Funkcja MD4

MD4 jest jednokierunkową funkcją skrótu zaprojektowaną przez Rona Rivesta. Skrót MD pochodzi od angielskiego określenia M*essage Digest* oznaczającego skrót wiadomości. Algorytm ten, dla danej wiadomości, wytwarza skrót wiadomości o długości 128 bitów.

Rivest podał swoje następujące cele przy projektowaniu tego algorytmu.

- ✓ Bezpieczeństwo. Powinno być obliczeniowo niewykonalne znalezienie dwóch wiadomości, które po skróceniu dają tę samą wartość. Żaden atak nie powinien być bardziej efektywny niż, atak brutalny.
- ✓ Bezpośrednie bezpieczeństwo. Bezpieczeństwo algorytmu MD4 nie powinno zależeć od żadnego założenia podobnego do trudności faktoryzacji liczb.
- ✓ Szybkość. Algorytm MD4 powinien być przystosowany do bardzo szybkich implementacji programowych. Powinien bazować na zbiorze prostych operacji bitowych na 32-bitowych argumentach.
- ✓ Prostota i zwartość. Algorytm MD4 powinien być tak prosty, jak tylko jest to możliwe, bez dużych struktur danych lub skomplikowanego programu.
- ✓ Zalecana architektura little-endian. Algorytm MD4 powinien być zoptymalizowany pod kątem architektur mikroprocesorów (w szczególności mikroprocesorów firmy Intel); większe i szybsze komputery dokonają niezbędnych translacji.

Po tym jak algorytm został po raz pierwszy zaprezentowany, Bert den Boer i Antoon Bosselaers przeprowadzili skuteczną kryptoanalizę dwóch z trzech cykli tego algorytmu. Ralph Merkle skutecznie zaatakował pierwsze dwa cykle . Eli Biham przeprowadził dyskusję możliwych ataków, za pomocą kryptoanalizy różnicowej, przeciwko dwóm z trzech cykli MD4. Pomimo że ataki te nie mogły być rozszerzone na cały algorytm, Rivest wzmocnił swój algorytm. Wynikiem tego był algorytm MD5.

Funkcja MD5

Funkcja MD5 jest udoskonaloną wersją funkcji MD4. Chociaż jest ona bardziej złożona od MD4, jej budowa jest podobna i również wytwarza skrót 128-bitowy.

Opis algorytmu

Po pewnym przetworzeniu wstępnym algorytm MD5 przetwarza tekst wejściowy w blokach o długości 5l2 bitów, podzielonych na szesnaście podbloków o długości 32 bity każdy. Na wyjściu algorytmu otrzymujemy zbiór czterech bloków po 32 bity każdy, które po konkatenacji tworzą pojedynczy skrót o długości 128 bitów.

Na początku wiadomość jest uzupełniana ciągiem binarnym tak, ażeby jej całkowita długość była o 64 bity krótsza od wielokrotności liczby 512. Ciąg uzupełniający składa się z pojedynczej jedynki dołączanej do końca wiadomości oraz takiej liczby zer, jaka jest potrzebna. Następnie do tak otrzymanego ciągu jest dołączany 64-bitowy ciąg stanowiący zapis długości wiadomości (długości przed dołączeniem ciągu uzupełniającego). Te dwa kroki służą do uzyskania długości wiadomości będącej wielokrotnością liczby 512 (jest to konieczne do realizacji dalszej części algorytmu) i jednocześnie zapewniają, że różne wiadomości nie będą miały tej samej postaci po dodaniu ciągu uzupełniającego.

W kolejnym kroku są nadawane wartości początkowe czterem zmiennym o długości 32 bity każda:
A = 01 23 45 67
B = 89 AB CD EF
C= FE DC BA 98
D= 76 54 32 10

Zmienne te nazywamy **zmiennymi łańcuchowymi** (ang. *chaining variables).*
Teraz rozpoczyna się główna pętla algorytmu. Pętla ta jest realizowana dla tylu 512-bitowych bloków, ile zawiera ich wiadomość.

Cztery zmienne są kopiowane na cztery inne zmienne: *A* na AA, *B* na BB, *C* na *CC* i *D* na *DD.*

Pętla główna składa się z czterech cykli (MD4 ma tylko trzy cykle), które są bardzo podobne. Każdy cykl składa się z 16 operacji. W każdej operacji jest obliczana nieliniowa funkcja trzech z czterech zmiennych A, B, *C i* D. Następnie do wyniku jest dodawana wartość pozostałej, czwartej zmiennej, pewien podblok wiadomości i pewna stała. Wynik jest przesuwany cyklicznie w prawo o zmienną liczbę bitów, a potem sumowany z jedną ze zmiennych A, B, *C* lub D. Ostatecznie wynik jest przypisywany jednej ze zmiennych A, B, *C* lub *D.*

W algorytmie występują cztery funkcje nieliniowe, jedna na każdy cykl:
F(X, Y, Z)=XY OR (NOT *X)Z*
G(X, *Y, Z) = XZ* OR Y(NOT Z)
H(X, Y, Z)=X ⊕ Y ⊕ Z
I(X, *Y, Z) = Y* ⊕ (X OR (NOT Z))

Funkcje te są tak zaprojektowane, że jeżeli odpowiednie bity zmiennych X, Y i Z są niezależne i nieobciążone, to każdy bit wyniku będzie także niezależny i nieobciążony. Funkcja F jest boolowską funkcją warunkową: jeśli X, to Y, w przeciwnym razie Z. Funkcja H jest operatorem parzystości.

Jeżeli M_j reprezentuje podblok j (0-15) wiadomości, a<<<s oznacza przesunięcie w lewo o s bitów, to następujące cztery operacje określone są wzorami:

FF(a, b, c, d, M_j, s, t_i) oznacza a=b+((a+F(b, c, d) + M_j +t_i) <<< s)
GG(a, b, c, d, M_j, s, t_i) oznacza a=b+((a+G(b, c, d) + M_j +t_i) <<< s)
HH(a, b, c, d, M_j, s, t_i) oznacza a=b+((a+H(b, c, d) + M_j +t_i) <<< s)
II(a, b, c, d, M_j, s, t_i) oznacza a=b+((a+I(b, c, d) + M_j +t_i) <<< s)

Cztery cykle algorytmu (64 kroki) mogą być opisane następująco:
Cykl l:
FF(a,.b, c, d, M[0], 7, Oxd76aa478)
FF(d, a, b, c, M[1], 12, Oxe8c7b756)
FF(c, d, a, b, M[2], 17, Ox242070db)
FF(b, c, d, a, M[3], 22, Oxc1bdceee)
FF(a, b, c, d, M[4], 7, Oxf57cOfaf)
FF(d, a, b, c, M[5], 12, Ox4787c62a)
FF(c, d, a, b, M[6], 17, Oxa8304613)
FF(b, c, d, cc, M[7], 22, Oxfd469501)
FF(a, b; c, d, M[8], 7, Ox698098d8)
FF(d, a, b, c, M[9], 12, Ox8b44f7af)
FF(c, d, a, b, M[10], 17, Oxffff5bbl)
FF(b, c, d, a, M[11], 22, Ox895cd7be)
FF(a, b, c, ct, M[12], 7, Ox6b901122)
FF(d, a, b, c, M[13], 12, Oxfd987193)
FF(c, d, a, b, M[14], 17, Oxa679438e)
FF(b, c, d, a, M[15], 22, Ox49b40821)

Cykl 2:
GG(a, b, c, d, M[1], 5, Oxf61e2562)
GG(d, a, b, c, M[6], 9, Oxc040b340)
GG(c, d, a, b, M[11], 14, Ox265e5a51)
GG(b, c, d, a, M[0], 20, Oxe9b6c7aa)
GG(a, b, c, d, M[5], 5, Oxd62flO5d)
GG(d, a, b, c, M[10], 9, 0x02441453)
GG(c, d, a, b, M[15], l4, Oxd8a1 e681)
GG(b, c, d, a, M[4], 20, Oxe7d3fbc8)
GG(a, b, c, d, M[9], 5, 0x21 el cde6)
GG(d, a, b, c, M[14], 9, Oxc33707d6)
GG(c, d, a, b, M[3], 14, Oxf4d50d87)
GG(b, c, d, a, M[8], 20, Ox455a14ed)
GG(a, b, c, d, M[13], 5, Oxa9e3e905)
GG(d, a, b, c, M[2] 9, Oxfcefa3f8)
GG(c, d, a, b, M[7], 14, Ox676f02d9)
GG(b, c, d, a, M[12], 20, Ox8d2a4c8a)

Cykl 3:
HH(a, b, c, d, M[5], 4, Oxfffa3942)
HH(d, a, b, c, M[8], 11, Ox8771f681)
HH(c, d, a, b, M[11], 16, Ox6d9d6122)
HH(b, c, d, a, M[14], 23, Oxfde5380c)
HH(a, b, c, d, M[1], 4, Oxa4beea44)
HH(d, a, b, c, M[4], 11, Ox4bdecfa9)
HH(c, d, a, b, M[7], 16, Oxf6bb4b60)
HH(b, c, d, a, M[10], 23, Oxbebfbc70)
HH(a, b, c, d, M[13], 4, Ox289b7ec6)
HH(d, a, b, c, M[0], 1l, Oxeaa127fa)
HH(c, d, a, b, M[3], 16, Oxd4ef3085)
HH(b, c, d, a, M[6], 23, Ox04881d05)
HH(a, b, c, d, M[9], 4, Oxd9d4d039)
HH(d, a, b, c, M[12], 11, Oxe6db99e5)
HH(c, d, a, b, M[15], 16, Oxlfa27cf8)
HH(b, c, d, a, M[2], 23, Oxc4ac5665)

Cykl 4:
II(a, b, c, d, M[0], 6, Oxf4292244)
II(d, a, b, c, M[7], 10, Ox411aff97)
II(c, d, a, h, M[14], 15, Oxab9423a7)
II(b, c, d, a, M[5], 21, Oxfc93a039)
II(a, b, c, d, M[12], 6, Ox655b59c3)
II(d, a, b, c, M[3], 10, OxBfOccc92)

II(c, d, a, h, M[10], 15, Oxffeff47d)
II(b, c, d, a, M[1], 2l, Ox85845dd1)
II(a, h, c, d, M[8], 6, Ox6fa87e4f)
II(d, a, b, c, M[15], 10, Oxfe2ce6e0)
II(c, d, a, b, M[6], 15, Oxa3014314)
II(b, c, d, a, M[13], 21, Ox4e0811a1)
II(a, b, c, d, M[4], 6, Oxt~7537e82)
II(d, a, b, c, M[11], 10, Oxbd3af235)
II(c, d, a, b, M[2], 15, Ox2ad7d2bb)
II(b, c, d, a, M[9], 21, Oxeb86d391)

Występujące w powyższych wzorach stałe t; były wybrane według następującej reguły:
W kroku *i* stała *t;* jest częścią całkowitą liczby 4294967296 x abs(sin(i)), gdzie i jest podane w radianach. (Zauważmy, że 4294967296 jest równe 2^{32}).

Po zakończeniu wszystkich powyższych operacji do wartości zmiennych A, B, *C i D* są dodawane odpowiednio wartości zmiennych *AA, BB, CC i DD,* a potem algorytm rozpoczyna przetwarzanie następnego bloku wiadomości. Wartość wyjściowa jest konkatenacją wartości zmiennych A, B, C oraz D.

Stopień bezpieczeństwa zapewniany przez funkcję MD5
Ron Rivest podał ulepszenia algorytmu MD5 w stosunku do MD4:

1. Dodany został czwarty cykl.
2. W każdym kroku algorytmu występuje obecnie inna wartość stałej addytywnej.
3. Funkcja g występująca w drugim cyklu została zmieniona z *(XY* OR *X7.* OR *YZ)* na *(XZ* OR *Y* NOT(Z)), aby uczynić funkcję *g* mniej symetryczną.
4. Każdy krok wykorzystuje teraz wynik kroku poprzedniego. Sprzyja to szybszemu występowaniu „efektu lawinowego".
5. Zmieniona została kolejność, w jakiej bloki wejściowe są włączane w cyklu 2 i 3, aby uczynić otrzymywane ciągi mniej podobnymi do siebie.
6. Wielkości przesunięć w każdym cyklu zostały w przybliżeniu zoptymalizowane tak, aby uzyskać szybszy „efekt lawinowy". Wielkości przesunięć w różnych cyklach są różne.

Tom Berson próbował zastosować kryptoanalizę różnicową przeciw pojedynczemu cyklowi algorytmu MD5, lecz jego atak jest jeszcze daleki od tego, aby być skutecznym przeciwko wszystkim czterem cyklom. Dużo bardziej skuteczny atak Berta den Boera i Antoona Bosselaersa może wyznaczać kolizje, wykorzystując funkcję kompresującą w MD5. Nie jest to zbyt użyteczne do ataków przeciw MD5 w praktycznych zastosowaniach, lecz oznacza to, że jeden z podstawowych wymogów projektowych algorytmu MD5 - zaprojektowanie funkcji kompresującej wolnej od kolizji - został naruszony. Jest dostatecznie dużo słabych punktów w algorytmie MD5 - należy być ostrożnym przy podejmowaniu decyzji o jego zastosowaniu.

Funkcja MD2
MD2 jest jeszcze jedną jednokierunkową funkcją skrótu zaprojektowaną przez Rona Rivesta. Funkcja ta, razem z MD5, jest wykorzystywana w protokole PEM. Wytwarza ona skrót o długości 128 bitów dla dowolnej wiadomości wejściowej. Struktura algorytmu jest podobna do MD4 i MD5, jest on wolniejszy i mniej bezpieczny.