

# A função *hash* criptográfica SHA-3

## 1 Definições

- Uma *função hash criptográfica*, ou função de resumo criptográfica (futuraamente denotada por  $h$ ), é um algoritmo matemático que mapeia uma quantidade de bytes qualquer<sup>1</sup> para uma palavra de tamanho fixo, ou seja,  $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$ ,  $n \in \mathbb{N}$ .

Para que seja resistente a diversos tipos de criptoanálise, uma função  $h : X \rightarrow Y$  deve respeitar algumas propriedades:

- Resistência à pré-imagem*: Para um resumo  $M' \in Y$ , é computacionalmente impraticável<sup>2</sup> encontrar a mensagem  $M \in X$  tal que  $h(M) = M'$ . Uma função matemática com esta propriedade é chamada de unidirecional.
- Resistência à segunda pré-imagem*: Para uma mensagem  $M_0 \in X$ , é computacionalmente impraticável encontrar uma segunda mensagem  $M_1 \in X$  tal que  $M_0 \neq M_1$  e  $h(M_0) = h(M_1)$ .
- Resistência à colisão*: Para duas mensagens  $M_0, M_1 \in X$ , é computacionalmente impraticável encontrar  $M_0 \neq M_1$  e  $h(M_0) = h(M_1)$ .

É importante notar que, embora as definições sejam extremamente parecidas, resistência à segunda pré-imagem e resistência à colisão são conceitos diferentes; um atacante não consegue escolher a primeira mensagem caso queira atacar a resistência à segunda pré-imagem; para a resistência à colisão, o atacante pode escolher livremente o par de mensagens.

- Algumas aplicações destas funções são enumeradas abaixo:
  - Podem ser utilizadas para verificar a integridade da mensagem, comparando resumos criptográficos calculados antes e depois da transmissão de mensagem e/ou arquivos.
  - Para evitar o armazenamento de senhas em texto claro, é possível armazenar apenas o resumo criptográfico de cada senha e compará-lo na autenticação do usuário.
  - Resumos criptográficos são comumente descritos como identificadores únicos seguros para um arquivo ou informação digital (por exemplo, *commits* em um sistema de controle de versão).
- O padrão SHA-3, descrito pelo documento FIPS 202 [5], é baseado em uma instância da família KECCAK de permutações matemáticas, selecionada pelo NIST (*National Institute of Standards and Technology*) e especificada neste documento.

## 2 O algoritmo SHA-3

- KECCAK é uma família de funções esponja. Este tipo de função é uma generalização do conceito da função de resumo criptográfica com saída infinita. Após a aplicação de uma função de preenchimento (*padding*) à mensagem  $M$ , a função esponja tem duas fases: a fase de absorção (*absorbing*), responsável por intercalar blocos de  $M$  com aplicações de uma função de permutação  $f$ , de modo iterativo; e a fase de compressão (*squeezing*), onde os blocos de saída, intercalados novamente pela permutação  $f$ , são concatenados para gerar uma palavra com um número de bits configurável pelo usuário. Esse processo pode ser observado na figura 1.
- A permutação  $f$  é descrita como uma sequência de operações num estado  $A$ , que é um vetor de elementos tridimensional em  $GF(2)$ .  $f$  é uma permutação iterativa, consistindo de uma sequência de rodadas. Uma rodada  $R$  consiste da composição de cinco etapas:  $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$ , como visto em 2:
  - A etapa  $\theta$  faz a soma XOR de um elemento de  $A$  e todos os elementos das colunas adjacentes indicadas.
  - A etapa  $\rho$  dispersa os elementos entre cortes transversais verticais de  $A$ .
  - A etapa  $\pi$  rearranja as posições de elementos em cortes transversais horizontais de  $A$ .
  - A etapa  $\chi$  tem como efeito fazer a soma XOR de cada bit em uma linha, de acordo com uma função não-linear de dois outros bits adjacentes.
  - A etapa  $\iota$  é utilizada para quebrar a simetria das operações acima, e sem esta etapa, todas as rodadas teriam a mesma saída. A soma XOR de alguns bits do estado  $A$  é feita com um bit específico de uma sequência gerada por um LFSR<sup>3</sup>, alimentado pelo índice da rodada atual.

<sup>1</sup>algumas funções desse tipo têm limites quanto ao tamanho da entrada, embora estes sejam extremamente grandes.

<sup>2</sup>o tempo ou recursos gastos para esta computação excedem a validade ou utilidade da informação desejada.

<sup>3</sup>linear-feedback shift register, um tipo de gerador de sequências pseudoaleatórias.

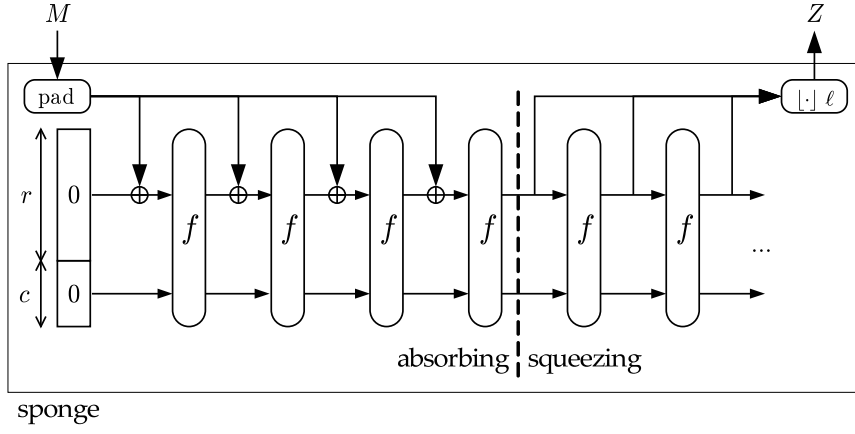


Figura 1: Uma construção esponja. Imagem retirada de [1].

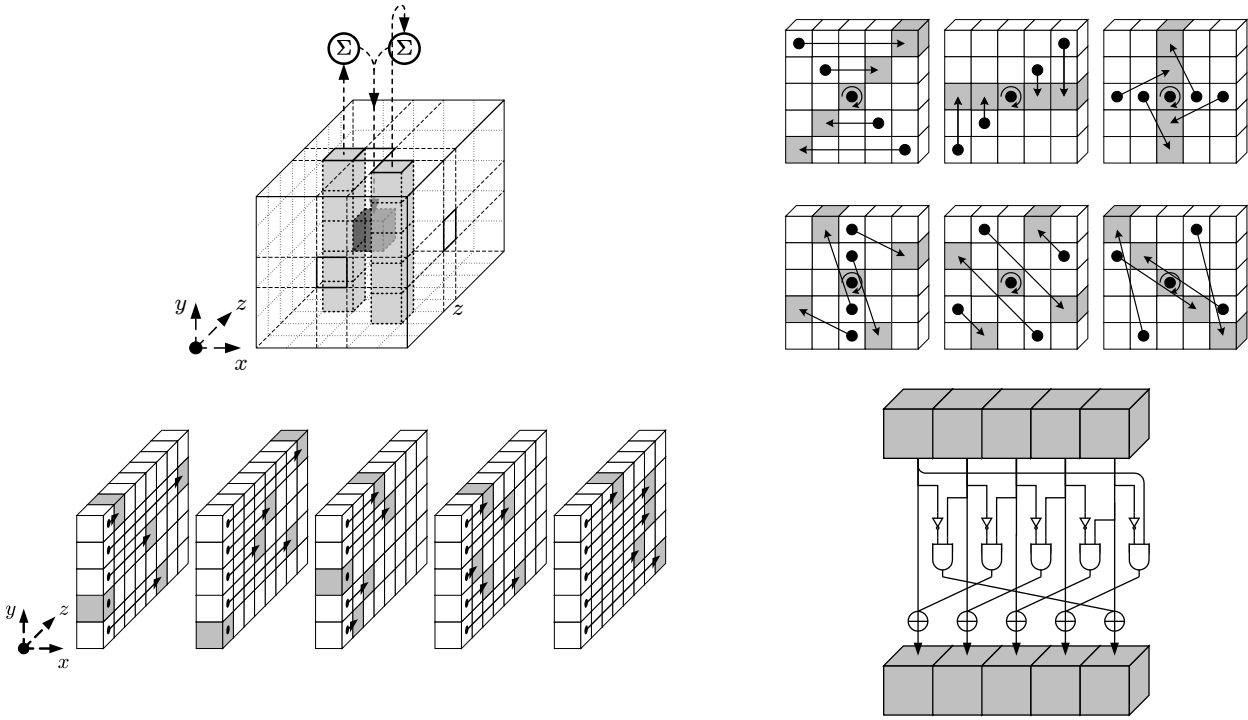


Figura 2: Da esquerda para a direita e de cima para baixo, as etapas  $\theta$ ,  $\pi$ ,  $\rho$  e  $\chi$ . Imagens retiradas de [2].

### 3 Especificações formais

- (a) O vetor de estados (*state array*) tem como função armazenar cada estado entre permutações  $f$  do KECCAK, de modo tridimensional. Este “cubo” tem dimensões  $5 \times 5 \times 2^\ell$ ,  $\ell \in [0, 6]$ , e  $2^\ell = w$ . O número total de bits neste cubo, denotado por  $b$ , é configurável pelo usuário e é geralmente chamado de *largura* do estado; esta, por sua vez, é dividida em  $r + c$  bits, chamados respectivamente de *taxa* e *capacidade*, como vistos na representação gráfica da construção esponja (1).

No vetor de estados, uma raia (*lane*) é um conjunto de  $w$  bits onde apenas a coordenada  $z$  muda; um corte transversal vertical (*slice*) é um conjunto de 25 bits onde apenas a coordenada  $z$  é fixa; uma linha é um conjunto de 5 bits onde apenas a coordenada  $x$  muda, e de modo análogo, uma coluna é um conjunto de 5 bits onde apenas a coordenada  $y$  muda.

- (b) Tome  $S$  como uma palavra de  $b$  bits que representa um estado na construção esponja; então, o vetor de estados correspondente é definido como

$$A[x][y][z] = S[w(5y + x) + z] \mid x, y \in \mathbb{Z}_5 \wedge z \in \mathbb{Z}_w$$

Por exemplo, se  $b = 200$ , então segue que  $w = 8$ , e portanto:

$$\begin{array}{cccc}
A[0, 0, 0] = S[0] & A[1, 0, 0] = S[8] & \cdots & A[4, 0, 0] = S[32] \\
\vdots & \vdots & \cdots & \vdots \\
A[0, 0, 7] = S[7] & A[1, 0, 7] = S[15] & \cdots & A[4, 0, 7] = S[39] \\
A[0, 1, 0] = S[40] & A[1, 1, 0] = S[48] & \cdots & A[4, 1, 0] = S[72] \\
\vdots & \vdots & \cdots & \vdots \\
A[0, 4, 7] = S[167] & A[1, 4, 7] = S[175] & \cdots & A[4, 4, 7] = S[199]
\end{array}$$

- (c) A construção de uma palavra  $S$  a partir de um vetor de estados  $A$  é feita pela concatenação ( $\parallel$ ) dos bits de tal modo que  $z$  seja incrementado primeiro, depois  $x$ , e por fim  $y$ ; a inversa da operação acima.

Por exemplo, se  $b = 200$  e  $w = 8$ :

$$S = A[0, 0, 0] \parallel \cdots \parallel A[0, 0, 7] \parallel A[1, 0, 0] \parallel \cdots \parallel A[4, 0, 7] \parallel A[0, 1, 0] \parallel A[0, 1, 1] \parallel \cdots \parallel A[4, 4, 7]$$

- (d) Todas as operações realizadas devem ser sobre módulo 5 nas coordenadas  $x$  e  $y$ , e módulo  $w$  na coordenada  $z$ . As adições e multiplicações entre os termos são sobre  $GF(2)$ , exceto quando notado. Índices omitidos significam que a operação é válida para todos os valores daquela porção do vetor de estados.

- $\theta : A[x][y][z] \leftarrow A[x][y][z] + \sum_{y'=0}^4 A[x-1][y'][z] + \sum_{y'=0}^4 A[x+1][y'][z-1]$

A etapa  $\theta$  foca na difusão de bits no vetor de estados  $A$ . Sem esta etapa, a permutação não proveria difusão significativa. Sua utilização leva a uma maior proteção contra criptoanálise linear e diferencial, além de ataques algébricos.

- $\rho : A[x][y][z] \leftarrow A[x][y][z - (t+1)(t+2)/2], \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$  em  $GF(5)^{2 \times 2}$

para  $0 \leq t < 24$ , ou  $t = -1$  se  $x = y = 0$ .

A etapa  $\rho$  consiste de movimentos entre bits de diferentes raia, para prover uma boa dispersão entre *slices*. Sem esta etapa, esta difusão seria muito lenta. As novas coordenadas  $x_t, y_t$  são definidas através de um processo iterativo de multiplicação de matrizes.

Tome  $M = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}$  e  $B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ :

$$\begin{array}{ll}
t = 0 \longrightarrow M^0 \cdot B = \begin{pmatrix} 1 \\ 0 \end{pmatrix} & t = 1 \longrightarrow M^1 \cdot B = \begin{pmatrix} 0 \\ 2 \end{pmatrix} \\
t = 2 \longrightarrow M^2 \cdot B = \begin{pmatrix} 2 \\ 6 \end{pmatrix} & t = 3 \longrightarrow M^3 \cdot B = \begin{pmatrix} 6 \\ 22 \end{pmatrix}
\end{array}$$

Então, pode-se verificar que existe uma relação de recorrência tal que  $(x_t, y_t) = (y_{t-1}, 2x_{t-1} + 3y_{t-1})$ . Estes valores serão dependentes apenas das dimensões do vetor de estados, então podem ser pré-computados para que o número de multiplicações entre matrizes seja reduzido.

- $\pi : A[x][y] \leftarrow A[x'][y'], \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$

A etapa  $\pi$  é uma transposição das raia, que provê difusão a longo prazo. Esta etapa mistura bits alinhados horizontalmente e verticalmente de modo que a criptoanálise diferencial seja dificultada, pois do contrário, suas trilhas poderiam ser simplificadas, de acordo com [4].

- $\chi : A[x] \leftarrow A[x] + (A[x+1] + 1) \cdot A[x+2]$

A etapa  $\chi$  é a única não-linear, trocando o valor do bit operado se seus dois próximos vizinhos à direita forem, respectivamente, 0 e 1. Sem esta etapa, a rodada  $R$  seria completamente linear. Pode ser vista como a aplicação paralela de  $5 \cdot w$  caixas-S<sup>4</sup> operando em cada linha do vetor de estados.

<sup>4</sup>*substitution box*, um componente básico de criptografia simétrica, responsável por mapear uma entrada de tamanho  $m$  para uma saída de tamanho  $n$  de modo a diluir a relação entre estes. São cuidadosamente construídas para resistir à criptoanálise linear e diferencial.

- $\iota : A \leftarrow A + RC[i_r]$ ,  
 $RC[i_r][x][y][z] = 0$ ,  
 $RC[i_r][0][0][2^j - 1] = rc[j + 7i_r] \forall 0 \leq j < \ell$ ,  
 $rc[t] = (x^t \pmod{x^8 + x^6 + x^5 + x^4 + 1}) \pmod{x}$  em  $GF(2)[x]$

A etapa  $\iota$  é a única assimétrica, e sem ela, KECCAK seria mais suscetível a ataques que exploram simetria entre rodadas. O vetor  $RC[i_r]$  guarda constantes geradas por um LFSR  $rc[t]$ , e somadas apenas à primeira linha do vetor de estados. Por conta disso, a perturbação será aumentada nas etapas  $\theta$  e  $\chi$  para todas as raias depois de apenas uma rodada.

- (e) A permutação  $KECCAK-p[b, n_r]$  consiste de  $n_r$  iterações de rodadas  $R$  sobre um estado de largura  $b$ . A permutação  $KECCAK-f$ , definida em [2], apenas é uma especialização da família acima, onde o número de rodadas deve ser correlacionado à profundidade do vetor de estados:  $n_r = 12 + 2 \cdot \ell$ . Então, a permutação  $KECCAK-p[1600, 24]$ , que define as seis funções SHA-3, é equivalente ( $\triangleq$ ) a  $KECCAK-f[1600]$ . Apenas existe uma diferença na indexação das rodadas das duas permutações.

- (f) Uma construção esponja é um modo de operação, baseado em uma permutação de tamanho fixo e uma regra de preenchimento (*padding*), que constrói uma função responsável por mapear uma entrada de tamanho qualquer para uma saída de tamanho desejável. Tal função é apropriadamente chamada de *função esponja*, e pode ser reconhecida como uma generalização de funções de resumo criptográficas, que têm saídas de tamanho fixo, e de cifras de fluxo (*stream ciphers*), restritas por entradas de tamanho fixo. A função esponja aplica iterativamente sua permutação interna aos estados intermediários, construídos por entradas ou saídas anteriores ao estado atual.

A construção aplica sua permutação  $f$  sobre estados de  $b$  bits. A entrada  $M$  é preenchida de modo que os bits extras, adicionados para tornar o tamanho dos blocos homogêneo, possam ser retirados ao final do procedimento. Então, é dividida em blocos de tamanho  $r$ , denotados  $M_r$ . Os  $b$  bits de cada estado são inicializados com zero e a construção procede à execução, em duas fases separadas.

Na fase de absorção (*absorbing*), os blocos  $M_r$  são “XORados” com os primeiros  $r$  bits do estado atual, intercalados com aplicações da permutação  $f$ . Quando todos os blocos  $M_r$  são processados, a esponja passa para a fase de compressão (*squeezing*), onde os primeiros  $r$  bits do estado são retornados como blocos de saída, também intercalados com aplicações da permutação  $f$ . O número de blocos de saída  $\ell$  é escolhido pelo usuário, e a saída  $Z$  é truncada de acordo. Os últimos  $c$  bits do estado nunca são diretamente afetados por  $M_r$ , e também nunca revelados durante a fase de compressão. Essencialmente, estão correlacionados com o nível de segurança da esponja.

- (g)  $KECCAK$  é a família de funções esponja definidas com a permutação  $KECCAK-p[b, 12 + 2\ell]$  junto a uma simples função de preenchimento  $\text{pad10*1}$ . Tal função adiciona os bits  $1 \parallel 0^{-m-2} \pmod{x} \parallel 1$  à palavra original, onde  $m$  é o resto da divisão inteira do tamanho da palavra pela largura  $x$  da esponja. O asterisco no nome da função significa que é necessário adicionar tantos “zeros” quanto necessário para preencher a palavra de maneira que esta seja igualmente divisível em blocos.
- (h) Uma função  $KECCAK[c](N, d)$  opera sobre uma palavra de bits de tamanho  $N$  e tamanho de saída  $d$ , com capacidade  $c$ . Ela pode ser definida como

$$\begin{aligned} KECCAK[r, c] &\triangleq \text{SPONGE}[KECCAK - f[r + c], \text{pad10*1}, r](N, d) \\ KECCAK[c] &\triangleq KECCAK[r = 1600 - c, c](N, d) \end{aligned}$$

e é a base para todas as quatro funções SHA-3. A função SPONGE é a representação matemática dos parâmetros explicados acima e será explorada em (i).

- i. As quatro funções *hash* SHA-3 recebem uma mensagem  $M$  como entrada e são definidas a partir da função  $KECCAK[c]$  especificada acima. Em cada caso, a capacidade é o dobro do tamanho do resumo criptográfico, e todas as mensagens são sufixadas com a palavra 01. Ou seja,

$$\begin{aligned} SHA3 - 224(M) &= KECCAK[448](M \parallel 01, 224) \\ SHA3 - 256(M) &= KECCAK[512](M \parallel 01, 256) \\ SHA3 - 384(M) &= KECCAK[768](M \parallel 01, 384) \\ SHA3 - 512(M) &= KECCAK[1024](M \parallel 01, 512) \end{aligned}$$

- ii. Duas outras funções, chamadas de *SHA-3 Extendable-Output Functions* (SHAKE, ou “Funções de saída estendida SHA-3”), são definidas concatenando 1111 como sufixo à mensagem  $M$ . Para qualquer tamanho de saída  $d$ , tem-se

$$\begin{aligned} SHAKE128(M, d) &= KECCAK[256](M \parallel 1111, d) \\ SHAKE256(M, d) &= KECCAK[512](M \parallel 1111, d) \end{aligned}$$

Os bits adicionados como sufixo às mensagens servem para diferenciar entradas da função KECCAK[*c*] provenientes de SHA-3 ou SHAKE. Esta estratégia é chamada de separação por domínios (*domain separation*).

- (i) A construção da esponja produz uma função SPONGE[*f*, pad, *r*], onde *f* é uma permutação de tamanho fixo, PAD é a função de preenchimento, que adiciona os bits necessários para que a mensagem seja corretamente dividida em blocos, e *r* é a taxa de bits da entrada que passará por dentro da esponja. O estado inicial da esponja é chamado de estado raiz, e consiste de  $0^b$  bits. Uma melhor descrição da construção pode ser encontrada em (f).
- (j) Tradicionalmente, usuários de funções *hash* esperam um nível de segurança que seja correlacionado com o tamanho da sua saída:  $2^{n/2}$  para resistência à colisão e  $2^n$  para resistência à (segunda) pré-imagem, onde *n* é o tamanho da saída. Esta característica é respeitada por todas as funções especificadas pelo NIST. A criptoanálise em instâncias de KECCAK-*f*[1600] mostra<sup>5</sup> que, ainda com 8 rodadas, o número de operações necessárias para obter alguma informação relevante é de  $2^{491}$ , e  $2^{1574}$  para as 24 rodadas propostas no documento [5].
- (k) A convenção para interpretar palavras em base hexadecimal como palavras de bits, para as entradas e saídas dos algoritmos apresentados, é diferente da usual: a ordem dos bits para cada byte completo é revertida. Ou seja, se uma palavra 0xfb23 deve ser interpretada, então:

$0xfb23 \longrightarrow 0b1111101100100011 \xrightarrow{SHA-3} 0b110111111000100$

Exemplos de entradas e saídas de rodadas e etapas intermediárias podem ser encontrados [aqui](#), onde as saídas de uma implementação podem ser comparadas a cada passo para garantir sua acurácia.

## 4 Implementação

- (a) Toda a documentação referente à implementação pode ser encontrada junto ao próprio código, localizado em [keccakf1600.py](#), no repositório fonte deste documento. Embora sua performance não seja competitiva, a legibilidade foi obtida omitindo constantes “mágicas”, escolhendo gerá-las ao longo da execução do programa.
- (b) A função Keccak contém três partes principais: a absorção, nas linhas 157–164; o preenchimento, em 166–170; e a compressão, em 172–178. A função keccak\_f\_1600, por sua vez, abriga as etapas  $\theta$  em 103–105,  $\rho$  e  $\pi$  mescladas em 107–111,  $\chi$  em 113–116 e  $\iota$  em 118–121. É possível substituir facilmente a etapa assimétrica pelo vetor de constantes RC, e embora trabalhoso, adaptar o código para utilizar o vetor de constantes *r* na etapa  $\rho$ , assim trazendo a implementação mais próxima do pseudocódigo apresentado em [3].
- (c) As funções podem ser chamadas diretamente a partir de um interpretador após importar o código-fonte; note que a função hex() é exclusiva do Python 3.5+; assim, usuários de outras versões podem utilizar o segundo método para obter o resumo em forma hexadecimal convencional. A entrada deve ser um vetor de bytes, facilmente criado em Python adicionado o caractere b à frente do conteúdo da palavra.

```
$ python
>>> from keccakf1600 import SHA3, SHAKE
>>> SHA3(224, b'').hex()
'6b4e03423667dbb73b6e15454f0eb1abd4597f9a1b078e3f5b5a6bc7'
>>> SHAKE(128, b'', 28).hex()
'7f9c2ba4e88f827d616045507605853ed73b8093f6efbc88eb1a6eac'

$ python2
>>> from binascii import hexlify
>>> from keccakf1600 import SHA3
>>> hexlify(SHA3(224, b''))
'6b4e03423667dbb73b6e15454f0eb1abd4597f9a1b078e3f5b5a6bc7'
```

A saída do programa, gerada por uma implementação à parte e ainda não publicada, pode ser encontrada junto ao arquivo shodan.out, mostrando o vetor de estados impresso a cada etapa concluída, em todas as rodadas.

```
Mensagem original
"How can you challenge a perfect, immortal machine?"
...
Resultado final
42af10210ea47baafdb41d6be3203c99969a10c00d1bec655553d07c
```

<sup>5</sup>como visto em <http://keccak.noekeon.org/Keccak-slides-at-NIST-Feb2013.pdf>

## 5 Segurança

- (a) O principal argumento fundamentando a escolha do KECCAK pelo NIST foi sua grande diferença em relação ao atual padrão, SHA-2. Enquanto o SHA-3 é uma função esponja (extremamente flexível e customizável) que abriga uma permutação com etapas focadas na proteção contra diversos tipos de criptoanálise, o SHA-2 é uma versão mais robusta de algoritmos já conhecidos e analisados exaustivamente, que operam repetidamente com álgebra booleana, rotações de bits e adições módulo  $2^{32}$ . Embora essa característica não afete sua segurança atualmente, é possível que ataques a esta família de algoritmos (principalmente ao padrão anterior, SHA-1) possam contribuir para que o SHA-2 seja ameaçado no futuro.
- (b) Não é possível afirmar um período de tempo com absoluta certeza, porém o [material publicado](#) sugere que as permutações KECCAK são extremamente resistentes a ataques convencionais, e são geralmente analisadas com métodos de vanguarda (*zero-sum*, *rotational cryptanalysis*) e/ou um número de rodadas reduzido. Resultados práticos só foram obtidos com um número de rodadas extremamente pequeno. Assim, é possível conjecturar que KECCAK manterá sua resistência por pelo menos um decênio, antes que o poder computacional médio evolua suficientemente para que buscas exaustivas possam ser realizadas.

## Referências

- [1] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Cryptographic sponge functions, January 2011. <http://sponge.noekeon.org/>.
- [2] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The KECCAK reference, January 2011. <http://keccak.noekeon.org/>.
- [3] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. KECCAK implementation overview, May 2012. <http://keccak.noekeon.org/>.
- [4] J. Daemen and G. Van Assche. Differential propagation analysis of KECCAK. In *Fast Software Encryption 2012*, 2012.
- [5] Morris J. Dworkin. SHA-3 standard: Permutation-based hash and extendable-output functions. Technical report, National Institute of Standards and Technology (NIST), July 2015.