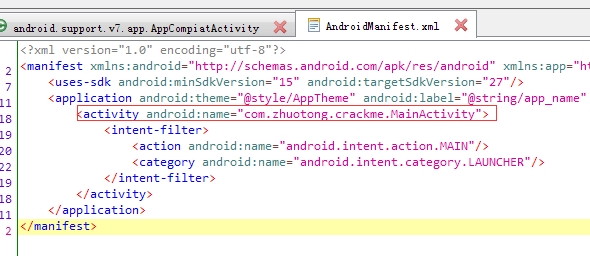
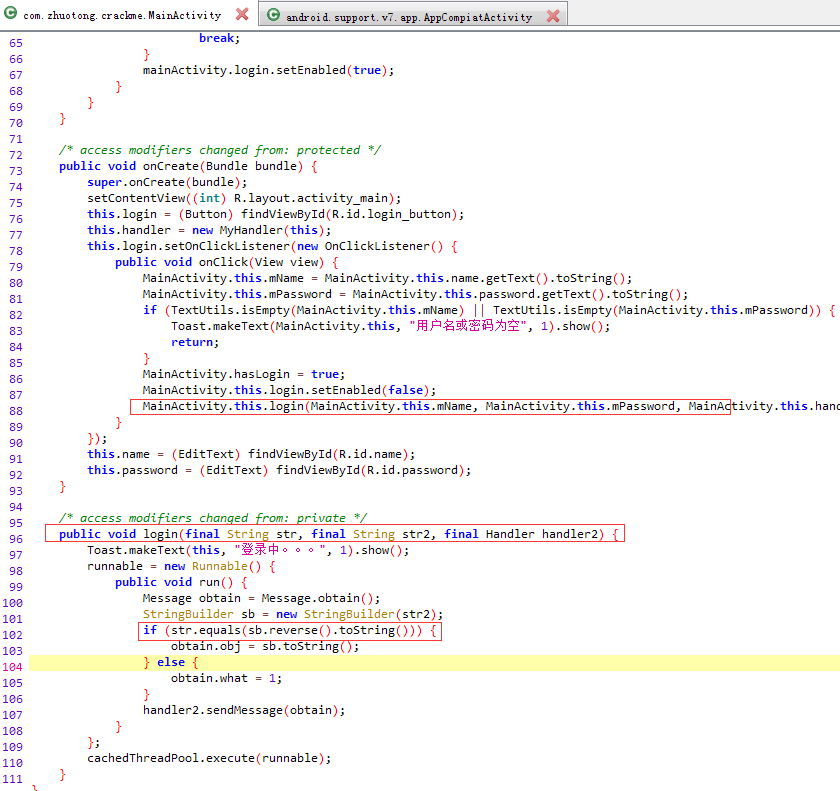
只有一个Activity页面



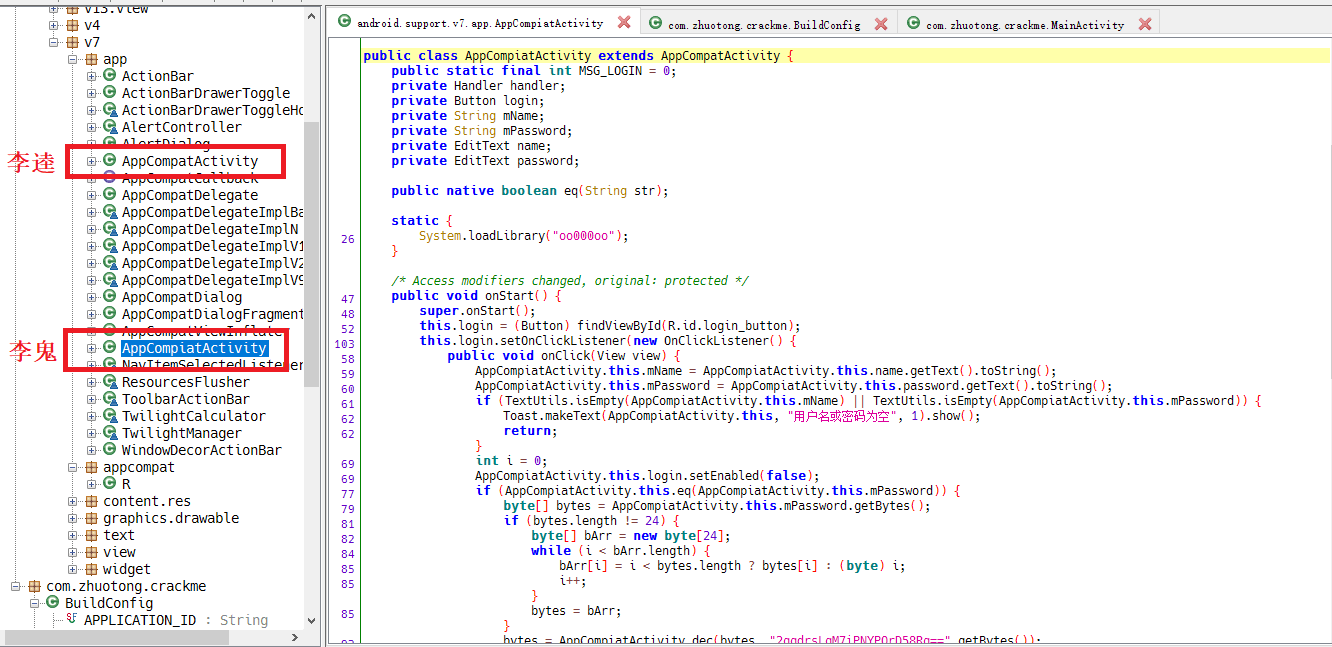
看起来只要把用户名跟密码 作为 颠倒的字符串就行

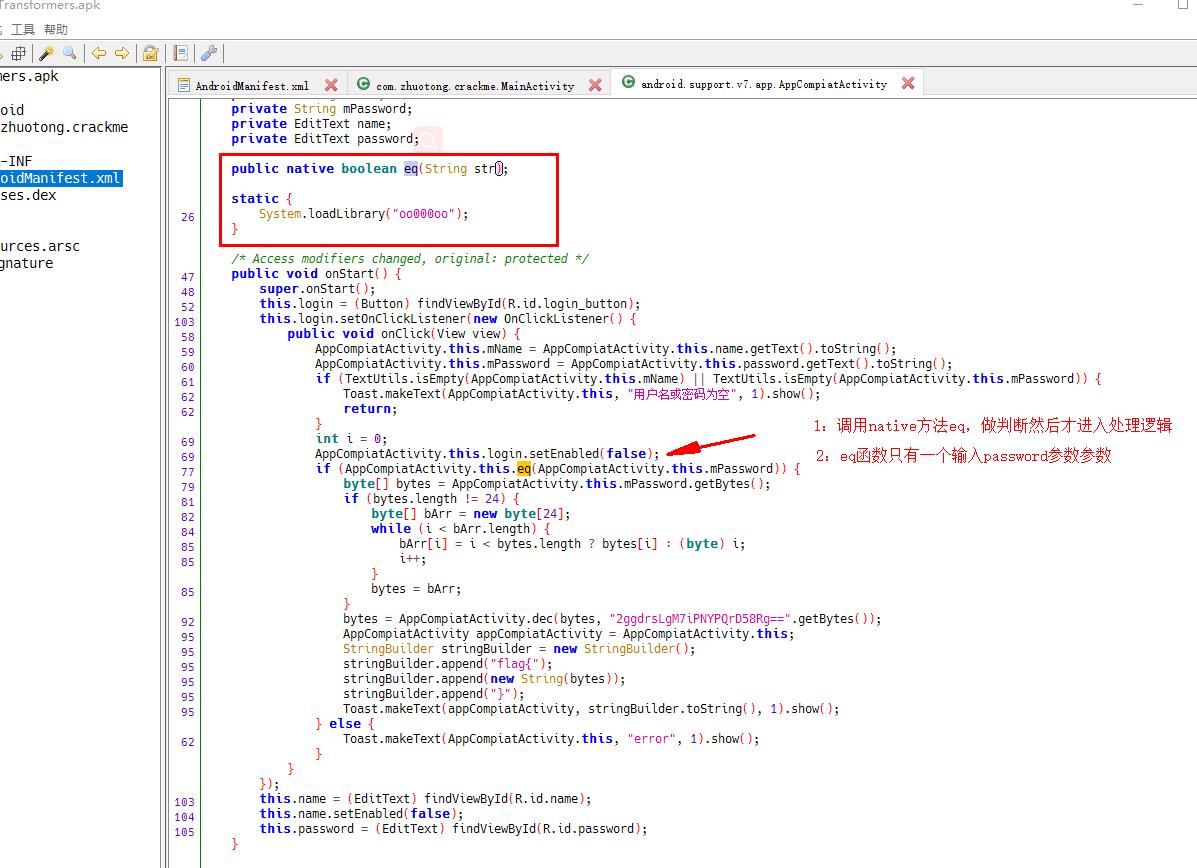
例如: name:123456，password:654321



然而并不是，再仔细观察就能发现 public class MainActivity extends AppCompiatActivity { 这行代码非常奇怪，正常情况下继承的应该是 AppCompatActivity 这个类，二者存在细微的差异，于是继续去看 AppCompiatActivity 的代码，发现**真实的点击效果是在OnStart中注册的OnClick中**

实现原理：MainActivity 继承 AppCompiatActivity，利用Activity的onStart事件会晚与OnCreate事件，从而实现在OnStart中注册的OnClick回调覆盖了OnCreate中注册的回调。



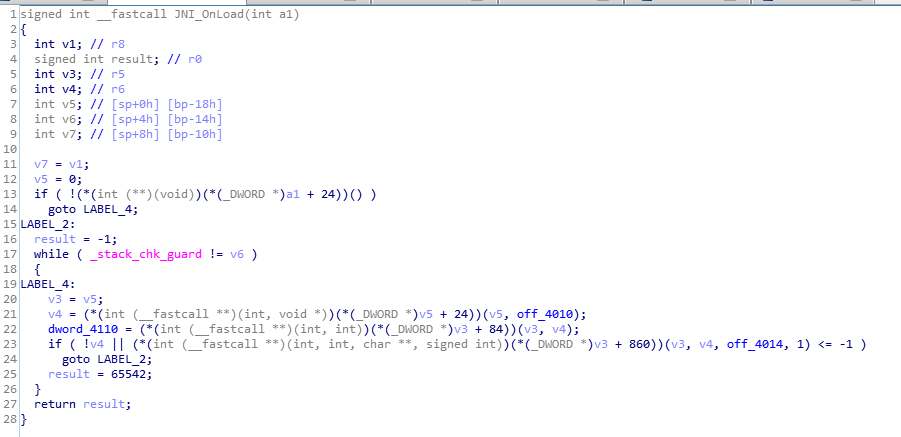


由此我们可以推断真正的校验逻辑在 AppCompaitActivity 这个类的 public native boolean eq(String str); 函数内，可以看到这个函数是一个 native 函数，那么具体的校验逻辑必然在 oo000oo 对应的 so 文件中

发现登陆时候使用了eq这个函数进行判断，eq改函数是前面的System.loadLibrary("oo000oo")中的，所以打开ida，对该so文件进行分析。

发现so文件中并没有eq该函数，所以判断在JNI\_LOAD或者在init\_array里面做了手脚。

JNI\_LOAD函数如下：



# JNI\_OnLoad

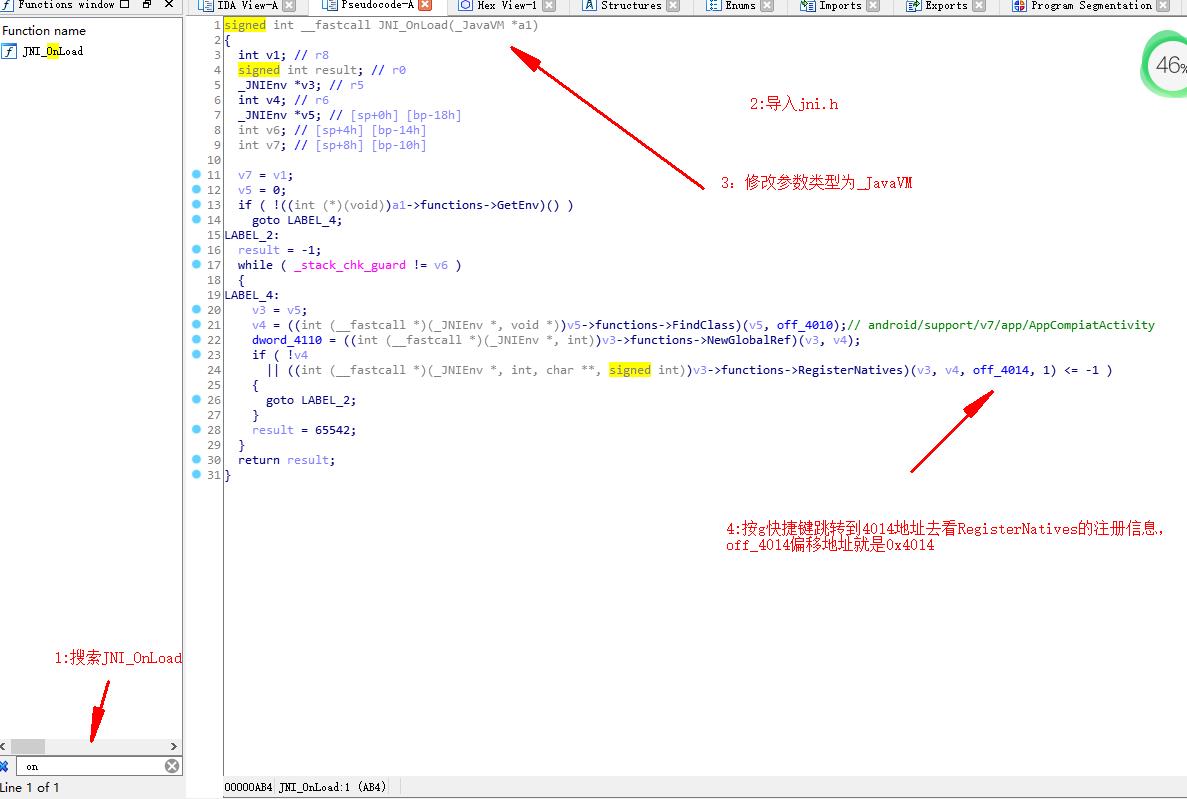
代码逻辑分析RegisterNatives

1：搜索JNI\_OnLoad，双击进入函数体

2：导入Jni.h文件。菜单路径：ida/File/Load file/Parse C header file

3：修改参数a1 类型为\_JavaVM

4：修改v3 类型 \_JNIEnv

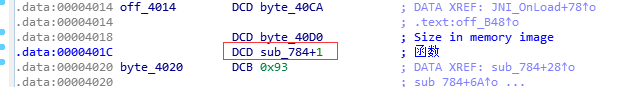


可以发现本题使用的是利用 jni 所提供的 RegisterNatives 方法来动态注册 Native 函数，很明显 off\_4014 上存储的就是需要注册的函数的地址，使用 ida 查看发现地址对应是 sub\_784() 函数。

按g快捷键直接跳转到0x4014

找到eq注册对应的sub\_784函数，这个位置存放的就是JNINativeMethod结构。

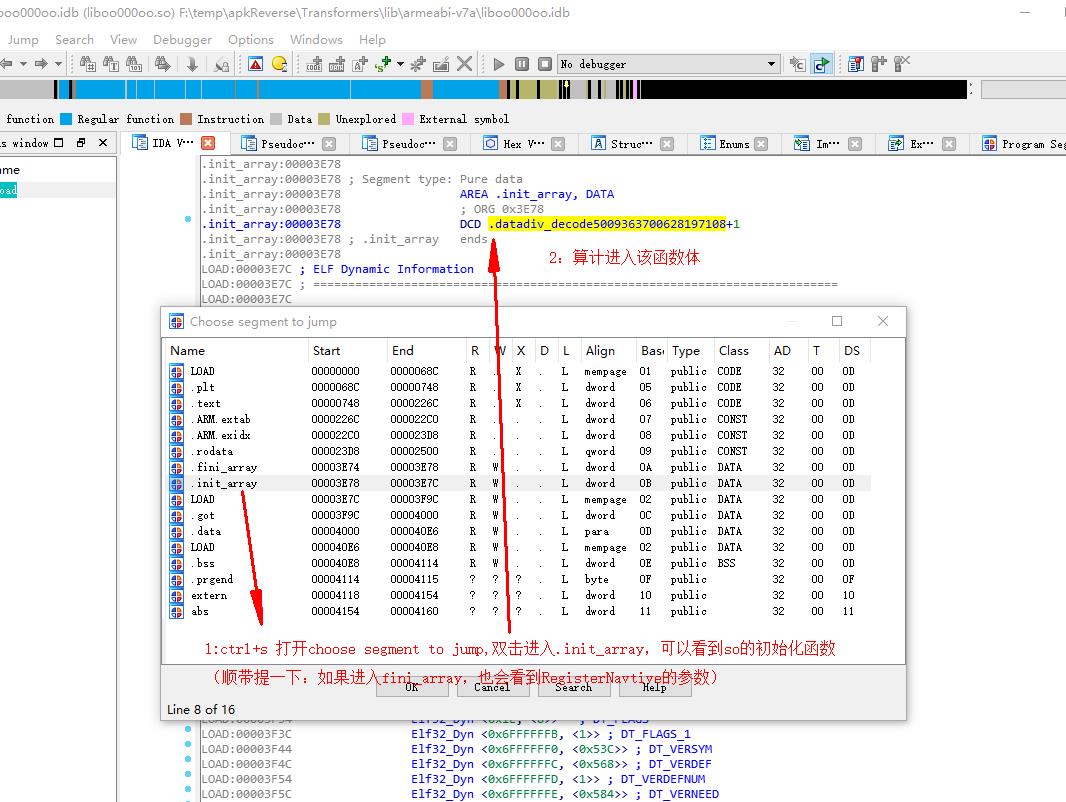
并且其中第三个变量sub\_784+1 指向的就是要注册的函数地址。



# 跟踪.init\_array（其实这步可以跳过）

另外根据so文件的加载流程应该是先加载init\_array，然后是JNI\_OnLoad

ida 中ctrl+s 进入init\_array，会先执行.datadiv\_decode5009363700628197108，其实就是通过一些运算得到一些初始化数据。



init\_array函数执行了datadiv\_decode5009363700628197108字符串解密函数

# IDA动态调试

adb install D:\Work\test\Crakeme04\Transformers.apk

adb push D:\Work\test\android\_server /data/local/tmp/

adb push D:\Work\test\android\_server64 /data/local/tmp/

adb push D:\tools\Mobile\mprop-master\armeabi-v7a\mprop /data/local/tmp/

查看任务栈

db shell dumpsys activity activities

adb shell dumpsys activity activities | findstr com.example.firecontrol

方法一、android9.0会失败

sailfish:/ $ su

sailfish:/ # cd /data/local/tmp/

sailfish:/data/local/tmp # ls

android\_server android\_server64 mprop

sailfish:/data/local/tmp # getprop ro.debuggable

0

sailfish:/data/local/tmp # chmod 777 mprop

sailfish:/data/local/tmp # ./mprop ro.debuggable 1

sscanf: Success

1|sailfish:/data/local/tmp # stop;start

方法二、感觉这个更好用一些

adb shell #adb进入命令行模式

su #切换至超级用户

magisk resetprop ro.debuggable 1

stop;start; #一定要通过该方式重启

cd /data/local/tmp/

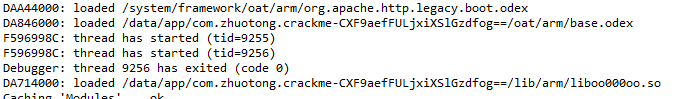
chmod 777 android\_server

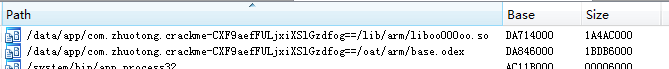
./android\_server

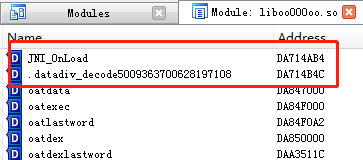
adb forward tcp:23946 tcp:23946

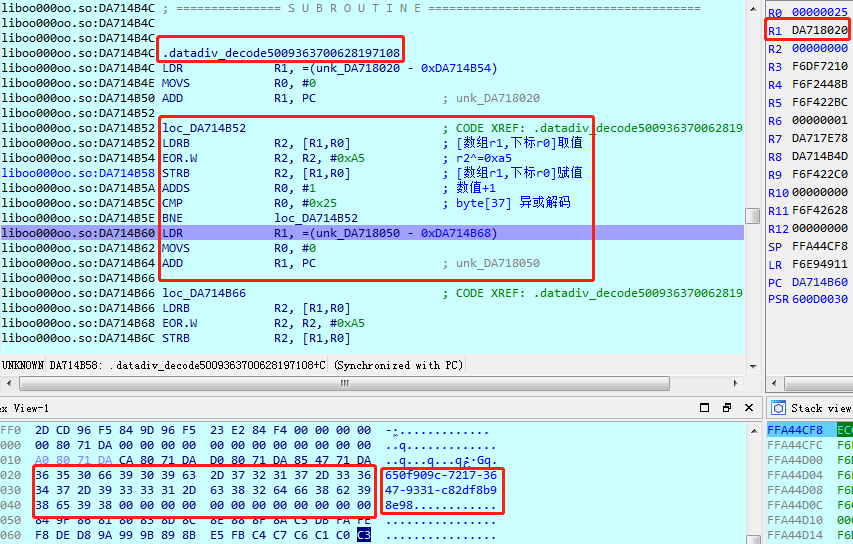
adb shell am start -D -n com.zhuotong.crackme/.MainActivity

jdb -connect com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=8700

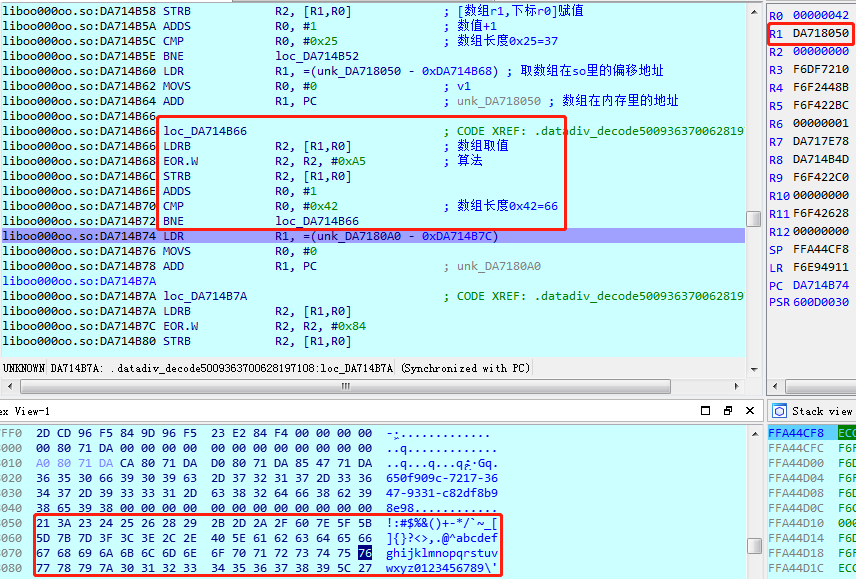




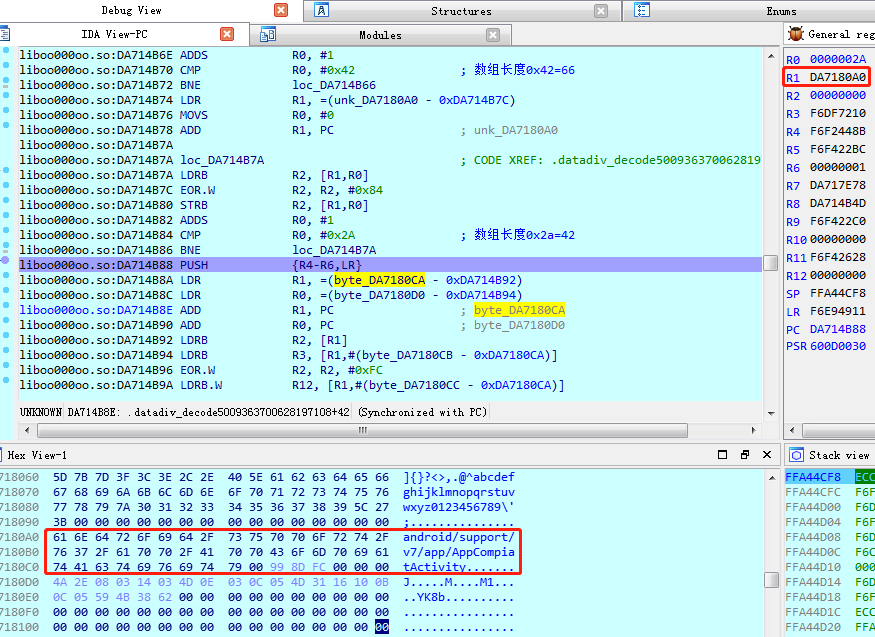




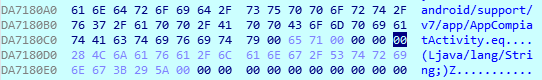
650f909c-7217-3647-9331-c82df8b98e98



!:#$%&()+-\*/`~\_[]{}?<>,.@^abcdefghijklmnopqrstuvwxyz0123456789\';



android/support/v7/app/AppCompiatActivity



Eq (Ljava/lang/String;)Z



650f909c-7217-3647-9331-c82df8b98e98

!:#$%&()+-\*/`~\_[]{}?<>,.@^abcdefghijklmnopqrstuvwxyz0123456789\';

android/support/v7/app/AppCompiatActivity

Eq (Ljava/lang/String;)Z

ini\_array部分==========

#解码36长度字符串byte\_4020：

byte\_4020 =650f909c-7217-3647-9331-c82df8b98e98+0x00（结束符）

#解码Base64为的64+1个编码字符byte\_4050=

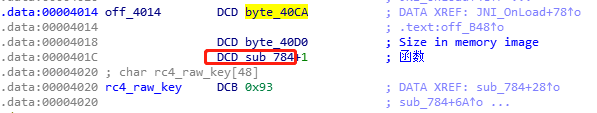
base64Chars = byte\_4050 =!:#$%&()+-\*/`~\_[]{}?<>,.@^abcdefghijklmnopqrstuvwxyz0123456789\';+0x00（结束符）

#app伪装java类的名称byte\_40A0：

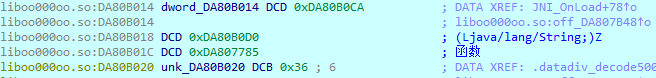
className\_Sign= android/support/v7/app/AppCompiatActivity +0x00（结束符）



动态注册

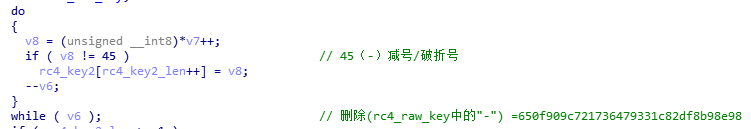


函数地址

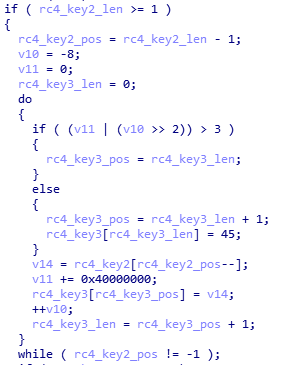


来到函数784

一、RC4密钥去掉-



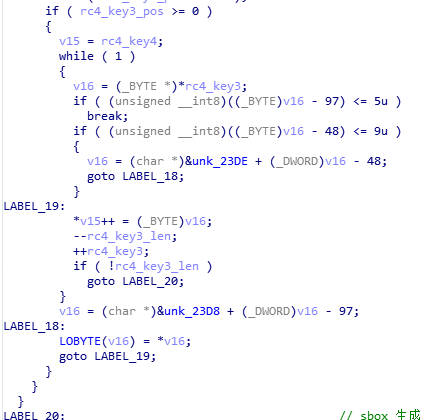
# RC4密钥去掉-  
# return 650f909c721736479331c82df8b98e98  
def getRc4Key2():  
 rc4key2 = ""  
 for i in range(len(rc4key)):  
 value = rc4key[i]  
 if ( ord(value) != 0x2d): # 字符串 转 ascii码  
 rc4key2 += value  
 return rc4key2



二、

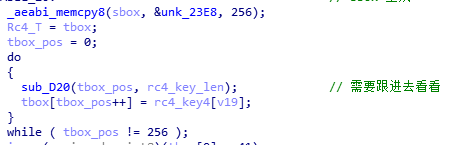
# return 89e89b8f-d28c-1339-7463-7127c909f056  
def getRc4Key3(key):  
 keylen = len(key)  
 rc4key2value1 = 0xFFFFFFF8  
 rc4key2value2 = 0x0  
 rc4key3 = ""  
  
 while keylen > 0:  
 temp = rc4key2value1 >> 2  
 temp2 = rc4key2value2 | temp  
 if temp2 <= 3:  
 rc4key3 += chr(0x2d)# ascii码 转 字符串  
  
 keylen -= 1  
 rc4key3 += key[keylen]  
  
 rc4key2value2 += 0x40000000  
 rc4key2value2 = fixedint.UInt32(rc4key2value2)  
 rc4key2value1 += 1  
 rc4key2value1 = fixedint.UInt32(rc4key2value1)  
  
 return rc4key3

三、



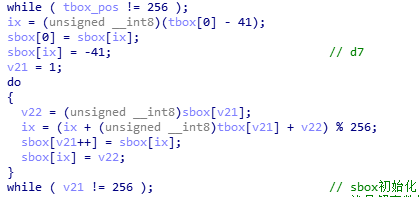
# 算法参数01：unk\_23DE(10位长度) =2409715836  
# 算法参数02：unk\_23D8（16位长度）=dbeafc2409715836  
# return 36f36b3c-a03e-4996-8759-8408e626c215  
def getRc4Key4(key):  
 keylen = len(key)  
 keypos = 0  
 retnkey = ""  
 unk\_23DE = "2409715836"  
 unk\_23D8 = "dbeafc2409715836"  
  
 while keylen > 0:  
 value = key[keypos]  
 valueascii = ord(value)  
 value97 = valueascii - 97  
 value48 = valueascii - 48  
 if (value97 <= 5) and (value97 >= 0):  
 value = unk\_23D8[value97]  
 elif (value48 <= 9) and (value48 >= 0):  
 value = unk\_23DE[value48]  
  
 retnkey += value  
 keylen -= 1  
 keypos += 1  
  
 return retnkey

四、根据密钥生成临时的数据key



# 根据密钥生成临时的数据key  
# 计算RC4 的临时256自己T向量，公式：iK[i]=(byte)aKey.charAt((i % aKey.length()));  
# retn =  
# 36f36b3c-a03e-4996-8759-8408e626c215  
# 36f36b3c-a03e-4996-8759-8408e626c215  
# 36f36b3c-a03e-4996-8759-8408e626c215  
# 36f36b3c-a03e-4996-8759-8408e626c215  
# 36f36b3c-a03e-4996-8759-8408e626c215  
# 36f36b3c-a03e-4996-8759-8408e626c215  
# 36f36b3c-a03e-4996-8759-8408e626c215  
# 36f3  
def getRc4Key5(key):  
 tbox = ""  
 pos = 0  
 keylen = len(key)  
  
 while pos < 256:  
 tbox += key[pos % keylen]  
 pos += 1  
  
 return tbox

五、sbox初始化



# 依据K将S-Box乱序  
# retn = sbox =  
# [0xf0,0x37,0xe1,0x9b,0x2a,0x15,0x17,0x9f,0xd7,0x58,0x4d,0x6e,0x33,0xa0,0x39,0xae,0x4,0xd0,  
# 0xbe,0xed,0xf8,0x66,0x5e,0x0,0xd6,0x91,0x2f,0xc3,0x10,0x4c,0xf7,0xa6,0xc1,0xec,0x6d,0xb,  
# 0x50,0x65,0xbb,0x34,0xfa,0xa4,0x2d,0x3b,0x23,0xa1,0x96,0xd5,0x1d,0x38,0x56,0xa,0x5d,0x4f,  
# 0xe4,0xcc,0x24,0xd,0x12,0x87,0x35,0x85,0x8e,0x6f,0xc6,0x13,0x9a,0xd3,0xfc,0xe7,0x8,0xac,  
# 0xb7,0xe9,0xb0,0xe8,0x41,0xaa,0x55,0x53,0xc2,0x42,0xbc,0xe6,0xf,0x8a,0x86,0xa8,0xcf,0x84,  
# 0xc5,0x48,0x74,0x36,0x7,0xeb,0x88,0x51,0xf6,0x7f,0x57,0x5,0x63,0x3e,0xfe,0xb8,0xc9,0xf5,  
# 0xaf,0xdf,0xea,0x82,0x44,0xf9,0xcd,0x6,0xba,0x30,0x47,0x40,0xde,0xfd,0x1c,0x7c,0x11,0x5c,  
# 0x2,0x31,0x2c,0x9c,0x5f,0x46,0x27,0xc4,0x83,0x73,0x16,0x90,0x20,0x76,0x7b,0xf2,0xe3,0xf3,  
# 0x77,0x52,0x80,0x25,0x9,0x26,0x3f,0xc7,0x18,0x1b,0xa3,0xff,0xfb,0xcb,0xa9,0x8c,0x54,0x7a,  
# 0x68,0xb4,0x70,0x4b,0xe2,0x49,0x22,0x7e,0xa5,0xb6,0x81,0x9d,0x4e,0x67,0xf1,0xa7,0x3c,0xd9,  
# 0x94,0xef,0x32,0x6b,0x1f,0xb1,0x60,0xb9,0x64,0x59,0x1,0xb3,0x7d,0xe0,0x6c,0xad,0x97,0x19,  
# 0xb5,0x3a,0xf4,0xd8,0x8d,0x98,0x3,0x93,0x1a,0xdc,0x1e,0x4a,0xc0,0x5a,0xe5,0xd1,0x3d,0x14,  
# 0xc8,0x79,0xbd,0x43,0xdb,0x69,0xd2,0x61,0x95,0x9e,0x21,0x45,0x89,0x2b,0xab,0x29,0xa2,0x8b,  
# 0x2e,0xd4,0xe,0x62,0xca,0x28,0xda,0x5b,0x72,0x8f,0x99,0x75,0xee,0x78,0xc,0x71,0xbf,0xdd,  
# 0xce,0x92,0x6a,0xb2]  
def getSboxK(key):  
 Sbox = [0xD7, 0xDF, 0x02, 0xD4, 0xFE, 0x6F, 0x53, 0x3C, 0x25, 0x6C, 0x99, 0x97, 0x06, 0x56, 0x8F, 0xDE, 0x40, 0x11,  
 0x64, 0x07, 0x36, 0x15, 0x70, 0xCA, 0x18, 0x17, 0x7D, 0x6A, 0xDB, 0x13, 0x30, 0x37, 0x29, 0x60, 0xE1, 0x23,  
 0x28, 0x8A, 0x50, 0x8C, 0xAC, 0x2F, 0x88, 0x20, 0x27, 0x0F, 0x7C, 0x52, 0xA2, 0xAB, 0xFC, 0xA1, 0xCC, 0x21,  
 0x14, 0x1F, 0xC2, 0xB2, 0x8B, 0x2C, 0xB0, 0x3A, 0x66, 0x46, 0x3D, 0xBB, 0x42, 0xA5, 0x0C, 0x75, 0x22, 0xD8,  
 0xC3, 0x76, 0x1E, 0x83, 0x74, 0xF0, 0xF6, 0x1C, 0x26, 0xD1, 0x4F, 0x0B, 0xFF, 0x4C, 0x4D, 0xC1, 0x87, 0x03,  
 0x5A, 0xEE, 0xA4, 0x5D, 0x9E, 0xF4, 0xC8, 0x0D, 0x62, 0x63, 0x3E, 0x44, 0x7B, 0xA3, 0x68, 0x32, 0x1B, 0xAA,  
 0x2D, 0x05, 0xF3, 0xF7, 0x16, 0x61, 0x94, 0xE0, 0xD0, 0xD3, 0x98, 0x69, 0x78, 0xE9, 0x0A, 0x65, 0x91, 0x8E,  
 0x35, 0x85, 0x7A, 0x51, 0x86, 0x10, 0x3F, 0x7F, 0x82, 0xDD, 0xB5, 0x1A, 0x95, 0xE7, 0x43, 0xFD, 0x9B, 0x24,  
 0x45, 0xEF, 0x92, 0x5C, 0xE4, 0x96, 0xA9, 0x9C, 0x55, 0x89, 0x9A, 0xEA, 0xF9, 0x90, 0x5F, 0xB8, 0x04, 0x84,  
 0xCF, 0x67, 0x93, 0x00, 0xA6, 0x39, 0xA8, 0x4E, 0x59, 0x31, 0x6B, 0xAD, 0x5E, 0x5B, 0x77, 0xB1, 0x54, 0xDC,  
 0x38, 0x41, 0xB6, 0x47, 0x9F, 0x73, 0xBA, 0xF8, 0xAE, 0xC4, 0xBE, 0x34, 0x01, 0x4B, 0x2A, 0x8D, 0xBD, 0xC5,  
 0xC6, 0xE8, 0xAF, 0xC9, 0xF5, 0xCB, 0xFB, 0xCD, 0x79, 0xCE, 0x12, 0x71, 0xD2, 0xFA, 0x09, 0xD5, 0xBC, 0x58,  
 0x19, 0x80, 0xDA, 0x49, 0x1D, 0xE6, 0x2E, 0xE3, 0x7E, 0xB7, 0x3B, 0xB3, 0xA0, 0xB9, 0xE5, 0x57, 0x6E, 0xD9,  
 0x08, 0xEB, 0xC7, 0xED, 0x81, 0xF1, 0xF2, 0xBF, 0xC0, 0xA7, 0x4A, 0xD6, 0x2B, 0xB4, 0x72, 0x9D, 0x0E, 0x6D,  
 0xEC, 0x48, 0xE2, 0x33]  
  
 ix = ord(key[0]) - 0x29  
 Sbox[0] = Sbox[ix]  
 Sbox[ix] = 0xd7  
 sboxpos = 1  
  
 while sboxpos < 256:  
 value = Sbox[sboxpos]  
 ix = (ix + ord(key[sboxpos]) + value)%256  
 Sbox[sboxpos] = Sbox[ix]  
 Sbox[ix] = value  
 sboxpos += 1  
  
 return Sbox

接下来就是 RC4 + base64混合加密的地方了

伪代码直译

# 根据伪代码得出  
def getNewBase64Old(password,tbox,rc4key4):  
 app\_password = password # 文本框输入的密码  
 app\_password\_length = len(app\_password)  
 app\_password\_length\_loc = app\_password\_length  
  
 b64\_salt\_size = ord(rc4key4[3])  
  
 app\_password\_tmp = math.floor(app\_password\_length/3)  
 pwd\_b64\_bitlen = 8 \* (3 + 3 \* app\_password\_tmp)  
 new\_base64\_length = math.floor(b64\_salt\_size + pwd\_b64\_bitlen/6)  
 pwdsaltstr = ""  
  
 new\_base64 = [0x00] \* 256  
  
  
 if app\_password\_length\_loc > 0:  
 txi = 0  
 pwd\_idx = 0  
 txj = 0  
 v44 = b64\_salt\_size  
  
 # 12345678 加密后 [76 6C 62 2D 22 3C 6E 30 36 77 6B 34]  
 # loc\_9A8  
 while pwd\_idx < app\_password\_length\_loc:  
 txi = (txi + 1)%256  
 tx = tbox[txi]  
 txj = (txj + tx) % 256  
 tbox[txi] = tbox[txj]  
 tbox[txj] = tx  
 Rc4\_T = tbox[txi]  
 tboxIndex = byteToInteger( tx + Rc4\_T )  
 needTodoBase64 = byteToInteger( tbox[tboxIndex] ^ ord(app\_password[pwd\_idx]) )  
  
 if pwd\_idx == 0:  
 # A0E  
 new\_base64\_index = v44 + pwd\_idx  
 base64table\_index = needTodoBase64>>2  
 base64table\_value = ord(base64table[base64table\_index])  
 base64table\_tmp = base64table\_value^7  
 new\_base64[new\_base64\_index] = base64table\_tmp  
 # 下面的会在下一轮加密的时候被替换掉  
 needbaser64\_tmp = 16 \* needTodoBase64  
 needbaser64\_tmp2 = needbaser64\_tmp & 0x30  
 new\_base64[new\_base64\_index+1] = needbaser64\_tmp2  
  
 else:  
 pwdtem = pwd\_idx%3  
 if pwdtem == 0:  
 new\_base64\_index = v44 + pwd\_idx  
 base64table\_index = needTodoBase64 >> 2  
 base64table\_value = ord(base64table[base64table\_index])  
 base64table\_tmp = base64table\_value ^ 7  
 new\_base64[new\_base64\_index] = base64table\_tmp  
 # 下面的会在下一轮加密的时候被替换掉  
 needbaser64\_tmp = 16 \* needTodoBase64  
 needbaser64\_tmp2 = needbaser64\_tmp & 0x30  
 new\_base64[new\_base64\_index + 1] = needbaser64\_tmp2  
 elif pwdtem == 1:  
 new\_base64\_index = v44 + pwd\_idx  
 base64table\_index\_temp2 = new\_base64[new\_base64\_index]  
 base64table\_index\_temp = needTodoBase64 >> 4  
 base64table\_index = base64table\_index\_temp + base64table\_index\_temp2  
 new\_base64[new\_base64\_index] = ord(base64table[base64table\_index])  
 # 下面的会在下一轮加密的时候被替换掉  
 needbaser64\_tmp = needTodoBase64 << 2  
 needbaser64\_tmp2 = needbaser64\_tmp & 0x3C  
 new\_base64[new\_base64\_index + 1] = needbaser64\_tmp2  
 pwd\_idx\_tem = pwd\_idx+1  
 if pwd\_idx\_tem == app\_password\_length\_loc:  
 new\_base64[new\_base64\_index + 1] = ord(base64table[needbaser64\_tmp2])  
 new\_base64[new\_base64\_index + 2] = 0x34  
  
 elif pwdtem == 2:  
 needbaser64\_tmp2 = needTodoBase64 & 0xC0  
 new\_base64\_index = v44 + pwd\_idx  
 v44 = v44 + 1  
 base64table\_index\_temp2 = new\_base64[new\_base64\_index]  
 base64table\_index\_temp = needbaser64\_tmp2 >> 6  
 base64table\_index = base64table\_index\_temp2 + base64table\_index\_temp  
 base64table\_value = ord(base64table[base64table\_index])  
 # 下面的会在下一轮加密的时候被替换掉  
 base64table\_tmp = base64table\_value ^ 0xf  
 new\_base64[new\_base64\_index] = base64table\_tmp  
 needbaser64\_tmp2 = needTodoBase64 & 0x3F  
 new\_base64[new\_base64\_index + 1] = ord(base64table[needbaser64\_tmp2])  
  
 pwd\_idx = pwd\_idx + 1  
 return new\_base64

优化代码

#优化后的代码  
def getNewBase64(password,tbox,rc4key4):  
 password\_length = len(password)  
 b64\_salt\_size = ord(rc4key4[3])  
  
 # 空数组  
 new\_base64 = []  
  
 if password\_length <= 0:  
 return  
  
 txi = 0  
 pwd\_idx = 0  
 txj = 0  
 new\_base64\_tmpValue = 0  
  
 while pwd\_idx < password\_length:  
 txi = (txi + 1) % 256  
 tx = tbox[txi]  
 txj = (txj + tx) % 256  
 tbox[txi] = tbox[txj]  
 tbox[txj] = tx  
 Rc4\_T = tbox[txi]  
 tboxIndex = byteToInteger(tx + Rc4\_T)  
 needTodoBase64 = byteToInteger(tbox[tboxIndex] ^ ord(password[pwd\_idx]))  
  
 pwdtem = pwd\_idx % 3  
 if pwdtem == 0:  
 base64table\_index = needTodoBase64 >> 2  
 base64table\_value = ord(base64table[base64table\_index])  
 base64table\_tmp = base64table\_value ^ 7  
 new\_base64.append(base64table\_tmp)  
 needbaser64\_tmp = 16 \* needTodoBase64  
 new\_base64\_tmpValue = needbaser64\_tmp & 0x30  
 elif pwdtem == 1:  
 base64table\_index\_temp = needTodoBase64 >> 4  
 base64table\_index = base64table\_index\_temp + new\_base64\_tmpValue  
 new\_base64.append(ord(base64table[base64table\_index]))  
 needbaser64\_tmp = needTodoBase64 << 2  
 new\_base64\_tmpValue = needbaser64\_tmp & 0x3C  
 pwd\_idx\_tem = pwd\_idx + 1  
 if pwd\_idx\_tem == password\_length:  
 new\_base64.append(ord(base64table[new\_base64\_tmpValue]))  
 new\_base64.append(0x34)  
 elif pwdtem == 2:  
 needbaser64\_tmp2 = needTodoBase64 & 0xC0  
 b64\_salt\_size = b64\_salt\_size + 1  
 base64table\_index\_temp = needbaser64\_tmp2 >> 6  
 base64table\_index = new\_base64\_tmpValue + base64table\_index\_temp  
 base64table\_value = ord(base64table[base64table\_index])  
 base64table\_tmp = base64table\_value ^ 0xf  
 new\_base64.append(base64table\_tmp)  
 needbaser64\_tmp2 = needTodoBase64 & 0x3F  
 new\_base64\_tmpValue = ord(base64table[needbaser64\_tmp2])  
 new\_base64.append(new\_base64\_tmpValue)  
 pwd\_idx = pwd\_idx + 1  
 return new\_base64

以下为newbase64加密过程

1

051 = {int} 118

052 = {int} 32

2

051 = {int} 118

052 = {int} 108

053 = {int} 36

3

051 = {int} 118

052 = {int} 108

053 = {int} 98

054 = {int} 45

4

051 = {int} 118

052 = {int} 108

053 = {int} 98

054 = {int} 45

055 = {int} 34

056 = {int} 16

5

051 = {int} 118

052 = {int} 108

053 = {int} 98

054 = {int} 45

055 = {int} 34

056 = {int} 60

057 = {int} 24

6

051 = {int} 118

052 = {int} 108

053 = {int} 98

054 = {int} 45

055 = {int} 34

056 = {int} 60

057 = {int} 110

058 = {int} 48

7

051 = {int} 118

052 = {int} 108

053 = {int} 98

054 = {int} 45

055 = {int} 34

056 = {int} 60

057 = {int} 110

058 = {int} 48

059 = {int} 54

060 = {int} 48

8

051 = {int} 118

052 = {int} 108

053 = {int} 98

054 = {int} 45

055 = {int} 34

056 = {int} 60

057 = {int} 110

058 = {int} 48

059 = {int} 54

060 = {int} 119

061 = {int} 36

跳出循环后的加密

051 = {int} 118

052 = {int} 108

053 = {int} 98

054 = {int} 45

055 = {int} 34

056 = {int} 60

057 = {int} 110

058 = {int} 48

059 = {int} 54

060 = {int} 119

061 = {int} 107

062 = {int} 52

算法分析

本题使用的是魔改的 RC4 和替换码表的 Base64 编码，本函数主要实现的功能如下：

对输入的字符串先调用一次 RC4 算法

再用魔改的 Base64 函数对加密后的字符串进行编码

RC4

先简单介绍 RC4 加密，该加密主要分为以下几个步骤：

根据密钥 key 生成 s-box

根据 s-box 生成伪随机的密钥流

异或按位加密明文

下面我们来看程序内部的 RC4 加密，和经典的 RC4 算法一样，程序内部的 RC4 算法也分为这样几个步骤：

base64是用64个可打印字符表示二进制所有数据方法。base64是先把字符串拆分成6bit,然后6bit存储到8bit中，最后长度会增加1/3。

由于2的6次方等于64，所以可以用每6个位元为一个单元，对应某个可打印字符。我们知道三个字节有24个位元，就可以刚好对应于4个Base64单元，即3个字节需要用4个Base64的可打印字符来表示。

# 参考

https://bbs.pediy.com/thread-262472.htm

https://bbs.pediy.com/thread-250348-1.htm

http://blog.syang.xyz/2019/04/kanxue-transformer/

https://bbs.pediy.com/thread-250413.htm

https://xz.aliyun.com/t/4614