 permutações. E devem ser aplicadas nessa ordem $\iota(\chi(\pi(\rho(\theta(\mathbf{A})))))$, isto é, as funções são aplicadas em cima da matriz A de bits (arranjo de estados). Todos os algoritmos estão explicados a seguir tem entrada A e saída A'. Sendo que os limites das variáveis já foram especificados anteriormente. Esses algoritmos de transformação dos bits retornarão ao final um arranjo de estados atualizado.

Algoritmo $\theta(A)$ (DRAFT FIPS 202, pg 19)

$C[x] = A[x,0] \oplus A[x,1] \oplus A[x,2] \oplus A[x,3] \oplus A[x,4], \quad x = 0,1,2,3,4$
$D[x] = C[x-1] \oplus rotl(C[x+1],1), \quad x = 0,1,2,3,4$
$A[x,y] = A[x,y] \oplus D[x], \quad x,y = 0,1,2,3,4$

O mapeamento θ é linear e foca na difusão e é invariante em todas as direções. Seu efeito pode ser descrito da seguinte maneira: ele trabalha no arranjo todo $5 \times 5 \times w$. Computa as paridades de cada coluna e combina elas com o operador $\oplus$. Então faz $\oplus$ com as paridades resultantes para cada bit-estado da seguinte forma: $A'[i][j][k] = A'[i][j][k] \oplus paridade(A[0\ldots4][j-1][k]) \oplus paridade(A[0\ldots4][j+1][k+1])$ **where** $i = 0\ldots4; j = 0\ldots4; k = 0\ldots(w-1)$ Então Esse algorítimo tem por efeito fazer $\oplus$ cada bit no arranjo de estado aos seus pares de duas colunas no arranjo. Os cálculos são feito usando limites modulares (mod 5) para não passarem no limite do arranjo (DRAFT FIPS 202, pg 20).

Algoritmo $\rho(A)$ (DRAFT FIPS 202, pg 20)

i. $\forall z$ *dentre* $0 \leq z < w$ *faça* $A'[0,0,z] = A[0,0,z]$
ii. $(x,y) = (1,0)$
iii. **Para** t **de** 0 **até** 23 :
$\forall z,$ *dentre* $0 \leq z < w,$ *faça* $A'[x,y,z] = A[x,y,(z–(t+1)(t+2)/2) \ mod \ w];$
$(x,y) = (y,(2x+3y) \ mod \ 5);$
iv. **Retorne** A'.

O módulo ρ rotaciona cada linha de $w - bits$ por um número triangular $0, 1, 3, 6, 10, 15$ O efeito do algoritmo acima (ρ) é rotacionar os bits de cada linha pela seu *mod length*, length também chamado *offset* o qual depende de x,y fixos nas coordenadas da linha (DRAFT FIPS 202, pg 20).

Algoritmo $\pi(A)$ (DRAFT FIPS 202, pg 21 Mapeamentos de passo)

i. $\forall(x, y, z) \mid 0 \leq x < 5$ **e** $0 \leq z < w$:
   $y' \leftarrow (x + 3y)$ **mod 5**;
   $A'[x, y, z] = A[y',\ x,\ z]$;
ii. **Retorne** A'

O módulo π permuta os w-bits. A permutação segue o seguinte padrão: $A[j][2 \times i + 3 \times j][\,] = A[i][j][\,]$. O efeito do algoritmo acima é rearranjar as posições das linhas. Note que y' sobre uma transformação linear e o arranjo permuta as posições de $x, y, z \rightarrow y', x, z$ e y' esta descrito acima no algoritmo.

Note que ambos os passos ρ e π podem ser computados de uma só vez. Criemos um arranjo auxiliar B (5x5) a partir do arranjo estado A. Veja que B[i,j] é a palavra com **w-bits**.
Algoritmo $\rho \cup \pi(A)$ (DRAFT FIPS 202, pg 21 Mapeamentos de passo)
$B[y, 2x + 3y] = rotl(A[x, y], r[x, y]),\ x, y = 0, 1, 2, 3, 4$

Algoritmo $\chi(A)$ (DRAFT FIPS 202, pg 22 Mapeamentos de passo)

i. $\forall(x, y, z) \mid 0 \leq x < 5, 0 \leq y < 5,$ **e** $0 \leq z < w$:
   $A'[x, y, z] = A[x, y, z] \ \oplus\ ((A[(x + 1)$ **mod 5**$,\ y,\ z] \ \oplus\ 1) \wedge\ A[(x + 2)$ **mod** $5,\ y,\ z])$;
ii. **Retorne** A'

Para fins de implementação fica mais fácil olhar para o algoritmo χ da seguinte forma: $A[x, y] = B[x, y] \oplus ((\bar{B}[x + 1, y]) \wedge B[x + 2, y]),\ x, y = 0, 1, 2, 3, 4$

Aqui $\bar{B}[i, j]$ denota o complemento bit a bit da linha no endereço [i,j], e $\wedge$ é a operação e lógico bit a bit. Em todos os passos os índices são módulo 5. Ele pega as linhas nos endereços [x+2, y] e faz $\wedge$ com as linhas nos endereços [x+1, y] e depois o resultado disso, faz $\oplus$'s com a linha no endereço [x,y] (DRAFT FIPS 202, pg 23, Figure 6: χ aplicado a uma única linha).

Algoritmo $\iota(A)$ (DRAFT FIPS 202, pg 19 Mapeamento)

RC[0] = 0x0000000000000001	RC[12] = 0x000000008000808B
RC[1] = 0x0000000000008082	RC[13] = 0x800000000000008B
RC[2] = 0x800000000000808A	RC[14] = 0x8000000000008089
RC[3] = 0x8000000080008000	RC[15] = 0x8000000000008003
RC[4] = 0x000000000000808B	RC[16] = 0x8000000000008002
RC[5] = 0x0000000080000001	RC[17] = 0x8000000000000080
RC[6] = 0x8000000080008081	RC[18] = 0x000000000000800A
RC[7] = 0x8000000000008009	RC[19] = 0x800000008000000A
RC[8] = 0x000000000000008A	RC[20] = 0x8000000080008081
RC[9] = 0x0000000000000088	RC[21] = 0x8000000000008080
RC[10] = 0x0000000080008009	RC[22] = 0x0000000080000001
RC[11] = 0x000000008000000A	RC[23] = 0x8000000080008008

Tabela 2: Tabela das constantes, que foram usadas na implementação keccak.py que vem em anexo são as constantes padrões Keccak (64 bits).

i. $\forall(x, y, z) \mid 0 \leq x < 5, 0 \leq y < 5,\ and\ 0 \leq z < w,$ **faça** :
   $A'[x, y, z] = A[x, y, z]$
ii. $RC = 0^w$
iii. $\forall j,\ 0$ **até** l :
   $RC[2^j - 1] = rc(j + 7 * i_r)$
iv. $\forall z \mid 0 \leq z < w$:
   $A'[0, 0, z] = A'[0, 0, z] \ \oplus \ RC[z]$
v. **Retorne** A'

Para fins de implementação fica mais fácil olhar para o algoritmo χ da seguinte forma:
$A[0, 0] = A[0, 0] \oplus RC[i]$

O módulo ι insiste em **desmembrar** as simetrias causadas pelos outros módulos. (Keccak Reference 3.0, Sec 2.3.5, pg 22, Propriedades do ι)

E como isso é feito? Isso é feito fazendo $\oplus$ de um elemento do arranjo (1 bit) com uma constante de rodada periodicamente aplicada. O módulo tem 24 constantes para escolher, que são definidas internamente pelo `Keccak-f` sem as constantes o mapeamento ι iria ser simétrico, causando todo os mapeamentos para serem simétricos, uma das razões do ι ser o último módulo executado é justamente para desmembrar a simetria, dando suscetibili-

dade a ataques como Slide (Keccak Reference 3.0, Sec 4.1.3, Ataques tipo Slide) [6]. Esse parâmetro novo aqui, dentre os algoritmos de modulo mapeadores, chamado i_r implica em $l+1$ que será reconhecida como a constante da rodada, denotada por RC. Cada valor gerado por $l+1$ surge da função baseada em um modo de operação chamado retroalimentação linear de registrador deslocamento (linear feedback shift register) essa função é chamada de rc (DRAFT FIPS 202, pg 23, $rc(t)$), note que no passo 2 (ii) temos a concatenação w de zeros.

O efeito do algoritmo acima (iota) é modificar alguns bits da Linha(0, 0) da maneira que fica dependente do índice i_r (i rodadas). As outras 24 linhas não são afetadas pelo algoritmo iota. Além da entrada ser o state array $\mathbf{A}$, também temos *round index* i_r.

(e) Explicar a permutação `Keccak-p[b, n]`.

O *Keccak-p* contém um tamanho fixo de string que serão permutadas, chamado de espessura da permutação, e também o número de iterações da transformação (rodada) interna. A espessura é o b e o número de rodadas (transformações internas) é n_r.

Dado um arranjo de estados A e também um índice i_r então a função de rodada Rodada [7] é uma transformação que resulta da aplicação dos Passos de Mapeamento θ, ρ, π, χ, ι nessa mesma ordem, isto é: $Rnd(A, i_r) = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$.

A permutação `KECCAK-p[b, n]` consiste de n_r iterações do procedimento Rnd, como especificado no Algoritmo 7, `KECCAK-p[b, n](S)`.

i. **Entrada**: Uma mensagem S de tamanho b; Número de rodadas n_r.
ii. **Saída**: Uma mensagem S' de tamanho b.
iii. **Converter** S em Arranjo de Estados A, como descrito na Sec. 3.1.2 [3].
iv. $l \leftarrow len(S)$
v. **Para** i_r de $(2l + 12 - n_r)$ **até** $(2l + 12 - 1)$:
$A \leftarrow Rnd(A,\ i_r)$.

[6] São ataques que exploram a simetria numa primitiva que consiste da iteração de um número idêntico de rodadas. No âmbito do ι. todas as rodadas são idênticas (no sentido que elas não permutam entre sí, mas os bits no arranjo de estados ganham anti-simetria ao decorrer das permutações). Nesse caso, isso permite em distinguir propriedades do Keccak-f: a distribuição do comprimento dos ciclos irá significativamente diferenciar de um estilo de permutação do tipo fortemente aleatória

[7] Rodada ou "Rnd" ou "Round" é uma função da permutação KECCAK-P, no código em Python3, que vem em anexo ao artigo, essa função se chama *Round*. Esse código está disponível em: http://keccak.noekeon.org/

vi. **Converter** A na **string** S' de tamanho b, como descrito na (DRAFT FIPS 202, sec. 3.1.3 [3])

vii. **Retornar** S'.

i. Compare esta função com a `Keccak-f`.

Keccak é uma família de funções hash baseadas na construção esponja. A esponja pode ser visualizada como uma matriz de tri dimensional $\mathbf{A} = a(x, y, z)_{5x5xw}$ Que também eh conhecida como 'estado' ou em (inglês.: 'state'), a qual é entrada para as fases "inchamento"(A entrada $\oplus$-ring é parte do estado) e "espremida"da esponja faz o papel de processar as saídas do hash O Keccak usa uma função chamada Keccak-f[b] que atua no 'estado' todo onde b pertence a 25, 50, 100, 200, 400, 800, 1600 O padrão oficial usa w = 64 como numero de entradas binarias na linha que significa a profundidade do arranjo estado, ou simplesmente estado (inglês.: *state array*). Exemplo: $5 \times 5 \times 65 = 1600$, isto é, $b = 1600$. Uma string s de $5 \times 5 \times w$ bits vai ser mapeada para b bits do 'matriz estados' (= arranjo estado) da seguinte maneira s[w(5*y+x)+z] = a[x][y][z]. E b, bits contíguos, no estado podem ser divididos entre a "parte exterior"(os primeiros r bits) e a "parte interior"ou a "capacidade"'c' (inglês.: *capacity*) consistindo de c = b - r bits ou c + r = b bits. 'c' eh escolhido para ser 2N onde N eh o tamanho da vazão que vai ser gerada pelo algoritmo. [6, 8]

O SHA3 utiliza `Keccak-p[1600, 24]` (1600 bits, 24 permutações), é equivalente a simplesmente `Keccak-f[1600]`. As rodadas do `Keccak-f[b]` são indexadas de 0 até $(11 + 2l)$. O resultado é indexado dentro do Passo 2 — Algoritmo 7 pois a rodada do `Keccak-p[b, n]` bate com a última rodada do `Keccak-f[b]`, e vice versa.

Por exemplo, `Keccak-p[1600, 19]` (1600 bits, 19 permutações) é equivalente a última rodada de número 19 do `Keccak-f[1600]`. Similarmente, `Keccak-f[1600]` é equivalente para com a 24ª rodada do `Keccak-p[1600, 30]` (1600 bits, 30 permutações); mas nesse caso, as rodadas precedentes para `Keccak-p[1600, 30]` são indexadas de -6 à -1 (DRAFT FIPS 202, sec. 3.4, pg 25).

(f) Descrever o *Framework Sponge Construction*.

A construção esponja para funções de hash. pi são blocos da string de entrada, zi são blocos do hash de saída. (DRAFT FIPS 202, Figure 7: The sponge construction). A construção `[f, pad, r]` com entrada sendo o comprimento, A "esponja" é uma função que lembra as "Redes de Feistel". A saída é arbitrária e se dá a partir de permutações de tamanho fixado

pela norma SHA3 e constrói a função esponja a partir da permutação f, o a regra de padding "pad"e o parâmetro bitrate r [7].

Tam. digest	224	256	384	512
Tam. mensagem	∞	∞	∞	∞
Tam. bloco (bitrate r)	1152	1088	832	576
Tam. palavra	64	64	64	64
Nro de rodadas	24	24	24	24
Cap. c	448	512	768	1024
Resis. colisões	2^{112}	2^{118}	2^{192}	2^{256}
Resis. 2^{a} pre-imagem	2^{224}	2^{256}	2^{384}	2^{512}

Tabela 3: Parâmetros SHA-3, Todas as medições estão em bits. Atente para as duas últimas linhas, pois as demais são para se habituar para a solução da construção esponja. Essa tabela mostra que essa função esponja tem as resistências dadas em quantidades de entradas que se tem que gerar para cair no caso do paradoxo do aniversário, e possivelmente ficar sujeito aos ataques vistos no começo do artigo 2. Todos esses valores nas últimas duas linhas são intratáveis computacionalmente.

i. Explique a figura 7 e o Algoritmo 8.

A figura 7 representa a seguite construção em forma paramétrica [2]:

$$Z = ESPONJA[f, pad, r](M, d)$$

que consiste em colocar "pad" e "inchar" o conteúdo fazendo $\oplus's$ do padding com os r bits menos significativos. r é o valor de rate, ou bitrate.

A analogia com esponja é porque a função "absorve" um número arbitrário da entrada (em bits) nos seu "estado".

Algoritmo 8: SPONGE[f, pad, r](M,d)

Entra com uma string Mensagem M
Entra com um inteiro d positivo
Saída com string Z de tamanho len(Z)=d

A. Seja $P = M \parallel pad(r,\ len(M)$ Concatena preenchimento de tamanho r
B. Seja n = len(Texto_Padding) / r

C. Capacidade = 1600 - r

D. **para cada** $P_0, \ldots, P_{n-1}$ para que seja uma sequencia de strings de tamanho r tal qual $P = P_0 \parallel, \ldots, \parallel P_{n-1}$

E. Seja $S = 0^h$

F. **para cada** i **de** 0 **até** $n - 1$

Faça XOR entre S e $(P_i \parallel 0^c)$)

Aplica o resultado na função *Keccak-f*

G. Seja Z uma string vazia.

H. Seja $Z = Z \parallel Truncado_{rbits}(S)$

I. **se** $d \leq |Z|$

Retorne $Truncado_d(Z)$

J. **do contrário**

continua processando concatenação de Z com S truncado r bits

K. Seja S=Keccak-f(S), e continue com o passo **H**

Note que a entrada d determina o número de bits que o algoritmo 8 retorna, mas isso não afeta os valores. A princípio, a vazão pode ser uma string infinita, e a computação **parará** no número desejado de vazão de bits.

(g) Explique a família de funções esponja Keccak, conforme seção 5.

Keccak é a família de "funções esponja" [7]. Uma característica dela é padding rule (regra de preenchimento) chamada *multi-rate padding* (preenchimento multi-frequência, explicado melhor nessa seção). Os parâmetros e as permutações intrínsecas para a família Keccak são descrições nessa seção também, e o conjunto de famílias da menor a maior ajudam a definir suficientemente as funções do SHA-3 na próxima seção.

A regra de preenchimento multi-frequência, denotada por $pad1(0^*)1$, é especificada no Algoritmo 9 (DRAFT FIPS 202, pg 27, $pad1(0^*)1(x, m)$).

Algorithm pad10*1(x, m) (DRAFT FIPS 202, pg 27)

i. **Entrada:** Inteiro Positivos x, m

ii. **Saída:** String Z, onde $m + len(Z)$ é positivo e múltiplo de x.

iii. $j \leftarrow -(m + 2)\ mod\ x$

iv. **Retorne** 1 || 0^j || 1

Para assegurar que a mensagem pode ser dividida corretamente em blocos de tamanho igual a r-bits, e será enxertado (padded) com valor 1 binário, zero ou mais bits 0 (zero) binário e ao final novamente com o valor 1 binário (por isso que o nome da função se chama pad10star1.

A família de funções esponjas Keccak é então a soma da função de permutação intrínseca padrão $KECCAK-p[b, 12+2\times l]$ (DRAFT FIPS 202, Sec 3.3) mais a regra de preenchimento $pad10^*1$ (DRAFT FIPS 202, Sec 5.1).

$$b = 1600$$

$$w = b \;//\; 25 = 64$$

$$l = int(log_2(w)) = 6$$

$$n_r = 12 + 2 \times l = 12 + 2 \times 6 = 24 \; permutações$$

A família é parametrizada por qualquer escolha de rate r e de capacidade c (define o nível de segurança) da qual $r + c$ pertence ao conjunto de possibilidades 25, 50, 100, 200, 400, 800, 1600. Nós obtemos a função esponja `Keccak[r,c]`, com parâmetros de capacidade c e bitrate (ou apenas rate) r (define a velocidade), se nós aplicarmos isso na função esponja `Keccak-f[r+c]` e também aplicando juntamento um preenchimento (padding) específico à mensagem de entrada.

Quando restringimos ao caso b=1600, a família Keccak é denotada simplesmente por Keccak[c]; nesse caso r é determinado pela escolha de c. Reusando a própria função veja (DRAFT FIPS 202, pg 27).

KECCAK[c] = [KECCAK-p[1600, 24], pad10*1, 1600 - c]

Assim, dado uma mensagem M e uma saída de tamanho d. Aplicamos M da seguinte forma na função esponja.

KECCAK[c](M, d) = [KECCAK-p[1600, 24], pad10*1, 1600 – c](M, d)

(h) Explique as especificações da função SHA3, conforme Seção 6.

O corpo SHA-3 contém 4 funções hash, e 2 XOFs 3. Explicarei cada uma delas a seguir nos itens abaixo.

i. Funções de hash SHA3.

Antes de mais nada deve-se entender que o SHA3 também utiliza M para Mensagem. porém ele adiciona 2 bits (ou 4 bits) concatenados à M em todas das funções hash (DRAFT FIPS 202, Sec 5.2, pg 28). Nesse caso são 2 bits, como está previsto no documento na seção 6.1, b_0b_1 são os bits a serem concatenados, mais descrições deles adiante.

Na seção 6.2 como podemos ver que o SHA3-224(M) utiliza o Keccak[448]. Com capacidade c de 448 bits. E assim sucessivamente, aumentando as capacidade para 512, 767, 1024. E é exatamente isso que é passado para a função esponja Keccak.

$$SHA3_{224}(M) = f[448](M||b_0b_1,\ 224)$$

$$SHA3_{256}(M) = f[512](M||b_0b_1,\ 256)$$

$$SHA3_{384}(M) = f[768](M||b_0b_1,\ 384)$$

$$SHA3_{512}(M) = f[1024](M||b_0b_1,\ 512)$$

Note que f é a função esponja Keccak. Em cada caso, a capacidade é dobrada em tamanho consecutivamente **dobra** o tamanho (comprimento) da resumo pós "trituração" (STALLINGS, Sec 11.1, pg 314 - Funções de Trituração Criptográficas), isto é: $c = 2 * d$. Os dois bits que são concatenados à mensagem (i.e, 01) suportam o *domain separation*; eles distinguem a mensagem para o SHA3. Para assegurar que a mensagem pode ser dividida em blocos pares de tamanho r (bitrate) a regra do padding é aplicada ou seja, o padrão "bit-a-bit"$pad10^*1$: bit 1, zero ou mais 0's (máximo de r - 1), 1 bit.

Isso é preciso pois está na norma da construção da esponja Keccak, a prova de conceito para o resumo criptográfico usando Keccak existe que a taxa seja codificada dessa maneira no bloco final ("multi rate padding"). O último DRAFT FIPS 202 coloca os bits de concatenação para as funções de resumo criptográfico SHA3 como sendo $b_0,\ b_1 = (0,\ 1)$. Isso provê *domain separation* para os SHAKEs (SHA3+Keccak)[8]. A seguir estão explicadas as funções SHAKE's.

[8]Separação de domínio é uma maneira de separar informação dentro de domínios logicamente definidos

Função	Rate [r]	Capacidade [c]	Vazão [n]
SHA3-224	1152	448	224
SHA3-256	1088	512	256
SHA3-384	832	768	384
SHA3-512	576	1024	512
SHAKE128	1344	256	d
SHAKE256	1088	512	d

Tabela 4: Tabela de parâmetros para a família SHA-3 (Keccak), note que r + c dá 1600 em todos os casos. Para os casos do SHAKE-n, temos que a vazão é entendida para fazer muito mais que as normais dentre 224 à 512. Isto é por que dependendo da aplicação pode ser necessário. Esses são os mesmos valores usando no código keccak.py que acompanha esse artigo.

ii. Funções de Saída (Vazão) Estendida.

As duas SHA3 XOFs 3, SHAKE128 e SHAKE256, são definidas de duas funções intermediárias (descritas abaixo), chamas RawSHAKE128 e RawSHAKE256, o qual são definidas a partir de? Keccak[c=capacity]) (DRAFT FIPS 202, Sec 5.2).

RawSHAKE128(M, d) = KECCAK[256](M || b_0b_1, d)

RawSHAKE256(M, d) = KECCAK[512](M || b_0b_1, d)

Os dois bits que são concatenados à mensagem, isto é.; $b_0b_1 = 11$ (DRAFT FIPS 202, pg 28), suportam *domain separation*. As duas XOFs são (só muda o nome de RawSHAKE128 para SHAKE128) e RawSHAKE256 para SHAKE256):

Nesse caso, os bits 11 são concatenados à mensagem para compatibilidade com o esquema de codificação Sakura. O Sakura-scheme é um hash distribuído em estrutura de árvore (grafo de nós direcional) [1]. Esse esquema flexibiliza o desenvolvimento dos XOFs. Além disso, possibilita a computação de forma paralela, permitindo computar, atualizar resumos criptográficos de mensagens longas com mais eficiência. As funções de vazão-estendida tem o sufixo que permite compatibilidade com o esquema Sakura de árvores hash. Para SHA3-n, o FIPS 202 configura $c = 2n$, assim quem estivesse implementando precisaria criar um complemento em cima dessa equação para ter 2 blocos de saída (como uma máscara), e isso iria ser penoso dado que o SHA3-n obtém seus resultados truncando a saída do Keccak. As funções de vazão-estendida são definidas em dois passos [1]:

$$SHAKE128(M, d) = RawSHAKE128(M||11, d)$$

$$SHAKE256(M, d) = RawSHAKE256(M||11, d)$$

Os dois XOF's SHA-3 podem também ser definidos diretamente a partir do Keccak, da seguinte maneira no código `keccak/keccak.py` as funções SHAKE recebem suas **mensagens** + `'F'` para satisfazer a implementação:

$$SHAKE128(M, d) = KECCAK[256](M||1111, d)$$

$$SHAKE256(M, d) = KECCAK[512](M||1111, d)$$

(i) Apresente a análise de segurança conforme Apêndice A.1.

Function	Largura de Saída	Colisão	Pré Imagem	2ª Pré Imagem
SHA-1	160	< 80	160	160-L(M)
SHA-224	224	112	224	Min(224, 256-L(M))
SHA-512/224	224	112	224	224
SHA-256	256	128	256	256-L(M)
SHA-512/256	256	128	256	256
SHA-384	384	192	384	384
SHA-512	512	256	512	512-L(M)
SHA3-224	224	112	224	224
SHA3-256	256	128	256	256
SHA3-384	384	192	384	384
SHA3-512	512	256	512	512
SHAKE128	d	min(d/2, 128)	>= min(d, 128)	>= min(d, 128)
SHAKE256	d	min(d/2, 256)	>= min(d, 256)	>= min(d, 256)

Tabela 5: Comparativa de Colisão, Pré Imagem e Segunda Pré Imagem em Função de quantos bits na Largura de Saída para cada implementação das famílias $SHA - n$ listadas nessa tabela. Porém vale notar, usando a tabela de parâmetros de cada implementação 4(h)i podemos ver que implicitamente existe a capacidade (c) de cada implementação. **O nível de segurança denota o número de computações que um atacante terá que realizar em ordem de quebrar a função as funções de resumo p.e.: o nível de segurança para 128 bits implica que o adversário deverá realizar 2^{128} computações.**

Vou continuar a partir da explicação no começo desse trabalho, pois no apêndice A.1.1 tem

muita recapitulação sobre as definições das propriedades de colisão, pré imagem e segunda pré imagem (Página 1, Propriedades 2). A primeira coluna da tabela 3 (pg 22) mostra as famílias de hash criptográfico, e colisão, pré imagem, segunda pré imagem em função da vazão (*output size*). Tanto SHA-1, SHA-2 e SHA-3 utilizam funções primitivas simples do tipo (+, and, or, xor, rotl, rotr, etc). A vazão crescer de cima quer dizer que está ligada a tecnologia de software, à implementação e o modelo que descreve matematicamente a evolução dos algoritmos. O SHA-1 apresenta $\leq 2^{80}$ hash's calculados para haver colisão, quer dizer que o SHA-1 está em ritmo de decadência quanto a sua força de resistência a colisões. O SHA-1 tem também um número exorbitante de 2^{160} para quebrar a resistência a pré imagem, e $2^{160} - Length(M)$ para segunda pré imagem. Suponha que a Mensagem tenha um comprimento de 1024 caracteres. Então $2^{160} - 1024$ tentativas para quebrar segunda pré imagem do SHA-1. Agora comparando o SHA-1 em relação ao SHA-224, SHA-512/224 [9], SHA-256, SHA-512/256 [10], SHA-384, SHA-512 e observando que a coluna de cima para baixo lista as funções de resumo criptográfico crescendo em complexidade e resistência. Você poderá ver com os dados abaixo das comparações de resistências, que da família SHA-1 para SHA-2 houve melhoria de resistência em todos.

i. SHA-224/SHA-1:
   - Colisão: 1.4x mais resistente (40%+)
   - Preimage: 1.4x mais resistente (40%+)
   - 2nd Preimage: 1.6x mais resistente (40%+)

ii. SHA-512/224/SHA-1:
   - Colisão: 1.4x mais resistente (40%+)
   - Preimage: 1.4x mais resistente (40%+)
   - 2nd Preimage: 1.4x mais resistente (40%+)

iii. SHA-256/SHA-1:
   - Colisão: 1.6x mais resistente (60%+)
   - Preimage: 1.6x mais resistente (60%+)
   - 2nd Preimage: 1.6x mais resistente (60%+)

iv. SHA-512/256/SHA-1:
   - Colisão: 1.6x mais resistente (60%+)
   - Preimage: 1.6x mais resistente (60%+)
   - 2nd Preimage: 1.6x mais resistente (60%+)

[9] SHA 512 tem 224 bits de output size pois o construção do SHA-512 trunca os 288 bits.
[10] SHA 512 tem 224 bits de output size pois o construção do SHA-512 trunca os 256 bits.

v. SHA-384/SHA-1:
   - Colisão: 2.4x mais resistente (140%+)
   - Preimage: 2.4x mais resistente (140%+)
   - 2nd Preimage: 2.4x mais resistente (140%+)

vi. SHA-512/SHA-1:
   - Colisão: 3.2x mais resistente (220%+)
   - Preimage: 3.2x mais resistente (220%+)
   - 2nd Preimage: 3.2x mais resistente (220%+)

Agora comparando também cada elemento da família SHA-3 com todos os da SHA-2, em colisão, pré imagem, segunda pré imagem. E configurando $d = 512\ bits$, pois a tabela permite d variável para SHAKE128[11] e SHAKE256, e também usando min(d/2, 128) para o SHAKE128, e $\min(d/2, 128)$ para o SHAKE256 respectivamente e assim sucessivamente para pré imagem e depois para segunda pré imagem. Veja que min( ) da o pior caso no sentido de resistência às três propriedades citadas na tabela 5. Finalmente, analisando os dados abaixo, podemos ver que alguns diminuíram suas resistência, e outros aumentaram bem pouco como é o caso do SHA-3 comparado ao SHA-224 com aumentos de colisões 14%, pré imagem 14%, e segunda pré imagem 0% (não aumentou nada a resistência), isso é por causa da barreira computacional, precisa melhorar a tecnologia e o modelo matemático necessita ser ainda mais complexo e também bem formulado para aumentar as resistências da tabela 5.[12]

i. SHA3-224/SHA-224:
   - Colisão: 1.0x mais resistente
   - Preimage: 1.0x mais resistente
   - 2nd Preimage: 1.0x mais resistente

ii. SHA3-256/SHA-224:
   - Colisão: 1.1428571428571428x mais resistente (14%+)
   - Preimage: 1.1428571428571428x mais resistente (14%+)
   - 2nd Preimage: 1.1428571428571428x mais resistente (14%+)

iii. SHA3-384/SHA-224:

[11] Veja que d é variável, pois XOFs é permitido setar a vazão de saída, dependendo a aplicação dos XOFS SHA-3.

[12] Para os casos onde matematicamente temos $2^n - length(M)$ eu considerei length(M) = 0, para facilitar nos cálculos, isso não altera na análise pois digamos que eu compare $\frac{SHA3-512}{SHA-256}$ você pode ver todos as resistências melhoraram 2x mais, a conta seria $\frac{SHA3-512}{SHA-256} - \frac{SHA3-512}{length(M)}$, para length(M) ¿ 0 e quanto maior o texto menor o valor, porém ele está dividindo um número grande 2^{512} pois o caso é apenas para segunda pré imagem que tem esse detalhe a mais na tabela.

- Colisão: 1.7142857142857142x mais resistente (71%+)
- Preimage: 1.7142857142857142x mais resistente (71%+)
- 2nd Preimage: 1.7142857142857142x mais resistente (71%+)

iv. SHA3-512/SHA-224:
- Colisão: 2.2857142857142856x mais resistente (128%+)
- Preimage: 2.2857142857142856x mais resistente (128%+)
- 2nd Preimage: 2.2857142857142856x mais resistente (128%+)

v. SHAKE128/SHA-224:
- Colisão: 1.1428571428571428x mais resistente (14%+)
- Preimage: 0.5714285714285714x menos resistente (42%-)
- 2nd Preimage: 0.5714285714285714x menos resistente (42%-)

vi. SHAKE256/SHA-224:
- Colisão: 2.2857142857142856x mais resistente
- Preimage: 1.1428571428571428x mais resistente
- 2nd Preimage: 1.1428571428571428x mais resistente

vii. SHA3-224/SHA-512/224:
- Colisão: 1.0x mais resistente
- Preimage: 1.0x mais resistente
- 2nd Preimage: 1.0x mais resistente

viii. SHA3-256/SHA-512/224:
- Colisão: 1.1428571428571428x mais resistente
- Preimage: 1.1428571428571428x mais resistente
- 2nd Preimage: 1.1428571428571428x mais resistente

ix. SHA3-384/SHA-512/224:
- Colisão: 1.7142857142857142x mais resistente
- Preimage: 1.7142857142857142x mais resistente
- 2nd Preimage: 1.7142857142857142x mais resistente

x. SHA3-512/SHA-512/224:
- Colisão: 2.2857142857142856x mais resistente
- Preimage: 2.2857142857142856x mais resistente
- 2nd Preimage: 2.2857142857142856x mais resistente

xi. SHAKE128/SHA-512/224:
- Colisão: 1.1428571428571428x mais resistente

- Preimage: 0.5714285714285714x mais resistente
- 2nd Preimage: 0.5714285714285714x mais resistente

xii. SHAKE256/SHA-512/224:
- Colisão: 2.2857142857142856x mais resistente
- Preimage: 1.1428571428571428x mais resistente
- 2nd Preimage: 1.1428571428571428x mais resistente

xiii. SHA3-224/SHA-256:
- Colisão: 0.875x mais resistente
- Preimage: 0.875x mais resistente
- 2nd Preimage: 0.875x mais resistente

xiv. SHA3-256/SHA-256:
- Colisão: 1.0x mais resistente
- Preimage: 1.0x mais resistente
- 2nd Preimage: 1.0x mais resistente

xv. SHA3-384/SHA-256:
- Colisão: 1.5x mais resistente
- Preimage: 1.5x mais resistente
- 2nd Preimage: 1.5x mais resistente

xvi. SHA3-512/SHA-256:
- Colisão: 2.0x mais resistente
- Preimage: 2.0x mais resistente
- 2nd Preimage: 2.0x mais resistente

xvii. SHAKE128/SHA-256:
- Colisão: 1.0x mais resistente
- Preimage: 0.5x mais resistente
- 2nd Preimage: 0.5x mais resistente

xviii. SHAKE256/SHA-256:
- Colisão: 2.0x mais resistente
- Preimage: 1.0x mais resistente
- 2nd Preimage: 1.0x mais resistente

xix. SHA3-224/SHA-512/256:
- Colisão: 0.875x mais resistente
- Preimage: 0.875x mais resistente

- 2nd Preimage: 0.875x mais resistente

xx. SHA3-256/SHA-512/256:
- Colisão: 1.0x mais resistente
- Preimage: 1.0x mais resistente
- 2nd Preimage: 1.0x mais resistente

xxi. SHA3-384/SHA-512/256:
- Colisão: 1.5x mais resistente
- Preimage: 1.5x mais resistente
- 2nd Preimage: 1.5x mais resistente

xxii. SHA3-512/SHA-512/256:
- Colisão: 2.0x mais resistente
- Preimage: 2.0x mais resistente
- 2nd Preimage: 2.0x mais resistente

xxiii. SHAKE128/SHA-512/256:
- Colisão: 1.0x mais resistente
- Preimage: 0.5x mais resistente
- 2nd Preimage: 0.5x mais resistente

xxiv. SHAKE256/SHA-512/256:
- Colisão: 2.0x mais resistente
- Preimage: 1.0x mais resistente
- 2nd Preimage: 1.0x mais resistente

xxv. SHA3-224/SHA-384:
- Colisão: 0.5833333333333334x mais resistente
- Preimage: 0.5833333333333334x mais resistente
- 2nd Preimage: 0.5833333333333334x mais resistente

xxvi. SHA3-256/SHA-384:
- Colisão: 0.6666666666666666x mais resistente
- Preimage: 0.6666666666666666x mais resistente
- 2nd Preimage: 0.6666666666666666x mais resistente

xxvii. SHA3-384/SHA-384:
- Colisão: 1.0x mais resistente
- Preimage: 1.0x mais resistente
- 2nd Preimage: 1.0x mais resistente

xxviii. SHA3-512/SHA-384:
- Colisão: 1.3333333333333333x mais resistente
- Preimage: 1.3333333333333333x mais resistente
- 2nd Preimage: 1.3333333333333333x mais resistente

xxix. SHAKE128/SHA-384:
- Colisão: 0.6666666666666666x mais resistente
- Preimage: 0.3333333333333333x mais resistente
- 2nd Preimage: 0.3333333333333333x mais resistente

xxx. SHAKE256/SHA-384:
- Colisão: 1.3333333333333333x mais resistente
- Preimage: 0.6666666666666666x mais resistente
- 2nd Preimage: 0.6666666666666666x mais resistente

xxxi. SHA3-224/SHA-512:
- Colisão: 0.4375x mais resistente
- Preimage: 0.4375x mais resistente
- 2nd Preimage: 0.4375x mais resistente

xxxii. SHA3-256/SHA-512:
- Colisão: 0.5x mais resistente
- Preimage: 0.5x mais resistente
- 2nd Preimage: 0.5x mais resistente

xxxiii. SHA3-384/SHA-512:
- Colisão: 0.75x mais resistente
- Preimage: 0.75x mais resistente
- 2nd Preimage: 0.75x mais resistente

xxxiv. SHA3-512/SHA-512:
- Colisão: 1.0x mais resistente
- Preimage: 1.0x mais resistente
- 2nd Preimage: 1.0x mais resistente

xxxv. SHAKE128/SHA-512:
- Colisão: 0.5x mais resistente
- Preimage: 0.25x mais resistente
- 2nd Preimage: 0.25x mais resistente

xxxvi. SHAKE256/SHA-512:

- Colisão: 1.0x mais resistente
- Preimage: 0.5x mais resistente
- 2nd Preimage: 0.5x mais resistente

(j) Gere os seus próprios exemplos (diferente do NIST) conforme Apêndice A.2.

Não consegui entender essa questão.

Algoritmo 10: h2b(H) (DRAFT FIPS 202, pg 33)

```
def h2b(h):
  return bin(int(h, 16))[2:].zfill(len(h) * 4)
```

Algorithm 11: b2h(S) (DRAFT FIPS 202, pg 33)

```
def bintohex(s):
  t = ''.join(chr(int(s[i:i+8],2)) for i in xrange(0, len(s), 8))
  return binascii.hexlify(t)
```

5. Apresente uma implementação do SHA3.

O código está comentado no arquivo `keccak/keccak.py`. E as 6 funções do SHA-3 estão contidas no arquivo `keccak/demo.py`. Mais 6 funções que processam blocos de dados para responder as questões abaixo.

(a) Descreva a implementação.

O anexo ao final desse pdf, contêm o código descrito 2.2.

(b) Mostre na implementação onde se da cada passo importante do cálculo do hash.

Em anexo no final desse `pdf` esta o código keccak/keccak.py. Em negrito está selecionado a parte importante para o cálculo do resumo criptográfico. As partes cruciais para o calculo do hash são 2.2:

i. Preenchendo (pad10*1)

ii. Absorvendo da Esponja (absorving)
iii. Espremendo da Esponja (squeezing)

(c) Execute a implementação passo a passo, mostrando o maior número possível de saídas.

As saídas estão em anexo ao final do `pdf` 2.1. Eu rodei o código para **M** = `'abc'` a fim de certificar que estava rodando corretamente. Os resultados bateram. Basta ir até o diretório do código keccak dentro do zip e rodar python3 demo.py. Ele lhe perguntará o que você deseja fazer 's' para verificar os tempos de processamento de 65 KB de dados (gerados internamente).

```
Retirado do site: http://www.di-mgt.com.au/sha_testvectors.html
Input message: "abc" (length 24 bits).

SHA-3-224 e642824c3f8cf24a d09234ee7d3c766f
          c9a3a5168d0c94ad 73b46fdf

SHA-3-256 3a985da74fe225b2 045c172d6bd390bd
          855f086e3e9d525b 46bfe24511431532

SHA-3-384 ec01498288516fc9 26459f58e2c6ad8d
          f9b473cb0fc08c25 96da7cf0e49be4b2
          98d88cea927ac7f5 39f1edf228376d25

SHA-3-512 b751850b1a57168a 5693cd924b6b096e
          08f621827444f70d 884f5d0240d2712e
          10e116e9192af3c9 1a7ec57647e39340
          57340b4cf408d5a5 6592f8274eec53f0
```

Rodando agora meu programa. Para conferir. Lembrando que a execução passo a passo, mostrando o maior número possível de saídas esta em anexo nesse pdf ao final dele 2.1.

```
/usr/bin/python3.4 keccak/demo.py
Verificar tempos de processamento de blocos? (s/n): n

Mensagem (para testar todos os digests sha3: abc
```

```
Comparar com www.di-mgt.com.au/sha_testvectors.html

'SHA3_224' - 0.00282 sec

Resumo: e642824c3f8cf24a d09234ee7d3c766f c9a3a5168d0c94ad 73b46fdf

'SHA3_256' - 0.00477 sec

Resumo: 3a985da74fe225b2 045c172d6bd390bd 855f086e3e9d525b 46bfe24511431532

'SHA3_384' - 0.00447 sec

Resumo: ec01498288516fc9 26459f58e2c6ad8d f9b473cb0fc08c25 96da7cf0e49be4b2
98d88cea927ac7f5 39f1edf228376d25

'SHA3_512' - 0.00381 sec

Resumo: b751850b1a57168a 5693cd924b6b096e 08f621827444f70d 884f5d0240d2712e
10e116e9192af3c9 1a7ec57647e39340 57340b4cf408d5a5 6592f8274eec53f0

'SHAKE128' - 0.00168 sec

Resumo: 5881092dd818bf5c f8a3ddb793fbcba7 4097d5c526a6d35f 97b83351940f2cc8
44c50af32acd3f2c dd066568706f509b c1bdde58295dae3f 891a9a0fca578378

'SHAKE256' - 0.00167 sec

Resumo: 483366601360a877 1c6863080cc4114d 8db44530f8f1e1ee 4f94ea37e78b5739
d5a15bef186a5386 c75744c0527e1faa 9f8726e462a12a4f eb06bd8801e751e4
```

6. Compare o SHA3, em termos de performance, com os hashes da família SHA2. Use um mesmo computador e implementações padrão para esta comparação. O resultado da comparação deve ser em termos de tamanhos de arquivos dos quais hashes são calculados e quanto tempo para hash (uma média) demora para ser calculado.

   Configurações para os ensaios com os resumos criptográficos eu não utilizei openssl speed, eu criei um programa para calcular o tempo processando blocos de dados. O conjunto de blocos de strings

(1024 bytes cada) é o mesmo para todos os ensaios.:

(a) Fluxo: `1024 bytes/bloco` × `64 blocos`
(b) Total: `65,536 KB` de informação
(c) Sistema Operacional: `Linux 3.13.0-55-generic Ubuntu`
(d) Data do Ensaio: Terça Junho 18 00:27:10 UTC 2015
(e) Arquitetura: `x86 64 GNU/Linux`
(f) Bibliotecas: hashlib (para família SHA-2)
(g) Linguagem: python3.4
(h) python 3 demo.py
(i) Função Esponja: Keccak (para família SHA-3)
(j) codificação: base 16

Tempos da familia SHA-3, processando os blocos para 64 bytes/bloco * 1024 blocos – totalizando 65 KB de dados

- 0.00025 sec 1024 blocos processados com sha3 256
- 1.67190 sec 1024 blocos processados com sha3 256
- 1.67232 sec 1024 blocos processados com sha3 384
- 1.64219 sec 1024 blocos processados com sha3 512
- 1.67886 sec 1024 blocos processados com shake128
- 1.68294 sec 1024 blocos processados com shake256

Tempos da familia SHA-2, processando os blocos

- 0.00161 segundos 1024blocos processados com sha224
- 0.00166 segundos 1024blocos processados com sha256
- 0.00179 segundos 1024blocos processados com sha384
- 0.00173 segundos 1024 blocos processados com sha512

7. Apresente uma crítica ao SHA3.

   (a) O que ele é melhor ou diferente em relação a outros hash?

   Suas 2 funções de vazão estendida são o diferencial (para melhor adaptação a implementações de hardware e software), as chamadas XOF's SHA-3 (e**X**tended **O**utput **F**unctions **S**ecure **H**ash **A**lgorithm) 3. Porém fazendo a análise de tempos de processamento de blocos de string no item anterior a família SHA-2 processa mais rapidamente, isso me diz que ela é mais adaptada à software (com optimizações para software). Porém o SHA-3 é mais adaptado a hardware, ou seja, o foco do SHA-3 é para ser implementado em hardware (p.e.: Sistemas Embarcados).

   Pela modelagem, a vazão de um XOF não afeta os bits que ele produz, isto é, não precisa ter valor para vazão. Teoricamente a vazão pode ser uma string infinita, e a aplicação / protocolo / sistema que utilizar a função simplesmente computa o valor desejado para tamanho inicial em bits da string. Em termos de retro-compatibilidade em padrões criptográficos mais antigos, essas funções XOFS irão se comportar como uma função hash quando foram processados entradas em fluxo (DRAFT FIPS 202, pg 31).

   (b) Quanto tempo você acha (e por que) o SHA3 será considerado seguro?

   O pesquisador Razvan Rosie publicou em 10 Dezembro de 2014 um artigo explorando *On quantum preimage attacks* onde ele propõe um ataque à pré imagem do SHA-3 (que por sua vez usa o *Keccak-f*) contra as funções de hash criptográfico. Baseado em aumentar a velocidade dos cálculos, usando computação quântica (inglês.: *Quantum preimage attack*). Resistência a pré imagem é uma propriedade **fundamental**, e o artigo de Rosie aborda um algoritmo capaz de identificar um valor de pré imagem x do espaço de entrada de $h \in \mathbb{Z}_2^{inf} \rightarrow \mathbb{Z}_2^N$ [8], atingindo o objetivo que é quebrar a regra de resistência a pré imagem (propriedade de uma via): $resumo_y = HashFunction_x(Messagem)$; $resumo_y \rightarrow Mensagem$ **mas nunca** $Mensagem \rightarrow resumo_y$. Isto é, do $resumo_y$ aponta para uma mensagem, mas **nunca** o contrário (isso é resistência a pré imagem).

   O por quê de eu ter trazido essa notícia logo acima do artigo do Razvan Rosie é que antes mesmo do SHA-3 estar vinculado, ativamente, ao mercado de resumos criptográfico. Já estão lançando técnicas bastante modernas para tentar quebrar o algoritmo SHA-3 de alguma forma. Forçando a pensar que ele pode ter uma vida curta. Porém a tabela a seguir mostra que não é verdade isso. Que o SHA-3 vai perdurar por um bom tempo ainda.

Funções	Ininterrupto	Enfraquecido	Quebrado	Depreciado
SHA-1	1995-2003 (9)	2004-2012 (9)	—	—
SHA-2 (Família)	2000-2007 (8)	2008-2012 (5)	—	—
SHA-3 (Keccak)	2009-2012 (4)	—	—	—

Tabela 6: Tabela comparativa das família SHA-1, SHA-2 e SHA-3 e sua estabilidade no mercado de resumos criptográficos, a tabela foi produzida a partir dos dados do site http://valerieaurora.org/hash.html Copyright 2007 - 2012 Valerie Aurora, licensed CC-BY-SA.

Segundo o site http://valerieaurora.org/hash.html (Valerie Aurora, 2007 — 2012) vemos que o SHA-3 está com estado Ininterrupto (Unbroken) desde 2009 até 2012 são 4 anos. Se o SHA-2 está desde 2000 até 2007 Ininterrupto e desde 2008 até 2012, Enfraquecido (Weakened) são 13 anos no total resistindo no mercado de resumos criptográficos, sendo que o SHA-2 é de uso oficial no Brasil. Agora o SHA-1 ainda não passou para o estado de Quebrado (Broken) e ele resiste desde 1995 até 2012, são 18 anos. Se somarmos 18 anos (SHA-1) mais 12 anos (SHA-2) dá 30 anos, talvez isso seja um exagero para o SHA-3 (30 anos de resistência). Daqui a 15 anos nós estaremos embarcando no mundo da computação quântica. Pode chutar um valor de aproximadamente 15 anos para o SHA-3 ser quebrado.

Referências

[1] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "SAKURA: a flexible coding for tree hashing,". Disponível em: pdf.

[2] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Cryptographic sponge functions," Janeiro 2011, Disponível em: pdf.

[3] NIST Computer Security Division (CSD). SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. 2014.

[4] Federal Information Processing Standards Publication 180-4, Secure Hash Standard (SHS), Information Technology Laboratory, National Institute of Standards and Technology, Março 2012, Disponível em: pdf.

[5] The SHA-3 Cryptographic Hash Algorithm Competition, Novembro 2007-Outubro 2012, Diponível na: web.

[6] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak Specifications," Submission to NIST (Round 3), Janeiro 2011, Disponível em: pdf.

[7] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "The Keccak reference, Version 3.0," Janeiro 2011, Disponível em: pdf.

[8] Rosie, Razvan. *On quantum preimage attacks*. Disponível em: arXiv:1412.3164

[9] Keccak: Known-answer and Monte Carlo test results, Version 3.0, January 14, 2011, [download disponível em: KeccakKAT-3.zip].

2 Anexos

2.1 Anexos

```
/usr/bin/python3.4 keccak/demo.py
Verificar tempos de processamento de blocos? (s/n): n

Mensagem (para testar todos os digests sha3: abc

Comparar com www.di-mgt.com.au/sha_testvectors.html
Gerando a função Keccak com (r=1152, c=448 (i.e. w=64))
String pronta para ser absorvida: 6162630600000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000080 (será
completada com 56 x '00')
Valor atual do State Array  antes da primeira rodada
['0000000006636261', '0000000000000000', '0000000000000000',
'0000000000000000', '0000000000000000']
['0000000000000000', '0000000000000000', '0000000000000000',
'0000000000000000', '0000000000000000']
['0000000000000000', '0000000000000000', '0000000000000000',
'0000000000000000', '0000000000000000']
['0000000000000000', '0000000000000000', '8000000000000000',
'0000000000000000', '0000000000000000']
['0000000000000000', '0000000000000000', '0000000000000000',
'0000000000000000', '0000000000000000']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(0,1[bits]) = 0
Rotacionar Esquerda(9223372036854775808,1[bits]) = 1
Rotacionar Esquerda(0,1[bits]) = 0
Rotacionar Esquerda(0,1[bits]) = 0
Rotacionar Esquerda(107176545,1[bits]) = 214353090
Depois da rotacao de theta A[ 4 ] ['0b0', '0b0', '0b0', '0b0', '0b0']
Depois de theta ['0b11001100011011000010011000010',
'0b11001100011011000010011000010', '0b11001100011011000010011000010',
'0b11001100011011000010011000010', '0b11001100011011000010011000010']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
```

```
Rotacionar Esquerda(107176545,0[bits]) = 107176545
Rotacionar Esquerda(0,36[bits]) = 0
Rotacionar Esquerda(0,3[bits]) = 0
Rotacionar Esquerda(0,41[bits]) = 0
Rotacionar Esquerda(0,18[bits]) = 0
Rotacionar Esquerda(107176544,1[bits]) = 214353088
Rotacionar Esquerda(107176544,44[bits]) = 3901806127163113574
Rotacionar Esquerda(107176544,10[bits]) = 109748781056
Rotacionar Esquerda(107176544,45[bits]) = 7803612254326227148
Rotacionar Esquerda(107176544,2[bits]) = 428706176
Rotacionar Esquerda(0,62[bits]) = 0
Rotacionar Esquerda(0,6[bits]) = 0
Rotacionar Esquerda(0,43[bits]) = 0
Rotacionar Esquerda(9223372036854775808,15[bits]) = 16384
Rotacionar Esquerda(0,61[bits]) = 0
Rotacionar Esquerda(9223372036854775808,28[bits]) = 134217728
Rotacionar Esquerda(9223372036854775808,55[bits]) = 18014398509481984
Rotacionar Esquerda(9223372036854775808,25[bits]) = 16777216
Rotacionar Esquerda(9223372036854775808,21[bits]) = 1048576
Rotacionar Esquerda(9223372036854775808,56[bits]) = 36028797018963968
Rotacionar Esquerda(214353090,27[bits]) = 28769984729579520
Rotacionar Esquerda(214353090,20[bits]) = 224765505699840
Rotacionar Esquerda(214353090,39[bits]) = 7161393010100404230
Rotacionar Esquerda(214353090,8[bits]) = 54874391040
Rotacionar Esquerda(214353090,14[bits]) = 3511961026560
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1100110001101100010011000010000000000000000', '0b0', '0b0',
'0b1000000000000000000000000000000000000000000000000000000000000000',
'0b1100110001101100010011000000']
Antes do modulo chi ['0b1100110001101100010011000010',
'0b1100110001101100010011000010', '0b1100110001101100010011000010',
'0b1100110001101100010011000010', '0b1100110001101100010011000010']
Depois do modulo chi
['0b1101100010011000000011001100011011000100110000100000000000110',
'0b1100110001101100010001000010000000000000000000000', '0b0',
'0b1000000000000000000000000000000000000000000000000000000000000000',
'0b1000000000000000000000000011001100011011000100110000000']
Antes do modulo iota ['0b1100110001101100010011000001',
```

```
'0b10000000000000000000000000000', '0b11011100011011000010011000000',
'0b11001100011011000111111100111011000100110000000000000000',
'0b11000110010001001100001000000000000000000000000000000000000110']
Depois do modulo iota ['0b11001100011011000100110000',
'0b10000000000000000000000000000', '0b11011100011011000010011000000',
'0b11001100011011000111111100111011000100110000000000000000',
'0b11000110010001001100001000000000000000000000000000000000000110']
Valor atual do State Array  após a rodada 1/24
['0000000006636260', '3626000000100066', '00000331B1208000',
'0000000006536261', '36260331B1308006']
['0000000008000000', '6C4CCC6C4C2000CC', '0000000000000000',
'6C4C0000080000CC', '0000CC6C44200000']
['000000000DC6C4C0', '0000000CC6C4C200', '0000000001000000',
'0000000CCA0206C0', '0000000000000000']
['0066363F9D898000', '0000000000004000', '008000198D898000',
'0066362610004000', '0080000000000000']
['6322610000000006', '0040000000000000', '63626100198D8986',
'0000000000000000', '00400000198D8980']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(6497230127455568554,1[bits]) = 12994460254911137108
Rotacionar Esquerda(7197423079036062086,1[bits]) = 14394846158072124172
Rotacionar Esquerda(7794101662696678509,1[bits]) = 15588203325393357018
Rotacionar Esquerda(3956077325000182150,1[bits]) = 7912154650000364300
Rotacionar Esquerda(7152938038919440038,1[bits]) = 14305876077838880076
Depois da rotacao de theta A[ 4 ]
['0b11011000100110000000110011000110110001001100001000000000000110',
'0b11001100011011000100010000100000000000000000000000', '0b0',
'0b10000000000000000000000000000000000000000000000000000000000000',
'0b10000000000000000000000001100110001101100010011000000']
Depois de theta
['0b10011100100001001001101101100100010110010011100111101001001001111',
'0b10101010101000100101010000111001101011000010100101101001001000001',
'0b10101010101000101001100001010101111010000000100101101001001000001',
'0b10101010001000101001100001010101111010000000100101101001001000001',
'0b10101010111000101001100001010101111100011000010011100000101000001']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(9417967577568079538,0[bits]) = 9417967577568079538
Rotacionar Esquerda(9417967577339268306,36[bits]) = 16755646337891353111
```

```
Rotacionar Esquerda(9417967577393711122,3[bits]) = 1556764324311482516
Rotacionar Esquerda(9427548749371903186,41[bits]) = 18093674421696022217
Rotacionar Esquerda(16253832577984761044,18[bits]) = 15741177804967085636
Rotacionar Esquerda(10567295684220106188,1[bits]) = 2687847294730660761
Rotacionar Esquerda(14469044171912328550,44[bits]) = 6005106505017211300
Rotacionar Esquerda(11853636272929241002,10[bits]) = 165942978657823378
Rotacionar Esquerda(11853636327786771882,45[bits]) = 12589061311145310924
Rotacionar Esquerda(11871650726296237482,2[bits]) = 10593114757765846698
Rotacionar Esquerda(9403132312045374064,62[bits]) = 2350783078011343516
Rotacionar Esquerda(9403129222065998448,6[bits]) = 11504459853518248992
Rotacionar Esquerda(9403129222082775664,43[bits]) = 13137989759219640602
Rotacionar Esquerda(9439157982804789872,15[bits]) = 5770896659302596991
Rotacionar Esquerda(16221052392263336950,61[bits]) = 15862689604315080830
Rotacionar Esquerda(1022313352942975211,28[bits]) = 4590539724143394761
Rotacionar Esquerda(7089787950814108230,55[bits]) = 2535863033418911565
Rotacionar Esquerda(1022313401191013450,25[bits]) = 4498595992955543545
Rotacionar Esquerda(1029576872546982538,21[bits]) = 6250332020216547641
Rotacionar Esquerda(1022313352973425290,56[bits]) = 9947941388769107610
Rotacionar Esquerda(11278310222138566951,27[bits]) = 2506762581069538523
Rotacionar Esquerda(12295482539353139489,20[bits]) = 4871419898906716709
Rotacionar Esquerda(12295557427407448353,39[bits]) = 339055117826009844
Rotacionar Esquerda(12259528630388484385,8[bits]) = 2492836848828228010
Rotacionar Esquerda(12313571826076016801,14[bits]) = 11967608341802805944
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1010011000010101011111000110000100111000001010000110101010111000',
'0b1101110000100011100110000010011010100111011110110110100001111110',
'0b1101101001110011111100101110100000011001101010011100001100100100',
'0b1000101000001110001011111111110010010011111111010001111010011010',
'0b1001001100000010010011011011111101011001100101001101011010101010']
Antes do modulo chi
['0b1001110010000100100110110110010001011001001110011111010010010111',
'0b1010101010100010010101000011100110101100001010010110100100100001',
'0b1010101010100010100110000101010111101000000010010110100100100001',
'0b1010101000100010100110000101010111101000000010010110100100100001',
'0b1010101011100010100110000101010111110001100001001110000010100001']
Depois do modulo chi
['0b1111011101010001010101000110000100111000110010000111101110111100',
'0b1001110000101001100110000011011010100101011000110110100001011010',
```

```
'0b1000000110100111110110011010000001000100100001100010010011001 00',
'0b1001111100001110001011111101110010010000111011000100011110011110',
'0b1001000000100010010111001000000100010000101101101101101111101011']
Antes do modulo iota
['0b1001101011001011010111100011111100111100010110111011101010100 0',
'0b1010111011010011100011111011111001010101100110101111110101100 1',
'0b1010000101100010110111011111010110011100110001010000100000 0',
'0b1000101100010011001111000010111011100000101110101001101100 1011',
'0b1001000001101100101000110000000011010011011001101100100010 1100']
Depois do modulo iota
['0b1001101011001011010111100011111100111100010110011011100010 1010',
'0b1010111011010011100011111011111001010101100110101111110101100 1',
'0b1010000101100010110111011111010110011100110001010000100000 0',
'0b1000101100010011001111000010111011100000101110101001101100 1011',
'0b1001000001101100101000110000000011010011011001101100100010 1100']
Valor atual do State Array  após a rodada 2/24
['26B2D78FCF166E2A', '13FA4F44C5B07985', '1653D833DD298B9A',
'561FA0479656CD3B', 'F751546138C87BBC']
['2BB4E3EF9566BF59', 'E9BF82868073106D', '459834C1008440A6',
'8D21131802ED6D4D', '9C299836A563685A']
['050B16EFACE62840', '9F3849889CD3BC22', 'E60D9928A60ED9BD',
'079457E785C14833', '40D3ECD022431264']
['22C4CF0BB82EA6CB', '47529E850B1538A2', '8845A452951BB412',
'70D797EE8938613E', '9F0E2FDC90EC479E']
['241B28C034D9B22C', 'D83814FEE379CB44', '04B6D86F0058BAD6',
'DB8405458E8A46DD', '90225C8110B6DBEB']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(18020888038392735276,1[bits]) = 17595032003075918937
Rotacionar Esquerda(4122211925677317189,1[bits]) = 8244423851354634378
Rotacionar Esquerda(8645070784073093030,1[bits]) = 17290141568146186060
Rotacionar Esquerda(2632164704057990647,1[bits]) = 5264329408115981294
Rotacionar Esquerda(1068132959541915092,1[bits]) = 2136265919083830184
Depois da rotacao de theta A[ 4 ]
['0b1111011101010001010101000110000100111000110010000111101110111100',
'0b1001110000101001100110000011011010100101011000110110100001011010',
'0b1000000110100111110110011010000001000100100001100010010011001 00',
'0b1001111100001110001011111101110010010000111011000100011110011110',
'0b1001000000100010010111001000000100010000101101101101101111101011']
```

```
Depois de theta
['0b1001110100001101101010001111101011011010110000110110111110110010',
'0b1111011001110101011001001010110101000111011010000111110001010100',
'0b1010101000111100010000010010111100000001001000000011001101010',
'0b1111010101010010110100110100011101110010111001110101001110010000',
'0b1111101001111111010100000000110101111001010111101110011111100101']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(17733935322147635076,0[bits]) = 17733935322147635076
Rotacionar Esquerda(18094808370865008375,36[bits]) = 9423482602043131260
Rotacionar Esquerda(15393964710557448686,3[bits]) = 12471253242202279798
Rotacionar Esquerda(17468760039776089957,41[bits]) = 14478734035830007754
Rotacionar Esquerda(17632269257973916546,18[bits]) = 11376556985740415689
Rotacionar Esquerda(8017139402655509723,1[bits]) = 16034278805311019446
Rotacionar Esquerda(10738644251967669555,44[bits]) = 3193959578463163005
Rotacionar Esquerda(16393277479859677564,10[bits]) = 179032300617855886
Rotacionar Esquerda(4317342420652322300,45[bits]) = 11366952956671939990
Rotacionar Esquerda(11853688028161121818,2[bits]) = 10521263965225384042
Rotacionar Esquerda(267465157419217658,62[bits]) = 9290238326209580222
Rotacionar Esquerda(5800028110211119558,6[bits]) = 2266917579320619412
Rotacionar Esquerda(17575433059461980381,43[bits]) = 1947507356201516503
Rotacionar Esquerda(11358155524709682546,15[bits]) = 2531802522964283088
Rotacionar Esquerda(1248405809263870902,61[bits]) = 13991108781440147574
Rotacionar Esquerda(2748337151796308624,28[bits]) = 4646939286126608609
Rotacionar Esquerda(18238097773283527398,55[bits]) = 8322244599075157029
Rotacionar Esquerda(8624386363267248024,25[bits]) = 7505399271193010163
Rotacionar Esquerda(66491578830374549,21[bits]) = 40090741071455266663
Rotacionar Esquerda(12375798412536144246,56[bits]) = 8551139059024465761
Rotacionar Esquerda(11316887234017456050,27[bits]) = 15480590995315780935
Rotacionar Esquerda(17759211400995568724,20[bits]) = 5392064979372304214
Rotacionar Esquerda(3066687788797134442,39[bits]) = 2594976175806227936
Rotacionar Esquerda(17677423791206519696,8[bits]) = 5968192490028896501
Rotacionar Esquerda(18050040394149646309,14[bits]) = 12107572109983186591
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1010100000000110101111001010111101110011111110010111111010011111',
'0b1100001000101010011001110100111100000011100101101010000001110110',
'0b1001110111100001101001100100110110001110000010111101001011001001',
'0b1110110101010111011111111010101100010110000011111000011101100001',
'0b1001001000000001100001001110101010001001101100000011110000110101010']
```

```
Antes do modulo chi
['0b1001110100001101101010001111101011011010110000110110111110110010',
'0b1111011001110101011001001010110101000111011010000111110001010100',
'0b1010101000111100010000010010111100000001001000000001100110101 0',
'0b1111010101010010110100110100011101110010111001110101001110010000',
'0b1111101001111110101000000001101011110010101111011100111111100101']
Depois do modulo chi
['0b1010000001000110100111011110111100010110111111001001111101110 0110',
'0b1100100010101010001001110100100111000110100110111000011101100000',
'0b1001110010010001001000100100110010010100010101111111001011001001',
'0b1110110101010110101101110101001001101110000110110001111010110 01',
'0b1110000100010001000010001000001110010110111010001100100001101011']
Antes do modulo iota
['0b1110010100011111010111101011100010010000110001011001001000000110',
'0b1110010101111111110110110100100010000101001000010110010001100000 1',
'0b1011111010001101001110111100011011100100101000100101011111010101',
'0b1101011011101110000110111111110111011001100110101011111110000101',
'0b1000010011101100101111101000100001110010011100010000110101111110']
Depois do modulo iota
['0b1100101000111110101111010111000100100001100010100010010100011100',
'0b1110010101111111110110110100100010000101001000010110010001100000 1',
'0b1011111010001101001110111100011011100100101000100101011111010101',
'0b1101011011101110000110111111110111011001100110101011111110000101',
'0b1000010011101100101111101000100001110010011100010000110101111110']
Valor atual do State Array  após a rodada 3/24
['651F5EB890C5128C', '08F22970E560D27D', '9302479862809FCF',
'61BA17B852A49C87', 'A0469DEF16F93EE6']
['E57FB6910A42C8C1', '5A7973C7854F77D6', 'EF12AABD8FF1EF16',
'9DEA9FDD49619517', 'C8AA2749C69B8760']
['BE8D3BC6E4A257D5', '0DA6F2A7DD6EF190', 'E50828D238E71DFB',
'10D756E0A77399C3', '9C91224C9457F2C9']
['D6EE1BFDDCCD5FC5', 'A3C42F6DB7C2C92C', '56F537EDD9A3F2AF',
'A376C4AF86592ED6', '76AB5BA9370D8F59']
['84ECBE8872710D7E', 'BB9247BE3DC8F62F', '3602350447E83DC0',
'C8024DECFB10635E', 'E111088396E8C86B']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(5150922543507139384,1[bits]) = 10301845087014278768
Rotacionar Esquerda(18009832367813665357,1[bits]) = 17572920661917779099
```

```
Rotacionar Esquerda(9796270125630873051,1[bits]) = 1145796177552194487
Rotacionar Esquerda(7189939359470062717,1[bits]) = 14379878718940125434
Rotacionar Esquerda(7849623082826850083,1[bits]) = 15699246165653700166
Depois da rotacao de theta A[ 4 ]
['0b1010000001000110100111011110111100010110111110010011111011100110',
'0b1100100010101010001001110100100111000110100110111000011101100000',
'0b1001110010010001001000100100110010010100010101111111001011001001',
'0b1110110101010110101101110101001001101110000110110001111101011001',
'0b1110000100010001000010001000001110010110111010001100100001101011']
Depois de theta
['0b1111111001101011001001110001110001110110001101010101110101111011',
'0b1001011010000111100111011011101010100110010101111110010011111101',
'0b1100001010111100100110001011111111110100100110111001000101010100',
'0b1010001000011011100001010110100101010111110000011110110011000100',
'0b1011111100111100101100100111000011110110001001001010101111110110']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(9813085194679470209,0[bits]) = 9813085194679470209
Rotacionar Esquerda(598975277976261324,36[bits]) = 1171266583469739384
Rotacionar Esquerda(6034103872193334744,3[bits]) = 11379342830127574722
Rotacionar Esquerda(4313975358137009608,41[bits]) = 1660579731556497071
Rotacionar Esquerda(7628241038640404339,18[bits]) = 15221011013631321971
Rotacionar Esquerda(10935533564984445381,1[bits]) = 3424323056259339147
Rotacionar Esquerda(14216046950761753710,44[bits]) = 16052777769892191270
Rotacionar Esquerda(10562640978044021288,10[bits]) = 6352334323280552522
Rotacionar Esquerda(4392372807512642196,45[bits]) = 743806003744177692
Rotacionar Esquerda(2639882443454836119,2[bits]) = 10559529773819344476
Rotacionar Esquerda(15825412518785077056,62[bits]) = 3956353129696269264
Rotacionar Esquerda(12074086774465707929,6[bits]) = 16425046543713691241
Rotacionar Esquerda(12507982277786193268,43[bits]) = 11415399566631822434
Rotacionar Esquerda(2191098677648892448,15[bits]) = 3193534321332850484
Rotacionar Esquerda(9124110648032947535,61[bits]) = 17281414895499976105
Rotacionar Esquerda(6906911214892820016,28[bits]) = 9020026582692897906
Rotacionar Esquerda(11784459260717182880,55[bits]) = 15010996081882598935
Rotacionar Esquerda(3366166389668586356,25[bits]) = 18324615024688852492
Rotacionar Esquerda(11319397647424722017,21[bits]) = 9655933734778806994
Rotacionar Esquerda(17753785062877157865,56[bits]) = 16858770133739072985
Rotacionar Esquerda(18332789711337512315,27[bits]) = 16407082847538731320
Rotacionar Esquerda(10846811652525384957,20[bits]) = 15828575433526962297
```

```
Rotacionar Esquerda(14032258489189634388,39[bits]) = 5604917071433195514
Rotacionar Esquerda(2920269186531978436,8[bits]) = 9719148803804415016
Rotacionar Esquerda(13780085158082554870,14[bits]) = 32145118933767740 95
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1011001001110000111101100010010010101011111101101010111111001111',
'0b1110111111010011111010110100100101011110011010111001110110101001',
'0b1101001100111011111001001101111101011101110011011010011101110011',
'0b1110100111111011001100010000111010111001101111011011011101011001',
'0b1001001010001010111111000110000111101011101010011011001011100']
Antes do modulo chi
['0b1111111001101011001001110001110001110110001101010101110101111011',
'0b1001011010000111100111011011101010100110010101111110010011111101',
'0b1100001010111100100110001011111111110100100110111001000101010100',
'0b1010001000011011100001010110100101011111000001111011001100010 0',
'0b1011111100111100101100100111000011110110001001001010101111110110']
Depois do modulo chi
['0b1111010010111001101010110001001101010101100100110111011111101001',
'0b1101101010100011000111001010101000101000110101111010101101000 00',
'0b1001101001011101110001001110111001100100001101100001100010011',
'0b1111001101101100110011000011101011100110111011111101001100110 01',
'0b1010010100110101111100100100011111010111100100100111010010110 11']
Antes do modulo iota
['0b1000100000000011000010101110101100011100110001011001010001100000 1',
'0b1111001011011000001101001100010001101011101100110100010111100 00',
'0b1100111000101110100011101011010110110010010100100110111000 1111',
'0b1010101110011001101010001100001011101110111100110111110110011 1010',
'0b1110110110111111100000001101011110101110000101010000100011100 0']
Depois do modulo iota
['0b1000000001100001010111010110101110011000101110101000110000 01',
'0b1111001011011000001101001100010001101011101100110100010111100 00',
'0b1100111000101110100011101011010110110010010100100110111000 1111',
'0b1010101110011001101010001100001011101110111100110111110110011 1010',
'0b1110110110111111100000001101011110101110000101010000100011100 0']
Valor atual do State Array  após a rodada 4/24
['080615D6B98BA8C1', 'DEC6ACC69CB51EB6', 'B6F79C6588F0E96F',
'0623C4E49D31E2D2', '7A5CD589AAC9BBE9']
['796C1A6235D9A2F0', 'D9BA66E2CF007865', '786AE4423B26EF63',
'1A7E97BC9B6D5E4E', '6D518E55146BD5A0']
```

```
['338BA3AD6C949B8F', 'E3503EDFD0EF6649', 'AF54B8CEF45C4D5F',
'AA655977C1ECD4A0', '134BB89DCC86C313']
['AB99A8C2EEF37B3A', '34109C4204EFC04C', '998E467D53C3C283',
'2E503E691D901F14', 'F9B6661D7377E999']
['3B6FE035EB854238', 'C252D57101C7AE12', 'CD48C6611D44DFAA',
'336E9263A8A7772F', '529AF923EBC93A5B']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(1328207609067630276,1[bits]) = 2656415218135260552
Rotacionar Esquerda(3823346029823874682,1[bits]) = 7646692059647749364
Rotacionar Esquerda(12323720110111391751,1[bits]) = 6200696146513231887
Rotacionar Esquerda(12640052193007402648,1[bits]) = 6833360312305253681
Rotacionar Esquerda(15138820387136121020,1[bits]) = 11830896700562690425
Depois da rotacao de theta A[ 4 ]
['0b1111010010111001101010110001001101010101100100110111011111101001',
'0b1101101010100011000111001010101000101000110101111010101100000',
'0b1001101001011101110001001110111001100100001101100001100010011',
'0b1111001101101100110011000011101011100110111011111110100110011001',
'0b1010010100110101111100100100011111010111100100100111010011011']
Depois de theta
['0b1110101011101011011101001110001000100110010111111101010010111',
'0b1100010011110001110000110101101101011011000110110000100110111110',
'0b1110001100010110101110110010101110101011000001001001001101101',
'0b1111011010011111000010011110010111001010100100011011100011100111',
'0b1011101101100111001011011011011010100100010111101101011001001011']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(9489385923183709137,0[bits]) = 9489385923183709137
Rotacionar Esquerda(17499613267761758688,36[bits]) = 4210898702131386573
Rotacionar Esquerda(13275666411581749407,3[bits]) = 13971610924106237181
Rotacionar Esquerda(2318983461799319594,41[bits]) = 1995187270346233873
Rotacionar Esquerda(12743190073663283496,18[bits]) = 11487618251377263459
Rotacionar Esquerda(7408361748905794302,1[bits]) = 14816723497811588604
Rotacionar Esquerda(7039974929838275629,44[bits]) = 12088459762077688714
Rotacionar Esquerda(6582393327193514497,10[bits]) = 7309180142172505453
Rotacionar Esquerda(10095374122852140036,45[bits]) = 13294786092884090472
Rotacionar Esquerda(8816834716199922266,2[bits]) = 16820594791090137449
Rotacionar Esquerda(17479716896588466084,62[bits]) = 4369929224147116521
Rotacionar Esquerda(4326012832926302632,6[bits]) = 163660201640094223
Rotacionar Esquerda(16949096315933959060,43[bits]) = 80122897205191865
```

```
Rotacionar Esquerda(15991640158524910664,15[bits]) = 15852556750749462262
Rotacionar Esquerda(9884055257409433953,61[bits]) = 3541349916389873196
Rotacionar Esquerda(7924220907093510553,28[bits]) = 16433780835471034318
Rotacionar Esquerda(8189003957370418437,55[bits]) = 9419510170303834746
Rotacionar Esquerda(13960843822382809067,25[bits]) = 18031193548917013954
Rotacionar Esquerda(5011246773388817503,21[bits]) = 14733584689334366576
Rotacionar Esquerda(6392061878538787940,56[bits]) = 7230728395505835740
Rotacionar Esquerda(8463876069528693399,27[bits]) = 9843038462497500627
Rotacionar Esquerda(7095669348979541214,20[bits]) = 1935097413440513934
Rotacionar Esquerda(2045434011540558445,39[bits]) = 12702744208317592250
Rotacionar Esquerda(17770933537177975015,8[bits]) = 11459943385011906550
Rotacionar Esquerda(6751906135079349029,14[bits]) = 16552651177583007596
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1110010110110110110101001000101111011010110010010101011101101100',
'0b1100010010010101100110111111000110111111000111000101100010101100',
'0b1001111101101100001101111110111110000100101000101100001101100011',
'0b1100100010110001011010100101010011010010111010010011110111011100',
'0b1110100101101110110000011101011111011001101101101010010010101001']
Antes do modulo chi
['0b1110101011101011011101001110001000100110010111111101010010111',
'0b1100010011110001110000110101101101011011000110110000100110111110',
'0b1110001100010110101110110010101110101011000001001001001101101',
'0b1111011010011111000010011110010111001010100100011011100011100111',
'0b1011101101100111001011011011011010100100010111101101011001001101']
Depois do modulo chi
['0b1100000111110100000100001000100011111010110000110011011101100110',
'0b1010111110111100111110101111000010011011100111001101100010110100',
'0b1001110100101100010101111111111011000010000100010100010010110000',
'0b1010110001110001011010100100001011011010110010011001110110100000',
'0b1101011011101100000000010101000111011000101101100010001101111011']
Antes do modulo iota
['0b1000001110101101001100110111100011010101101101010000111111100000',
'0b1001010011010110100110100111011000010011001110010001111011111',
'0b1101011010010100011001011011010001011100111101001000000000111100',
'0b1100110110010110000111111100010010101011101000111010100011110011',
'0b1100111001000011111101110001110110101010101100110001011010001']
Depois do modulo iota
['0b1000001110101101001100110111100011010101101101011000111101101011',
```

```
'0b1001010011010110100110100111011000010011001110010001111011111',
'0b1101011010010100011001011011010001011100111101001000000111100',
'0b1100110110010110000111111100010010101011101000111010100011110011',
'0b1100111001000011111101110001110110101010101100110001011010001']
Valor atual do State Array  após a rodada 5/24
['83AD3378D5B58F6B', '6BA2C63D319EF2CA', '209A67D0EA4B22B5',
'CE7937068EDCB9E1', 'C1F41088FAC33766']
['2535A69D84CE47BF', '22DA495B65CE3D8E', 'C0C044601796C4F9',
'7C901182AF305FAA', '2BEF3EBC26E7362C']
['35A5196D173D203C', '074535161698E83B', 'FA5FB8DAD3817DC3',
'DF9A65CAC3A4D36A', '9D2C57FD84228960']
['CD961FC4ABA3A8F3', 'A0E08E4DEF95B25F', '416F49BD3CD89565',
'537ED20B44AF4FF5', '5638B5216D64CED0']
['0CE43F71DAAB3169', '890892D79737E27B', '5007B719B1CB13D2',
'0F314E687F1F5891', '6B760151D8B6237B']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(7482069880878561115,1[bits]) = 14964139761757122230
Rotacionar Esquerda(823426245056208184,1[bits]) = 1646852490112416368
Rotacionar Esquerda(3547955994484548165,1[bits]) = 7095911988969096330
Rotacionar Esquerda(5366546629431420289,1[bits]) = 10733093258862840578
Rotacionar Esquerda(5931148613191102834,1[bits]) = 11862297226382205668
Depois da rotacao de theta A[ 4 ]
['0b1100000111110100000100001000100011111010110000110011011101100110',
'0b1010111110111100111110101111000010011011100111001101100010100',
'0b1001110100101100010101111111111011000010000100010100010010110000',
'0b1010110001110001011010100100001011011010110010011001110110100000',
'0b1101011011101100000000010101000111011000101101100010001101111011']
Depois de theta
['0b1010100010101111001011111011111010011011010011111111011111000111',
'0b1011111001001100101110011111010111001000110000011111101101000110',
'0b1000100011111101000010101010001100110100011001001001110000001',
'0b1100001110011011001100100111011011011010000000000000111001110001',
'0b1111111011010101100001100000011001101111110100101110001111011010']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(468289790332437596,0[bits]) = 468289790332437596
Rotacionar Esquerda(11594278578422729864,36[bits]) = 15919319278121283329
Rotacionar Esquerda(12715800300569340683,3[bits]) = 9492682036006967389
Rotacionar Esquerda(5207462618194977732,41[bits]) = 5064166460360708709
```

```
Rotacionar Esquerda(9887300160053819998,18[bits]) = 18190336514330338522
Rotacionar Esquerda(3402365740262537672,1[bits]) = 6804731480525075344
Rotacionar Esquerda(7372162770938001036,44[bits]) = 16674795757214085441
Rotacionar Esquerda(4886496539067327289,10[bits]) = 4704812029654656271
Rotacionar Esquerda(16462321219694623069,45[bits]) = 13775311206489109448
Rotacionar Esquerda(14816268244442655097,2[bits]) = 3924840756641965543
Rotacionar Esquerda(2681470684408910180,62[bits]) = 670367671102227545
Rotacionar Esquerda(14225847378326124328,6[bits]) = 6563772601103927857
Rotacionar Esquerda(18443261281945996818,43[bits]) = 15740247821766580944
Rotacionar Esquerda(4954893130060179124,15[bits]) = 12143493094185771617
Rotacionar Esquerda(6173220851275603971,61[bits]) = 7689181634050532352
Rotacionar Esquerda(5901907645819482075,28[bits]) = 13791067684720245915
Rotacionar Esquerda(16361277553766795664,55[bits]) = 14443474427807787972
Rotacionar Esquerda(4612982856879375696,25[bits]) = 17255126402517174582
Rotacionar Esquerda(14762848739727546831,21[bits]) = 10864803341724654597
Rotacionar Esquerda(10425746091394240171,56[bits]) = 12362574151155185806
Rotacionar Esquerda(6077493207496128455,27[bits]) = 18045149368203525310
Rotacionar Esquerda(13712539386846115469,20[bits]) = 11437199389305332939
Rotacionar Esquerda(616941103396309441,39[bits]) = 11755767786090550553
Rotacionar Esquerda(14094914944831065713,8[bits]) = 11183131503390257603
Rotacionar Esquerda(18362730417877083098,14[bits]) = 7026068369052188597
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1100001100000011001101111110100101110001111011010111111110110101',
'0b1101010101101010111010111110101000001110000101000100100000000000',
'0b1111110001110001000011101010111011110100101111010001001001101101 0',
'0b1010101110010000101011111011000011010101000001111111100010001110',
'0b1101100111011111010101110111111001101110011110101001011110011 1']
Antes do modulo chi
['0b1010100010101111001011111011111010011011010011111111011111000111',
'0b1011111001001100101110011111010111001000110000011111101101000 1101',
'0b1000100011111101000010101010001100110100011001001001001110000 01',
'0b1100001110011011001100100111011011011010000000000000111001110001',
'0b1111111011010101100001100000011001101111110100101110001111011010']
Depois do modulo chi
['0b1000000010000001110111111001010000001010101111000000111010110100',
'0b1101010001011010111110111111011101001111110010111010010001000000',
'0b1111110101100001001110101110101111101010010110101010110011111011',
'0b1010111100010000011011111011000011010001010101111011101110001111',
```

```
’0b1111011001000111110101011111100000011100100101010000111001100011’]
Antes do modulo iota
[’0b1111001101111101000101000111001000001101101010100111011001100’,
’0b1011111001100111011101101111111011010010000111010011001001000111’,
’0b1111101000001111111000001001100101010000000011101011100101001011’,
’0b1111101101101111001011101010101100011010001011100011110010110000’,
’0b1010100100100111111111010110000001100011110100010101000001000000’]
Depois do modulo iota
[’0b1111001101111101000101000111011000001101101010100111011001101’,
’0b1011111001100111011101101111111011010010000111010011001001000111’,
’0b1111101000001111111000001001100101010000000011101011100101001011’,
’0b1111101101101111001011101010101100011010001011100011110010110000’,
’0b1010100100100111111111010110000001100011110100010101000001000000’]
Valor atual do State Array  após a rodada 6/24
[’1E6FA28EC1B54ECD’, ’E3EFCE64D2173544’, ’BB70851B1D057960’,
’90B9A9127DF89C4D’, ’8081DF940ABC0EB4’]
[’BE6776FDA43A648F’, ’A2BA2CB5E4CB474B’, ’C328897573EF5C5D’,
’2A693E860D29EB53’, ’6A2D7DF74FCBA440’]
[’FA0FC132A01D7296’, ’4B1744B78333BAF0’, ’8B378E2E49F00D2E’,
’993C37CA008B22C3’, ’FD613AEBEA5AACFB’]
[’FB6F2EAB1A2E3CB0’, ’7468CAAA5F206D61’, ’425A7C8DA6893D81’,
’F8EB4AAF79FA2651’, ’AF106FB0D157BB8F’]
[’2A49FF5818F45040’, ’8C328F6717FCE9A0’, ’9314B5CB5568F09B’,
’4F4F8290C95F127D’, ’F647D5F81C950E63’]
Antes de theta [’0b0’, ’0b0’, ’0b0’, ’0b0’, ’0b0’]
Rotacionar Esquerda(17444943484446526526,1[bits]) = 16443142895183501437
Rotacionar Esquerda(2459329364236166409,1[bits]) = 4918658728472332818
Rotacionar Esquerda(10684904884995580401,1[bits]) = 2923065696281609187
Rotacionar Esquerda(5663877691060761571,1[bits]) = 11327755382121523142
Rotacionar Esquerda(10025510517630055460,1[bits]) = 1604276961550559305
Depois da rotacao de theta A[ 4 ]
[’0b1000000010000001110111111001010000001010101111000000111010110100’,
’0b1101010001011010111110111111011101001111110010111010010001000000’,
’0b1111110101100001001110101110101111110101001011010101011001111111011’,
’0b1010111100010000011011111011000011010001010101111011101110001111’,
’0b1111011001000111110101011111100000011100100101010000111001100011’]
Depois de theta
[’0b1010001010001111101001000001000100110100110000011100001100’,
```

```
'0b1110100000100110100111001111001100000001101001001010110111111000',
 '0b1111111011010101101101111101111101001000011010110100101010000011',
'0b1011010001101110001110101101001001111100111000101100100011011 1',
'0b1110100010011000011010011111100010100101111101000001111101 1011']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(13025614196459201875,0[bits]) = 13025614196459201875
Rotacionar Esquerda(14987348605017331 37,36[bits]) = 14642593345093318307
Rotacionar Esquerda(5810809711252232456,3[bits]) = 9592989542598756418
Rotacionar Esquerda(5892057673257916206,41[bits]) = 5633542041344833797
Rotacionar Esquerda(9287014860107643870,18[bits]) = 7727616166410716040
Rotacionar Esquerda(3210113101985270642,1[bits]) = 6420226203970541284
Rotacionar Esquerda(7915496403818035581,44[bits]) = 5447058521552103591
Rotacionar Esquerda(9544277737905865926,10[bits]) = 15012788623253903889
Rotacionar Esquerda(13478033526630617943,45[bits]) = 17468046102623331891
Rotacionar Esquerda(4850902195471849366,2[bits]) = 956864708177845849
Rotacionar Esquerda(7059593022581307069,62[bits]) = 6376584274072714671
Rotacionar Esquerda(1846680481192661888,6[bits]) = 7507086354073051142
Rotacionar Esquerda(5890634946862482163,43[bits]) = 17012236052056388009
Rotacionar Esquerda(11011951887502324316,15[bits]) = 2878623843624045673
Rotacionar Esquerda(5304262496472104774,61[bits]) = 14498090867341176808
Rotacionar Esquerda(3435304871700209282,28[bits]) = 5101947155644074617
Rotacionar Esquerda(10771537185792158108,55[bits]) = 14864902530379155124
Rotacionar Esquerda(2749792046363418636,25[bits]) = 10962710646840119922
Rotacionar Esquerda(5187658765421159582,21[bits]) = 9596356980822835144
Rotacionar Esquerda(17319309346755809458,56[bits]) = 12893905290886937488
Rotacionar Esquerda(183027524212819724,27[bits]) = 9378350641943171572
Rotacionar Esquerda(16728230433432841720,20[bits]) = 14929461673613754985
Rotacionar Esquerda(9181392612646823235,39[bits]) = 1932785034850072530
Rotacionar Esquerda(3250348462494495287,8[bits]) = 1985723081660970797
Rotacionar Esquerda(8380131264958171099,14[bits]) = 954504454482615571
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1101001111110001010010111110100000011111011011011101000 10011',
'0b1100100100110011100100001101010001100101001101001010111111101000',
'0b1101011001111110000000011111010111101111011110100000001110001000',
'0b1011001011110000010110101000110000010110110110000011110111001 0000',
'0b1101010001110111011101100001111001010000110001011110010110 01']
Antes do modulo chi
['0b1010001010001111101001000001000100110100110000011100001100',
```

```
'0b1110100000100110100111001111001100000001101001001010110111111000',
'0b1111111011010101101101111101111101001000011010110100101010000011',
'0b1011010001101110001110101101001001111100111000101100100011011 1',
'0b1110100010011000011010011111100010100101111101000000111110111011']
Depois do modulo chi
['0b1000110001011001000010001111010000001110011011001010101101101 11',
'0b1000000000000111000101011010110101110000011000010101111111101000',
'0b1001011000110001000011110110101011011111101111001100011100010 10',
'0b1111011111100000001110111000110100011010000100010101110110010011',
'0b1000101101000111111001010110000111100101000000000101110001001001']
Antes do modulo iota
['0b1000011000100010011110001100100110001001101100111010001011011',
'0b1000110110011001110010011001001001000101101101010000010011111011',
'0b1100100100011000011000011011011101100100011001100001001001010 0',
'0b1001001001101110100110101001111001010001000001100101100111100100',
'0b1001000111011100010110000111011011011010001001111001100111011 01']
Depois do modulo iota
['0b1001000011000100010011110001100110110001001101101111010011011010',
'0b1000110110011001110010011001001001000101101101010000010011111011',
'0b1100100100011000011000011011011101100100011001100001001001010 0',
'0b1001001001101110100110101001111001010001000001100101100111100100',
'0b1001000111011100010110000111011011011010001001111001100111011 01']
Valor atual do State Array  após a rodada 7/24
['90C44F19B136F4DA', '4ABFD78D95E066E7', 'E4059EA7FDDC31BA',
'35ED58D5CBC0DF88', '462C847A073655B7']
['46CCE4C922DA827B', 'BD7AC80AEC8EA458', '8C3025BDC096C18A',
'F4A6B64971C8E622', '40038AD6B830AFE8']
['C91861B769198494', '6BA227DCE806C50B', 'F8135E20DF0452F2',
'0B8F8C9D28B3A349', '4B1887B56FDE638A']
['926E9A9E510659E4', 'EC963D1104E572CB', '4058313A37C32981',
'27F46D12392A4C0D', 'FBE03B8D1A115D93']
['48EE2C3B6D13CCED', '8A66E218089C0EB1', '1B9382FFD165F18A',
'1E16543F91E458A3', '8B47E561E5005C49']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(18057155572973796302,1[bits]) = 17667567072238040989
Rotacionar Esquerda(14694496812478790345,1[bits]) = 10942249551248029075
Rotacionar Esquerda(17520791645848964685,1[bits]) = 16594839217988377755
Rotacionar Esquerda(4436142343540283407,1[bits]) = 8872284687080566814
```

```
Rotacionar Esquerda(14236015598120232764,1[bits]) = 10025287122530913913
Depois da rotacao de theta A[ 4 ]
['0b1000110001011001000010001111010000011100110110010101010110110111',
'0b1000000000000111000101011010110101011100000110000101010111111101000',
'0b1001011000110001000011110110101011011111101111001100011100010010',
'0b1111101111100000001110111000110100011010000100010101110110010011',
'0b1000101101000111111001010110000111100101000000000101110001001001']
Depois de theta
['0b1111100010101000100110110100111011000010000011000101010110000011',
'0b1110000000010100101000011111110000111110000101111010111111011100',
'0b1100110001111000100101000111001101100001101011001000111010111110',
'0b1000001111100110100110010010010010101101101001000001110110100111',
'0b1111001101000001010001111100100001010010101101010001110001111101']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(6375925309260012360,0[bits]) = 6375925309260012360
Rotacionar Esquerda(10264686072716979689,36[bits]) = 8295312311958624659
Rotacionar Esquerda(119337588266035974,3[bits]) = 954700706128287792
Rotacionar Esquerda(6544015918000584310,41[bits]) = 15739215062604684425
Rotacionar Esquerda(9246370983164027775,18[bits]) = 15397213263233941830
Rotacionar Esquerda(1798350985007002696,1[bits]) = 3596701970014005392
Rotacionar Esquerda(17235303494265485047,44[bits]) = 16388457151193113147
Rotacionar Esquerda(4172856497349744548,10[bits]) = 11807172259231994087
Rotacionar Esquerda(13753127043520127076,45[bits]) = 13550372348347398522
Rotacionar Esquerda(15576881444229520414,2[bits]) = 6967293555789426811
Rotacionar Esquerda(17932999108510242543,62[bits]) = 18318307832409724347
Rotacionar Esquerda(10442568491011450591,6[bits]) = 4241596771188979684
Rotacionar Esquerda(16485443109975373223,43[bits]) = 17414644767634575799
Rotacionar Esquerda(6666277636441460436,15[bits]) = 12689014118974893633
Rotacionar Esquerda(524902406759093983,61[bits]) = 16206513865340744411
Rotacionar Esquerda(9592845054733161823,28[bits]) = 651818349974653468
Rotacionar Esquerda(4930121474359143669,55[bits]) = 8836684413150779862
Rotacionar Esquerda(13493476662473976222,25[bits]) = 1218102560682312939
Rotacionar Esquerda(10896903532834784986,21[bits]) = 9289881636337542962
Rotacionar Esquerda(12599854781798748788,56[bits]) = 8407899091141041938
Rotacionar Esquerda(4479435470014715267,27[bits]) = 11350224088329179446
Rotacionar Esquerda(4036677167193911260,20[bits]) = 9795602276398891090
Rotacionar Esquerda(3683422351055856574,39[bits]) = 3860111656500760172
Rotacionar Esquerda(9504452446423096743,8[bits]) = 16616352628361504643
```

```
Rotacionar Esquerda(17528370150409313405,14[bits]) = 5904804795891285200
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b10100011111001000010100101011010100011100011111011111011010000',
'0b11100000111010010001101001111110101001110001001111110010110111011',
'0b11010101101011011110001111100010100011011111111000000001010001110',
'0b11101001010111011011011101011010010101011001010100111110001001 0',
'0b11000001011000011001100100100110001111010110010011100000111101 1']
Antes do modulo chi
['0b11111000101010001001101101001110110000100000110001010110000011',
'0b11100000000101001010000111111100001111100001011110111111011100',
'0b11001100011110001001010001110011011000011010110010001110111110',
'0b10000011111001101001100100100100101011011010010000011101101001 11',
'0b11110011010000010100011111001000010100101011010001110001111101']
Depois do modulo chi
['0b11110010111101100011110000011111010001100001110100011100111000 11',
'0b11001100001100101011010011101001010001010010010011100101001 1001',
'0b11011111101110001100001101101010110001111111011000010110001000 10',
'0b10110101101000001110110111101110011001100110000010111100100 11',
'0b11000000011000011001000100101111011010010100110001110000011111 1']
Antes do modulo iota
['0b10010001111101111010111010011011110010010110101100010101100110 0',
'0b10000010010111110011101001111101001000110010100100011110 0',
'0b11000111001000100111010111001110010001111101111110100010011011',
'0b11101010001010001100010101000000000011011100101010001010100 10',
'0b11111011001001100111100111110011010001000000101100010011100100 11']
Depois do modulo iota
['0b11001000111110111101011101001101111001001011010100001010110001 01',
'0b10000010010111110011101001111101001000110010100100011110 0',
'0b11000111001000100111010111001110010001111101111110100010011011',
'0b11101010001010001100010101000000000011011100101010001010100 10',
'0b11111011001001100111100111110011010001000000101100010011100100 11']
Valor atual do State Array  após a rodada 8/24
['C8FBD74DE4B50AC5', 'E32F3EF39A91C43B', 'A0BF2BAB44744977',
'88E588607B92643A', 'F2F63C1F461D1CE3']
['0104BE74FA46523C', '37F0E89EE5C32118', '4DDECF1FC5851AB1',
'B50E36DAC5C5B17E', '66195A74A2927299']
['31C89D7391F7E89B', 'DCC509E6DB2094E4', '01C35291349484AF',
'C6DB28AD941C4F13', 'DFB8C36AC7F61622']
```

```
['1D4518A801B95152', '631EB198C117F793', 'E77D103DD1DE01F5',
'39186F1CB65B6E65', '16B41DBDCCCC1793']
['FB2679F3440B1393', 'B0CE15038E104957', '1501DF1B93A2EE1E',
'446BDFDDE30E1909', '6030C897B4A6383F']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(15837606352201830147,1[bits]) = 13228468630694108679
Rotacionar Esquerda(2224348323905091714,1[bits]) = 4448696647810183428
Rotacionar Esquerda(9674619127219547451,1[bits]) = 902494180729543287
Rotacionar Esquerda(4455027787913451508,1[bits]) = 8910055575826903016
Rotacionar Esquerda(2185535622829503139,1[bits]) = 4371071245659006278
Depois da rotacao de theta A[ 4 ]
['0b1111001011110110001111000001111101000110000111010001110011100011',
'0b1100110000110010101101001110100101000101001001001110010100110001',
'0b1101111110111000110000110110101010110001111111011000010110001000010',
'0b1011010110100000111011011110111001100110011000000101111001001 1',
'0b1100000001100001100100010010111101101001010011000111000001111111']
Depois de theta
['0b1001000000111000011000011101010101011000110111000010100100111110',
'0b1101110011110011010101101000000101001000111000010111101011100100',
'0b1100101010100101100111110011111001011011000010100011110010111111',
'0b1010110001011110000100010100100000100110101111110001111111101110',
'0b1101101011011010110001000110001001011110110101010011000001000010']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(4808807869276078902,0[bits]) = 4808807869276078902
Rotacionar Esquerda(10034926510537612239,36[bits]) = 8928875512599906279
Rotacionar Esquerda(13515051213737894248,3[bits]) = 15886689341355395909
Rotacionar Esquerda(10881434070655539361,41[bits]) = 9453410957762839832
Rotacionar Esquerda(8170092647326931552,18[bits]) = 18439750971073611143
Rotacionar Esquerda(13891170418919032732,1[bits]) = 9335596764128513849
Rotacionar Esquerda(1448065097720636095,44[bits]) = 8803395160398924820
Rotacionar Esquerda(18387474834983753539,10[bits]) = 13095275839620976636
Rotacionar Esquerda(4681164771923555380,45[bits]) = 7964202657152883890
Rotacionar Esquerda(10603287810072169200,2[bits]) = 5519663092869573570
Rotacionar Esquerda(8643284095838477315,62[bits]) = 15995879079241783040
Rotacionar Esquerda(11138239310418743237,6[bits]) = 11871041065836605798
Rotacionar Esquerda(15460686190790611419,43[bits]) = 16469345754081749262
Rotacionar Esquerda(3472599126470169729,15[bits]) = 10610729534007318552
Rotacionar Esquerda(14001103722297097066,61[bits]) = 6361823983714525037
```

```
Rotacionar Esquerda(17121860278647911248,28[bits]) = 6028876470164246803
Rotacionar Esquerda(15021386638087038484,55[bits]) = 749914586156793107
Rotacionar Esquerda(11791463931671992441,25[bits]) = 17413809734502795107
Rotacionar Esquerda(66565910467972283 03,21[bits]) = 14501288018392615966
Rotacionar Esquerda(2383326180459777635,56[bits]) = 7143011677647286670
Rotacionar Esquerda(5196081854552085662,27[bits]) = 6152885369659515271
Rotacionar Esquerda(15921164220958341860,20[bits]) = 7499775511838445365
Rotacionar Esquerda(7301126228491443807,39[bits]) = 14019476609420480406
Rotacionar Esquerda(12420383823919783918,8[bits]) = 6778278245421805228
Rotacionar Esquerda(15770132971968081986,14[bits]) = 12761116349075338934
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1011000100011000100101111011010101001100000100001011011010110110',
'0b1011000010010011011110100110100111100001101110101011111011011101',
'0b1111111111100111001001111100111101101001100000011100010110000111',
'0b1100011001000010001001101000110100010001010001000110001100011110',
'0b1001100100110011100100001010110101010100101001100101011110000010']
Antes do modulo chi
['0b1001000000111000011000011101010101011000110111000010100100111110',
'0b1101110011110011010101101000000101001000111000010111101011100100',
'0b1100101010100101100111110011111100101101100001010001111001011111',
'0b1010110001011110000100010100100000100110101111110001111111101110',
'0b1101101011011010110001000110001001011110110101010011000001000010']
Depois do modulo chi
['0b1000100100011011001101111011010111001100101000001011101010110110',
'0b1110000010111011011110100110110010100001101100101010001010010001',
'0b1101101111010111011011111111110110110101011000000101101011100001',
'0b1001001101010011001111100011110100011001001000100110111111101110',
'0b1001110100110011111000001000100101010011101001101000011110100001']
Antes do modulo iota
['0b1100011000111000010111111111001100011010010011001010001000111100',
'0b1100011111000001010001110111111010100111011101001110100010101010011',
'0b1101000010001110101101001100101110111100001011101000100100111000',
'0b1101000101110001001100111010001011110010000011001110100110011111',
'0b1110101111011110000111110010111111100011010101001010110000100']
Depois do modulo iota
['0b1100011000111000010111111111001100011010010011001010001010110110',
'0b1100011111000001010001110111111010100111011101001110100010101010011',
'0b1101000010001110101101001100101110111100001011101000100100111000',
```

```
'0b1101000101110001001100111010001011110010000011001110100110011111',
'0b1110101111011110000111111001011111110001101010100101010110000100']
Valor atual do State Array  após a rodada 9/24
['C6385FF31A4CA2B6', '731BD14A88320804', 'D48ECD9437315BAE',
'8B9AACDD00A6CD1E', '891B37B5CCA0BAB6']
['C7C28EFD4EE9D153', '4A928E01B4CD1B87', 'CC31EEE8C0750808',
'6D24CA1FD4D15CA0', '705DBD3650D95149']
['D08EB4CBBC2E8938', 'AAAF60F23F5B59EA', '504C6079B3C74660',
'5E19D826AB35E494', 'DBD76FFB6AC0B5C1']
['D17133A2F20CE99F', '79A99CF9BE3317E7', 'D59AD2FA10FF2E7A',
'870282EB78005819', '49A99F1E8C9137EE']
['1D7BC3E5FC6A9584', '0B587B5BC3DA7D1B', '8E07A7E20325E554',
'1255442F51315118', '4E99F044A9D343D1']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(16273713391121277077,1[bits]) = 14100682708533002539
Rotacionar Esquerda(1400116033806261992,1[bits]) = 2800232067612523984
Rotacionar Esquerda(3310277806402010155,1[bits]) = 6620555612804020310
Rotacionar Esquerda(2711600332843723265,1[bits]) = 5423200665687446530
Rotacionar Esquerda(2125300463618066118,1[bits]) = 4250600927236132236
Depois da rotacao de theta A[ 4 ]
['0b1000100100011011001101111011010111001100101000001011101010110110',
'0b1110000010111011011110100110110010100000110110010101000101001001',
'0b1101101111010111011011111111110110110101011000000101101011000001',
'0b1001001101010011001111100011110100011001001000100110111111101110',
'0b1001110100110011111000001000100101010011101001101000011110100001']
Depois de theta
['0b1001111000010110011001001001000001010111000010001100101100010001',
'0b1100111010100001110111000010011110010110111000100100000111101110',
'0b1100110011011010001111001101111011110000101101000110001000110110',
'0b1011110101001001100110000111011000101110011100101000110010010001',
'0b1011001100101001010001101100001001100100111101100110010011101110']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(2321154701660113308,0[bits]) = 2321154701660113308
Rotacionar Esquerda(2435520412957325945,36[bits]) = 1768691524663201430
Rotacionar Esquerda(3927295965967409682,3[bits]) = 12971623654029725841
Rotacionar Esquerda(3998925675131929269,41[bits]) = 15782137494131404218
Rotacionar Esquerda(18119663492818271918,18[bits]) = 16701401512080567767
Rotacionar Esquerda(5240264664497795858,1[bits]) = 10480529328995591716
```

```
Rotacionar Esquerda(8156150565858123921,44[bits]) = 3605438347184152521
Rotacionar Esquerda(10452178855300260604,10[bits]) = 3919585075926921796
Rotacionar Esquerda(4759008765228690673,45[bits]) = 11934020324039343821
Rotacionar Esquerda(3529277171808093709,2[bits]) = 14117108687232374836
Rotacionar Esquerda(7978660895936643949,62[bits]) = 6606351242411548891
Rotacionar Esquerda(8504703418753798347,6[bits]) = 9345440662666097373
Rotacionar Esquerda(16896177739511144099,43[bits]) = 7274755218091348747
Rotacionar Esquerda(8047363681837971129,15[bits]) = 18253336862307170262
Rotacionar Esquerda(3760663717101845911,61[bits]) = 16610984029133588402
Rotacionar Esquerda(15255818968649779188,28[bits]) = 6845634708080654568
Rotacionar Esquerda(3821841151053190730,55[bits]) = 2673595512901484395
Rotacionar Esquerda(447257683626192510,25[bits]) = 18210081742022011380
Rotacionar Esquerda(16082249487316144883,21[bits]) = 1617097906273904116
Rotacionar Esquerda(5366151620010630130,56[bits]) = 17458899286944227035
Rotacionar Esquerda(11391402878578510609,27[bits]) = 9419355966781043492
Rotacionar Esquerda(7444711952828408046,20[bits]) = 16230048445323703566
Rotacionar Esquerda(14761177656890803302,39[bits]) = 12998007989555064696
Rotacionar Esquerda(6819800289910539849,8[bits]) = 11874931288400349534
Rotacionar Esquerda(6454963803782853238,14[bits]) = 2943186601408042597
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1010001101100001001100100111101100110010011101100101100110010l',
'0b1110011010000110000100011111011100111010001100011100011110110010',
'0b1110011111000111010011000010111111111010101110111111011011010111',
'0b1111001001001010011110000110011001110111101000011001111011011011',
'0b1100001111101010000010111000111000101110000100010001100000110100']
Antes do modulo chi
['0b1001111000010110011001001001000001010111000010001100101100010001',
'0b1100111010100001110111000010011110010110111000100100000111011l0',
'0b1100110011011010001111001101111011110001011010001100010001100110',
'0b1011110101001001100110000111011000101110011100101000110010010l1',
'0b1011001100101001010001101100001001100100111101100110010011101l0']
Depois do modulo chi
['0b1110101101000101011110100011101100111011001111100000000100100',
'0b1000110101110100011001011110111001110001111010111000010101101l0',
'0b1110011101000110111000000010111111001101100111111111110100001l0',
'0b1110101001001001110110011110010001100111101010101101001001001',
'0b1110011111111101010001111000010100010110010000010000011100010100']
Antes do modulo iota
```

```
['0b1100100110000100110110110100111111011010010110111001001100111110',
'0b1001011000000000110101001001001111011101001000100111100001111001',
'0b1110110101110100010000011111110111001000110100000010111100000100',
'0b1010010011011100010011100001010010001101111100000000001101100100',
'0b1100101111001110010010100111000110110101011010101110000011001011']
Depois do modulo iota
['0b1100100110000100110110110100111111011010010110111001001000010110',
'0b1001011000000000110101001001001111011101001000100111100001111001',
'0b1110110101110100010000011111110111001000110100000010111100000100',
'0b1010010011011100010011100001010010001101111100000000001101100100',
'0b1100101111001110010010100111000110110101011010101110000011001011']
Valor atual do State Array  após a rodada 10/24
['64C26DA7ED2DC916', '200917BB03417B3D', '4C7D57575AC5010A',
'1657349B6F5BAC6C', '3AD15E8ECECF8024']
['4B00D493DD227879', 'E0A6BF136A665B42', 'F604674A886ED1A3',
'BC9EAC4929A20685', '46BA32F738F5C2B4']
['ED7441FDC8D02F04', '81F9A7C7B6F632D7', 'BFB470F43EB5CD75',
'B4FC2AD739064B7E', 'E746E02FCD9FFD0E']
['A4DC4E148DF00364', 'D19B67B0769F4B04', '346F3198112AFA4D',
'FDE0E732F30C96F2', 'EA49D9E467AAD249']
['CBCE4A71B56AE0CB', '6E1FCCFCB1933FE9', 'B48832E66D1E7F7C',
'C3011A1F2F75CD71', 'E7FA8F0A2C810714']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(18361919101390906949,1[bits]) = 18277094129072262283
Rotacionar Esquerda(9595556274304096493,1[bits]) = 744368474898641371
Rotacionar Esquerda(2365602740237154836,1[bits]) = 4731205480474309672
Rotacionar Esquerda(10853352637706693315,1[bits]) = 3259961201703835015
Rotacionar Esquerda(12512403483632434628,1[bits]) = 6578062893555317641
Depois da rotacao de theta A[ 4 ]
['0b1110101101000101011110100011101100111011001111100000000000100100',
'0b1000110101110100011001011110111001110001111010111000010101101100',
'0b1110011101000110111000000010111111001101100111111111111010000111 0',
'0b1110101001001001110110011110010001100111101010101101001001001001',
'0b1110011111111101010001111000010100010110010000001000001110001010 0']
Depois de theta
['0b1000001010011001110100011111111111011011100001111000001101110 01',
'0b1111010010011110000100100001100001101111111001100000110010100 1',
'0b1001110011011011010101100101111011101110100100111011110010010011',
```

```
'0b1001000111010100011011111001010101000100101001101001001111010100',
'0b1001110001100111001110010111101100001111100011010100011010001001']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(1151231940374392670,0[bits]) = 1151231940374392670
Rotacionar Esquerda(2322527469423746609,36[bits]) = 13938046088292624089
Rotacionar Esquerda(9678186136394303820,3[bits]) = 3638512796316224100
Rotacionar Esquerda(14981185586487141676,41[bits]) = 9892818074392450457
Rotacionar Esquerda(11598414528154322563,18[bits]) = 7079607457308115927
Rotacionar Esquerda(9797981455908943650,1[bits]) = 1149218838108335685
Rotacionar Esquerda(5140512149767329629,44[bits]) = 7022752794747144359
Rotacionar Esquerda(2740964034305490632,10[bits]) = 2842071924970561688
Rotacionar Esquerda(8532944926878402331,45[bits]) = 16204917712626283217
Rotacionar Esquerda(14551050678513660918,2[bits]) = 2863970492925988827
Rotacionar Esquerda(17512085681819947879,62[bits]) = 18213079475737150681
Rotacionar Esquerda(5295772710553437134,6[bits]) = 6888060148648047506
Rotacionar Esquerda(58065786222534424,43[bits]) = 3384666253889385227
Rotacionar Esquerda(10021929646758225952,15[bits]) = 9652664796110144906
Rotacionar Esquerda(860759911381757201,61[bits]) = 2413437998136413602
Rotacionar Esquerda(13709171104058040582,28[bits]) = 13972099056074492967
Rotacionar Esquerda(1479813657424841711,55[bits]) = 17837144785436822053
Rotacionar Esquerda(2084001352516634132,25[bits]) = 6958167029618169784
Rotacionar Esquerda(6194439421182008216,21[bits]) = 4228958508198575842
Rotacionar Esquerda(7716615563023450139,56[bits]) = 1975698068567114624
Rotacionar Esquerda(4705391896598659513,27[bits]) = 18405681771682883399
Rotacionar Esquerda(4406636471935140649,20[bits]) = 5215660503876227704
Rotacionar Esquerda(11302722655545965715,39[bits]) = 5322772960550530935
Rotacionar Esquerda(10508146517454787540,8[bits]) = 1530761778054062606 5
Rotacionar Esquerda(11270039793222633097,14[bits]) = 14870538400718546713
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1100111001011110110000111110001101010001101000100110011100011001',
'0b1000010111111001000001000100101000011001011001110011010010100010',
'0b1100010001111111101000101111001000110100000111010000011110101111',
'0b1101101101011000101101110110011111000010111100100001110000000',
'0b1001111011111101101110111111110101000110000011011100111111011011']
Antes do modulo chi
['0b1000001010011001110100011111111111011011100001111000001101110 01',
'0b1111010010011110000100100001100001101111111001100000110010100 1',
'0b1001110011011011010101100101111011101110100100111011110010010011',
```

```
'0b1001000111010100011011111001010101000100101001101001001111010100',
'0b1001110001100111001110010111101100001111100011010100011010001001']
Depois do modulo chi
['0b1010111001011010110000111100011100010001111000100110011110111000',
'0b1010010111111101100000100110101001011001001010000111111111111010',
'0b1100100011101011010111011111010011001100011111000010010010000101',
'0b1101101101010111101111111111101111100111101010010000110001100 0',
'0b1001001011010011011001010110010101000000100101110001011111111 1']
Antes do modulo iota
['0b1011100011111111101011011101111100000110101011100010101 10',
'0b1111001111111000110111100001010000100110110000000000010000100011',
'0b1011111111001011111000011110011100011010101011011010100110 1101',
'0b1101100101111110000011100100010001010010010000010110011101000111',
'0b1111010010010101110100100000101101101001110001100110000110001011']
Depois do modulo iota
['0b1011100011111111101011011001111100000110111011100010111 11',
'0b1111001111111000110111100001010000100110110000000000010000100011',
'0b1011111111001011111000011110011100011010101011011010100110 1101',
'0b1101100101111110000011100100010001010010010000010110011101000111',
'0b1111010010010101110100100000101101101001110001100110000110001011']
Valor atual do State Array  após a rodada 11/24
['0171FF5B3E0DDC5F', '7175D32C6D438447', 'EAB640A672F47212',
'3B117B43DF13B6A4', 'AE5AC3C711E267B8']
['F3F8DE1426C00423', '88E0D9591211E2E9', '3362990ECB7DC746',
'2063F0CD0A2272D4', '297F609A964A1FFA']
['2FF2F879C6AB6A6D', 'CBF8DB756932F393', '428020353035D4FE',
'D9AF9D446233B891', '323AD77D331E1245']
['D97E0E4452416747', '41E9C53667B423DB', '3D7B01DB11B52298',
'61F12F7C761061CD', '1B6AF7FEF9EA4318']
['F495D20B69C6618B', '778A54BCD53C9AAD', '6F6ACDAE6DAF2535',
'510B5B9FCEF9D599', '24B4D9595025C5FF']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(292241990042004555,1[bits]) = 584483980084009110
Rotacionar Esquerda(14503057449686558215,1[bits]) = 10559370825663564815
Rotacionar Esquerda(17449023209770567861,1[bits]) = 16451302345831584107
Rotacionar Esquerda(9998372184100826336,1[bits]) = 1550000294492101057
Rotacionar Esquerda(17334360986854995165,1[bits]) = 16221977900000438715
Depois da rotacao de theta A[ 4 ]
```

```
['0b1010111001011010110000111100011100010001111000100110011110111000',
'0b1010010111111101100000100110101001011001001010000111111111111010',
'0b1100100011101011010111011111010011001100011111000010010010000101',
'0b1101101101010111011111111111011111001111010100100001100011000',
'0b1001001011010011011001010110010101000000100101110001011111111']
Depois de theta
['0b1011110101011101101010110001110111010101110010101100011010110110',
'0b1110100111100000001000010000000101001001100010101111101111010 0',
'0b1000010011110110111111101001111111011100110110101100110100 1011',
'0b1000011011011001111100100100001111011100001011100010000 10110',
'0b1101111011001110110001100000111001010000001101011001001111000 1']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(9487998962458978345,0[bits]) = 9487998962458978345
Rotacionar Esquerda(8152928775427436629,36[bits]) = 2781823066358962279
Rotacionar Esquerda(12479232587140881947,3[bits]) = 7600140328579297501
Rotacionar Esquerda(6603356593870918449,41[bits]) = 15106870539381877773
Rotacionar Esquerda(8523072848564049405,18[bits]) = 2766606273275615520
Rotacionar Esquerda(1400546385445452949,1[bits]) = 2801092770890905898
Rotacionar Esquerda(16932047995249924667,44[bits]) = 14529666484533137867
Rotacionar Esquerda(12241546699677338433,10[bits]) = 10004594420809008807
Rotacionar Esquerda(2590602907524619017,45[bits]) = 3125819741926837539
Rotacionar Esquerda(1553805732700349055,2[bits]) = 6215222930801396220
Rotacionar Esquerda(790034833480601394,62[bits]) = 9420880745224926156
Rotacionar Esquerda(15213755298054568550,6[bits]) = 14449647242595703220
Rotacionar Esquerda(11727554769819691486,43[bits]) = 6003004828256005208
Rotacionar Esquerda(15941481556800610232,15[bits]) = 14015718009023000221
Rotacionar Esquerda(10316138671200311317,61[bits]) = 12818732379968508674
Rotacionar Esquerda(16705816554138634594,28[bits]) = 5837897215120338874
Rotacionar Esquerda(18204798927037254930,55[bits]) = 9907446631100496870
Rotacionar Esquerda(389593952298600279,25[bits]) = 4997531099942539323
Rotacionar Esquerda(13634278000978943499,21[bits]) = 17534528461060155093
Rotacionar Esquerda(10217782967362415199,56[bits]) = 6885384648319413354
Rotacionar Esquerda(13645250590631708342,27[bits]) = 17198778815659830616
Rotacionar Esquerda(4213126523758231284,20[bits]) = 9513051758029678464
Rotacionar Esquerda(2395281304958972747,39[bits]) = 11194160389365421051
Rotacionar Esquerda(607316502768706070,8[bits]) = 7899072119112341000
Rotacionar Esquerda(4013746871599981809,14[bits]) = 17032865593260002796
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
```

```
['0b1110110001100000111001010000001101011001001111000100110111101100',
'0b1011000111100101010010010010111011011010110100100001001100000010',
'0b1001100110010011110101101111111010101111111010111011001001000000',
'0b1011111100011011100110011011011011110010000000110101100011010101',
'0b1010110010000001110100001010011011011100100011110001001111111100']
Antes do modulo chi
['0b1011110101011101101010110001110111010101110010101100011010110110',
'0b1110100111100000001000010000000101001001100010101111101111010100',
'0b1000010011110110111111101001111111011100110110101100110100101011',
'0b1000011011011001111100100100001111011100001011100010000101101110',
'0b1101111011001110110001100000111001010000001101011001001111000101']
Depois do modulo chi
['0b1010010001100011011111110010010111011000011001110101110000101110',
'0b1101011110010001001111000001110001101111010000100100110000001010',
'0b1110111001100100111100010010101101000111111101111101100110110100',
'0b1011111100111001100110110011001011110110001000110111110001001101',
'0b1011111000000101010101001110011011010111100011110000001110111110']
Antes do modulo iota
['0b1001000111100000011001011101100101101110101001000000010000111001',
'0b1110000111110001111001110000100011111001101001001111111100111',
'0b1000111000011111110011000010100110100111011101101110010010000l',
'0b1100110111010100010011010111101110110011100011111101011110111000',
'0b1001000010111100000101001001111100111000011011111011001111010101']
Depois do modulo iota
['0b1001000111100000011001011101100111101110101001000000010000110011',
'0b1110000111110001111001110000100011111001101001001111111100111',
'0b1000111000011111110011000010100110100111011101101110010010000l',
'0b1100110111010100010011010111101110110011100011111101011110111000',
'0b1001000010111100000101001001111100111000011011111011001111010101']
Valor atual do State Array  após a rodada 12/24
['91E065D9EEA40433', '69B2B6066B2F934E', '5F6E30161D2F2570',
'F0DB2873E3F786D4', 'A4637F25D8675C2E']
['387C79C23E693FE7', '860506098A22AAA2', 'F9FD565C1A96E2DD',
'6B6104EE515EA19B', '35E44F071BD09302']
['2387F30A69DDB921', 'E00255AD507E9FB4', '473A1D8CBB1F191B',
'6D042E3D42EA3602', 'EE64F12B47F7D9B4']
['66EA26BDD9C7EBD8', '669B85161380307F', '97DB7D044E2C86C5',
'62A3C761CB362F8D', '5F9CCD997B11BC4D']
```

```
['90BC149F386FB3D5', 'C9D8101F54C297E2', '9D192DD0B69BD20B',
'511B73BBD79BDE0D', '5F02AA736BC781DE']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(11598581774211416517,1[bits]) = 4750419474713281419
Rotacionar Esquerda(16963699778872511096,1[bits]) = 15480655484035470577
Rotacionar Esquerda(14197235514409410765,1[bits]) = 9947726955109269915
Rotacionar Esquerda(9186682311286336267,1[bits]) = 18373364622572672534
Rotacionar Esquerda(8957058446511364856,1[bits]) = 17914116893022729712
Depois da rotacao de theta A[ 4 ]
['0b1010010001100011011111110010010111011000011001110101110000101110',
'0b1101011110010001001111000001110001101111010000100100110000010',
'0b1110111001100100111100010010101101000111111101111101100110110100',
'0b1011111100111001100110110011001011110110001000110111100010011101',
'0b1011111000000101010101001110011011010111100011110000000111011110']
Depois de theta
['0b1001100111111110011100110011100110000101111110000000100100010011',
'0b1000011110010100001100011011010001100100111111000110001111111',
'0b1101001111111001111111010011011100011010011010001000110010001001',
'0b1100010000000011100000110000101001001101000111011101001011100000',
'0b1100010100111111010011001101111001101100101100011010100111000011']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(12641923483074309299,0[bits]) = 12641923483074309299
Rotacionar Esquerda(499123811419756391,36[bits]) = 7625005869382756196
Rotacionar Esquerda(2096061405012267425,3[bits]) = 16768491240098139400
Rotacionar Esquerda(6375796391460193112,41[bits]) = 7459844106886452034
Rotacionar Esquerda(12550779181784079189,18[bits]) = 5525074991157524661
Rotacionar Esquerda(14062838457813982535,1[bits]) = 9678932841918413455
Rotacionar Esquerda(3215162345499747499,44[bits]) = 10829664834230019499
Rotacionar Esquerda(5375572522851193277,10[bits]) = 7456529434175534378
Rotacionar Esquerda(14699764584468446838,45[bits]) = 9209537431827658851
Rotacionar Esquerda(7152731094076971499,2[bits]) = 10164180302598334381
Rotacionar Esquerda(8472726962568422702,62[bits]) = 11341553777496881483
Rotacionar Esquerda(15205922565210645123,6[bits]) = 13948352340584603892
Rotacionar Esquerda(7908604424593070405,43[bits]) = 11442005554869736615
Rotacionar Esquerda(13627999513238062747,15[bits]) = 3507513424014597776
Rotacionar Esquerda(13250207542207255125,61[bits]) = 13185490988844376650
Rotacionar Esquerda(16522386135205960378,28[bits]) = 7560600031230997738
Rotacionar Esquerda(9147200324721933813,55[bits]) = 18068292932135170494
```

```
Rotacionar Esquerda(8688649743176821356,25[bits]) = 15014913011384133777
Rotacionar Esquerda(8589386711577654243,21[bits]) = 3932981123598968436
Rotacionar Esquerda(4939064981663908451,56[bits]) = 7152995032339490306
Rotacionar Esquerda(11096433222785435923,27[bits]) = 14713189925743752089
Rotacionar Esquerda(610593008908224063,20[bits]) = 3581598638837630868
Rotacionar Esquerda(15274518024311835785,39[bits]) = 3766773910104611725
Rotacionar Esquerda(7062138468315359600,8[bits]) = 126528665195999330
Rotacionar Esquerda(7106581733597304035,14[bits]) = 16833274077249067175
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1110100110011011110011011001011000110101001110001101100010100111',
'0b1011011011111100010001100011000111010011001011101110001001001010',
'0b1001100101011010000001001101110011011010101011010111000101101101',
'0b1100011010001001000101100010101011011101011101010011000000010',
'0b1000110100001110011011000010001000010001011001010110011110101101']
Antes do modulo chi
['0b1001100111111110011100110011100110000101111110000000100100010011',
'0b1000011110010100001100011011010001100100111111000110001111111',
'0b1101001111111001111111010011011100011010011010001000110010001001',
'0b1100010000000011100000110000101001001101000111011101001011100000',
'0b1100010100111111010011001101111001101100101100011010100111000011']
Depois do modulo chi
['0b1111100110010001010111010001011001011101101110010001100110101111',
'0b1010011111101100000001100110010110010010100011101110000101011110',
'0b1101001011011000001011100010011001011101011010111000110001010101',
'0b1000010100101001011110100100101000011001010101010100010011001100',
'0b1110111110010100010111001000001100000000011010110110011100011001']
Antes do modulo iota
['0b1010011111111000100101011010010111001011001011110001111001011011',
'0b1010000011101101001001001010101100100110010110001011110011100010',
'0b1001011000011111010101010111001111110111000110011011001010001110',
'0b1100101000000101010010000100101100001101110010111110001110010011',
'0b1001100100100101010011110101101011101100110100000111001101001010']
Depois do modulo iota
['0b1010011111111000100101011010010111000101100101111010111100001111100',
'0b1010000011101101001001001010101100100110010110001011110011100010',
'0b1001011000011111010101010111001111110111000110011011001010001110',
'0b1100101000000101010010000100101100001101110010111110001110010011',
'0b1001100100100101010011110101101011101100110100000111001101001010']
```

```
Valor atual do State Array  após a rodada 13/24
['A7F12B4B165EBC3C', 'B65E72D99CB187FB', '57C127EE091E8C24',
'30F4E317FE2EC264', 'F9915D165DB919AF']
['A0ED24AB2658BCE2', '26FE3CFC6370B7F7', '6885A3C15B624F00',
'37CE780A2D902CC3', 'A7EC0665928EE15E']
['961F5573F719B28E', 'C01280AF5880F096', '9C73AD9CB9E7A004',
'8393FD279CC17268', '0D2D82E265D6B8C5']
['CA05484B0DCBE393', '795466904CDAEDF4', '243A6116D9F4D528',
'BC8670A8B3088F09', '4294BD250CAAA266']
['99254F5AECD0734A', 'B93FC8A11F9E31FC', 'BC4E08EBFDDBFF20',
'77E7B3A0FA528B00', 'EF945C83006B6719']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(10437479971776961682,1[bits]) = 2428215869844371749
Rotacionar Esquerda(4270327578913687848,1[bits]) = 8540655157827375696
Rotacionar Esquerda(5749026558608513222,1[bits]) = 11498053117217026444
Rotacionar Esquerda(18325208695432054091,1[bits]) = 18203673317154556567
Rotacionar Esquerda(13989127681107829385,1[bits]) = 9531511288506107155
Depois da rotacao de theta A[ 4 ]
['0b1111100110010001010111010001011001011101101110010001100110101111',
'0b1010011111101100000001100110010110010010100011101110000101011110',
'0b1101001011011000001011100010011001011101011010111000110000101',
'0b1000010100101001011110100100101000011001010101010010001001100110',
'0b1110111110010100010111001000001100000000011010110110011100011001']
Depois de theta
['0b1100100001111101000011001000000001011110010100110001000111010',
'0b1101100011000100001100001010011110110001010001100111100100011011',
'0b1100011010100011100111001101010000101111111110110110010100010000',
'0b1000100100011010101000110001001101000110100001110111111110110011',
'0b1001000001101001000010101101010100101001000110101110101100110 0']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(8652490255279325266,0[bits]) = 8652490255279325266
Rotacionar Esquerda(9155780570606862476,36[bits]) = 9736792049209695926
Rotacionar Esquerda(5331606872026975968,3[bits]) = 5759366828796704514
Rotacionar Esquerda(1578424856358248445,41[bits]) = 14033210029979604615
Rotacionar Esquerda(5100245328540127012,18[bits]) = 15594430502173481758
Rotacionar Esquerda(214958672803870498,1[bits]) = 429917345607740996
Rotacionar Esquerda(10546271321151244078,44[bits]) = 13867402550939893153
Rotacionar Esquerda(8410293843423674447,10[bits]) = 15958157317191581138
```

```
Rotacionar Esquerda(14839848474139581741,45[bits]) = 14674339187481771702
Rotacionar Esquerda(980119777495384357,2[bits]) = 3920479109981537428
Rotacionar Esquerda(6379644855300170042,62[bits]) = 10818283250679818318
Rotacionar Esquerda(7479785320811880990,6[bits]) = 17537658689221592985
Rotacionar Esquerda(10609222070135524634,43[bits]) = 5505884301750311437
Rotacionar Esquerda(3130648018202327094,15[bits]) = 2730466555037685177
Rotacionar Esquerda(12900035789476713022,61[bits]) = 15447562528966752839
Rotacionar Esquerda(17804931882825777627,28[bits]) = 7484340593665080627
Rotacionar Esquerda(17306568295501361020,55[bits]) = 13724744758408458935
Rotacionar Esquerda(4931667018147705303,25[bits]) = 882257354350453146
Rotacionar Esquerda(8891583997289548982,21[bits]) = 1235710029719366824
Rotacionar Esquerda(12683407042697218239,56[bits]) = 13812545020004771784
Rotacionar Esquerda(3611679231288067194,27[bits]) = 53099968391608857
Rotacionar Esquerda(7809831452209790091,20[bits]) = 9600981935639152161
Rotacionar Esquerda(14313456475865572624,39[bits]) = 18280823797605886487
Rotacionar Esquerda(9879388035773595571,8[bits]) = 1919399059831894921
Rotacionar Esquerda(2601465081162545868,14[bits]) = 10425079498087270662
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1001000010101101010100101001000110101110101100110000100100000110',
'0b1101011001100000110001000100011010000000101100101011101001000111',
'0b1101100001101010100010111110101100111100100100010001101100011110',
'0b1011111110110000000001001000001110000001011110011010011111001000',
'0b1101100110100001010110111110101001101111001010000010010010100']
Antes do modulo chi
['0b1100100001111101000011001000000001011110010100110001000111010',
'0b1101100011000100001100001010011111011000101000110011110010001011',
'0b1100011010100011100111001101010000101111111111011011001010000010000',
'0b1000100100011010101000110001001101000110100001110111111110110011',
'0b1001000001101001000010101101010100101001000110101110101100110 0']
Depois do modulo chi
['0b1000011001101011110111011010100001000101110010011010010100111',
'0b1010110010000001100011001100100110000000011010000111000010001111',
'0b1010100110101010001011110010111011110010010001100110101000011 1',
'0b1110001011000000001100010001111010000100010100101000110110111 0',
'0b1111000110000010000100101101010011010010010001011000100101']
Antes do modulo iota
['0b1110100000110111100011010010011000110000101010000000010010111 10',
'0b1011010001110111011001100100011011111001111001010011000011 0001',
```

```
’0b10011110101101110111110011110110111100111010000111001000110’,
’0b10110001110101001100101001000111101000010010000110110110101l001’,
’0b11010111101000001100101100001111001111000101110001100000001001110’]
Depois do modulo iota
[’0b11110100000110111100011010010011000110000101010000000011011010101’,
’0b10110100011101110110011001000110111110011110010100110000110001’,
’0b10011110101101110111110011110110111100111010000111001000110’,
’0b10110001110101001100101001000111101000010010000110110110101l001’,
’0b11010111101000001100101100001111001111000101110001100000001001110’]
Valor atual do State Array  após a rodada 14/24
[’F41BC693185402D5’, ’D174E205BCDB5901’, ’CCE19499F40A1B0B’,
’79349FB3C68B6CF8’, ’10CD7BB508B934A7’]
[’2D1DD991BE794C31’, ’053D1211FED74E95’, ’5BAD25DD491A7743’,
’EA3881A70830BB86’, ’5640C664C0343847’]
[’09EB77CF6F3A0E46’, ’E1E354ACCA6F9598’, ’CC76E0CA9608E98C’,
’1F364742C455D7C9’, ’2A6A8BCBBC919A87’]
[’58EA6523D090DB59’, ’A7A0189FBCE7DE9F’, ’4766C78530619F92’,
’25E832FD2C9B4DA8’, ’38B00C47A114A36E’]
[’D7A0CB0F3C5C604E’, ’BC7564AC9BF72637’, ’C9F28CB341466A07’,
’42BDFB2FAB763ECD’, ’1E30425A9A491625’]
Antes de theta [’0b0’, ’0b0’, ’0b0’, ’0b0’, ’0b0’]
Rotacionar Esquerda(3350634742242572964,1[bits]) = 6701269484485145928
Rotacionar Esquerda(15397273555116716113,1[bits]) = 12347803036523880611
Rotacionar Esquerda(16964937220303844306,1[bits]) = 15483130366898136997
Rotacionar Esquerda(5361385854197637932,1[bits]) = 10722771708395275864
Rotacionar Esquerda(6892696425018948533,1[bits]) = 13785392850037897066
Depois da rotacao de theta A[ 4 ]
[’0b10000110011010111101110110101000010001011100100110100101001l1’,
’0b10101100100000011000110011001001100000000110100001110000100011l’,
’0b10101001101010100010111100101110111100100100011001101010000111’,
’0b11100010110000000011000100011110100001000101001010001101101110’,
’0b11110001100000100001001011010100110100100100010110001001011’]
Depois de theta
[’0b10001001110110101100110111100111100111000001101101100000000011111’,
’0b10011000001101101100100010000001101000000010111100111111111’,
’0b11111100100101010010110100011010111101000100101000111100011111’,
’0b11011001001000000010001000000010110011111010000000100111110l0110’,
’0b10010100001000001011111000111000101110011111101100100101001l101’]
```

```
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(16321906530002916529,0[bits]) = 16321906530002916529
Rotacionar Esquerda(4288852369327692373,36[bits]) = 18296336740162537498
Rotacionar Esquerda(2266364904608749602,3[bits]) = 18130919236869996816
Rotacionar Esquerda(5652769820588313917,41[bits]) = 7933788507847812994
Rotacionar Esquerda(13922880580954601002,18[bits]) = 611565885880730848
Rotacionar Esquerda(2706401229189038615,1[bits]) = 5412802458378077230
Rotacionar Esquerda(17421860150271169923,44[bits]) = 2690970168692573943
Rotacionar Esquerda(1520148756212387470,10[bits]) = 7105824169882433620
Rotacionar Esquerda(6006652974150108553,45[bits]) = 6390935987368871336
Rotacionar Esquerda(5228282505845685537,2[bits]) = 2466385949673190533
Rotacionar Esquerda(3765410727288866314,62[bits]) = 10164724718676992386
Rotacionar Esquerda(11749289307694688834,6[bits]) = 14084752744078020776
Rotacionar Esquerda(3807258332612097165,43[bits]) = 16979812659745145115
Rotacionar Esquerda(13818800708651319955,15[bits]) = 2234843738088792035
Rotacionar Esquerda(3554031913359636230,61[bits]) = 14279312044452118240
Rotacionar Esquerda(4058997828540062449,28[bits]) = 5774570949840291664
Rotacionar Esquerda(12346736102626659727,55[bits]) = 14399604729517065926
Rotacionar Esquerda(6797811951061230016,25[bits]) = 16717873929384668507
Rotacionar Esquerda(7244277821154425761,21[bits]) = 683708007484002587
Rotacionar Esquerda(278398102670084292,56[bits]) = 14124375924022430472
Rotacionar Esquerda(4966739161379614751,27[bits]) = 11416745456104336183
Rotacionar Esquerda(171377725024550143,20[bits]) = 12835373337948464653
Rotacionar Esquerda(9100251531439316543,39[bits]) = 1337322520177231549
Rotacionar Esquerda(7822771250473740246,8[bits]) = 10381080160645928556
Rotacionar Esquerda(5336870133857882781,14[bits]) = 1713363744276828804
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1011111000111000101110011111110110010010100111010100100100000100',
'0b1100011000101010010011101010011000111110100001100111111011100000',
'0b1000011111001011011111101110110110001010101100000100111000000',
'0b1100010000000011110111010001000110011001011011111100101100001000',
'0b1000100011101001011100111101000010100101001000111101001000000101']
Antes do modulo chi
['0b1000100111011010110011011110011110011100000110110110000000011111',
'0b1001100000110110110010001000000110100000001011110011111111',
'0b1111110010010101001011010001101011110100010010100011110000111111',
'0b1101100100100000001000100000001011001111010000000100111110010110',
'0b1001010000100000101111100011100010111001111110110010010100111101']
```

```
Depois do modulo chi
['0b1001010011111001001110010001100000010101011110101000011000010',
'0b1100100001010100100011010100110111100101111011001011111011101101',
'0b1000100000011101101000110011100111011001101010010001010001100000',
'0b1010010110001010010111010100001010011001001111111100101100000000',
'0b1100000111111111111100011000101001000011110100011110010110000001']
Antes do modulo iota
['0b1010000010011101001111001000011001100100110001100101010110111001',
'0b1100110111110111100110101111100100011100010111000011001000000',
'0b1100011000111101110001100101000110110101111100010101001011111101',
'0b1001110001100100001111011010110011111000000011010101101101110011',
'0b1001110100011010010010001100110011111010101010110111000011011101']
Depois do modulo iota
['0b1010100000100111010011110010000110011001001100010001010100110000',
'0b1100110111110111100110101111100100011100010111000011001000000',
'0b1100011000111101110001100101000110110101111100010101001011111101',
'0b1001110001100100001111011010110011111000000011010101101101110011',
'0b1001110100011010010010001100110011111010101010110111000011011101']
Valor atual do State Array  após a rodada 15/24
['A8274F2199311530', '25013B552A3C96F7', 'FD267A96B049039F',
'E97D0C4F7D7C152A', '129F272302AF50C2']
['19BEF35F238B8640', 'B200600BC6F122A5', '7D97B27FF668FB50',
'48B00B627C60C4B8', '642A46A6F2F65EED']
['631EE328DAF8A97D', 'D36717DB3BF9C28C', 'E06D674BD834ADDB',
'D3130167A2735662', '881DA339D9A91460']
['9C643DACF80D5B73', 'E0EAA7D3B910CFB9', 'A29CE87EB21C385C',
'0573E20A6349FFD4', 'A58A5D42993FCB00']
['9D1A48CCF556E1BB', 'ABC5CC3548F667C4', '12AF1BDF2D4BD6B8',
'E31A799E354F6680', '60FFF8C521E8F2C1']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(1101454890675396259,1[bits]) = 2202909781350792518
Rotacionar Esquerda(15055353247298665456,1[bits]) = 11663962420887779297
Rotacionar Esquerda(10716207419276992164,1[bits]) = 2985670764844432713
Rotacionar Esquerda(4313717523626795918,1[bits]) = 8627435047253591836
Rotacionar Esquerda(15274286024450736325,1[bits]) = 12101827975191921035
Depois da rotacao de theta A[ 4 ]
['0b1001010011111001001110010001100000010101011110101000011000010',
'0b1100100001010100100011010100110111100101111011001011111011101101',
```

```
'0b1000100000011101101000110011100111011001101010010001010001100000',
'0b1010010110001010010111010100001010011001001111111100101100000000',
'0b1100000111111111111100011000101001000011110100011110010110000001']
Depois de theta
['0b1000011101101011101110100100010010110111111010101001111111101101',
'0b1010111011011111000111100010100110111011010110001000001110000010',
'0b1011101101011000011010101000101111110110111100110000101101001111',
'0b1001011011001111100101001111000010110110011001011101010000101111',
'0b1010011101110100011000101110111000011101011001011101101111101110']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(10189507253173980152,0[bits]) = 10189507253173980152
Rotacionar Esquerda(4391531505261287560,36[bits]) = 17334786540422670895
Rotacionar Esquerda(5067054072054036405,3[bits]) = 36429444429013188010
Rotacionar Esquerda(13342780659753534907,41[bits]) = 2273041769894027849
Rotacionar Esquerda(13282629539782229875,18[bits]) = 9566955479034159445
Rotacionar Esquerda(6279892982839665107,1[bits]) = 12559785965679330214
Rotacionar Esquerda(13846301917433419137,44[bits]) = 15949528877381171862
Rotacionar Esquerda(11619434284395607464,10[bits]) = 150779678441251461
Rotacionar Esquerda(10578170352436525213,45[bits]) = 9733319317796977361
Rotacionar Esquerda(15700214627196113120,2[bits]) = 7460626287655797635
Rotacionar Esquerda(15780725555815112821,62[bits]) = 8556867407381166109
Rotacionar Esquerda(6607254135822096570,6[bits]) = 17035895071004044950
Rotacionar Esquerda(14288650122982346289,43[bits]) = 11705293253453326497
Rotacionar Esquerda(9564225728822434742,15[bits]) = 8813613801969238621
Rotacionar Esquerda(3785564666437776722,61[bits]) = 5084881601732109994
Rotacionar Esquerda(5631925311851522502,28[bits]) = 13107461305355373027
Rotacionar Esquerda(17286391066100774996,55[bits]) = 3060181432143951388
Rotacionar Esquerda(8378545868755675790,25[bits]) = 2811089858454916390
Rotacionar Esquerda(11684149970805785400,21[bits]) = 1137168390928221390
Rotacionar Esquerda(49224123083676452 92,56[bits]) = 78014483291757782 02
Rotacionar Esquerda(2439524455483265005,27[bits]) = 9903321569811158900
Rotacionar Esquerda(6300411723495784898,20[bits]) = 17387794272168015608
Rotacionar Esquerda(13499657032419183439,39[bits]) = 8756589621010580987
Rotacionar Esquerda(10867068187441484847,8[bits]) = 14957844928587378582
Rotacionar Esquerda(6033189038233677294,14[bits]) = 10114455484791231726
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1000110001011101110000111010110010111011011110111001010011101110',
'0b1000110100100010010000011100000001111000010100101100110101010010',
```

```
'0b1000010011000100101001110101000111111101110011101110000101010101',
'0b1101100010001000100111111101011111010100001011001000011100110 10',
'0b1100111100010010111100000010100100111011010101001000011100000 11']
Antes do modulo chi
['0b1000011101101011101110100100010010110111110101010011111110 1101',
'0b1010111011011110001111000101001101110110101100010000011100001 0',
'0b1011101101011000011010101000101111110110111100110000101101001111',
'0b1001011011001111100101001111000010110110011001011101010000101111',
'0b1010011101110100011000101110111000011101011001011101101111011 10']
Depois do modulo chi
['0b1101110001001101110110111010111010000001011110111010010011101000',
'0b1101001100111110000101000000010010000101100000100001011001 0',
'0b1100010011100110000011110101010111101101110010001100110101000101',
'0b1110011010100010011110110101101101100000001110110101110010001',
'0b1101111101111101001101001011000000111011000011111000101100000 11']
Antes do modulo iota
['0b1010111101001001111001001110110001000101011011111000101111011001',
'0b1011011101100101000010101011010001000111001100100000000011100001',
'0b1010110101001101000001011101101110001111101010000100001010000110',
'0b1000101101101001100011110010111101111001110011001101011111110100',
'0b1001110100000000011100100000110011101100000010001110011111111 10']
Depois do modulo iota
['0b1011110100100111100100111011000100010101101111000010111101 1010',
'0b1011011101100101000010101011010001000111001100100000000011100001',
'0b1010110101001101000001011101101110001111101010000100001010000110',
'0b1000101101101001100011110010111101111001110011001101011111110100',
'0b1001110100000000011100100000110011101100000010001110011111111 10']
Valor atual do State Array  após a rodada 16/24
['2F49E4EC456F0BDA', 'D0D01C825B23FED8', '22644D3EC3B73881',
'0EE82CF023944FDE', 'DC4DDBAE817BA4E8']
['B7650AB4473200E1', '745C7A95980934A9', '720E56082BD0B980',
'3675BC45E67EF390', '0699F0A0242C10B2']
['AD4D05DB8FA84286', '24FFA8ECFB120C06', '2742F81484E24D67',
'E59DA03C67D42D34', 'C4E60F55EDC8CD45']
['8B698F2F79CCD7F4', '88D19A8DA614EA77', '0613EDB016D2A307',
'FB7B9F1A6CD3D639', '1CD44F6B6C076B91']
['27401C833B0239FE', '2C7DA2EED927141C', '1985AFD925B70479',
'0FCB76F054389A55', '6FBE9A581D87C583']
```

```
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(2657113439059261468,1[bits]) = 5314226878118522936
Rotacionar Esquerda(7547647370622692120,1[bits]) = 15095294741245384240
Rotacionar Esquerda(3004139973278817558,1[bits]) = 6008279946557635116
Rotacionar Esquerda(7879312978021963533,1[bits]) = 15758625956043927066
Rotacionar Esquerda(11045210232861730743,1[bits]) = 3643676392013909871
Depois da rotacao de theta A[ 4 ]
['0b1101110001001101110110111010111010000001011110111010010011101000',
'0b1101001100111110000101000000010010000101100000100001011001 0',
'0b1100010011100110000011110101010111101101110010001100110101000101',
'0b1110011010100010011110110101101101100000001110110101110010001',
'0b1101111101111101001101001011000000111011000011111000101100000 11']
Depois de theta
['0b1100011101101101111100101001001010000101110110010011011010010001',
'0b1110110111001110110011001110000100000100011101000001011001011',
'0b1101111111000110001001100110100111101001011010100101111100111100',
'0b1111111010001100110010101110110100010100101111110011110 1000',
'0b1110100100111101011001101100100000110010010010101010111111111010']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(841883636409478383,0[bits]) = 841883636409478383
Rotacionar Esquerda(10629083277446326228,36[bits]) = 268664881209765583
Rotacionar Esquerda(9919769180618483123,3[bits]) = 5571177150109658524
Rotacionar Esquerda(12650210546109739201,41[bits]) = 10007424291528175517
Rotacionar Esquerda(263180046171872971,18[bits]) = 247188005745069724
Rotacionar Esquerda(11017254099012325215,1[bits]) = 3587764124315098815
Rotacionar Esquerda(4353081076461225262,44[bits]) = 2617404775481921167
Rotacionar Esquerda(7839238997508980097,10[bits]) = 3047061385540666803
Rotacionar Esquerda(13899410498776964080,45[bits]) = 17545487709303642845
Rotacionar Esquerda(7226193104395330971,2[bits]) = 10458028343871772269
Rotacionar Esquerda(6186267353334397617,62[bits]) = 6158252856760987308
Rotacionar Esquerda(409847732462697392,6[bits]) = 7783510803903081473
Rotacionar Esquerda(5835746251605462871,43[bits]) = 1619812108833284018
Rotacionar Esquerda(8191390142904279351,15[bits]) = 15345930213449775318
Rotacionar Esquerda(7943201330017306185,61[bits]) = 3298743175465857225
Rotacionar Esquerda(13611970904342629084,28[bits]) = 12746515901953505014
Rotacionar Esquerda(9546223692529350290,55[bits]) = 5278849332918210715
Rotacionar Esquerda(6310356335540824118,25[bits]) = 5662176815455610311
Rotacionar Esquerda(5293097909767377723,21[bits]) = 10387589384494460571
```

```
Rotacionar Esquerda(13674113104326647639,56[bits]) = 6322425185613506399
Rotacionar Esquerda(14370408697133872785,27[bits]) = 10677693543859580820
Rotacionar Esquerda(2141982362375717579,20[bits]) = 11079427826560129949
Rotacionar Esquerda(16124617752176123708,39[bits]) = 13055828048158602484
Rotacionar Esquerda(573195578186791400,8[bits]) = 17610859499851737095
Rotacionar Esquerda(8403351197219903482,14[bits]) = 12454993156514946343
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1010110011011001000001100100100101010101111111101001110100100111',
'0b1011011100011101111101011100001100101011100010111100001100100 1',
'0b1101101110001100000010111001111011001011000000111010011100',
'0b1010111101111011100010000110101011010110111100111100111010111111',
'0b1001000100100010011000010101111010100111011100011001011001101101']
Antes do modulo chi
['0b1100011101101101111100101001001010000101110110010011011010010001',
'0b1110110111001110110011001110000100000100011101000001011001011',
'0b1101111111000110001001100110100111101001011010100101111100111100',
'0b1111111010001100110010101110110100010100101111110011110100 0',
'0b1110100100111101011001101100100000110010010010101011111111010']
Depois do modulo chi
['0b1000100010001001000001010000101101010001111111101101111100100111',
'0b1001001100010101111101001100001110111011010011011110011100000 0',
'0b1001111011010101010000110101110011100110010001011101110100111 00',
'0b1010100001011011111000001111100010110110111100111110011100010100',
'0b1001100100100010010111000111100100100100010100011000011001111110']
Antes do modulo iota
['0b1100110000110111000001011010110010011100001101011110111011111',
'0b1111010011110100011111001011110000001010110000000101001011110110',
'0b1100110101101001001110011101100111011100010000000111110111 1001',
'0b1011110001101111110010111001001000001011111011010110111010100100',
'0b1110000101011011000000000101100001101100010101111100101011001000']
Depois do modulo iota
['0b1001100110000110111000001011010110010011100001100011110111011101',
'0b1111010011110100011111001011110000001010110000000101001011110110',
'0b1100110101101001001110011101100111011100010000000111110111 1001',
'0b1011110001101111110010111001001000001011111011010110111010100100',
'0b1110000101011011000000000101100001101100010101111100101011001000']
Valor atual do State Array  após a rodada 17/24
['9986E0B593863DDD', 'A452E2C6960BEE86', '3AABBCCEF574CE96',
```

```
’930ED936C5690E53’, ’8889050B51FEDF27’]
[’F4F47CBC0AC052F6’, ’2BEC18EC3CB149DC’, ’41D1A9798FEF4D9C’,
’635E9891950A9CEB’, ’24C57D30EED379C0’]
[’335A4E7677101F79’, ’DC66D5FEDD5E2401’, ’4D9C37D636AB235F’,
’C4E61B38A168F824’, ’4F6AA1AE7322EE9C’]
[’BC6FCB920BED6EA4’, ’D70CDD592828D68B’, ’2941122BCE9642BA’,
’54F5BBB99499B056’, ’542DF07C5B79E714’]
[’E15B00586C57CAC8’, ’43823C6FF3411B92’, ’A42DFE6F41422494’,
’CEB5015F5522871D’, ’99225C792451867E’]
Antes de theta [’0b0’, ’0b0’, ’0b0’, ’0b0’, ’0b0’]
Rotacionar Esquerda(14363895534497648194,1[bits]) = 10281046995285744773
Rotacionar Esquerda(13513840293474256507,1[bits]) = 8580936513238961399
Rotacionar Esquerda(12607511017982811607,1[bits]) = 6768277962256071599
Rotacionar Esquerda(3326319064131250449,1[bits]) = 6652638128262500898
Rotacionar Esquerda(224082348955456574,1[bits]) = 448164697910913148
Depois da rotacao de theta A[ 4 ]
[’0b1000100010001001000001010000101101010001111111101101111100100111’,
’0b1001001100010101111101001100001110111011010011011110011100000’,
’0b1001111011010101010000110101110011100110010001011101110100111100’,
’0b1010100001011011111000001111100010110110111100111100111000010100’,
’0b1001100100100010010111000111100100100100010100011000011001111110’]
Depois de theta
[’0b1000000100011111010110000110010111001010010111001010101000110 0’,
’0b1000110000001011101011100010001011001101101110101000110001101011’,
’0b1110011110100100011100101011110001010000010010110001101100110111’,
’0b1111110011100011001000110110111001111000000100000001001010111111’,
’0b1100011110110010001111011010110000011100111000011100111101010 1’]
Antes de rho e pi [’0b0’, ’0b0’, ’0b0’, ’0b0’, ’0b0’]
Rotacionar Esquerda(4107855570323343433,0[bits]) = 4107855570323343433
Rotacionar Esquerda(6084526828800567138,36[bits]) = 1143481257876344478
Rotacionar Esquerda(10655136440296254189,3[bits]) = 11454115227531827052
Rotacionar Esquerda(2083798593983011632,41[bits]) = 11652606887392153539
Rotacionar Esquerda(4746767677059137372,18[bits]) = 11544442912703055743
Rotacionar Esquerda(15013707275538708047,1[bits]) = 11580670477367864479
Rotacionar Esquerda(6910101904753430805,44[bits]) = 4688623237986329385
Rotacionar Esquerda(12137007485265083592,10[bits]) = 13636903304917361313
Rotacionar Esquerda(11746892677987012162,45[bits]) = 12810581914219242689
Rotacionar Esquerda(4002496730521420635,2[bits]) = 16009986922085682540
```

```
Rotacionar Esquerda(11533915313197890427,62[bits]) = 16718536883581636318
Rotacionar Esquerda(15810633512358426737,6[bits]) = 15756364810623523958
Rotacionar Esquerda(15503423769028253362,43[bits]) = 3942222771115669466
Rotacionar Esquerda(12968709116753852247,15[bits]) = 1017111743289874941
Rotacionar Esquerda(4510056507469975929,61[bits]) = 2869600072647440943
Rotacionar Esquerda(8419193846717717002,28[bits]) = 2741629635326209987
Rotacionar Esquerda(9549528508349286578,55[bits]) = 6431777292243456004
Rotacionar Esquerda(2539535667382348925,25[bits]) = 8649590626788473980
Rotacionar Esquerda(12911150242152522767,21[bits]) = 15539149777407468979
Rotacionar Esquerda(2985082105312580420,56[bits]) = 4911576871552976915
Rotacionar Esquerda(2326063137345186444,27[bits]) = 14669553658308476592
Rotacionar Esquerda(10091350854557404267,20[bits]) = 16297642669481967802
Rotacionar Esquerda(16691592272068418359,39[bits]) = 2705990422625869352
Rotacionar Esquerda(18222447474638852799,8[bits]) = 16367046932739309564
Rotacionar Esquerda(3597407892207596501,14[bits]) = 2583590427243662459
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1000111101101011000001110011100001110011110101010011000111011',
'0b1001111101001011011110000011111111000110010101111110100010111',
'0b1010000000110110000110011010100111111101011100010000011101111111',
'0b1000100001010010110110100100100010110111111000010001000000010011',
'0b1101111000101110111001100100011111110101100100010000110101101100']
Antes do modulo chi
['0b1000000100011111010110000110010111001010010111001010101000110 0',
'0b1000110000001011101011100010001011001101101110101000110001101011',
'0b1110011110100100011100101011110001010000010010110001101100110111',
'0b1111110011100011001000110110111001111000000100000001001010111111',
'0b1100011110110010001111011010110000011100111000011100111101010 1']
Depois do modulo chi
['0b1100011110010111001010011010000000111001011010101101111010110 11',
'0b1110011111110010000111000000011110110001001001010111101000010111',
'0b1111101000111111000010011010100101111101110100110001011100011111',
'0b1000000011000110010101100000101010111011111100111001000000011101',
'0b1100111101101100101001010000111110110101000010010001110101101100']
Antes do modulo iota
['0b1111101001101000101011100001010110011011010110011100100110 11',
'0b1110101101110100011001101101001000111000001000011110001000011 1',
'0b1000000010110110110010101111100101001010000111010000110010010111',
'0b1111011100101001011100000011100110010011111000000011110100100 01',
```

```
'0b1100110010001001101101000001011010011110000001110100000011110110']
Depois do modulo iota
['0b1000111110100110100010101110000101011001101101011001110000011011',
'0b1110101101110100011001101101001000111000001000011110001000011',
'0b1000000010110110110010101111100101001010000111010000110010010111',
'0b1111011100101001011100000011100110010011111100000001111010010001',
'0b1100110010001001101101000001011010011110000001110100000011110110']
Valor atual do State Array  após a rodada 18/24
['8FA68AE159B59C1B', '801375BAD9E16308', '16ED573325CF3F92',
'CFA62E44E0FCE5B3', '63CB94D01CB56F5B']
['3ADD19B48E087887', 'C3249FC8C4A8603B', '98E7BB13B9EC4D42',
'B1C475C0AD5CE101', 'E7F21C07B1257A17']
['80B6CAF94A1D0C97', '598BB07080BE9FF6', '781D955917277C7F',
'E3A3A828101ED77C', 'FA3F09A97DD3171F']
['7B94B81CC9F01E91', '0DC3F695550197C2', 'FD606F4DEDA322A3',
'858911EDFBA9EF5D', '40632B055DF9C81D']
['CC89B4169E0740F6', 'D9702356CEDED1C7', '7B851DB5F3A95E04',
'81B66889DC61CD51', 'CF6CA50FB5091D6C']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(14848244555658615488,1[bits]) = 11249745037607679361
Rotacionar Esquerda(8138580127791346184,1[bits]) = 16277160255582692368
Rotacionar Esquerda(11096459125540188610,1[bits]) = 3746174177370825605
Rotacionar Esquerda(17368430429772961570,1[bits]) = 16290116785836371525
Rotacionar Esquerda(9426128194934126188,1[bits]) = 405512316158700761
Depois da rotacao de theta A[ 4 ]
['0b1100011110010111001010011010000000111001011010101101111101011011',
'0b1110011111111001000011100000001111011000100100101011111010000010111',
'0b1111101000111111000010011010100101111101110100110001011100011111',
'0b1000000011000110010101100000101010111011111100111001000000011101',
'0b1100111101101100101001010000111111011010100001001000111010110110 0']
Depois de theta
['0b1111111110010101101101010101010111110010011011001111001001000000',
'0b1111011101011000011110110000010010111111111110011100111000011 00',
'0b1100110011000010010100000101100100100110000101010000101000000100',
'0b1101110000111101000010101000000001011001100100000010101010000011 0',
'0b1010011001100101000010010001010010110111101000010000000011101 11']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(16334725273882197688,0[bits]) = 16334725273882197688
```

```
Rotacionar Esquerda(6326160283780389412,36[bits]) = 12583517253178266667
Rotacionar Esquerda(17122925543352659508,3[bits]) = 7856195830854414759
Rotacionar Esquerda(1622044543028460594,41[bits]) = 2663989324115203578
Rotacionar Esquerda(11646208446250164821,18[bits]) = 10629246724995450494
Rotacionar Esquerda(16368111977952326004,1[bits]) = 14289479882195100393
Rotacionar Esquerda(11533962106535883335,44[bits]) = 3685204540041646670
Rotacionar Esquerda(4233369173020233098,10[bits]) = 18431919924683614442
Rotacionar Esquerda(7996057784712611262,45[bits]) = 6392804553578124969
Rotacionar Esquerda(13421960486557418427,2[bits]) = 16794353798810570478
Rotacionar Esquerda(16942486147589342935,62[bits]) = 18070679592179499445
Rotacionar Esquerda(7283764540954211335,6[bits]) = 4992328778330735065
Rotacionar Esquerda(9650949057369228602,43[bits]) = 1308810467077539631
Rotacionar Esquerda(41364847734430694,15[bits]) = 8831013181027713097
Rotacionar Esquerda(9689362536657086273,61[bits]) = 3517013326295829736
Rotacionar Esquerda(6721124558433499646,28[bits]) = 15008619179409695666
Rotacionar Esquerda(2532255170683026764,55[bits]) = 11966506421176277662
Rotacionar Esquerda(8161575040697436977,25[bits]) = 9505108793481135994
Rotacionar Esquerda(16868845009418452 64,21[bits]) = 10413439469708569888
Rotacionar Esquerda(1393439536683946268,56[bits]) = 2023055756252153873
Rotacionar Esquerda(18416825631945585216,27[bits]) = 12651569655110741418
Rotacionar Esquerda(8911565392824821516,20[bits]) = 15575136148451211971
Rotacionar Esquerda(7377221836520196612,39[bits]) = 9603084199319049801
Rotacionar Esquerda(15869852209800828166,8[bits]) = 4398469492910655196
Rotacionar Esquerda(5994999783764295799,14[bits]) = 11611008764569572556
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1010000100100010100101101111010000100000000111011101010011001100',
'0b1100001100111011110000111111001000000000101101100011001110100 0',
'0b1001001110000010101010111001010010001001010101101000011001111110',
'0b1110000010011010101100111101111000000011100000000100000010001',
'0b1110100100010001100001111100110010111010010101100000111011101110']
Antes do modulo chi
['0b1111111110010101101101010101010111110010011011001111001001000000',
'0b1111011101011000011110110000010010111111111100111001110000 1100',
'0b1100110011000010010100000101100100100110000101010001010000000100',
'0b1101110000111101000010101000000010110011001000000101010100000110',
'0b1010011001100101000010010001010010110111101000010000000 1110111']
Depois do modulo chi
['0b1011000000100110111101101111010000100000100111011101010010001010',
```

```
'0b1110001110101001011100101111001001000000101110000101001010100 1',
'0b1001001010000010101110110001010000000001010111111000011101101110',
'0b1110000110011110101100011100010011000001110000001100000010000',
'0b1110110100000001100001111110110011111010011100100001010011100100']
Antes do modulo iota
['0b1110001010111001000111100011100100011101011111111111101110011001',
'0b1111010101001011010100111010111001100010111011000010001010010110',
'0b1000100111011101100000000111010010100111111011011100100110010 11',
'0b1111111011011001001100100001000010000100111111010100001010110101 0',
'0b1111101110000011111100110101101110010101110100101000010111110100']
Depois do modulo iota
['0b1110001010111001000111100011100100011101011111111011101110010011',
'0b1111010101001011010100111010111001100010111011000010001010010110',
'0b1000100111011101100000000111010010100111111011011100100110010 11',
'0b1111111011011001001100100001000010000100111111010100001010110101 0',
'0b1111101110000011111100110101101110010101110100101000010111110100']
Valor atual do State Array  após a rodada 19/24
['E2B91E391D7F7B93', 'B3A652010FD47A4E', '3309D2DF786557E3',
'D213F0052F40C710', 'B026F6F4209DD48A']
['F54B53AE62EC2296', 'C894F24A70C3BCCB', '4D4EE05BF7D2F9E7',
'98B6CEDD935675BB', '38EA5CBC902E14A9']
['44EEC03A53F6E4CB', '794A5033CA48015D', '0168C512EBE00758',
'7946C4995255665D', '9282BB14015F876E']
['FED9321084FA856A', 'AEA5A8757471942A', 'FBDA05CF373E20FA',
'D90E2E310A37A5E3', '1C33D63898381810']
['FB83F35B95D285F4', '86A9F63C51E55B2C', '4C4481F38A921C4D',
'363E143D00D154EB', 'ED0187ECFA7214E4']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(3059261574797265118,1[bits]) = 6118523149594530236
Rotacionar Esquerda(14461467056102151659,1[bits]) = 10476190038494751703
Rotacionar Esquerda(15912273349211137534,1[bits]) = 13377802624712723453
Rotacionar Esquerda(16968508452435807161,1[bits]) = 15490272831162062707
Rotacionar Esquerda(6216740887386242384,1[bits]) = 12433481774772484768
Depois da rotacao de theta A[ 4 ]
['0b1011000000100110111101101111010000100000100111011101010010001010',
'0b1110001110101001011100101111001001000000101110000101001010100 1',
'0b1001001010000010101110110001010000000001010111111000011101101110',
'0b1110000110011110101100011100010011000001110000001100000010000',
```

```
'0b1110110100000001100001111110110011111010011100100001010011100100']
Depois de theta
['0b1100000001111001101011110111010110111110101000111000101111010100',
'0b1001000101101010000010100111101000011100001000001001011111110111',
'0b1110001011011101111000101001010110011111011000011101100000110000',
'0b1101100011011001000111110111001000001100000011001000111010011110',
'0b1001110101011110110111100110110101100100010011000100101110111010']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(6713744249483895190,0[bits]) = 6713744249483895190
Rotacionar Esquerda(5394836563838138515,36[bits]) = 992913102207251545
Rotacionar Esquerda(18121320318575754958,3[bits]) = 15843354032639178359
Rotacionar Esquerda(4705187368260001647,41[bits]) = 10790306409709696749
Rotacionar Esquerda(4906372385002938353,18[bits]) = 13745442959204356187
Rotacionar Esquerda(8395547200443051209,1[bits]) = 16791094400886102418
Rotacionar Esquerda(1130502436721076812,44[bits]) = 10566782810081697767
Rotacionar Esquerda(13722181642975614938,10[bits]) = 13541762314060917497
Rotacionar Esquerda(7602361799124124333,45[bits]) = 6941644822876315481
Rotacionar Esquerda(4723534456690822571,2[bits]) = 447393753053738669
Rotacionar Esquerda(11590854171001361600,62[bits]) = 2897713542750340400
Rotacionar Esquerda(16041204985170737860,6[bits]) = 12066194996901884215
Rotacionar Esquerda(10573303728681600123,43[bits]) = 730670100819854739
Rotacionar Esquerda(7496570769384629209,15[bits]) = 10786885679140615172
Rotacionar Esquerda(16111538992269451118,61[bits]) = 15849000429315845101
Rotacionar Esquerda(14725084948083492232,28[bits]) = 16510051478724255787
Rotacionar Esquerda(9727560132918867747,55[bits]) = 10503379073403121851
Rotacionar Esquerda(7426213830967059653,25[bits]) = 4925184037402582636
Rotacionar Esquerda(14359688424112301947,21[bits]) = 10470464603274406139
Rotacionar Esquerda(2916052749066327667,56[bits]) = 8298014145412752982
Rotacionar Esquerda(13869309447708576724,27[bits]) = 12534956330693414267
Rotacionar Esquerda(5239099501302533111,20[bits]) = 6039574510812040016
Rotacionar Esquerda(16347471354676762672,39[bits]) = 12748591520650709711
Rotacionar Esquerda(7812777478413764430,8[bits]) = 7822674513292119660
Rotacionar Esquerda(11339745473181010874,14[bits]) = 13230266268787844951
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1011011111001101101011001000100110001001011101110101001101010111',
'0b1101101111110010111101011110101100111010011000100110101111101101',
'0b1011111011000001100111111000101101111111110001010001000001011011',
'0b1110011001010000111011111100110100001100111111001100110010010110',
```

```
’0b1100011010101110110001111110111110101111111001101101010101’]
Antes do modulo chi
[’0b1100000001111001101011110111010110111110101000111000101111010100’,
’0b1001000101101010000010100111101000011100001000001001011111110111’,
’0b1110001011011101111000101001010110011111011000011101100000110000’,
’0b1101100011011001000111110111001000001100000011001000111010011110’,
’0b1001110101011110110111100110110101100100010011000100101110111010’]
Depois do modulo chi
[’0b1101010001101110011001001110100001001001111110001110010011011 0’,
’0b1100100100110010011101001110111100001001010100100110000010111101’,
’0b1011100010110011100100111000000111101011111001110011000001111110’,
’0b1110011001010101111011011000110100011111001101001010110010101 10’,
’0b1001011111111010000110110101111010100111001111001110001100010011 0’]
Antes do modulo iota
[’0b1010101001011110001111011010110001101010000111101100001100001 10’,
’0b1101101000100010111101011010001100011001100010011010100000011 00’,
’0b1010100100001101101100111110010100001000110111011101011111011010’,
’0b1111111011101000111000101110111100100000110001100011111011011’,
’0b1000000110101011111101111100000001001111100000001111011101 00’]
Depois do modulo iota
[’0b1101010100101111000111101101011010110101000011110110000110001100’,
’0b1101101000100010111101011010001100011001100010011010100000011 00’,
’0b1010100100001101101100111110010100001000110111011101011111011010’,
’0b1111111011101000111000101110111100100000110001100011111011011’,
’0b1000000110101011111101111100000001001111100000001111011101 00’]
Valor atual do State Array  após a rodada 20/24
[’D52F1ED6B50F618C’, ’03E8C0FB25BFB78F’, ’2CB29596CFDBC297’,
’D96A92D40279E87B’, ’351B993A127E3936’]
[’6D117AD18CC4D40C’, ’73D1E8349F3C8258’, ’407CB6461437D6D3’,
’4458A520A4FD5F5B’, ’C93274EF095260BD’]
[’A90DB3E508DDD7DA’, ’8FF5856D90EFF137’, ’D619C240F34E0E7F’,
’2D8BD962065F87EC’, ’B8B39381EBE7307E’]
[’1FDD1C5DE418C7DB’, ’09D73F789C44605D’, ’D9E641B3CC4D28AB’,
’1967BF4791ED3D2D’, ’732AF6C68F9A5656’]
[’081ABF7C04F80F74’, ’94D1B91CA37A449B’, ’B2ED384C0BD34ACF’,
’BDBC5782985DF7FD’, ’97F436BD4E79C626’]
Antes de theta [’0b0’, ’0b0’, ’0b0’, ’0b0’, ’0b0’]
Rotacionar Esquerda(7118550290773762086,1[bits]) = 14237100581547524172
```

```
Rotacionar Esquerda(15122129255325792351,1[bits]) = 11797514436942033087
Rotacionar Esquerda(1468743384870681116,1[bits]) = 2937486769741362232
Rotacionar Esquerda(11548564454382303621,1[bits]) = 4650384835055055627
Rotacionar Esquerda(501153842935016181,1[bits]) = 1002307685870032362
Depois da rotacao de theta A[ 4 ]
['0b1101010001101110011001001110100001001001111110001110010011 0110',
'0b1100100100110010011101001110111100001001010100100110000010111101',
'0b1011100010110011100100111000000111101011111001110011000001111110',
'0b1110011001010101111011011000110100011111001101001010110010101 10',
'0b1001011111111010000110110101111010100111001111001110001100010 0110']
Depois de theta
['0b1011001001000101110110111011100001100011111000100101101100000 0',
'0b1101000010111000100110110011101100000011110101001100111010010 11',
'0b1010000100111001011111000101010111100001011000011001111110001000',
'0b1101010101000000001100100010010100001010001110011111001101000 00',
'0b1000111001111110110110010110100101000100111111110110100111010000']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(12754184754547546181,0[bits]) = 12754184754547546181
Rotacionar Esquerda(630947543182339525,36[bits]) = 8403396193225881385
Rotacionar Esquerda(14762054410608635411,3[bits]) = 7415970842611773598
Rotacionar Esquerda(8794956122882965010,41[bits]) = 3169448868621057535
Rotacionar Esquerda(7911231211735168701,18[bits]) = 6592282307723638569
Rotacionar Esquerda(12008150109833522629,1[bits]) = 5569556145957493643
Rotacionar Esquerda(15464424510709749778,44[bits]) = 3278951936098994443
Rotacionar Esquerda(3078423039057177469,10[bits]) = 16358699463925953706
Rotacionar Esquerda(12437388992843495959,45[bits]) = 5134936538197299833
Rotacionar Esquerda(3575010383568838353,2[bits]) = 14300041534275353412
Rotacionar Esquerda(7402988663704835721,62[bits]) = 6462433184353596834
Rotacionar Esquerda(752823685818270413,6[bits]) = 11287227744950203202
Rotacionar Esquerda(11247710527310994017,43[bits]) = 6400473654842953126
Rotacionar Esquerda(10657881572646599861,15[bits]) = 3704569014553070068
Rotacionar Esquerda(17934212420534165201,61[bits]) = 4547619561780464602
Rotacionar Esquerda(5206010422583649071,28[bits]) = 6408980929389524846
Rotacionar Esquerda(15351998246743495695,55[bits]) = 570416326860130410
Rotacionar Esquerda(13609382552987110584,25[bits]) = 12039920714262363258
Rotacionar Esquerda(9814007100836198009,21[bits]) = 7981381392270427723
Rotacionar Esquerda(3236315388978429097,56[bits]) = 12190375249398018172
Rotacionar Esquerda(3211478774285113024,27[bits]) = 8126680338599807927
```

```
Rotacionar Esquerda(15039941633373425483,20[bits]) = 12947916329423014793
Rotacionar Esquerda(11617453422074240904,39[bits]) = 12740617721360558832
Rotacionar Esquerda(7683168531627440544,8[bits]) = 11536272283412308074
Rotacionar Esquerda(10267883246603692496,14[bits]) = 13139904197496677279
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1011011001011010010100010011111111011010011101000010001110011111',
'0b1111110001110001100011111001011010100110001010110010101111011010',
'0b1011011011111000111110111010100110110101111010110111011100101001',
'0b1010100100101100111010011011001110110011000101010011000001111100',
'0b1100011001110011111110100000000101011001111010000011110110100100']
Antes do modulo chi
['0b1011001001000101110110111011100001100011111000100101101100000',
'0b1101000010111000100110110011101100000011110101001100111101001011',
'0b1010000100111001011111000101010111100001011000011001111110001000',
'0b1101010101000000001100100010010100001010001110011111100110100000',
'0b1000111001111110110110010110100101000100111111110110100111010000']
Depois do modulo chi
['0b1011110110101101001011001001101111001101000110100101000101010101',
'0b1001110000011100010110101010100110101001101101101100001101011011',
'0b1100101111011000001111011100010001110010011001011001011101101001',
'0b1010110100110100111100011111001100111111000011000000010001110100',
'0b1100000000110011011101100000001010111011010100000011001100001100']
Antes do modulo iota
['0b1110000010101101111101111111010110011011101010110101110011100001',
'0b1110010111011100001000000001011111110100000011000011101111000',
'0b1101110010110010010110111000111010001010010101011010111110110011',
'0b1111001111000110110001101001101100000011110000000100101100110101',
'0b1110100110101010011011001110110111111011000111101100101110011001 0']
Depois do modulo iota
['0b1100000101011011111011111110101000110111010101111011100011000000',
'0b1110010111011100001000000001011111110100000011000011101111000',
'0b1101110010110010010110111000111010001010010101011010111110110011',
'0b1111001111000110110001101001101100000011110000000100101100110101',
'0b1110100110101010011011001110110111111011000111101100101110011001 0']
Valor atual do State Array  após a rodada 21/24
['60ADF7F51BABDC60', '0B81BD7F8AD28B42', 'C8CB4DC9AF6D2C32',
'6E663656EBB35E0B', 'BB5A59379A34A295']
['1CBB8402FE818778', 'B2B018CFB1D589E8', '5EF6D0543A05391C',
```

```
’07A3F1811B6DA65D’, ’9C1C5AA9A9B6C35B’]
[’6E592DC7452AD7B3’, ’9CAD5B50E030B342’, ’FC720949B37DAB7B’,
’A41A128F18FBE8E8’, ’CBD83DC472659769’]
[’F3C6C69B03C04B35’, ’64F6DDD0944B3E7D’, ’6B016E6E27A8C4A2’,
’63AA41E01E3A4277’, ’AD34F1F33F0C0474’]
[’E9AA6CEDF63D9732’, ’0CDAA6048BD51D65’, ’74CC14523CAE28F0’,
’32702C495BEC795D’, ’C0337602BB50330C’]
Antes de theta [’0b0’, ’0b0’, ’0b0’, ’0b0’, ’0b0’]
Rotacionar Esquerda(5598120798505767664,1[bits]) = 11196241597011535328
Rotacionar Esquerda(8467592930634453511,1[bits]) = 16935185861268907022
Rotacionar Esquerda(11242595392945335188,1[bits]) = 4038446712181118761
Rotacionar Esquerda(9338699449737396703,1[bits]) = 230654825765241791
Rotacionar Esquerda(586580959415849132,1[bits]) = 1173161918831698264
Depois da rotacao de theta A[ 4 ]
[’0b1011101101011010010110010011011111001101000110100101000101010101’,
’0b1001110000011100010110101010100110101001101101101100001101011011’,
’0b1100101111011000001111011100010001110010011001011001011101101001’,
’0b1010110100110100111100011111001100111111000011000000010001110100’,
’0b1100000000110011011101100000001010111011010100000011001100001100’]
Depois de theta
[’0b1101110001100000001001010010101001110000111101001010000101100 1’,
’0b1000001011110000010101101010010101111101111110100100110010111’,
’0b1000111100110100110110110111001011101000110110000011101101001 01’,
’0b1000010111011010100001100011100011100100000101100011101011100 0’,
’0b1001100011100010010011001111111011110101011001101110011100000 0’]
Antes de rho e pi [’0b0’, ’0b0’, ’0b0’, ’0b0’, ’0b0’]
Rotacionar Esquerda(8815026850125527135,0[bits]) = 8815026850125527135
Rotacionar Esquerda(451265688433681223,36[bits]) = 2780467477269937163
Rotacionar Esquerda(8404172124125803404,3[bits]) = 11893144771877772387
Rotacionar Esquerda(16806999885833875210,41[bits]) = 5863147758666035870
Rotacionar Esquerda(17533321695397769997,18[bits]) = 8988880660009045323
Rotacionar Esquerda(16764531869127114720,1[bits]) = 15082319664544677825
Rotacionar Esquerda(5878940624552361290,44[bits]) = 16705158446279137760
Rotacionar Esquerda(9190565662797023200,10[bits]) = 3299761112280433150
Rotacionar Esquerda(9786590574907804383,45[bits]) = 1827319030873642866
Rotacionar Esquerda(17292854529176299975,2[bits]) = 13831185895576545055
Rotacionar Esquerda(13650614009867725291,62[bits]) = 17247711557749095034
Rotacionar Esquerda(3120190264670092485,6[bits]) = 15224736201790402890
```

```
Rotacionar Esquerda(9928745709303131810,43[bits]) = 10625421222431486305
Rotacionar Esquerda(2214252289267794299,15[bits]) = 5574572827417087837
Rotacionar Esquerda(105799980851260713,61[bits]) = 2319068006820101541
Rotacionar Esquerda(1790088395693199283,28[bits]) = 10798920501767797438
Rotacionar Esquerda(8147693692313499621,55[bits]) = 17489880018440324283
Rotacionar Esquerda(15180384408233261392,25[bits]) = 7015832456572262174
Rotacionar Esquerda(1521051602572448719,21[bits]) = 10084979939182027643
Rotacionar Esquerda(4954436739189483749,56[bits]) = 16520542303197956264
Rotacionar Esquerda(3969933287580575833,27[bits]) = 6116426242007285834
Rotacionar Esquerda(1179392061015542167,20[bits]) = 12487069944803034592
Rotacionar Esquerda(5159556966395944357,39[bits]) = 3895282328686484666
Rotacionar Esquerda(2411292282717572792,8[bits]) = 8548269943283431457
Rotacionar Esquerda(5508226149330303424,14[bits]) = 5305222040564798236
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1001001100111111110111101010110011011100111000000010011000111100',
'0b1000000010111011111100000100001011010001111100001111011010010101',
'0b1111100101111101110101101010101011100110000110111110011010100101 1',
'0b1110010101000100110000011011000111111011011101101100010001010100 0',
'0b1011111111111001000111110010010101001000000111011101001110001111 1']
Antes do modulo chi
['0b1101110001100000001001010010101001110000111101001010000101100 1',
'0b1000001011110000010101101010010101111101111110100100110010111',
'0b1000111100110100110110110111001011101000110110000011101101001 01',
'0b1000010111011010100001100011100011100100000101100011101011100 0',
'0b1001100011100010010011001111111011110101011001101110011100000 0']
Depois do modulo chi
['0b1100110000011111010011100101111001001110011001001001001010111100',
'0b1000001011000111111100010100001111000000110000111000111001010 1',
'0b1111110101111101110101101111110111011100100110101100110101000001',
'0b1100011101010010110101010110000001110100101110111010110100101010 01',
'0b1010111101010010101111111011110101000001101101101101000111001111 0']
Antes do modulo iota
['0b1101010011101000101010001110001010110111010101101011000010111 10',
'0b1001010111011001011110001111001101110111100011011110001010111101',
'0b1111000101011011001101111110000011110101100010010101110011101010 1',
'0b1011101101010001110001001001101010000001011010001000001101111 10',
'0b1110101101011010011111001100011001001100001010001110001001111010']
Depois do modulo iota
```

```
['0b1110101001110100010101000111000101011101010111101100011011110',
'0b1001010111011001011110001111001101110111100011011110001010111101',
'0b1111000101011011001101111110000011110101100010010101110011110101',
'0b1011101101010001110001001001101010000001011010001000001101111110',
'0b1110101101011010011111001100011001001100001010001110001001111010']
Valor atual do State Array  após a rodada 22/24
['EA7454715BABD8DE', 'EF54AE8953179BFA', 'D37FF04A49FCE165',
'B9B50BF368E18B38', 'CC1F4E5E4E6492BC']
['95D978F3778DE2BD', 'B519FB2689F946F0', '8528FC28EE65B4E6',
'8C8AF0111F099568', '082C7F143C0C38E5']
['F15B37C1EB12B9D5', 'C5E9A0E056B5996B', '69435F3669941254',
'F7E08EBB078E8AA1', '7EBEEB7DDC9ACD41']
['5DA8E24D40B441BE', '6682F0306493720A', '8DCB1ABFDB5F815E',
'5DFDF43189ADCF1F', 'C752D581D2EEB4A9']
['EB5A7CC64C28E27A', 'B3E88C662F9EA6BF', '98AEF8AB4D27D9BB',
'115215D770EAAAFE', 'AF52BF7A836DA39E']
Antes de theta ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(5390396547820589268,1[bits]) = 10780793095641178536
Rotacionar Esquerda(3058420511881371442,1[bits]) = 6116841023762742884
Rotacionar Esquerda(10263867101137727760,1[bits]) = 2080990128565903905
Rotacionar Esquerda(15171977114186510383,1[bits]) = 11897210154663469151
Rotacionar Esquerda(4036498263487553650,1[bits]) = 8072996526975107300
Depois da rotacao de theta A[ 4 ]
['0b1100110000011111010011100101111001001110011001001001001010111100',
'0b1000001011000111111100010100001111000000110000111000111001010101',
'0b1111110101111101110101101111110111011100100110101100110101000001',
'0b1100011101010010110101010110000001110100101110111010110100100101001',
'0b1010111101010010101111111011110101000001101101101101000111001111 0']
Depois de theta
['0b1100100110011011010001011100000101000000010101001000110100100 0',
'0b1111011001010101111000000011101000100010011111011000100100010001',
'0b1000000011000111011101000101001111000010111010110111111001011010 1',
'0b1110010010101101001010101011111100110010011111000001010101110 1',
'0b1010001001010110010000001010100100111010001110000010010011010 10']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(12494382356690864473,0[bits]) = 12494382356690864473
Rotacionar Esquerda(15188910963464188730,36[bits]) = 7285473641072795840
Rotacionar Esquerda(13135756834273945682,3[bits]) = 12852334305643807381
```

```
Rotacionar Esquerda(1925359974778146873,41[bits]) = 14997112332990112866
Rotacionar Esquerda(12414979660234142717,18[bits]) = 8711360064045625643
Rotacionar Esquerda(9490009933753976300,1[bits]) = 533275793798400985
Rotacionar Esquerda(15708023813159213286,44[bits]) = 13082514501013008171
Rotacionar Esquerda(12181752829735044989,10[bits]) = 4115903821029176996
Rotacionar Esquerda(749030717724388380,45[bits]) = 3279607119806923770
Rotacionar Esquerda(16073183441845008553,2[bits]) = 8952501546251379367
Rotacionar Esquerda(9606266157722604432,62[bits]) = 2401566539430651108
Rotacionar Esquerda(15206224157914220051,6[bits]) = 13967654273613399284
Rotacionar Esquerda(4570308444099272865,43[bits]) = 794051884686673380
Rotacionar Esquerda(15844994830406710187,15[bits]) = 67319041380396518 26
Rotacionar Esquerda(14880272283853826894,61[bits]) = 15695092090763892073
Rotacionar Esquerda(3954119974753563733,28[bits]) = 12459040441593888178
Rotacionar Esquerda(279259223376685573,55[bits]) = 180689413265477429
Rotacionar Esquerda(8685858609040029132,25[bits]) = 14213979361399346364
Rotacionar Esquerda(15174638049091530866,21[bits]) = 11314703926912766692
Rotacionar Esquerda(11401079124207162771,56[bits]) = 10637001788904340821
Rotacionar Esquerda(3631820429830988616,27[bits]) = 9403701952184006283
Rotacionar Esquerda(17750340046642055441,20[bits]) = 261815539442083166
Rotacionar Esquerda(9279513460321516725,39[bits]) = 8484318530365958625
Rotacionar Esquerda(4119468403073156445,8[bits]) = 3119498985283607865
Rotacionar Esquerda(5848804088855925354,14[bits]) = 14417472968069911626
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1100100000010101001001110100011100000100100110101001010001001010',
'0b1101100111010000001010110001100110110011000001100101010101101001',
'0b1111000111001001111011111010010110011111111011010110001001010 11',
'0b1001001110011110001110001100010100001110110101100111110101010101',
'0b1111100011110110101100101110010101001101110010111000101010011 1']
Antes do modulo chi
['0b1100100110011011010001011100000101000000010101001000110100 1000',
'0b1111011001010101111000000011101000100010011111011000100100010001',
'0b1000000011000111011101000101001111000010111010110111110010110101',
'0b1110010010101101001010101011111100110010011111000001010101 1101',
'0b1010001001010110010000001010100100111010001110000010010011010 10']
Depois do modulo chi
['0b1101100010011111001011100101011011000101000110101111001001101000',
'0b1101101011010000001010111000000100110011000101000101010100100101',
'0b1011100001110101111101111101111001000011111001100011010100001111',
```

```
’0b1111011010000101001010100110000000100010110110101111110100010101’,
’0b1111110101111000100110010101001000100110011011001101001101101110’]
Antes do modulo iota
[’0b1010011101100101111101101110111000101100010011111000100110011101’,
’0b1110010111011111011110110000100010101000011011111111100110011’,
’0b1101100110101100111110001101010000011010110001101111010001’,
’0b1001101010000100000101001000010001001001110010011011000001010111’,
’0b1010100011010100001111011001011001001001001100001000100001000100’]
Depois do modulo iota
[’0b1010011101100101111101101110111010101100010011111000100110011100’,
’0b1110010111011111011110110000100010101000011011111111100110011’,
’0b1101100110101100111110001101010000011010110001101111010001’,
’0b1001101010000100000101001000010001001001110010011011000001010111’,
’0b1010100011010100001111011001011001001001001100001000100001000100’]
Valor atual do State Array  após a rodada 23/24
[’A765F6EEAC4F899C’, ’218E8B9BE9C6BD2B’, ’4B1509FA677A01EE’,
’B86537CE243F5BF5’, ’D89F2E56C51AF268’]
[’1CBBEF61150DFF33’, ’0E2127D0B1976434’, ’620C83E5C7771694’,
’09A4C508E286AB68’, ’DAD02B8133145525’]
[’0366B3E3506B1BD1’, ’EBDF9E8ADB35CDF5’, ’95E662F9D803B4BE’,
’2C48AFCC8F0C57E9’, ’B875F7DE43E6350F’]
[’9A842908939360AF’, ’217B34AD0D5DA992’, ’BB8C9877D81FE6A1’,
’5D6C063F38D4EF78’, ’F6852A6022DAFD15’]
[’546A1ECB24984424’, ’8281D025745D8B37’, ’59A3D64860C8AB64’,
’D160623754896022’, ’7EBC4CA9133669B6’]
Antes de theta [’0b0’, ’0b0’, ’0b0’, ’0b0’, ’0b0’]
Rotacionar Esquerda(7461011345914476111,1[bits]) = 14922022691828952222
Rotacionar Esquerda(6832144088894664193,1[bits]) = 13664288177789328386
Rotacionar Esquerda(1262477951935981614,1[bits]) = 2524955903871963228
Rotacionar Esquerda(3603887681396016865,1[bits]) = 7207775362792033730
Rotacionar Esquerda(8527176320996755957,1[bits]) = 17054352641993511914
Depois da rotacao de theta A[ 4 ]
[’0b1101100010011111001011100101011011000101000110101111001001101000’,
’0b1101101011010000001010111000000100110011000101000101010100100101’,
’0b1011100001110101111101111101111001000011111001100011010100001111’,
’0b1111011010000101001010100110000000100010110110101111110100010101’,
’0b1111110101111000100110010101001000100110011011001101001101101110’]
Depois de theta
```

```
['0b1001011011011100101100000010100101110010110110010010011010 1100',
'0b1001111111100000101001110111011010101010111000111011011100001',
'0b1000101010111011111101011000001011011010010010101000111011001011',
'0b1011101011010010100000111100101110111011101100100011011010001',
'0b1000001110010100010011101111010110001010100110101101001001110010']
Antes de rho e pi ['0b0', '0b0', '0b0', '0b0', '0b0']
Rotacionar Esquerda(6517780400579011555,0[bits]) = 6517780400579011555
Rotacionar Esquerda(16261890440960251212,36[bits]) = 6636428770097853222
Rotacionar Esquerda(18334307671603638702,3[bits]) = 17547252856862248311
Rotacionar Esquerda(7463046618860686032,41[bits]) = 13697030653242161094
Rotacionar Esquerda(12212678725213630043,18[bits]) = 11124261961732105712
Rotacionar Esquerda(16895636113347520732,1[bits]) = 15344528152985489849
Rotacionar Esquerda(14255854127879221699,44[bits]) = 8006340609557972704
Rotacionar Esquerda(2317188414512060418,10[bits]) = 11617695025527261312
Rotacionar Esquerda(16901134954011573349,45[bits]) = 9767389850076854873
Rotacionar Esquerda(5293418659716013760,2[bits]) = 2726930565154503425
Rotacionar Esquerda(1122994687193278461,62[bits]) = 4892434690225707519
Rotacionar Esquerda(2777638674681262215,6[bits]) = 11748178516214817225
Rotacionar Esquerda(15088966079721493165,43[bits]) = 13733004182925189965
Rotacionar Esquerda(18378130400970768562,15[bits]) = 2169948688122216326
Rotacionar Esquerda(2099647220599803255,61[bits]) = 16403356967070833070
Rotacionar Esquerda(9417792729144801334,28[bits]) = 7984578138995370889
Rotacionar Esquerda(3707388626424252587,55[bits]) = 6168165283653823396
Rotacionar Esquerda(1630057411364369450,25[bits]) = 2922760739949985345
Rotacionar Esquerda(7474719086052969659,21[bits]) = 3242029475783571313
Rotacionar Esquerda(16985305583271314401,56[bits]) = 16279307508468439171
Rotacionar Esquerda(2717689323158129068,27[bits]) = 5973376526189443424
Rotacionar Esquerda(2880097993732189921,20[bits]) = 11374592439245504386
Rotacionar Esquerda(4998421103807991499,39[bits]) = 2686227452144566637
Rotacionar Esquerda(841372946696390353,8[bits]) = 12477289543470862603
Rotacionar Esquerda(9481289932021092978,14[bits]) = 1422401525453201637
Depois de rho e pi e da rotacao em A[ 4 ][ 4 ]
['0b1001110111101011000101010011010110100100111001010000011100101',
'0b1110001110100100011011100100000010111010001011111001100110101110',
'0b1001101001100001010100010101000001011100101101110101001011110000',
'0b1110000111101011101101111110110101101111100111000101100010000011',
'0b1001011101100000000000111001101010111100110000011110110000000 1']
Antes do modulo chi
```

```
['0b1001011011011100101100000010100101110010110110010010011010 1100',
'0b1001111111100000101001110111011010101010111000111011101110 0001',
'0b1000101010111011111010110000010110110100100101010001110110 01011',
'0b1011101011010010100000111100101110111011101100100011011010 0001',
'0b1000001110010100010011101111010110001010100110101101001001 110010']
Depois do modulo chi
['0b1101101011000101010010111001101001011111100000101001001110 0101',
'0b1110010101101000110111001001100001111000011111111001101101 01100',
'0b1011100101101000000110010100000100101001001111101000010110 110000',
'0b1110110111110011111100110111011110111010101001100000110101 0000101',
'0b1100011100000010010000011001101010011100110100010110010000 0001']
Antes do modulo iota
['0b1100101011110010100011000011111111001100100000101100001011101110',
'0b1100110010101011111110000010011010010000001010101011111111 00',
'0b1101110001110100101001100110111000101000100011110100111110 111001',
'0b1111001111000111101110100101110111110000000001101101100011 1100000',
'0b1100011101000110010111101101111011100110101100101011001101 10110']
Depois do modulo iota
['0b1001010111100101000110000111111010011001000001001000001011 100110',
'0b1100110010101011111110000010011010010000001010101011111111 00',
'0b1101110001110100101001100110111000101000100011110100111110 111001',
'0b1111001111000111101110100101110111110000000001101101100011 1100000',
'0b1100011101000110010111101101111011100110101100101011001101 10110']
Valor atual do State Array  após a rodada 24/24
['4AF28C3F4C8242E6', '6F763C7DEE3492D0', 'AD940C8D16A5A3C9',
'64BC8CB8DF6FB473', '36B152E697E0A4E5']
['0CCABF826902ABFC', '99D20BDE76964D8A', '93A41F8127120CD1',
'8BC63CD2DC961858', '72B46E4C3C3FCDAC']
['DC74A66E288F4FB9', '2629EE8E929EE0C3', '3ACEFA28DD051AB1',
'E9BA9AB532C68102', 'B9681941293E85B0']
['F3C7BA5DE00DB1E0', '421C700B5AC40420', '40D8DB3ABEE40881',
'0C1934D5C078DEE6', 'EDF3F36F754C1A85']
['63A32F6F735959B6', 'CF8939E828151D26', '248F658225EA896C',
'FC30CBC674A13738', '31C09066A7345901']
Valor após ter absorvido: e642824c3f8cf24a d09234ee7d3c766f c9a3a5168d0c94ad
73b46fdfb88cbc64 e5a4e097e652b136 fcab026982bfca0c 8a4d9676de0bd299
d10c1227811fa493 581896dcd23cc68b accd3f3c4c6eb472 b94f8f286ea674dc
c3e09e928eee2926 b11a05dd28face3a 0281c632b59abae9 b0853e29411968b9
```

```
e0b10de05dbac7f3 2004c45a0b701c42 8108e4be3adbd840 e6de78c0d534190c
851a4c756ff3f3ed b65959736f2fa363 261d1528e83989cf 6c89ea2582658f24
3837a174c6cb30fc 015934a76690c031
Valor após ter espremido: e642824c3f8cf24a d09234ee7d3c766f c9a3a5168d0c94ad
73b46fdfb88cbc64 e5a4e097e652b136 fcab026982bfca0c 8a4d9676de0bd299
d10c1227811fa493 581896dcd23cc68b accd3f3c4c6eb472 b94f8f286ea674dc
c3e09e928eee2926 b11a05dd28face3a 0281c632b59abae9 b0853e29411968b9
e0b10de05dbac7f3 2004c45a0b701c42 8108e4be3adbd840 e6de78c0d534190c
851a4c756ff3f3ed b65959736f2fa363 261d1528e83989cf 6c89ea2582658f24
3837a174c6cb30fc 015934a76690c031
'SHA3_224' - 0.01320 sec

Resumo: e642824c3f8cf24a d09234ee7d3c766f c9a3a5168d0c94ad 73b46fdf


Process finished with exit code 0
```

2.2 Código

Apresentação do código comentado (escrito em Python 3) `keccak/keccak.py`. Abaixo segue autores e licença do código. Extraído do código em si disponível em: http://keccak.noekeon.org/.

```
# The Keccak sponge function, designed by Guido Bertoni, Joan Daemen,
# Michaël Peeters and Gilles Van Assche. For more information, feedback or
# questions, please refer to our website: http://keccak.noekeon.org/
#
# Implementation by Renaud Bauvin,
# hereby denoted as "the implementer".
#
# To the extent possible under law, the implementer has waived all copyright
# and related or neighboring rights to the source code in this file.
# http://creativecommons.org/publicdomain/zero/1.0/
```