第一次编程作业实验报告

刘曼姝 1901210656 2019.10.15

一、 实验目的与要求：

实现对 SM3 哈希函数做长度扩展攻击的可展示实例。

二、 实验设备、系统环境与软件：

Windows 10 64 位操作系统的笔记本电脑，安装了 ActivePerl 5.20.2、Visual Studio 2019、GmSSL 2.5.4、Notepad++。

三、 实验原理：

1、SM3 简介

SM3 是国产的哈希函数（标准），在商用密码体系中主要用于数字签名及验证、消息认证码生成及验证、随机数生成等，其算法公开[1]，在 2010 年作为国家加密标准发布，使用于敏感但非机密的应用中。SM3 生成 256 位哈希值，使用 128 位分组和 256 位椭圆曲线密码算法（公钥密码算法）。

2、SM3 算法[2]

（0）SM3 算法中运用的符号、常量与函数定义：

$ABCDEFGH$：8个字寄存器或它们的值的串联
$B^{(i)}$：第i个消息分组
CF：压缩函数
FF_j：布尔函数，随j的变化取不同的表达式
GG_j：布尔函数，随j的变化取不同的表达式
IV：初始值，用于确定压缩函数寄存器的初态
P_0：压缩函数中的置换函数
P_1：消息扩展中的置换函数
T_j：常量，随j的变化取不同的值
m：消息
m'：填充后的消息
mod：模运算
$\wedge$：32比特与运算
$\vee$：32比特或运算

$\oplus$：32比特异或运算

$\neg$：32比特非运算

+：mod2^{32}算术加运算

$\lll k$：循环左移k比特运算

$\leftarrow$：左向赋值运算符

初始值

IV=7380166f 4914b2b9 172442d7 da8a0600 a96f30bc 163138aa e38dee4d b0fb0e4e

常量

$$T_j = \begin{cases} 79cc4519 & 0 \le j \le 15 \\ 7a879d8a & 16 \le j \le 63 \end{cases}$$

布尔函数

$$FF_j(X,Y,Z) = \begin{cases} X \oplus Y \oplus Z & 0 \le j \le 15 \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) & 16 \le j \le 63 \end{cases}$$

$$GG_j(X,Y,Z) = \begin{cases} X \oplus Y \oplus Z & 0 \le j \le 15 \\ (X \wedge Y) \vee (\neg X \wedge Z) & 16 \le j \le 63 \end{cases}$$

式中X,Y,Z 为字。

置换函数

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

$$P_1(X) = X \oplus (X \lll 15) \oplus (X \lll 23)$$

式中X为字。

（1）填充：

假设消息 m 长度为 Lbit。首先将 1bit 的“1”添加到消息末尾，然后再添加 k 个“0”，k 是满足 L+1+k≡448(mod 512)的最小非负整数，最后再添加 64bit 用来表示长度 L（二进制），填充后的消息 m′的比特长度即为 512 的倍数。

（2）迭代压缩：

① 迭代：

将填充后的消息m'按512比特进行分组：$m' = B^{(0)}B^{(1)}\cdots B^{(n-1)}$

其中$n=(l+k+65)/512$。

对m'按下列方式迭代：

FOR i=0 TO n-1

$V^{(i+1)} = CF(V^{(i)}, B^{(i)})$

ENDFOR

其中*CF*是压缩函数，$V^{(0)}$为256比特初始值*IV*，$B^{(i)}$为填充后的消息分组，迭代压缩的结果为$V^{(n)}$。

② 消息扩展：

将消息分组$B^{(i)}$按以下方法扩展生成132个字$W_0, W_1, \cdots, W_{67}, W'_0, W'_1, \cdots, W'_{63}$，用于压缩函数$CF$：

a)将消息分组$B^{(i)}$划分为16个字$W_0, W_1, \cdots, W_{15}$。

b)FOR j=16 TO 67

$W_j \leftarrow P_1(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} \lll 15)) \oplus (W_{j-13} \lll 7) \oplus W_{j-6}$

ENDFOR

c)FOR j=0 TO 63

$W'_j = W_j \oplus W_{j+4}$

ENDFOR

③ 压缩：

令*A,B,C,D,E,F,G,H*为字寄存器,*SS*1,*SS*2,*TT*1,*TT*2为中间变量,压缩函数$V^{i+1} = CF(V^{(i)}, B^{(i)}), 0 \leq i \leq n-1$。计算过程描述如下：

$ABCDEFGH \leftarrow V^{(i)}$

FOR j=0 TO 63

$SS1 \leftarrow ((A \lll 12) + E + (T_j \lll j)) \lll 7$

$SS2 \leftarrow SS1 \oplus (A \lll 12)$

$TT1 \leftarrow FF_j(A, B, C) + D + SS2 + W'_j$

$TT2 \leftarrow GG_j(E, F, G) + H + SS1 + W_j$

$D \leftarrow C$

$C \leftarrow B \lll 9$

$B \leftarrow A$

$A \leftarrow TT1$

$H \leftarrow G$

$G \leftarrow F \lll 19$

$F \leftarrow E$

$E \leftarrow P_0(TT2)$

ENDFOR

$V^{(i+1)} \leftarrow ABCDEFGH \oplus V^{(i)}$

其中，字的存储为大端格式。

④ 杂凑值：

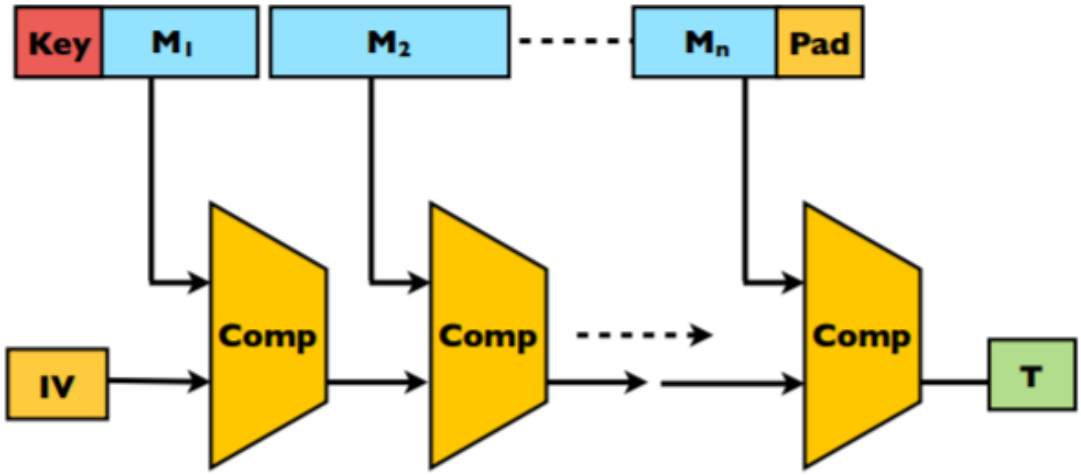
$ABCDEFGH \leftarrow V^{(n)}$

输出256比特的杂凑值$y = ABCDEFGH$。

3、长度扩展攻击[3]

长度扩展攻击（length extension attack）指针对某些允许包含额外信息的加密哈希函数的攻击手段，攻击者可以在不知道密钥 K 的情况下，附加任何数据并生成有效的哈希。Merkle–Damgård 形式的哈希函数易受长度扩展攻击，SM3 就属于这种形式。

Merkle–Damgård 形式的哈希函数原理如图，可表示成 T=H(K || M)形式：

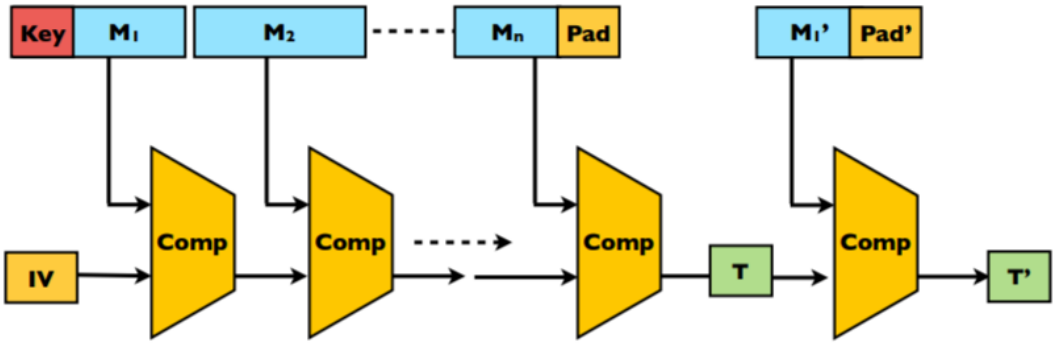


对于 T=H(K || M)形式的哈希，在以下条件满足的情况下，攻击者可以通过该方法获取“H(K || 一定规则构造的 M)”：

a. 知道 H 的算法且该算法满足 Merkle–Damgård 哈希函数特征；

b. 不知道或不可控 K 的具体值，但知道 K 的长度，并可控制 M 的值；

c. 可以得到 T=H(K || M)的值。



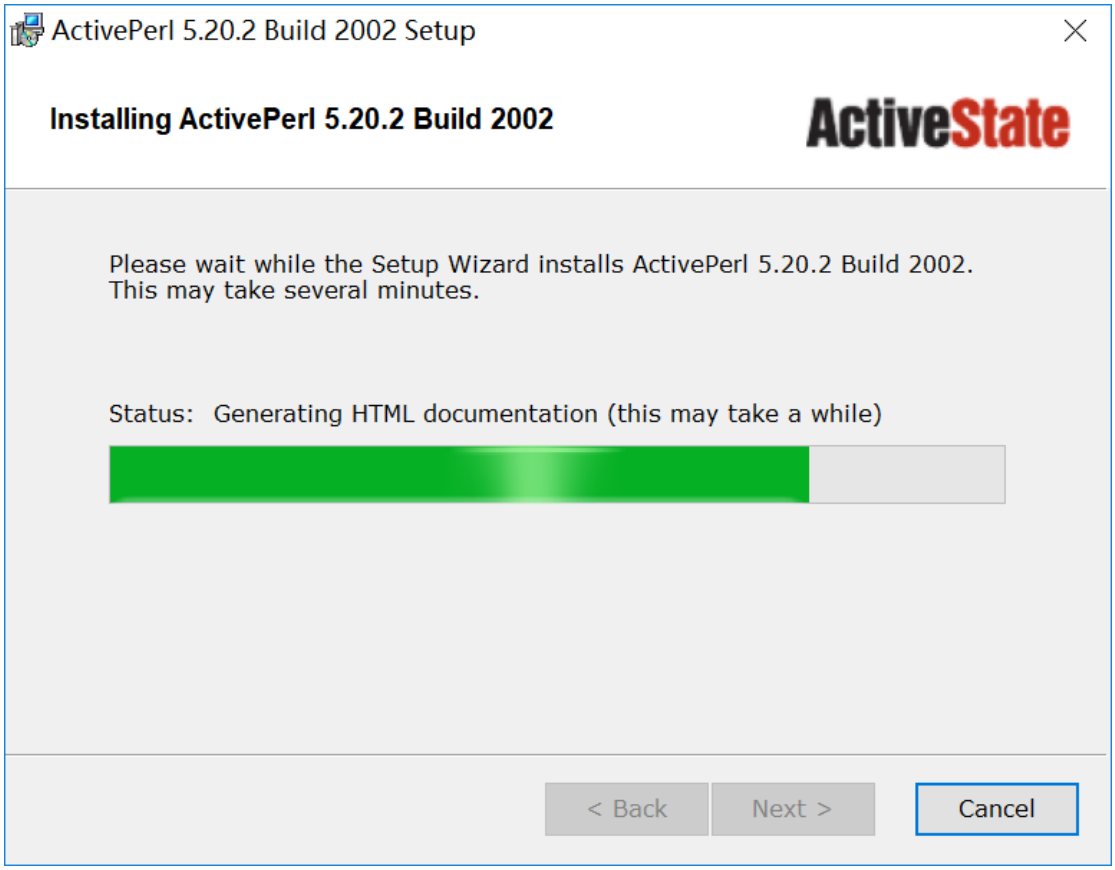
即可根据上图，通过已知的 K 的长度，确定 Pad 填充，并将扩展的消息 M’附加在后面，形成 T’ = Hash(K || M || Pad || M’)。而在攻击的实际操作中，将对 M’操作的这一轮的初始向量 IV 改为 H(K || M)，即可得到与 T’ = Hash(K || M || Pad ||

M')相等的哈希值，从而伪造有效的哈希值。

四、 实验步骤与设计思路：

1、 GmSSL 在 Window 下的编译和安装[4]

（1） 安装 ActivePerl（Visual Studio 之前已安装）：



（安装好后，如果没有勾选设置路径，可以输入命令：

set path=C:\Users\0\Downloads\ActivePerl\perl\bin;%path%;

设置 perl 路径；输入命令：

perl -v

查看是否设置正确。）

（2）以管理员身份打开 Visual Studio Tools 下的 Developer Command Prompt 控制台并运行命令：

perl C:\Users\0\Desktop\GmSSL-master\Configure VC-WIN32

```
C:\Windows\System32>perl C:\Users\0\Desktop\GmSSL-master\Configure VC-WIN32
Configuring GmSSL version 2.5.4 (0x1010004fL)
    no-asan         [default]  OPENSSL_NO_ASAN
    no-crypto-mdebug [default]  OPENSSL_NO_CRYPTO_MDEBUG
    no-crypto-mdebug-backtrace [default]  OPENSSL_NO_CRYPTO_MDEBUG_BACKTRACE
    no-ec_nistp_64_gcc_128 [default]  OPENSSL_NO_EC_NISTP_64_GCC_128
    no-egd          [default]  OPENSSL_NO_EGD
    no-fuzz-afl     [default]  OPENSSL_NO_FUZZ_AFL
    no-fuzz-libfuzzer [default]  OPENSSL_NO_FUZZ_LIBFUZZER
    no-gmieng       [default]  OPENSSL_NO_GMIENG
    no-heartbeats   [default]  OPENSSL_NO_HEARTBEATS
    no-md2          [default]  OPENSSL_NO_MD2 (skip dir)
    no-msan         [default]  OPENSSL_NO_MSAN
    no-rc5          [default]  OPENSSL_NO_RC5 (skip dir)
    no-sctp         [default]  OPENSSL_NO_SCTP
    no-sdfeng       [default]  OPENSSL_NO_SDFENG
    no-skfeng       [default]  OPENSSL_NO_SKFENG
    no-ssl-trace    [default]  OPENSSL_NO_SSL_TRACE
    no-ssl3         [default]  OPENSSL_NO_SSL3
    no-ssl3-method  [default]  OPENSSL_NO_SSL3_METHOD
    no-ubsan        [default]  OPENSSL_NO_UBSAN
    no-unit-test    [default]  OPENSSL_NO_UNIT_TEST
    no-weak-ssl-ciphers [default]  OPENSSL_NO_WEAK_SSL_CIPHERS
    no-zlib         [default]
    no-zlib-dynamic [default]
Configuring for VC-WIN32
```

```
It looks like you don't have either nmake.exe or dmake.exe on your PATH,
so you will not be able to execute the commands from a Makefile.  You can
install dmake.exe with the Perl Package Manager by running:

    ppm install dmake
```

```
CC              =cl
CFLAG           =-W3 -wd4090 -Gs0 -GF -Gy -nologo -DOPENSSL_SYS_WIN32 -DWIN32_LEAN_AND_MEAN -DL_ENDIAN -D_CRT_SECURE_NO_DEPRECATE -DUNICODE -D_UNICODE /MD /O2
SHARED_CFLAG    =
DEFINES         =OPENSSL_USE_APPLINK DSO_WIN32 NDEBUG OPENSSL_THREADS OPENSSL_NO_STATIC_ENGINE OPENSSL_PIC OPENSSL_BN_ASM_PART_WORDS OPENSSL_IA32_SSE2 OPENSSL_
BN_ASM_MONT OPENSSL_BN_ASM_GF2m SHA1_ASM SHA256_ASM SHA512_ASM RC4_ASM MD5_ASM RMD160_ASM AES_ASM VPAES_ASM WHIRLPOOL_ASM GHASH_ASM ECP_NISTZ256_ASM PADLOCK_
ASM GMI_ASM POLY1305_ASM
LFLAG           =/nologo /debug
PLIB_LFLAG      =
EX_LIBS         =ws2_32.lib gdi32.lib advapi32.lib crypt32.lib user32.lib
APPS_OBJ        =win32_init.o ../ms/applink.o
CPUID_OBJ       =x86cpuid.o
UPLINK_OBJ      =../ms/uplink.o
BN_ASM          =bn-586.o co-586.o x86-mont.o x86-gf2m.o
EC_ASM          =ecp_nistz256.o ecp_nistz256-x86.o
DES_ENC         =des-586.o crypt586.o
AES_ENC         =aes-586.o vpaes-x86.o aesni-x86.o
BF_ENC          =bf-586.o
CAST_ENC        =c_enc.o
RC4_ENC         =rc4-586.o
RC5_ENC         =rc5-586.o
MD5_OBJ_ASM     =md5-586.o
SHA1_OBJ_ASM    =sha1-586.o sha256-586.o sha512-586.o
RMD160_OBJ_ASM=rmd-586.o
CMLL_ENC        =cmll-x86.o
MODES_OBJ       =ghash-x86.o
PADLOCK_OBJ     =e_padlock-x86.o
GMI_OBJ         =e_gmi-x86.o
CHACHA_ENC      =chacha-x86.o
POLY1305_OBJ    =poly1305-x86.o
BLAKE2_OBJ      =
PROCESSOR       =
RANLIB          =true
ARFLAGS         =/nologo
PERL            =C:\Perl\bin\perl.exe

THIRTY_TWO_BIT mode
BN_LLONG mode

Configured for VC-WIN32.
```

提示 PATH 中没有 nmake.exe、dmake.exe，需要安装，输入命令：

ppm install dmake

```
C:\Windows\System32>ppm install dmake
Downloading ActiveState Package Repository dbimage...done
Syncing site PPM database with .packlists...done
Downloading dmake-4.11.20080107...done
Unpacking dmake-4.11.20080107...done
Generating HTML for dmake-4.11.20080107...done
Updating files in site area...done
 130 files installed
```

然后输入命令：nmake

此时提示错误：

nasm: fatal: unable to open output file `crypto\aes\aes-586.obj'

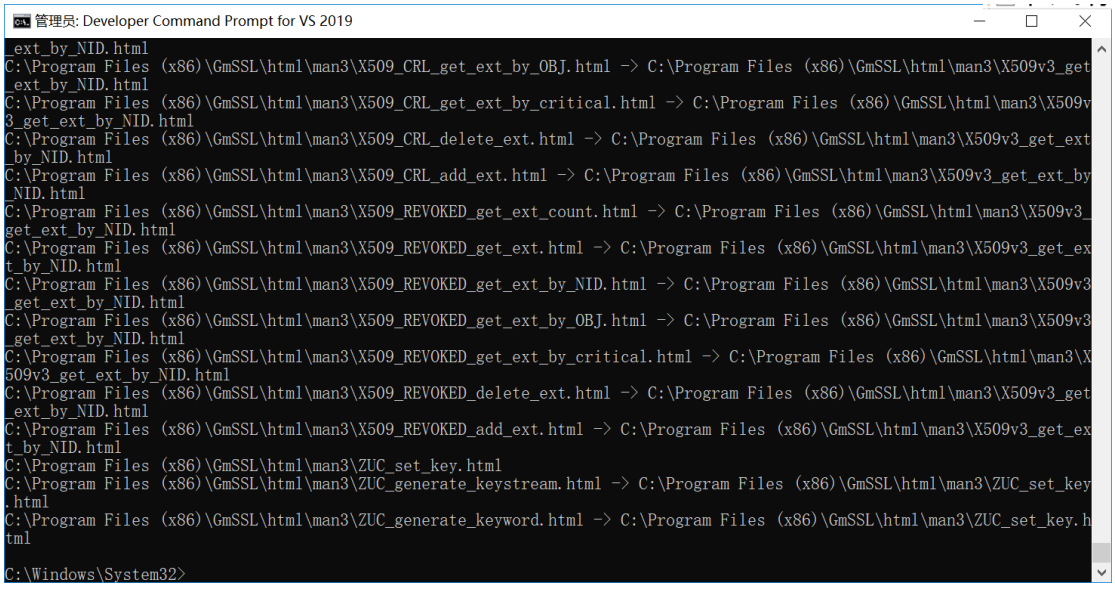
NMAKE : fatal error U1077: “C:\Users\0\AppData\Local\bin\NASM\nasm.EXE”: 返回代码“0x1”

解决方法是把上一步命令换成：

perl C:\Users\0\Desktop\GmSSL-master\Configure VC-WIN32 no-asm

再次 Configure，然后再输入命令：nmake。

最后输入命令：nmake install



安装之后执行 gmssl 命令行工具检查是否成功：

set path=C:\Program Files (x86)\GmSSL\bin;%path%;

gmssl version

```
C:\Windows\System32>set path=C:\Program Files (x86)\GmSSL\bin;%path%;

C:\Windows\System32>gmssl -v
Invalid command '-v'; type "help" for a list.

C:\Windows\System32>gmssl version
GmSSL 2.5.4 - OpenSSL 1.1.0d  3 Sep 2019

C:\Windows\System32>
```

说明安装成功。

2、SM3 的长度扩展攻击[5]

（1）获得原有“密钥 || 消息”的 SM3 哈希值 T=H（K || M）：

根据 T = Hash(K || M)，设密钥 K=“Passw0rd”，长度为 8；消息 M=“abc”，长度为 3；将 K || M 存储在 sm3.txt 中，其总长度为 11。

输入命令：

gmssl sm3 C:\Users\0\Desktop\sm3.txt

得到哈希值为：

“e0108e33795dc84661bcb9afcd3bf5d62cce9c97cb397f43073fc07afee23387”。

```
C:\Windows\System32>gmssl sm3 C:\Users\0\Desktop\sm3.txt
SM3(C:\Users\0\Desktop\sm3.txt)= e0108e33795dc84661bcb9afcd3bf5d62cce9c97cb397f43073fc07afee23387
```

（2） 求长度扩展攻击后应得的 SM3 哈希值 T’ = Hash(K || M || Pad || M’)：

（首先可以在 Notepad++中安装插件 HexEditor.dll，以方便打 16 进制数。）以 T’ = Hash(K || M || Pad || M’)的形式做长度扩展。因为 K || M 的总长度为 11 个字节，即 11*8=88=\x58bit，填充的最后还要预留 8 个字节的长度字段（SM3 采用大端存储），故不表示长度的填充字段应为 64-11-8=45 字节，因此：

Pad=“\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x58”； 设在此后填充的 M’=“abcdefgh”，然后将 K || M 连接在前面，形成“K || M || Pad || M’”：

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	50	61	73	73	77	30	72	64	61	62	63	80	00	00	00	00	Passw0rdabc€....
00000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	58	X
00000040	61	62	63	64	65	66	67	68									abcdefgh_

Passw0rdabc€ Xabcdefgh

存储在 sm31.txt 中。输入命令：

gmssl sm3 C:\Users\0\Desktop\sm31.txt

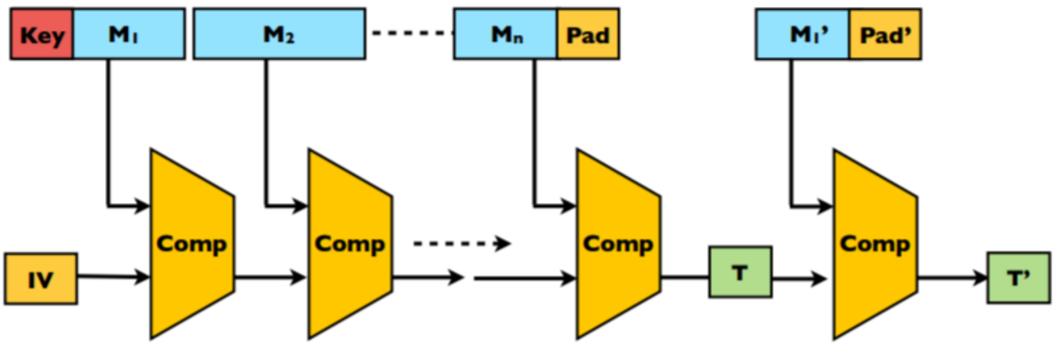
得到长度扩展后的哈希值应该为：

“aa84972131db07e8674a11ba46d373bce433b0f510d996808d314bf6e894dacd”。

```
C:\Windows\System32>gmssl sm3 C:\Users\0\Desktop\sm31.txt
SM3(C:\Users\0\Desktop\sm31.txt)= aa84972131db07e8674a11ba46d373bce433b0f510d996808d314bf6e894dacd
```

服务器将根据该值检查发送的数据。

（3） 执行实际的攻击：



根据该原理图，思路是求：“aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabcdefgh” 的哈希值，其中前 64 个字符可以是任意的，只要其长度是 64，能让 Hash 函数做一块运算即可。

修改程序，使得对如上字符串的前 64 个字符（一个整块）还以原来的初始向量 IV 做运算，而对于接下来超过 64 个字符的 M’部分，则将初始向量设置为 H(K || M)：“e0108e33795dc84661bcb9afcd3bf5d62cce9c97cb397f43073fc07afee23387”，以 4 个字节为一组：

```
context->intermediateHash[0] = 0x7380166F;
context->intermediateHash[1] = 0x4914B2B9;
context->intermediateHash[2] = 0x172442D7;
context->intermediateHash[3] = 0xDA8A0600;
context->intermediateHash[4] = 0xA96F30BC;
context->intermediateHash[5] = 0x163138AA;
context->intermediateHash[6] = 0xE38DEE4D;
context->intermediateHash[7] = 0xB0FB0E4E;
```

改为

```
context->intermediateHash[0] = 0xE0108E33;
context->intermediateHash[1] = 0x795DC846;
context->intermediateHash[2] = 0x61BCB9AF;
context->intermediateHash[3] = 0xCD3BF5D6;
context->intermediateHash[4] = 0x2CCE9C97;
context->intermediateHash[5] = 0xCB397F43;
context->intermediateHash[6] = 0x073FC07A;
context->intermediateHash[7] = 0xFEE23387;
```

。

运行程序得到如下实验最终结果：

```
Message:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabcdefgh
Hash:
   aa849721 31db07e8 674a11ba 46d373bc e433b0f5 10d99680 8d314bf6 e894dacd
请按任意键继续. . .
```

五、实验结果分析：

从实验结果可以看出，攻击者使用“任意 64 字节 ||M’=abcdefgh”计算出的 SM3 哈希值与之前计算出的 T’ = Hash(K || M || Pad || M’) 的值相一致，所不同的是，攻击者没有使用过密钥 K=Passw0rd，说明攻击成功了。

六、实验中的问题与总结：

本次实验我基于 Windows 10 下安装 GmSSL，以及基于 Visual Studio 使用 C 语言编程实现了对 SM3 哈希函数做长度扩展攻击的可展示实例。

实验中我遇到了几个问题。首先是安装 GmSSL 过程中遇到了一些错误，比如提示没有安装 NASM、WIN32 Console 模块、dmake 以及没有设置好路径等，这些都在尝试中解决了。然后是攻击的环节，由于我参考了 MD5 的长度扩展攻击，而 MD5 填充的长度字段的存储方式是小端的，一开始对于 SM3 的填充我也用小端方式进行了，因此造成了攻击不成功，后来改为大端方式，解决了这一问题；另外，填充需要输入 16 进制数，一开始我没有用 Notepad++的 16 进制插件输入，而是从网上复制的，造成编码可能有些问题，后来我用 16 进制插件输入，解决了这一问题。我还有一个在实际背景下应用的问题：实际情况中，一般是服务器生成密钥 K，虽然攻击者不需要知道 K 的值，但也需要知道 K 的长度，这一长度的获取是否容易。

【参考资料】

[1] https://baike.baidu.com/item/SM3/4421797?fr=aladdin （SM3_百度百科）.

[2] https://blog.csdn.net/a344288106/article/details/80094878 （SM3 密码杂凑算法实现及说明）.

[3] https://blog.csdn.net/szuaurora/article/details/78125585 （MD5 的长度扩展攻击详解，中文版）.

[4] http://gmssl.org/docs/install.html （GmSSL 在 Window 下的编译和安装）.

[5] https://github.com/iagox86/hash_extender（MD5 的长度扩展攻击详解，英文版）.