UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

UMA AVALIAÇÃO CRÍTICA SOBRE OS ATAQUES ÀS FUNÇÕES MD5 E SHA1

TRABALHO DE GRADUAÇÃO

Aluno: Francisco de Assis Mesquita Valadares (famv@cin.ufpe.br)
Orientador: Prof. Ruy de Queiroz Guerra (ruy@cin.ufpe.br)

27 de Março de 2006

“O Binômio de Newton é tão belo como a Vênus de Milo.
O que há é pouca gente para dar por isso...”
(Fernando Pessoa - Poesias de Álvaro Campos)

“Não deis aos cães o que é santo, nem lanceis aos porcos as vossas perolas, para não acontecer que as calquem aos pés e, voltando-se, vos despedacem.”
(Jesus Cristo)

“People demand freedom of speech to make up for the freedom of thought which they avoid”
(Soren Aabye Kierkegaard)

Agradecimentos

Em primeiro lugar a Deus, por dar essa oportunidade de realizar esse trabalho com coragem e dedicação.

A meus familiares, por me aturarem todo santo dia.

Ao meu orientador e professor Ruy de Queiroz, pelo seu incentivo ao meu desenvolvimento na área de criptografia.

Ao colaborador longínquo Mads Rasmussen, por ter me mostrado as partes fundamentais do assunto correlato a este trabalho, a prova resumida da construção MD e também pela sua paciência.

Ao meu ex-professor e amigo, André Paegle por ter me ensinado a gostar de matemática. O *ex* é apenas uma questão temporal, pois ele sempre será meu mestre.

A Todos aqueles colegas perdidos no tempo e espaço e aos grandes colegas que persistem em minha memória como Arlei Calazans(Magão), Bruno Bourbon(Gato-Mestre) e em especial a Jarbas Jacome(Jabah) pela *malandragem* e Alexandre Sarmento(Asas) o único com capacidade de entender a *força da idéia* de primeira.

A Liliane por quem tenho bastante afeição.

Resumo

Funções hash criptográficas são funções bastante difundidas no contexto das aplicações computacionais. Por se tratar de uma ferramenta de segurança e por possuir propriedades como resistência a colisões e unidirecionalidade, propriedades construídas a partir de premissas matemáticas e computacionais completamente não-triviais, é sempre interessante quando tais propriedades são transpostas. Um dos trabalhos mais importantes nessa linha nos últimos anos foi apresentado no CRYPTO em Agosto de 2004 por uma equipe de chineses para várias funções hash criptográficas, dentre elas o MD5 e o SHA-1. Apesar deles apenas mostrarem entradas que geravam colisões, as conseqüências desse resultado desdobram-se até os dias atuais. Esse trabalho foca o quanto possível no design desses dois algoritmos e nos resultados obtidos pelos pesquisadores com o intuito de fomentar uma análise crítica em relação ao uso prático dessas duas funções hash criptográficas.

Palavras-chave: Funções Hash Criptográficas, paradigma Merkle-Damgard, Cifras de Bloco, Integridade, Criptoanálise, Colisões.

Abstract

Criptographic Hash Functions are well known in the context of computer applications.
Due to its usage as a security tool and to having properties such as collision resistance and one-wayness, being these properties constructed from non-trivial mathematical and computational primitives, it's always important that these properties are transposed. One of the most important works in this field for the past few years was presented at the CRYPTO in August, 2004 by a chinese team for many cryptographic hash functions, including MD5 and SHA-1. Even though they showed inputs that generated collisions, consequences of these results have furthered researchers until today. This article focuses as much as possible on the design of these two algorithms and on the results obtained by researchers with the objective of doing a crítical analysis about the practical usage of these two cryptographic hash functions.

Key Words: Criptographic Hash Functions, Merkle-Damgard paradigm, Block Cipher, Integrity, Cryptoanálise, Collisions.

Sumário

Lista de Imagens

Lista de Tabelas

1 - Introdução

É inegável o papel desempenhado pela tecnologia da informação no mundo contemporâneo. Milhares e milhares de informações de variadas procedências e interesses trafegam via ondas eletromagnéticas pelo ar, via fótons por fibra ótica e via cabos coaxiais, sendo monitoradas e propagadas por antenas, repetidores, servidores, roteadores, dando suporte às mais diversas e complexas redes de comunicação, a citar, por exemplo, a Internet. É neste cenário vasto que surgem vários desafios, criados a partir de serviços e interações entre os usuários envolvidos dentro da rede. Com advento da Internet, veio a necessidade de se criarem mecanismos de garantia de integridade de informação digital. Apesar do meio ser novo, a necessidade é algo bem antigo e bem desenvolvido em outros ambientes. Por exemplo, documentos feitos em papel já são analisados com maestria pelos documentoscopistas formados pelas ciências forenses.

Uma outra possibilidade seria o uso da criptografia clássica. Apesar da criptografia ser uma ciência tão antiga quanto as primeiras civilizações humanas, as técnicas clássicas criptográficas não são capazes de dar suporte à integridade da informação digital de forma eficiente. Seria necessário o desenvolvimento da *criptografia moderna* para dar suporte á construção das *funções hash criptográficas*. Tais funções em tese seriam capazes de fornecer integridade de forma eficiente, ideal para o meio digital. Essa característica estaria fundamentada em propriedades matemáticas e computacionais.

Basicamente, esta função recebe uma informação digital(mensagem, arquivo, etc) de um tamanho arbitrário e retorna uma saída(string) de tamanho fixo. As funções mais usadas atualmente pelas aplicações são a *Message Digest-5* ou *MD5* [Rivest92] e a *Standart Hash Algorithm* ou *SHA-1*[FIPS180-2].

Recentemente, a comunidade cientifica da área de segurança vem assistindo apreensiva ao desenvolvimento de ataques a essas funções. Além do mais, não existe ainda um consenso comum de que modificação ou algoritmo usar no lugar do MD5 e SHA-1. Por exemplo, pesquisadores de renome na área como Bruce Schneier e Paul Hoffman[RFC4270] discordam no que se deve fazer neste sentindo. Enquanto que o primeiro recomenda a migração para funções hash criptográficas consideradas ainda fortes, o segundo recomenda o contrário.

Esse trabalho visa um estudo em detalhes dessas duas funções em nível de teoria, algoritmo e forma de uso mais freqüente, para compreender o por quê da possibilidade de tais ataques e suas conseqüências, objetivando uma análise crítica de ambas no intuito de verificar sua usabilidade na prática. Não é do interesse deste trabalho analisar em detalhes o algoritmo desenvolvido para realizar o ataque em si.

1.1 - Objetivos

Objetivo principal:

- Entender até que ponto os ataques ao MD5 e ao SHA-1 podem comprometer na prática, a segurança das aplicações mais corriqueiras que fazem uso deles.

Objetivos Secundários:

- Avaliar algumas propostas feitas pelos pesquisadores e órgãos de outros países sobre possíveis modificações/substituições dos algoritmos MD5 e SHA-1.

- Tentar descobrir a melhor proposta e se possível, sugerir uma proposta.

1.2 - Quem deverá ler este trabalho

A complexidade do tema é inerente a área de criptografia, por isso houve uma tentativa de facilitar para o leitor ao máximo possível o desenvolvimento deste trabalho. Alguns conceitos foram explicados de uma forma mais simples que a definição real, outros devem ser melhor estudados a partir das referências, tudo isso não deixando de citar/explicar o que era essencial para a compreensão deste trabalho de graduação.

Assume-se que o leitor deve ter noções de elementos básicos do "mundo" da ciência da computação como lógica booleana, operadores binários, processamento, bits, bytes, strings, cadeias binárias, expressões regulares, algoritmos, complexidade, criptografia, etc.

1.3 - Como se deverá ler este trabalho

Este trabalho está recheado de definições e referências. Elas servirão tanto para orientar o leitor no decorrer da leitura como para encontrar informações mais detalhadas de algum conceito. No tocante às definições, quando elas forem usadas pela primeira vez, serão identificadas em itálico. Exemplo: *recurso aceitável.* Caso o termo seja usado novamente, ele não mais virá em itálico, mas a definição dele ainda vale. Por isso é recomendável a leitura do início ao fim do trabalho sem transpassar capítulos e/ou seções. Caso haja uso de uma definição sem ela ter sido explicada, o leitor deverá olhar a sua respectiva referência para sanar sua dúvida.

Os termos *string(s), substring(s),* sem especificar o tipo referem-se a representações de caracteres usadas em um computador como ascii, unicode, etc. Caso contrário o tipo será especificado. Exemplo: *string binária.*

Ocorrerá em alguns casos a tipificação de uma definição já dada em algum instante anterior. Por exemplo, *chave* foi definida como string, mas aparecerá no texto chave binária. Apesar disso, o termo isolado *chave* ainda é uma string.

Além de tudo isso, algumas funções simples definidas nos apêndices serão usadas, deve-se ler sua definição no apêndice para saber o seu propósito. Exemplo: *TamBin().*

Mais uma vez, é recomendável a leitura do início ao fim do trabalho sem transpassar capítulos e/ou seções devido ao encadeamento lógico dos conceitos e das explicações.

1.4 - Organização do Trabalho

O trabalho está organizado em 10 capítulos: O capítulo 2 apresenta as definições fundamentais; o capítulo 3 define e taxonomiza as funções hash criptográficas; o capítulo 4 mostra de forma formal os interesses de um adversário em atacar as funções hash criptográficas do tipo CDM e CAM; o capítulo 5 disseca sobre as funções de compressão sob o paradigma Merkle-Damgard; o capítulo 6 mostra os parâmetros de ataque às funções hash criptográficas do tipo CDM e CAM além de discutir sobre provas de segurança e paradigmas de construção de funções hash criptográficas; o capítulo 7 mostra em detalhes os algoritmos MD5, SHA-1, HMAC-MD5 e HMAC –SHA-1, além de explicar e mostrar mais alguns outros detalhes desses algoritmos; o capítulo 8 mostra o estado da arte aos ataques aos algoritmos MD5 e SHA-1 além de exibir a prova de conceito desses ataques; o capítulo 9 faz uma análise crítica dos ataques; o capítulo 10 refere-se a conclusão; as Referências Bibliográficas mostra as referências que embasaram a construção desse trabalho e o Apêndice contém algumas funções e dois vetores que geram colisão no MD5.

2 - Definições Iniciais

2. 1 - Limites Computacionais

Define-se como *tempo computacional aceitável* pela abordagem da teoria da complexidade computacional, o tempo realizado pelo computador para realizar alguma operação algorítmica cujo número de passos é superiormente limitado assintoticamente por um polinômio que é função do tamanho da entrada do algoritmo. Isso é bem intuitivo, haja vista que um tempo limitado exponencialmente cresceria absurdamente e seria impraticável(coisa de milhares ou milhões de anos para ser efetuada por completo). Por exemplo, sendo *p* a entrada do algoritmo(representada como uma string),

x = TamBin((String2Bin(*p*))) o seu tamanho em bits(vide apêndice) então o algoritmo realiza *poly(x*) (poly é um polinômio qualquer não especificado) passos, o seu tempo de execução será *c*∗*poly(x*), onde *c* é uma constante indicando o tempo de um passo realizado pelo processador. Tendo em vista *c*∗*poly(x*), o algoritmo obviamente é limitado superiormente por *poly(x*).

Observe que a descrição em linguagem de máquina do algoritmo é polinomial também, pois o número de instruções de máquina por passo de um algoritmo é em geral muito pequena(em torno de 200). Caso ele precise, por exemplo, de memória, esse recurso deverá ser limitado superiormente por *poly(x*) também para ser um *recurso aceitável*. Um *adversário prático* que queira "quebrar" qualquer *esquema*(neste trabalho, entende-se esquema como qualquer construção algorítmica voltada a garantir requisitos de segurança criptográficos de uma aplicação) de forma prática, deverá possuir um algoritmo de tempo computacional aceitável e de recurso aceitável. Como já foi falado, não faz sentido pensar em um adversário que vá passar um tempo gigantesco para atacar um esquema. Ele não seria prático. A classe de algoritmos mais poderosa em termos de recursos nessas condições são os *algoritmos polinomiais probabilísticos* (conhecidos como *BPP, Bounded Error Probabilistic Polynomial Time* [Zoo]). Essa classe de algoritmos se dá ao luxo de possuir um gerador de números aleatórios e de usá-los caso

seja preciso. Além do mais, suas respostas podem ser falso-positivos e/ou falso-negativos ou ser correta com 100% de chance de acerto e 0% de erro, formando-se assim subclasses contidas na classe maior BPP(*ZPP, RP* etc...). É lógico que o número de vezes em que esta classe utiliza o gerador e o tempo que o gerador leva para gerar tais números também são limitados superiormente por *poly(x).*

A abordagem conhecida como *provable security*[ProvSec] utiliza-se desses conceitos mais o conceito de redução. *Redução* nada mais é que provar que quebrar um esquema criptográfico é equivalente a resolver um *problema difícil*(problema onde não se conhece um algoritmo de tempo e recurso aceitáveis para solucioná-lo*)* mostrando assim a dificuldade também de se quebrar o esquema para todos os algoritmos BPP(ou todos os adversários práticos).

2. 2 - Parametrização dos Recursos do Adversário

A abordagem mostrada anteriormente é muito boa para provar segurança de esquemas, todavia, ela não é suficiente quando se quer provar esquemas que não foram construídos a partir de problemas difíceis e quando se precisa de uma noção mais exata, mais concreta, um número que parametrize o algoritmo ou o esquema, de todos ou boa parte dos recursos que o adversário necessitará para quebrar o esquema em questão, não fornecido pela própria natureza assintotica da abordagem reducionista. Uma abordagem surgida para tentar suprimir essa dificuldade é conhecida como *concrete security*[ConcSec] . O adversário é o mesmo(algoritmo BPP) mas surgem daí outros conceitos como *famílias de funções*, *black-box model* e *oráculo*, formalismos auxiliadores na medida dos recursos do adversário(vide seções seguintes deste capítulo).

Esse tipo de parametrização fornecido por essa filosofia é bem mais fácil e intuitivo de ser trabalhado. Por exemplo, se para realizar um determinado tipo de ataque, um adversário precise rodar um algoritmo 2^{100} vezes(quantidade obtida mediante essa abordagem), mesmo se cada execução completa desse algoritmo levasse 2^{-20} segundos(um número praticamente impossível de ser obtido para um adversário prático, ou seja, um algoritmo da classe BPP), o tempo para realização do ataque seria maior que 10^{15} anos, este não seria um ataque prático nem seria *computacionalmente viável*. Por isso 2^{100} seria um valor muito *alto em termos práticos*. Uma outra interpretação seria que a probabilidade do adversário quebrar o esquema em uma tentativa seria 2^{-100}, um valor

extremamente *baixo em termos práticos*(menor que 1 e extremamente pequeno). O *valor mediano em termos práticos* seria um valor acima de 1 e menor que um valor alto em termos práticos.

Esta abordagem será a adotada na maior parte deste trabalho e quando for necessário falar de adversários que tentam quebrar coisas, implicitamente, está se falando de algoritmos analisados a partir dessa perspectiva.

2. 3 - Famílias de Funções

Família de funções é o conjunto de funções formadas pelo mapeamento $F: K \times D \rightarrow I$, com K, D e I conjuntos finitos pertencentes a {0,1}* - {ε}. Esse K é o indexador da das funções F. Por exemplo, se K = {0,1}, D = {0,1} e I = {0,1}, e definirmos F como:

$$F(k,d) = k \text{ xor } (d \,||\, 1),\ k \text{ pertencente a } K \text{ e } d \text{ pertencente a } D$$

Com o operador “xor” representando a operação binária “OU-EXCLUSIVO”.

Assim teríamos que a família de funções de F seria $\{f_1, f_2\}$ com

$$F(0,d) = f_1(d) = d \,||\, 1 \text{ e } F(1,d) = f_2(d) = 1 \text{ xor } d \,||\, 1.$$

Com o operador “||” representando a operação *concatenação.* Sendo assim, f_1 e f_2 seriam *instâncias* da família de F.

Uma *Família de Permutações* é uma família de funções onde todas as suas instâncias são funções bijetivas. Aqui e no resto deste trabalho, a palavra permutação segue exatamente a idéia matemática.

2. 4 - Famílias de Funções e de Permutações Randômicas

Suponha que nós não sabemos nada sobre a família F, e que uma instância f dela nos é fornecida, mas nós não temos acesso ao algoritmo de f, apenas poderemos usá-la para cálculo de entrada-saída, ou seja, fornecendo-lhe uma entrada *x*, obtemos f(*x*). Esse modelo é conhecido como *modelo da caixa preta(black-box model*) , pois assim como numa caixa preta, o que há dentro é desconhecido. O algoritmo escrito em linguagem natural abaixo especifica melhor o ocorrido:

1º- Sorteia-se uma instância de F, ou seja, sorteia-se um elemento pertencente a K aleatoriamente, formando uma instância sem que nós saibamos quem é essa instância;

2º- Fornece-se a “interface” de f, ou seja, como uma caixa preta;

3º- Fornecemos a f tantas entradas quanto quisermos e obtemos as respostas dessas entradas;

4º- Fim.

O detalhe aqui é que para uma mesma entrada *x* fornecida, a saída é a mesma, pois f é escolhido apenas uma vez. O sorteio e o acesso de f nos é desconhecido, funcionando também como uma espécie de *oráculo* de F. A idéia aqui é que se pode consultar o oráculo quantas vezes quiser, pois ele sabe tudo sobre F. Quando se diz que se tem apenas acesso ao *oráculo de F*, supõe-se que os passos 1 e 2 são realizados pelo oráculo e cabe a quem pergunta apenas consultá-lo como no passo 3. Na verdade, é a mesma coisa apenas dita de uma forma diferente mas muito usada pela comunidade acadêmica da área de criptografia.

Uma *família de funções randômicas* são funções obtidas de acordo com o procedimento acima e que dada uma instância de F, essa instância deve possuir propriedades probabilísticas específicas[Goldreich et al]. Assim, se F preenche esses pré-requisitos, F seria tal família e suas instâncias seriam *funções randômicas*. O nome randômico não tem relação com o comportamento da entrada-saída e sim a forma como a instância é sorteada, até porque, por exemplo, uma função constante poderia pertencer a F. Embora as propriedades probabilísticas tenham relação com o comportamento entrada-saída da instância, elas estão mais para garantir que qualquer instância possui a mesma probabilidade de ser sorteada, do que garantir o comportamento aleatório de uma instância de F dada. Um *oráculo randômico* seria um oráculo de uma família de funções randômicas.

A partir desses conceitos, uma *família de permutações randômicas* seria uma família de funções randômicas onde as instâncias seriam bijeções e seriam *permutações randômicas.* Tal família possuiria outras propriedades probabilísticas parecidas[LubyRackoff].

2. 5 - Funções e Permutações Pseudo-Randômicas

Seja um adversário(algoritmo) *M*, com o intuito de dizer se uma dada instância de uma família de funções pertence ou não à família randômica. Ele retorna 1 se a instância não pertence, e retorna 0, caso contrário. Seja *Q*, uma família de funções publicamente conhecida que não é a família randômica. Suponha que é dado a *M* um oráculo, mas *M* não tem como saber se esse oráculo é o oráculo de *Q* ou o oráculo randômico. Seja b pertencente a {0,1} e *q* o número de consultas de *M* ao oráculo.

Considere estes dois experimentos para ilustrar a situação:

Experimento 1

1º- o oráculo de *Q* pega uma instância de *Q*, chamada *f*;

2º- *M* faz *q* consultas ao oráculo, portanto fornece entradas a *f* e obtém as saídas;

3º- *M* retorna *b*;

4º- Fim.

Experimento 2

1º- o oráculo randômico gera uma instância, chamada *f*;

2º- *M* faz *q* consultas ao oráculo, portanto fornece entradas a *f* e obtém as saídas;

3º- *M* retorna *b*;

4º- Fim.

Defina $Prob[Exp_i(M) =1]$ com *i* pertencente a {1,2}, a probabilidade de *M* retornar 1 no experimento *i*, probabilidade tomada sobre o sorteio da instância pelo oráculo e sobre a aleatoriedade usada por *M* internamente.

Se *M* está fazendo seu trabalho correto ele deverá retornar 1 no primeiro caso e 0 no segundo caso ou retornar 1 no primeiro caso e retornar 1 no segundo caso a menor quantidade de vezes possível.

Define-se como *vantagem de M em relação a Q*, $Vant_Q(M)$, a medida deste trabalho, quantitativamente como:

$$Vant_Q(M) = Prob[Exp_1(M) =1] - Prob[Exp_2(M) =1]$$

Sendo assim a família *Q* é *indistinguível* de uma família randômica se $Vant_Q(M)$ é baixo em termos práticos para todos os *M* possíveis e *q* é um valor mediano em termos práticos. Portanto a eficiência de *M* em quebrar *Q* é medida por $Vant_Q(M)$ e *q*. Se a família *Q* é indistinguível de uma família randômica, suas instâncias são *funções pseudo-randômicas*. Se essas funções são bijeções, elas são *permutações pseudo-randômicas.* Em qualquer dos dois casos, uma instância de *Q* seria "parecida" com uma instância de uma família randômica, portanto as funções seriam *seguras*, já que o adversário não saberá qual instância foi sorteada, nem observando o comportamento da entrada-saída. Detalhes sobre esses conceitos podem ser consultados em [Modern1].

2. 6 - Paradoxo do Aniversário - Colisão numa Função Randômica

Uma *colisão numa função randômica* f advinda da família randômica $F: K \times D \rightarrow I$, são duas entradas aleatórias e distintas x_1 e x_2 tais que $f(x_1) = f(x_2)$. Pela própria natureza da família randômica(propriedades), era de se esperar que achar tais valores no domínio de f, com f como black-box, para cada dois valores sorteados e verificados, deveria haver |D| verificações(vide apêndice para definição do operador | |). Surpreendentemente, isto não ocorre. Para se achar uma colisão com grande probabilidade(quase 100%), basta realizar parte inteira de $|D|^{½}$ mais *c* verificações, onde esse *c* é um natural pequeno(em torno de 6). O *paradoxo do aniversário* recebe esse nome porque para que numa turma haja duas pessoas que fazem aniversário no mesmo dia com probabilidade maior que 50% basta que a turma possua $365^{½} \cong 20$ pessoas o que é extremamente contra-intuitivo.

Este resultado possui uma importância fundamental. Por exemplo, um adversário que queira distinguir uma família de permutações $P: K \times D \rightarrow I$ de uma família randômica $F: K \times D \rightarrow I$, basta sortear aleatoriamente de D, $|D|^{½} + 6$ duplas e fazer um total de $2*(|D|^{½} + 6)$ consultas ao oráculo. Este número de consultas é o suficiente para garantir que se for um oráculo randômico, uma colisão será achada. Além do mais, serve como parâmetro de segurança para as funções pseudo-randômicas. Tendo em vista que as funções pseudo-randômicas devem ser "parecidas" com funções randômicas, deve-se realizar o mesmo número de consultas nelas para se achar uma colisão com alta probabilidade. Se o número de consultas for um valor mediano em termos práticos, um adversário prático teria condições de saber se a instância é uma função randômica ou pseudo-randômica. Ainda, para qualquer função matemática de domínio e contradomínio

finito de tamanho *w* e que não seja bijetiva, com acesso a ela apenas como um black-box, basta gerar no máximo $w^{½} + 6$ duplas e fazer um total de $2*(w^{½} + 6)$ consultas ao oráculo para saber se ela é uma função randômica ou não. Mais utilidades desse resultado será visto no decorrer desse trabalho. A prova sobre este resultado pode ser vista em [Modern2].

3 - Definição Função Hash

Funções Hash são objetos muitos importantes no âmbito das aplicações em computação. Em geral, são funções que recebem uma string de tamanho arbitrário(para propósitos práticos é arbitrário, pois na verdade o tamanho da entrada está limitado por um número extremamente grande, algo da ordem de 2^{64} bits) e geram como saída uma string de tamanho fixo. Essa saída é nomeada pela comunidade mais geralmente como *hash-code*, *message digest*, *hash value,* dependendo da aplicação da função hash. O seu processamento é conhecido como *hashing*. Este processamento deve ser computacionalmente viável(não levar tempo) de ser calculado para qualquer computador comum. Em português é comum chamar a saída de *resumo, valor de hash, valor da função hash* ou simplesmente *o hash* de uma string dada. Essas funções fazem parte de uma classe de funções conhecida como *funções de compactação*, já que em muitos casos o tamanho da saída é reduzido em relação ao tamanho da entrada.

As *funções hash criptográficas* são funções hash com propriedades adicionais interessantes dentro do contexto dos esquemas criptográficos. Comumente os esquemas usados nas aplicações têm como requisitos principais(abaixo, a string pode ser vista como alguma informação digital importante):

a) **Integridade**: garantir que uma string não foi alterada;

b) **Confidencialidade**: garantir que ninguém a não ser o dono da string seja capaz de interpretá-la corretamente;

c) **Acessibilidade**: garantir que somente usuários com permissão possam ter acesso a determinados recursos;

d) **Autenticação**: ter certeza da relação de posse, autoria ou identificação usuário-string;

e) **Não-Repúdio**: garantir que não será negada pelo usuário a relação de posse ou autoria de uma string.

Cada grupo de funções hash criptográficas possuem certas propriedades específicas que ajudam a preencher alguns desses requisitos. Sendo assim, dividem-se as funções hash criptográficas em dois grandes grupos de acordo com a presença de uma *chave*(uma string de tamanho fixo). E cada grupo subdivide-se em outros subgrupos cada um com suas características específicas. Antes de descrever esses agrupamentos, mister se faz descrever as propriedades fundamentais das funções hash criptográficas mais formalmente.

3. 1 - Definição Formal - Funções Hash Criptográficas

Dada uma função de hash criptográfica H de domínio $D = \{0,1\}^* - \{\varepsilon\}$ e contradomínio $I = \{0,1\}^m - \{\varepsilon\}$ com *x,w* pertencente a *D* e *y* pertencente a *I,* todos distintos entre si, ela deverá possuir algumas ou todas essas propriedades:

- Resistência à primeira pré-imagem ou Resistência a pré-imagem ou Unidirecionalidade

Dado *y* com *y* = H(*x*), não é computacionalmente viável achar *x*. Por isso o termo *unidirecionalidade,* pois é fácil de calcular, mas difícil de se calcular a inversa.

- Resistência à segunda pré-imagem

Dados *x,y* com *y* = H(*x*) não é computacionalmente viável achar um *w* tal que *y* = H(*w*). O detalhe aqui é que *x* e *y* já são conhecidos.

- Resistência a colisões

Não é computacionalmente viável, achar *x* e *w* quaisquer tais que H(x) = H(*w*). O detalhe aqui é que *x* e *w* não são desconhecidos. Diz-se, quando se acha tais *x* e *w,* que se encontrou uma *colisão* na função hash criptográfica. Essa propriedade é de extrema importância e aparecerá em maior parte deste trabalho.

Vale salientar que na definição formal de H, seu domínio pode possuir tamanho infinito o que não é verdade na prática.

Essas são propriedades fundamentais, pois quase todos os esquemas que se valem de funções hash criptográficas possuem essas três propriedades. Existem também três propriedades desejáveis em qualquer função hash criptográfica:

- Confusão e Difusão

Conceitos primeiramente desenvolvidos por Claude Shannon em 1949[Shannon] para construção de *cifras de bloco*[BCip] seguras. Cifras de bloco nada mais são que algoritmos criptográficos cujo propósito é garantir confidencialidade valendo-se de uma *chave secreta* conhecida apenas pelo seu dono. Esse algoritmo criptografa de bloco em bloco de bits, daí o nome.

Confusão tenta diminuir ao máximo a relação estatística entre a chave e a saída da função hash criptográfica, enquanto a difusão tenta diminuir ao máximo a relação estatística entre a entrada e a saída da função hash criptográfica. Uma característica marcante da difusão é caso haja alteração de 1 bit da entrada da função hash criptográfica todos os bits da saída podem ser alterados com probabilidade ½ de forma uniforme e independente entre si. Ou seja, em média temos ½ ∗|*l*| bits alterados na saída, pelo menos metade dos bits alterados na saída. Tal feito é conhecido como *efeito avalanche*. Esses conceitos ajudam a evitar ataques *lineares*[LinCrypt].

- Resistência a colisões próximas

Não é computacionalmente viável achar *x* e *w* quaisquer tais que H(x*)* e H(*w*) difiram somente em poucos bits. Tal conceito é uma condição necessária, mas não suficiente para evitar ataques *diferenciais*[DifCrypt].

- Unidirecionalidade parcial

Não é computacionalmente viável, dado uma substring de *x* e H(x*)* achar x.

Estas três últimas propriedades emergem a partir da presença na função hash criptográfica das três propriedades fundamentais. A seguir os dois grandes grupos de funções hash criptográficas:

3.1.1 – Funções hash criptográficas sem chave

Não precisam de uma chave no hashing. Este grupo se subdivide em:

a) *Código de Detecção de Modificação(CDM)*

Funções hash criptográficas com o intuito de garantir integridade, confiabilidade e acessibilidade. Elas se subdividem em:

(a1) *Funções hash criptográficas Unidirecionais*

Funções com apenas a propriedade de unidirecionalidade. Podem ser usadas para garantir confidencialidade e acessibilidade. Não serão vistas em detalhes neste presente trabalho por estar fora de escopo.

(a2) *Funções hash criptográficas Resistentes a Colisões*

Funções com todas as propriedades fundamentais das funções de hash criptográficas. Comumente usada para garantir integridade embora por causa de serem também unidirecionais, possam ser usadas em alguns casos para garantir confidencialidade e acessibilidade. Uma parte delas é desenvolvida de forma *customizada*, isto é, projetadas para possuírem bom desempenho em software.

Um exemplo prático de funções hash criptográficas customizadas são o *MD5* e o *SHA-1*, base deste trabalho. Uma descrição de ambos poderá ser vista no capítulo 7.

b) *Outras Aplicações*

Outras aplicações para funções hash criptográficas sem chave não serão abordadas neste presente trabalho por fugir ao escopo.

3.1.2 - Funções hash criptográficas com chave

Precisam de uma chave no hashing. Este grupo se subdivide em:

a) *Código de Autenticação de Mensagem(CAM)*

Funções hash criptográficas com o intuito de garantir integridade e autenticação.

Geralmente construídos a partir de um CDM resistente a colisões e uma chave *K*. Existem várias formas de fazer isso como, por exemplo, um CAM *C* aplicado sobre uma string *S*, a partir de uma função de hash criptográfica H do tipo CDM resistente a colisões:

$$C_K(S) = H(K \,||\, S)$$

Uma outra forma de obter um CAM seria construí-lo a partir de um CDM customizado resistente a colisões. CAM´s customizados são conhecidos como *HMAC´*s(Hash Message Authethication Code)[Hmac] e são construídos da seguinte forma:

Seja H uma função de hash criptográfica do tipo CDM customizado, *K* uma chave, *ipad* e *opad* strings constantes do mesmo tamanho que *K*. O HMAC de uma string *S* é definido abaixo:

$$HMAC_K(S) = H((K \text{ xor } opad) \;||\; H((K \text{ xor } ipad) \,||\, S))$$

Dois exemplos práticos de CAM customizados são conhecidos como *HMAC-MD5* e o *HMAC-SHA-1*, descritos nesse trabalho. Uma descrição de ambos poderá ser vista no capítulo 7.

b) *Outras Aplicações*

Outras aplicações para funções hash criptográficas com chave não serão abordadas neste presente trabalho por fugir ao escopo.

A seguir a figura que representa bem todos esses agrupamentos:

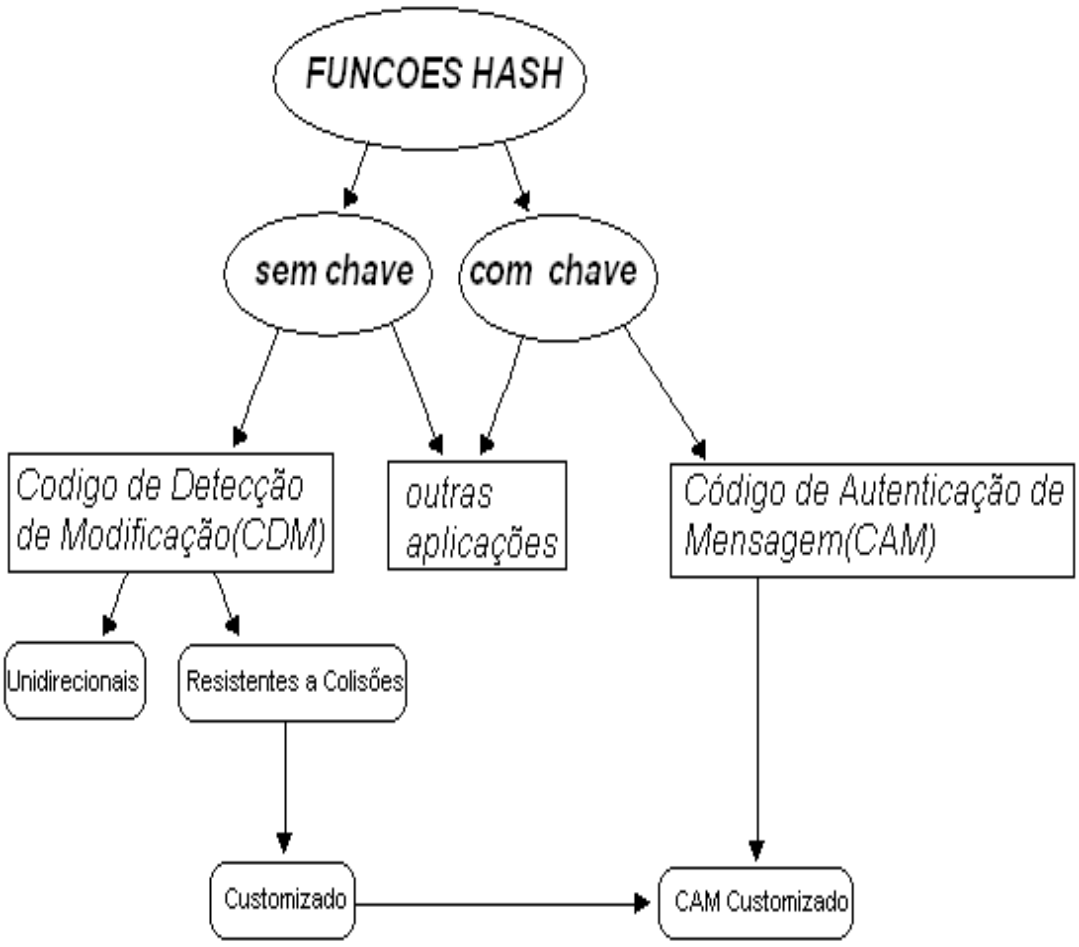


Figura 1 – Divisão das funções hash criptográficas.

4 - Interesses do Adversário

4. 1 - Quebrar um CDM Resistente a Colisões

Como já foi definido, um CDM resistente a colisões possui todas as três propriedades fundamentais e por causa disso, qualquer esquema montado a partir dele pode ser usado para garantir apenas integridade e/ou apenas confidencialidade e/ou apenas acessibilidade. Um exemplo prático desses dois últimos requisitos é o armazenamento dos hashes das senhas de *logon* nos servidores. Os servidores poderiam armazenar em seus bancos de dados o valor do hash da senha. No processo de logon do usuário, o servidor calcularia o hash da senha fornecida pelo usuário e verificaria se é igual à armazenada. Caso positivo ele teria acesso ao sistema. A propriedade fundamental aqui a ser explorada por um adversário que quisesse acesso ao sistema seria a unidirecionalidade ou a resistência à segunda pré-imagem. Caso ele tivesse acesso apenas ao hash da senha, ele poderia tentar inverter, descobrir a senha e ter a mesma acessibilidade que o dono da senha ou ele poderia tentar achar outra string cujo hash fosse igual ao hash da senha, tendo acesso ao sistema da mesma forma. Esse tipo de esquema, com apenas o valor do hash puro da senha, não é usado. Hoje em dia, é recomendável aplicar variações do mesmo, mas seu detalhamento foge ao escopo deste trabalho.

Por isso a análise neste tópico ater-se-á a dois esquemas usuais que fornecem integridade, autenticação e não-repúdio. Eles estão descritos da forma mais abstrata possível para facilitar a compreensão e porque não é interesse deste trabalho discuti-los a fundo. A seguir uma descrição de cada um.

4.1.1 - Uso de uma função H para garantir integridade

Seja uma string *S* e uma função de hash criptográfica H, nos moldes da definição formal dada na seção 3.1. Um usuário U_1 calcula H(*S*)(aqui chamado de RESUMO) e disponibiliza *S* e H(*S*) para um usuário U_2. O usuário U_2 então calcula o resumo de *S* obtendo um RESUMO' e verifica se RESUMO = RESUMO'. Caso sejam distintos, significa que o *S* usado por U_2 não é o mesmo *S* usado por U_1, indicando que o *S* original foi modificado. A figura abaixo ilustra bem a situação:

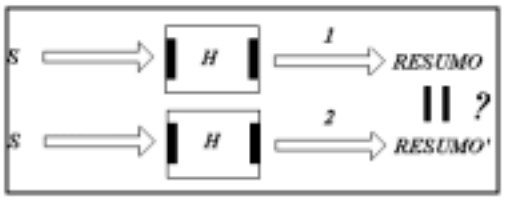


Figura 2 – Garantia de integridade.

Um adversário neste caso teria o intuito de quebrar a integridade de *S*. Para isso ele teria de quebrar a propriedade de ser resistente à segunda pré-imagem de H, pois dado o *S* e RESUMO ele seria capaz de produzir um *S'* tal que H(*S'*) = RESUMO, portanto a modificação *S* para *S'* não seria percebida por H e isso seria uma quebra de integridade de *S*.

Aparentemente a propriedade de H não ser resistente a colisões, não gera implicações de quebra de integridade de *S*, já que o adversário poderia sortear strings x_1 e x_2 tais que H(x_1) = H(x_2) com x_1=*S* ou x_2=*S,* mas isto é muito improvável haja vista o domínio de H é praticamente arbitrário. Dependendo da natureza da aplicação onde H é usado, nem sempre isso é verdade, como será mostrado mais adiante neste trabalho.

4.1.2 - Assinatura Digital com chaves públicas

Antes da descrição do esquema de assinatura digital faz se necessária uma breve descrição do que seria a *criptografia de chave pública* ou *criptografia assimétrica*[PkC].

A criptografia assimétrica envolve o uso de duas chaves distintas, uma pública e uma privada. A *chave privada* é mantida sempre em segredo e nunca deve ser divulgada. Em contrapartida, a chave pública não é secreta e pode ser livremente distribuída e compartilhada com qualquer usuário. Uma chave pública e sua correspondente chave privada são matematicamente relacionadas, mas não é computacionalmente aceitável derivar a chave privada a partir de sua chave pública associada. A prova disso é construída via redução(proven security) ao problema de fatorar o produto de dois primos grandes(por exemplo,cada primo possui em torno de 500 bits). A chave pública e sua chave privada são comumente denominadas de *par de chaves criptográficas*. Devido a sua relação matemática de inversão, uma *mensagem*(string) que foi *encriptada*(aplicando-se o algoritmo de criptografia assimétrico) com a chave pública pode ser *desencriptada*(aplicando-se o mesmo algoritmo de criptografia assimétrico só que com a chave inversa) com sua chave privada correspondente e uma mensagem que foi encriptada com a chave privada pode ser desencriptada com sua chave pública correspondente.

Uma assinatura digital de chaves públicas valer-se-ia da estrutura do esquema de criptografia assimétrica. Mais detalhadamente, o processo de assinatura e verificação da assinatura ocorreriam da seguinte forma:

Um usuário U_1 que quisesse assinar digitalmente uma string *S* usaria uma função de hash criptográfica H nos moldes da definição formal dada na seção 3.1 e obteria H(*S*) = RESUMO. Ao valor RESUMO seria então aplicado o algoritmo de criptografia assimétrico junto com a chave privada do usuário obtendo-se a assinatura digital do mesmo(vide figura abaixo).

Figura 3 – Modo de aplicar uma assinatura digital.

Logo após isso, U_1 enviaria *S* mais a assinatura digital para um usuário U_2. O usuário U_2 pegaria a chave pública de U_1, aplicaria o algoritmo de criptografia assimétrico junto com

a chave pública de U_1 e obteria um RESUMO'. O mesmo U_2 também pegaria S e aplicaria H(S). Se H(S) = RESUMO' ou melhor, RESUMO = RESUMO a assinatura é mesmo de U_1 e S não foi alterado, caso contrário ou S foi alterado ou a assinatura não é de U_1 (vide figura abaixo).

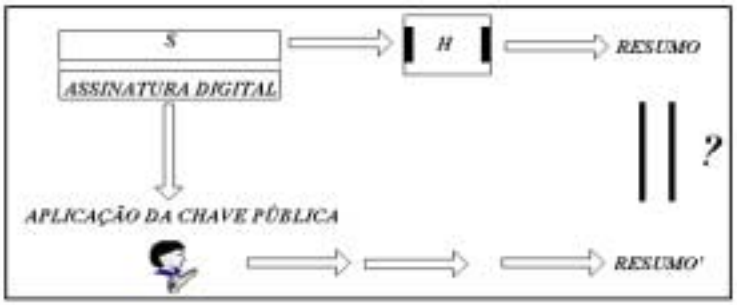


Figura 4 – Modo de verificar uma assinatura digital.

A assinatura digital está fornecendo integridade a *S*, autenticação do emissor(U_1) de *S*(via assinatura digital) e não-repúdio(U_1 não poderá negar que não foi ele quem enviou).

O interesse do adversário seria tentar quebrar essas propriedades. Ele poderá atacar o algoritmo de criptografia assimétrico, mas a análise deste aqui está fora de escopo. Outra possibilidade seria atacar H. Para isso, ele poderia atacar a integridade de *S* fornecida por H da mesma forma mostrada quando H é usado isoladamente para garantir integridade, ou seja, explorando a resistência à segunda pré-imagem ou a resistência a colisões. Isso se justifica haja vista que a assinatura é em cima do resultado de H e não diretamente de *S*.

4. 2 - INTERESSES DO ADVERSÁRIO EM QUEBRAR UM CAM

Um CAM deverá funcionar da seguinte forma:

Um usuário U_1 possui uma string *M* e uma chave secreta *K*. Esta chave secreta também é de conhecimento do usuário U_2. U_1 deseja enviar *M* por qualquer meio que faça chegar a U_2 (telefone, correio, internet etc...). U_1 então aplica o algoritmo do CAM junto com a chave *K* sobre *M* e obtém uma *Tag*(nada mais é uma string, resultado da aplicação do

algoritmo). U_1 envia pelo meio *M* anexado a *Tag* para U_2. U_2 então aplica o algoritmo verificador *VF* usando a mesma chave *K*, sobre *M* e *Tag*. U_2 obtém uma resposta Sim ou Não de *VF* acusando se a *Tag* enviada é mesmo relativa a *M*(vide figura 5). Uma resposta não quer dizer que ou *M* foi alterado ou a *Tag* foi alterada ou a chave usada não foi *K*. É fácil notar que esse esquema fornece integridade de *M* e autenticação de U_1, tendo em vista que U_2 sabe que apenas U_1 possui *K* e poderia gerar *Tag*. O interesse de um adversário aqui seria tentar quebrar esses dois requisitos atacando a resistência à segunda pré-imagem do CAM ou deduzindo a chave *K*, ambos para criar novos *M´*s válidos.

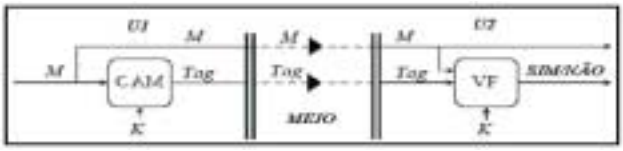


Figura 5 – Como ocorre o algoritmo da CAM.

5 - Funções Hash Criptográficas Iteradas

As *funções de hash criptográficas iteradas* são funções hash criptográficas de domínio $\{0,1\}^{\&} - \{\varepsilon\}$, com o valor de $|\{0,1\}^{\&}| = 2^{\&}$ (vide apêndice) bastante elevado, construídas a partir de uma *função de compressão*. Essa função é o coração das funções de hash criptográficas iteradas.

Funções de compressão são definidas da seguinte forma:

$f : (\{0,1\}^m - \{\varepsilon\}) \times (\{0,1\}^l - \{\varepsilon\}) \rightarrow (\{0,1\}^m - \{\varepsilon\})$, l *e* m *inteiros positivos diferentes de zero, com* l > m .

5. 1 - O Paradigma Merkle-Damgard

Seja uma função de hash criptográfica iterada H que recebe uma string binária *M* como entrada e retorna uma string binária H_n, de tamanho *m*(ou seja, H(*M*) = H_n). H é construída usando-se a função de compressão f , através de três etapas:

1ª - Pré-processamento

Divida *M* em substrings binárias M_1, M_2, M_3, ..., M_i, ..., M_n com TamBin(M_i) = *l* (vide apêndice). Caso TamBin(*M*) não seja um múltiplo de l, realize as seguintes ações em seqüência:

a)concatene ao lado direito de *M*, o bit 1 e depois tantos zeros quanto forem necessários até TamBin(*M*1 || 000...000) seja o menor inteiro positivo possível respeitando a seguinte relação:

TamBin(*M*1 || 000...000) é congruente a *l* - *&* módulo *l*.

b)calcule TamBit(*M*) (vide apêndice), concatene a esquerda de TamBit(*M*), zeros até TamBin (000...000 || TamBit(*M*)) = *&*. Após isso, concatene *M*1 || 000...000 || 000...000 || TamBit(*M*) obtendo a nova string binária *M*1000...000TamBit(*M*).

c)Divida *M*1000...000TamBit(*M*) em substrings binárias M_1, M_2, M_3, ..., M_i, ..., M_n com TamBin(M_i) = *l*.

No final, ou *M* será usado por f ou *M*1000...000TamBit(*M*) será usado por f. O que importa é que ambas as strings binárias são múltiplos de *l*. O bit "1" foi concatenado para evitar ambigüidade com possíveis zeros da esquerda de *M*. Todo esse processo de aumentar o tamanho da string binária até ser o menor múltiplo de *l* possível, é conhecido como *padding*(preenchimento).

2ª- Iteração com f

$IV = 0^m$;

$H_0 = f(IV , M_0)$;

$H_i = f(H_{i-1} , M_i)$, $i = 1 ... n$;

A constante *IV* é conhecida como *Initial Vector* ou *Vetor de Inicialização* que serve como primeiro valor de entrada para f além de M_0. Essa constante poderá ser qualquer valor variando conforme a função H que se deseja construir. O seu tamanho em bits deve ser igual a *m*.

H_0 é $f(IV, M_0)$ e $H_1, H_2 , ...H_i, ...H_n$ são os resultados do cálculo na $(i-1)$-ésima iteração de f. Esses valores são conhecidos como *chaining values*. A figura 6 ilustra bem esse conceito:

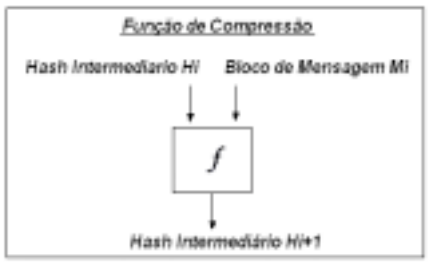


Figura 6 – *f* e o chaining value.

3ª- Saída

Retorna H_n;

Fim;

Sendo *n* o maior valor de *i*, H_n é resultado do cálculo após *n-1* iterações será o valor de H(*M*). A figura abaixo retrata bem o processo de iteração do hash criptográfico iterado:

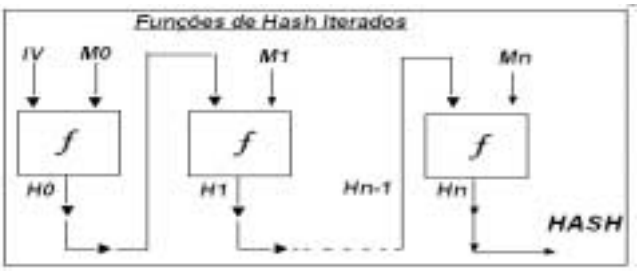


Figura 7 – o Hash H iterado a partir de *f*.

Essa forma de construção de H foi proposta inicialmente de uma forma mais simples, porém menos segura, independentemente por Ralph Merkle[Merkle] e Ivan Damgard[Damgard] com o intuito de montar um framework de construção para funções hash criptográficas. A concatenação do tamanho de *M* à direita *M10... 0* na etapa 1ª é conhecida como *MD-Strengthening*. A adição do bit 1, o padding e o MD-Strengthening foram adicionados à construção de Merkle e Damgard para torná-las mais seguras.

Construir funções hash criptográficas a partir de funções de compressão conforme mostrada acima ficou conhecido como *paradigma de construção MD* ou *Merkle-Damgard*. Portanto a função H é uma função do tipo Merkle-Damgard.

Há de se ressaltar duas propriedades extremamente interessantes de H quando construído sob o paradigma Merkle-Damgard. Quando uma colisão ocorre em H, H(x_1) = H(x_2) com x_1 e x_2 distintos e de mesmo tamanho, então para três strings binárias quaisquer *p,q e t*, H(q || x_1 || p) = H(q || x_2 || p) e H(x_1 || t) = H(x_2 || t). Isto se deve unicamente à iteração em f . Como o chaining value de x_1 e x_2 são os mesmos em cada caso(isto decorre da colisão, suponha o valor igual a H_i no primeiro caso e H'_i no segundo caso) depois do hashing interno de x_1 e x_2 em H(q || x_1 || p) = H(q || x_2 || p) e em H(x_1 || t) = H(x_2 || t) , o cálculo posterior de ambos será f (H_i, p) e f (H'_i, t) respectivamente.

6 – Parâmetros de Segurança

Para qualquer função hash criptográfica possuindo uma ou todas as propriedades fundamentais, seu design e construção ideais devem objetivar a dificuldade de se atacar qualquer uma dessas propriedades na prática. Essa dificuldade pode ser medida em função da quantidade de uso da função hash criptográfica e do seu contradomínio. Por exemplo, para a função de hash criptográfica segundo definição formal da seção 3.1 ser unidirecional, o número de tentativas do adversário que queira invertê-la deverá ser $|I| = 2^m$ aplicações de H para um |I| grande. Isto se justifica, pois se a função H é unidirecional, a melhor coisa que o adversário poderá fazer é calcular |I| entradas distintas para se ter uma esperança de que o resultado seja igual ao que ele queria inverter. A tabela abaixo mostra esse tipo de parametrização:

PROPRIEDADES FUNDAMENTAIS	*PARÂMETRO DE SEGURANÇA*
Unidirecionalidade	$\|I\| = 2^m$
Resistência à Segunda Pré-Imagem	$\|I\| = 2^m$
Resistência a Colisões	$\|I\|^{1/2} = 2^{m/2}$

Tabela 1 – Parâmetros de segurança para funções hash.

Funções hash criptográficas com esses parâmetros são consideradas *seguras*, desde que 2^m seja um valor muito alto em termos práticos.

O valor relativo à resistência a colisões é explicado pelo paradoxo do aniversário, pois tal paradoxo se aplica a qualquer função matemática. Esse tipo de abordagem de

segurança pode ser realizado via concrete security, haja vista a medição praticamente exata de que recursos o adversário necessitará, tudo dependerá apenas como H é definido.

Para se tentar provar esses parâmetros de segurança para H, uma primeira tentativa seria tratar H numa função pseudo-randômica, o que é bem natural, faria H uma função de hash criptográfica "boa" para ser usada por causa das propriedades das funções randômicas. Se H é uma função pseudo-randômica de cara ela seria uma função resistente a colisões e satisfaria a terceira propriedade. Sendo *A* um adversário de H, *Q* o oráculo da família ao qual H pertence e *q* a quantidade de consultas ao oráculo, a $Vant_Q(A)$ usando o paradoxo do aniversário seria no máximo q^2 dividido por 2^m. Portanto *A* teria que fazer uma quantidade de consultas próxima a $q = 2^{m/2}$ para possuir vantagem próxima de 1. É trivial notar que se H é resistente a colisões então H é resistente à segunda pré-imagem e como H é uma função pseudo-randômica, o melhor que A poderia fazer era tentar $2^m + 1$ entradas distintas para quebrar a resistência a segunda pré-imagem.

Unidirecionalidade não implica em resistência a colisões ou resistência a segunda pré-imagem conforme mostrado no trabalho de Simon[Simon98]. Além disso, resistência a colisões ou resistência à segunda pré-imagem não implica em unidirecionalidade. Por exemplo, se H fosse construída a partir de outra função criptográfica resistente a colisões G(G definida nos moldes da definição formal da seção 3.1), da seguinte forma:

$H(x) = 1 \,||\, x$, se $TamBin(x) = \log_2 |I|$;

$H(x) = 0 \,||\, G(x)$, qualquer outro caso.

Não é difícil verificar a resistência a colisões e a resistência à segunda pré-imagem de H(*x*), mas H(*x*) não é unidirecional, tendo em vista que se $H(x) = 1 || z$ com *z* uma string binária arbitrária, então $x = z$.

Uma vantagem dessa filosofia seria a prova de segurança de H para resistência a colisões e resistência a segunda pré-imagem. A grande desvantagem é a questão de se tratar H como uma caixa-preta, impossível de ser obtido na prática, e uma outra desvantagem seria não ter como provar a unidirecionalidade de H. O paradigma Merkle-

Damgard veio ajudar a tentar sanar o problema da caixa-preta, centrando toda segurança na função de compressão.

6. 1 - Prova da Segurança de H pela Construção MD

A seguir a prova resumida da segurança de H construída a partir de f pelo paradigma Merkle-Damgard:

Seja H uma função hash criptográfica iterada e f sua função de compressão conforme definição do capítulo 5. Antes da prova, convém definir:

- Colisão na função de compressão f

Dados um T constante e x_1, x_2 entradas distintas, $f(T, x_1) = f(T, x_2)$

- Pseudo-colisão na função de compressão f

Dados T_1, T_2 constantes distintas e x_1, x_2 entradas distintas, $f(T_1, x_1) = f(T_2, x_2)$

A definição de resistência a colisões e resistência a pseudo-colisões de f segue a mesma idéia da definição de H da seção 3.1. A função de compressão f é segura se for unidirecional, resistente a colisões, resistente a pseudo-colisões e se for resistente à segunda pré-imagem.

Teorema 1: Se f é resistente a pseudo-colisões então H é resistente a colisões

Prova: Suponha que H não seja resistente a colisões. Sejam strings binárias distintas M e M' ,entradas para H e $M_1, M_2, M_3, ..., M_i, ..., M_n$, $M'_1, M'_2, M'_3, ..., M'_i, ..., M'_n$ o resultado de M e de M' respectivamente após o pré-processamento com H(M) = H(M'). Seja $H_1, H_2, ...H_i, ...H_n$ e $H'_1, H'_2, ...H'_i, ...H'_n$ os chaining values de H(M) e H(M') respectivamente.

Caso TamBin(*M*) seja diferente de TamBin(*M'*) então M_n *e* M'_n são distintos e obtém-se uma pseudo-colisão em f, $f(H_n, M_n) = H(M) = H(M') = f(H'_n, M'_n)$. Caso TamBin(*M*) = TamBin(*M'*) divide-se em dois casos:

a)Se a partir de um *i*, $H_i = H'_i$ para todo *i*, seja um *j* maior que *i +1* tal que M_j seja diferente de M'_j. Então se obtém uma pseudo-colisão em f, $f(H_{j-1}, M_j) = H_j = H'_j = f(H'_{j-1}, M'_j)$.

b)Se existe um *i* com H_i diferente de H'_i, seja esse *i* máximo. Então se obtém uma pseudo-colisão em f, $f(H_i, M_{i+1}) = H_{i+1} = H'_{i+1} = f(H'_i, M'_{i+1})$. CQD.

Teorema 2: Se f é unidirecional então H é unidirecional.

Prova: Também pela contrapositiva. Seja *M* o valor obtido a partir de H(*M*). É fácil calcular $M_1, M_2, M_3, ..., M_i, ..., M_n$ o resultado após o pré-processamento. É fácil inverter f para qualquer iteração, pois com $M_1, M_2, M_3, ..., M_i, ..., M_n$ é trivial obterem-se os chaining values usados em f e, portanto, invertê-lo. CQD.

Desse jeito toda segurança de H reside em f. Se f é segura então H também o é e os parâmetros de segurança valem para H.

6.1.1 - Problema da Abordagem Merkle-Damgard

Apesar do paradigma Merkle-Damgard transferir toda a segurança para a função de compressão f, ele não dá detalhes de como construir f. Seria interessante que houvesse alguma forma de construir f e ao mesmo tempo, tentar garantir as premissas de segurança necessárias para a função hash criptográfica iterada H a partir de f, resolvendo o problema de como provar que H é segura.

6. 2 - Prova da Segurança de H via Construção Baseada em Cifras de Bloco

Uma proposta visando resolver o problema da construção de f pode ser encontrada em [PGV]. Esse paper propõe que as funções hash iteradas sejam construídas a partir de cifras de bloco, ou seja, f seria uma cifra de bloco. No mesmo paper, há 64 variações diferentes para f, mas este trabalho focará apenas em uma, conhecida como *Davies-Meyer*[DaviesMeyer]. Seja E_K uma cifra de bloco com chave binária *K,* então f seria definido:

$$f\,(H_{i-1}, M_i) = H_i = E_{Mi}\,(H_{i-1}) \text{ xor } H_{i-1}\ ,$$

com $M_1, M_2, M_3, ..., M_i, ..., M_n$ o resultado do pré-processamento de f e $H_1, H_2, ...H_i, ...H_n$ os chaining values de f.

Uma cifra de bloco é bijetiva, portanto para uma única chave binária de *E*, f construído dessa forma nunca possuiria colisão. Mas para cada iteração de f, uma nova chave binária M_i é usada em *E*, sendo assim f na verdade não é apenas uma única função mas possíveis instâncias da família de todas as cifras de bloco *E* possíveis(cada elemento da família de *E* é definido por uma chave binária diferente). Como se tratam de f´*S* diferentes, duas entradas diferentes concerteza gerarão dois f´*S* diferentes e existirá a possibilidade dessas duas entradas distintas resultarem na mesma saída. Apesar disso, convencionou-se tratar f como apenas uma única função e assim estas saídas iguais seriam ou uma colisão ou pseudo-colisão em f. Define-se f construído desse modo como *uma cifra de bloco operando no modo Davies-Meyer sob o paradigma Merkle-Damgard* e com isso define-se H construído a partir desse f como *uma função hash do tipo Davies-Meyer sob o paradigma Merkle-Damgard*.

A análise da segurança de f no modo Davies-Meyer pode ser encontrada em [Rogaway et al]. Este paper aplica a filosofia do concrete security para deduzir os parâmetros de segurança de f e de H. Rogaway e colaboradores partem do pressuposto

que *E* é uma permutação pseudo-randômica e um possível adversário *A* teria apenas acesso à caixa-preta de *E* e de sua inversa. Ele chega a conclusão que se *A* possui apenas acesso as caixas-pretas de *E* e sua inversa, f é resistente a colisões, resistente à segunda pré-imagem e unidirecional, segurança conforme os parâmetros mostrados anteriormente, implicando que H também é seguro como já foi mostrado aqui (seção 6.1).

6.2.1 - Problema da Abordagem de Construção de f via Cifras de Bloco

Novamente o problema da caixa-preta aparece. Para construir uma permutação pseudo-randômica *E*, a descrição interna deve ser inacessível ao adversário e deve existir uma família de permutações randômicas, o que é impossível na prática. Devido a esse inconveniente, uma forma adotada na prática é construir *E* tão próximo a uma permutação pseudo-randômica quanto possível e a partir daí deduzir todas as propriedades de segurança necessárias.

6.2.2 - Construção Heurística da Cifra de Bloco

A construção de *E*, se dá tentando garantir confusão e difusão conforme definição de Claude Shannon[Shannon], efeito avalanche, *efeito avalanche restrito*[Webster & Tavares] e alguns outros comportamentos conseqüentes da confusão e difusão.

Por se tratar de uma heurística, essa construção baseia-se em estruturas algorítmicas empiricamente mostradas como boas estatisticamente para prover tais comportamentos, a citar: *cifra de transposição, cifra de substituição, cifra produto, box de substituição(S-Box), redes de permutação, redes de substituição e permutação, redes de Feistel, redes de Feistel balanceadas e não-balanceadas, estratégia da trilha larga(wide trail strategy)*. Um detalhamento de cada um pode ser encontrado em [Transp], [Subst], [ProdCip], [SubstBox], [SubsPermNet], [Feistelnet], [UnbFeistelnet] e [WideTrail] respectivamente.

6. 3 - Parâmetros de Segurança para CAM´s

Como já foi mostrado na seção 4.2, o objetivo de um adversário em atacar um CAM seria criar(falsificar) novas strings *M* válidas. Para isso ele teria dois caminhos: ou atacar a resistência à segunda pré-imagem ou tentar deduzir a chave secreta. Se ele consegue deduzir a chave, todo o esquema construído a partir do CAM é inutilizado inclusive ele consegue deduzir novas strings. Pode ser que ele consiga atacar a resistência à segunda pré-imagem do CAM com menos esforço em relação à tentativa de dedução da chave obtendo strings autenticadas. Sendo assim não é do interesse dele tentar deduzir a chave secreta. Um bom CAM deve então dificultar ao máximo essas duas tarefas. Deduzir a chave secreta deve ser tão difícil quanto tentar chutar todas as possibilidades possíveis(*força-bruta*). Atacar a segunda pré-imagem deverá ser o menor esforço entre deduzir a chave secreta ou tentar atacar a própria propriedade fundamental. Dessa forma, o ataque à segunda pré-imagem no CAM tem o mesmo parâmetro de segurança da função hash criptográfica usada para construir o CAM. Sendo X = TamBin(String2Bin(K)), K a chave secreta e H a função de hash criptográfica segundo definição formal da seção 3.1, a tabela abaixo ilustra bem esses conceitos:

ATAQUES AO CAM	*PARÂMETRO DE SEGURANÇA*
Força Bruta na Chave Secreta	2^X
Falsificação de Mensagem	Mínimo de ($\|I\| = 2^m$), 2^X)

Tabela 2 – Parâmetros de segurança para CAM´s.

A chave secreta K deve ser gerada aleatoriamente para dificultar o trabalho do adversário, além do mais X deve ser de um tamanho que dificulte a força bruta.

7 – Algoritmos de Hash e de HMAC

Antes da descrição detalhada de cada algoritmo, alguns detalhes precisam ser definidos:

a) As operações de soma são sempre processadas módulo 2^{32};

b) Um deslocamento de *s* bits para a esquerda é representado por <<< s.

c) As operações lógicas binárias AND, OR, XOR e NOT são representadas por "^", "v", "xor" e "¬" respectivamente.

7. 1 - O Algoritmo *MESSAGE DIGEST 4* - MD4

O algoritmo MD4[Rivest90, Rivest91], o quarto da série de algoritmos criados por Ronald Rivest, é um CDM customizado cuja função hash do tipo Davies-Meyer sob o paradigma Merkle-Damgard. Ele foi feito para ser bastante eficiente em máquinas de arquitetura de 32 bits(como a Intel, por exemplo) e para softwares de alta-performance.

Nada melhor que as palavras do próprio autor[Rivest91] para descrever os objetivos do algoritmo:

*"**Security**.* **It is computationally infeasible to find two messages that hashed to the same value. No attack is more efficient than brute force.**

***Direct Security*. MD4's security is not based on any assumption, like the difficulty of factoring.**

***Speed.* MD4 is suitable for high-speed software implementations. It is based on a simple set of bit manipulations on 32-bit operands.**

***Simplicity and Compactness.* MD4 is as simple as possible, without large data structures or a complicated program.**

Favor Little-Endian Architectures. **MD4 is optimized for microprocessor architectures (specifically Intel microprocessors); larger and faster computers make any necessary translations."**

Sua saída é uma string binária de 128 bits. Ele divide a entrada em blocos de 512 bits e cada bloco de 512 bits é subdividido em blocos de 32 bits resultando em 16 sub-blocos.

Embora seja um algoritmo extremamente prático, fraquezas foram apontadas pelos pesquisadores [den Boer & Bosselaers, Biham92]. Sendo assim, Ronald Rivest preocupou-se em desenvolver um algoritmo mais seguro, uma versão mais forte do MD4, para tentar suprimir as fraquezas encontradas pelos criptoanalistas, resultando no MD5[Rivest92]. As principais diferenças entre eles são:

a) Uma quarta fase foi adicionada no MD5, mas no MD4 existem apenas três;

b) No MD5, cada passo tem uma constante aditiva única *t*, mas no MD4 as constantes são reutilizadas;

c) A função usada na fase II do MD5, G(X,Y,Z) = (X ^ Z) v (Y ^ ¬Z) é diferente no MD4; ela é descrita como G(X,Y,Z) = ((X ^ Y) v (X ^ Z)) v (Y ^ Z). O objetivo dessa mudança foi deixá-la menos simétrica;

d) Após as fases para cada bloco de bits no MD5, adiciona-se o resultado com o resultado anterior, promovendo um efeito avalanche mais rápido;

e) A ordem em que blocos de 32 bits M_i no MD4 eram acessados nas fases 2 e 3 foi trocada para diminuir a semelhança entre elas;

f) Os valores do deslocamento *s* foram otimizados para causar um efeito avalanche de mais qualidade.

Para entender estas mudanças, vide [Rivest91] e a descrição do MD5 na seção seguinte, tendo em vista que neste trabalho não é de interesse dar uma descrição do MD4 em detalhes, por estar fora de escopo do tema proposto.

7. 2 - O Algoritmo *MESSAGE DIGEST 5* - MD5

O algoritmo MD5[Rivest92] funciona de forma muito parecida ao MD4, pois possuem design semelhante. Portanto o MD5 é um CDM customizado cuja função hash é do tipo Davies-Meyer sob o paradigma Merkle-Damgard. O MD5 foi feito para operar com a arquitetura *little-endian*[LittleEnd]. Apesar disso, o MD5 é bem mais complexo. Antes da descrição do mesmo, algumas definições precisam ser observadas:

a) As funções não lineares são uma para cada fase:

$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$ FASE I

$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$ FASE II

$H(X,Y,Z) = X \text{ xor } Y \text{ xor } Z$ FASE III

$I(X,Y,Z) = Y \text{ xor } (X \vee \neg Z)$ FASE IV

b) A entrada pode ter tamanho de até 2^{64} -1 bits;

c) A saída possui tamanho de 128 bits;

Abaixo segue uma descrição detalhada do que ocorre com a execução do algoritmo:

1º- *Vetor de inicialização(IV):* quatro variáveis de 32 bits *A*, *B*,*C*, *D* são inicializadas com os respectivos valores fixados pela definição do algoritmo:

A: 0x01234567

B: 0x 89abcdef

C: 0xfedcba98

D: 0x76543210

2º- *Verificação de Tamanho da entrada e Padding*: primeiro, o tamanho da entrada *M*(uma string binária) é ajustado com a concatenação de um bit 1 seguido de tantos zeros

quanto forem necessários, até que TamBin(*M*)(vide apêndice) seja o menor possível tal que TamBin(*M*) seja congruente a 448 módulo 512(ficará algo do tipo *M10....00*). Então esse novo *M* será também concatenado com uma string binária *'T'* de 64 bits que representa o tamanho da entrada em bits(a entrada então poderá ter tamanho máximo de 2^{64} – 1 bits). Sendo assim, o tamanho da nova entrada *M10....00'T'* será um múltiplo exato de 512 bits. Por exemplo, se *M* possuir 513 bits teremos um bit 1, quatrocentos e quarenta e seis bits 0 e 64 bits concatenados com *M*, formando uma nova string de tamanho 1024 que é múltiplo de 512. Depois, a entrada modificada *M* é dividida em blocos de 512 bits, e logo após cada bloco de 512 bits é subdivido em 16 blocos de 32 bits para serem aproveitadas no próximo passo (vide figura abaixo).

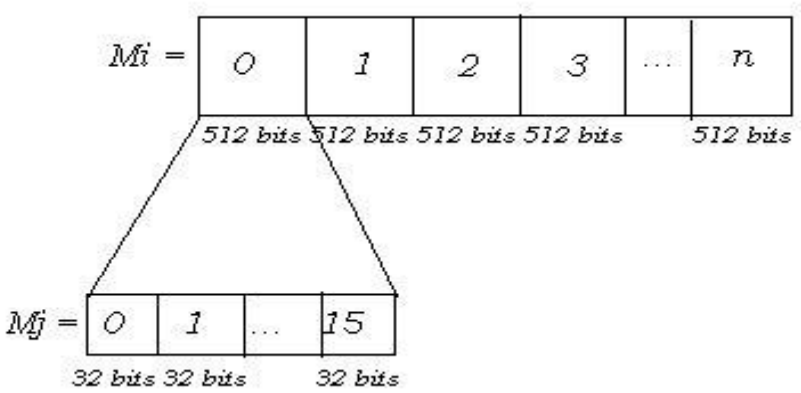


Figura 8 – Entrada sendo subdividida em blocos no MD5.

3º- <u>*Laço Principal*</u>: Após o passo 2, o laço principal é executado uma vez para cada bloco M_i existente:

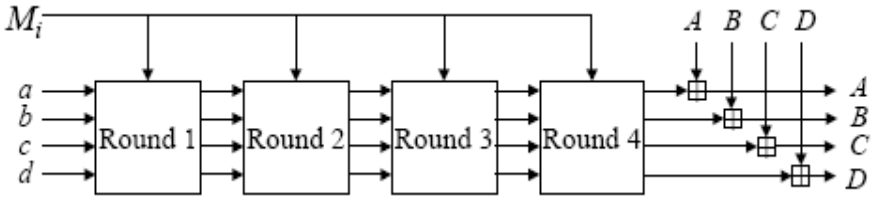


Figura 9 - Figura copiada de [Schneier], página 437.

Dentro do laço principal temos os seguintes subpassos:

(3a) os valores *A, B, C* e *D* são copiados respectivamente para as variáveis auxiliares *a, b, c* e *d* que então são utilizadas em cada fase(round) do laço principal;

(3b) como o bloco M_i recebido já foi dividido em 16 subblocos de 32 bits cada(M_j, com j indo de "0" a "15") na etapa 2, cada um desses subblocos é processado e aproveitado dentro de uma fase, resultando na execução de 16 interações em cada fase, que é o laço interno do algoritmo MD5.

(3c) em cada uma das 16 interações, três das quatro variáveis são utilizadas como entrada para uma função não linear. Ao resultado dessa função, é somado o valor da variável não utilizado por ela, o bloco de 32 bits M_j e uma constante t_p; este resultado intermediário sofre um deslocamento de *s* bits para a esquerda(<<< *s*) e então é somado a ele o valor de uma das variáveis(*a,b,c,d*); finalmente, o valor de uma dessas variáveis é substituído pelo resultado dessas operações; a figura abaixo ilustra a execução de um passo:

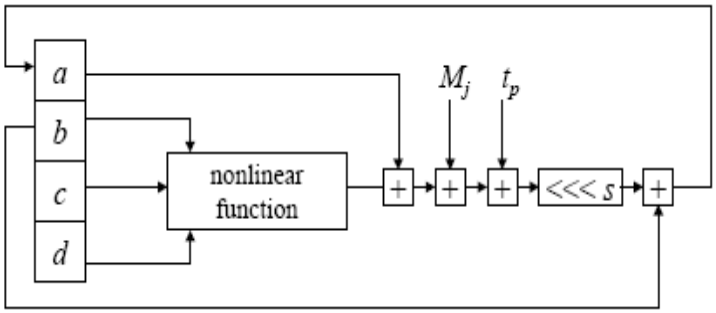


Figura 10 - Figura copiada de [Schneier], página 438.

A fórmula geral de uma interação numa fase a partir das funções não-lineares é definida abaixo:

$$\#\#(@, x, y, z, M_j, s, t_p) :: @ = ((\ \$(x,y,d) + v + M_j + t_p)\ <<< s) + x,$$

onde o caractere # é o caracter *F,G,H* ou *I* que representa as funções não-lineares, ## é uma nova função desenvolvida a partir de uma das 4 funções não-lineares e operadores sobre as outras variáveis envolvidas, $ é uma das funções não lineares , e

as variáveis *@, x, y* e *z* são substituídas por *a,b,c* e *d* não necessariamente nesta ordem. A ordem no qual elas são substituídas muda a cada interação de uma fase, sendo que a ordem inicial é (*a,b,c,d*) seguida de (*d,a,b,c*), (*c,d,a,b*), (*b,c,d,a*) e então volta-se para (*a,b,c,d*) iniciando-se novamente a seqüência.

A ordem de acesso aos blocos de 16 bits M_j não é necessariamente seqüencial, variando para cada interação; *j* então varia assumindo os seguintes valores na ordem em que são solicitados:

FASE 1 : {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}

FASE 2 : {1,6,11,0,5,10,15,4,9,14,3,8,13,2,7,12}

FASE 3 : {5,11,14,1,4,7,10,13,0,3,6,9,12,15,2}

FASE 4 : {0,7,14,5,12,3,10,1,8,15,6,13,4,11,2,9}

A constante t_p é diferente em cada interação; assim, tendo o algoritmo quatro fases e cada fase realizando 16 interações, tem-se no resultado do processamento de um M_i um total de 64 interações, ou seja, o laço principal terminará realizando 64 interações para cada M_i. Portanto, o índice *p* de *t* indica a interação atual do laço principal, e então *t* é dado por $t = (2^{32})* abs(sen(p))$, *p* indo de 1 até 64 , *p* em radianos, *abs* sendo a função valor absoluto de um número(vide apêndice) e *sen* a função seno.

O valor de *s* também varia para cada interação, mas seguindo um padrão diferente. Para cada fase, há uma seqüência de quatro valores para *s*. A cada um das 16 interações do laço interno, um valor de *s* é utilizado de tal forma que quando se chega ao último valor, volta-se a utilizar o primeiro; as seqüências de valores de *s* para cada fase são:

FASE 1 : {7,12,17,22}

FASE 2 : {5,9,14,20}

FASE 3 : {4,11,16,23}

FASE 4 : {6,10,15,21}

Desta forma, a tabela com cada variável com seu valor respectivo e com uma nova função ## segue abaixo:

FASE I	FASE II
$FF(a,b,c,d,M_0,7,0xd76aa478)$	$GG(a,b,c,d,M_1,5,0xf61e2562)$
$FF(d,a,b,c,M_1,12,0xe8c7b756)$	$GG(d,a,b,c,M_6,9,0xc040b340)$
$FF(c,d,a,b,M_2,17,0x242070db)$	$GG(c,d,a,b,M_{11},14,0x265e5a51$
$FF(b,c,d,a,M_3,22,0xc1bdceee)$	$GG(b,c,d,a,M_0,20,0xe9b6c7aa)$
$FF(a,b,c,d,M_4,7,0xf57c0faf)$	$GG(a,b,c,d,M_5,5,0xd62f105d)$
$FF(d,a,b,c,M_5,12,0x4787c62a)$	$GG(d,a,b,c,M_{10},9,0x02441453)$
$FF(c,d,a,b,M_6,17,0xa8304613)$	$GG(c,d,a,b,M_{15},14,0xd8a1e681$
$FF(b,c,d,a,M_7,22,0xfd469501)$	$GG(b,c,d,a,M_4,20,0xe7d3fbc8)$
$FF(a,b,c,d,M_8,7,0x698098d8)$	$GG(a,b,c,d,M_9,5,0x21e1cde6)$
$FF(d,a,b,c,M_9,12,0x8b44f7af)$	$GG(d,a,b,c,M_{14},9,0xc33707d6)$
$FF(c,d,a,b,M_{10},17,0xffff5bb1)$	$GG(c,d,a,b,M_3,14,0xf4d50d87)$
$FF(b,c,d,a,M_{11},22,0x895cd7be)$	$GG(b,c,d,a,M_8,20,0x455a14ed)$
$FF(a,b,c,d,M_{12},7,0x6b901122)$	$GG(a,b,c,d,M_{13},5,0xa9e3e905)$
$FF(d,a,b,c,M_{13},12,0xfd987193)$	$GG(d,a,b,c,M_2,9,0xfcefa3f8)$
$FF(c,d,a,b,M_{14},17,0xa6794383)$	$GG(c,d,a,b,M_7,14,0x676f02d9)$
$FF(b,c,d,a,M_{15},22,0x49b40821)$	$GG(b,c,d,a,M_{12},20,0x8d2a4c8a$
FASE III	**FASE IV**
$HH(a,b,c,d,M_5,4,0xfffa3942)$	$II(a,b,c,d,M_0,6,0xf4292244)$
$HH(d,a,b,c,M_8,11,0x8771f681)$	$II(d,a,b,c,M_7,10,0x432aff97)$
$HH(c,d,a,b,M_{11},16,0x6d9d6122)$	$II(c,d,a,b,M_{14},15,0xab9423a7)$
$HH(b,c,d,a,M_{14},23,0xfde5380c)$	$II(b,c,d,a,M_5,21,0xfc93a039)$
$HH(a,b,c,d,M_1,4,0xa4beea44)$	$II(a,b,c,d,M_{12},6,0x655b59c3)$
$HH(d,a,b,c,M_4,11,0x4bdecfa9)$	$II(d,a,b,c,M_3,10,0x8f0ccc92)$
$HH(c,d,a,b,M_7,16,0xf6bb4b60)$	$II(c,d,a,b,M_{10},15,0xffeff47d)$
$HH(b,c,d,a,M_{10},23,0xbebfbc70)$	$II(b,c,d,a,M_1,21,0x85845dd1)$
$HH(a,b,c,d,M_{13},4,0x289b7ec6)$	$II(a,b,c,d,M_8,6,0x6fa87e4f)$
$HH(d,a,b,c,M_0,11,0xeaa127fa)$	$II(d,a,b,c,M_{15},10,0xfe2ce6e0)$
$HH(c,d,a,b,M_3,16,0xd4ef3085)$	$II(c,d,a,b,M_6,15,0xa3014314)$
$HH(b,c,d,a,M_6,23,0x04881d05)$	$II(b,c,d,a,M_{13},21,0x4e0811a1)$
$HH(a,b,c,d,M_9,4,0xd9d4d039)$	$II(a,b,c,d,M_4,6,0xf7537e82)$
$HH(d,a,b,c,M_{12},11,0xe6db99e5)$	$II(d,a,b,c,M_{11},10,0xbd3af235)$
$HH(c,d,a,b,M_{15},16,0x1fa27cf8)$	$II(c,d,a,b,M_2,15,0x2ad7d2bb)$
$HH(b,c,d,a,M_2,23,0xc4ac5665)$	$II(b,c,d,a,M_9,21,0xeb86d391)$

Figura 11 – Tabela para as respectivas funções e valores das variáveis em cada fase do MD5.

(3d) encerrada uma fase, as variáveis (*a,b,c,d*) são repassadas para a fase seguinte;

(3e) após a realização das quatro fases, os valores (*A,B,C,D*) são atualizados somando-se a eles os valores de (a,b,c,d) respectivamente, da seguinte forma;

$A = A + a$

$B = B + b$

$C = C + c$

$D = D + d$

(3f) enquanto houver blocos M_i a serem processados, passa-se para o próximo bloco M_i e o algoritmo volta para a etapa 3a novamente;

Observação Importante: O design da função de compressão no modo Davies-Meyer sob o paradigma Merkle-Damgard fica bastante evidente neste 3º passo. Além das interações típicas de funções do tipo representada pelo laço principal, o xor realizado com o resultado da cifra de bloco na definição é equivalente à soma do subpasso 3e por causa da operação modular. Além disso, o padding do 2ºpasso mostra a característica marcante do processo Merkle-Damgard.

4º- *Saída*: após todos os blocos M_i de 512 bits terem sido processados, o valor do hash é dado por $A||B||C||D$, resultando em um hash de 4*32 = 128 bits.

7.2.1 – Parâmetros de Segurança para o MD5

Abaixo a tabela com os parâmetros de segurança para o MD5, segundo o tamanho do seu contradomínio(128 bits) :

PROPRIEDADES FUNDAMENTAIS	*PARÂMETRO DE SEGURANÇA*
Unidirecionalidade	2^{128}
Resistência a Segunda Pré-Imagem	2^{128}
Resistência a Colisões	2^{64}

Tabela 3 – Parâmetros de segurança para o MD5.

Esses parâmetros valem tanto para a função de compressão do MD5 quanto para o algoritmo MD5 completo. No primeiro caso, resistência a pseudo-colisões possuem o mesmo valor que resistência a colisões na função de compressão. No segundo caso, deve ser levado em conta que o IV da entrada é uma constante pré-definida fixa, característica do próprio algoritmo conforme descrito na seção anterior. O objetivo de design não leva em conta a possibilidade de se encontrarem colisões no MD5 para um outro IV fixo qualquer ou para qualquer IV sorteado aleatoriamente, mas acredita-se que os parâmetros de segurança devam valer também para esses casos.

7.2.2 – Efeito Avalanche no MD5

Como o MD5 é uma função hash do tipo Davies-Meyer sob o paradigma Merkle-Damgard é de se esperar que possua efeito avalanche provocado pela propriedade da difusão, tendo em vista que sua construção está fundamentada na mesma heurística de construção das cifras de bloco. A título de ilustração desse comportamento, segue o exemplo abaixo:

> Seja a seguinte string: "*Os homens fazem as ferramentas e as ferramentas refazem os homens*". Seja *X* = String2Bin(*Os homens fazem as ferramentas e as ferramentas refazem os homens*) (vide apêndice).

Então MD5(*X*) = **0x 7A 5A 17 E0 51 17 DA B4 98 E2 2C 3B B1 F7 45 F6**

O caractere “e” no computador é representado em ascii como 01100101.

Para verificar o efeito avalanche, é suficiente mudar apenas um bit da entrada. Para isso, basta modificar o caractere “e” do meio da frase pelo caractere “d” tendo em vista que “d” é representado por 0110010**0**. Seja X’ = String2Bin(*Os homens fazem as ferramentas **d** as ferramentas refazem os homens*).

Então MD5(X’) = **0x 5D 7F 08 0C 96 62 2A 4F 7D B6 A1 5D 7F 2A 17 D0**, ou seja, 55,5% dos bits(71 bits de 128) dessa saída são diferentes em suas respectivas posições em relação a saída de MD5(X). Esse resultado está dentro do esperado.

7. 3 - O Algoritmo *SECURE HASH ALGORITHM 1* - SHA-1

A especificação original do SHA, foi publicada em 1992 pela National Security Agency - NSA[NSA] e pelo National Institute of Standards and Technology - NIST[NIST] para ser usada no padrão do governo americano(FIPS – Federal Information Processing Standard) de assinaturas digitais[FIPS](a publicação ficou conhecida como FIPS 180) e depois corrigida em 1995(a publicação ficou conhecida como FIPS 180-1). Segundo o NIST, isto foi feito para corrigir um problema no algoritmo original, problema esse que diminuía a segurança, embora o NIST não tenha justificado porque esse problema comprometia o algoritmo. O algoritmo original ficou conhecido como SHA-0 e o algoritmo revisado como SHA-1. A publicação atual do SHA-1 é conhecida como FIPS 180-2(mais recente que a FIPS 180-1).

O algoritmo SHA-1[FIPS180-2] funciona em alguns pontos de forma semelhante ao MD4 e, portanto, ao MD5. Sendo assim, ele também é um CDM customizado cuja função hash é do tipo Davies-Meyer sob o paradigma Merkle-Damgard. Ele trabalha com blocos de 512 bits, realizando o mesmo processo de verificação de tamanho e padding que o MD5 realiza, inclusive na concatenação de uma string binária que representa o tamanho da entrada, de tamanho 64 bits. Portanto a entrada pode ter tamanho de até $2^{64} – 1$ bits. Além do mais, ele divide cada bloco de 512 bits em subblocos de 16 por 32 bits da mesma forma que o MD5. Possui também quatro fases(rounds) para cada bloco de 512

bits processados. Mas as semelhanças acabam por aqui, pois existem várias diferenças. Por exemplo, a saída do algoritmo resulta numa string binária de 160 bits, diferente do MD5, que tem como saída uma string binária de 128 bits; o SHA-1 foi desenvolvido para atuar em uma arquitetura *big-endian*[LittleEnd]. A seguir, a descrição do SHA-1:

1º- *Vetor de inicialização(IV)*: diferente do MD5, cinco variáveis de 32 bits *A*, *B*,*C*, *D e E* são inicializadas com os respectivos valores fixados pela definição do algoritmo:

A: 0x67452301

B: 0xefcdab89

C: 0x98badcfe

D: 0x10325476

E: 0xe3d2e1f0

2º- *Verificação de Tamanho da entrada e Padding*: Assumindo *M* uma string binária como entrada, ocorre da mesma forma que o MD5, conforme já descrito;

3º- *Pré-processamento de novos blocos*: a partir dos 16 subblocos de 32 bits cada obtidos na etapa 2, 80 W_t novos blocos(t indo de 0 até 79) de 32 bits são gerados para serem usados no algoritmo, da seguinte forma:

$W_t = M_t$, de *t* indo de 0 até 15;

$W_t = (W_{t-3}$ xor W_{t-8} xor W_{t-14} xor $W_{t-16}) <<<1$, de *t* indo de 16 até 79;

4º- *Execução*: por ser mais simples que o MD5, pode-se imaginar o SHA-1 como um grande laço principal de 80 interações com uma variável *t* indo de 0 até 79 , para cada bloco de 512 bits obtidos na etapa 2. As fases(rounds) ocorrem em função do valor de *t* da seguinte forma:

FASE 1 : *t* indo de 0 até 19 ;

FASE 2 : *t* indo de 20 até 39 ;

FASE 3 : *t* indo de 40 até 59 ;

FASE 4 : *t* indo de 60 até 79 ;

A execução subdivide-se assim:

(4a) os valores *A, B, C, D* e *E* são copiados respectivamente para as variáveis auxiliares *a, b, c, d* e a variável *e* ;

(4b) o algoritmo executa o seguinte laço(em pseudo-código):

Para *t* indo de 0 até 79 faça

{

$TEMP = (a <<< 5) + f(b,c,d) + e + W_t + cte$

$e = d$

$d = c$

$c = (b <<< 30)$

$b = a$

$a = TEMP$

}

A função *f* e a variável *cte* são usadas conforme tabela abaixo:

FASES	*CONSTANTE cte*	*FUNÇÃO f(x,y,z)*
FASE 1	**0x5a827999**	**(x ^ y) xor (¬x ^ z)**
FASE 2	**0x6ed9eba1**	**x xor y xor z**
FASE 3	**0x8f1bbcdc**	**(x ^ y) xor (x ^ z) xor (y ^ z)**
FASE 4	**0xca62c1d6**	**x xor y xor z**

Tabela 4 – constantes e funções usadas em cada fase do SHA-1.

(4c) após a realização das quatro fases, os valores (*A,B,C,D,E*) são atualizados somando-se a eles os valores de (*a,b,c,d,e*) módulo 2^{32} respectivamente assim:

$A = A + a$

$B = B + b$

$C = C + c$

$D = D + d$

$E = E + e$

(4d) enquanto houver blocos M_i a serem processados, passa-se para o próximo bloco M_i e o algoritmo volta para a etapa 4a novamente. Pode-se ilustrar o processo com a figura abaixo:

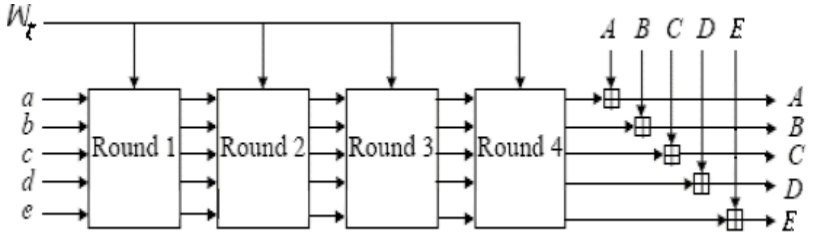


Figura 12 – Execução do SHA-1 para um conjunto de W_t ´s.

Observação Importante: O design da função de compressão no modo Davies-Meyer sob o paradigma Merkle-Damgard fica bastante evidente neste 4º passo. Além das interações típicas de funções do tipo representada pelo laço principal, o xor realizado com o resultado da cifra de bloco na definição é equivalente à soma do subpasso 4c por causa da operação modular. Além disso, o padding do 2º passo mostra a característica marcante do processo Merkle-Damgard.

5º- *Saída*: após toda etapa 4 ter sido concluída, o algoritmo retorna *A||B||C||D||E* como resposta. Repare que cada variável possui 32 bits e a saída possui 5*32 =160 bits.

7.3.1 – Parâmetros de Segurança para o SHA-1

Abaixo a tabela com os parâmetros de segurança para o SHA-1, segundo o tamanho do seu contradomínio(160 bits) :

PROPRIEDADES FUNDAMENTAIS	*PARÂMETRO DE SEGURANÇA*
Unidirecionalidade	2^{160}
Resistência a Segunda Pré-Imagem	2^{160}
Resistência a Colisões	2^{80}

Tabela 5 – Parâmetros de segurança para o SHA-1.

Esses parâmetros valem tanto para a função de compressão do SHA-1 quanto para o algoritmo SHA-1 completo. No primeiro caso, resistência a pseudo-colisões possuem o mesmo valor que resistência a colisões na função de compressão. No segundo caso, deve ser levado em conta que o IV da entrada é uma constante pré-

definida fixa, característica do próprio algoritmo conforme descrito na seção anterior. O objetivo de design não leva em conta a possibilidade de se encontrar colisões no SHA-1 para um outro IV fixo qualquer ou para qualquer IV sorteado aleatoriamente, mas acredita-se que os parâmetros de segurança devam valer também para esses casos.

7.3.2 – Efeito Avalanche no SHA-1

Como o SHA-1 é uma função hash do tipo Davies-Meyer sob o paradigma Merkle-Damgard é de se esperar que possua efeito avalanche provocado pela propriedade da difusão, tendo em vista que sua construção está fundamentada na mesma heurística de construção das cifras de bloco. A título de ilustração desse comportamento, segue o exemplo abaixo:

> Seja a seguinte string: "*Os homens fazem as ferramentas e as ferramentas refazem os homens*". Seja *X* = String2Bin(*Os homens fazem as ferramentas e as ferramentas refazem os homens*) (vide apêndice).
>
> Então SHA-1(*X*) = **C1 F3 19 7E 42 E9 51 C2 22 81 FE 4F E7 57 E9 45 30 48**
>
> O caractere "e" no computador é representado em ascii como 01100101.
>
> Para verificar o efeito avalanche, é suficiente mudar apenas um bit da entrada. Para isso, basta modificar o caractere "e" do meio da frase pelo caractere "d" tendo em vista que "d" é representado por 0110010**0**. Seja X' = String2Bin(*Os homens fazem as ferramentas **d** as ferramentas refazem os homens*).
>
> Então SHA-1(X') = **72 D9 8B D8 3E 68 04 9E A0 97 93 FC 18 51 7F 49 C4 5A**, ou seja, 46,9% dos bits(75 bits de 160) dessa saída são diferentes em suas respectivas posições em relação a saída de SHA-1(X). Como 0.469 é próximo a 0.5, esse resultado está dentro do esperado.

7. 4 - O Algoritmo *HASH* MESSAGE AUTHENTICATION CODE – *MD5*(HMAC-MD5)

O HMAC-MD5 nada mais é que uma função HMAC(CAM customizado) com a função de hash MD5 conforme definição dada na seção 3.1.2. Tanto a entrada, a saída e a chave secreta *K* são strings binárias. As constantes *ipad* e *opad* são definidas assim: *ipad* = 0x363636...36 e *opad* = 0x5c5c5c...5c com a repetição dos hexadecimais 36 e 5c 64 vezes em *ipad* e *opad* respectivamente. A única operação a mais que o HMAC-MD5 realiza em relação a definição dada na seção 3.1.2 é o padding de *K* com zeros até *K* possuir o mesmo tamanho que *ipad* e *opad*.

Os protocolos TLs[TLs] e IPSec[IPSec] fazem uso dele. Sua especificação pode ser vista em [RFC2403].

7.4.1 – Parâmetros de Segurança para o HMAC- MD5

Abaixo a tabela com os parâmetros de segurança para o HMAC-MD5, segundo o tamanho do contradomínio do MD5 e da chave secreta *K*(128 e 512 bits respectivamente):

ATAQUES AO CAM	*PARÂMETRO DE SEGURANÇA*
Força Bruta na Chave Secreta	2^{512}
Falsificação de Mensagem	Mínimo de (2^{128}, 2^{512})

Tabela 6 – Parâmetros de segurança para HMAC-MD5.

7. 5 - O Algoritmo *HASH* MESSAGE AUTHENTICATION CODE – *SHA-1*(HMAC-SHA-1)

O HMAC-SHA-1 nada mais é que uma função HMAC(CAM customizado) com a função de hash SHA-1 conforme definição dada na seção 3.1.2. Tanto a entrada, a saída e a chave secreta *K* são strings binárias. As constantes *ipad* e *opad* são definidas assim: *ipad* = 0x363636...36 e *opad* = 0x5c5c5c...5c com a repetição dos hexadecimais 36 e 5c 64 vezes em *ipad* e *opad* respectivamente. A única operação a mais que o HMAC-SHA-1 realiza em relação a definição dada na seção 3.1.2 é o padding de *K* com zeros até *K* possuir o mesmo tamanho que *ipad* e *opad*.

Os protocolos TLs[TLs] e IPSec[IPSec] fazem uso dele, semelhante ao HMAC-MD5. Sua especificação pode ser vista em [RFC2404].

7.5.1 – Parâmetros de Segurança para o HMAC- SHA-1

Abaixo a tabela com os parâmetros de segurança para o HMAC-SHA-1, segundo o tamanho do contradomínio do SHA-1 e da chave secreta *K*(160 e 512 bits respectivamente):

ATAQUES AO CAM	*PARÂMETRO DE SEGURANÇA*
Força Bruta na Chave Secreta	2^{512}
Falsificação de Mensagem	Mínimo de(2^{160}, 2^{512})

Tabela 7 – Parâmetros de segurança para HMAC-SHA-1.

8 - Estado da Arte dos Ataques ao MD5 e ao SHA-1

A maioria absoluta dos ataques a funções hash criptográficas usadas na prática referem-se à capacidade de se acharem colisões. Mesmo se a complexidade do ataque não permitir uma implementação prática, uma função hash é considerada quebrada caso o parâmetro de colisão tenha sido reduzido em relação ao parâmetro previsto teoricamente. Isto tudo é levado em conta devido à quebra da própria filosofia das funções hash criptográficas quando o parâmetro de segurança é reduzido.

Como mostrado na seção 5.1, uma colisão terá como conseqüência a obtenção de novas colisões devido ao paradigma Merkle-Damgard. Quando uma colisão ocorre em H, $H(x_1) = H(x_2)$ com x_1 e x_2 distintos e de mesmo tamanho, então para três strings binárias quaisquer *p,q e t*, $H(q \| x_1 \| p) = H(q \| x_2 \| p)$ e $H(x_1 \| t) = H(x_2 \| t)$. A obtenção de várias colisões de um mesmo valor é conhecido como *multicolisões*.

8.1 – Ataques ao MD5

Em 1993, den Boer e Bosselaers[Boer & Bosselaers 93] acharam duas entradas que geram uma colisão no MD5 usando dois IV´s distintos cuja diferença era de apenas 4 bits. Em 1996, Hans Dobbertin[Dobbertin96] demonstrou uma pseudo-colisão na função de compressão do MD5. Em março de 2004 o projeto *MD5CRK*[MD5CRK] teve o intuito de achar colisões no MD5 via força bruta através de computadores distribuídos. Como o parâmetro de segurança do MD5 para resistência a colisões é 2^{64}, acreditava-se o uso do poder de milhares de computadores distribuídos pelo mundo poderiam gerar colisões fáceis no MD5. Este projeto foi finalizado quando em 17 de Agosto de 2004, uma equipe de pesquisadores da Shandong University, China, liderada pela pesquisadora Xiaoyum Wang, anunciou na *CRYPTO 2004*[CRYPTO2004] que poderiam achar várias colisões no

MD5 na ordem de 2^{37} passos[Wang et al]. Como exemplo, eles divulgaram duas entradas de 1024 bits que tinham o mesmo MD5, mas não entraram em detalhes sobre o método. Detalhes sobre sua metodologia de ataque só vieram a ser conhecidos em 2005[Wang & Yu] na *EUROCRYPT 2005*[EUROCRYPT2005].

O método de Wang e colaboradores basicamente é capaz de achar várias entradas de 1024 bits que colidem para qualquer IV. Isto torna o ataque mais poderoso, pois o IV não é o pré-definido. Sendo M_1, M_2, M_1' e M_2' vetores de 16 posições com cada posição possuído 4 bytes(cada um possuirá 512 bits), escritos em little-endian, achados por esse método, a relação entre eles será:

$$MD5(M_1 \,||\, M_2) = MD5(M_1' \,||\, M_2'),$$

$$M_1' = M_1 + \Delta 1 (\text{módulo } 2^{32}),$$

$$M_2' = M_2 + \Delta 2 (\text{módulo } 2^{32}),$$

com $\Delta 1 = (0,0,0,0,0,2^{31},...,2^{15},...,2^{31},0)$ e $\Delta 2 = (0,0,0,0,0,2^{31},...,-2^{15},...,2^{31},0)$ vetores de 16 posições com as posições 4, 11 e 14 diferente de zero e na forma little-endian. Eles usaram o supercomputador IBM P690 para achar tais vetores. Segundo a própria descrição deles, para achar M_1 e M_1', levou em média uma hora e para achar M_2' e M_2 levou em média 5 minutos. É fácil perceber que o método de Wang e colaboradores é capaz de achar múltiplas strings binárias no qual diferem apenas em 6 posições de 4 bytes em pouquíssimas horas num supercomputador.

Logo após esse divulgação, Vlastimil Klima[Klima] divulgou um trabalho independente e obteve um ataque para qualquer IV na ordem de 2^{33} passos. Umas das poucas semelhanças em relação ao ataque de Wang e colaboradores, é a capacidade de achar várias entradas de 1024 bits e caso se interprete essas entradas como vetores de 16 posições de 4 bytes cada em little-endian, 6 posições diferem entre-si, 3 em cada bloco de 512 bits mas não necessariamente nas mesmas posições. Além de diferenças no algoritmo e em algumas idéias usadas para desenvolvê-lo, o grande diferencial está relacionado ao tempo necessário em achar tais colisões. O método de Klima leva apenas alguns dias em um PC comum de 1,6 GHz, portanto seu método é computacionalmente viável.

8.2 – Ataques ao SHA-1

Em 2004, Bruce Schneier e John Kelsey[Kelsey & Schneier] , publicaram um paper onde mostram como reduzir o parâmetro de segurança de resistência a segunda pré-imagem do SHA-1 para 2^{106}. No ínicio de 2005, Rijmen e Oswald[Rijmen & Oswald] publicaram um ataque numa versão reduzida do SHA-1. Eles anunciaram uma colisão no passo 53(fase 3 do algoritmo) ao invés de 80 com o parâmetro de segurança um pouco menor em relação a 2^{80}. Em fevereiro de 2005, uma colisão na etapa 58(antepenúltima iteração da fase 3 do algoritmo) foi anunciada pela equipe liderada por Xiaoyum Wang[Wang, Lisa Yin & Yu] com complexidade 2^{33}, menor que o resultado anteriormente obtido por Rijmen e Oswald. Além disso, previram usando a mesma técnica que uma colisão no SHA-1 poderia ser obtida na ordem de 2^{69} passos. Em Agosto de 2005, na *CRYPTO 2005*[CRYPTO2005], foi divulgada a técnica completa[Wang, Lisa Yin & Yu 2005] em que poderia obter-se uma colisão no SHA-1 na ordem de 2^{69} passos. Na mesma conferência, também foi citado um melhoramento nesse ataque pela mesma pesquisadora junto com Andrew Yao e Frances Yao[Wang, Yao & Yao] cujo palestrante foi Adi Shamir. Eles anunciaram que conseguiriam reduzir à complexidade do ataque a resistência a colisões no SHA-1 para 2^{63} passos caso superassem algumas dificuldades, mas não deram detalhes de como implementá-los. Em ambos os casos nenhuma string binária que gerava colisão no SHA-1 foi divulgada, diferente no caso do MD5, como mostrado na seção anterior.

8.3 – Ataques Práticos

Primeiramente quando os ataques à resistência a colisões do MD5 foram surgindo, muitos pesquisadores viam com certo ceticismo a implicação prática desses resultados. Até que começaram a aparecer trabalhos com sugestões e resultados a vir de encontro as essas premissas.

No trabalho proposto por Ondrej Mikle[Mikle], constrói-se um descompactador e 2 binários compactados com o mesmo MD5, usando os vetores de 1024 bits divulgados no trabalho de Wang [Wang et al]. Quando esses dois binários são descompactados pelo descompactador proposto, o conteúdo de ambos são arquivos diferentes. Nesse mesmo trabalho, ele argumenta que a pessoa responsável pela distribuição do software de uma empresa poderia inserir código malicioso em qualquer software sem ser notado. Ele descreve a situação no qual essa pessoa após receber o software validado, injeta no código do mesmo um dos vetores de 1024 bits do MD5 e o código malicioso. A partir daí é calculado o MD5 de tudo e essa pessoa disponibilizaria esse software para distribuição. Um possível vírus procuraria no computador da vítima esse software e caso achasse, usaria um dos vetores injetados como um booleano que acusaria a presença do código do interesse do vírus ou não. Mikle não dá mais detalhes de como fazê-lo nem exemplifica melhor tal situação.

Aproveitando-se dessa idéia, Dan Kaminski[Kaminski] propõe um exemplo mais detalhado. Sejam x_1 e x_2, conforme mostrados no início desse capítulo e na seção 5.1. Defina x_1 *e* x_2 os vetores obtidos para o MD5 no trabalho de Wang e colaboradores. Seja o código malicioso C. Seja $E_K(C)$ a encriptação de C usando uma cifra de bloco E cuja chave K = SHA-1(x_1). Seja $C_1 = x_1 \,||\, E_K(C)$ e $C_2 = x_2 \,||\, E_K(C)$. Kamisnki exemplifica então criando dois arquivos binários *fire.bin* e *ice.bin* o primeiro contendo apenas C_1 e o segundo contendo apenas C_2. A idéia aqui é que um possível software que contivesse *fire.bin* , facilmente forneceria o código C a um vírus, pois a chave x_1 estaria presente em *fire.bin*, não ocorrendo o mesmo com *ice.bin.* Como já foi citado, tendo em vista MD5(x_1)=MD5(x_2) então MD5($x_1 \,||\, E_K(C)$)=MD5($x_2 \,||\, E_K(C)$), ou seja MD5(C_1)=MD5(C_2) e *fire.bin* e *ice.bin* possuiriam o mesmo MD5.

O mesmo Kaminski conseguiu criar um programa[Confoo] que gera páginas web com o mesmo MD5, baseado na estrutura binária do javascript e nos vetores de 1024 bits obtidos por Wang e colaboradores. Na sua página é possível encontrar uma explicação da idéia de construção desse programa.

Um resultado na mesma linha de Kaminski relativo a paginas web, foi obtido por Magnus Daum e Stefan Lucks[PoisonedPs]. Eles conseguiram gerar dois arquivos postscripts com dizeres diferentes cujo MD5 é o mesmo. Apesar de eles exibirem os arquivos, eles não deram detalhes de como obtê-lo e nem geraram novos postscripts.

Outro resultado extremamente interessante foi apontado por Arjen Lenstra e colaboradores[Lenstra et al]. Eles conseguiram gerar dois *certificados digitais* [CertDigital] contendo chaves públicas diferentes mas com a mesma assinatura da autoridade certificadora. A assinatura do certificado digital é feita em cima de uma função hash customizada(vide seção 4.1.2). No caso, a função usada foi o MD5. Eles calcularam IV´s específicos que se adequavam a cada certificado e a partir daí calcularam dois vetores de 1024 bits que geram colisão no MD5. Com isso, em cada certificado foi injetado um vetor desses gerando colisão na hora da geração da assinatura digital.

8.4 – Prova de conceito dos ataques práticos

Para constatação e prova de conceito do citado nas seções 8.1 e 8.3 foram testadas e desenvolvidas três demonstrações:

1º- Cálculo de uma colisão usando o ataque de Wang e colaboradores;

2º- Desenvolvimento de duas páginas web com o mesmo MD5 usando a ferramenta de Dan Kaminski;

3º- Geração de dois arquivos postscripts com o mesmo MD5.

Abaixo o que foi obtido em cada demonstração:

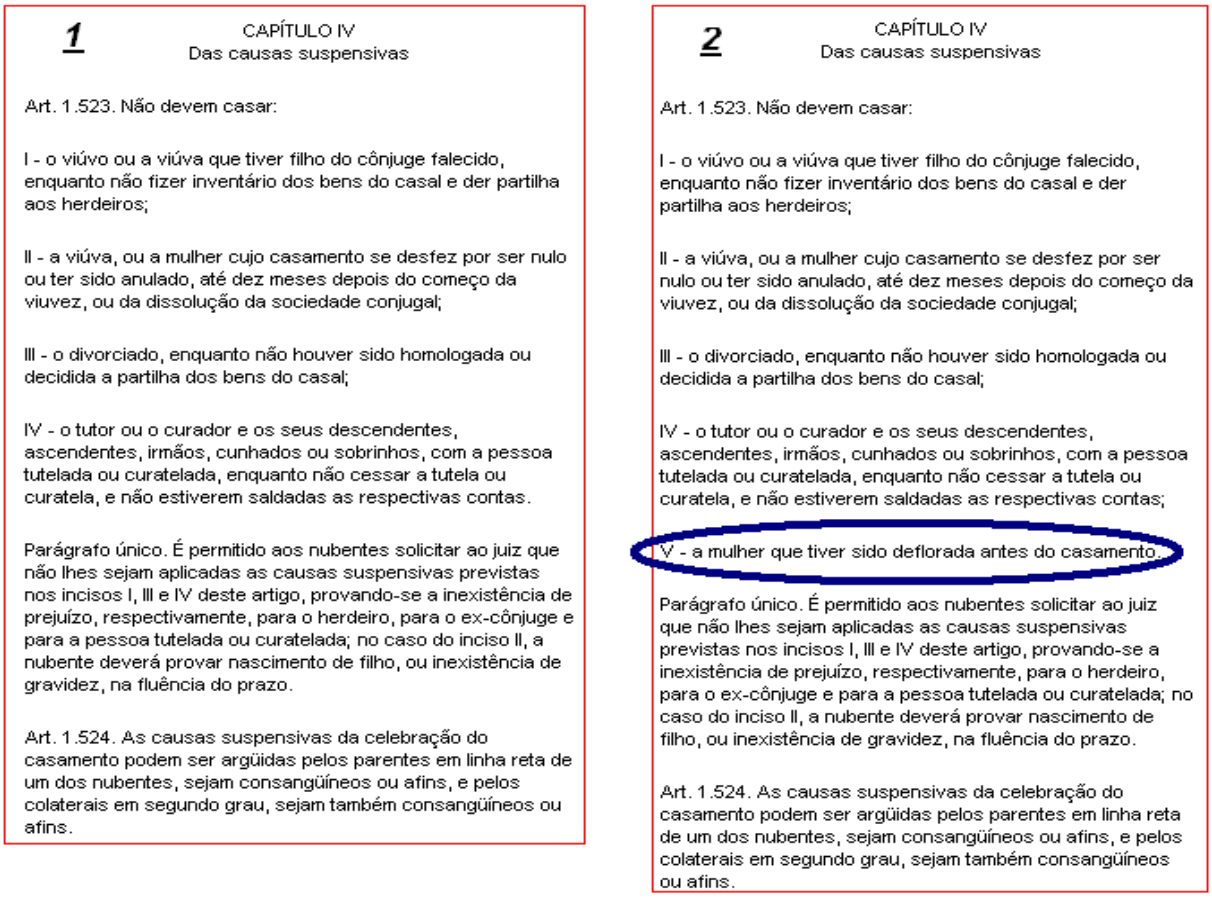
1º- Cálculo de uma colisão usando o ataque de Wang e colaboradores

Para se verificar a eficiência do ataque de Wang e colaboradores foi usado o algoritmo disponível em [Stach & Liu], algoritmo cujo propósito é achar colisões no MD5 usando tal ataque. Os dois vetores de 1024 bits que geram colisão no MD5 foram achados em 28 dias, usando-se um servidor GNU/Linux do Centro de Informática[Cin]

cujo processador é um Pentium® 4 de 2,4 GHz. Isto mostra como é fácil se calcularem colisões no MD5. Os vetores e o valor MD5 deles podem ser vistos no apêndice.

2º Desenvolvimento de duas páginas web com o mesmo MD5 usando a ferramenta de Dan Kaminski;

Para a prova de conceito de que era possível gerar de forma simples duas paginas web com o mesmo MD5 foi usado uma parte da página do novo código civil[Civil]. As páginas geradas são *codigo_civil_1.html* e *código_civil_2.html*. A diferença entre elas é nada mais nada menos do que um parágrafo a mais em uma das páginas. A página com o parágrafo a mais contém a seguinte frase: ***"V - a mulher que foi deflorada no casamento"***. A figura abaixo mostra um trecho da página onde a modificação é visível:



1

CAPÍTULO IV
Das causas suspensivas

Art. 1.523. Não devem casar:

I - o viúvo ou a viúva que tiver filho do cônjuge falecido, enquanto não fizer inventário dos bens do casal e der partilha aos herdeiros;

II - a viúva, ou a mulher cujo casamento se desfez por ser nulo ou ter sido anulado, até dez meses depois do começo da viuvez, ou da dissolução da sociedade conjugal;

III - o divorciado, enquanto não houver sido homologada ou decidida a partilha dos bens do casal;

IV - o tutor ou o curador e os seus descendentes, ascendentes, irmãos, cunhados ou sobrinhos, com a pessoa tutelada ou curatelada, enquanto não cessar a tutela ou curatela, e não estiverem saldadas as respectivas contas.

Parágrafo único. É permitido aos nubentes solicitar ao juiz que não lhes sejam aplicadas as causas suspensivas previstas nos incisos I, III e IV deste artigo, provando-se a inexistência de prejuízo, respectivamente, para o herdeiro, para o ex-cônjuge e para a pessoa tutelada ou curatelada; no caso do inciso II, a nubente deverá provar nascimento de filho, ou inexistência de gravidez, na fluência do prazo.

Art. 1.524. As causas suspensivas da celebração do casamento podem ser argüidas pelos parentes em linha reta de um dos nubentes, sejam consangüíneos ou afins, e pelos colaterais em segundo grau, sejam também consangüíneos ou afins.

2

CAPÍTULO IV
Das causas suspensivas

Art. 1.523. Não devem casar:

I - o viúvo ou a viúva que tiver filho do cônjuge falecido, enquanto não fizer inventário dos bens do casal e der partilha aos herdeiros;

II - a viúva, ou a mulher cujo casamento se desfez por ser nulo ou ter sido anulado, até dez meses depois do começo da viuvez, ou da dissolução da sociedade conjugal;

III - o divorciado, enquanto não houver sido homologada ou decidida a partilha dos bens do casal;

IV - o tutor ou o curador e os seus descendentes, ascendentes, irmãos, cunhados ou sobrinhos, com a pessoa tutelada ou curatelada, enquanto não cessar a tutela ou curatela, e não estiverem saldadas as respectivas contas;

V - a mulher que tiver sido deflorada antes do casamento.

Parágrafo único. É permitido aos nubentes solicitar ao juiz que não lhes sejam aplicadas as causas suspensivas previstas nos incisos I, III e IV deste artigo, provando-se a inexistência de prejuízo, respectivamente, para o herdeiro, para o ex-cônjuge e para a pessoa tutelada ou curatelada; no caso do inciso II, a nubente deverá provar nascimento de filho, ou inexistência de gravidez, na fluência do prazo.

Art. 1.524. As causas suspensivas da celebração do casamento podem ser argüidas pelos parentes em linha reta de um dos nubentes, sejam consangüíneos ou afins, e pelos colaterais em segundo grau, sejam também consangüíneos ou afins.

Figura 13 – Duas páginas web diferentes com o mesmo MD5.

As duas páginas possuem **MD5 = 48b86a444bef6a806b8cbf30a510c61f** (calculado utilizando o utilitário *md5sum* do Unix). As duas páginas podem ser vistas em [Exemplos] (recomenda-se usar a versão 6.0 para XP® SP.x em diante do Internet Explorer®).

3º- Geração de dois arquivos postscripts com o mesmo MD5

Como prova de conceito da geração de postscripts, foi usado um modelo de procuração de movimentação de conta bancaria, segundo descrição encontrada em [Procuracao]. Foram criados dois arquivos, a saber, *ProcuracaoConta1.ps* e *ProcuracaoConta2.ps*. A figura abaixo ilustra o conteúdo desses dois postscripts:

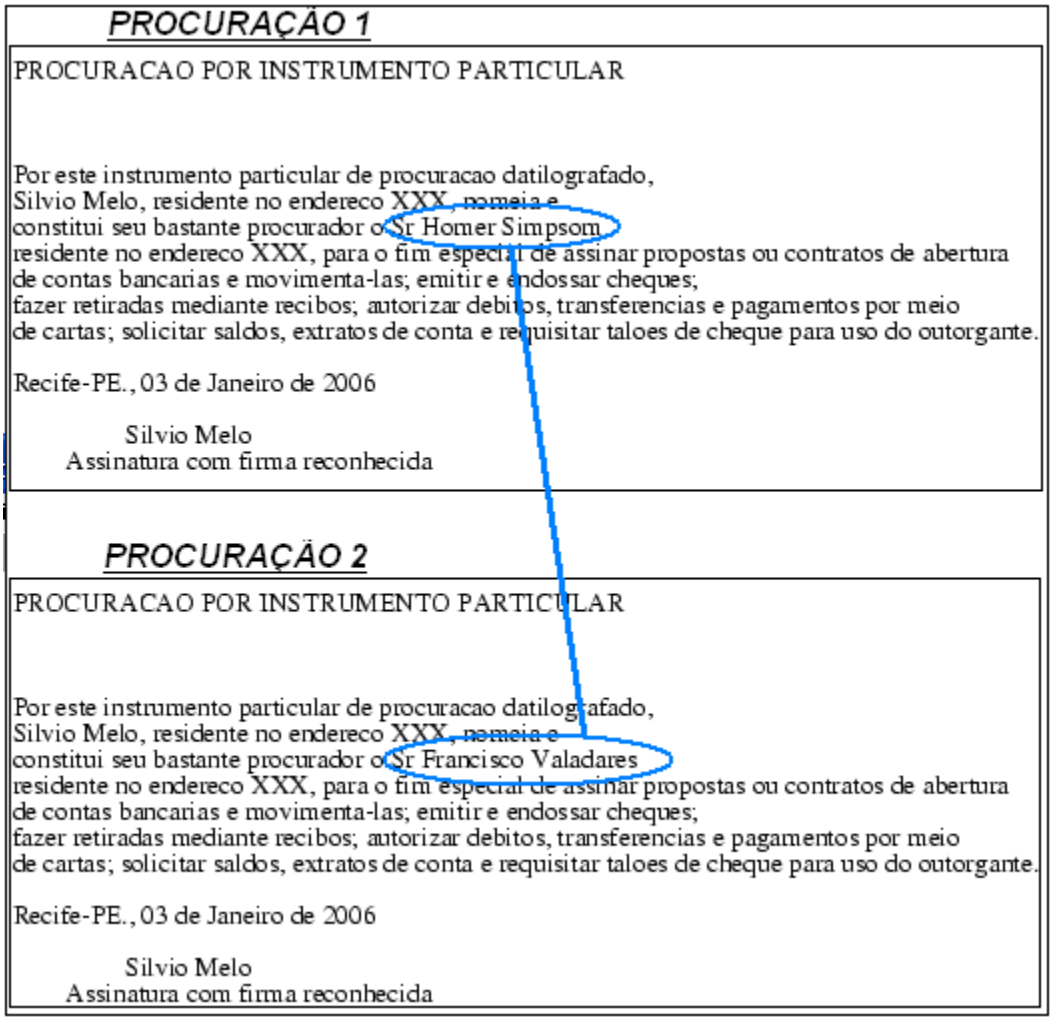


Figura 14 – Dois arquivos postscripts diferentes com o mesmo MD5.

As partes circuladas ilustram exatamente a única diferença entre os dois postscripts. As partes completadas com “XXX” não foram preenchidas apenas por uma questão de conveniência todavia poderiam ser completadas com qualquer endereço. Os dois arquivos possuem **b00b750c2ea5a830293dfd01402f3baa**(calculado utilizando o utilitário md5sum do Unix) como resultado do MD5. Eles podem ser baixados em [Exemplos].

9 - Análise Crítica

9.1 – Porque os resultados do capítulo 8 são ruins

Os ataques discutidos na seção 8.3 e as provas de conceito obtidas na seção 8.4 estão de acordo com os interesses do adversário mostrados na seção 4.1. Além disso, obtê-los foi computacionalmente viável o que possibilita um ataque prático.

Como já foi mostrado, achar colisões no MD5 é extremamente viável mesmo para qualquer IV´s. Isto é a base de todos os outros ataques.

Em relação à idéia de infecção de software proposta por Mikle e Kamisnki, não parece algo de impacto prático verdadeiro, tendo em vista que o atacante poderia infectar o software e calcular o MD5 depois sem precisar injetar vetor nenhum. Todavia, se a injeção dos vetores é possível, isso já é um grande motivo para não usar mais o MD5, haja vista o desejo dos desenvolvedores de software em se evitar fluxos e situações inesperadas e difíceis de remediar. Softwares que checam a integridade de arquivos de um sistema operacional usando o MD5 poderiam ser burlados utilizando-se da mesma idéia da injeção dos vetores discutida por Mikle e Kaminski. Alguns exemplos de softwares dessa natureza são o Tripware[Tripware], Integrit[Integrit] e o Dragon Squire[Dragon].

A obtenção de duas páginas web e dois arquivos postscripts com o mesmo MD5, fere o principio da integridade e além do mais são extremamente perigosos para serem usados em esquemas onde a assinatura digital está presente.

Um exemplo interessante para o caso das páginas web seriam prontuários médicos voltados para a web. Esses prontuários geralmente estão escritos em código orientado a web como, por exemplo, html, javascript e xml. A assinatura digital do médico no prontuário web seria na verdade uma assinatura sobre o código utilizado. Esta assinatura garantiria a integridade desse prontuário. Imaginando que a função hash

utilizada na assinatura seja o MD5, a prova de conceito fornecida na seção 8.4 seria totalmente aplicável neste contexto. Uma possível situação para se aplicar esse ataque seria a acusação de erro médico. A ligação entre o erro e o médico estaria exatamente na assinatura digital do médico sobre aquele prontuário com o diagnostico errado. O médico então assim que emite o prontuário original aplica o ataque ao MD5 e injeta os vetores que geram colisão. Caso o prontuário original esteja com um erro(sendo ele acusado de erro médico por causa disso), o médico desonesto faz uma alteração no prontuário com o diagnostico correto, obtendo-se um novo código cujo resultado da assinatura ainda é o mesmo. Uma discussão sobre prontuários web pode ser vista em [Prontuario]

Quanto ao caso dos dois postscripts, uma situação preocupante seria num cartório totalmente informatizado. Supondo que *Silvio Melo* queira fornecer uma procuração digital para *Homer Simpson*, dando-lhe poderes totais sobre a sua conta bancária a *Homer*. *Silvio* então pede a sua secretária a emissão dessa procuração digital. A secretária desonesta gera essa procuração valendo-se dos ataques ao MD5 e injeta os vetores que geram colisão na procuração. Então ela entrega essa procuração a Sílvio, que a assina digitalmente. Sílvio coloca a procuração juntamente com sua assinatura digital(usada junto com o MD5) num disquete e a entrega a seu office-boy, *Francisco*, para ser autenticada num cartório. *Francisco* fez um acordo com a secretária desonesta, portanto ele abre esse arquivo e faz as alterações necessárias para que a procuração passe poderes para ele mesmo movimentar a conta de *Silvio Melo*. Após isso, ele salva o novo arquivo e a assinatura antiga no disquete, apaga o antigo postscript e leva o disquete ao cartório. A única coisa que o cartório faz é verificar a assinatura de Silvio através de sua chave pública. Como a assinatura de *Silvio* estará OK, Francisco consegue a autenticação do documento, dando plenos poderes a ele. Isto é possível porque o novo documento que *Francisco* gerou, possui o mesmo MD5 que o documento que *Silvio* emitiu e como a assinatura é feita na verdade sobre o valor do hash do documento(vide seção 4.1.2, U_1 seria *Silvio Melo* e U_2 seria o cartório), na prática, o novo documento funciona como o documento que *Silvio* assinou.

9.2 – Como criar arquivos Postscript com o mesmo MD5

Postscript na verdade é uma linguagem de programação com o intuito de descrever qualquer página para uma impressora. Essa linguagem é *interpretada*. O

interpretador recebe um conjunto de instruções e a partir desses comandos exibe o resultado na tela. Essas instruções são repassadas ao interpretador em *pós-ordem.*

O ataque proposto pelos pesquisadores se vale da seguinte estrutura básica:

(i) - (X_1) (X_2) eq {C_1} {C_2} ifelse

X_1 e X_2 são strings e *eq*, *ifelse*, C_1 e C_2 são instruções da linguagem postscript.

O interpretador ao ler **(i),** primeiro verifica se X_1 e X_2 são iguais(instrução *eq*). Caso afirmativo, ele executará C_1 e em caso negativo, executará C_2 (instrução *ifelse*).

Com isso, fica fácil construir dois arquivos com o mesmo MD5 conforme mostrado abaixo:

Qualquer arquivo postscript começa com um cabeçalho seguido de instruções. Esse cabeçalho geralmente possui menos de 512 bits. Caso ele possua 512 bits calcula-se o MD5 do cabeçalho. Caso possua menos de 512 bits ou mais de 512 bits, preenche-se com caracteres ascii irrelevantes para o arquivo, a partir da posição logo após o cabeçalho. Isto é feito até que o cabeçalho mais o ascii e o caracter "(" possuam um número de bits múltiplo de 512 bits(esse ultimo caracter é o parênteses que inicia o X_1) Após esse processo, calcula-se o MD5 do cabeçalho mais o ascii e o caracter "(". O resultado desse cálculo será o *IV* usado no cálculo dos dois vetores de 1024 bits que colidem para o MD5, usando o ataque de Wang ou o de Klima . Sejam *VETOR 1* e *VETOR 2* esses vetores. Tanto no caso do resultado de Daum e Lucks, como nos postscripts apresentados neste trabalho, o cabeçalho possui menos de 512 bits.

Sejam *arquivo1* e *arquivo2*, arquivos postscripts. Seja o *CONJUNTO DE INSTRUÇÕES 1*, conjunto que descreve como o *arquivo1* deve aparecer e *CONJUNTO DE INSTRUÇÕES 2* idem para o *arquivo2*. Então a estrutura básica do *arquivo1* poderia ser dessa forma:

(*VETOR 2*) (*VETOR 2*) eq { *CONJUNTO DE INSTRUÇÕES 1*} { *CONJUNTO DE INSTRUÇÕES 2*} ifelse

E a estrutura básica do *arquivo2* seria dessa forma:

(*VETOR 1*) (*VETOR 2*) eq { *CONJUNTO DE INSTRUÇÕES 1*} { *CONJUNTO DE INSTRUÇÕES 2*} ifelse

Como fazer MD5(*arquivo1*) = MD5(*arquivo2*) ? Basta fazer o *arquivo1* e o *arquivo2* conterem o cabeçalho mais o ascii e o caracter "(" usados para calcular *VETOR 1* e *VETOR 2*. As duas figuras abaixo mostram a estrutura do *arquivo2* e do *arquivo1* respectivamente(A instrução da linguagem postscript *showpage* é para exibir o resultado na saída).

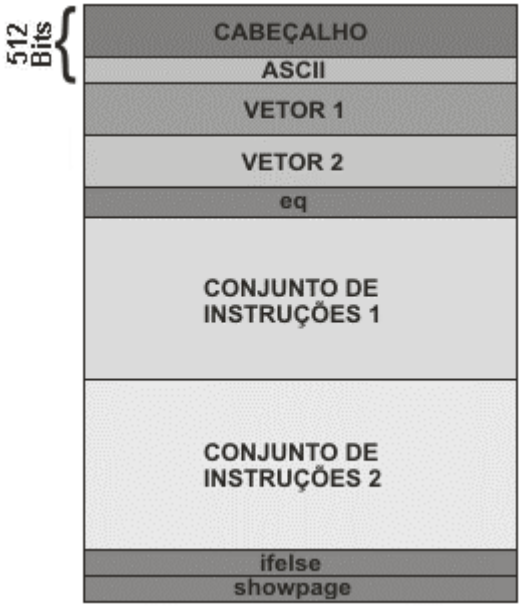


Figura 15 – Estrutura do arquivo 2.

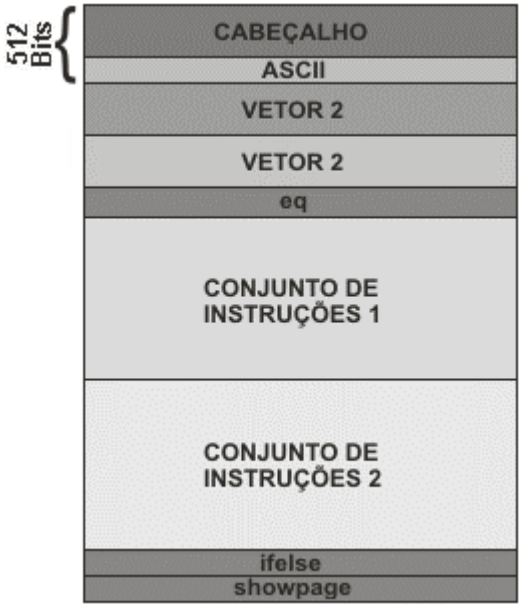


Figura 16 – Estrutura do arquivo 1.

Já foi citado na seção 5.1 e no início do capítulo 8, uma colisão ocorre em H, H(x_1) = H(x_2) com x_1 e x_2 distintos e de mesmo tamanho, então para três strings binárias quaisquer *p,q e t*, H(*q* || x_1 || *p*) = H(*q* || x_2 || *p*) . O H é o MD5, *q* funciona como cabeçalho mais o ascii, x_1 e x_2 como *VETOR 1* e *VETOR 2* respectivamente e o *p* funciona como o resto tanto do *arquivo1* como do *arquivo2*. Por esse fato MD5(*arquivo1*) = MD5(*arquivo2*).

Foi usando exatamente essas idéias que Daum e Lucks montaram os dois arquivos postscripts com o mesmo MD5. Os arquivos postscripts usados como prova de conceito deste trabalho também foram construídos valendo-se dessas idéias. É mister levar em conta que Daum e Lucks não descreveram como obter esses dois postscripts, portanto a descrição mostrada aqui serve de contribuição ao estudo das conseqüências das colisões ao MD5.

9.3 – Uso do MD5

Pelos resultados obtidos pelos pesquisadores e pelas provas de conceitos mostradas neste trabalho fica claro que o uso do algoritmo MD5 pode ser bastante perigoso e inesperado. Colisões são achadas em tempo computacional aceitável e o seu uso em linguagens declarativas que possuem operadores condicionais pode ser

inacreditavelmente prático como a linguagem Postscript e a linguagem Html. Tentar circunscrever os limites de execução de um determinado aplicativo ou esquema com o MD5 incluído pode ser bastante enfadonho e foge da filosofia da engenharia de software. O próprio NIST desaconselha o seu uso.Tendo em vista tudo isso, não é recomendável mais o seu uso sob nenhuma circunstância.

9.4 – Uso do SHA-1

O parâmetro de ataque 2^{63} obtido pelos pesquisadores, ainda não é um parâmetro bom para um adversário prático. Por exemplo, mesmo um computador executando 1 Mega SHA-1's por segundo(1024^2 execuções é uma superestimativa, serve para se ter uma idéia do limite superior de tempo), levaria 2^{43} segundos para se achar uma colisão e como um ano possui aproximadamente 1 Giga segundos, uma colisão seria achada em 8192 anos. Logicamente que o uso de computadores distribuídos e o uso da computação paralela reduziriam drasticamente o tempo necessário, mas mesmo assim os recursos mobilizados seriam enormes. Como se pode constatar, diferentemente do MD5, ainda não é o caso de se abolir o uso do SHA-1.

Em vários outros casos de ataques às funções hash, os primeiros ataques foram um prenúncio de ataques computacionalmente viáveis advindos mais adiante. Tendo em vista essa tendência, o NIST prevê a aposentadoria do SHA-1 para o ano de 2010, isso se não surgir até lá ataques computacionalmente viáveis. Ele recomenda após esse prazo a migração para versões consideradas mais fortes da família SHA, como o *SHA-256*, o *SHA-384* e o *SHA-512*(detalhes de cada um podem ser visto em [FIPS180-2]). Acredita-se que sejam mais fortes que o SHA-1 devido dentre muitos fatores, ao tamanho da saída(256,384 e 512 bits respectivamente), resultando no aumento do parâmetro de segurança em relação ao SHA-1. Aparentemente é uma boa estratégia a ser adotada.

9.5 – Uma Proposta de Construção

Uma da idéias iniciais propostas pelos pesquisadores para contornar possíveis ataques à resistência a colisões nas funções hash sob o paradigma Merkle-Damgard, era realizar uma construção *em cascata*. Como exemplo de tal construção, sendo H_1 e H_2

funções hash sob o paradigma Merkle-Damgard de contradomínios $\{0,1\}^m - \{\varepsilon\}$ e $\{0,1\}^n - \{\varepsilon\}$ respectivamente e *M* uma string binária, a nova função hash H construída em cascata será:

$$H(M) = H_1(M) \,||\, H_2(M)$$

Era de se esperar que H possuísse como parâmetro de segurança em relação à resistência a colisões, $(2^{(m+n)})^{1/2}$. Assim, se H(M) = MD5(M) || SHA-1(M) seu parâmetro de segurança em relação à resistência a colisões seria 2^{164}, o que resolveria o problema de desenvolvimento de uma nova função hash para ataques computacionalmente viáveis no tocante a resistência a colisões, como no caso referenciado para o MD5 no capítulo 8. Surpreendentemente isso não é verdade, o parâmetro de segurança é da ordem apenas da função hash de maior contradomínio, ou seja, neste caso seria 2^{80}, contradomínio do SHA-1. Este resultado foi apresentado na CRYPTO 2004 por Antoine Joux. No seu trabalho, ainda há uma maneira de se obterem multicolisões para uma mesma função hash sob o paradigma Merkle-Damgard, caso apenas algumas colisões sejam obtidas na função de compressão. Isto lançou por terra a idéia da função hash em cascata e pôs em cheque o paradigma de construção MD.

Há pouco tempo atrás, um trabalho divulgado por alguns pesquisadores[GNDV2006], objetiva resolver esse problema da multicolisão, sugerindo algumas mudanças na função de compressão para que o design de segurança obtido a partir do paradigma Merkle-Damgard ainda possa ser usado. Nesse mesmo trabalho, há uma proposta de uma função hash conhecida com *3C+*, provada por esses pesquisadores ser imune a multicolisões. Resta saber se essas mudanças serão acatadas pela comunidade.

9.6 – CAM e HMAC

Suponha o esquema descrito na seção 4.2. O interesse do adversário seria criar novas strings válidas. Suponha que o primeiro CAM descrito na seção 3.1.2 seja usado neste esquema e H seja a função SHA-1 com M uma string binária de 256 bits, ou seja, *Tag* = SHA-1(*K* || *M*). Seja *M'* o resultado do padding e da concatenação do tamanho de *M* em *M*. Assim SHA-1(*K* || *M*) = SHA-1(*K* || *M'*) = SHA-1(*K* || *M* || 1000...000 ||

0...0101010000) = *Tag* com 1000...000 possuindo 112 bits e 0...0101010000 possuindo 64 bits. Suponha que o adversário intercepte *M* e *Tag*. Seja *f* a função de compressão do SHA-1. O adversário poderá criar uma nova mensagem *N*, se *N* = *M'* || *J*, sendo *J* uma string binária arbitrária. Para isso, basta o adversário calcular *padd*, *tamanho* e *f* (*Tag*, *J* || *padd* || *tamanho*), sendo *padd* e *tamanho* strings binárias que representam o padding de *J* e o tamanho de *J*, respectivamente. Isto se deve ao fato de na verdade *f* (*Tag*, *J* || *padd* || *tamanho*) = *f* (*f* (IV, *K* || *M* || 1000...000 || 0...0101010000) , *J* || *padd* || *tamanho*) = SHA-1(*K* || *M'* || *J*) = SHA-1(*K* || *N*). Então o adversário manda *N* e *f* (*Tag*, *J* || *padd* || *tamanho*) como sua respectiva *Tag para* U_2. O algoritmo *VF* de U_2 irá validar com um *sim*, apesar do adversário não conhecer a chave. Por causa desse tipo de ataque, esse tipo de algoritmo não é usado. Os adotados como padrão são o HMAC-MD5 e o HMAC –SHA-1.

Apesar dos resultados recentes em relação às colisões obtidas no MD5 e no SHA-1, como mesmo cita [Hmac], sua segurança está mais baseada na propriedade do MD5 e do SHA-1 serem resistentes à segunda pré-imagem e a chave ser grande, aleatória e secreta do que eles serem resistentes a colisões. Além do mais, a mais recente recomendação da *ECRYPT*[ECRYPT1.1], entidade européia responsável pela avaliação da criptografia usada na Europa, mostra que o uso dos HMAC´s MD5 e SHA-1 só deveriam ser abolidos caso surjam ataques que achem colisões em tempo computacionalmente viável para IV´s aleatórios e secretos. Mas os ataques recentes não vão nessa direção, portanto não há motivo para deixar de usá-los, por enquanto.

9.7 – Porque os ataques ao MD5 e ao SHA-1 foram possíveis

Como já foi citado no capítulo 7, tanto o MD5 como o SHA-1 são baseados em cifras de bloco sob a égide do paradigma Merkle-Damgard. Na época de suas construções(ambos em 1992), os paradigmas vigentes de construção de cifras de bloco baseavam-se principalmente nas construções sobre as funções de Feistel, como por exemplo as redes de Feistel balanceadas e não-balanceadas, isso é tão verdadeiro que tanto o MD5 quanto o SHA-1 podem ser enxergados de certa forma como uma Rede de Feistel não-balanceada heterogênea[UnbFeistelnet].

Na ultima década até os dias atuais, estas funções estiveram sob fogo cruzado de vários criptoanalistas espalhados pelo mundo inteiro a citar: Eli Biham, Adi Shamir, Joux, Dobbertin, R. Chen, Chabaud, Wang, Yu, Yin, Feng, H. Chen, Paddon, Hawkes, Rose, Van Rompay, Lai, Preneel, Biryukov, Vandewalle dentre outros. Um grande exemplo de uma cifra de bloco construída sob o paradigma vigente da época que veio a sucumbir diante desses ataques foi o *DES*[DES]. O DES era um algoritmo bastante difundido e usado. Ataques diferenciais e lineares computacionalmente viáveis contra o DES foram suficientes para derrubá-lo.

No próprio trabalho de Wang e Yu[Wang & Yu], encontram-se elementos destes tipos de ataques já bastante discutidos pela comunidade. Portanto não é leviano afirmar que os paradigmas usados para o design e arquitetura do MD5 e do SHA-1 em relação as cifras de blocos estivessem ultrapassados, embora não se encontre muita coisa nesta linha de defesa na comunidade da área. Talvez seja mais lógico analisar outras estratégias mais atuais de construção de cifras de bloco e não apenas propor versões mais fortes de algoritmos da mesma família, conforme mencionado na seção 9.4.

Um caminho nesse sentido seria a estratégia da trilha larga[WideTrail]. Essa estratégia fundamentou inclusive a construção do *AES*[AES], substituto do DES. Um exemplo de função hash criptográfica construída sob esta estratégia é o *Whirlpool*[Whirlpool]. Essa função hash não é tão usada por ser considerada mais lenta que o MD5 e o SHA-1. Atualmente nenhum ataque computacionalmente viável explorando a estratégia da trilha larga foi apresentado, não afetando assim nem o AES nem o Whirlpool no tocante a construção usando essa estratégia.

Em suma, a questão dos ataques ao MD5 e ao SHA-1 aparenta ter uma problemática mais profunda e sutil que simplesmente estender o tamanho da saída para se ter uma boa margem no parâmetro de segurança como propôs o NIST e deve ser levada em consideração.

10 - Conclusões

Este trabalho tentou entender exatamente as implicações práticas dos ataques às funções hash MD5 e SHA-1 e o porquê desses ataques em relação à arquitetura dos mesmos. Analisar todas as aplicações e esquemas que fazem uso deles seria surreal devido ao tempo levado para elaborar esta análise. Por isso esse trabalho tentou analisar o mínimo possível, todavia que tivessem a importância e relevância consideráveis, como no caso das assinaturas digitais.

A análise crítica do capítulo 9 é o ápice desse intuito e prova realmente que esses objetivos foram alcançados com êxito, tendo em vista a capacidade crítica de compreensão de uso prático desses algoritmos nas aplicações e nos esquemas criptográficos pelo leitor deste documento, fornecidos pela leitura e compreensão deste documento.

No decorrer da finalização deste trabalho, novos ataques bem mais eficientes surgiram contra o MD5, ataques capazes de encontrar colisões em questão de segundos. Isto vem dar crédito a afirmação de que o MD5 não deve ser mais usado conforme analisado no capítulo 9. As referências destes ataques podem ser vistas em [Stevens2006] e em [Klima2006].

10.1 - Dificuldades Encontradas

Como a abordagem de construção a funções hash customizadas é feita de forma heurística, envolvem muitos detalhes cuja compreensão não é nada trivial. Essa foi a maior dificuldade de elaboração desse trabalho. Um outro problema foi analisar os arquivos postscripts fornecidos na internet pelos pesquisadores e a partir daí fazer a engenharia reversa necessária para compreender e gerar os arquivos postscripts apresentados neste trabalho. Isso se deve à não explicação de como os postscripts originais foram obtidos.

10.2 - Trabalhos Futuros

Os objetivos secundários não foram alcançados devido ao tempo necessário para uma análise extensa da gama de propostas e geração de uma nova proposta. Esse seria um bom trabalho para se fazer no futuro.

Referências Bibliográficas

[Rivest90] R.L. Rivest, *The MD4 Message Digest Algorithm*, RFC 1186, Out 1990.

[Rivest91] R.L. Rivest, *The MD4 Message Digest Algorithm*, Advances in Cryptology-CRYPTO '90 Proceedings, Springer-Verlag, 1991, pág. 303-311**.**

[Rivest92] R.L. Rivest, *The MD5 Message Digest Algorithm*, RFC 1321, Abr 1992.

[den Boer & Bosselaers] B. den Boer and A. Bosselaers, *An Attack on the Last Two Rounds of MD4*, Advances in Cryptology-CRYPTO '91 Proceedings, Springer-Verlag, 1992, pág. 194-203.

[Biham92] E. Biham, *On the Applicability of Differential Cryptanalysis to Hash Functions*, lecture at EIES Workshop on Cryptographic Hash Functions, Mar 1992.

[FIPS180-2] National Institute of Standards and Technology, NIST FIPS PUB 180-2, *Secure Hash Standard*, U.S. Department of Commerce, Ago 2002.

[RFC4270] RFC 4270 - *Attacks on Cryptographic Hashes in Internet Protocols,* Disponível em: ***http://www.ietf.org/rfc/rfc4270.txt***. Ùltimo acesso em 01 de Dezembro de 2005**.**

[Schneier] Bruce Schneier, *Applied Cryptography: protocols, algorithms and source code in C*, Second Edition, New York: John Wiley, 1996.

[LittleEnd] Wikipedia, *Little Endian,* Disponível em: ***http://pt.wikipedia.org/wiki/Little_endian***. Ùltimo acesso em 02 de Janeiro de 2006**.**

[FIPS] *Proposed Federal Information Processing Standard for Secure Hash Standard*, Federal Register, vol. 57, n. 21, 31 Jan 1992, pág. 3747-3749.

[NIST] NIST- National Institute of Standarts and Technology*,* Disponível em: ***http://www.itl.nist.gov*****.** Ùltimo acesso em 02 de Janeiro de 2006.

[NSA] NSA- National Security Agency, Disponível em***: http://www.nsa.gov***. Ùltimo acesso em 02 de Janeiro de 2006.

[RFC2403] RFC 2403 - *The Use of HMAC-MD5-96 within ESP and AH* , Disponível em**:** ***http://www.rfc-editor.org/rfc/rfc2403.txt***. Ùltimo acesso em 02 de Janeiro de 2006.

[RFC2404] RFC 2404 -*The Use of HMAC-SHA-1-96 within ESP and AH*, Disponível em**:** ***http://www.rfc-editor.org/rfc/rfc2404.txt***. Ùltimo acesso em 02 de Janeiro de 2006.

[TLs] TLS - *Transport Layer Security*, Disponível em**:** ***http://www.ietf.org/html.charters/tls-charter.html***. Ultimo acesso em 02 de Janeiro de 2006.

[IPSec] Wikipedia, IPSec – *Internet Protocol Security*, Disponível em**:** ***http://en.wikipedia.org/wiki/IPsec****.* Ultimo acesso em 02 de Janeiro de 2006.

[Modern1] *Introduction to Modern Cryptography*, Disponível em***:*** ***http://www-cse.ucsd.edu/~mihir/cse107/w-prf.pdf***. Ultimo acesso em 02 de Janeiro de 2006.

[Modern2] *Introduction to Modern Cryptography*, Disponível em**:** ***http://www-cse.ucsd.edu/~mihir/cse107/w-birthday.pdf***. Ultimo acesso em 02 de Janeiro de 2006.

[Goldreich et al] O. Goldreich, S. Goldwasser e S. Micali. *How to construct random functions*. Journal of the ACM, Vol. 33, No. 4, 1986, pág. 210-217.

[LubyRackoff] M. Luby e C. Rackoff. *How to construct pseudorandom permutations from pseudorandom functions*. SIAM J. Comput, Vol. 17, No. 2, Abr 1988.

[Zoo] *Complexity Zoo*, Disponível em: ***http://qwiki.caltech.edu/wiki/Complexity_Zoo#bpp.*** Ultimo acesso em 02 de Janeiro de 2006.

[ProvSec] Wikipedia, *Provable Security*, Disponível em: ***http://en.wikipedia.org/wiki/Provable_security****.* Ultimo acesso em 02 de Janeiro de 2006**.**

[ConcSec] Wikipedia, *Concrete Security*, Disponível em: ***http://en.wikipedia.org/wiki/Concrete_security.*** Ultimo acesso em 02 de Janeiro de 2006.

[Shannon] Claude E. Shannon, *Communication Theory of Secrecy Systems*, Disponível em: ***http://www.cs.ucla.edu/~jkong/research/security/shannon****.* Ultimo acesso em 02 de Janeiro de 2006.

[BCip] Wikipedia, *Block Cipher*, Disponível em: ***http://en.wikipedia.org/wiki/Block_cipher****.* Ultimo acesso em 02 de Janeiro de 2006.

[LinCrypt] Wikipedia, *Linear Cryptanalysis*, Disponível em: ***http://en.wikipedia.org/wiki/Linear_cryptanalysis****.* Ultimo acesso em 02 de Janeiro de 2006.

[DifCrypt] Wikipedia, *Differential Cryptanalysis*, Disponível em: ***http://en.wikipedia.org/wiki/Differential_cryptanalysis****.* Ultimo acesso em 02 de Janeiro de 2006.

[PkC] Wikipedia, *Public-Key Cryptography*, Disponível em: ***http://en.wikipedia.org/wiki/Public_key****.* Ultimo acesso em 02 de Janeiro de 2006.

[Hmac] *Message Authentication using hash functions - The HMAC Construction*, Disponível em: ***http://www-cse.ucsd.edu/users/mihir/papers/hmac.html#hmac-cryptobytes****.* Ultimo acesso em 02 de Janeiro de 2006.

[Merkle] Ralph Merkle. *One Way Hash Functions and DES*. Advances in Cryptology - Crypto'89, Lecture Notes in Computer Sciences, Vol. 435, Springer-Verlag, pág. 428-446, 1989.

[Damgard] Ivan Damgard. *A design principle for hash functions*. Proc. of CRYPTO 89, pág. 416--427, 1989.

[Simon98] Dan Simon, *Finding collisions on a one-way street: can secure hash functions be based on general assumptions?*, Advances in Cryptology- Eurocrypt '98, pág. 334-345, 1998.

[PGV] B. Preneel, R. Govaerts, and J. Vandewalle. *Hash functions based on block ciphers: A synthetic approach.* Advances in Cryptology - Crypto'93, Lecture Notes in Computer Sciences, Springer-Verlag, pág. 368-378, 1994.

[DaviesMeyer] Wikipedia, *Hash Functions Based in Block Ciphers*, Disponível em***: http://en.wikipedia.org/wiki/Hash_functions_based_on_block_ciphers***.* Ultimo acesso em 02 de Janeiro de 2006.

[Rogaway et al] Phil Rogaway, John Black e Tom Shrimpton. *Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV***.** Advances in Cryptology - Crypto'02, Lecture Notes in Computer Sciences, Vol. 2442, Springer-Verlag, pág. 320-335, 2002.

[Webster & Tavares] A. F. Webster e S. E. Tavares. *On the design of S-boxes***.** Advances in Cryptology - Crypto'85, Springer-Verlag, Berlim, pág. 523-534, 1986.

[Transp] Wikipedia, *Transposition Cipher*, Disponível em: ***http://en.wikipedia.org/wiki/Transposition_cipher****.* Ultimo acesso em 02 de Janeiro de 2006.

[Subst] Wikipedia, *Substituition Cipher*, Disponível em: ***http://en.wikipedia.org/wiki/Substitution_cipher.*** Ultimo acesso em 02 de Janeiro de 2006.

[ProdCip] Wikipedia, *Product Cipher*, Disponível em: ***http://en.wikipedia.org/wiki/Product_cipher.*** Ultimo acesso em 02 de Janeiro de 2006.

[SubstBox] Wikipedia, *Substituition Box*, Disponível em***: http://en.wikipedia.org/wiki/S-box****.* Ultimo acesso em 02 de Janeiro de 2006.

[SubstPermNet] Wikipedia, *Substituition-Permutation Network*, Disponível em: ***http://en.wikipedia.org/wiki/Substituition-permutation_network*** . Ultimo acesso em 02 de Janeiro de 2006.

[Feistelnet] Wikipedia, *Feistel Cipher*, Disponível em: ***http://en.wikipedia.org/wiki/Feistel_network*** . Ultimo acesso em 02 de Janeiro de 2006.

[UnbFeistelnet] *Unbalanced Feistel Network and Block-Cipher Design*, Disponível em: ***http://www.schneier.com/paper-unbalanced-feistel.pdf****.* Ultimo acesso em 02 de Janeiro de 2006.

[WideTrail] *The Wide Trail Design Strategy*, Disponível em: ***http://www.iaik.tugraz.at/aboutus/people/rijmen/ima.pdf***. Ultimo acesso em 02 de Janeiro de 2006.
[Boer & Bosselaers 93] Bert den Boer and Antoon Bosselaers, *Collisions for the Compression Function of MD5*, EUROCRYPT 1993, pág. 293–304.

[Dobbertin96] Hans Dobbertin, *Cryptanalysis of MD5 compress,* Announcement on Internet, Maio 1996.

[MD5CRK] Wikipedia, *MD5CRK*, Disponível em: ***http://en.wikipedia.org/wiki/MD5CRK***. Ultimo acesso em 02 de Janeiro de 2006.

[CRYPTO2004] CRYPTO 2004, Disponível em: ***http://www.iacr.org/conferences/crypto2004***. Ultimo acesso em 02 de Janeiro de 2006.

[Wang et al] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu, *Collisions for hash Functions MD4, MD5, HAVAL-128 and RIPEMD,* RUMP SESSION, CRYPTO 2004.

[EUROCRYPT2005] EUROCRYPT2005, Disponível em: ***http://www.brics.dk/eurocrypt05***. Ultimo acesso em 02 de Janeiro de 2006.

[Wang & Yu] Xiaoyun Wang e Hongbo Yu, *How to Break MD5 and Other Hash Functions,* EUROCRYPT 2005.

[Klima] Vlastimil Klima, *Find MD5 Collisions - A Toy for a notebook,* Disponível em: ***http://eprint.iacr.org/2005/075***. Ultimo acesso em 02 de Janeiro de 2006.

[Kelsey & Schneier] John Kelsey e Bruce Schneier, *Second pre-images on n-bit Hash Functions for Much less than 2^n work,* Disponível em: ***http://eprint.iacr.org/2004/304***. Ultimo acesso em 02 de Janeiro de 2006.

[Rijmen & Oswald] Vicent Rijmen e Elisabeth Oswald, *Update on SHA-1,* Disponível em: ***http://eprint.iacr.org/2005/010***. Ultimo acesso em 02 de Janeiro de 2006.

[Wang, Lisa Yin & Yu] Xiaoyun Wang, Yiquin Lisa Yin e Hongbo Yu, *Collision Search Attacks on SHA-1,* Fev 2005.

[CRYPTO2005] CRYPT2005, Disponível em: ***http://www.iacr.org/conferences/crypto2005***. Ultimo acesso em 02 de Janeiro de 2006.

[Wang, Lisa Yin & Yu 2005] Xiaoyun Wang, Yiquin Lisa Yin e Hongbo Yu**,** *Finding Collisions in the Full SHA-1,* CRYPTO 2005.

[Wang, Yao & Yao] Xiaoyun Wang, Andrew Yao e Frances Yao**,** *New Collision Search for SHA-1,* RUMP SESSION*,* CRYPTO 2005.

[ECRYPT1.1] ECRYPT Network of Excellence**,** *Recent Collision Attacks on Hash Functions: ECRYPT Position Paper,* revisition 1.1, Fev de 2005.

[GNDV2006] Praveen Gauravaram, William Millan, Ed Dawson e Kapali Viswanathan, *Constructing Secure Hash Functions by Enhancing Merkle-Damgard Construction,* Disponível em: ***http://eprint.iacr.org/2006/061***. Ultimo acesso em 25 de Março de 2006.

[Klima2006] Vlastimil Klima, *Tunnels in Hash Functions: MD5 Collisions Within a Minute,* Disponível em: ***http://eprint.iacr.org/2006/105***. Ultimo acesso em 25 de Março de 2006.

[Stevens2006] Marc Stevens, *Fast Collision Attack on MD5,* Disponível em: ***http://eprint.iacr.org/2006/104***. Ultimo acesso em 25 de Março de 2006.

[Mikle] Ondrej Mikle, *Practical Attacks on Digital Signatures Using MD5 Message Digest,* Disponível em: ***http://eprint.iacr.org/2004/356***. Ultimo acesso em 25 de Março de 2006.

[Kamisnki] Dan Kaminski, *MD5 To Be Considered Harmful Someday,* Disponível em: ***http://www.doxpara.com/md5_someday.pdf***. Ultimo acesso em 25 de Março de 2006.

[Confoo] confoo.pl, *Conflating Web Pages,* Disponível em: ***http://www.doxpara.com/research/md5/confoo.pl***. Ultimo acesso em 25 de Março de 2006.

[PoisonedPs] Stefan Lucks e Magnus Daum , *Attacking Hash Functions by Poisoned Messages "The Story of Alice and her Boss",* Disponível em: ***http://www.cits.rub.de/MD5Collisions***. Ultimo acesso em 25 de Março de 2006.

[Lenstra et al] Arjen Lenstra, Xiaoyun Wang e Benne de Weger, *Colliding X.509 Certificates,* Disponível em: ***http://eprint.iacr.org/2005/067***. Ultimo acesso em 25 de Março de 2006.

[CertDigital] Wikipedia, *Public-Key Certificates*, Disponível em: ***http://en.wikipedia.org/wiki/Public_key_certificate***. Ultimo acesso em 25 de Março de 2006.

[Stach & Liu] md5collision.c, *Faster implementation of techniques in How to Break MD5 and Other Hash Functions, by Xiaoyun Wang, et al,* Disponível em: ***http://www.stachliu.com/md5coll.c***. Ultimo acesso em 25 de Março de 2006.

[Cin] Centro de Informática, Disponível em: ***http://www.cin.ufpe.br***. Ultimo acesso em 25 de Março de 2006.

[Civil] Código Civil, *LEI No 10.406, DE 10 DE JANEIRO DE 2002.*, Disponível em: ***http://www.presidencia.gov.br/ccivil_03/LEIS/2002/L10406.htm*** .Ultimo acesso em 25 de Março de 2006.

[Exemplos] Centro de Informática, Disponível em: ***http://www.cin.ufpe.br/~famv/TG***. Ultimo acesso em 25 de Março de 2006.

[Procuracao] Disponível em: ***http://www.tributario.adv.br/arquivos/MODELOS%20DE%20PROCURACAO%20PARA%20ABERTURA%20E%20MOVIMENTACAO%20DE%20CONTA%20BANCARIA.doc***. Ultimo acesso em 25 de Março de 2006.

[Prontuario] Cláudio Giulliano Alves da Costa, *Desenvolvimento e Avaliação Tecnológica de um Sistema de Prontuário Eletrônico do Paciente, Baseado nos Paradigmas da World Wide Web e da Engenharia de Software*, Disponível em: **http://www.medsolution.com.br/claudio/dissertacao/#pdf**. Ultimo acesso em 25 de Março de 2006.

[Tripware] Tripware, Disponível em: ***http:// www.tripwire.com***. Ultimo acesso em 25 de Março de 2006.

[Integrit] Integrit, Disponível em: ***http:// integrit.sourceforge.net/***. Ultimo acesso em 25 de Março de 2006.

[Dragon] Dragon Squire, Disponível em: ***http://nbg.com/products/enterasys/dragonserver.asp***. Ultimo acesso em 25 de Março de 2006.

[DES] Wikipedia, *Data Encryption Standart*, Disponível em: ***http://en.wikipedia.org/wiki/Data_Encryption_Standard***. Ultimo acesso em 25 de Março de 2006.

[AES] Wikipedia, *Advanced Encryption Standart*, Disponível em: ***http://en.wikipedia.org/wiki/Advanced_Encryption_Standard*** . Ultimo acesso em 25 de Março de 2006.

[Whirlpool] Wikipedia, *Whirlpool*, Disponível em: ***http://en.wikipedia.org/wiki/WHIRLPOOL***. Ultimo acesso em 25 de Março de 2006.

Apêndice A: Função Conversor String/Representação Binária(String2Bin)

Recebe uma string(ascii, unicode, etc...) e converte para sua representação binária no computador.
Exemplo: String2Bin(h) = 01101000(representação binária em ascii)

Apêndice B: Função Tamanho String Binária(TamBin)

Recebe uma string binária e retorna um inteiro positivo representando seu tamanho.
Exemplo: TamBin(01101000) = 8.

Apêndice C: Função Tamanho String Binária em Bits(TamBit)

Recebe uma string binária e retorna um número na base 2 representando o seu tamanho.
Exemplo: TamBit(01101000) = 100.

Apêndice D: Função Tamanho de Conjunto(| |)

Recebe um conjunto de tamanho finito e retorna ou um inteiro positivo ou uma função representando a cardinalidade do conjunto.
Exemplo: Seja $C = \{0,1\}^3 - \{\varepsilon\}$ e $D = \{0,1\}^m - \{\varepsilon\}$, $|C| = 8$, $|D| = 2^m$.

Apêndice E: Função Valor Absoluto(abs)

É a mesma definição matemática da função modulo.
Exemplo: abs(-0,2345345345) = 0,2345345345 , abs(2,5) = 2,5.

Apêndice F: Vetores que geram o mesmo MD5

Tipo de dados em linguagem C.

```
unsigned int m0[32] = {
0x607a153b, 0x3c88e650, 0x3f1adc10, 0x80692ae1,
0xdeffe8a8, 0x4e60507e, 0x207992c5, 0x74a18935,
0x05342d55, 0x00742209, 0x81a8d65f, 0x4ea931a0,
0x54a9b70b, 0x5f1a86f5, 0x47cc85b3, 0x358a178a,
0x039f6efc, 0xcb68f232, 0xd35f5784, 0xf0b89790,
0x2665f9b3, 0x2c86326f, 0x165c390c, 0xbdedcacf,
0x7db6bf39, 0xc8b7fb1f, 0x4aadfb0a, 0xfe3edcb6,
0x24ca9511, 0x6e97977d, 0x49b67e0b, 0xa46e4abc,
```

};

unsigned int m1[32] = {
0x607a153b, 0x3c88e650, 0x3f1adc10, 0x80692ae1,
0x5effe8a8, 0x4e60507e, 0x207992c5, 0x74a18935,
0x05342d55, 0x00742209, 0x81a8d65f, 0x4ea9b1a0,
0x54a9b70b, 0x5f1a86f5, 0xc7cc85b3, 0x358a178a,
0x039f6efc, 0xcb68f232, 0xd35f5784, 0xf0b89790,
0xa665f9b3, 0x2c86326f, 0x165c390c, 0xbdedcacf,
0x7db6bf39, 0xc8b7fb1f, 0x4aadfb0a, 0xfe3e5cb6,
0x24ca9511, 0x6e97977d, 0xc9b67e0b, 0xa46e4abc,
};

MD5(m1) = MD5(m2) = 51ceea056fad7d6908c9eee673184626