浙江大学

本科实验报告

课程名称： 计算机组成与设计

姓　　名： 吴江南

学　　院： 信息与电子工程学院

专　　业： 信息工程

学　　号： 3160104699

指导教师： 刘鹏

2018 年 11 月 13 日

目录

一、 系统需求分析.....1
1.1 项目描述.....1
1.2 开发工具.....1
二、 Office 文档结构和加密原理.....1
2.1 文档结构分析和数据提取.....1
2.1.1 docx 文件介绍.....1
2.1.2 数据提取.....1
2.2 Office 2010 加密技术.....2
2.2.1 Office 2010 加密技术和加密.....2
2.2.2 系统设计中涉及的加密算法.....2
2.2.3 加密算法详解.....3
2.3 SHA1 算法简介.....3
2.3.1 简介.....3
2.4 AES128 算法简介.....4
三、 系统设计.....5
3.1 总体设计思路.....5
3.1.1 解密思路.....5
3.1.2 具体实现.....5
3.2 子模块的功能实现.....6
3.2.1 文件读取.....6
3.2.2 Base64 解码.....6
3.2.3 SHA1 计算哈希值.....7
3.2.4 AES 加解密.....8
3.2.5 结果比较.....8
四、 解密测试.....9
4.1 Base64 解码测试.....9
4.2 解密测试.....9
五、 性能优化.....11
六、 心得与体会.....12
七、 参考文献.....12
[1]李玲双，加密文档的破解与实现，哈尔滨工业大学，2017.6.....12
八、 附录.....12
8.1 附件.....12
8.2 源代码.....12

一、系统需求分析

1.1项目描述

1. 编写 C 代码对所提供的 Microsoft office 2010 版本的 docx 文件进行解密.
2. 输入为文件路径，输出为解密后的密码.
3. 不能使用别人的解密代码.
4. 密码是三位数字，从 000 到 999.
5. 已提供文件: SHA-1 and AES function .

1.2 开发工具

1. 语言：C 语言
2. 软件：Dev-C++ 5.10
3. 编译环境：TDM-GCC 4.8.1 64-bit Debug

二、Office 文档结构和加密原理

2.1 文档结构分析和数据提取

2.1.1 docx 文件介绍

docx 格式的文件本质上是一个 ZIP 文件。将一个 docx 文件的后缀改为 ZIP 后是可以用解压工具打开或是解压的。事实上，Word2007 的基本文件就是 ZIP 格式的，他可以算作是 docx 文件的容器。

docx 格式文件的主要内容是保存为 XML 格式的，但文件并非直接保存于磁盘。它是保存在一个 ZIP 文件中，然后取扩展名为 docx。将.docx 格式的文件后缀改为 ZIP 后解压, 可以看到解压出来的文件夹中有 word 这样一个文件夹，它包含了 Word 文档的大部分内容。其中的 document.xml 文件则包含了文档的主要文本内容。

2.1.2 数据提取

系统只需用到其中的 EncryptionInfo.xml 文件中的第 2 个 saltValue、encryptedVerifierHashInput 和 encryptedVerifierHashValue.

提取方法：更改文档后缀名.docx 为.zip，然后采用解压工具解压即可得到 EncryptionInfo 文件，在程序运行中需要用户将 EncryptionInfo 文件保存到改程序目录下，然后再控制窗口中根据提示输入文件名：EncryptionInfo

2.2 Office 2010 加密技术

2.2.1 Office 2010 加密技术和加密

Office 可用的加密算法取决于可通过 Windows 操作系统中的 API（应用程序编程接口）访问的算法。Office 2010 除了继续支持加密 API (CryptoAPI) 之外，还支持通过 Service Pack 2 (SP2) 最先在 2007 Microsoft Office system 中提供的 CNG（CryptoAPI：下一代加密技术）。

CNG 可实现更灵活的加密，这时，可指定主机上支持的不同加密和哈希算法以便在文档加密过程中使用。CNG 还可以实现更高的扩展性加密，在此情况下，可使用第三方加密模块。

当 Office 使用 CryptoAPI 时，加密算法取决于 CSP（加密服务提供程序）中可用的那些加密算法，CSP 是 Windows 操作系统的一部分。以下注册表项包含了计算机上安装的 CSP 的列表：

HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Cryptography/Defaults/Provider

以下 CNG 加密算法或系统上安装的任何其他 CNG 加密程序扩展可用于 Office 2010 或 2007 Office system SP2：

AES、DES、DESX、3DES、3DES_112 和 RC2

以下 CNG 哈希算法或系统上安装的任何其他 CNG 加密程序扩展可用于 Office 2010 或 2007 Office system SP2：

MD2、MD4、MD5、RIPEMD-128、RIPEMD-160、SHA-1、SHA256、SHA384 和 SHA512

尽管在加密 Open XML 格式文件（.docx, .xslx, .pptx 等）时可使用 Office 2010 设置去更改执行加密的方式，默认值 — AES（高级加密标准）、128 位密钥长度、SHA1 和 CBC（加密块链接）— 提供了强加密，应该适用于大多数组织。AES 加密是可用的业界最强标准算法，并由美国国家安全局 (NSA) 选择用作美国政府的标准。Windows XP SP2、Windows Vista、Windows 7、Windows Server 2003 和 Windows Server 2008 均支持 AES 加密。

2.2.2 系统设计中涉及的加密算法

在 office2010 版本中：docx 格式的文件，采用 AES 加密算法，密钥长度强制为 128 位。这种加密算法下，共有 2 的 128 次方个密钥，以现代家用计算机的算力，如果要尝试完这些密钥组合，要从地球诞生跑到地球毁灭。如果没有好的密码字典以及社会工程学的配合，破解几乎不可能。

所以此次 project 需要解出来的密码比较简单，由 3 个数字组成，否则不可能实现解密。

2.2.3 加密算法详解

1. Word 加密使用到的字段：

Salt：随机生成 16byte 的数据（Base64 编码存储）

VerifierHashInput：随机生成 16byte 的数据(AES128 加密后 Base64 存储存储)

VerifierHashValue：用户密码加盐经过多次 SHA1 计算得到的哈希值(AES128 加密后 Base64 存储)

2. 加密过程

>>用户输入密码

>>随机生成 16 字节的 salt，与用户输入的密码一起通过哈希算法 SHA1 得到摘要值 H1，H1 分别与两个块密钥 BlockKey1/2 生成两把密钥，用于 AES 加密，这保证了同样的密码会生成不同的密钥。

>>AES128 算法使用密钥开始加密。

2.3 SHA1 算法简介

2.3.1 简介

SHA1 是一种安全哈希算法。对于长度小于 2^64 位的消息，SHA1 会产生一个 160 位的消息摘要。它对消息进行位处理。

在已提供的 hash_fun.c 中 SHA1 是对 u32 (unsigned int)类型的数据进行处理，因此所有的消息（char 类型等等）都需要进行位运算转成 u32 进行处理。

由于已经提供了图中 F 部分（逻辑运算）的代码，所以在本程序中最重要的是自己按照 SHA1 的分组规则，对消息进行分组、扩展、补位设计。

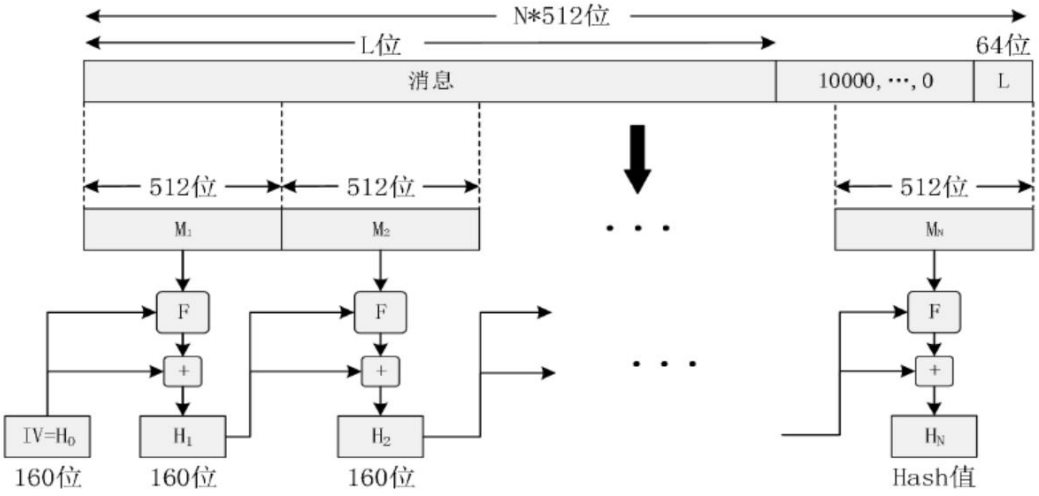


图 2-1 SHA1 生成消息摘要

2.4 AES128 算法简介

AES128 是一种对称的分组加密技术，使用 128 位分组加密数据，由四个不同的变换组成，包括一个置换和三个替代：

字节代替(SubBytes)： 用一个 S 盒完成分组的字节到字节的代替。

行移位(ShiftRows):一个简单的置换。

列混淆(MixColumns)：利用域 GF(28)上的算术特性的一个代替。

轮密钥加(AddRoundKey):当前分组和扩展密钥的一部分进行按位 XOR(异或)。

明文分组的长度为 128 位即 16 字节，密钥长度可以为 16 字节。在已提供的代码的基础上，需要利用 salt、password、BlockKey 等消息生成密钥 rkey（128Bit），在进行密钥扩展得到最终解密/加密所需的 key（1408Bit）。

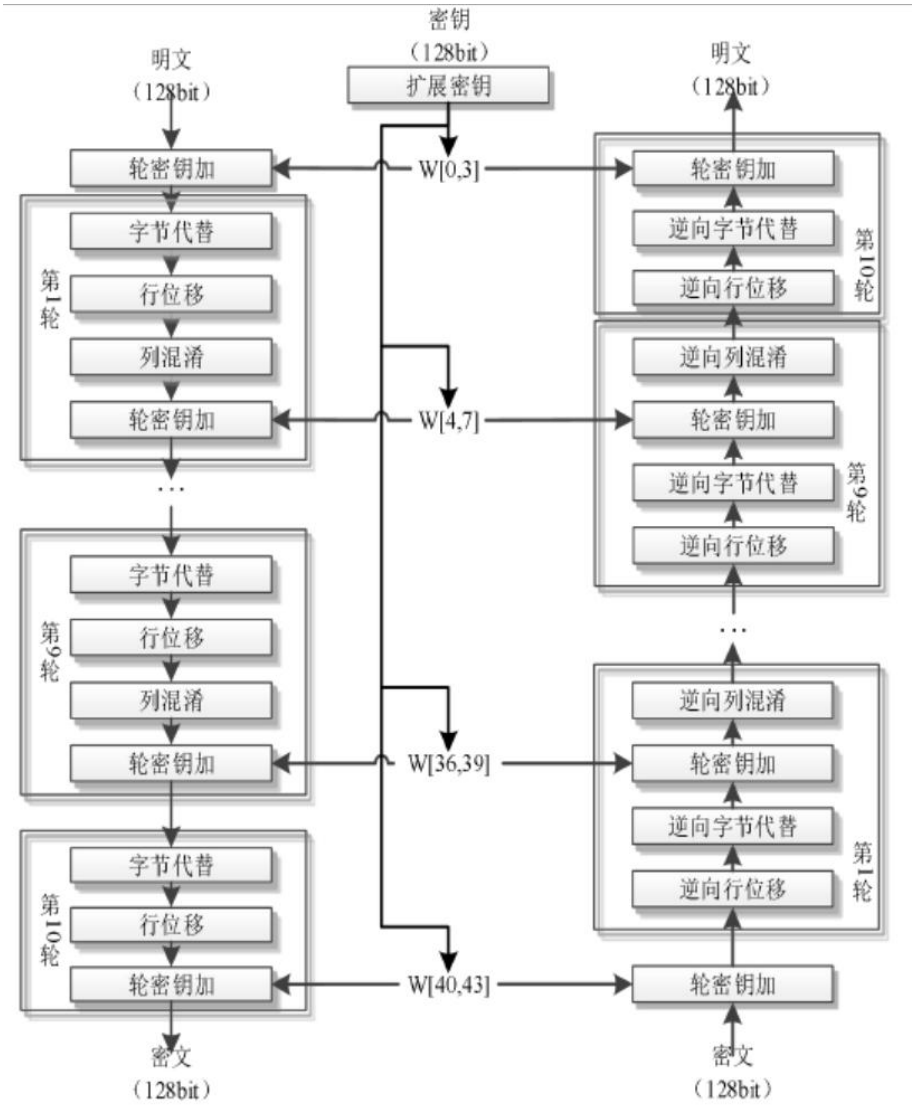


图 2-2 AES128 加密和解密

三、系统设计

3.1 总体设计思路

3.1.1 解密思路

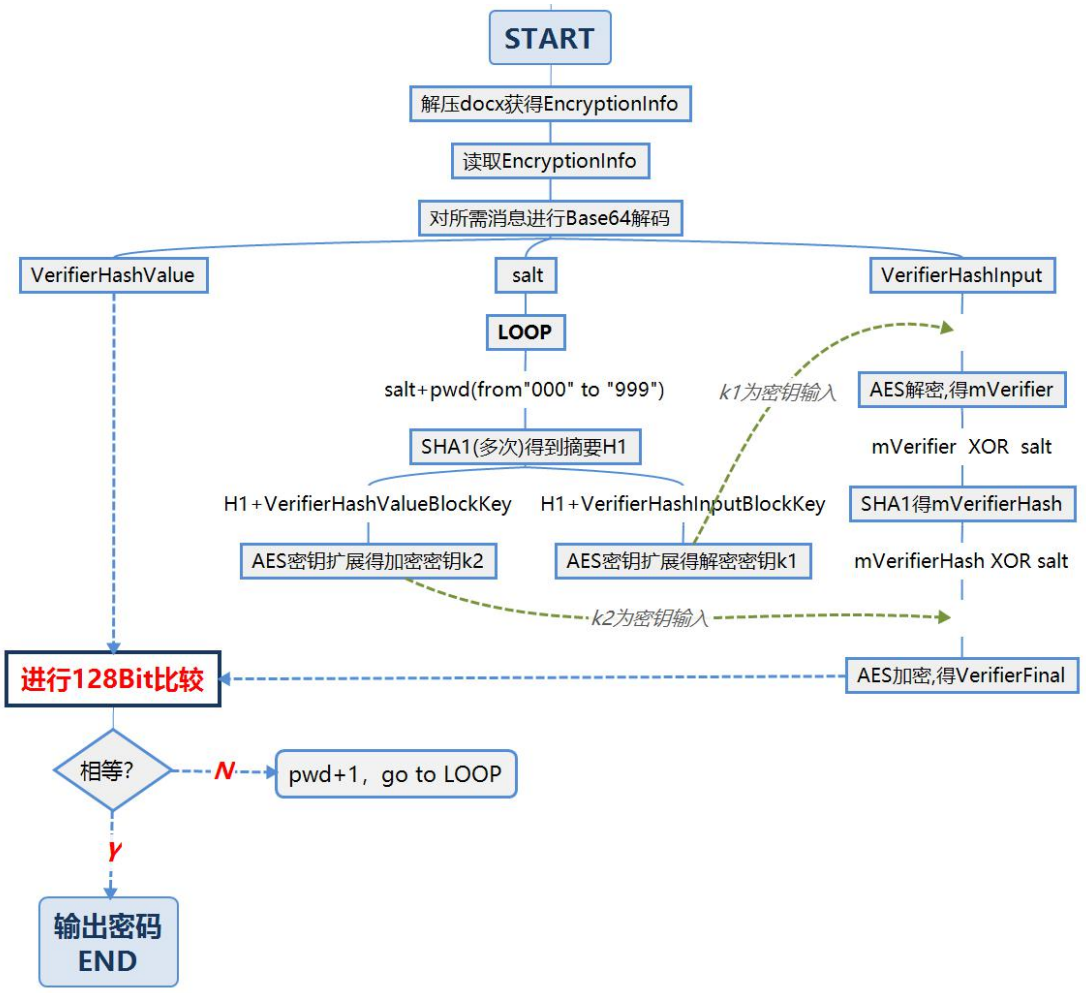


图 3.1 系统流程图

3.1.2 具体实现

系统程序主要分为五个子模块：文件读取、Base64 解码、SHA1 计算哈希值、AES 加解密、结果比较。

工程文件：DecryptDocx.dev，其中包含文件如下：

主程序：main.c

文件读取：**void ReadInfo(int start)** （在 main.c 中）
Base64 解码：Base64_decode.h 和 Base64_decode.c
SHA1：hash_func.c 、hash_func.h
AES 加解密：AES128.h

3.2 子模块的功能实现

3.2.1 文件读取

函数声明：**void ReadInfo(int start);**
参数：start=1（默认）
功能：
>>用户输入文件名：EncryptionInfo

```
please input the filename:  EncryptionInfo

readline 1: 」
readline 2: version="1.0"
readline 3: encoding="UTF-8"
readline 4: standalone="yes"?>
readline 5: <encryption
readline 6: xmlns="http://schemas.microsoft.com/office/2006/encryption"
readline 7: xmlns:p="http://schemas.microsoft.com/office/2006/keyEncryptor/password"><keyData
readline 8: saltSize="16"
readline 9: blockSize="16"
readline 10: keyBits="128"
readline 11: hashSize="20"
readline 12: cipherAlgorithm="AES"
readline 13: cipherChaining="ChainingModeCBC"
readline 14: hashAlgorithm="SHA1"
readline 15: saltValue="A5X18dBPMHuCfgMj1psKrQ=="/><dataIntegrity
readline 16: encryptedHmacKey="VWK8q/7VcFrazz+VSPrNSjFnI3cW11JrICoWhPsQDmA="
readline 17: encryptedHmacValue="aWnABAmOMRTALiSUaG8J9EPtwCW7y2TCwHSRSxU7Ns8="/><keyEncryptors><keyEncryptor
readline 18: uri="http://schemas.microsoft.com/office/2006/keyEncryptor/password"><p:encryptedKey
readline 19: spinCount="100000"
readline 20: saltSize="16"
readline 21: blockSize="16"
readline 22: keyBits="128"
readline 23: hashSize="20"
readline 24: cipherAlgorithm="AES"
readline 25: cipherChaining="ChainingModeCBC"
readline 26: hashAlgorithm="SHA1"
readline 27: saltValue="yXOrA3RFqF3Pa2gq97Cb8g=="
readline 28: encryptedVerifierHashInput="cvlZ7FEpIE2HXTKQ7eE3zg=="
readline 29: encryptedVerifierHashValue="gtwIzAujRYyy27il7DREHKiUWxx5BjSubwdZB7cthD8="
readline 30: encryptedKeyValue="Gt65fGzdNivmzclHlHzzDw=="/></keyEncryptor></keyEncryptors></encryption>
```

>>在屏幕上输出文档内容
>>在 readline 27-29 即为所需信息，用户可以复制使用

3.2.2 Base64 解码

函数声明：**int base64_decode(const char * base64, unsigned char * bindata);**
输入：**const char * base64** 以 base64 编码的字符串
输出：**unsigned char * bindata** 解码后的字符串
功能：Base64 解码

由于程序均在 u32（unsigned int）类型上操作，因此需要使用移位运算把 char 类型转为 u32 类型。

3.2.3 SHA1 计算哈希值

函数声明：**int** sha1_transform(**const u32** w0[4], **const u32** w1[4], **const u32** w2[4], **const u32** w3[4], **u32** digest[5]);

输入：四个 **const u32** 为分组扩展等处理后的消息

输出：**u32** digest[5]　运算后的哈希值

功能：哈希值运算

1. 设计中最主要的是对 sha1_transform 输入消息的分组，由于设计中的消息长度不超过 512bit，所以可以对函数中某些不变的亮进行赋值后，再也不改变，可以减少程序的运行时间。

参数分组图如下所示：（注：消息占用 w0 和 w1，要进行消息后面的 10000...补位）

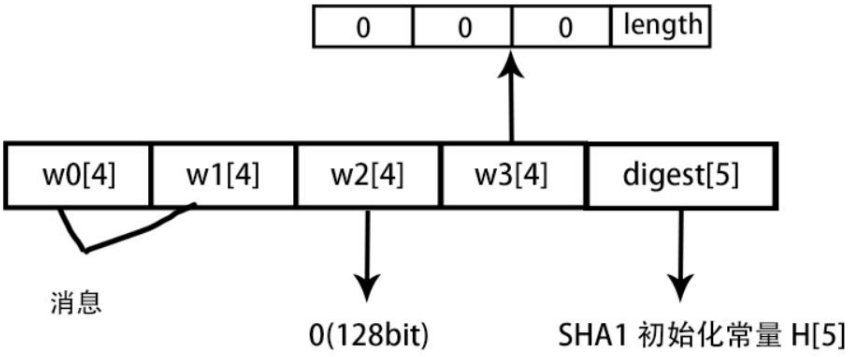


图 3.2 参数分组图示

2. 在进行 10,0000 次 SHA1 迭代的时候，每一次的消息是当前循环次数(u32 loop)的字节高低位转换后，在拼接前一次的输出，作为下一次迭代的输入。

代码实现如下：

```
// 10,0000 循环
    for(m=0;m<100000;m++){
        loop=m;
    //高低字节交换
        a=(loop>>24)&0x000000ff;
        b=(loop>>8)&0x0000ff00;
        c=(loop<<8)&0x00ff0000;
        d=(loop<<24)&0xff000000;
        loop=a|b|c|d;
        w0[0]=loop; w0[1]=digest[0];
        w0[2]=digest[1];  w0[3]=digest[2];
        w1[0]=digest[3];  w1[1]=digest[4];
        memcpy(digest,H,20);  //  初始化 digest
```

```
        sha1_transform (w0,w1,w2,w3,digest);
    }
```

3.2.4 AES 加解密

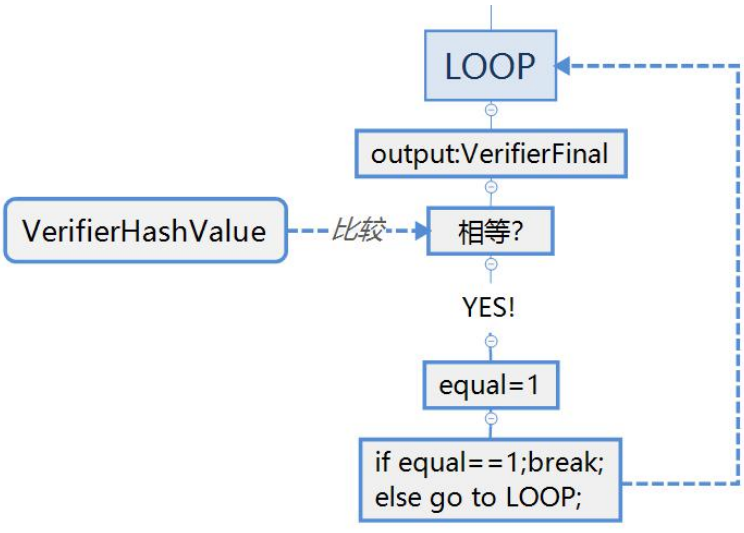
函数声明：

```
    void AES128_ExpandKey (u32 *userkey, u32 *rek, u32 *s_te0, u32
*s_te1, u32 *s_te2, u32 *s_te3, u32 *s_te4);
    void AES128_InvertKey (u32 *rdk, u32 *s_td0, u32 *s_td1, u32
*s_td2, u32 *s_td3, u32 *s_td4, u32 *s_te0, u32 *s_te1, u32 *s_te2,
u32 *s_te3, u32 *s_te4);
    void AES128_decrypt (const u32 *in, u32 *out, const u32 *rdk,
u32 *s_td0, u32 *s_td1, u32 *s_td2, u32 *s_td3, u32 *s_td4);
    void AES128_encrypt (const u32 *in, u32 *out, const u32 *rek,
u32 *s_te0, u32 *s_te1, u32 *s_te2, u32 *s_te3, u32 *s_te4);
```

功能：对 128bit 密钥 userkey 进行扩展，变成 1408bit 的扩展密钥 rek/rdk；
利用扩展密钥对消息进行加解密运算。

3.2.5 结果比较

定义参数 equal ，用于比较输出结果。



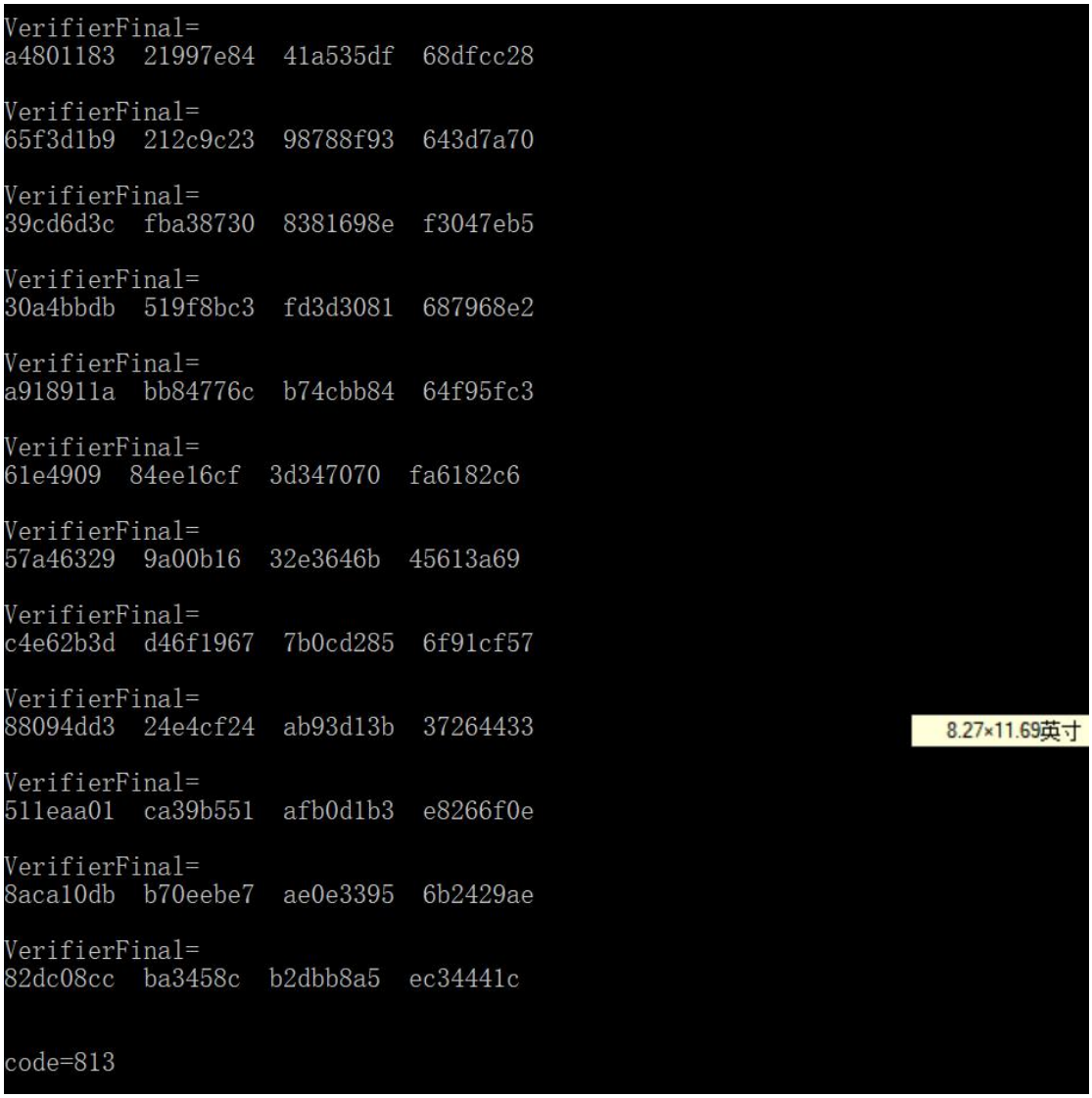
四、解密测试

4.1 Base64 解码测试

```
readline 27: saltValue="yXOrA3RFqF3Pa2gq97Cb8g=="
readline 28: encryptedVerifierHashInput="cvlZ7FEpIE2HXTKQ7eE3zg=="
readline 29: encryptedVerifierHashValue="gtwIzAujRYyy27il7DREHKiUWxx5BjSubwdZB7cthD8="
readline 30: encryptedKeyValue="Gt65fGzdNivmzclHlHzzDw==" /></keyEncryptor></keyEncryptors></encryption>


please input the saltValue:
yXOrA3RFqF3Pa2gq97Cb8g==

please input the encryptedVerifierHashInput:
cvlZ7FEpIE2HXTKQ7eE3zg==

please input the encryptedVerifierHashValue:
gtwIzAujRYyy27il7DREHKiUWxx5BjSubwdZB7cthD8=


salt=
c97d2b03  7445a85d  cf6b682a  f7b09bf2

VerInput=
72f959ec  5129204d  875d3290  ede137ce

VerValue=
82dc08cc  ba3458c  b2dbb8a5  ec34441c  a8945b1c  790634ae  6f075907  b72d843f
```

4.2 解密测试

测试 1：
测试条件：password 从 800-820 循环
输出：每次计算后的 VerifierFinal
结果：遇到正确的密码 813 后，循环 LOOP 结束，最后一个输出的 VerifierFinal 即为正确值。



测试 2：

对正确密码运算得到的中间值测试：

```
salt=
c97d2b03  7445a85d  cf6b682a  f7b09bf2

VerInput=
72f959ec  5129204d  875d3290  ede137ce

VerValue=
82dc08cc  ba3458c  b2dbb8a5  ec34441c  a8945b1c  790634ae  6f075907  b72d843f

H0=
257f2e91  49387105  61bef601  9113cf99  f5c8d7cb

H1=
3080ac5e  f0ade9bc  b69b9bc6  6dec37f0  2355eac3

rkey1=
74d67f81  f7a9c58d  3713db1c  f257830b

rkey2=
77f4ff30  cc5d344  c25e0551  39c9585f

k1=
318f1ad6  767f0841  398097ed  5e1db169  ff3a8d7f  c411f512  ae9f63d1  393ac893
27a50557  3b2b786d  6a8e96c3  97a5ab42  d2ceed4f  1c8e7d3a  51a5eeae  fd2b3d81
eb9a957e  ce409075  4d2b9394  ac8ed32f  f06d8363  25da050b  836b03e1  e1a540bb
8550b93e  d5b78668  a6b106ea  62ce435a  c54ff3b3  50e73f56  73068082  c47f45b0
7bffd618  95a8cce5  23e1bfd4  b779c532  cb1021b2  ee571afd  b6497331  94987ae6
74d67f81  f7a9c58d  3713db1c  f257830b

k2=
77f4ff30  cc5d344  c25e0551  39c9585f  ab9e3022  a75be366  6505e637  5cccbe68  e
2307568  456b960e  206e7039  7ca2ce51  dcbba478  99d03276  b9be424f  c51c8c1e  4
8dfd6de  d10fe4a8  68b1a6e7  adad2af9  cd3a4f4b  1c35abe3  74840d04  d92927fd  4
8f61b7e  54c3b09d  2047bd99  f96e9a64  974e58e7  c38de87a  e3ca55e3  1aa4cf87  5
ec44f45  9d49a73f  7e83f2dc  64273d5b  89e37606  14aad139  6a2923e5  e0e1ebe  14
91d8ad  3b0994  6a122a71  641c34cf
```

测试 3：最终测试
password 从 000-999 循环

```
code=813

--------------------------------
Process exited after 56.67 seconds with return value 10
请按任意键继续. . .
```

运行时间 56.67 seconds.
输出密码 code=813

五、性能优化

本程序循环次数很多，因此主要优化是对在一些固定不变的参量赋值的时候，减少在 for 循环里面的重复赋值，降低计算量。

六、心得与体会

1. 论文中有一些算法的细节描述并不详细，因此需要自己找到其中隐含的细节点，大量搜索其他资料，才能实现解密算法。
2. 这次 Project 加深了我对大小端编码的理解：
（1）在 SHA1 进行 10,0000 迭代的时候对当前循环次数 loop 的高低字节转换；
（2）是否了解 password 是用 utf-16 的小端编码存储，一个 password 是 16bit；这也是能否完成这次设计的关键之处；
（3）memcpy 是把一个指针指向的内存拷贝指定字节到另一个指针指向的内存，在把 char 转 u32 的时候，我本来想用 memcpy，但是由于 windows（x64）的小端编码，把 char[]数组拷贝 4 个字节到 u32(同样 4 个字节)的时候，与我们日常想象的顺序不一样，转换之后的 u32 无法进行正确的 sha1 和 AES128 运算，所以最后还是用移位运算来实现从 char 到 u32 的转换。

七、参考文献

[1]李玲双，加密文档的破解与实现，哈尔滨工业大学，2017.6

八、附录

8.1附件

1. DecryptDocx.dev　　//C 工程文件（在 code 文件夹中）
2. Readme.txt　　//使用说明
3. EncryptionInfo　　//拥有解密所需信息的文件　（在 code 文件夹中）

8.2源代码

```
/*
* Copyright(C),2018 ZJU
* FileName:  DecryptDocx.c
* Author:    WuJiangNan
* Version:  1st
* Date:  2018/11/13
*
*Descriptions:
    程序对 Microsoft office 2010 版本的 docx 格式的文档进行解密
```

```
    密码由3个数字组成，from “000” to “999”.

    Input   : 文件名EncryptionInfo
    Output  : EncryptionInfo里的所有信息

    Input   : EncryptionInfo里面解密所需信息
              ——saltValue(the 2nd one)
              ——encryptedVerifierHashInput
              ——encryptedVerifierHashValue
    Output  : Base64解码后的信息(16进制)

    Return  : 解密后的3位密码

*/


#include <memory.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h>
#include "hash_func.h"
#include "AES128.h"
#include "Base64_decode.h"

typedef unsigned int u32;

void ReadInfo();
void AES128_ExpandKey (u32 *userkey, u32 *rek, u32 *s_te0, u32
*s_te1, u32 *s_te2, u32 *s_te3, u32 *s_te4);
void AES128_InvertKey (u32 *rdk, u32 *s_td0, u32 *s_td1, u32 *s_td2,
u32 *s_td3, u32 *s_td4, u32 *s_te0, u32 *s_te1, u32 *s_te2, u32
*s_te3, u32 *s_te4);
void AES128_decrypt (const u32 *in, u32 *out, const u32 *rdk, u32
*s_td0, u32 *s_td1, u32 *s_td2, u32 *s_td3, u32 *s_td4);
void AES128_encrypt (const u32 *in, u32 *out, const u32 *rek, u32
*s_te0, u32 *s_te1, u32 *s_te2, u32 *s_te3, u32 *s_te4);

void main(){

    u32 i,j,k,m;
    unsigned char salt_base64[30],salt_char[20];
    unsigned char VerInput_base64[60],VerInput_char[40];
    unsigned char VerValue_base64[60],VerValue_char[40];
```

```
    u32 salt[4],VerInput[4],VerValue[8];


/*        提取 EncryptionInfo 的信息并输出        */
    ReadInfo(1);    //用户输入 EncryptionInfo


/*        用户输出解密所需信息      */
    getchar();
    printf("\n\nplease input the saltValue:  \n");
    gets(salt_base64);
    printf("\nplease input the encryptedVerifierHashInput:  \n");
    gets(VerInput_base64);
    printf("\nplease input the encryptedVerifierHashValue:  \n");
    gets(VerValue_base64);
    puts("\n");

/*        对所需 info 进行 base64 解码        */

    // 解码得到 char 类型 salt_char、VerInput_char...
    base64_decode( salt_base64, salt_char ); //saltValue 进行
salt_base64 解码
    base64_decode( VerInput_base64, VerInput_char );
    base64_decode( VerValue_base64, VerValue_char );


    //char 类型转 u32
     i=0;
    for(k=0;k<4;k++){
    salt[k]=(salt_char[i]<<24)+(salt_char[i+1]<<16)+(salt_char[
i+2]<<8)+(salt_char[i+3]);

    VerInput[k]=(VerInput_char[i]<<24)+(VerInput_char[i+1]<<16)
              +(VerInput_char[i+2]<<8)+(VerInput_char[i+3]);
    i+=4;
    }

    i=0;
    for(k=0;k<8;k++){
    VerValue[k]=(VerValue_char[i]<<24)+(VerValue_char[i+1]<<16)
             +(VerValue_char[i+2]<<8)+(VerValue_char[i+3]);
    i+=4;
    }
```

```
    //解码后的信息 输出
    puts("salt=");
    for(j=0;j<4;j++){
    printf("%x  ",salt[j]);
    }
    puts("\n");

    puts("VerInput=");
    for(j=0;j<4;j++){
    printf("%x  ",VerInput[j]);
    }
    puts("\n");

    puts("VerValue=");
    for(j=0;j<8;j++){
    printf("%x  ",VerValue[j]);
    }
    puts("\n");

    // 参数定义
    u32 pwd;          //密码 password
    u32 mid,left,right;       //密码的左中右 数字
    int equal;          // 最终比较Verifier 相等与否

    u32 w0[4],w1[4],w2[4],w3[4],digest[5];
    u32 H0[5],H1[5];                // 摘要
    const u32 H[5]={SHA1M_A,SHA1M_B,SHA1M_C,SHA1M_D,SHA1M_E};
//SHA1 算法的初始化常量


    // 10,0000 次SHA1 迭代 参数定义
    u32 loop;    //当前迭代次数
    u32 a,b,c,d;

    // AES 密钥扩展 参数定义
    u32 rkey1[4],rkey2[4];
    u32 k1[44],k2[44];     // 4*4*11 Byte

    // AES 加解密 参数定义
    u32
mVerifier[4],mVerifier_2[4],mVerifierHash[4],VerifierFinal[4];
    u32 out[4]={0,0,0,0};    //初始化
    u32 in[4];
```

```
    u32 rdk[4],rek[4];

/***————————pwd  000-999 循环————————***/
    for(pwd=800;pwd<816;pwd++){

/*        salt + pwd 拼接, SHA1 扩展分组          */

        right=pwd%10+48;
        mid=(pwd/10)%10+48;
        left= pwd/100+48;       // pwd 为utf-16 编码

        memcpy(w0,salt,16);
        w1[0]=(left<<24)+(mid<<8);
        w1[1]=(right<<24)+(1<<15);
        w1[2]=w1[3]=0;
        memset(w2,0,16);
        memset(w3,0,16);
        w3[3]=176;          //SHA1 补位规则: 最后补上消息长度
        memcpy(digest,H,20);          //初始化 digest

/*       一轮 SHA1 计算得到摘要H0       */
        sha1_transform (w0,w1,w2,w3,digest);
        memcpy(H0,digest,20);
    /*
    // 输出 H0
        puts("H0=");
        for(j=0;j<5;j++){
        printf("%x  ",H0[j]);
        }
        puts("\n");*/

/*       H0 进行 10 万次 SHA1 迭代  得 H1        */

        //对一些不变参量的赋值
        memset(w1,0,16);
        memset(w2,0,16);
        memset(w3,0,16);
        w1[2]=1<<31;
        w3[3]=192;

        // 10,0000 循环
        for(m=0;m<100000;m++){
            loop=m;
        //高低字节交换
```

```
            a=(loop>>24)&0x000000ff;
            b=(loop>>8)&0x0000ff00;
            c=(loop<<8)&0x00ff0000;
            d=(loop<<24)&0xff000000;
            loop=a|b|c|d;
            w0[0]=loop; w0[1]=digest[0];  w0[2]=digest[1];
w0[3]=digest[2];
            w1[0]=digest[3];  w1[1]=digest[4];
            memcpy(digest,H,20);  //  初始化 digest

            sha1_transform (w0,w1,w2,w3,digest);
        }
        memcpy(H1,digest,20);     // 得到摘要 H1

        // 输出 H1
        /*
        puts("H1=");
        for(j=0;j<5;j++){
        printf("%x  ",H1[j]);
        }
        puts("\n");  */

/*       H1 与两个 BlockKey 生成 rkey1/rkey2     */

        // SHA1 得到 rkey1
        memcpy(w0,H1,16);
        w1[0]=H1[4];
        w1[1]=encryptedVerifierHashInputBlockKey[0];
        w1[2]=encryptedVerifierHashInputBlockKey[1];
        w1[3]=1<<31;
        memset(w2,0,16);
        memset(w3,0,16);
        w3[3]=224;     //消息长度 160+64=224bit
        memcpy(digest,H,20);
        sha1_transform (w0,w1,w2,w3,digest);
        memcpy(rkey1,digest,16);   //rkey1 取 SHA1 输出的前 128bit
    /*
        puts("rkey1=");
        for(j=0;j<4;j++){
        printf("%x  ",rkey1[j]);
        }
        puts("\n"); */
```

```
        // SHA1 得到rkey2
        w1[1]=encryptedVerifierHashValueBlockKey[0];
        w1[2]=encryptedVerifierHashValueBlockKey[1];
        memcpy(digest,H,20);
        sha1_transform (w0,w1,w2,w3,digest);
        memcpy(rkey2,digest,16);

    /*
        puts("rkey2=");
        for(j=0;j<4;j++){
        printf("%x  ",rkey2[j]);
        }
        puts("\n"); */

/*      rkey1/2  AES 密钥扩展  得扩展密钥k1,k2      */
        AES128_ExpandKey (rkey2,k2,te0,te1,te2,te3,te4);
        AES128_ExpandKey (rkey1,k1,te0,te1,te2,te3,te4);
        AES128_InvertKey
(k1,td0,td1,td2,td3,td4,te0,te1,te2,te3,te4);


/*      k1 对明文VerifierHashInput 进行AES 解密得mVerifier      */



        AES128_decrypt
(VerInput,mVerifier,k1,td0,td1,td2,td3,td4);
    /*
        puts("mVerifier=");
        for(j=0;j<4;j++){
        printf("%x  ",mVerifier[j]);
        }
        puts("\n");*/

/*      mVerifier SHA1 得 mVerifierHash     */
        for(j=0;j<4;j++)    mVerifier_2[j]=mVerifier[j]^salt[j];
        memcpy(w0,mVerifier_2,16);
        memset(w1,0,16);
        w1[0]=1<<31;
        memset(w2,0,16);
        memset(w3,0,16);
        w3[3]=128;      //消息长度128bit
        memcpy(digest,H,20);
        sha1_transform (w0,w1,w2,w3,digest);
```

```
        memcpy(mVerifierHash,digest,16);

        /*
        puts("mVerifierHash=");
        for(j=0;j<4;j++){
        printf("%x  ",mVerifierHash[j]);
        }
        puts("\n"); */

/*      k2 对 mVerifierHash 加密得最终 VerifierFinal  */
        for(j=0;j<4;j++)
    mVerifierHash[j]=mVerifierHash[j]^salt[j];

    AES128_encrypt(mVerifierHash,VerifierFinal,k2,te0,te1,te2,t
e3,te4);

        puts("VerifierFinal=");
        for(j=0;j<4;j++){
        printf("%x  ",VerifierFinal[j]);
        }
        puts("\n");


/*      判断两个 Verifier 是否相等    */
        equal=1;          // 判断标志
        for(j=0;j<4;j++){
           if(VerifierFinal[j]==VerValue[j]) ;
           else {
              equal=0;break;
           }
        }
        if(equal==1) {
        printf("\ncode=%d%d%d\n",pwd/100,(pwd/10)%10,pwd%10);
//相等则输出
        break;
        }

    }

}

/*      读取文件 输出文件内容        */
 void ReadInfo(int start){
```

```
    char* buffer;
    long lSize;
    FILE *fp;
    char filename[50];
    int q;
    char saltValue[50],HashInput[50],HashValue[50],tem[100];

    printf("please input the filename:  ");
    scanf("%s",filename);
    printf("\n");

    fp = fopen(filename,"r");
    if (fp==NULL)
    {
      printf("reading error\n");
      exit (1);
    }

    // 将光标停在文件的末尾
    fseek (fp , 0 , SEEK_END);
    //返回文件的大小（单位是 bytes）
    lSize = ftell (fp);
    //将光标重新移回文件的开头
    rewind (fp);
    //将文件的内容读取到 buffer 中
    buffer = (char*)malloc(lSize);
    rewind(fp);
    fread (buffer,1,lSize,fp);
    rewind(fp);
    int i=1;
    while(!feof(fp)){
       fscanf(fp,"%s",buffer);
       printf("readline %d: %s\n",q+1,buffer);
    q++;
    }
    fclose (fp);
}
```