

由于 SM3 和 SHA256 均为 Merkel-Damgard 散列函数，其满足加密前将待加密的明文按一定规则扩展到规定长度（SM3 为 512bit）的倍数，且按照固定长度将明文进行分块，前一个块的加密结果为后一个块的初始向量，因此满足这种条件的均可以被长度扩展攻击，对于 $SM3(salt+data)$ ，不知道 salt 但是知道 salt 长度，且满足 Merkel-Damgard 散列函数的情况下，并且 data 可以被任意控制的情况下，可以计算出 $SM3(salt+data)$ 。由于 SM3 需要先填充再运算，且后一个块的初始变量为前一个块的加密结果，因此我们可以把 $SM(salt+data)$ 的值替换为后一个块的初始 IV，这样就可以计算 $SM3(salt+data+padding+append)$ ，此时满足 $SM3(salt+data+padding+append)=SM3(SM3(salt+data),append)$ 。

新消息=原消息+原消息填充+补充的消息，可以计算出新消息的 SM3 加密后的杂凑值。

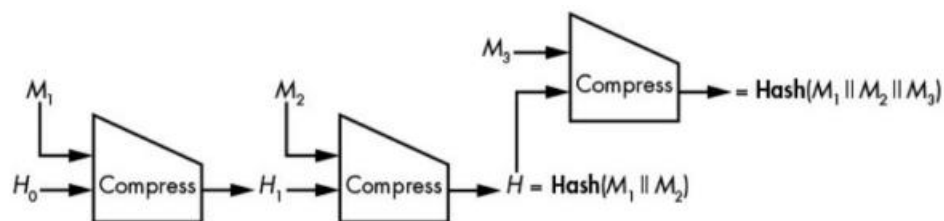


Figure 6-9: The length-extension attack

由于 SM3 输出的是十六进制字符串，我的字符串经过 padding 之后输出的也为十六进制字符串，这里再写一个以十六进制字符串为输入的 SM3_2,只需将 padding 进行变化，直接填充十六进制字符，不必先转为十六进制字符串再填充。

```

//假设输入为十六进制字符串
string padding2(string s)
{
    int s_l = s.size() * 4; //二进制长度
    s = s + "8"; //加1
    while (s.size() % 128 != 112) { //1+1+k=448mod512
        s = s + "0"; //加0
    }
    s = s + uint2hex(s_l, 16);
    return s;
}
  
```

将 SM3 中的填充换成 padding2，由于十六进制字符串与字符串类似，其一位都由 4bit 二进制表示，所以其他部分并不修改。

```
string SM3_2(string s)
{
    uint32_t V[8] = { 0x7380166F, 0x4914B2B9, 0x172442D7, 0xDA8A0600, 0xA96F30BC, 0x163138AA, 0xE38DEE4D, 0
    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 = padding2(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);
    }
}
```

首先，我们取字符串 a="abc"，代入执行 SM3(a) 输出为 V 的十六进制字符串。

```
string a = "abc";
uint32_t V[8] = { 0x7380166F, 0x4914B2B9, 0x172442D7, 0xDA8A0600, 0xA96F30BC, 0x163138AA, 0xE38DEE4D, 0
int size = int(a.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;
a = padding(a);
//cout << a << endl;
int num = (size + k + 65) / 512; //分块
string B = "";
for (int i = 0; i < num; i++)
{
    B = a.substr(i * 128, 128);
    CF(B, V);
}
```

我们扩展的消息取为 b="def"，将 SM3(a) 的结果作为字符串 b 的初始向量 iv，继续进行加密，得到加密后的结果。

```

string b="def";
int size2 = int(b.size()) * 4;//二进制长度
int n2 = (size2 + 1) % 512;
int k2;
if (n2 < 448)
    k2 = 448 - n2;
else//比特不足以加上64bit长度
    k2 = 960 - n2;
b = padding(b);
int num2 = (size2 + k2 + 65) / 512;//分块
string B2 = "";
for (int i = 0; i < num2; i++)
{
    B2 = b.substr(i* 128, 128);
    CF(B2, V);
}

```

将 a 以及填充字符串以及 b 合成新的字符串（十六进制），即，此时 $c=a+b$

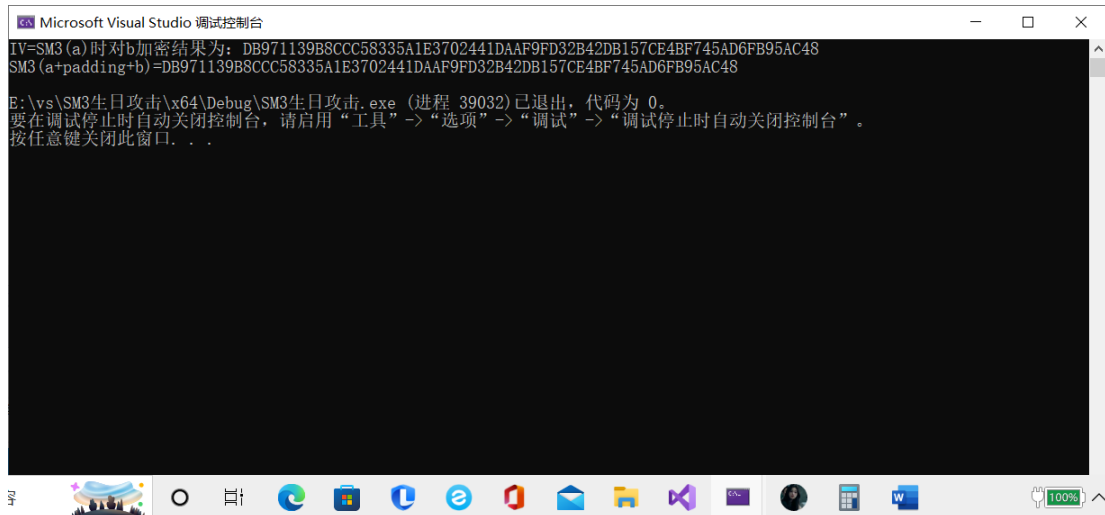
（将 b 转为 16 进制字符串表示形式，此处 a 为已经经过填充后的 a），带入输入为十六进制字符串的 SM3_2，得到结果。

```

string V_n = "";
for (int i = 0; i < 8; i++)
{
    V_n += uint2hex(V[i], 8);
}
string c = a + "646566";
//cout << c << endl;
cout << "IV=SM3(a)时对b加密结果为: " << V_n << endl;
cout << "SM3(a+padding+b)=" << SM3_2(c) << endl;

```

最后两个结果相同，即为长度扩展成功，即 $SM3(iv, salt+data+padding+append)=SM3(iv=SM3(iv, salt+data), append)$ 。

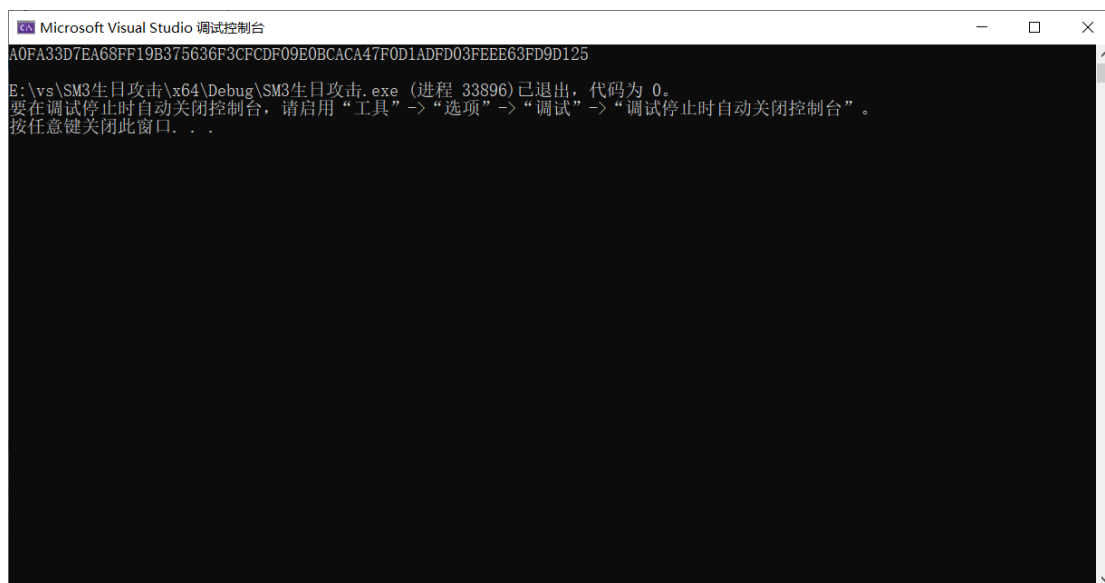


```
Microsoft Visual Studio 调试控制台
IV=SM3(a)时对b加密结果为: DB971139B8CCC58335A1E3702441DAAF9FD32B42DB157CE4BF745AD6FB95AC48
SM3(a+padding+b)=DB971139B8CCC58335A1E3702441DAAF9FD32B42DB157CE4BF745AD6FB95AC48

E:\vs\SM3生日攻击\x64\Debug\SM3生日攻击.exe (进程 39032) 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...
```

其中, 取以下字符串, 若最后能输出 d, 也可以说明 SM3 能被长度扩展攻击, 由于 sm3 输出为 16 进制字符串, 我们将 string 全部转为十六进制字符串带输入为十六进制字符串的 SM3_2 中计算。

```
//长度扩展攻击
string M1 = "abc";
string M2 = SM3(M1);
string M3 = str2hex("abc");
string d = SM3_2(M2+M3);
cout << d << endl;
```



```
Microsoft Visual Studio 调试控制台
A0FA33D7EA68FF19B375636F3CFCD09E0BCACA47F0D1ADFD03FEE63FD9D125

E:\vs\SM3生日攻击\x64\Debug\SM3生日攻击.exe (进程 33896) 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...
```

本次实验中, 字符串转换代码有借鉴。

```

//十六进制转为字符串
string hex2str(const string& hex)
{
    string str;
    for (size_t i = 0; i < hex.length(); i = i + 2)
    {
        string byte = hex.substr(i, 2);
        char chr = (char)(int)strtol(byte.c_str(), NULL, 16);
        str.push_back(chr);
    }
    return str;
}

//字符串转为十六进制
string str2hex(string s)
{
    string ret;
    static const char* hex = "0123456789ABCDEF";
    for (auto i : s)
    {
        ret.push_back(hex[(i >> 4) & 0xf]);
        ret.push_back(hex[i & 0xf]);
    }
    return ret;
}

```