# Diffie-Hellman

[编辑](javascript:;) [锁定](http://baike.baidu.com/view/10812319.htm)

Diffie-Hellman:一种确保共享KEY安全[穿越](http://baike.baidu.com/subview/287172/14317184.htm)不安全网络的方法，它是OAKLEY的一个组成部分。Whitefield与Martin Hellman在1976年提出了一个奇妙的密钥交换协议，称为Diffie-Hellman密钥交换协议/算法(Diffie-Hellman Key Exchange/Agreement Algorithm).这个机制的巧妙在于需要安全通信的双方可以用这个方法确定对称密钥。然后可以用这个密钥进行加密和解密。但是注意，这个密钥交换协议/算法只能用于密钥的交换，而不能进行消息的加密和解密。双方确定要用的密钥后，要使用其他对称密钥操作加密算法实际加密和解密消息。

Diffie-Hellman密钥交换算法的安全性依赖于这样一个事实：虽然计算以一个素数为模的指数相对容易，但计算离散对数却很困难.对于大的素数，计算出离散对数几乎是不可能的. 下面给出例子.密钥交换基于素数q = 97和97的一个原根a = 5.A和B分别选择私有密钥XA = 36和XB = 58.每人计算其公开密钥 YA = 5^36 mod 97= 50 ,YB = 5^58 mod 97= 44 在他们相互获取了公开[密钥](http://baike.baidu.com/view/934.htm)之后，各自通过计算得到双方共享的秘密密钥如下： K = (YB)^XA mod 97 = 44^36 mod 97= 75 ,K = (YA)^XB mod 97 = 50^58 mod 97= 75 从|50,44|出发，攻击者要计算出75很不容易. 下图给出了一个利用Diffie-Hellman计算的简单协议.[1]

**Diffie-Hellman协议**

[编辑](javascript:;)

假设用户A希望与用户B建立一个连接，并用一个共享的秘密密钥加密在该连接上传输的报文.用户A产生一个一次性的私有密钥XA，并计算出公开密钥YA并将其发送给用户B.用户B产生一个私有密钥XB，计算出公开密钥YB并将它发送给用户A作为响应.必要的公开数值q和a都需要提前知道.另一种方法是用户A选择q和a的值，并将这些数值包含在第一个报文中. 下面再举一个使用Diffie-Hellman算法的例子.假设有一组用户（例如一个局域网上的所有用户），每个人都产生一个长期的私有密钥XA，并计算一个公开密钥YA.这些公开密钥数值，连同全局公开数值q和a都存储在某个中央目录中.在任何时刻，用户B都可以访问用户A 的公开数值，计算一个秘密密钥，并使用这个密钥发送一个加密报文给A.如果中央目录是可信任的，那么这种形式的通信就提供了保密性和一定程度的鉴别功能.因为只有A和B可以确定这个密钥，其它用户都无法解读报文（保密性）.接收方A知道只有用户B才能使用此密钥生成这个报文（鉴别）. Diffie-Hellman算法具有两个吸引力的特征： 仅当需要时才生成密钥，减小了将密钥存储很长一段时间而致使遭受攻击的机会. 除对全局参数的约定外，密钥交换不需要事先存在的基础结构.

**原根的定义**

[编辑](javascript:;)

设m是正整数，a是整数，若a模m的阶等于φ(m)，则称a为模m的一个原根。（其中φ(m)表示m的[欧拉函数](http://baike.baidu.com/view/107769.htm)）

假设一个数g对于P来说是原根，那么g^i mod P的结果两两不同,且有 1<g<P, 0<i<P,那么g可以称为是P的一个原根,归根到底就是g^(P-1) = 1 (mod P)当且仅当指数为P-1的时候成立.(这里P是素数).

简单来说，g^i mod p ≠ g^j mod p （p为素数）

其中i≠j且i, j介於1至(p-1)之间

则g为p的原根。

求原根目前的做法只能是从2开始枚举，然后暴力判断g^(P-1) = 1 (mod P)是否当且当指数为P-1的时候成立

而由于原根一般都不大，所以可以暴力得到.

**原根的性质**

[编辑](javascript:;)

1）可以证明，如果正整数(*a*,*m*) = 1和正整数 d 满足a^d≡1(mod m)，则 d 整除 φ(*m*)。因此*Ordm*(*a*)整除φ(*m*)。在例子中，当*a*= 3时，我们仅需要验证 3 的 1 、2、3 和 6 次方模 7 的[余数](http://baike.baidu.com/view/1068391.htm)即可。

2）记δ = *Ordm*(*a*)，则a^1，……a^(δ-1)模 m 两两不[同余](http://baike.baidu.com/view/79282.htm)。因此当*a*是模*m*的原根时，a^0,a^1，……a^(δ-1)构成模 m 的[简化剩余系](http://baike.baidu.com/view/1685888.htm)。

3）模*m*有原根的充要条件是*m*= 1,2,4,*p*,2*p*,*p^n*，其中*p*是奇质数，*n*是任意[正整数](http://baike.baidu.com/view/464125.htm)。

4）对正整数(*a*,*m*) = 1，如果 a 是模 m 的原根，那么 a 是**整数模n乘法群**（即加法群 **Z**/m**Z**的可逆元，也就是所有与 m [互素](http://baike.baidu.com/view/47657.htm)的正整数构成的[等价类](http://baike.baidu.com/view/732593.htm)构成的乘法群）**Z**n的一个[生成元](http://baike.baidu.com/view/3599892.htm)。由于**Z**n有 φ(*m*)个元素，而它的生成元的个数就是它的可逆元个数，即 φ(φ(*m*))个，因此当模*m*有原根时，它有φ(φ(*m*))个原根。

**原根的例子**

[编辑](javascript:;)

设*m*= 7，则φ（7）等于6。

设*a*= 2，由于2^3=8≡1(mod 7)，2^6=64≡1(mod7)，而2!=3，2^3≡2^6(mod7)，所以 2 不是模 7 的一个原根。设*a*= 3，由于3^1≡3(mod 7)，3^2≡2(mod 7)，3^3≡6(mod 7)，3^4≡4(mod 7)，3^5≡5(mod 7)，3^6≡1(mod 7)，所以 3 是模 7 的一个原根。

补充一点，根据原根的性质1，只需要验证3^1，3^2，3^3，3^6即可，这样可以简化运算。

# Diffie-Hellman密钥交换算法

之前做过的一个项目中用过DH算法（Diffie-Hellman），这种密钥交换技术的目的在于使得两个用户安全地交换一个共享密钥（shared secret）以便用于以后的报文加密。

DH算法的有效性依赖于计算离散对数的难度。简言之，可以如下定义离散对数：首先定义一个素数*p*的原根，为其各次幂产生从1 到*p*-1的所有整数根，也就是说，如果*g*是素数*p*的一个原根，那么数值

*g* mod *p*, g2 mod *p*, ..., gp-1 mod *p*

是各不相同的整数，并且以某种排列方式组成了从1到*p*-1的所有整数。

对于一个整数*b*和素数*p*的一个原根*g*，可以找到惟一的指数*i*，使得

*b* = *gi* mod *p*     其中0 ≤ *i*≤ (*p*-1)

指数 *i*称为*b*的以*g*为基数的模*p*的离散对数或者指数。该值被记为ind*g ,p*(b)。

下面是DH算法的原理：

有两个全局公开的参数，一个素数p和一个整数g，g是p的一个原根。

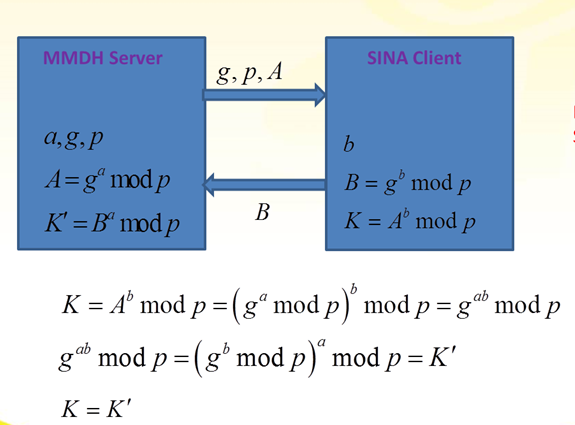
服务端的私钥和公钥分别是a和A，客户端的私钥和公钥分别是b和B；

服务端根据a、p、g，可以计算出公钥A；

服务端将g, p, A明文传送给客户端，客户端可以计算自己的公钥B，以及共享密钥K；

客户端将B明文发送给服务端，服务端也可以计算出共享密钥K。

具体如下图：



根据上述流程，客户端于服务端之间明文传输的是g、p、A、B，然后双方就可以得到一个同样的共享密钥K，并以K对后续要传输的数据进行对称加密、解密即可。

由于计算K需要一方的私钥和另一方的公钥，所以理论上，只要双方的私钥a、b不被第三方知道，这个共享密钥K就是安全可用的。

**DH算法具有两个吸引力的特征：**

1. 仅当需要时才生成密钥，减小了将密钥存储很长一段时间而致使遭受攻击的机会；
2. 除对全局参数的约定外，密钥交换不需要事先存在的基础结构；

然而，该技术也存在许多不足：

1. 没有提供双方身份的任何信息；
2. 计算密集性，因此容易遭受阻塞性攻击，即对手请求大量的密钥。受攻击者花费了相对多的计算资源来求解无用的幂系数而不是在做真正的工作；
3. 没办法防止重演攻击；
4. 容易遭受中间人的攻击。第三方C在和A通信时扮演B；和B通信时扮演A。A和B都与C协商了一个密钥，然后C就可以监听和传递通信量。中间人的攻击按如下进行：

* B在给A的报文中发送他的公开密钥YB。
* C截获并解析该报文。C将B的公钥保存下来并给A发送报文，该报文具有B的用户ID但使用C的公钥YC，但仍按照好像是来自B的样子被发送出去。A收到C的报文后，将YC和B的用户ID存储在一块。类似地，C使用YC向B发送好像来自A的报文。
* B基于私钥XB和YC计算共享密钥K1，A基于私钥XA和YC计算共享密钥K2，C使用私钥XC和YB计算K1，并使用XC和YA计算K2。
* 从现在开始，C就可以转发A发给B的报文或转发B发给A的报文，在途中根据需要修改它们的密文。使得A和B都不知道他们在和C共享通信。

Oakley算法是对Diffie-Hellman密钥交换算法的优化，它保留了后者的优点，同时克服了其弱点。

Oakley算法具有五个重要特征：

1、它采用称为cookie程序的机制来对抗阻塞攻击。

2、它使得双方能够协商一个全局参数集合。

3、它使用了现时来保证抵抗重演攻击。

4、它能够交换Diffie-Hellman公开密钥。

5、它对Diffie-Hellman交换进行鉴别以对抗中间人的攻击。

Oakley可以使用三个不同的鉴别方法：

1、数字签名：通过签署一个相互可以获得的散列代码来对交换进行鉴别；每一方都使用自己的私钥对散列代码加密。散列代码是在一些重要参数上生成的，如用户ID和现时。

2、公开密钥加密：通过使用发送者的私钥对诸如ID和现时等参数进行加密来鉴别交换。

3、对称密钥加密：通过使用某种共享密钥对交换参数进行对称加密，实现交换的鉴别。

# [RC4加密算法](http://www.aslike.net/showsoft.asp?id=64)

RC4加密算法是大名鼎鼎的RSA三人组中的头号人物Ron Rivest在1987年设计的密钥长度可变的流加密算法簇。之所以称其为簇，是由于其核心部分的S-box长度可为任意，但一般为256字节。该算法的速度可以达到[DES](http://www.aslike.net/showsoft.asp?id=18)加密的10倍左右，且具有很高级别的非线性。RC4起初是用于保护商业机密的。但是在1994年9月，它的算法被发布在互联网上，也就不再有什么商业机密了。RC4也被叫做ARC4（Alleged RC4——所谓的RC4），因为RSA从来就没有正式发布过这个算法。

**原理**

RC4算法的原理很简单，包括初始化算法（KSA）和伪随机子密码生成算法（PRGA)两大部分。假设S-box的长度为256，密钥长度为Len。先来看看算法的初始化部分（用C代码表示）：

其中，参数1是一个256长度的char型数组，定义为: unsigned char sBox[256];

参数2是密钥，其内容可以随便定义：char key[256];

参数3是密钥的长度，Len = strlen(key);

void rc4\_init(unsigned char \*s, unsigned char \*key, unsigned long Len)

{

int i =0, j = 0, k[256] = {0};

unsigned char tmp = 0;

for(i=0;i<256;i++)

{

s[i]=i;

k[i]=key[i%Len];

}

for (i=0; i<256; i++)

{

j=(j+s[i]+k[i])%256;

tmp = s[i];

s[i] = s[j];     //交换s[i]和s[j]

s[j] = tmp;

}

}

在初始化的过程中，密钥的主要功能是将S-box搅乱，i确保S-box的每个元素都得到处理，j保证S-box的搅乱是随机的。而不同的S-box在经过伪随机子密码生成算法的处理后可以得到不同的子密钥序列，将S-box和明文进行xor运算，得到密文，解密过程也完全相同。

再来看看算法的加密部分（用C代码表示）：

其中，参数1是上边rc4\_init函数中，被搅乱的S-box;

参数2是需要加密的数据data;

参数3是data的长度.

void rc4\_crypt(unsigned char \*s, unsigned char \*Data, unsigned long Len)

{

int x = 0, y = 0, t = 0, i = 0;

unsigned char tmp;

for(i=0;i<Len;i++)

{

x=(x+1)%256;

y=(y+s[x])%256;

tmp = s[x];

s[x] = s[y];     //交换s[x]和s[y]

s[y] = tmp;

t=(s[x]+s[y])%256;

Data[i] ^= s[t];

}

}

最后，在main函数中，调用顺序如下：

void main()

{

unsigned char s[256] = {0};//S-box

char key[256] = {"just for test"};

char pData[512] = "这是一个用来加密的数据Data";

ULONG len = strlen(pData);

printf("pData = %s\n",pData);

printf("key = %s, length = %d\n",key,strlen(key));

rc4\_init(s,(unsigned char \*)key,strlen(key));//初始化

rc4\_crypt(s,(unsigned char \*)pData,len);//加密

printf("pData = %s\n\n",pData);

rc4\_crypt(s,(unsigned char \*)pData,len);//解密

printf("pData = %s\n\n",pData);

}

因此最终的完整程序是：

//程序开始

#include<stdio.h>

#include<string.h>

typedef unsigned long ULONG;

void rc4\_init(unsigned char \*s, unsigned char \*key, unsigned long Len) //初始化函数

{

int i =0, j = 0；

char k[256] = {0};

unsigned char tmp = 0;

for(i=0;i<256;i++)

{

s[i]=i;

k[i]=key[i%Len];

}

for (i=0; i<256; i++)

{

j=(j+s[i]+k[i])%256;

tmp = s[i];

s[i] = s[j]; //交换s[i]和s[j]

s[j] = tmp;

}

}

void rc4\_crypt(unsigned char \*s, unsigned char \*Data, unsigned long Len) //加解密

{

int i = 0, j = 0, t = 0;

unsigned long k = 0;

unsigned char tmp;

for(k=0;k<Len;k++)

{

i=(i+1)%256;

j=(j+s[i])%256;

tmp = s[i];

s[i] = s[j]; //交换s[x]和s[y]

s[j] = tmp;

t=(s[i]+s[j])%256;

Data[k] ^= s[t];

}

}

void main()

{

unsigned char s[256] = {0},s2[256] = {0};    //S-box

char key[256] = {"just for test"};

char pData[512] = "这是一个用来加密的数据Data";

ULONG len = strlen(pData);

printf("pData = %s\n",pData);

printf("key = %s, length = %d\n\n",key,strlen(key));

rc4\_init(s,(unsigned char \*)key,strlen(key));   //已经完成了初始化

printf("完成对S[i]的初始化，如下：\n\n");

for (int i=0; i<256; i++)

{

printf("%-3d ",s[i]);

}

printf("\n\n");

for(i=0;i<256;i++)//用s2[i]暂时保留经过初始化的s[i]，很重要的！！！

{

s2[i]=s[i];

}

printf("已经初始化，现在加密:\n\n");

rc4\_crypt(s,(unsigned char \*)pData,len);//加密

printf("pData = %s\n\n",pData);

printf("已经加密，现在解密:\n\n");

rc4\_init(s,(unsigned char \*)key, strlen(key));   //初始化密钥

rc4\_crypt(s2,(unsigned char \*)pData,len);//解密

printf("pData = %s\n\n",pData);

}

//程序完

**漏洞**

由于[RC4算法](http://www.aslike.net/showsoft.asp?id=64)加密是采用的xor，所以，一旦子密钥序列出现了重复，密文就有可能被破解。关于如何破解xor加密，请参看Bruce Schneier的Applied Cryptography一书的1.4节Simple XOR，在此我就不细说了。那么，RC4算法生成的子密钥序列是否会出现重复呢？由于存在部分弱密钥，使得子密钥序列在不到100万字节内就发生了完全的重复，如果是部分重复，则可能在不到10万字节内就能发生重复，因此，推荐在使用RC4算法时，必须对加密密钥进行测试，判断其是否为弱密钥。其不足主要体现于，在无线网络中IV（初始化向量）不变性漏洞。

而且，根据目前的分析结果，没有任何的分析对于密钥长度达到128位的RC4有效，所以，RC4是目前最安全的加密算法之一，大家可以放心使用！

# [RC4加密算法及其实现](http://www.cnblogs.com/zwios/p/4196839.html)

　RC4加密算法(http://en.wikipedia.org/wiki/RC4)是大名鼎鼎的RSA三人组中的头号人物Ron Rivest在1987年设计的密钥长度可变的流加密算法簇。之所以称其为簇，是由于其核心部分的S-box长度可为任意，但一般为256字节。该算法的速度可以达到DES加密的10倍左右，且具有很高级别的非线性。RC4起初是用于保护商业机密的。但是在1994年9月，它的算法被发布在互联网上，也就不再有什么商业机密了。RC4也被叫做ARC4（Alleged RC4——所谓的RC4），因为RSA从来就没有正式发布过这个算法。

　　RC4的原理很简单，包括初始化算法（KSA/SetKey）和伪随机子密码生成算法（PRGA/Transform)两大部分。实现代码如下(VS2008):

**[cpp]** [view plaincopy](http://blog.csdn.net/ynnmnm/article/details/6335560)

1. //rc4.h
2. template<class T> inline void
3. swap(T& i, T& j)
4. {
5. T tmp = i;
6. i = j;
7. j = tmp;
8. }
9. class RC4
10. {
11. public:
12. void SetKey(const char\* key, int keylen);
13. void Transform(char\* output, const char\* input, int len);
14. private:
15. unsigned char key\_[256];
16. };
17. /////////////////////////////////////////////
18. //rc4.cc
19. #include "rc4.h"
20. void RC4::SetKey(const char\* key, int keylen)
21. {
22. for (int i = 0; i < 256; i++)
23. {
24. key\_[i] = i;
25. }
26. int j = 0;
27. for (int i = 0; i < 256; i++)
28. {
29. j = (j + key\_[i] + key[i%keylen]) % 256;
30. swap(key\_[i], key\_[j]);
31. }
32. }
33. void RC4::Transform(char\* output, const char\* input, int len)
34. {
35. int i = 0, j = 0;
36. for (int k = 0; k < len; k++)
37. {
38. i = (i + 1) % 256;
39. j = (j + key\_[i]) % 256;
40. swap(key\_[i], key\_[j]);
41. unsigned char subkey = key\_[(key\_[i] + key\_[j]) % 256];
42. output[k] = subkey ^ input[k];
43. }
44. }
45. /////////////////////////////////////////
46. //main.cc
47. #include <stdio.h>
48. #include <string.h>
49. #include <stdlib.h>
50. #include "rc4.h"
51. int main()
52. {
53. char input[256] = "Times teaches all things to him who lives forever but I have not the luxury of eternity.";
54. char output[256] = "";
55. printf("before encrypt:%s/n", input);
56. const int keylen = 16;
57. char key[keylen] = "";
58. for (int i = 0; i < keylen; i++)
59. {
60. key[i] = rand() % 256;
61. }
62. RC4 rc4encrypt, rc4decrypt;
63. rc4encrypt.SetKey(key, keylen);
64. rc4decrypt.SetKey(key, keylen);
65. rc4encrypt.Transform(output, input, strlen(input));
66. printf("after encrypt: %s/n", output);
67. rc4decrypt.Transform(output, output, strlen(input));
68. printf("after decrypt: %s/n", output);
69. return 0;
70. }

# [RC4加密算法的原理及实现](http://blog.csdn.net/lc_910927/article/details/37599161)

       RC4于1987年提出，和DES算法一样，是一种对称加密算法，也就是说使用的密钥为单钥（或称为私钥）。但不同于DES的是，RC4不是对明文进行分组处理，而是字节流的方式依次加密明文中的每一个字节，解密的时候也是依次对密文中的每一个字节进行解密。

       RC4算法的特点是算法简单，运行速度快，而且密钥长度是可变的，可变范围为1-256字节(8-2048比特)，在如今技术支持的前提下，当密钥长度为128比特时，用暴力法搜索密钥已经不太可行，所以可以预见RC4的密钥范围任然可以在今后相当长的时间里抵御暴力搜索密钥的攻击。实际上，如今也没有找到对于128bit密钥长度的RC4加密算法的有效攻击方法。

**在介绍RC4算法原理之前，先看看算法中的几个关键变量：**

       1、密钥流：RC4算法的关键是根据明文和密钥生成相应的密钥流，密钥流的长度和明文的长度是对应的，也就是说明文的长度是500字节，那么密钥流也是500字节。当然，加密生成的密文也是500字节，因为密文第i字节=明文第i字节^密钥流第i字节；

       2、状态向量S：长度为256，S[0],S[1].....S[255]。每个单元都是一个字节，算法运行的任何时候，S都包括0-255的8比特数的排列组合，只不过值的位置发生了变换；

       3、临时向量T：长度也为256，每个单元也是一个字节。如果密钥的长度是256字节，就直接把密钥的值赋给T，否则，轮转地将密钥的每个字节赋给T；

       4、密钥K：长度为1-256字节，注意密钥的长度keylen与明文长度、密钥流的长度没有必然关系，通常密钥的长度趣味16字节（128比特）。

**RC4的原理分为三步：**

1、初始化S和T

for i=0 to 255 do

   S[i]=i;

   T[i]=K[ imodkeylen ];

2、初始排列S

j=0;

for i=0 to 255 do

   j= ( j+S[i]+T[i])mod256;

   swap(S[i],S[j]);

3、产生密钥流

i,j=0;

for r=0 to len do  //r为明文长度，r字节

   i=(i+1) mod 256;

   j=(j+S[i])mod 256;

   swap(S[i],S[j]);

   t=(S[i]+S[j])mod 256;

   k[r]=S[t];

**下面给出RC4加密解密的C++实现：**

加密类：

/\*

加密类

\*/

class RC4 {

public:

/\*

构造函数，参数为密钥长度

\*/

RC4(int kl):keylen(kl) {

srand((unsigned)time(NULL));

for(int i=0;i<kl;++i){ //随机生产长度为keylen字节的密钥

int tmp=rand()%256;

K.push\_back(char(tmp));

}

}

/\*

由明文产生密文

\*/

void encryption(const string &,const string &,const string &);

private:

unsigned char S[256]; //状态向量，共256字节

unsigned char T[256]; //临时向量，共256字节

int keylen; //密钥长度，keylen个字节，取值范围为1-256

vector<char> K; //可变长度密钥

vector<char> k; //密钥流

/\*

初始化状态向量S和临时向量T，供keyStream方法调用

\*/

void initial() {

for(int i=0;i<256;++i){

S[i]=i;

T[i]=K[i%keylen];

}

}

/\*

初始排列状态向量S，供keyStream方法调用

\*/

void rangeS() {

int j=0;

for(int i=0;i<256;++i){

j=(j+S[i]+T[i])%256;

//cout<<"j="<<j<<endl;

S[i]=S[i]+S[j];

S[j]=S[i]-S[j];

S[i]=S[i]-S[j];

}

}

/\*

生成密钥流

len:明文为len个字节

\*/

void keyStream(int len);

};

void RC4::keyStream(int len) {

initial();

rangeS();

int i=0,j=0,t;

while(len--){

i=(i+1)%256;

j=(j+S[i])%256;

S[i]=S[i]+S[j];

S[j]=S[i]-S[j];

S[i]=S[i]-S[j];

t=(S[i]+S[j])%256;

k.push\_back(S[t]);

}

}

void RC4::encryption(const string &plaintext,const string &ks,const string &ciphertext) {

ifstream in;

ofstream out,outks;

in.open(plaintext);

//获取输入流的长度

in.seekg(0,ios::end);

int lenFile=in.tellg();

in.seekg(0, ios::beg);

//生产密钥流

keyStream(lenFile);

outks.open(ks);

for(int i=0;i<lenFile;++i){

outks<<(k[i]);

}

outks.close();

//明文内容读入bits中

unsigned char \*bits=new unsigned char[lenFile];

in.read((char \*)bits,lenFile);

in.close();

out.open(ciphertext);

//将明文按字节依次与密钥流异或后输出到密文文件中

for(int i=0;i<lenFile;++i){

out<<(unsigned char)(bits[i]^k[i]);

}

<span style="white-space:pre"> </span>out.close();

delete []bits;

}

解密类：

/\*

解密类

\*/

class RC4\_decryption{

public:

/\*

构造函数，参数为密钥流文件和密文文件

\*/

RC4\_decryption(const string ks,const string ct):keystream(ks),ciphertext(ct) {}

/\*

解密方法，参数为解密文件名

\*/

void decryption(const string &);

private:

string ciphertext,keystream;

};

void RC4\_decryption::decryption(const string &res){

ifstream inks,incp;

ofstream out;

inks.open(keystream);

incp.open(ciphertext);

//计算密文长度

inks.seekg(0,ios::end);

const int lenFile=inks.tellg();

inks.seekg(0, ios::beg);

//读入密钥流

unsigned char \*bitKey=new unsigned char[lenFile];

inks.read((char \*)bitKey,lenFile);

inks.close();

//读入密文

unsigned char \*bitCip=new unsigned char[lenFile];

incp.read((char \*)bitCip,lenFile);

incp.close();

//解密后结果输出到解密文件

out.open(res);

for(int i=0;i<lenFile;++i)

out<<(unsigned char)(bitKey[i]^bitCip[i]);

out.close();

}

程序实现时，需要注意的是，状态向量数组S和临时向量数组T的类型应设为unsigned char，而不是char。因为在一些机器下，将char默认做为signed char看待，在算法中计算下标i，j的时候，会涉及char转int，如果是signed的char，那么将char的8位拷贝到int的低8位后，还会根据char的符号为，在int的高位补0或1。由于密钥是随机产生的，如果遇到密钥的某个字节的高位为1的话，那么计算得到的数组下标为负数，就会越界。

程序运行示例

main函数：

int main(){

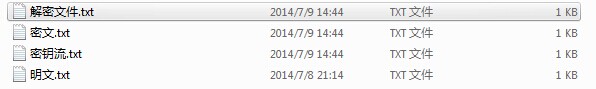
RC4 rc4(16); //密钥长16字节

rc4.encryption("明文.txt","密钥流.txt","密文.txt");

RC4\_decryption decrypt("密钥流.txt","密文.txt");

decrypt.decryption("解密文件.txt");

}



明文：我爱小兔子！

密文：'柀L&t餥6洲

密钥流：镈膺嚬3屽u

解密文件：我爱小兔子！

这是第一篇网络安全方面的博客，如有错误，欢迎指正！

# MD5

[编辑](javascript:;) [锁定](http://baike.baidu.com/view/10812319.htm)

Message Digest Algorithm MD5（中文名为[消息摘要算法](http://baike.baidu.com/view/2313810.htm)第五版）为计算机安全领域广泛使用的一种散列函数，用以提供消息的完整性保护。该算法的文件号为RFC 1321（R.Rivest,MIT Laboratory for Computer Science and RSA Data Security Inc. April 1992）。

**MD5**即Message-Digest Algorithm 5（信息-摘要算法5），用于确保信息传输完整一致。是计算机广泛使用的杂凑算法之一（又译[摘要算法](http://baike.baidu.com/view/10961371.htm)、[哈希算法](http://baike.baidu.com/view/273836.htm)），主流编程语言普遍已有MD5实现。将数据（如汉字）运算为另一固定长度值，是杂凑算法的基础原理，MD5的前身有MD2、[MD3](http://baike.baidu.com/view/2535629.htm)和[MD4](http://baike.baidu.com/view/444142.htm)。

MD5算法具有以下特点：

1、压缩性：任意长度的数据，算出的MD5值长度都是固定的。

2、容易计算：从原数据计算出MD5值很容易。

3、抗修改性：对原数据进行任何改动，哪怕只修改1个字节，所得到的MD5值都有很大区别。

4、强抗碰撞：已知原数据和其MD5值，想找到一个具有相同MD5值的数据（即伪造数据）是非常困难的。

MD5的作用是让大容量信息在用[数字签名](http://baike.baidu.com/view/7626.htm)软件签署私人[密钥](http://baike.baidu.com/view/934.htm)前被"[压缩](http://baike.baidu.com/subview/786588/12546221.htm)"成一种保密的格式（就是把一个任意长度的字节串变换成一定长的[十六进制](http://baike.baidu.com/view/230306.htm)数字串）。除了MD5以外，其中比较有名的还有[sha-1](http://baike.baidu.com/view/94209.htm)、[RIPEMD](http://baike.baidu.com/view/260854.htm)以及Haval等。

中文名

消息摘要算法

外文名

Message Digest Algorithm MD5

别    称

摘要算法

提出时间

1991年

应用学科

信息技术，计算机科学

适用领域范围

软件下载站、论坛数据库、系统文件安全

目录

1. 1 [发展历史](http://baike.baidu.com/item/MD5#1)
2. ▪ [MD2](http://baike.baidu.com/item/MD5#1_1)
3. ▪ [MD4](http://baike.baidu.com/item/MD5#1_2)
4. ▪ [MD5](http://baike.baidu.com/item/MD5#1_3)
5. 2 [MD5应用](http://baike.baidu.com/item/MD5#2)
6. ▪ [一致性验证](http://baike.baidu.com/item/MD5#2_1)
7. ▪ [数字签名](http://baike.baidu.com/item/MD5#2_2)
8. ▪ [安全访问认证](http://baike.baidu.com/item/MD5#2_3)
9. 3 [算法原理](http://baike.baidu.com/item/MD5#3)
10. ▪ [原理](http://baike.baidu.com/item/MD5#3_1)
11. ▪ [MD5加密字符串实例](http://baike.baidu.com/item/MD5#3_2)
12. ▪ [C++实现](http://baike.baidu.com/item/MD5#3_3)
13. ▪ [JAVA实现](http://baike.baidu.com/item/MD5#3_4)
14. ▪ [VB2010实现](http://baike.baidu.com/item/MD5#3_5)
15. ▪ [JavaScript实现](http://baike.baidu.com/item/MD5#3_6)
16. ▪ [伪代码实现](http://baike.baidu.com/item/MD5#3_7)
17. 4 [弱点](http://baike.baidu.com/item/MD5#4)

**MD5发展历史**

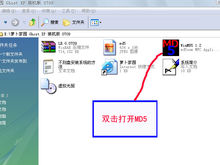
[编辑](javascript:;)

**MD2**

Rivest在1989年开发出MD2算法。在这个算法中，首先对信息进行数据补位，使信息的字节长度是16的倍数。然后，以一个16位的检验和追加到信息末尾，并且根据这个新产生的信息计算出散列值。后来，Rogier和Chauvaud发现如果忽略了检验将和MD2产生冲突。MD2算法[加密](http://baike.baidu.com/view/40927.htm)后结果是唯一的（即不同信息加密后的结果不同）。

**MD4**

为了加

[](http://baike.baidu.com/pic/MD5/212708/0/020e66f09887dc877831aa53?fr=lemma&ct=single)MD5

强算法的安全性，Rivest在1990年又开发出MD4算法。MD4算法同样需要填补信息以确保信息的比特位长度减去448后能被512整除（信息比特位长度mod 512 = 448）。然后，一个以64位[二进制](http://baike.baidu.com/view/18536.htm)表示的信息的最初长度被添加进来。信息被处理成512位damg?rd/merkle迭代结构的区块，而且每个区块要通过三个不同步骤的处理。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。

尽管MD4算法在安全上有个这么大的漏洞，但它对在其后才被开发出来的好几种信息安全加密算法的出现却有着不可忽视的引导作用。

**MD5**

1991年，Rivest开发出技术上更为趋近成熟的[md5算法](http://baike.baidu.com/view/706946.htm)。它在MD4的基础上增加了"安全-带子"（safety-belts）的概念。虽然MD5比MD4复杂度大一些，但却更为安全。这个算法很明显的由四个和MD4设计有少许不同的步骤组成。在[MD5算法](http://baike.baidu.com/view/706946.htm)中，信息-摘要的大小和填充的必要条件与MD4完全相同。Den boer和Bosselaers曾发现MD5算法中的假冲突（pseudo-collisions），但除此之外就没有其他被发现的加密后结果了。

**MD5应用**

[编辑](javascript:;)

**MD5一致性验证**

MD5的

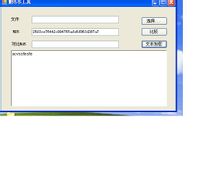
[](http://baike.baidu.com/pic/MD5/212708/0/e6508eef7570d4abcf1b3e5f?fr=lemma&ct=single)md5

典型应用是对一段信息（Message）产生信息摘要（Message-Digest），以防止被篡改。比如，在[Unix](http://baike.baidu.com/view/8095.htm)下有很多软件在下载的时候都有一个文件名相同，文件扩展名为.md5的文件，在这个文件中通常只有一行文本，大致结构如：

MD5 (tanajiya.tar.gz) = 38b8c2c1093dd0fec383a9d9ac940515

这就是tanajiya.tar.gz文件的数字签名。MD5将整个文件当作一个大文本信息，通过其不可逆的字符串变换算法，产生了这个唯一的MD5信息摘要。为了让读者朋友对MD5的应用有个直观的认识，笔者以一个比方和一个实例来简要描述一下其工作过程：

大家都知道，地球上任何人都有自己独一无二的[指纹](http://baike.baidu.com/view/5628.htm)，这常常成为司法机关鉴别罪犯身份最值得信赖的方法；与之类似，MD5就可以为任何文件（不管其大小、格式、数量）产生一个同样独一无二的

[](http://baike.baidu.com/pic/MD5/212708/0/bd7faf351443f1cfa71e125a?fr=lemma&ct=single)md5

“[数字指纹](http://baike.baidu.com/view/962481.htm)”，如果任何人对文件做了任何改动，其[MD5值](http://baike.baidu.com/view/1047696.htm)也就是对应的“数字指纹”都会发生变化。

我们常常在某些软件下载站点的某软件信息中看到其MD5值，它的作用就在于我们可以在下载该软件后，对下载回来的文件用专门的软件（如Windows MD5 Check等）做一次[MD5校验](http://baike.baidu.com/view/274197.htm)，以确保我们获得的文件与该站点提供的文件为同一文件。

具体来说文件的MD5值就像是这个文件的“[数字指纹](http://baike.baidu.com/view/962481.htm)”。每个文件的MD5值是不同的，如果任何人对文件做了任何改动，其[MD5值](http://baike.baidu.com/view/1047696.htm)也就是对应的“数字指纹”就会发生变化。比如下载服务器针对一个文件预先提供一个MD5值，用户下载完该文件后，用我这个算法重新计算下载文件的MD5值，通过比较这两个值是否相同，就能判断下载的文件是否出错，或者说下载的文件是否被篡改了。

利用MD5算法来进行文件校验的方案被大量应用到软件下载站、论坛数据库、系统文件安全等方面。

**MD5数字签名**

MD5的典型应用是对一段Message(字节串)产生fingerprint([指纹](http://baike.baidu.com/view/5628.htm)），以防止被“篡改”。举个例子，你将一段话写在一个叫 readme.txt文件中，并对这个readme.txt产生一个MD5的值并记录在案，然后你可以传播这个文件给别人，别人如果修改了文件中的任何内容，你对这个文件重新计算MD5时就会发现（两个MD5值不相同）。如果再有一个第三方的认证机构，用MD5还可以防止文件作者的“抵赖”，这就是所谓的数字签名应用。

**MD5安全访问认证**

MD5还广泛用于操作系统的登陆认证上，如Unix、各类[BSD系统](http://baike.baidu.com/view/4225732.htm)登录密码、[数字签名](http://baike.baidu.com/view/7626.htm)等诸多方面。如在Unix系统中用户的密码是以MD5（或其它类似的算法）经Hash运算后存储在文件系统中。当用户登录的时候，系统把用户输入的密码进行MD5 Hash运算，然后再去和保存在文件系统中的MD5值进行比较，进而确定输入的密码是否正确。通过这样的步骤，系统在并不知道用户密码的明码的情况下就可以确定用户登录系统的合法性。这可以避免用户的密码被具有系统管理员权限的用户知道。MD5将任意长度的“字节串”映射为一个128bit的大整数，并且是通过该128bit反推原始字符串是困难的，换句话说就是，即使你看到源程序和算法描述，也无法将一个MD5的值变换回原始的字符串，从数学原理上说，是因为原始的字符串有无穷多个，这有点象不存在反函数的数学函数。所以，要遇到了md5密码的问题，比较好的办法是：你可以用这个系统中的md5（）函数重新设一个密码，如admin，把生成的一串密码的Hash值覆盖原来的Hash值就行了。

正是因为这个原因，现在被黑客使用最多的一种破译密码的方法就是一种被称为"跑字典"的方法。有两种方法得到字典，一种是日常搜集的用做密码的字符串表，另一种是用排列组合方法生成的，先用MD5程序计算出这些字典项的MD5值，然后再用目标的MD5值在这个字典中检索。我们假设密码的最大长度为8位字节（8 Bytes），同时密码只能是字母和数字，共26+26+10=62个字节，排列组合出的字典的项数则是P（62,1）+P（62,2）….+P（62,8），那也已经是一个很天文的数字了，存储这个字典就需要TB级的[磁盘阵列](http://baike.baidu.com/view/63423.htm)，而且这种方法还有一个前提，就是能获得目标账户的密码MD5值的情况下才可以。这种加密技术被广泛的应用于Unix系统中，这也是为什么Unix系统比一般操作系统更为坚固一个重要原因。

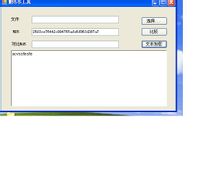
**MD5算法原理**

[编辑](javascript:;)

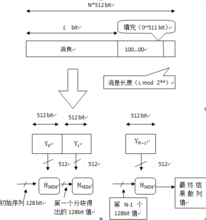
**MD5原理**

对MD5算法简要的叙述可以为：MD5以512位分组来处理输入的信息，且每一分组又被划分为16个32位子分组，经过了一系列的处理后，算法的输出由四个32位分组组成，将这四个32位分组级联后将生成一个128位散列值。

总体流程如下图所示，

[](http://baike.baidu.com/pic/MD5/212708/0/bd7faf351443f1cfa71e125a?fr=lemma&ct=single)

表示第i个分组，每次的运算都由前一轮的128位结果值和第i块512bit值进行运算。

[](http://baike.baidu.com/pic/MD5/212708/0/c75c10385343fbf2ee0d9594b17eca8065388f67?fr=lemma&ct=single)MD5算法的整体流程图

**1.填充**

在MD5算法中，首先需要对信息进行填充，使其位长对512求余的结果等于448，**并且填充必须进行，即使其位长对512求余的结果等于448**。因此，信息的位长（Bits Length）将被扩展至N\*512+448，N为一个非负整数，N可以是零。

填充的方法如下：

**1)** 在信息的后面填充一个1和无数个0，直到满足上面的条件时才停止用0对信息的填充。

**2)** 在这个结果后面附加一个以64位二进制表示的填充前信息长度（单位为Bit），如果二

进制表示的填充前信息长度超过64位，则取低64位。

经过这两步的处理，信息的位长=N\*512+448+64=(N+1）\*512，即长度恰好是512的整数倍。这样做的原因是为满足后面处理中对信息长度的要求。

**2. 初始化变量**

初始的128位值为初试链接变量，这些参数用于第一轮的运算，以大端[字节序](http://baike.baidu.com/view/2194385.htm)来表示，他们分别为： A=0x01234567，B=0x89ABCDEF，C=0xFEDCBA98，D=0x76543210。

（每一个变量给出的数值是高字节存于内存低地址，低字节存于内存高地址，即大端字节序。在程序中变量A、B、C、D的值分别为0x67452301，0xEFCDAB89，0x98BADCFE，0x10325476）

**3. 处理分组数据**

每一分组的算法流程如下：

第一分组需要将上面四个链接变量复制到另外四个变量中：A到a，B到b，C到c，D到d。从第二分组开始的变量为上一分组的运算结果，即A = a， B = b， C = c， D = d。

主循环有四轮（MD4只有三轮），每轮循环都很相似。第一轮进行16次操作。每次操作对a、b、c和d中的其中三个作一次非线性函数运算，然后将所得结果加上第四个变量，文本的一个子分组和一个常数。再将所得结果向左**环移**一个不定的数，并加上a、b、c或d中之一。最后用该结果取代a、b、c或d中之一。

以下是每次操作中用到的四个非线性函数（每轮一个）。

F( X ,Y ,Z ) = ( X & Y ) | ( (~X) & Z )

G( X ,Y ,Z ) = ( X & Z ) | ( Y & (~Z) )

H( X ,Y ,Z ) =X ^ Y ^ Z

I( X ,Y ,Z ) =Y ^ ( X | (~Z) )

（&是与（And），|是或（Or），~是非（Not），^是异或（Xor））

这四个函数的说明：如果X、Y和Z的对应位是独立和均匀的，那么结果的每一位也应是独立和均匀的。

F是一个逐位运算的函数。即，如果X，那么Y，否则Z。函数H是逐位奇偶操作符。

假设Mj表示消息的第j个子分组（从0到15），常数ti是4294967296\*abs( sin(i) ）的整数部分，i 取值从1到64，单位是弧度。（4294967296=232）

现定义：

FF(a ,b ,c ,d ,Mj ,s ,ti ) 操作为 a = b + ( (a + F(b,c,d) + Mj + ti) << s)

GG(a ,b ,c ,d ,Mj ,s ,ti ) 操作为 a = b + ( (a + G(b,c,d) + Mj + ti) << s)

HH(a ,b ,c ,d ,Mj ,s ,ti) 操作为 a = b + ( (a + H(b,c,d) + Mj + ti) << s)

II(a ,b ,c ,d ,Mj ,s ,ti) 操作为 a = b + ( (a + I(b,c,d) + Mj + ti) << s)

**注意：“<<”表示循环左移位，不是左移位。**

这四轮（共64步）是：

第一轮

FF(a ,b ,c ,d ,M0 ,7 ,0xd76aa478 )

FF(d ,a ,b ,c ,M1 ,12 ,0xe8c7b756 )

FF(c ,d ,a ,b ,M2 ,17 ,0x242070db )

FF(b ,c ,d ,a ,M3 ,22 ,0xc1bdceee )

FF(a ,b ,c ,d ,M4 ,7 ,0xf57c0faf )

FF(d ,a ,b ,c ,M5 ,12 ,0x4787c62a )

FF(c ,d ,a ,b ,M6 ,17 ,0xa8304613 )

FF(b ,c ,d ,a ,M7 ,22 ,0xfd469501)

FF(a ,b ,c ,d ,M8 ,7 ,0x698098d8 )

FF(d ,a ,b ,c ,M9 ,12 ,0x8b44f7af )

FF(c ,d ,a ,b ,M10 ,17 ,0xffff5bb1 )

FF(b ,c ,d ,a ,M11 ,22 ,0x895cd7be )

FF(a ,b ,c ,d ,M12 ,7 ,0x6b901122 )

FF(d ,a ,b ,c ,M13 ,12 ,0xfd987193 )

FF(c ,d ,a ,b ,M14 ,17 ,0xa679438e )

FF(b ,c ,d ,a ,M15 ,22 ,0x49b40821 )

第二轮

GG(a ,b ,c ,d ,M1 ,5 ,0xf61e2562 )

GG(d ,a ,b ,c ,M6 ,9 ,0xc040b340 )

GG(c ,d ,a ,b ,M11 ,14 ,0x265e5a51 )

GG(b ,c ,d ,a ,M0 ,20 ,0xe9b6c7aa )

GG(a ,b ,c ,d ,M5 ,5 ,0xd62f105d )

GG(d ,a ,b ,c ,M10 ,9 ,0x02441453 )

GG(c ,d ,a ,b ,M15 ,14 ,0xd8a1e681 )

GG(b ,c ,d ,a ,M4 ,20 ,0xe7d3fbc8 )

GG(a ,b ,c ,d ,M9 ,5 ,0x21e1cde6 )

GG(d ,a ,b ,c ,M14 ,9 ,0xc33707d6 )

GG(c ,d ,a ,b ,M3 ,14 ,0xf4d50d87 )

GG(b ,c ,d ,a ,M8 ,20 ,0x455a14ed )

GG(a ,b ,c ,d ,M13 ,5 ,0xa9e3e905 )

GG(d ,a ,b ,c ,M2 ,9 ,0xfcefa3f8 )

GG(c ,d ,a ,b ,M7 ,14 ,0x676f02d9 )

GG(b ,c ,d ,a ,M12 ,20 ,0x8d2a4c8a )

第三轮

HH(a ,b ,c ,d ,M5 ,4 ,0xfffa3942 )

HH(d ,a ,b ,c ,M8 ,11 ,0x8771f681 )

HH(c ,d ,a ,b ,M11 ,16 ,0x6d9d6122 )

HH(b ,c ,d ,a ,M14 ,23 ,0xfde5380c )

HH(a ,b ,c ,d ,M1 ,4 ,0xa4beea44 )

HH(d ,a ,b ,c ,M4 ,11 ,0x4bdecfa9 )

HH(c ,d ,a ,b ,M7 ,16 ,0xf6bb4b60 )

HH(b ,c ,d ,a ,M10 ,23 ,0xbebfbc70 )

HH(a ,b ,c ,d ,M13 ,4 ,0x289b7ec6 )

HH(d ,a ,b ,c ,M0 ,11 ,0xeaa127fa )

HH(c ,d ,a ,b ,M3 ,16 ,0xd4ef3085 )

HH(b ,c ,d ,a ,M6 ,23 ,0x04881d05 )

HH(a ,b ,c ,d ,M9 ,4 ,0xd9d4d039 )

HH(d ,a ,b ,c ,M12 ,11 ,0xe6db99e5 )

HH(c ,d ,a ,b ,M15 ,16 ,0x1fa27cf8 )

HH(b ,c ,d ,a ,M2 ,23 ,0xc4ac5665 )

第四轮

II(a ,b ,c ,d ,M0 ,6 ,0xf4292244 )

II(d ,a ,b ,c ,M7 ,10 ,0x432aff97 )

II(c ,d ,a ,b ,M14 ,15 ,0xab9423a7 )

II(b ,c ,d ,a ,M5 ,21 ,0xfc93a039 )

II(a ,b ,c ,d ,M12 ,6 ,0x655b59c3 )

II(d ,a ,b ,c ,M3 ,10 ,0x8f0ccc92 )

II(c ,d ,a ,b ,M10 ,15 ,0xffeff47d )

II(b ,c ,d ,a ,M1 ,21 ,0x85845dd1 )

II(a ,b ,c ,d ,M8 ,6 ,0x6fa87e4f )

II(d ,a ,b ,c ,M15 ,10 ,0xfe2ce6e0 )

II(c ,d ,a ,b ,M6 ,15 ,0xa3014314 )

II(b ,c ,d ,a ,M13 ,21 ,0x4e0811a1 )

II(a ,b ,c ,d ,M4 ,6 ,0xf7537e82 )

II(d ,a ,b ,c ,M11 ,10 ,0xbd3af235 )

II(c ,d ,a ,b ,M2 ,15 ,0x2ad7d2bb )

II(b ,c ,d ,a ,M9 ,21 ,0xeb86d391 )

所有这些完成之后，将a、b、c、d分别在原来基础上再加上A、B、C、D。

即a = a + A，b = b + B，c = c + C，d = d + D

然后用下一分组数据继续运行以上算法。

**4. 输出**

最后的输出是a、b、c和d的级联。

当你按照我上面所说的方法实现MD5算法以后，你可以用以下几个信息对你做出来的程序作一个简单的测试，看看程序有没有错误。

MD5 ("") = d41d8cd98f00b204e9800998ecf8427e

MD5 ("a") = 0cc175b9c0f1b6a831c399e269772661

MD5 ("abc") = 900150983cd24fb0d6963f7d28e17f72

MD5 ("message digest") = f96b697d7cb7938d525a2f31aaf161d0

MD5 ("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b

MD5 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz") =

f29939a25efabaef3b87e2cbfe641315

MD5 ("8a683566bcc7801226b3d8b0cf35fd97") =cf2cb5c89c5e5eeebef4a76becddfcfd

**MD5加密字符串实例**

现以字符串“jklmn”为例。

该字符串在内存中表示为：6A 6B 6C 6D 6E（从左到右为低地址到高地址，后同），信息长度为40 bits， 即0x28。

对其填充，填充至448位，即56字节。结果为：

6A 6B 6C 6D 6E 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

剩下64位，即8字节填充填充前信息位长，按小端字节序填充剩下的8字节，结果为。

6A 6B 6C 6D 6E 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 28 00 00 00 00 00 00 00

（64字节，512 bits）

初始化A、B、C、D四个变量。

将这64字节填充后数据分成16个小组（程序中对应为16个数组），即：

M0：6A 6B 6C 6D （这是内存中的顺序，按照小端字节序原则，对应数组M(0)的值为0x6D6C6B6A，下同）

M1：6E 80 00 00

M2：00 00 00 00

.....

M14：28 00 00 00

M15：00 00 00 00

经过“**3. 分组数据处理**”后，a、b、c、d值分别为0xD8523F60、0x837E0144、0x517726CA、0x1BB6E5FE

在内存中为a：60 3F 52 D8

b：44 01 7E 83

c：CA 26 77 51

d：FE E5 B6 1B

a、b、c、d按内存顺序输出即为最终结果：603F52D844017E83CA267751FEE5B61B。这就是字符串“jklmn”的MD5值。

**MD5C++实现**

[?](http://baike.baidu.com/item/MD5)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140 | #include<iostream>  #include<string>  using namespace std;  #define shift(x, n) (((x) << (n)) | ((x) >> (32-(n))))//右移的时候，高位一定要补零，而不是补充符号位  #define F(x, y, z) (((x) & (y)) | ((~x) & (z)))  #define G(x, y, z) (((x) & (z)) | ((y) & (~z)))  #define H(x, y, z) ((x) ^ (y) ^ (z))  #define I(x, y, z) ((y) ^ ((x) | (~z)))  #define A 0x67452301  #define B 0xefcdab89  #define C 0x98badcfe  #define D 0x10325476  //strBaye的长度  unsigned int strlength;  //A,B,C,D的临时变量  unsigned int atemp;  unsigned int btemp;  unsigned int ctemp;  unsigned int dtemp;  //常量ti unsigned int(abs(sin(i+1))\*(2pow32))  const unsigned int k[]={          0xd76aa478,0xe8c7b756,0x242070db,0xc1bdceee,          0xf57c0faf,0x4787c62a,0xa8304613,0xfd469501,0x698098d8,          0x8b44f7af,0xffff5bb1,0x895cd7be,0x6b901122,0xfd987193,          0xa679438e,0x49b40821,0xf61e2562,0xc040b340,0x265e5a51,          0xe9b6c7aa,0xd62f105d,0x02441453,0xd8a1e681,0xe7d3fbc8,          0x21e1cde6,0xc33707d6,0xf4d50d87,0x455a14ed,0xa9e3e905,          0xfcefa3f8,0x676f02d9,0x8d2a4c8a,0xfffa3942,0x8771f681,          0x6d9d6122,0xfde5380c,0xa4beea44,0x4bdecfa9,0xf6bb4b60,          0xbebfbc70,0x289b7ec6,0xeaa127fa,0xd4ef3085,0x04881d05,          0xd9d4d039,0xe6db99e5,0x1fa27cf8,0xc4ac5665,0xf4292244,          0x432aff97,0xab9423a7,0xfc93a039,0x655b59c3,0x8f0ccc92,          0xffeff47d,0x85845dd1,0x6fa87e4f,0xfe2ce6e0,0xa3014314,          0x4e0811a1,0xf7537e82,0xbd3af235,0x2ad7d2bb,0xeb86d391};  //向左位移数  const unsigned int s[]={7,12,17,22,7,12,17,22,7,12,17,22,7,          12,17,22,5,9,14,20,5,9,14,20,5,9,14,20,5,9,14,20,          4,11,16,23,4,11,16,23,4,11,16,23,4,11,16,23,6,10,          15,21,6,10,15,21,6,10,15,21,6,10,15,21};  const char str16[]="0123456789abcdef";  void mainLoop(unsigned int M[])  {      unsigned int f,g;      unsigned int a=atemp;      unsigned int b=btemp;      unsigned int c=ctemp;      unsigned int d=dtemp;      for (unsigned int i = 0; i < 64; i++)      {          if(i<16){              f=F(b,c,d);              g=i;          }else if (i<32)          {              f=G(b,c,d);              g=(5\*i+1)%16;          }else if(i<48){              f=H(b,c,d);              g=(3\*i+5)%16;          }else{              f=I(b,c,d);              g=(7\*i)%16;          }          unsigned int tmp=d;          d=c;          c=b;          b=b+shift((a+f+k[i]+M[g]),s[i]);          a=tmp;      }      atemp=a+atemp;      btemp=b+btemp;      ctemp=c+ctemp;      dtemp=d+dtemp;  }  /\*  \*填充函数  \*处理后应满足bits≡448(mod512),字节就是bytes≡56（mode64)  \*填充方式为先加一个1,其它位补零  \*最后加上64位的原来长度  \*/  unsigned int\* add(string str)  {      unsigned int num=((str.length()+8)/64)+1;//以512位,64个字节为一组      unsigned int \*strByte=new unsigned int[num\*16];    //64/4=16,所以有16个整数      strlength=num\*16;      for (unsigned int i = 0; i < num\*16; i++)          strByte[i]=0;      for (unsigned int i=0; i <str.length(); i++)      {          strByte[i>>2]|=(str[i])<<((i%4)\*8);//一个整数存储四个字节，i>>2表示i/4 一个unsigned int对应4个字节，保存4个字符信息      }      strByte[str.length()>>2]|=0x80<<(((str.length()%4))\*8);//尾部添加1 一个unsigned int保存4个字符信息,所以用128左移      /\*      \*添加原长度，长度指位的长度，所以要乘8，然后是小端序，所以放在倒数第二个,这里长度只用了32位      \*/      strByte[num\*16-2]=str.length()\*8;      return strByte;  }  string changeHex(int a)  {      int b;      string str1;      string str="";      for(int i=0;i<4;i++)      {          str1="";          b=((a>>i\*8)%(1<<8))&0xff;   //逆序处理每个字节          for (int j = 0; j < 2; j++)          {              str1.insert(0,1,str16[b%16]);              b=b/16;          }          str+=str1;      }      return str;  }  string getMD5(string source)  {      atemp=A;    //初始化      btemp=B;      ctemp=C;      dtemp=D;      unsigned int \*strByte=add(source);      for(unsigned int i=0;i<strlength/16;i++)      {          unsigned int num[16];          for(unsigned int j=0;j<16;j++)              num[j]=strByte[i\*16+j];          mainLoop(num);      }      return changeHex(atemp).append(changeHex(btemp)).append(changeHex(ctemp)).append(changeHex(dtemp));  }  unsigned int main()  {      string ss;  //    cin>>ss;      string s=getMD5("abc");      cout<<s;      return 0;  } |

**MD5JAVA实现**

参考

[?](http://baike.baidu.com/item/MD5)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158 | public class MD5{      /\*      \*四个链接变量      \*/      private final int A=0x67452301;      private final int B=0xefcdab89;      private final int C=0x98badcfe;      private final int D=0x10325476;      /\*      \*ABCD的临时变量      \*/      private int Atemp,Btemp,Ctemp,Dtemp;        /\*      \*常量ti      \*公式:floor(abs(sin(i+1))×(2pow32)      \*/      private final int K[]={          0xd76aa478,0xe8c7b756,0x242070db,0xc1bdceee,          0xf57c0faf,0x4787c62a,0xa8304613,0xfd469501,0x698098d8,          0x8b44f7af,0xffff5bb1,0x895cd7be,0x6b901122,0xfd987193,          0xa679438e,0x49b40821,0xf61e2562,0xc040b340,0x265e5a51,          0xe9b6c7aa,0xd62f105d,0x02441453,0xd8a1e681,0xe7d3fbc8,          0x21e1cde6,0xc33707d6,0xf4d50d87,0x455a14ed,0xa9e3e905,          0xfcefa3f8,0x676f02d9,0x8d2a4c8a,0xfffa3942,0x8771f681,          0x6d9d6122,0xfde5380c,0xa4beea44,0x4bdecfa9,0xf6bb4b60,          0xbebfbc70,0x289b7ec6,0xeaa127fa,0xd4ef3085,0x04881d05,          0xd9d4d039,0xe6db99e5,0x1fa27cf8,0xc4ac5665,0xf4292244,          0x432aff97,0xab9423a7,0xfc93a039,0x655b59c3,0x8f0ccc92,          0xffeff47d,0x85845dd1,0x6fa87e4f,0xfe2ce6e0,0xa3014314,          0x4e0811a1,0xf7537e82,0xbd3af235,0x2ad7d2bb,0xeb86d391};      /\*      \*向左位移数,计算方法未知      \*/      private final int s[]={7,12,17,22,7,12,17,22,7,12,17,22,7,          12,17,22,5,9,14,20,5,9,14,20,5,9,14,20,5,9,14,20,          4,11,16,23,4,11,16,23,4,11,16,23,4,11,16,23,6,10,          15,21,6,10,15,21,6,10,15,21,6,10,15,21};          /\*      \*初始化函数      \*/      private void init(){          Atemp=A;          Btemp=B;          Ctemp=C;          Dtemp=D;      }      /\*      \*移动一定位数      \*/      private    int    shift(int a,int s){          return(a<<s)|(a>>>(32-s));//右移的时候，高位一定要补零，而不是补充符号位      }      /\*      \*主循环      \*/      private void MainLoop(int M[]){          int F,g;          int a=Atemp;          int b=Btemp;          int c=Ctemp;          int d=Dtemp;          for(int i = 0; i < 64; i ++){              if(i<16){                  F=(b&c)|((~b)&d);                  g=i;              }else if(i<32){                  F=(d&b)|((~d)&c);                  g=(5\*i+1)%16;              }else if(i<48){                  F=b^c^d;                  g=(3\*i+5)%16;              }else{                  F=c^(b|(~d));                  g=(7\*i)%16;              }              int tmp=d;              d=c;              c=b;              b=b+shift(a+F+K[i]+M[g],s[i]);              a=tmp;          }          Atemp=a+Atemp;          Btemp=b+Btemp;          Ctemp=c+Ctemp;          Dtemp=d+Dtemp;        }      /\*      \*填充函数      \*处理后应满足bits≡448(mod512),字节就是bytes≡56（mode64)      \*填充方式为先加一个0,其它位补零      \*最后加上64位的原来长度      \*/      private int[] add(String str){          int num=((str.length()+8)/64)+1;//以512位，64个字节为一组          int strByte[]=new int[num\*16];//64/4=16，所以有16个整数          for(int i=0;i<num\*16;i++){//全部初始化0              strByte[i]=0;          }          int    i;          for(i=0;i<str.length();i++){              strByte[i>>2]|=str.charAt(i)<<((i%4)\*8);//一个整数存储四个字节，小端序          }          strByte[i>>2]|=0x80<<((i%4)\*8);//尾部添加1          /\*          \*添加原长度，长度指位的长度，所以要乘8，然后是小端序，所以放在倒数第二个,这里长度只用了32位          \*/          strByte[num\*16-2]=str.length()\*8;              return strByte;      }      /\*      \*调用函数      \*/      public String getMD5(String source){          init();          int strByte[]=add(source);          for(int i=0;i<strByte.length/16;i++){          int num[]=new int[16];          for(int j=0;j<16;j++){              num[j]=strByte[i\*16+j];          }          MainLoop(num);          }          return changeHex(Atemp)+changeHex(Btemp)+changeHex(Ctemp)+changeHex(Dtemp);        }      /\*      \*整数变成16进制字符串      \*/      private String changeHex(int a){          String str="";          for(int i=0;i<4;i++){              str+=String.format("%2s", Integer.toHexString(((a>>i\*8)%(1<<8))&0xff)).replace(' ', '0');            }          return str;      }      /\*      \*单例      \*/      private static MD5 instance;      public static MD5 getInstance(){          if(instance==null){              instance=new MD5();          }          return instance;      }        private MD5(){};        public static void main(String[] args){          String str=MD5.getInstance().getMD5("");          System.out.println(str);      }  } |

[?](http://baike.baidu.com/item/MD5)

|  |  |
| --- | --- |
| 1 |  |

**MD5VB2010实现**

[?](http://baike.baidu.com/item/MD5)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90 | Imports System    Imports System.Security.Cryptography    Imports System.Text      Module Example      '哈希输入字符串并返回一个 32 字符的十六进制字符串哈希。        Function GetMd5Hash(ByVal input As String) As String            '创建新的一个 MD5CryptoServiceProvider 对象的实例。            Dim md5Hasher As New MD5CryptoServiceProvider()            '输入的字符串转换为字节数组，并计算哈希。            Dim data As Byte() = md5Hasher.ComputeHash(Encoding.Default.GetBytes(input))            '创建一个新的 StringBuilder 收集的字节，并创建一个字符串。            Dim sBuilder As New StringBuilder()            '通过每个字节的哈希数据和格式为十六进制字符串的每一个循环。            For i As Integer = 0 To data.Length - 1                sBuilder.Append(data(i).ToString("x2"))            Next            '返回十六进制字符串。            Return sBuilder.ToString()        End Function          '验证对一个字符串的哈希值。        Function VerifyMd5Hash(ByVal input As String, ByVal hash As String) As Boolean            '哈希的输入。            Dim hashOfInput As String = GetMd5Hash(input)            '创建 StringComparer 的哈希进行比较。            Dim comparer As StringComparer = StringComparer.OrdinalIgnoreCase            Return comparer.Compare(hashOfInput, hash) = 0        End Function          Sub Main()            Dim source As String = "Hello World!"            Dim hash As String = GetMd5Hash(source)            Console.WriteLine($"进行MD5加密的字符串为：{source}，加密的结果是：{hash}。")            Console.WriteLine("正在验证哈希……")            If VerifyMd5Hash(source, hash) Then                Console.WriteLine("哈希值是  相同的。")    Else                Console.WriteLine("哈希值是不相同的。")    EndIf        EndSub    EndModule      '此代码示例产生下面的输出：      '进行MD5加密的字符串为：Hello World!，加密的结果是：ed076287532e86365e841e92bfc50d8c。    '正在验证哈希……    '哈希值是相同的。 |

**MD5JavaScript实现**

[?](http://baike.baidu.com/item/MD5)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203 | function md5(string) {      function md5\_RotateLeft(lValue, iShiftBits) {          return (lValue << iShiftBits) | (lValue >>> (32 - iShiftBits));      }      function md5\_AddUnsigned(lX, lY) {          var lX4, lY4, lX8, lY8, lResult;          lX8 = (lX & 0x80000000);          lY8 = (lY & 0x80000000);          lX4 = (lX & 0x40000000);          lY4 = (lY & 0x40000000);          lResult = (lX & 0x3FFFFFFF) + (lY & 0x3FFFFFFF);          if (lX4 & lY4) {              return (lResult ^ 0x80000000 ^ lX8 ^ lY8);          }          if (lX4 | lY4) {              if (lResult & 0x40000000) {                  return (lResult ^ 0xC0000000 ^ lX8 ^ lY8);              } else {                  return (lResult ^ 0x40000000 ^ lX8 ^ lY8);              }          } else {              return (lResult ^ lX8 ^ lY8);          }      }      function md5\_F(x, y, z) {          return (x & y) | ((~x) & z);      }      function md5\_G(x, y, z) {          return (x & z) | (y & (~z));      }      function md5\_H(x, y, z) {          return (x ^ y ^ z);      }      function md5\_I(x, y, z) {          return (y ^ (x | (~z)));      }      function md5\_FF(a, b, c, d, x, s, ac) {          a = md5\_AddUnsigned(a, md5\_AddUnsigned(md5\_AddUnsigned(md5\_F(b, c, d), x), ac));          return md5\_AddUnsigned(md5\_RotateLeft(a, s), b);      };      function md5\_GG(a, b, c, d, x, s, ac) {          a = md5\_AddUnsigned(a, md5\_AddUnsigned(md5\_AddUnsigned(md5\_G(b, c, d), x), ac));          return md5\_AddUnsigned(md5\_RotateLeft(a, s), b);      };      function md5\_HH(a, b, c, d, x, s, ac) {          a = md5\_AddUnsigned(a, md5\_AddUnsigned(md5\_AddUnsigned(md5\_H(b, c, d), x), ac));          return md5\_AddUnsigned(md5\_RotateLeft(a, s), b);      };      function md5\_II(a, b, c, d, x, s, ac) {          a = md5\_AddUnsigned(a, md5\_AddUnsigned(md5\_AddUnsigned(md5\_I(b, c, d), x), ac));          return md5\_AddUnsigned(md5\_RotateLeft(a, s), b);      };      function md5\_ConvertToWordArray(string) {          var lWordCount;          var lMessageLength = string.length;          var lNumberOfWords\_temp1 = lMessageLength + 8;          var lNumberOfWords\_temp2 = (lNumberOfWords\_temp1 - (lNumberOfWords\_temp1 % 64)) / 64;          var lNumberOfWords = (lNumberOfWords\_temp2 + 1) \* 16;          var lWordArray = Array(lNumberOfWords - 1);          var lBytePosition = 0;          var lByteCount = 0;          while (lByteCount < lMessageLength) {              lWordCount = (lByteCount - (lByteCount % 4)) / 4;              lBytePosition = (lByteCount % 4) \* 8;              lWordArray[lWordCount] = (lWordArray[lWordCount] | (string.charCodeAt(lByteCount) << lBytePosition));              lByteCount++;          }          lWordCount = (lByteCount - (lByteCount % 4)) / 4;          lBytePosition = (lByteCount % 4) \* 8;          lWordArray[lWordCount] = lWordArray[lWordCount] | (0x80 << lBytePosition);          lWordArray[lNumberOfWords - 2] = lMessageLength << 3;          lWordArray[lNumberOfWords - 1] = lMessageLength >>> 29;          return lWordArray;      };      function md5\_WordToHex(lValue) {          var WordToHexValue = "",          WordToHexValue\_temp = "",          lByte, lCount;          for (lCount = 0; lCount <= 3; lCount++) {              lByte = (lValue >>> (lCount \* 8)) & 255;              WordToHexValue\_temp = "0" + lByte.toString(16);              WordToHexValue = WordToHexValue + WordToHexValue\_temp.substr(WordToHexValue\_temp.length - 2, 2);          }          return WordToHexValue;      };      function md5\_Utf8Encode(string) {          string = string.replace(/\r\n/g, "\n");          var utftext = "";          for (var n = 0; n < string.length; n++) {              var c = string.charCodeAt(n);              if (c < 128) {                  utftext += String.fromCharCode(c);              } else if ((c > 127) && (c < 2048)) {                  utftext += String.fromCharCode((c >> 6) | 192);                  utftext += String.fromCharCode((c & 63) | 128);              } else {                  utftext += String.fromCharCode((c >> 12) | 224);                  utftext += String.fromCharCode(((c >> 6) & 63) | 128);                  utftext += String.fromCharCode((c & 63) | 128);              }          }          return utftext;      };      var x = Array();      var k, AA, BB, CC, DD, a, b, c, d;      var S11 = 7,      S12 = 12,      S13 = 17,      S14 = 22;      var S21 = 5,      S22 = 9,      S23 = 14,      S24 = 20;      var S31 = 4,      S32 = 11,      S33 = 16,      S34 = 23;      var S41 = 6,      S42 = 10,      S43 = 15,      S44 = 21;      string = md5\_Utf8Encode(string);      x = md5\_ConvertToWordArray(string);      a = 0x67452301;      b = 0xEFCDAB89;      c = 0x98BADCFE;      d = 0x10325476;      for (k = 0; k < x.length; k += 16) {          AA = a;          BB = b;          CC = c;          DD = d;          a = md5\_FF(a, b, c, d, x[k + 0], S11, 0xD76AA478);          d = md5\_FF(d, a, b, c, x[k + 1], S12, 0xE8C7B756);          c = md5\_FF(c, d, a, b, x[k + 2], S13, 0x242070DB);          b = md5\_FF(b, c, d, a, x[k + 3], S14, 0xC1BDCEEE);          a = md5\_FF(a, b, c, d, x[k + 4], S11, 0xF57C0FAF);          d = md5\_FF(d, a, b, c, x[k + 5], S12, 0x4787C62A);          c = md5\_FF(c, d, a, b, x[k + 6], S13, 0xA8304613);          b = md5\_FF(b, c, d, a, x[k + 7], S14, 0xFD469501);          a = md5\_FF(a, b, c, d, x[k + 8], S11, 0x698098D8);          d = md5\_FF(d, a, b, c, x[k + 9], S12, 0x8B44F7AF);          c = md5\_FF(c, d, a, b, x[k + 10], S13, 0xFFFF5BB1);          b = md5\_FF(b, c, d, a, x[k + 11], S14, 0x895CD7BE);          a = md5\_FF(a, b, c, d, x[k + 12], S11, 0x6B901122);          d = md5\_FF(d, a, b, c, x[k + 13], S12, 0xFD987193);          c = md5\_FF(c, d, a, b, x[k + 14], S13, 0xA679438E);          b = md5\_FF(b, c, d, a, x[k + 15], S14, 0x49B40821);          a = md5\_GG(a, b, c, d, x[k + 1], S21, 0xF61E2562);          d = md5\_GG(d, a, b, c, x[k + 6], S22, 0xC040B340);          c = md5\_GG(c, d, a, b, x[k + 11], S23, 0x265E5A51);          b = md5\_GG(b, c, d, a, x[k + 0], S24, 0xE9B6C7AA);          a = md5\_GG(a, b, c, d, x[k + 5], S21, 0xD62F105D);          d = md5\_GG(d, a, b, c, x[k + 10], S22, 0x2441453);          c = md5\_GG(c, d, a, b, x[k + 15], S23, 0xD8A1E681);          b = md5\_GG(b, c, d, a, x[k + 4], S24, 0xE7D3FBC8);          a = md5\_GG(a, b, c, d, x[k + 9], S21, 0x21E1CDE6);          d = md5\_GG(d, a, b, c, x[k + 14], S22, 0xC33707D6);          c = md5\_GG(c, d, a, b, x[k + 3], S23, 0xF4D50D87);          b = md5\_GG(b, c, d, a, x[k + 8], S24, 0x455A14ED);          a = md5\_GG(a, b, c, d, x[k + 13], S21, 0xA9E3E905);          d = md5\_GG(d, a, b, c, x[k + 2], S22, 0xFCEFA3F8);          c = md5\_GG(c, d, a, b, x[k + 7], S23, 0x676F02D9);          b = md5\_GG(b, c, d, a, x[k + 12], S24, 0x8D2A4C8A);          a = md5\_HH(a, b, c, d, x[k + 5], S31, 0xFFFA3942);          d = md5\_HH(d, a, b, c, x[k + 8], S32, 0x8771F681);          c = md5\_HH(c, d, a, b, x[k + 11], S33, 0x6D9D6122);          b = md5\_HH(b, c, d, a, x[k + 14], S34, 0xFDE5380C);          a = md5\_HH(a, b, c, d, x[k + 1], S31, 0xA4BEEA44);          d = md5\_HH(d, a, b, c, x[k + 4], S32, 0x4BDECFA9);          c = md5\_HH(c, d, a, b, x[k + 7], S33, 0xF6BB4B60);          b = md5\_HH(b, c, d, a, x[k + 10], S34, 0xBEBFBC70);          a = md5\_HH(a, b, c, d, x[k + 13], S31, 0x289B7EC6);          d = md5\_HH(d, a, b, c, x[k + 0], S32, 0xEAA127FA);          c = md5\_HH(c, d, a, b, x[k + 3], S33, 0xD4EF3085);          b = md5\_HH(b, c, d, a, x[k + 6], S34, 0x4881D05);          a = md5\_HH(a, b, c, d, x[k + 9], S31, 0xD9D4D039);          d = md5\_HH(d, a, b, c, x[k + 12], S32, 0xE6DB99E5);          c = md5\_HH(c, d, a, b, x[k + 15], S33, 0x1FA27CF8);          b = md5\_HH(b, c, d, a, x[k + 2], S34, 0xC4AC5665);          a = md5\_II(a, b, c, d, x[k + 0], S41, 0xF4292244);          d = md5\_II(d, a, b, c, x[k + 7], S42, 0x432AFF97);          c = md5\_II(c, d, a, b, x[k + 14], S43, 0xAB9423A7);          b = md5\_II(b, c, d, a, x[k + 5], S44, 0xFC93A039);          a = md5\_II(a, b, c, d, x[k + 12], S41, 0x655B59C3);          d = md5\_II(d, a, b, c, x[k + 3], S42, 0x8F0CCC92);          c = md5\_II(c, d, a, b, x[k + 10], S43, 0xFFEFF47D);          b = md5\_II(b, c, d, a, x[k + 1], S44, 0x85845DD1);          a = md5\_II(a, b, c, d, x[k + 8], S41, 0x6FA87E4F);          d = md5\_II(d, a, b, c, x[k + 15], S42, 0xFE2CE6E0);          c = md5\_II(c, d, a, b, x[k + 6], S43, 0xA3014314);          b = md5\_II(b, c, d, a, x[k + 13], S44, 0x4E0811A1);          a = md5\_II(a, b, c, d, x[k + 4], S41, 0xF7537E82);          d = md5\_II(d, a, b, c, x[k + 11], S42, 0xBD3AF235);          c = md5\_II(c, d, a, b, x[k + 2], S43, 0x2AD7D2BB);          b = md5\_II(b, c, d, a, x[k + 9], S44, 0xEB86D391);          a = md5\_AddUnsigned(a, AA);          b = md5\_AddUnsigned(b, BB);          c = md5\_AddUnsigned(c, CC);          d = md5\_AddUnsigned(d, DD);      }      return (md5\_WordToHex(a) + md5\_WordToHex(b) + md5\_WordToHex(c) + md5\_WordToHex(d)).toLowerCase();  } |

**MD5伪代码实现**

//Note:Allvariablesareunsigned32bitsandwrapmodulo2^32whencalculatingvarint[64]r,k//rspecifiestheper-roundshiftamountsr[0..15]:={7,12,17,22,7,12,17,22,7,12,17,22,7,12,17,22}r[16..31]:={5,9,14,20,5,9,14,20,5,9,14,20,5,9,14,20}r[32..47]:={4,11,16,23,4,11,16,23,4,11,16,23,4,11,16,23}r[48..63]:={6,10,15,21,6,10,15,21,6,10,15,21,6,10,15,21}//Usebinaryintegerpartofthesinesofintegersasconstants:forifrom0to63k[i]:=floor(abs(sin(i+1））×2^32）//Initializevariables:varinth0:=0x67452301varinth1:=0xEFCDAB89varinth2:=0x98BADCFEvarinth3:=0x10325476//Pre-processing:append"1"bittomessageappend"0"bitsuntilmessagelengthinbits≡448(mod512）appendbitlengthofmessageas64-bitlittle-endianintegertomessage//Processthemessageinsuccessive512-bitchunks:foreach512-bitchunkofmessagebreakchunkintosixteen32-bitlittle-endianwordsw[i],0≤i≤15//Initializehashvalueforthischunk:varinta:=h0varintb:=h1varintc:=h2varintd:=h3//Mainloop:forifrom0to63if0≤i≤15thenf:=(bandc)or((notb)andd)g:=ielseif16≤i≤31f:=(dandb)or((notd)andc)g:=（5×i+1）mod16elseif32≤i≤47f:=bxorcxordg:=（3×i+5）mod16elseif48≤i≤63f:=cxor(bor(notd))g:=（7×i)mod16temp:=dd:=cc:=bb:=((a+f+k[i]+w[g])leftrotater[i])+ba:=temp//Addthischunk'shashtoresultsofar:h0:=h0+ah1:=h1+bh2:=h2+ch3:=h3+dvarintdigest:=h0appendh1appendh2appendh3//(expressedaslittle-endian)MD5加密工具

利用MD5的算法原理，可以使用各种计算机语言进行实现，形成各种各样的MD5加密校验工具。有很多的在线工具可以实现这一点，这些在线工具一般是采用[JavaScript](http://baike.baidu.com/view/16168.htm)语言实现，使用非常方便快捷。[2]

MD5优势

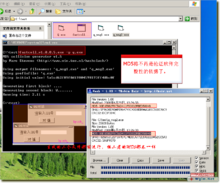
Van oorschot和Wiener曾经考虑过一个在散列中暴力搜寻冲突的函数（brute-force hash function），而且他们猜测一个被设计专门用来搜索MD5冲突的机器（这台机器在1994年的制造成本大约是一百万美元）可以平均每24天就找到一个冲突。但单从1991年到2001年这10年间，竟没有出现替代MD5算法的MD6或被叫做其他什么名字的新算法这一点，我们就可以看出这个瑕疵并没有太多的影响MD5的安全性。上面所有这些都不足以成为MD5的在实际应用中的问题。并且，由于MD5算法的使用不需要支付任何版权费用的，所以在一般的情况下（非绝密应用领域。但即便是应用在绝密领域内，MD5也不失为一种非常优秀的中间技术），MD5怎么都应该算得上是非常安全的了。

**MD5弱点**

[编辑](javascript:;)

2004年8月17日的美国[加州](http://baike.baidu.com/view/526914.htm)圣巴巴拉的国际[密码学](http://baike.baidu.com/view/25311.htm)会议（Crypto’2004）上，来自中国山东大学的[王小云](http://baike.baidu.com/view/350813.htm)教授做了破译MD5、HAVAL-128、 MD4和RIPEMD算法的报告，公布了MD系列算法的破解结果。宣告了固若金汤的世界通行密码标准MD5的堡垒轰然倒塌，引发了密码学界的轩然大波。(注意:并非是真正的破解，只是加速了杂凑冲撞）

令世界顶尖密码学家想象不到的是，破解MD5之后，2005年2月，王小云教授又破解了另一国际密码SHA－1。因为SHA－1在美国等国际社会有更加广泛的应用，密码被破的消息一出，在国际社会的反响可谓石破天惊。换句话说，王小云的研究成果表明了从理论上讲电子签名可以伪造，必须及时添加限制条件，或者重新选用更为安全的密码标准，以保证电子商务的安全。

[](http://baike.baidu.com/pic/MD5/212708/0/a005b33434b25e8fd0a2d3ce?fr=lemma&ct=single)MD5验证可执行文件不再可靠的消息

MD5破解工程权威网站是为了公开征集专门针对MD5的攻击而设立的，网站于2004年8月17日宣布：“[中国研究](http://baike.baidu.com/view/832876.htm)人员发现了完整MD5算法的碰撞；Wang,Feng,Lai,Yu公布了MD5、MD4、HAVAL-128、RIPEMD-128几个 Hash函数的碰撞。这是近年来[密码学](http://baike.baidu.com/view/25311.htm)领域最具实质性的研究进展。使用他们的技术，在数个小时内就可以找到MD5碰撞。……由于这个里程碑式的发现，MD5CRK项目将在随后48小时内结束”。

在2004年8月之前，国际密码学界对[王小云](http://baike.baidu.com/subview/350813/7544439.htm)这个名字并不熟悉。2004年8月，在美国[加州](http://baike.baidu.com/view/526914.htm)圣芭芭拉召开的国际密码大会上，并没有被安排发言的王小云教授拿着自己的研究成果找到[会议主席](http://baike.baidu.com/view/1334437.htm)，没想到慧眼识珠的会议主席破例给了她15分钟时间来介绍自己的成果，而通常发言人只被允许有两三分钟的时间。王小云与助手展示了MD5、SHA-0及其他相关杂凑函数的杂凑冲撞。所谓杂凑冲撞指两个完全不同的讯息经杂凑函数计算得出完全相同的杂凑值。根据[鸽巢原理](http://baike.baidu.com/view/39756.htm)，以有长度限制的杂凑函数计算没有长度限制的讯息是必然会有冲撞情况出现的。可是，一直以来，电脑保安专家都认为要任意制造出冲撞需时太长，在实际情况上不可能发生，而王小云等的发现可能会打破这个必然性。就这样，王小云在国际会议上首次宣布了她及她的研究小组的研究成果——对[MD4](http://baike.baidu.com/view/444142.htm)、MD5、HAVAL-128和[RIPEMD](http://baike.baidu.com/view/260854.htm)等四个著名密码算法的破译结果。

在公布到第三个成果的时候，会场上已经是掌声四起，报告不得不一度中断。报告结束后，所有与会专家对他们的突出工作报以长时间的掌声，有些学者甚至起立鼓掌以示他们的祝贺和敬佩。由于版本问题，作者在提交[会议论文](http://baike.baidu.com/view/3781591.htm)时使用的一组常数和先行标准不同，在发现这一问题之后，王小云教授立即改变了那个常数，在很短的时间内就完成了新的数据分析，这段有惊无险的小插曲更证明了他们论文的信服力，攻击方法的有效性，验证了研究工作的成功。

2005年8月，王小云、[姚期智](http://baike.baidu.com/view/575406.htm)，以及姚期智妻子[姚储枫](http://baike.baidu.com/view/1801633.htm)（即为Knuth起名[高德纳](http://baike.baidu.com/view/119444.htm)的人）联手于国际密码讨论年会尾声部份提出SHA-1杂凑函数杂凑冲撞演算法的改良版。此改良版使破解SHA-1时间缩短。

2006年6月8日，[王小云](http://baike.baidu.com/view/350813.htm)教授于中国科学院第13次院士大会和[中国工程院](http://baike.baidu.com/view/11142.htm)第8次院士大会上以“国际通用Hash函数的破解”获颁[陈嘉庚科学奖](http://baike.baidu.com/view/262790.htm)信息技术科学奖。

2009年，冯登国、谢涛二人利用差分攻击，将MD5的碰撞算法复杂度从王小云的2^42进一步降低到2^21，极端情况下甚至可以降低至2^10。仅仅2^21的复杂度意味着即便是在2008年的计算机上，也只要几秒便可以找到一对碰撞。[1]

空字符串

MD5("")= d41d8cd98f00b204e9800998ecf8427e[3]

PHP md5() 函数定义和用法

md5() 函数计算字符串的 MD5 散列。[4]

md5() 函数使用 RSA 数据安全，包括 MD5 报文摘要算法。

来自 RFC 1321 的解释 - MD5 报文摘要算法：*MD5 报文摘要算法将任意长度的信息作为输入值，并将其换算成一个 128 位长度的"指纹信息"或"报文摘要"值来代表这个输入值，并以换算后的值作为结果。MD5 算法主要是为数字签名应用程序而设计的；在这个数字签名应用程序中，较大的文件将在加密（这里的加密过程是通过在一个密码系统下[如：RSA]的公开密钥下设置私有密钥而完成的）之前以一种安全的方式进行压缩。*

如需计算文件的 MD5 散列，请使用 md5\_file() 函数。

语法

md5(*string,raw*)

参数描述*string*必需。规定要计算的字符串。*raw*可选。规定十六进制或二进制输出格式：

* TRUE - 原始 16 字符二进制格式
* FALSE - 默认。32 字符十六进制数

技术细节

返回值：

如果成功则返回已计算的 MD5 散列，如果失败则返回 FALSE。

PHP 版本：

4+

更新日志：

在 PHP 5.0 中，*raw* 参数变成可选的。

实例

输出 md5() 的结果：

<?php$str="Hello";echo"Thestring:".$str."<br>";echo"TRUE-Raw16characterbinaryformat:".md5($str,TRUE)."<br>";echo"FALSE-32characterhexnumber:".md5($str)."<br>";?>