SM3 算法要经过填充、消息扩展、迭代压缩以及生成杂凑值。

1. 填充

将输入为 l 比特的消息 m 填充为 512bit 的倍数，首先填充一个“1”，再填充 k 个“0”，满足（l+1+k）mod 512=448;然后再添加 64bit 的消息长度。若剩余不足 64bit 可多延一些 bit 数，达到 1024bit。对填充的字符串先转化为 16 进制再进行操作，需要注意的是 bit 数为二进制的长度。（此处代码有参考）

```
//填充
string padding(string s) {
    string s_hex = "";
    for (int i = 0; i < s.size(); i++)
    {
        s_hex += int2hex((int)s[i]);
    }
    int s_l = s_hex.size() * 4;//二进制长度
    s_hex = s_hex + "8";//加1
    while (s_hex.size() % 128 != 112) {//l+1+k=448mod512
        s_hex = s_hex + "0";//加0
    }
    string s_len = int2hex(s_l);//数据长度十六进制
    while (s_len.size() != 16) //长度为64bit
    {
        s_len = "0" + s_len;
    }
    s_hex = s_hex + s_len;
    return s_hex;
}
```

2. 消息扩展

要生成 132 个消息字，每个消息字为 32bit，即 8 个十六进制数。首先，要将 512bit 的消息分成 16 个消息字，并利用这 16 个消息字生成另外的消息字。对W_{16}到W_{67}，利用$W_j = P_1(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} <<< 15) \oplus (W_{j-13} <<< 7) \oplus W_{j-6})$计算。对于$W'_0$到$W'_{63}$，利用$W'_j = W_j \oplus W_{j+4}$可计算得到。

其中$P_1(X) = X \oplus (X <<< 15) \oplus (X <<< 23)$。

```
//消息扩展
for (int i = 0; i < 16; i++)
{
    W[i] = str2uint(str.substr(8 * i, 8));
}
for (int i = 16; i < 68; i++)
{
    W[i] = P_1(W[i - 16] ^ W[i - 9] ^ Left(W[i - 3], 15)) ^ Left(W[i - 13], 7) ^ W[i - 6];
}
for (int i = 0; i < 64; i++)
{
    W_1[i] = W[i] ^ W[i + 4];
}
```

```
uint32_t P_1(uint32_t x)
{
    return (x ^ Left(x, 15) ^ Left(x, 23));
}
```

3. 压缩函数

令 A,B,C,D,E,F,G,H 分别等于 V[i](0=<i=<7)，对 j 从 0 到 63，执行：$SS1 = ((A <<< 12) + E + (T_j <<< j)) <<< 7, SS2 = SS1 \oplus (A <<< 12), TT1 = FF_j(A,B,C) + D + SS2 + W'_j, TT2 = GG_j(E,F,G) + H + SS1 + W_j, D = C, C = B <<< 9, B = A, A = TT1, H = G, G = F <<< 19, F = E, E = P\_0(TT2)$.

然后，再将 A,B,C,D,E,F,G,H 分别与 V[i](0=<i=<7)异或得到新的 V[i](0=<i=<7)。

```
A = V[0];
B = V[1];
C = V[2];
D = V[3];
E = V[4];
F = V[5];
G = V[6];
H = V[7];
```

```
for (int j = 0; j < 64; j++)
{
    SS1 = (Left((Left(A, 12) + E + Left(Ti(j), j)), 7));
    SS2= SS1 ^ Left(A, 12);
    TT1 = FF(A, B, C, j) + D + SS2 + W_1[j];
    TT2 = GG(E, F, G, j) + H + SS1 + W[j];
    D = C;
    C = Left(B, 9);
    B = A;
    A = TT1;
    H = G;
    G = Left(F, 19);
    F = E;
    E = P_0(TT2);
}
V[0] = A ^ V[0];
V[1] = B ^ V[1];
V[2] = C ^ V[2];
V[3] = D ^ V[3];
V[4] = E ^ V[4];
V[5] = F ^ V[5];
V[6] = G ^ V[6];
V[7] = H ^ V[7];
```

其中$P_0(X) = X \oplus (X \lll 9) \oplus (X \lll 17)$.

```
uint32_t P_0(uint32_t x)
{
    return (x ^ Left(x, 9) ^ Left(x, 17));
}
```

$T_j = 79cc4519(0 =< j =< 15); 7a879d8a(16 =< j =< 63)$

```
uint32_t T[2] = { 0x79CC4519,0x7A879D8A };

//T_i
uint32_t Ti(int i) {
    if(i<=15)
        return T[0];
    else
        return T[1];
}
```

$FF_j(X,Y,Z) = X \oplus Y \oplus Z(0 =< j =< 15); (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)(16 =< j =< 63)$

```
//FF_i
uint32_t FF(uint32_t X, uint32_t Y, uint32_t Z, int j)
{
    uint32_t ffi = 0;
    if (j >= 0 && j <= 15)
        ffi = X ^ Y ^ Z;
    else if (j >= 16 && j <= 63)
        ffi = (X & Y) | (X & Z) | (Y & Z);
    return ffi;
}
```

$GG_j(X,Y,Z) = X \oplus Y \oplus Z(0 =< j =< 15); (X \wedge Y) \vee (\neg X \wedge Z)(16 =< j =< 63)$

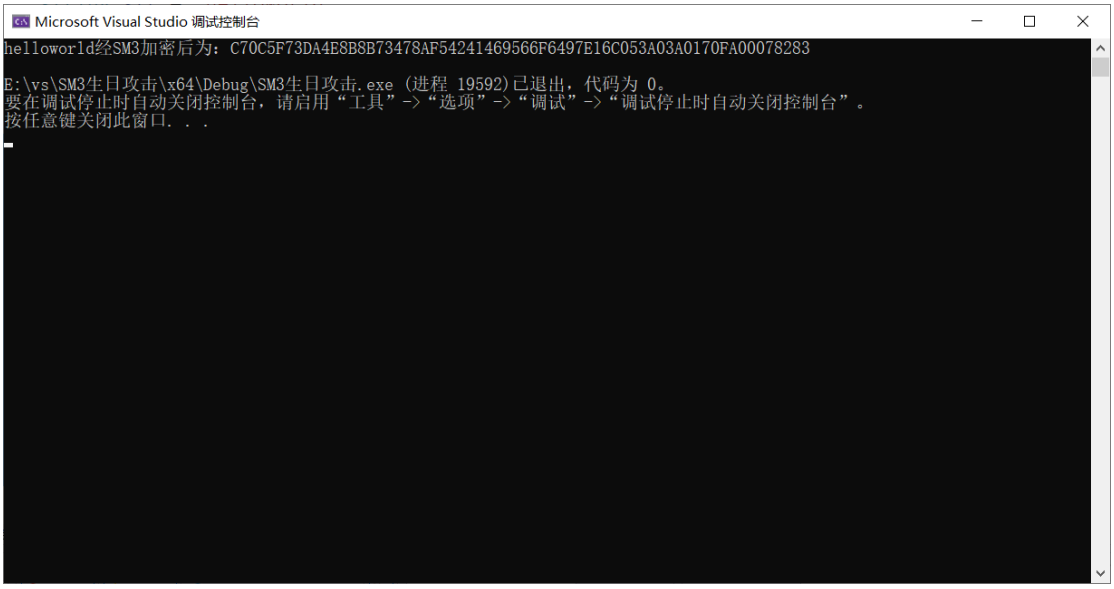
```
uint32_t GG(uint32_t X, uint32_t Y, uint32_t Z, int j)
{
    uint32_t ggi = 0;
    if (j >= 0 && j <= 15)
        ggi = X ^ Y ^ Z;
    else if (j >= 16 && j <= 63)
        ggi = (X & Y) | ((~X) & Z);
    return ggi;
}
```

4. 求杂凑值

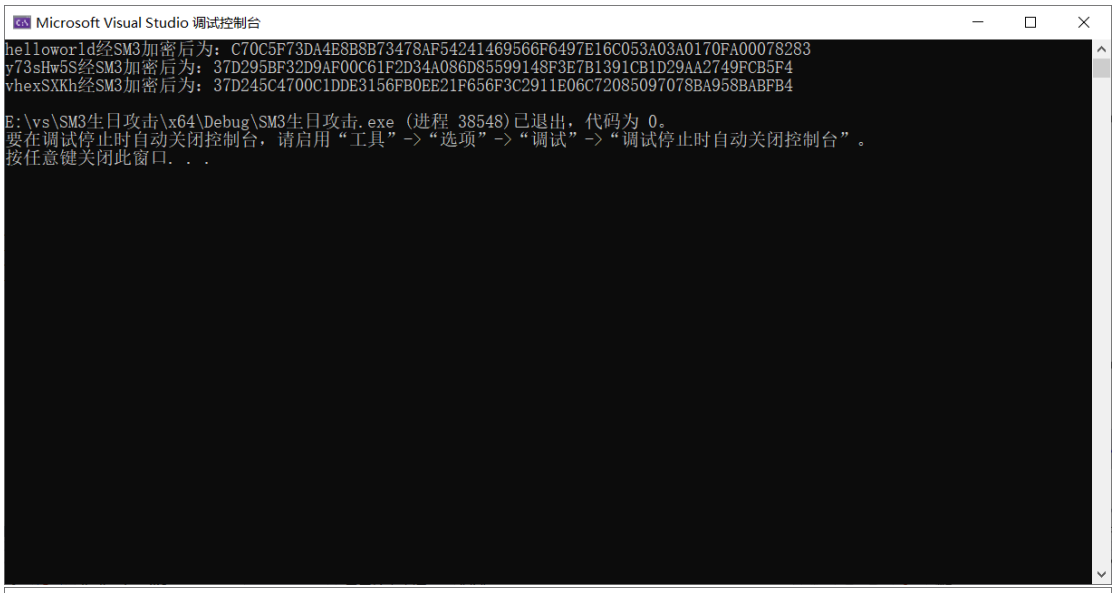
利用上述流程进行迭代压缩求值，其中 IV=7380166f 4914b2b9 172442d7 da8a0600 a96f30bc 163138aa e38dee4d b0fb0e4e。

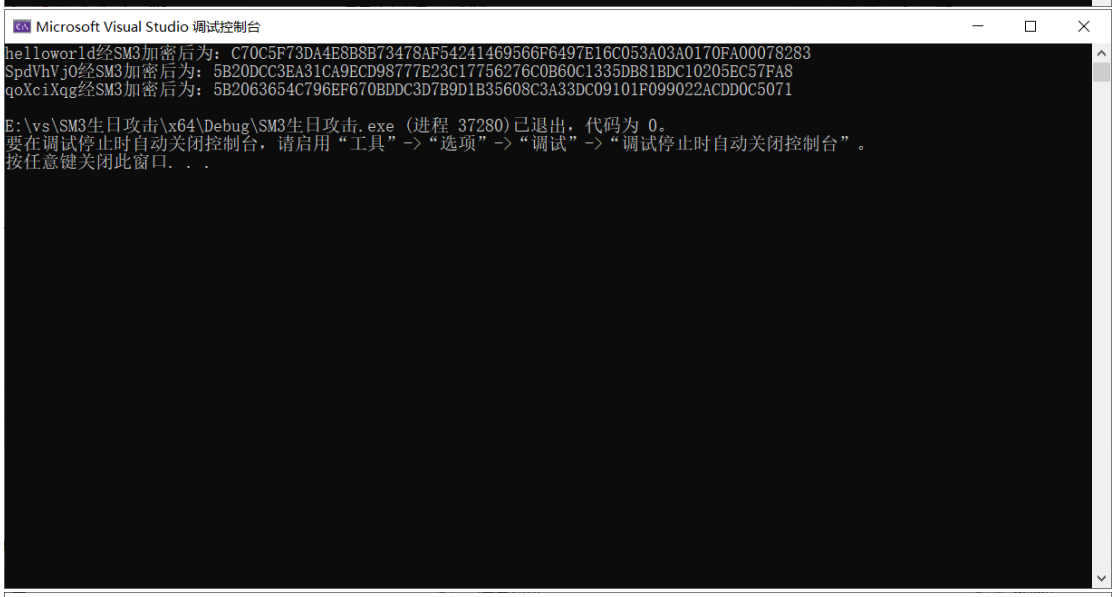
```
//迭代实现
string SM3(string s)
{
    //IV=7380166f 4914b2b9 172442d7 da8a0600 a96f30bc 163138aa e38dee4d b0fb0e4e
    uint32_t V[8] = { 0x7380166F, 0x4914B2B9, 0x172442D7, 0xDA8A0600, 0xA96F30BC, 0x163138AA, 0xE38DEE4D, 0xB0FB0E4E };
    int size = int(s.size()) * 4;//二进制长度
    int n = (size + 1) % 512;
    int k;
    if (n < 448)
        k = 448 - n;
    else//比特不足以加上64bit长度
        k = 960 - n;
    s = padding(s);
    int num = (size + k + 65) / 512;//分块
    string B="";
    for (int i = 0; i < num; i++)
    {
        B = s.substr(i * 128, 128);
        CF(B,V);
    }
```

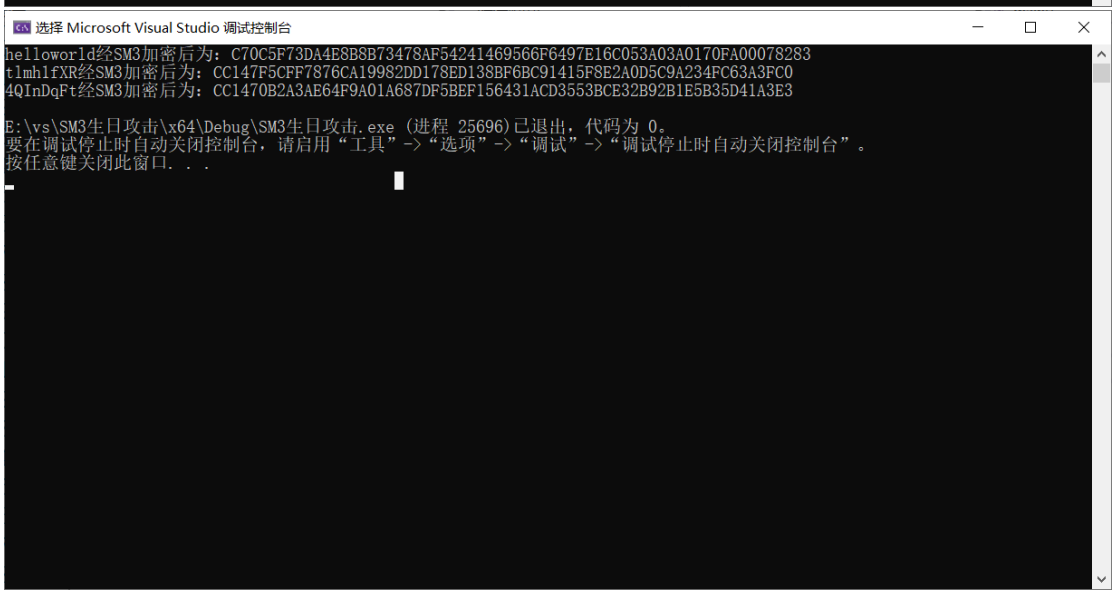
我们可以实验代入“helloworld”时输出的杂凑值为C70C5F73DA4E8B8B73478AF54241469566F6497E16C053A03A0170FA00078283。



生日攻击：随机生成两个字符串，找到加密后杂凑值相同的两个并输出，我们这里取加密后前 16、20bit 相同即输出。前两张图为前 16bit 相同即十六进制前 4 位数，后一张为前 20bit 相同即 16 进制前 5 位数。


Microsoft Visual Studio 调试控制台
helloworld经SM3加密后为：C70C5F73DA4E8B8B73478AF54241469566F6497E16C053A03A0170FA00078283
y73sHw5S经SM3加密后为：37D295BF32D9AF00C61F2D34A086D85599148F3E7B1391CB1D29AA2749FCB5F4
vhexSXKh经SM3加密后为：37D245C4700C1DDE3156FB0EE21F656F3C2911E06C72085097078BA958BABFB4
E:\vs\SM3生日攻击\x64\Debug\SM3生日攻击.exe (进程 38548)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .


Microsoft Visual Studio 调试控制台
helloworld经SM3加密后为：C70C5F73DA4E8B8B73478AF54241469566F6497E16C053A03A0170FA00078283
SpdVhVjO经SM3加密后为：5B20DCC3EA31CA9ECD98777E23C17756276C0B60C1335DB81BDC10205EC57FA8
qoXciXqg经SM3加密后为：5B2063654C796EF670BDDC3D7B9D1B35608C3A33DC09101F099022ACDD0C5071
E:\vs\SM3生日攻击\x64\Debug\SM3生日攻击.exe (进程 37280)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .


选择 Microsoft Visual Studio 调试控制台
helloworld经SM3加密后为：C70C5F73DA4E8B8B73478AF54241469566F6497E16C053A03A0170FA00078283
tlmhlfXR经SM3加密后为：CC147F5CFF7876CA19982DD178ED138BF6BC91415F8E2A0D5C9A234FC63A3FC0
4QInDqFt经SM3加密后为：CC1470B2A3AE64F9A01A687DF5BEF156431ACD3553BCE32B92B1E5B35D41A3E3
E:\vs\SM3生日攻击\x64\Debug\SM3生日攻击.exe (进程 25696)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .

（有参考）

```
//生成随机字符串
string randomstr(int l)
{
    static string c= "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
    string ret = "";
    std::mt19937 my_rand(std::random_device{}());//利用伪随机数生成器
    ret.resize(l);
    uint32_t len = c.length();
    for (int i = 0; i < l; i++)
    {
        ret += c[my_rand() % (len - 1)];
    }
    return ret;
}
```

```
//生日攻击
while (1)
{
    string a = randomstr(8);
    string b = randomstr(8);
    string result_a = SM3(a).substr(0, 6);//取前20bit相同
    string result_b = SM3(b).substr(0, 6);
    if (result_a == result_b)
    {
        cout << a << "经SM3加密后为：" << SM3(a) << endl;
        cout << b << "经SM3加密后为：" << SM3(b) << endl;
        break;
    }
}
```