Função de Hash Criptográfica SHA-3

Quênio César Machado dos Santos (14100868)

INE5429 - Segurança em Computadores

Florianópolis, 21/06/2016

Sumário

1 Função Hash Criptográfica

Uma função *hash* é uma função que aceita um bloco de dados de tamanho variável como entrada e produz um valor de tamanho fixo como saída, chamado de valor *hash*. Esta função tem a forma:

$$h = H(M)$$

Onde:

- H é a função *hash* que gerou o valor h.
- h é o valor *hash* de tamanho fixo gerado pela função *hash*.
- M é o valor de entrada de tamanho variável.

Espera-se que uma função *hash* produza valores h que são uniformemente distribuídos no contra-domínio e que são aparentemente aleatórios, ou seja, a mudança de apenas um *bit* em M causará uma mudança do valor h. Por esta característica, as funções *hash* são muito utilizadas para verificar se um determinado bloco de dados foi indevidamente alterado.

As funções *hash* apropriadas para o uso em segurança de computadores são chamadas de "função *hash* criptográfica". Este tipo de função *hash* é implementada por um algoritmo que torna inviável computacionalmente encontrar:

- um valor M dado um determinado valor h:

$$M \mid H(M) = h$$

- dois valores M_1 e M_2 que resultem no mesmo valor h:

$$(M_1, M_2) \mid H(M_1) = H(M_2)$$

Os principais casos de uso de funções *hash* criptográficas são:

- *Autenticação de Mensagens*: é um serviço de segurança onde é possível verificar que uma mensagem não foi alterada durante sua transmissão e que é proveniente do devido remetente.
- *Assinatura Digital*: é um serviço de segurança que permite a uma entidade assinar digitalmente um documento ou mensagem.
- *Arquivo de Senhas de Uma Via*: é uma forma de armazenar senhas usando o valor *hash* da senha, permitindo sua posterior verificação sem a necessidade de armazenar a senha em claro, cifrá-la ou decifrá-la.
- *Detecção de Perpetração ou Infeção de Sistemas*: é um serviço de segurança em que é possível determinar se arquivos de um sistema foram alterados por terceiros sem a autorização dos usuários do sistema.

1.1 Propriedades

Como observado na seção anterior, uma função *hash* criptográfica precisa ter certas propriedades para permitir seu uso em segurança de computadores. Nas seções a seguir estão destacadas algumas dessas propriedades.

Antes, defini-se dois termos usados a seguir:

- *Pré-Imagem*: um valor M do domínio de uma função *hash* dada pela fórmula $h = H(M)$ é denominado de "pré-imagem" do valor h.
- *Colisão*: para cada valor h de tamanho n *bits* existe necessariamente mais de uma pré-imagem correspondente de tamanho m *bits* se $m > n$, ou seja, existe uma "colisão".

O número de pré-imagens de m *bits* para cada valor h de n *bits* é calculado pela formula: $2^{\frac{m}{n}}$. Se permitimos um tamanho em *bits* arbitrariamente longo para as pré-imagens, isto aumentará ainda mais a probabilidade de colisão durante o uso de uma função *hash*. Entretanto, os riscos de segurança são minimizados se a função de *hash* criptográfica oferecer as propriedades descritas nas próximas seções.

1.1.1 Resistente a Pré-Imagem

Uma função *hash* criptográfica é resistente a pré-imagem quando esta é uma função de uma via. Ou seja, embora seja computacionalmente fácil gerar um valor h a partir de uma pré-imagem M usando a função de *hash*, é computacionalmente inviável gerar uma pré-imagem a partir do valor h.

Se uma função *hash* não for resistente à pré-imagem, é possível atacar uma mensagem autenticada M para descobrir o valor secreta S usada na mensagem, permitindo assim ao perpetrante enviar uma outra mensagem M_2 ao destinatário no lugar do remetente sem que o destinatário perceba a violação da comunicação. O ataque ocorre da seguinte forma:

- O perpetrante tem conhecimento do algoritmo de *hash* $h = H(M)$ usado na comunicação entre as partes.

- Ao escutar a comunicação, o perpetrante descobre qual é a mensagem M e o valor de *hash* h.
- Visto que a inversão da função de *hash* é computacionalmente fácil, o perpetrante calcula $H^{-1}(h)$.
- Como $H^{-1}(h) = S \parallel M$, o perpetrante descobre S.

Desta forma, o perpetrante pode utilizar a chave secreta S no envio de uma mensagem M_2 para o destinatário sem que este perceba a violação.

1.1.2 Resistente a Segunda Pré-Imagem

Uma função *hash* criptográfica é resistente a segunda pré-imagem quando esta função torna inviável computacionalmente encontrar uma pré-imagem alternativa que gera o mesmo valor h da primeira pré-imagem.

Se uma função de *hash* não for resistente a segunda pré-imagem, um perpetrante conseguirá substituir uma mensagem que utiliza um determinado valor de *hash*, mesmo que a função de *hash* seja de uma via, ou seja, resistente a pré-imagem.

1.1.3 Resistente a Colisão

Uma função *hash* criptográfica é resistente a colisão quando esta tornar inviável computacionalmente encontrar duas pré-imagens quaisquer que possuam o mesmo valor de *hash*. Neste caso, diferentemente da resistência a segunda pré-imagem, não é dado uma pré-imagem inicial para a qual precisa se achar uma segunda pré-imagem, mas é suficiente encontrar duas pré-imagens quaisquer tal que $H(M_1) = H(M_2)$.

Quando uma função *hash* é resistente a colisão, está é consequente resistente a segunda pré-imagem. Porém, nem sem sempre uma função resistente a segunda pré-imagem será resistente a colisão. Por isto, diz-se que uma função *hash* resistente a colisão é uma função de *hash* forte.

Se uma função *hash* não for resistente a colisão, então é possível para uma parte forjar a assinatura de outra parte. Por exemplo, se Alice deseja que Bob assine um documento dizendo que deve 100 reais a ela, caso Alice saiba que um documento contendo o valor de 1000 reais contém o mesmo valor de *hash* que o documento original, Alice pode fazer com que Bob seja responsável por uma dívida maior que a original, pois a assinatura valerá para ambos os documentos.

1.1.4 Uso das Propriedades de Funções *Hash*

Abaixo, temos uma tabela que mostra quais propriedades das funções *hash* são necessárias para alguma das aplicações de segurança de computadores:

Aplicação	Resistente a Pre-Imagem	Resistente a Segunda Pre-Imagem	Resistente a Colisão
Autenticação de Mensagens	X	X	X
Assinatura Digital	X	X	X
Infecção de Sistemas		X	

Arquivo de Senhas de Uma Via	X		

No caso da infecção de sistemas, não há problema em usar uma função de hash com fácil inversão, pois não é necessário embutir um valor secreto na geração do valor de *hash* de um arquivo. Já, num arquivo de *hash* de senhas, a inversão permitiria descobrir a senha a partir do valor de *hash*.

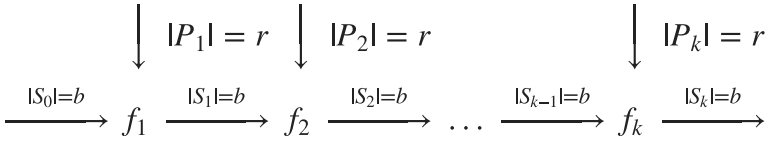
Se a função de *hash*, porém, permitir o descobrimento de uma segunda pré-imagem, seria possível infectar um arquivo de um sistema sem detecção, pois seu valor de *hash* não mudaria. Isto não seria um problema para um arquivo de *hash* de senhas, pois o perpetrante não possui a senha, que é a primeira pré-imagem e, portanto, não teria condições de descobrir a segunda pré-imagem.

2 SHA-3

SHA-3 é uma função *hash* criptográfica publicada pelo NIST em agosto de 2015 para substituir o SHA-2 como o padrão para os sistemas de informação dos departamentos e das agências do governo americano. SHA-3 provavelmente será adotado por sistemas operacionais e também por organizações privadas e públicas de todo o mundo, assim como foi o caso do SHA-2 e SHA-1.

2.1 A Estrutura do SHA-3

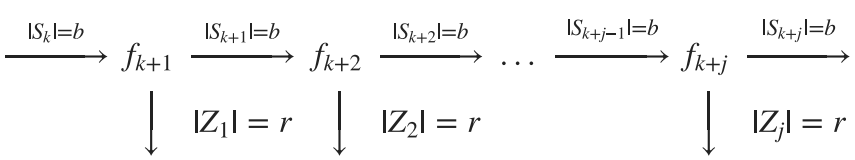
A estrutura de entrada do SHA-3 segue a estrutura genérica de outras funções *hash* iterativas, onde o resultado de uma função de compressão f é iterativamente aplicado sobre a mesma função, juntamente com o próximo bloco P_i da mensagem de entrada, como ilustrado no diagrama abaixo:



No esquema ilustrado acima, uma mensagem de entrada de n *bits* é dividida em k blocos de tamanho r *bits*: P_1, P_2, ..., P_n. O último bloco é sempre preenchido para que tenha r *bits*. Cada bloco é processado com a saída S_{i-1} da execução anterior da função f. O símbolo f_i, com $1 \leq i \leq k$, representa a execução da função f na iteração i gerando o resultado S_i de b *bits*. Após todos os blocos P_i serem processados, produz-se o valor S_k.

Apesar de seguir o esquema genérico, descrito acima, na sua estrutura de entrada, o SHA-3 possui uma característica peculiar, descrita abaixo, na sua estrutura de saída. Combinando ambas as estruturas, o SHA-3 permite um número variável de *bits* tanto na entrada como na saída. Este fato o torna mais flexível e aplicável não somente como função *hash*, mas também como um gerador de números pseudo-aleatórios; além de permitir outras aplicações. Devido a esta característica, os criadores do SHA-3 chamam sua estrutura de função *esponja*.

Observe a estrutura de saída do SHA-3 no diagrama abaixo:



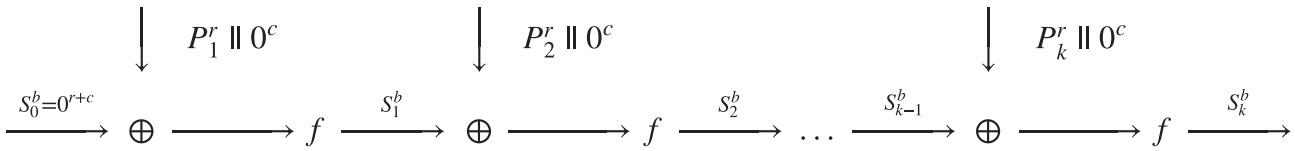
No esquema ilustrado acima, o valor S_k proveniente da estrutura de entrada serve como valor inicial da estrutura de saída. Após processar os k blocos da mensagem de entrada, e usando a mesma função de compressão f, a função esponja gera uma sequência de j blocos: Z_1, Z_2, ..., Z_j. O número de blocos de saída j é determinado pelo número de *bits* de saída desejado. Se l *bits* são necessários na saída, então:

$$(j-1) \times r < l \leq j \times r$$

Esta flexibilidade no número de l *bits* de saída é o que dá o nome *esponja* à função do SHA-3. De acordo com esta analogia, quando a estrutura de entrada está consumindo os n *bits* da mensagem de entrada, diz-se que a função *esponja* está "absorvendo" os *bits* de entrada. E quando a estrutura de saída está gerando os l *bits* de saída, diz-se que a função *esponja* está sendo "espremida" para liberar os *bits* de saída.

2.2 A Fase de Absorção

A primeira fase da função esponja se chama de *absorção* e refere-se ao processamento dos blocos da mensagem de entrada. Veja a fase de absorção na ilustração abaixo:



Como ilustrado na figura acima, existe uma variável de estado s que é utilizada nesta fase. Estava variável serve como entrada e saída de cada iteração que aplica a função de compressão f. Inicialmente, s contém 0 em todos os seus *bits*. Seu valor vai se modificando em cada iteração.

O tamanho de s é de b *bits*, onde:

$$b = r + c$$

Como visto na seção anterior, r é o tamanho de cada bloco P_i da mensagem de entrada. Também é chamado de *bitrate*, ou vazão de *bits*, pois representa o número de *bits* consumidos em cada iteração da função esponja.

O número de *bits* c é chamado de *capacidade* e representa o nível de segurança atingido pela função esponja. Dado o valor padrão de $b = r + c = 1600$ no SHA-3, quanto maior o numero c, maior a segurança da função, porém menor o *bitrate*.

Em cada iteração, a fase de absorção ocorre da seguinte forma:

- O próximo bloco P_i da mensagem de entrada é preenchido com zeros ($P_i^r \| 0^c$) para aumentar seu tamanho de r para b *bits*.
- Ao resultado do passo anterior, uma operação *XOR* é aplicada, tendo como segundo operando o valor de s_{i-1} proveniente da iteração anterior, ou zeros se for a primeira iteração ($S_0^b = 0^{r+c}$).
- O resultado da operação *XOR* serve então de entrada para a função de compressão f. O resultado desta função é o novo valor S_i da variável s, que é usada como entrada da próxima iteração, juntamente com o próximo bloco P_{i+1} da mensagem de entrada.

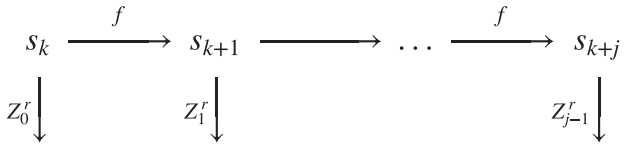
Se o tamanho desejado da saída da função esponja é menor que o tamanho de s - ou seja, se $l \leq b$ - então os primeiros l *bits* de s_k - retornado pela última iteração - é o resultado da função esponja. Caso deseja-se $l > b$, então a fase de "espremer a esponja" inicia-se, como descrito na

próxima seção.

2.3 "Espremendo a Esponja"

Na faze de absorção da função esponja, descrita na seção anterior, foram consumidos todos os blocos da mensagem de entrada, resultando num valor final de s de b *bits*. Se o tamanho desejado de saída (l) da função esponja for $l > b$, diz-se que é preciso *espremer a esponja* para obter uma saída com o número de *bits* desejado.

A fase de *espremer a esponja* é ilustrada abaixo:



Observando a ilustração acima, pode-se descrever esta fase da seguinte forma:

- Primeiramente, os primeiros r *bits* de s_k são colocados num bloco Z_0.
- Então o valor de s_k é aplicado na função f para se obter novo valor de s_{k+1}.
- Os primeiro r *bits* do novo valor de s_{k+1} são colocados num bloco Z_1.
- Este processo se repete até que se tenha j blocos (Z_0, Z_1, ..., Z_{j-1}) tal que $(j-1) \times r < l \leq j \times r$.

Ao final deste processo, a saída da função esponja serão os primeiros l *bits* dos blocos concatenados $Z_0 \,||\, Z_1 \,||\, \ldots \,||\, Z_{j-1}$.

2.4 Função de Compressão Keccak

Nas seções anteriores, foi dado uma visão geral da estrutura da função esponja utilizada por SHA-3. Nesta seção, o foco será a função de compressão utilizada em cada iteração do SHA-3, que é chamada de *Keccak* pelos seus autores.

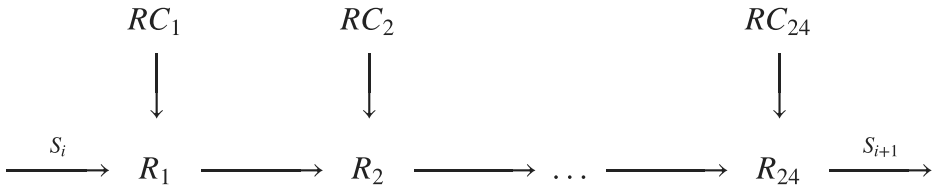
Como visto anteriormente, a função Keccak (f) tem como entrada um valor s de b *bits*, onde $b = r + c = 1600$. No processamento interno da função f, o valor s é organizado numa matriz 5×5 com valores de 64 *bits* em cada uma de suas células. Esta matriz pode ser linearizada num vector de *bits*, correspondendo ao valor s, usando a seguinte fórmula:

$$s[64(5y + x) + z] = M[x, y, z]$$

Onde:

- M é a matriz 5×5 com valores de 64 *bits*.
- x é o índice de coluna na matriz, que vai de 0 a 4.
- y é o índice de linha na matriz, também de 0 a 4.
- z é o índice de *bit* de uma célula na matriz, que vai de 0 a 63.

Uma vez criada a matriz, a função f vai executar 24 rodadas de processamento:



Observe acima que todas as rodadas são idênticas, exceto pela constante RC_i diferente em cada rodada. Cada uma das rodadas, consiste de 5 passos. Cada passo executa uma operação de permutação ou substituição sobre a matriz.

A aplicação dos cinco passos de cada rodada é expressa pela composição das seguintes funções:

$$R_i = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

Onde cada passo tem sua fórmula na tabela seguinte:

Função	Fórmula
Theta	$\theta : M[x,y,z] \leftarrow M[x,y,z] \oplus \sum_{y'=0}^{4} M[x-1,y',z] \oplus \sum_{y'=0}^{4} M[x+1,y',z-1]$
Rho	$\rho : M[x,y,z] \leftarrow \begin{cases} M[x,y,z], \text{ se } x=y=0 \\ M[x,y,z-\frac{(t+1)(t+2)}{2}], \text{ onde } 0 \leq t < 24 \text{ e } \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \\ \text{em } GF(5)^{2\times 2} \end{cases}$
Pi	$\pi : M[x,y] \leftarrow M[x',y'], \text{ onde } \begin{pmatrix} x \\ y \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}$
Chi	$\chi : M[x,y,z] \leftarrow M[x,y,z] \oplus (\neg M[x+1,y,z] \wedge M[x+2,y,z])$
Iota	$\iota : M[x,y,z] \leftarrow M[x,y,z] \oplus RC(i)$, onde RC é uma tabela com um valor para cada rodada

Baseado na tabela acima, aqui estão algumas características de cada passo:

- *Theta* (θ) é uma função de substituição que utiliza *bits* das colunas anteriores e posteriores, além dos *bits* da célula sendo substituída. Cada *bit* substituído depende de outros 11 *bits*, o que provê uma difusão de alto grau.
- *Rho* (ρ) é uma função de permutação dos *bits* dentro de cada célula. Sem esta função, a difusão entre as células, ocorreria de forma muito lenta.
- *Pi* (π) é também uma função de permutação, mas entre células. A rotação não se dá nos *bits* de uma célula, mas entre as células da matriz.
- *Chi* (χ) é uma função de substituição baseada no valor do *bit* corrente e dos *bits* em posições correspondentes das duas próximas células. Sem esta função, SHA-3 seria completamente linear.
- *Iota* (ι) é uma função de substituição baseada numa tabela - chamada *RC*, ou seja, constantes de rodada - que usará um valor constante e diferente para cada rodada do SHA-3.

2.5 Parâmetros do SHA-3

O SHA-3 define um algoritmo padrão para uso com parâmetros diferentes, dependendo do

nível de segurança e tamanho de *bits* desejados na saída. A tabela abaixo enumera os parâmetros normalmente utilizados com o SHA-3:

Tamanho do Valor de *Hash* (l)	Tamanho do Bloco (r)	Capacidade (c)	Resistência a Colisão	Resistência à Segunda Pré-Imagem
224	1152	448	2^{112}	2^{224}
256	1088	512	2^{128}	2^{256}
384	832	768	2^{192}	2^{384}
512	576	1024	2^{256}	2^{512}

Como visto nas seções anteriores, quanto maior o tamanho do bloco (*r* ou *bitrate*), maior a vazão de *bits*, porém menor a segurança do SHA-3. Isto é evidente nos valores de resistência a colisão e à segunda pré-imagem, que mostram que quanto menor é o *bitrate*, maior é a resistência.

Observe também que, para os tamanhos *l* de valor de *hash* da tabela acima, não há necessidade de se usar a fase de *espremer a esponja* do algoritmo do SHA-3, pois *l* é sempre menor que *r* nestes casos.

3 Referências

1. *William Stalling*, Cryptography and Network Security, Sixth International Edition, 2014.
2. FIPS PUB 202.
3. Wikipedia.