MD6 e a competição para escolha do SHA-3

Valdson Silva Cleto

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Ciência da Computação

Orientador: Prof. Dr. Routo Terada

São Paulo, fevereiro de 2011

Resumo

Neste trabalho abordamos a função de hash MD6 no contexto da competição do instituto norte-americano NIST para escolha de uma nova função de hash padrão, que receberá o título de SHA-3. O MD6 foi um dos candidatos na primeira fase da competição, mas se retirou da competição por não atender ao critério de desempenho em termos de velocidade exigido pelo NIST dos candidatos a SHA-3. Este desempenho relativamente baixo deve-se principalmente à tentativa no projeto do MD6 de torná-lo demonstravelmente resistente a ataques diferenciais. A eficiência dos ataques diferenciais contra as funções de hash existentes foi um dos principais motivos para a criação da competição para escolha do SHA-3. Analisamos uma possível modificação de parâmetros da função de compressão do MD6 na tentativa de se conseguir um melhor desempenho sem enfraquecer a resistência da função aos ataques diferenciais.

Palavras-chave: hash, MD6, SHA-3.

Abstract

This work is about MD6 hash function in the context of the NIST competition to choose a new hash function standard that will be called SHA-3. The MD6 was a candidate in the first round of the competition, but MD6's author has suggested that MD6 was not ready for the next SHA-3 round because NIST had stated that to be competitive a SHA-3 candidate really needs to be at least as fast as the existing SHA-2 algorithms, and MD6 is not. This low performance is mainly due to the try on the MD6 project of making it provable resistant to differential attacks. The efficiency of the differential attacks against the old hash functions was one of the mainly reasons for the creation of the competition to choose the SHA-3. We analyze a possible parameter modification in the MD6 compression function, trying to get a better performance without making the function weaker to the differential attacks.
Keywords: hash, MD6, SHA-3.

Sumário

Lista de Tabelas

Capítulo 1

Introdução

Uma função de hash recebe uma informação binária, chamada de mensagem e produz uma representação condensada, chamada de resumo da mensagem. Uma função de hash criptográfica é uma função de hash que é projetada para garantir certas propriedades de segurança.

Uma função de hash criptográfica pode garantir a certeza da integridade de dados. Se algum dado da mensagem é alterado, então o resumo da mensagem não será mais válido. Mesmo se os dados forem armazenados em um lugar inseguro, sua integridade pode ser verificada a qualquer momento recalculando-se o resumo e verificando-se se ele não mudou. Funções de hash com chave são usadas na geração de um código de autenticação de mensagem, ou MAC (Message Authentication Code).

O instituto norte-americano NIST (National Institute of Standards and Technology) estabeleceu como padrão em agosto de 2002, cinco funções de hash criptográficas: SHA-1, SHA-224, SHA-256, SHA-384 e SHA-512, sendo estas quatro últimas conhecidas como a família SHA-2.

Em fevereiro de 2005, Wang, Yin e Yu apresentaram um trabalho demonstrando métodos relativamente eficientes para encontrar colisões na função de hash SHA-1. Como a família SHA-2 também pode vir a ser vulnerável, o NIST decidiu desenvolver um novo algoritmo de hash através de uma competição pública. O algoritmo vencedor da competição receberá o título de SHA-3, e a princípio deverá ser anunciado em 2012.

Em outubro de 2008 foram inscritos os algoritmos concorrentes. Um desses algoritmos é o MD6, projetado por Ron Rivest. O algoritmo de hash MD6 foi projetado para ser mais robusto em relação a fragilidades conhecidas até então, e para já estar preparado para a provável evolução do poder computacional nos próximos anos, que se dará não pelo aumento da frequência de clock dos processadores, mas pela multiplicação do número de unidades de processamento em um sistema, o que favorece um algoritmo que possiblite a exploração da capaciade de realizar cálculos de forma paralela, independente.

Porém, em julho de 2009 Ron Rivest emitiu um comunicado informando que naquele momento o MD6 não atenderia os requisitos de velocidade necessários para um candidato a SHA-3 e portanto não recomendava que o MD6 passasse para a segunda fase da competição. Assim, o MD6 não apareceu na lista dos candidatos que passaram à segunda fase.

O NIST estabeleceu que, para ser competitivo, um candidato a SHA-3 precisaria ser no mínimo tão rápido quanto o SHA-2 em plataformas de referência. Embora o MD6 seja muito rápido em sistemas multiprocessados, nas plataformas de referência ele é bem mais lento que o SHA-2.

Ron Rivest alertou os organizadores da competição que o algoritmo de SHA-3 que viesse a ser escolhido deveria ser demonstravelmente resistente a ataques diferenciais, visto que foi o poder

surpreendente dos ataques diferenciais que estimulou a competição para escolha do SHA-3.

O que torna o MD6 significativamente mais lento que os outros competidores nas plataformas de referência é o número de rodadas da função de compressão que teve que ser adotado justamente para torná-lo demonstravelmente resistente a ataques diferenciais.

Com a ajuda de um programa computacional tentamos encontrar uma diferente configuração de parâmetros da função de compressão do MD6 tal que a demonstração da resistência aos ataques diferenciais exija um número menor de rodadas. Tentamos também descobrir se é possível realizar a demonstração da resistência aos ataques diferenciais adotando-se um número menor de rodadas.

Capítulo 2

Função de hash

2.1 Definições básicas

Uma função de hash com chave pode ser considerada como uma família de funções de hash, pois cada chave escolhida define uma função de hash diferente. Desta forma, uma função de hash sem chave pode ser definida como uma família de funções de hash com chave, cujo conjunto de chaves possíveis contém apenas uma chave. Uma família de funções de hash com chave pode ser definida como uma 4-tupla $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, onde:

- $\mathcal{X}$ é um conjunto de mensagens possíveis, finito ou infinito, $|\mathcal{X}| = N$;
- $\mathcal{Y}$ é um conjunto finito de resumos de mensagem possíveis $|\mathcal{Y}| = M$;
- $\mathcal{K}$ é um conjunto finito de chaves possíveis.
- Cada elemento K pertencente a $\mathcal{K}$ define uma função de hash h_k pertencente a $\mathcal{H}$, sendo h_k uma função de $\mathcal{X}$ em $\mathcal{Y}$.

Uma função de hash definida para um conjunto $\mathcal{X}$ finito de mensagens possíveis, é chamada de *função de compressão.*

Para uma chave K pertencente a $\mathcal{K}$, um par (x, y) pertencente a $\mathcal{X} \times \mathcal{Y}$ é *válido* se $h_k(x) = y$.

2.2 Segurança das funções de hash

Em muitas aplicações criptográficas das funções de hash, é desejável que a única maneira de se obter um par (x, y) válido, seja escolhendo-se x primeiro e então calculando-se y pela aplicação da função de hash a x. Em determinados protocolos outras propriedades são necessárias. A seguir mostraremos as principais propriedades que uma função de hash criptográfica deve ter para que ela seja considerada segura.

2.2.1 Resistência à primeira pré-imagem

O problema da primeira pré-imagem consiste em, dado um resumo de mensagem y, encontrar uma mensagem x tal que $h(x) = y$. Ou seja, obter um par (x, y) válido pelo caminho inverso, conhecendo-se inicialmente y. Se este problema não pode ser resolvido de maneira eficiente, então a função de hash é considerada resistente à primeira pré-imagem.

2.2.2 Resistência à segunda pré-imagem

O problema da segunda pré-imagem consiste em, dada uma mensagem x, encontrar uma mensagem x', sendo $x \neq x'$, tal que $h(x) = h(x')$. Se este problema não pode ser resolvido de maneira eficiente, então a função de hash é considerada resistente à segunda pré-imagem.

2.2.3 Resistência à colisão

O problema da colisão consiste em encontrar um par de mensagens x, x', sendo $x \neq x'$, tal que $h(x) = h(x')$. Se este problema não pode ser resolvido de maneira eficiente, então a função de hash é considerada resistente à colisão.

2.2.4 Algoritmos no modelo de oráculo aleatório

Como já foi dito, para uma função de hash ser ?ideal?, a única maneira de se obter o resumo de uma mensagem x é realmente aplicando-se a função de hash à mensagem x, e isto deve ser verdadeiro mesmo que se conheçam previamente os resumos de muitas outras mensagens.

Bellare e Rogaway apresentaram um modelo matemático de uma função de hash ?ideal?. Neste modelo, uma função de hash h é escolhida aleatoriamente dentro do conjunto de todas as funções de $\mathcal{X}$ em $\mathcal{Y}$, e não é dada nenhuma fórmula ou algoritmo para o cálculo dos valores da função h. A única maneira de se obter o valor de $h(x)$ é perguntando a um oráculo. Isto seria equivalente a obter o valor de $h(x)$ consultando-se um gigantesco livro de números aleatórios no qual para cada x possível existe um valor de $h(x)$ completamente aleatório.

Um oráculo aleatório verdadeiro não existe no mundo real, mas uma função de hash bem projetada deveria se comportar como se fosse um oráculo aleatório. Portanto, é útil o estudo da resistência de um algoritmo no modelo de oráculo aleatório em relação aos problemas da primeira pré-imagem, segunda pré-imagem e colisão.

Se analisarmos a probabilidade de sucesso no caso médio de algoritmos triviais que tentem resolver cada um dos problemas no modelo de oráculo aleatório calculando $h(x)$ para q valores de x , concluiremos:

- Para um algoritmo que tente resolver o problema da primeira pré-imagem a probabilidade de sucesso no caso médio será: $1 - (1 - 1/M)^q$. Se considerarmos que q é pequeno se comparado a M, esta probabilidade de sucesso será aproximadamente q/M.

- A análise para um algoritmo que tente resolver o problema da segunda pré-imagem será parecida com a análise anterior, a única diferença é que neste algoritmo uma das q aplicações de h será utilizada para o cálculo de $h(x)$ do valor x conhecido inicialmente. A probabilidade de sucesso deste algoritmo será: $1 - (1 - 1/M)^{q-1}$.

- Para um algoritmo que tente resolver o problema da colisão a probabilidade de sucesso será: $1 - ((M-1)/M) \cdot ((M-2)/M) \ldots ((M-q+1)/M)$.

A análise das probabilidades de sucesso dos algoritmos no modelo de oráculo aleatório nos mostra que resolver o problema da colisão é mais fácil do que resolver os problemas da primeira pré-imagem e da segunda pré-imagem.

A análise do algoritmo que tenta resolver o problema da colisão está relacionada com a famosa questão do paradoxo do aniversário. O paradoxo do aniversário diz que em um grupo de 23 pessoas escolhidas aleatoriamente, a probabilidade de que duas delas façam aniversário no mesmo dia é maior que 50%. Obviamente isto não é na verdade um paradoxo, mas algo que vai contra nossa intuição natural: a maioria das pessoas consideraria uma grande coincidência encontrar duas pessoas que fazem aniversário no mesmo dia dentro de uma sala com 23 pessoas, quando na verdade isto é mais provável do que não encontrar esta ?coincidência?. No caso do paradoxo do aniversário, teríamos $M = 365$ (ou 366 se considerarmos o dia 29 de fevereiro), e para $q = 23$ podemos calcular que a probabilidade de colisão (duas pessoas fazerem aniversário no mesmo dia) será maior do que $1/2$. Fazendo uma analogia do paradoxo do aniversário com o problema da colisão, $\mathcal{X}$ seria o conjunto de todos os seres humanos, $\mathcal{Y}$ seria o conjunto de todas as datas de aniversário e $h(x)$ seria a data de aniversário de cada pessoa x.

Essa analogia com o paradoxo do aniversário ajuda a dar uma idéia de como a dificuldade para se encontrar uma colisão realizando q aplicações da função h é bem menor do que imaginaríamos ao olhar para o valor de M de um determinado conjunto $\mathcal{Y}$ ($|\mathcal{Y}| = M$), ou seja, para que uma função de hash seja resistente à colisão, o número de bits do resumo da mensagem deve ser bem maior do que o número que usaríamos baseados apenas em nossa intuição. Por exemplo, um resumo de mensagem de 40 bits seria muito inseguro, haveria uma probabilidade de 50% de se encontrar uma colisão calculando-se o resumo de um pouco mais do que 2^{20} (aproximadamente 1 milhão) mensagens escolhidas aleatoriamente, o que não seria nada difícil. Por isso que o tamanho mínimo aceitável de um resumo de mensagem é 128 bits (o que exigiria o cálculo de mais do que 2^{64} resumos de mensagens) e normalmente recomenda-se resumos de mensagem de 160 bits ou mais.

2.3 Funções de hash iteradas

Consideramos até agora as funções de hash chamadas de funções de compressão, que são aquelas definidas para um conjunto $\mathcal{X}$ finito de mensagens possíveis. Uma função de compressão pode ser estendida para uma função com um domínio infinito através de um método que descreve como aplicar repetidamente esta função de compressão a pedaços de tamanho fixo de uma mensagem pertencente a um conjunto infinito de mensagens possíveis. A função de hash estendida por este método é chamada função de hash iterada e o método de extensão de domínio utilizado é chamado de modo de operação da função.

Se $\mathcal{X}$ é finito então toda mensagem X pertencente a $\mathcal{X}$ pode ser representada por uma sequência de bits de comprimento fixo, e neste caso a função de compressão pode ser chamada também de uma função de comprimento da entrada fixo. Se $\mathcal{X}$ é infinito, então temos uma função de comprimento da entrada variável. Um modo de operação é conhecido como um extensor de domínio, pois tem a função de estender o domínio finito de uma função de compressão de comprimento da entrada fixo para um domínio infinito de uma função de comprimento da entrada variável.

Se uma função de compressão recebe $m + t$ bits na entrada ($t > 0$) e gera uma saída de m bits, para calcular o resumo de uma mensagem x de comprimento maior do que $m+t$, podemos, a partir desta função de compressão, construir uma função de hash iterada seguindo estes passos:

- A partir da menasgem x construir uma cadeia cujo comprimento seja um múltiplo de t, e dividí-la em blocos de comprimento igual a t. Isto pode ser feito concatenando-se a mensagem

x com uma cadeia de bits (de zeros por exemplo) de comprimento tal que o comprimento da cadeia resultante da concatenação seja um múltiplo de t. Normalmente esta cadeia concatenada contém também o valor do comprimento da mensagem x. É importante que não seja possível que duas mensagens diferentes resultem em uma cadeia igual, ou seja, cada mensagem x deve resultar na construção de uma cadeia de comprimento múltiplo de t diferente, caso contrário seria fácil encontrar uma colisão.

- Aplicar a função de compressão repetidamente, obtendo a cada aplicação da função uma saída de comprimento igual a m. A entrada da função de compressão deve ser a concatenação da saída de comprimento m resultante da aplicação anterior da função de compressão com cada um dos blocos de comprimento t construídos no primeiro passo. Portanto a entrada terá o comprimento $m + t$. Na primeira aplicação da função de compressão não existe uma saída anterior, neste caso deve-se utilizar um valor inicial que é uma cadeia de bits de comprimento m.

- A saída da função de hash iterada será a saída obtida após a última aplicação da função de compressão ao último bloco de comprimento t da cadeia formada no primeiro passo. Opcionalmente poderá ser aplicada a esta última saída uma função de transformação final para, por exemplo, se obter uma saída da função de hash iterada com um comprimento diferente de m.

Esta é uma construção genérica para uma função de hash iterada.

2.3.1 A construção Merkle-Damgård

A construção Merkle-Damgård pode ser vista como um caso especial da construção genérica de uma função de hash iterada. A função de hash resultante desta construção satisfaz algumas propriedades de segurança, como resistência à colisão se a função de compressão utilizada tiver essa propriedade.

Esta construção segue o processo da construção genérica, mas com algumas particularidades:

- Se a função de compressão recebe $m+t$ bits na entrada e gera uma saída de m bits, t deve ser maior ou igual a 2. Isto porque um bit da entrada será utilizado para diferenciar a primeira chamada à função de compressão das demais. Por esse mesmo motivo, a cadeia formada a partir da mensagem x que será aplicada à entrada da função de compressão durante a iteração deverá ter um comprimento múltiplo de $t-1$ e deverá ser dividida em blocos de comprimento igual a $t-1$. Esta cadeia será formada pela própria mensagem x, concatenada a d zeros, sendo d o número de bits necessários para completar o último bloco de comprimento $t-1$, concatenada a um bloco adicional de comprimento $t-1$ que contém a representação binária de d . Este bloco adicional contendo a representação binária de d tem a função de garantir que cada mensagem x resulte em uma cadeia de bits diferente após esta preparação, de forma a preservar a resistência à colisão.

- O valor de entrada sobre o qual será aplicada repetidamente a função de compressão será formado como na construção genérica, pela concatenação dos m bits da saída da última aplicação da função de compressão com o próximo bloco da cadeia de bits preparada a partir

da mensagem x. A diferença é que o comprimento de cada bloco da cadeia de bits é $t-1$ bits para que seja concatenado também um bit que deve ser 0 na primeira chamada da função de compressão, e 1 nas demais chamadas.

A função de hash iterada na construção Merkle-Damgård será uma função de hash resistente à colisão se a função de compressão utilizada for resistente à colisão. Isto pode ser provado, pois é fácil demonstrar que se uma colisão pode ser encontrada para a função de hash iterada então uma colisão poderá ser encontrada na função de compressão. As inclusões de d no último bloco da cadeia de bits formada a partir da mensagem x, e do bit que diferencia a entrada na primeira aplicação da função de compressão das demais, garantem que uma colisão na função de hash iterada será necessariamente acompanhada de uma colisão em uma das aplicações da função de compressão.

Capítulo 3

MD6

3.1 Especificação do MD6

A constante evolução do poder computacional disponível e da descoberta de fragilidades nas funções de hash estabelecidas exigem o desenvolvimento de algoritmos de hash mais seguros e eficientes. Ron Rivest projetou o algoritmo de hash MD5 em 1991. Naquela época o poder computacional disponível era significativamente inferior ao poder computacional disponível hoje. A resistência à colisão do MD5 foi quebrada em 2004.

O algoritmo de hash MD6 foi projetado para ser mais robusto em relação a fragilidades hoje conhecidas, e para já estar preparado para a provável evolução do poder computacional nos próximos anos, que se dará não pelo aumento da frequência de clock dos processadores, mas pela multiplicação do número de unidades de processamento em um sistema, o que favorece um algoritmo que possibilite a exploração da capacidade de realizar cálculos de forma paralela, independente. Por isso o modo de operação do MD6 inclui um modelo estruturado em árvore, que diferentemente do modelo sequencial (como a construção Merkle-Damgård apresentada), possibilita a execução repetida da função de compressão de forma paralela.

No modo de operação em árvore do MD6, cada nó da árvore representa uma aplicação da função de compressão do MD6, cuja taxa de compressão é de 4 para 1.

Como boa parte dos sistemas pode não ter a capacidade de armazenamento necessária para a execução do modo de operação em árvore, o MD6 inclui também a alternativa de um modo de operação sequencial. Na verdade, o MD6 possibilita a combinação dos dois modos, podendo ser configurado gradualmente para operar desde apenas no modo em árvore, parte no modo em árvore e parte no modo sequencial, até apenas no modo sequencial. Um dos parâmetros da função limita quantos níveis serão calculados no modo em árvore; quando o limite é atingido o cálculo da função passa a ser executado no modo sequencial.

O tamanho do bloco da mensagem de entrada na função de compressão do MD6 é grande se comparado a outras funções: 512 bytes. A chave da função de compressão do MD6 pode ter até 512 bits.

Cada aplicação da função de compressão recebe como parte da entrada uma identificação única da localização do nó na estrutura sobre o qual a função de compressão está sendo aplicada. Isto torna o algoritmo resistente ao ataque do tipo gerar e colar.

Para aumentar a resistência a ataques de extensão, onde se calcula o resumo de uma mensagem útil para o cálculo do resumo de outra mensagem, a entrada da função de compressão do MD6 tem

também um bit para diferenciar a aplicação final da função de compressão das demais.

Operações cujo tempo de execução é dependente dos dados sobre os quais elas estão operando podem resultar em vulnerabilidade a determinados tipos de ataque. Entre estas operações estão rotações dependentes dos dados, busca de dados em tabelas (S-boxes) e algumas operações complexas como multiplicação. Por isso a função de compressão do MD6 utiliza apenas operações em palavras de 64 bits e apenas as operações $\oplus$ (Ou exclusivo), $\wedge$ (E) e deslocamento de bits por uma quantidade fixa.

O número de bits no resumo da mensagem do MD6, d, pode variar de 1 a 512, e portanto inclui os tamanhos exigidos para o SHA-3: 224, 256, 384 e 512 bits. O número de rodadas (cada rodada tem 16 passos), r, é variável e pode ser fornecido como um dado de entrada. Se o número de rodadas não é fornecido será utilizado um valor padrão de acordo com o tamanho do resumo da mensagem: $r = 40 + (d/4)$.

3.1.1 A função de compressão do MD6

A função de compressão do MD6 trabalha com palavras de 64 bits.

Os parâmetros de entrada são: 64 palavras de dados; 8 palavras para a chave; 1 palavra para o identificador único U; 1 palavra de parâmetros V contendo: a quantidade de bits que foram preenchidos para completar os dados, o tamanho da chave (em bytes), o bit z (que diferencia a última aplicação da função de compressão das demais), o limite de altura da árvore do modo de operação L, o tamanho em bits do resumo da mensagem d e o número r de rodadas.

Estas 74 palavras são concatenadas a uma constante de 15 palavras, resultando em um total de 89 palavras. Estas 89 palavras passam por um mapeamento 1 pra 1, e as 89 palavras que resultam do mapeamento são truncadas para que se obtenham as 16 palavras que serão a saída da função de compressão. Portanto a taxa de compressão é de 4 para 1, já que entram 64 palavras de dados e saem 16.

Algoritmo 1 Função de compressão

Entrada: $A[0 \cdots 88]$ de $A[0 \cdots 16r + 88]$
  para $i = 89$ **a** $16r + 88$ **:**
    $x = S_i \oplus A[i - 17] \oplus A[i - 89] \oplus (A[i - 18] \wedge A[i - 21]) \oplus (A[i - 31] \wedge A[i - 67])$
    $x = x \oplus (x >> r_i)$
    $A[i] = x \oplus (x << l_i)$

  retorne $A[16r + 73 \cdots 16r + 88]$

No algoritmo 1, $A[0..88]$ é o vetor com as 89 palavras de entrada. r é o número de rodadas. A cada rodada são calculadas 16 novas palavras. $A[0..16r+88]$ é o vetor completo com as 89 palavras de entrada mais as $16r$ palavras calculadas nas r rodadas. Cada palavra é calculada a partir das 89 palavras imediatamente anteriores a ela no vetor. As 16 palavras calculadas na última rodada são a saída da função de compressão.

Se a memória do sistema é limitada pode-se utilizar um vetor de apenas 89 palavras que deve ser deslocado a cada nova palavra calculada de forma a incluir a nova palavra calculada e excluir a palavra que não será mais utilizada nos próximos cálculos.

As escolhas dos índices relativos 17, 18, 21, 31 e 67 visam otimizar a difusão. As constantes S_i mudam ao final de cada rodada. As quantidades de deslocamento de bits se repetem a cada rodada

e são definidas pela tabela 3.1, que também visa à obtenção da difusão máxima.

Tabela 3.1: *Quantidade de deslocamento de bits*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
r_i	10	5	13	10	11	12	2	7	14	15	7	13	11	7	6	12
l_i	11	24	9	16	15	9	27	15	6	2	29	8	15	5	31	9

3.1.2 O modo de operação do MD6

O modo de operação do MD6 pode ser representado pelo algoritmo 2.

Algoritmo 2 Modo de operação

Entrada:

M : A mensagem para a qual se deseja obter o resumo.
d : O comprimento do resumo da mensagem desejado (em bits).
K : 8 palavras contendo uma chave de keylen bytes.
L : O número máximo de níveis, ou o número de passos paralelos.
r : O número de rodadas.

Saída:

D : O resumo da mensagem desejado.
Inicialização:
$l = 0, M_0 = M$.
Loop principal, nível por nível:
$l = l + 1$.
Se $l = L + 1$, retorne SEQ(M_{l-1}, d, K, L, r) como a saída da função.
$M_l =$ PAR(M_{l-1}, d, K, L, r, l).
Se o comprimento de M_l é igual a 16 palavras, retorne os últimos d bits de M_l como a saída da função. Senão continue o loop.

O modo de operação do MD6 é parametrizado pelo parâmetro ?nível máximo? L. Se L é igual ao valor máximo, então o modo de operação será completamente em árvore e a operação SEQ nunca será usada. Se $L = 0$, então o modo de operação será iterativo, parecido com o Merkle-Damgård e a operação PAR nunca será usada. Para outros valores de L, a operação PAR será usada no máximo L vezes. Se após L chamadas a PAR o comprimento de M_l ainda não é igual a 16 palavras, a operação SEQ será usada.

A operação PAR, descrita pelo algoritmo 3, é uma operação de compressão em paralelo que calcula o nível l da árvore a partir do nível $l - 1$.

A operação SEQ, descrita pelo algoritmo 4, é uma operação de hash sequencial como a construção Merkle-Damgård, produzindo um resumo de saída final.

3.2 O projeto do MD6

Descreveremos agora algumas das razões que direcionaram as decisões tomadas no projeto do MD6.

Algoritmo 3 Operação PAR

Entrada:

M_{l-1} : Uma mensagem de comprimento m_{l-1} em bits.
d : O comprimento do resumo da mensagem desejado (em bits).
K : 8 palavras contendo uma chave de *keylen* bytes.
L : O número máximo de níveis, ou o número de passos paralelos.
r : O número de rodadas.
l : O número do nível, $1 \leq l \leq L$.

Saída:

M_l : Uma mensagem de comprimento em bits igual a $1024 \cdot \lceil (m_{l-1}/4096) \rceil$.

Inicialização:

Q = vetor de comprimento igual a 15 palavras contendo a parte fracionária de raíz de 6.
f_r representa a função de compressão do MD6 que mapeia um bloco B de dados de entrada de 64 palavras em uma saída C de 16 palavras, usando r rodadas para o cálculo (f_r também recebe como entrada 25 palavras de informação auxiliar).

Compactação:

Se necessário, e apenas se necessário, completar a entrada M_{l-1} adicionando bits iguais a 0 até que o comprimento de M_{l-1} se torne um múltiplo de 64 palavras. Então M_{l-1} pode ser visto como uma sequência $B_0, B_1, \ldots, B_{j-1}$ de blocos de 64 palavras, onde $j = \lceil (m_{l-1}/4096) \rceil$.
Para cada bloco B_i, $i = 0, 1, \ldots, j-1$, calcule C_i em paralelo assim:

- Seja p o número de bits usados para completar o bloco B_i; $0 \leq p < 4096$ (p só pode ser diferente de 0 se $i = j - 1$).
- Seja $z = 1$ se $j = 1$, caso contrário seja $z = 0$ (desta forma z só será igual a 1 no último bloco a ser comprimido no cálculo completo do MD6).
- Seja V uma palavra com o valor $r||L||z||p||keylen||d$.
- Seja $U = l \cdot 2^{56} + i$ um identificador de nó único, uma palavra cujo valor será único para esta operação da função de compressão.
- Seja $C_i = f_r(Q||K||U||V||B_i)$ (C_i tem comprimento igual a 16 palavras).

retorne $M_l = C_0||C_1||\ldots||C_{j-1}$.

Algoritmo 4 Operação SEQ

Entrada:

M_L : Uma mensagem de comprimento m_L em bits.
d : O comprimento do resumo da mensagem desejado (em bits).
K : 8 palavras contendo uma chave de *keylen* bytes.
L : A altura máxima da árvore.
r : O número de rodadas.

Saída:

D : O resumo da mensagem de comprimento igual a d bits.

Inicialização:

Q = vetor de comprimento igual a 15 palavras contendo a parte fracionária de raíz de 6.
f_r representa a função de compressão do MD6 que mapeia um bloco B de dados de entrada de 64 palavras em uma saída C de 16 palavras, usando r rodadas para o cálculo (f_r também recebe como entrada 25 palavras de informação auxiliar).

Loop principal:

Seja C_{-1} o vetor de zeros de comprimento igual a 16 palavras (Este é o vetor inicial ?IV?).
Se necessário, e apenas se necessário, completar a entrada M_L adicionando bits iguais a 0 até que o comprimento de M_L se torne um múltiplo de 48 palavras. Então M_L pode ser visto como uma sequência $B_0, B_1, \ldots, B_{j-1}$ de blocos de 48 palavras, onde $j = \lceil (m_L/3072) \rceil$.
Para cada bloco B_i, $i = 0, 1, \ldots, j-1$, em sequência, calcule C_i assim:

- Seja p o número de bits usados para completar o bloco B_i; $0 \leq p < 3072$ (p só pode ser diferente de 0 se $i = j - 1$).
- Seja $z = 1$ se $i = j - 1$, caso contrário seja $z = 0$ (desta forma z só será igual a 1 no último bloco a ser comprimido no cálculo completo do MD6).
- Seja V uma palavra com o valor $r||L||z||p||keylen||d$.
- Seja $U = L \cdot 2^{56} + i$ um identificador de nó único, uma palavra cujo valor será único para esta operação da função de compressão.
- Seja $C_i = f_r(Q||K||U||V||C_{i-1}||Bi)$ (C_i tem comprimento igual a 16 palavras).

retorne os últimos d bits de C_{j-1} como a saída da função.

3.2.1 As entradas da função de compressão

A função de compressão utilizada em uma função de hash iterada através de um modo de operação como o Merkle-Damgård terá duas entradas principais: um bloco da mensagem e a saída da última aplicação da função de compressão. Estas duas entradas normalmente são utilizadas para diferentes propósitos no cálculo da função de compressão.

No modo de operação em árvore do MD6 as funções de compressão que estão nas folhas da árvore recebem como entrada apenas blocos da mensagem e as demais funções de compressão recebem como entrada apenas blocos de dados que são a saída das funções de compressão que estão no nível anterior da árvore. Por isso a função de compressão do MD6 é mais genérica e não diferencia se a entrada contém um bloco da mensagem ou uma saída da aplicação da função de compressão do nível anterior.

Além das entradas principais, uma função de compressão pode ter entradas auxiliares. As funções de compressão tradicionais não contém entradas auxiliares, mas essas entradas são recomendadas pois elas podem ajudar na redução de vulnerabilidades do modo de operação Merkle-Damgård tradicional e também ajudam na demonstração de provas da segurança da função de hash. Por isso entradas auxiliares estão embutidas na entrada da função de compressão do MD6, que são: a chave K, o identificador único de nó U e a palavra de controle V.

3.2.2 Decisões influenciadas pela evolução da disponibilidade de memória

Embora a disponibilidade de memória seja cada vez menos um problema, algumas aplicações ainda são bem limitadas. O MD6 foi projetado de forma a ser implementável usando no máximo 1 KB de memória RAM. O cálculo de uma função de compressão pode ser realizado com menos de 1 KB de RAM e se o modo de operação for configurado com $L = 0$, ou seja, no modo de cálculo apenas sequencial, o cálculo de toda a função de hash pode ser executado com menos de 1 KB de RAM. Caso o sistema tenha uma disponibilidade de memória maior pode-se aumentar o valor de L, o que possibilita a realização do cálculo das funções de compressão em paralelo, aumentando assim a eficiência da função de hash.

Uma estratégia sugerida por Stefan Lucks para o projeto de uma função de hash é a de fazer que a saída da função de compressão, que será utilizada na entrada da próxima função de compressão da cadeia, tenha o dobro do comprimento do resumo final calculado pela função de hash completa. Como o SHA-3 deve poder calcular um resumo final de até 512 bits, então as saídas da função de compressão devem ter 1024 bits para atender essa estratégia. Por isso que a saída da função de compressão do MD6 tem 1024 bits, e como o fator de compressão escolhido foi 4, o bloco de dados na entrada é de 4096 bits. Este comprimento para o bloco de dados da entrada é relativamente grande se comparado a outras funções de hash como o MD5 (640 bits), o SHA-1 (672 bits) e os membros da família SHA-2 (768 bits ou 1536 bits), e é possibilitado pelo aumento da disponibilidade de memória nos sistemas atuais.

Existem várias vantagens em se ter uma função de compressão com entrada de comprimento grande: fica mais fácil adicionar as entradas auxiliares no cálculo da função de compressão, já que essas entradas auxiliares ocuparão uma fração menor da entrada total, causando um menor impacto no desempenho da função; uma entrada de comprimento grande possibilita um maior grau de paralelismo no cálculo da função de compressão o que possibilita que os 16 passos de um round

do MD6 sejam cálculados de uma só vez, em paralelo; uma entrada de comprimento grande pode ter o tamanho suficiente para que muitas mensagens comumente usadas caibam nessa entrada, o que possibilita que o resumo dessas mensagens seja calculado em apenas uma função de compressão; uma entrada de comprimento grande deve dificultar ataques a essa função, como o ataque diferencial, devido ao aumento do número de passos no cálculo da função de compressão.

3.2.3 Multiprocessamento

O modo de operação em árvore do MD6 permite que muitas funções de compressão sejam calculadas em paralelo, o que garante um altíssimo desempenho no cálculo da função de hash quando realizado em sistemas multiprocessados. O parâmetro L permite a configuração do MD6 desde um modo completamente sequencial, para sistemas onde a memória é muito limitada, até o modo completamente em árvore. A memória necessária será proporcional à altura da árvore.

3.2.4 A chave do MD6

A chave de 64 bytes incluída na entrada da função de compressão do MD6 pode ser usada como:

- uma chave secreta para um código de autenticação de mensagem;
- um valor público para variar os resumos calculados pelo MD6, o que pode ser utilizado por exemplo no cálculo dos resumos de senhas antes de armezená-los dificultando o ataque de um adversário que poderia calcular previamente um dicionário dos resumos das senhas mais usadas, mas que com essa variação ele precisaria refazer os cálculos para cada valor público usado como chave;
- um índice aleatório para a função de hash, o que tem aplicações interessantes e benefícios de segurança.

A chave do MD6 é relativamente grande porque:

- o fato do bloco de entrada da função de compressão do MD6 ser grande faz com que o custo de adicionar essa chave seja relativamente pequeno, como já dissemos;
- porque facilita a utilização de chaves baseadas em texto;
- porque torna-se possível utilizar a própria saída do MD6 como uma chave, mesmo que a saída seja a maior possível, ou seja, 512 bits.

O tamanho da chave do MD6 é variável entre 0 e 512 bits e a inclusão do comprimento da chave na entrada da função de compressão torna essa função segura em relação a determinados ataques já que isso garante que duas chaves diferentes provocarão um comportamento diferente do MD6, mesmo que uma chave seja o prefixo da outra.

3.2.5 A presença das entradas auxiliares em cada função de compressão

Uma característica importante do MD6 é que as entradas auxiliares estão presentes no cálculo de todas as funções de compressão. Por exemplo, a chave do MD6 não é simplesmente anexada

à mensagem de entrada da função de hash completa, ela estará presente na entrada de todas as funções de compressão. Isto é uma proteção contra ataques de extensão do comprimento.

Outra informação presente em todas as funções de compressão é o identificador único de nó U. Com este identificador cada função de compressão ?sabe? exatamente em que posição ela está dentro da árvore. Através da entrada auxiliar z, a função de compressão ?sabe? ainda se ela é a última função de compressão no cálculo do MD6. Desta forma qualquer cálculo parcial da função de hash pode ser usado em apenas um lugar dentro da árvore de cálculo completa. Isto evita ataques do tipo ?copiar e colar?, onde se move o cálculo de uma função de hash de um lugar para outro e também evita ataques que realizam cálculos parciais da função de hash na esperança de que eles possam servir em algum outro lugar para se achar uma colisão.

Informações de controle também estão presente em todas as funções de compressão. Essas informações também especificam a estrutura da árvore, indicando quantos bits de mensagem tem na entrada de uma função de compressão que é uma folha da árvore, ou quantas funções de compressão são filhas de uma função que não é folha. Outras informações de controle incluídas na entrada da função de compressão são: o tamanho do resumo da mensagem desejado d, o numero de rodadas r e o parâmetro do modo de operação L.

A inclusão destas informações na entrada da função de compressão garante que qualquer parâmetro que seja alterado provocará um comportamento diferente da função de hash, sem relação com o comportamento anterior à mudança.

3.2.6 Conjunto de instruções limitado

O conjunto de instruções utilizado na definição do MD6 é bem restrito o que dá vantagens em termos de segurança, simplicidade e eficiência.

Não são utilizadas operações cujos tempos de execução dependem dos dados de entrada pois estas operações possibilitam um tipo de ataque que explora essa característica. Por isso não são usadas instruções complexas como:

- multiplicação e divisão;
- deslocamentos ou rotações por quantidade dependente da entrada;
- saltos condicionais dependentes da entrada;
- buscas em tabelas onde o índice da tabela é dependente da entrada, e por isso não é utilizado ?S-box?, um componente muito utilizado em criptografia.

As operações utilizadas são apenas: XOR, AND e deslocamento por uma quantidade fixa. Com essas instruções pode-se construir as outras operações utilizadas: NOT, OR e rotação por uma quantidade fixa.

3.2.7 Registrador de deslocamento de realimentação não linear

O MD6, assim como muitas outras funções de hash, pode ser visto como um registrador de deslocamento de realimentação não linear, ou NLFSR (Nonlinear feedback shift register).

Se olharmos o algoritmo da função de compressão veremos as posições relativas utilizadas na realimentação, que são: $t_0 = 17$, $t_1 = 18$, $t_2 = 21$, $t_3 = 31$, $t_4 = 67$ e $t_5 = 89$. As posições t_0, t_1 e t_2

foram escolhidas intencionalmente como um valor pequeno para que aquelas diferenças se propaguem rapidamente, mas não tão pequenas para permitir o paralelismo da função de compressão, por isso o menor valor, t_0, é 17, o que permite que os 16 passos de uma rodada sejam executados em paralelo, de uma vez só. Além disso, t_0 é primo em relação ao comprimento de um round. Esta é uma posição de realimentação linear para um efeito avalanche mais rápido. A posição $n = 89$ foi necessariamente escolhida como uma posição para realimentação linear porque se deseja que a operação de realimentação seja inversível. Todas as posições foram escolhidas para não serem múltiplos de 16 com resíduos diferentes no módulo 16. É necessário também que $t_4 - t_3$ seja diferente de $t_2 - t_1$ para que as portas AND peguem entradas em distâncias diferentes uma da outra. As posições t_1 a t_4 foram escolhidas por uma programa computacional de busca por força bruta que procurou valores que minimizassem a quantidade de passos (o valor conseguido foi 102) que devem ser calculados para garantir que cada um dos últimos 16 valores calculados dependam de todas as 89 palavras de entrada da função de compressão.

As constantes de rodada S_i são geradas por uma recorrência de relação um pra um, ou seja, a partir de S_i pode-se determinar S_{i+1} e vice-versa.

3.3 Implementação do MD6 por software

A implementação do MD6 é a mesma para todos os tamanhos de resumo de mensagem. O desempenho do MD6-512 nas plataformas de referência da competição pública do NIST para escolha do SHA-3 é:

- Na plataforma de 32 bits: 22 MB/seg
- Na plataforma de 64 bits: 49 MB/seg

Já o desempenho do SHA-512 (pertencente à família SHA-2) nas mesmas plataformas é:

- Na plataforma de 32 bits: 38 MB/seg
- Na plataforma de 64 bits: 202 MB/seg

O NIST estabeleceu que para ser competitivo um candidato a SHA-3 precisaria ser no mínimo tão rápido quanto o SHA-2 nas plataformas de referência. Este é o motivo pelo qual Ron Rivest comunicou que o MD6 não estaria pronto para passar à segunda fase da competição, por ele estar longe de ser tão rápido quanto o SHA-2 nas plataformas de referência.

O modo de operação em árvore do MD6 proporciona uma grande eficiência do algoritmo quando executando em um sistema com múltiplas unidades de processamento, o que não é caso das plataformas de referência. Em sistemas com múltiplas unidades de processamento o MD6 tende a ter um desempenho muito superior ao SHA-2, visto que a velocidade de processamento do MD6 cresce linearmente com o aumento do número de unidades de processamento, enquanto a velocidade do SHA-2 permanece constante. O aumento do número de unidades de processamento não faz nenhum efeito no desempenho do SHA-2, já que o seu modo de operação não permite paralelismo, os cálculos precisam ser executados sequencialmente. Para se ter uma idéia, o MD6-256 ultrapassa a velocidade de 1 GB/seg em um sistema com 15 unidades de processamento. Utilizando uma placa de vídeo

padrão, é possível alcançar a velocidade de 375 MB/seg, com 4 placas a velocidade de 1.5 GB/seg é alcançada.

Em um processador de 8 bits, utilizando-se apenas o modo sequencial ($L = 0$), é necessário menos de 1 KB de RAM para rodar o MD6. Com uma frequência de clock de 20 MHz a função de compressão para o MD6-224 é executada em 110 ms para um programa escrito em C, e 44 ms para um programa escrito em assembler.

A maior parte do tempo de processamento no cálculo do MD6 é gasta no ciclo interno da função de compressão. A função de compressão é relativamente simples e não há muito o que possa ser otimizado na sua implementação.

As informações adicionais acrescentadas às entradas de todas as funções de compressão para aumentar a segurança do MD6 contribuem para tornar o processamento mais lento. Mas o principal motivo para que o processamento do MD6 nas plataformas de referência seja relativamente lento é o alto número de rodadas da função de compressão. O número de rodadas, definido de acordo com o número de bits que se deseja para o resumo da mensagem, deve tentar garantir que um ataque diferencial padrão ao MD6 seja menos eficiente que um ataque pelo paradoxo do aniversário (força bruta), de acordo com a demonstração apresentada adiante da carga mínima de trabalho de um ataque diferencial padrão ao MD6.

3.4 Implementação do MD6 por hardware

A implementação do MD6-512 em um FPGA de 14K unidades lógicas utilizou 5.3K unidades lógicas em uma implementação para calcular uma rodada por vez, e 7.9K unidades lógicas em uma implementação para calcular 2 rodadas por vez. Com uma frequência de clock de 100 MHz foi medida a velocidade de 240 MB/seg na implementação para calcular 2 rodadas por vez, independentemente do tamanho do resumo da mensagem configurado.

Capítulo 4

Análise da segurança do MD6

Como já comentado, ataques do tipo gerar e colar contra o MD6 não funcionam porque as funções de compressão recebem como um dos parâmetros de entrada um identificador único da localização do nó na estrutura sobre o qual a função de compressão está sendo aplicada. Este tipo de ataque faz cálculos especulativos na esperança de ?cortar e colar este cálculo em algum lugar onde ele se encaixe?.

Assim como na construção Merkle-Damgård, é possível provar que o modo de operação do MD6 preserva algumas das propriedades criptográficas da função de compressão.

O NIST estabeleceu alguns requisitos de segurança para um candidato a SHA-3. Alguns desses requisitos, para um resumo de mensagem de d bits, são:

- Resistência à colisão de aproximadamente $d/2$ bits.
- Resistência à primeira pré-imagem de aproximadamente d bits.
- Resistência à segunda pré-imagem de aproximadamente $d - k$ bits para qualquer mensagem mais curta do que $2k$ bits.

A tese de mestrado de Christopher Yale Crutchfield mostra que o modo de operação do MD6 preserva as propriedades da função de compressão de resistência à colisão, resistência à primeira pré-imagem e pseudo aleatoriedade. Para a propriedade de resistência à segunda pré-imagem ele não conseguiu demonstrar que ele preserva esta propriedade, embora ele tenha conseguido reduzí-la a uma propriedade com garantias de segurança mais fracas.

Pode ser mostrado que a versão com chave do MD6 não é vulnerável à criptanálise linear quando usada como um MAC de acordo com as recomendações do NIST.

Concentraremos nossa atenção na análise da resistência do MD6 aos ataques diferenciais.

4.1 Criptanálise diferencial

Os ataques diferenciais se mostraram muito poderosos quando aplicados às funções de hash existentes e várias delas não resistiram ou se mostraram vulneráveis à essa técnica, por exemplo: MD4, RIPEMD, MD5, SHA-0 e SHA-1. Este fato foi o principal motivo para que o NIST iniciasse a competição para a escolha do SHA-3. Portanto deveria ser fundamental que um candidado a SHA-3 demonstrasse ser seguro contra ataques diferenciais.

Nesta seção mostraremos a análise da resistência do MD6 aos ataques diferenciais que busquem encontrar uma colisão na função de hash. Estes ataques consistem em se escolher pares de mensagens de entrada com determinadas diferenças tentando-se encontrar um par tal que o par de resumo da mensagem na saída da função de hash não tenha diferença, o que significa encontrar uma colisão. Se a probabilidade de se encontrar esse par de mensagens não é desprezível, então calculando-se o resumo das mensagens de uma quantidade suficiente de pares de mensagens de entrada pode-se encontrar uma colisão.

Antes de mais nada deve-se estabelecer uma forma de medir a diferença entre duas mensagens e esta forma pode variar de acordo com as operações envolvidas na função de hash. A forma de medida mais utilizada e que será utilizada nesta demonstração é o ou-exclusivo.

Um *caminho diferencial* é um conjunto de diferenças entre o par de entradas, todos os estados intermediários e o par de saídas. Para o MD6 podemos expressar um caminho diferencial como: ΔA_i para $i = 0, \ldots, t + n - 1$.

É fácil notar que um *caminho diferencial de colisão* é um caminho onde $\Delta A_i = 0$ para $i = t + n - c, \ldots, t + n - 1$.

A propriedade mais importante de um caminho diferencial é a sua *probablidade* associada. A probabilidade de um determinado passo i de um caminho diferencial, p_i, é definida como a probabilidade de que o par de saída do passo siga o caminho diferencial, dado que o par de entrada satisfaz a diferença especificada pelo caminho diferencial.

O conjunto formado pela diferença na entrada, a diferença na saída e a probabilidade associada a um passo i é chamado de *característica diferencial* do passo i.

A probabilidade total de um caminho diferencial, p, é o produto das probabilidade em todos os passos, assumindo que o cálculo dos passos são independentes entre si. Logicamente o cálculo de cada passo não é indenpendente dos outros passos, mas, para uma função bem projetada, após um certo número de rodadas a saída de cada passo parece ser aleatória e o cálculo em cada passo parece ser independente dos outros. Isto normalmente é assumido na maior parte dos trabalhos existentes sobre criptanálise diferencial.

O MD6, após 10 rodadas, passa com sucesso por uma bateria de testes estatísticos, o que suporta essa presunção de aleatoriedade e independência de cálculo entre os passos.

A carga de trabalho de um ataque de colisão diferencial usando um determindo caminho diferencial é calculado como o inverso de alguma probabilidade p', que pode ser muito maior do que p. Primeiro porque escolhendo-se apropriadamente o par de entradas pode-se forçar $p_i = 1$ para os primeiros passos. Segundo porque para uma determinada diferença na entrada e saída podem existir vários caminhos com a mesma diferença na entrada e saída, mas valores intermediários diferentes. Então a probabilidade total seria a soma das probabilidades de cada um destes caminhos.

Definimos D_i como o peso de Hamming de uma determinada diferença ΔA_i, ou seja, o número de bits diferentes entre A_i e A'_i, ou $D_i = |\Delta A_i|$. Então, para um caminho diferencial $\{\Delta A_i\}$ definimos um *caminho diferencial de padrão de peso* como $\{D_i\}$.

Quando usamos o caminho diferencial de padrão de peso conseguimos separar o efeito das posições t_i do efeito dos deslocamentos de bits, em se tratando de criptanálise diferencial. Grosseiramente falando, as posições t_i influenciam na propagação das diferenças de um passo para o outro, enquanto que os deslocamentos de bit influenciam na mistura das diferenças dentro de uma palavra.

Como a sequencia D_i reflete principamente como as diferenças se propagam de um passo para o outro, ela facilita o estudo do efeito das posições t_i, sem nos importarmos muito em como as diferenças estão distribuidas na palavra. Isto simplifica bastante nossa análise.

4.1.1 Análise das operações realizadas em um passo

Analisemos as propriedades diferenciais de cada uma das 3 diferentes operações contidas em cada passo: XOR, AND e o operador g, que representa as operações de deslocamentos de bits. Adotaremos a seguinte notação:

- X, Y, Z para as entradas e saídas de w bits
- $\Delta X, \Delta Y, \Delta Z$ para as diferenças
- D_X, D_Y, D_Z para os pesos de Hamming
- x, y, z para um bit de palavras de w bits

A propriedade diferencial da operação XOR é direta, $\Delta Z = \Delta X \oplus \Delta Y$. Em termos do peso de Hamming, temos que: $\max(D_X, D_Y) - \min(D_X, D_Y) \leq D_Z \leq D_X + D_Y$.

Uma operação AND entre duas paravras de w bits pode ser vista como um conjunto de w portas AND independentes. Se os bits de entrada de cada porta AND forem x e y e a saída for z, o comportamento diferencial da porta AND depende das diferenças nas entradas, ou seja, Δx e Δy. Consideramos estes dois casos:

- Chamamos de porta AND "inativa" quando $\Delta x = \Delta y = 0$ e portanto temos que $\Pr[\Delta z = 0] = 1$.
- Chamamos de porta AND "ativa" quando $\Delta x = 1$ ou $\Delta y = 1$ e portanto temos que $\Pr[\Delta z = 0] = \Pr[\Delta z = 1] = 1/2$

Em termos do peso de Hamming, temos que: $0 \leq D_Z \leq D_X + D_Y$.

As portas AND ativas serão fundamentais na demonstração da carga de trabalho mínima de um ataque diferencial, já que esta é a única operação não trivial em termos de probabilidades diferenciais. Uma porta AND ativa sempre contribui com um probabilidade igual a 1/2 para a probabilidade total do caminho diferencial, não importa qual seja a diferença de saída da porta AND. O número total de portas AND ativas em um caminho diferencial está diretamente relacionado à probabilidade total do caminho.

O operador $g_{r,l}$ faz um espalhamento dos bits dentro de uma palavra. Sabemos que se $Z = g_{r,l}(X)$, então $\Delta Z = g_{r,l}(\Delta X)$.

A combinação de um descocamento e um XOR pode no máximo dobrar o número de diferenças, como são realizadas duas combinações de operações (uma com deslocamento pra direita e outra com deslocamento pra esquerda) temos que: $D_Z \leq 4D_X$.

Cada par de quantidade de deslocamentos (r, l) foi escolhido de forma que se $0 < D_X \leq 4$ então $D_Z \geq 2$.

Ou seja, para que a diferença na saída seja de apenas um bit é necessário que a diferença na entrada seja de 5 ou mais bits. Isto foi projetado desta forma para impedir a propagação de diferenças

de apenas um bit, dificultando a obtenção de caminhos diferenciais com pesos de Hamming muito baixos, sendo impossivel conseguir um caminho onde todos os pesos são no máximo 1.

Se $D_X > 4$ então $D_Z > 0$, já que se existem diferenças na entrada devem existir diferenças na saída.

Vamos agora combinar em duas partes as operações executadas em um passo:

$$X = A_{i-t_0} \oplus A_{i-t_5} \oplus (A_{i-t_1} \wedge A_{i-t_2}) \oplus (A_{i-t_3} \wedge A_{i-t_4}), \tag{4.1}$$

$$A_i = g(X). \tag{4.2}$$

Usando as desigualdadas apresentadas para cada operação podemos derivar limites superior e inferior para D_X:

$$D_X \leq UB_X = \sum_{k=0}^{5} D_{i-t_k}, \tag{4.3}$$

$$D_X \geq LB_X = \max(D_{i-t_0}, D_{i-t_5}) - \min(D_{i-t_0}, D_{i-t_5}) \sum_{k=1}^{4} D_{i-t_k}. \tag{4.4}$$

Focando no peso de Hamming ao invés de se focar no real valor das diferenças perde-se certa precisão na análise, mas evita-se a complicação de ter que analizar como as diferenças de bit individualmente podem se alinhar de um operação para outra, além de possibilitar a busca de caminhos diferenciais de padrões de peso válidos através de uma busca auxiliada por um programa computacional.

4.1.2 Carga mínima de trabalho de um ataque diferencial padrão

O objetivo agora é provar que ataques diferenciais padrão contra o MD6 são menos eficientes para encontrar colisões do que o ataque pelo paradoxo de aniversário. Ou seja, precisamos provar que a probabilidade de se encontrar qualquer caminho diferencial de colisão na função de compressão do MD6 é no máximo $2^{-d/2}$, o que significa dizer que a carga de trabalho de um ataque diferencial padrão é no mínimo $2^{d/2}$, que é o limite teórico do paradoxo do aniversário.

Como vimos, cada porta AND ativa em um caminho diferencial contribui com a probabilidade de $1/2$, então se o número de portas AND ativas em um caminho diferencial válido do MD6 é no mínimo $d/2$, a probabilidade associada a este caminho será no máximo $2^{-d/2}$.

Cada diferença de bit em um caminho diferencial de padrões de peso pode ativar até 4 portas AND em 4 passos distintos, uma para cada posição t_1, t_2, t_3 e t_4. Em alguns casos uma diferença de bit pode não ativar as 4 portas AND, e estes casos devem ser levados em consideração para não contarmos portas AND ativas a mais:

- Se duas diferenças de bit ativam a mesma porta AND.
- Se duas portas AND são ativadas no mesmo passo.
- Se uma porta AND está além do limite de rodadas. Só contamos as portas AND ativas que tem as duas entradas dentro do limite de rodadas em que está sendo feita a busca.

Para fazer a busca de caminhos diferenciais de padrões de peso possíveis desejamos eliminar o máximo possível de padrões inválidos. Utilizando (4.3) e (4.4) e as desigualdades mostradas para a função g, podemos eliminar os seguintes valores de D_i em um determinado passo i:

1. $D_i = 0$ e $LB_X > 0$
2. $D_i > 4UB_X$
3. $D_i = 1$ e $UB_X < 5$

A tabela 4.1 mostra os resultados obtidos pelo grupo de trabalho do MD6 através de um programa computacional para buscar a número mínimo de portas AND ativas em qualquer caminho diferencial de até s rodadas.

Tabela 4.1: *Número mínimo de portas AND ativas em qualquer caminho diferencial de até s rodadas*

s	≤ 5	6	7	8	9	10	11	12	13	14	15
Número mínimo de portas AND ativas	0	3	4	4	4	4	7	13	19	20	26

O programa que escrevi para fazer obter esses resultados roda relativamente rápido até o cálculo de 14 rodadas, mas com 15 rodadas ele já fica bem lento e isto deve explicar porque o grupo de trabalho do MD6 também só conseguiu o resultado para até 15 rodadas.

Agora utilizamos o valor do número mínimo de portas AND ativas em $s = 15$ rodadas, 26, para expandir o resultado a um número r qualquer de rodadas através da fórmula: $AAG_r \geq AAG_s \times \lfloor r/s \rfloor$, onde AAG_x é o número mínimo de portas AND ativas em x rodadas (AAG = Active AND Gate).

Antes disso deve-se tomar o cuidado de deixar uma margem de segurança, porque alguém que tente atacar a função pode conseguir penetrar algumas rodadas no começo do cálculo do hash manipulando as entradas e influenciando o comportamento do caminho diferencial. Estabeleceu-se uma margem de segurança conservadora de 15 rodadas, ou seja, substitui-se na fórmula o número de rodadas r por $r - 15$.

A tabela 4.2 mostra o resultado final da carga de trabalho mínima de um ataque diferencial padrão ao MD6 e compara com a carga de trabalho de um ataque pelo paradoxo do aniversário, mostrando que a carga de trabalho de um ataque diferencial é maior que a carga de trabalho de um ataque pelo paradoxo do aniversário, que é o que queríamos demonstrar.

Tabela 4.2: *Resultado final da carga de trabalho mínima de um ataque diferencial padrão ao MD6 (LB é a carga de trabalho mínima e BB é a carga de trabalho de um ataque pelo paradoxo do aniversário)*

d	r	$r-15$	$\lfloor \frac{r-15}{15} \rfloor$	$AAG_{r-15} \geq$	$LB \geq$	BB
40	50	35	2	52	2^{52}	2^{20}
80	60	45	3	78	2^{78}	2^{40}
128	72	57	3	78	2^{78}	2^{64}
160	80	65	4	104	2^{104}	2^{80}
224	96	81	5	130	2^{130}	2^{112}
256	104	89	5	130	2^{130}	2^{128}
384	136	121	8	208	2^{208}	2^{192}
512	168	153	10	260	2^{260}	2^{256}

Capítulo 5

Análise de outras configurações para a função de compressão do MD6

Ao apresentar a demonstração da segurança do MD6 contra ataques diferenciais padrão, vimos que ela é dependente do número de rodadas utilizado na função de compressão. O número de rodadas deve garantir uma quantidade mínima de portas AND ativas na execução da função de compressão pois a resistência a um ataque diferencial está diretamente relacionada a essa quantidade.

Como a velocidade de processamento do MD6 é relativamente baixa devido principalmente ao número de rodadas, um caminho para se tentar resolver o problema da velocidade é diminuindo o número de rodadas da função de compressão.

A quantidade mínima de portas AND ativas em um determindado número de rodadas foi calculado através de um programa computacional e entre os principais parâmetros utilizados por esse programa estão as posições t_0, t_1, t_2, t_3, t_4 e t_5, que são utilizadas no cálculo de cada passo da função de compressão.

Como visto em 3.2.7 na página 16, as posições t_0 e t_5 foram fixadas por motivos especiais em 17 e 89, respectivamente. As posições t_1, t_2, t_3 e t_4 foram escolhidas através de um programa computacional buscando minimizar a quantidade de passos que devem ser calculados para garantir que cada um dos últimos 16 valores calculados dependam de todas as 89 palavras de entrada da função de compressão. E de fato foram escolhidas as posições com as quais se alcança essa situação no menor número de passos, 102.

As posições foram escolhidas de acordo com um critério, mas teoricamente elas tambem tem grande influência na quantidade de portas AND ativas em um determinadao número de rodadas e não se sabe qual é a configuração que maximizaria essa quantidade.

Decidimos então testar outras configurações para as posições $t_1, t_2, t_3,$ e t_4 para comparar os resultados obtidos em relação à quantidade de portas AND ativas em um determinado número de rodadas.

Talvez exista uma configuração de posições que fique bem próxima da melhor configuração em relação ao critério que foi utilizado na escolha dessas posições, mas que resulte em um maior número de portas AND ativas em um determinado número de rodadas possibilitando uma redução do número de rodadas necessárias para a demonstração da resistência da função de compressão aos ataques diferenciais padrão o que por consequencia melhoraria o desempenho do MD6 em termos da velocidade de processamento.

5.1 Levantando configurações candidatas

O primeiro passo foi fazer o levantamento de outras possíveis configurações para as posições $t_1, t_2, t_3,$ e t_4. Com pequenas modificações no programa utilizado para a descoberta da configuração que precisa do menor número de passos para que cada uma das últimas 16 palavras calculadas dependam de todas as 89 palavras da entrada, obtemos as melhores configurações de acordo com esse critério.

A tabela 5.1 lista as configurações obtidas. A primeira linha mostra a configuração usada no MD6. Com essa configuração, após 102 passos de execução da função de compressão cada uma das últimas 16 palavras calculadas dependem de todas as 89 palavras da entrada.

Tabela 5.1: *Levantamento de outras configurações*

Passos	t_1	t_2	t_3	t_4
102	18	21	31	67
103	18	22	42	78
103	19	22	26	59
104	18	21	41	79
104	18	21	44	67
104	19	22	45	75
106	18	19	24	63
106	18	20	27	67
106	18	21	27	67
106	18	21	43	83
106	18	21	44	84
106	18	21	61	84
106	18	22	30	74
106	18	22	42	67
106	20	22	35	66

5.2 Escrevendo um software para cálculo da quantidade mínima de portas AND ativas

O software para cálculo da quantidade mínima de portas AND ativas em um determinado número de rodadas não foi disponibilizado pelo grupo de trabalho do MD6. Em [**?**] é apresentado um algoritmo do procedimento geral utilizado, sem detalhes do cálculo.

Em [**?**] consta também o caminho diferencial de padrões de peso encontrado na busca do limite inferior de portas AND ativas em 15 rodadas: $D_{54} = 1, D_{71} = 2, D_{143} = 2, D_{232} = 2$. É explicado também o cálculo das 26 portas AND ativas deste caminho: $1 + 2 + 2 = 7$. Cada uma dessas 7 diferenças podem ativar até 4 portas AND, portanto teríamos 28 portas AND ativas. Mas de acordo com um dos critérios para não se contar portas AND a mais, só se deve contar as portas AND que tem ambas as entradas dentro do limite do número de rodadas onde está sendo feita a busca. Quando a diferença da posição 232 é entrada de uma porta AND na posição t_4, a posição t_3, que é a outra entrada da porta AND, estará além do limite das 15 rodadas. 15 rodadas são 240 passos, a posição 232 está a 8 passos do limite e a posição t_3 está a 36 passos da posição t_4. Então não é contada uma das vezes que as diferenças do passo 232 é entrada de uma porta AND, por isso

se considera $28 - 2 = 26$ portas AND ativas em 15 rodadas.

Comecei implementando um programa utilizando o algoritmo de busca apresentado em [**?**], mas fazendo o cálculo de uma maneira ligeiramente diferente, porque, olhando o resultado apresentado (o caminho diferencial de padrões de peso e o número total de portas AND ativas desse caminho), não havia entendido o método utilizado no cálculo. Foram duas diferenças principais:

1. Fiz o cálculo das portas AND *ativadas* durante a execução dos passos. O programa do grupo de trabalho do MD6 calcula as portas AND *criadas* durante a execução dos passos, ou seja, uma diferença que ative uma porta AND durante a execução não será contada se a diferença está na entrada, e será contada uma diferença criada na execução dos passos, mas que não será ativada durante a execução dos passos (como é o caso do padrão de peso do índice 232 do caminho apresentado).

2. Implementei o programa para fazer a busca a partir da entrada, antes do cálculo do primeiro passo. Considerei que cada índice da entrada poderia assumir qualquer padrão de peso e assim, quando a busca chegava no índice do primeiro passo já era possível calcular limites para o padrão de peso daquele índice. Já o programa utilizado pelo grupo de trabalho do MD6 começa a busca a partir do primeiro passo considerando que na entrada pode haver qualquer padrão de peso. E se na entrada pode have qualquer padrão de peso então não existe limite para o padrão de peso nos primeiros 89 passos.

O primeiro item implica no segundo, já que para contar as portas AND que serão ativadas durante a execução dos passos é necessário saber quais são os padrões de peso da entrada.

O problema de implementar o programa para fazer a busca a partir da entrada é que ele ganha 89 níveis de profundidade em relação ao programa que começa a execução a partir do primeiro passo, sem se importar com a entrada, o que faz uma enorme diferença no tempo de execução do programa.

O primeiro programa implementado começou a ter dificuldade com apenas 3 rodadas.

Então comecei a modificar o algoritmo de busca tentando obter resultados relevantes mais rapidamente, mantendo a metodologia de calcular as portas AND ativadas durante a execução dos passos. Foram aproximadamente 5 versões de programa mudando o algoritmo de busca.

O algoritmo original de busca apresentado em [**?**] começa a busca para 1 rodada e vai incrementando o número de rodadas. Em cada busca com um determinado número de rodadas ele começa procurando caminhos com 0 portas AND ativas e vai incrementando esse número até encontrar um caminho válido que não ultrapasse esse valor, sendo esse então o valor mínimo de portas AND ativas para aquele número de rodadas.

Tentei aproveitar melhor o processamento modificando o programa para fazer a busca simultaneamente em todas as quantidades de rodadas que se desejava obter o resultado, no caso de 1 a 15. Assim as somas feitas para um determinado número de rodadas seriam úteis no cálculo de um número maior de rodadas. Além disso, em vez de fazer a busca incrementando o número máximo de portas AND ativas que se deseja encontrar, o que faz o programa recomeçar a busca do início para cada novo valor máximo, fiz a busca procurar valores menores do que o menor valor encontrado até então, assim em apenas uma varredura já se conseguiria todos os valores mínimos de todas as diferentes quantidades de rodadas.

Inicialmente parecia que a estratégia seria interessante e que a configuração que eu estava testando (a segunda configuração da tabela 5.1) se comportava de modo bem diferente, parecendo que com ela o número mínimo de portas AND ativas seria bem maior. Enquanto com a configuração original o programa rapidamente encontrava caminhos com 33 portas AND ativas, com a configuração em teste o programa não conseguia achar caminhos com menos do que 106 portas AND ativas.

Mas após algumas semanas os resultados das duas configurações ficaram parecidos e não conclusivos, pois o programa ainda estava rodando.

A tabela 5.2 mostra os valores mínimos de portas AND ativas encontrados por esta versão do programa.

Tabela 5.2: *Número mínimo de portas AND ativas encontradas pela versão do programa com algoritmo de busca modificado e metodologia de cálculo que conta as portas AND ativadas na execução dos passos. Resultado não conclusivo.*

s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Primeira config. da tab. 5.1	0	0	1	1	2	4	5	5	10	12	18	22	26	29	33
Segunda config. da tab. 5.1	0	0	1	2	3	4	5	9	11	15	19	22	26	29	33

Percebi então que com esta estratégia é muito difícil conseguir um resultado útil. O algoritmo apresentado em [**?**] apresenta rapidamente algum resultado que pode ser usado na comparação entre diferentes configurações, porque ele já fornece o resultado final nos números de rodadas menores. Entendi também a metodologia de cálculo das portas AND ativas usada pelo grupo de trabalho do MD6, contando as portas AND criadas na execução dos passos em vez de contar as ativadas. E finalmente entendi como ele fez a busca começando pelo primeiro passo do cálculo, e não pelos valores da entrada.

Com essa compreensão e o programa modificado de acordo recomecei os testes. Consegui então chegar exatamente aos mesmos resultados apresentado na tabela 4.1.

Testando então as diferentes configurações, chega-se aos mesmos resultados. Curiosamente chega-se também sempre ao mesmo caminho diferencial de padrões de peso para $s = 15$: $D_{54} = 1, D_{71} = 2, D_{143} = 2, D_{232} = 2$

A variação das posições t_1, t_2, t_3 e t_4 parece não influir tanto no número de portas AND ativas quando se supunha.

Capítulo 6

Conclusão

As funções de hash têm grande aplicação em criptografia e são usadas largamente em um amplo conjunto de importantes aplicações, incluindo: autenticação e integridade de mensagem, assinatura digital e inúmeras outras. A segurança destas aplicações frequentemente depende diretamente das propriedades de segurança da função de hash que está por trás. Se a função de hash falhar em ser tão segura quanto se acreditava que ela seria, então a segurança da aplicação também estará comprometida. Portanto, existe um forte interesse em se provar da forma mais rigorosa possível que uma função de hash de fato tem as propriedades de segurança desejadas e que é resistente a uma variedade de ataques.

O MD6 demonstra ser seguro contra várias formas de ataque conhecidas. É relativamente simples, altamente paralelizável e razoavelmente eficiente.

No entanto algumas propriedades de segurança do MD6 ainda carecem de provas, ou de provas mais rigorosas. Não está provado se o modo de operação do MD6 preserva a resistência à segunda pré-imagem da função de compressão. Também não está provado de forma rigorosa que ele preserva a imprevisibilidade.

A eficiência do MD6 é excelente em sistemas com múltiplas unidades de processamento, mas nas plataformas de referência da competição do NIST para escolha do SHA-3 ela não é suficiente para torná-la competitiva.

Além do alerta feito por Ron Rivest de que é extremamente importante que o algoritmo de SHA-3 que vier a ser escolhido seja demonstravelmente resistente a ataques diferenciais, ele sugere também que o SHA-3 deveria ter um modo de operação em árvore para que seja adaptável para uso em sistemas com múltiplas unidades de processamento.

Testamos outras configurações para as posições t_1, t_2, t_3 e t_4 utilizadas na função de compressão, na esperança de encontrar uma configuração que permitisse uma redução no número de rodadas sem afetar a demonstrabilidade da segurança do MD6 contra ataques diferenciais padrão, o que tornaria a função mais eficiente, com um desempenho melhor em termos de velocidade.

A princípio todas as configurações testadas parecem se comportar da mesma forma, e portanto até o momento não foi possível obter um incremento na eficiência do MD6.

Seria interessante buscar entender o mecanismo que está por trás da obtenção dos mesmos resultados para qualquer configuração testada, talvez esta compreensão ajude a descobrir o que poderia ser mudado para melhorar a eficiência da função.

Referências Bibliográficas

[1] A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.

[2] Stinson, Douglas R. - Cryptography Theory and Practice Third Edition - 2006

[3] Schneier, Bruce - Applied Cryptography Second Edition: protocols, algorithms, and source code in C.

[4] Rivest, Ronald L. - The MD6 hash function, a proposal to NIST for SHA-3 - http://groups.csail.mit.edu/cis/md6/docs/2009-04-15-md6-report.pdf

[5] Crutchfield, Christopher Yale - Security Proofs for the MD6 Hash Function - http://groups.csail.mit.edu/cis/md6/docs/2008-06-crutchfield_ms_thesis.pdf

[6] NIST - Cryptographic Hash Algorithm Competition - http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

[7] Rivest, Ronald L. - The MD6 hash function (Crypto â08 Slides) - http://groups.csail.mit.edu/cis/md6/docs/2008-08-20-Rivest-TheMD6HashFunction.ppt

[8] Heys, Howard M. - A Tutorial on Linear and Differential Cryptanalysis - Electrical and Computer Engineering / Faculty of Engineering and Applied Science / Memorial University of Newfoundland / St. John's, NF, Canada A1B 3X5 - http://www.engr.mun.ca/ howard/PAPERS/ldc_tutorial.pdf