**JAVA 实现MD4算法**

首先说说MD4的发展：MD4是麻省理工学院教授Ronald Rivest于1990年设计的一种信息摘要算法。它是一种用来测试信息完整性的密码散列函数的实行。其摘要长度为128位，一般128位长的MD4散列被表示为32位的十六进制数字。这个算法影响了后来的算法如MD5、SHA 家族和RIPEMD等。1991年Den Boer和Bosselaers发表了一篇文章指出MD4的短处，至今未能找到基于MD4以上改进的算法有任何可以用来进攻的弱点。2004年8月有人报告在计算MD4时可能发生杂凑冲撞。Den boer和Bosselaers以及其他人很快的发现了攻击MD4版本中第一步和第三步的漏洞。Dobbertin向大家演示了如何利用一部普通的个人电脑在几分钟内找到MD4完整版本中的冲突（这个冲突实际上是一种漏洞，它将导致对不同的内容进行加密却可能得到相同的加密后结果）。毫无疑问，MD4就此被淘汰掉了。

public class MD4 {  
  
 private static int A, B, C, D;  
  
 private static int X[] = new int[16];  
  
 private static int F(int X, int Y, int Z) {  
 return (X & Y) | ((~X) & Z);  
 }  
  
 private static int G(int X, int Y, int Z) {  
 return (X & Y) | (X & Z) | (Y & Z);  
 }  
  
 private static int H(int X, int Y, int Z) {  
 return X ^ Y ^ Z;  
 }  
  
 private static int lshift(int x, int s) {  
 if (s == 0) {  
 return x;  
 }  
 return (((x << s) & 0xFFFFFFFF) | ((x >> (32 - s)) & (0x7FFFFFFF >> (31 - s))));  
 }  
  
 private static int ROUND1(int a, int b, int c, int d, int k, int s) {  
 return (lshift(a + F(b, c, d) + X[k], s));  
 }  
  
 private static int ROUND2(int a, int b, int c, int d, int k, int s) {  
 return (lshift(a + G(b, c, d) + X[k] + (int) 0x5A827999, s));  
 }  
  
 private static int ROUND3(int a, int b, int c, int d, int k, int s) {  
 return (lshift(a + H(b, c, d) + X[k] + (int) 0x6ED9EBA1, s));  
 }  
  
 public static void mdfour64(int M[]) {  
 int j;  
 int AA, BB, CC, DD;  
  
 for (j = 0; j < 16; j++) {  
 X[j] = M[j];  
 }  
  
 AA = A;  
 BB = B;  
 CC = C;  
 DD = D;  
  
 A = ROUND1(A, B, C, D, 0, 3);  
 D = ROUND1(D, A, B, C, 1, 7);  
 C = ROUND1(C, D, A, B, 2, 11);  
 B = ROUND1(B, C, D, A, 3, 19);  
 A = ROUND1(A, B, C, D, 4, 3);  
 D = ROUND1(D, A, B, C, 5, 7);  
 C = ROUND1(C, D, A, B, 6, 11);  
 B = ROUND1(B, C, D, A, 7, 19);  
 A = ROUND1(A, B, C, D, 8, 3);  
 D = ROUND1(D, A, B, C, 9, 7);  
 C = ROUND1(C, D, A, B, 10, 11);  
 B = ROUND1(B, C, D, A, 11, 19);  
 A = ROUND1(A, B, C, D, 12, 3);  
 D = ROUND1(D, A, B, C, 13, 7);  
 C = ROUND1(C, D, A, B, 14, 11);  
 B = ROUND1(B, C, D, A, 15, 19);  
  
 A = ROUND2(A, B, C, D, 0, 3);  
 D = ROUND2(D, A, B, C, 4, 5);  
 C = ROUND2(C, D, A, B, 8, 9);  
 B = ROUND2(B, C, D, A, 12, 13);  
 A = ROUND2(A, B, C, D, 1, 3);  
 D = ROUND2(D, A, B, C, 5, 5);  
 C = ROUND2(C, D, A, B, 9, 9);  
 B = ROUND2(B, C, D, A, 13, 13);  
 A = ROUND2(A, B, C, D, 2, 3);  
 D = ROUND2(D, A, B, C, 6, 5);  
 C = ROUND2(C, D, A, B, 10, 9);  
 B = ROUND2(B, C, D, A, 14, 13);  
 A = ROUND2(A, B, C, D, 3, 3);  
 D = ROUND2(D, A, B, C, 7, 5);  
 C = ROUND2(C, D, A, B, 11, 9);  
 B = ROUND2(B, C, D, A, 15, 13);  
  
 A = ROUND3(A, B, C, D, 0, 3);  
 D = ROUND3(D, A, B, C, 8, 9);  
 C = ROUND3(C, D, A, B, 4, 11);  
 B = ROUND3(B, C, D, A, 12, 15);  
 A = ROUND3(A, B, C, D, 2, 3);  
 D = ROUND3(D, A, B, C, 10, 9);  
 C = ROUND3(C, D, A, B, 6, 11);  
 B = ROUND3(B, C, D, A, 14, 15);  
 A = ROUND3(A, B, C, D, 1, 3);  
 D = ROUND3(D, A, B, C, 9, 9);  
 C = ROUND3(C, D, A, B, 5, 11);  
 B = ROUND3(B, C, D, A, 13, 15);  
 A = ROUND3(A, B, C, D, 3, 3);  
 D = ROUND3(D, A, B, C, 11, 9);  
 C = ROUND3(C, D, A, B, 7, 11);  
 B = ROUND3(B, C, D, A, 15, 15);  
  
 A += AA;  
 B += BB;  
 C += CC;  
 D += DD;  
  
 A &= 0xFFFFFFFF;  
 B &= 0xFFFFFFFF;  
 C &= 0xFFFFFFFF;  
 D &= 0xFFFFFFFF;  
 }  
  
 public static void copy64(int M[], byte in[], int offset) {  
 int i;  
 for (i = 0; i < 16; i++) {  
 M[i] = ((in[offset + i \* 4 + 3] << 24) & 0xFF000000)  
 | ((in[offset + i \* 4 + 2] << 16) & 0xFF0000)  
 | ((in[offset + i \* 4 + 1] << 8) & 0xFF00)  
 | (((int) in[offset + i \* 4 + 0]) & 0xFF);  
 }  
 }  
  
 public static void copy64(int M[], byte in[]) {  
 copy64(M, in, 0);  
 }  
  
 public static void copy4(byte out[], int offset, int x) {  
 out[offset] = (byte) (x & 0xFF);  
 out[1 + offset] = (byte) ((x >> 8) & 0xFF);  
 out[2 + offset] = (byte) ((x >> 16) & 0xFF);  
 out[3 + offset] = (byte) ((x >> 24) & 0xFF);  
 }  
  
 public static byte[] mdfour(byte in[]) {  
 byte out[] = new byte[16];  
 byte buf[] = new byte[128];  
 int n = in.length;  
 int M[] = new int[16];  
 int b = n \* 8;  
 int i;  
 int offset;  
  
 A = 0x67452301;  
 B = 0xefcdab89;  
 C = 0x98badcfe;  
 D = 0x10325476;  
  
 offset = 0;  
 while (n > 64) {  
 copy64(M, in, offset);  
 mdfour64(M);  
 offset += 64;  
 n -= 64;  
 }  
  
 for (i = 0; i < 128; i++) {  
 buf[i] = (i + offset < in.length) ? in[offset + i] : 0;  
 }  
 buf[n] = (byte) 0x80;  
  
 if (n <= 55) {  
 copy4(buf, 56, b);  
 copy64(M, buf);  
 mdfour64(M);  
 } else {  
 copy4(buf, 120, b);  
 copy64(M, buf);  
 mdfour64(M);  
 copy64(M, buf, 64);  
 mdfour64(M);  
 }  
  
 for (i = 0; i < 128; i++) {  
 buf[i] = 0;  
 }  
 copy64(M, buf);  
  
 copy4(out, 0, A);  
 copy4(out, 4, B);  
 copy4(out, 8, C);  
 copy4(out, 12, D);  
  
 A = B = C = D = 0;  
 return out;  
 }  
  
 private static final char[] HEX\_DIGITS = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f' };  
  
 public static String toHexString(byte[] b) {  
 return toHexString(b, 0, b.length);  
 }  
  
 public static String toHexString(byte[] b, int off, int len) {  
 char[] buf = new char[len \* 2];  
 for (int i = 0, j = 0, k; i < len;) {  
 k = b[off + i++];  
 buf[j++] = HEX\_DIGITS[(k >>> 4) & 0x0F];  
 buf[j++] = HEX\_DIGITS[k & 0x0F];  
 }  
 return new String(buf);  
 }  
  
 private static String getTestResult(String s) {  
 return toHexString(mdfour(s.getBytes()));  
 }  
  
 public static void main(String[] args) {  
 System.out.println(getTestResult(""));  
 System.out.println(getTestResult("a"));  
 System.out.println(getTestResult("abc"));  
 System.out.println(getTestResult("message digest"));  
 System.out.println(getTestResult("abcdefghijklmnopqrstuvwxyz"));  
 System.out.println(getTestResult("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"));  
System.out.println(getTestResult("12345678901234567890123456789012345678901234567890123456789012345678901234567890"));  
 }  
  
}