安卓

RSA

乍一看题目以为本题与 RSA 加解密相关，那么我们开始吧

静态分析：

先使用 JEB 分析 JAVA 层逻辑

```
    static {
        System.loadLibrary("hello-libs");
    }

    public MainActivity() {
        super();
    }

    public void onClickTest(View arg3) {
        this.n.setText("Empty Input");
        if(this.stringFromJNI(this.m.getText().toString())) {
            this.n.setText("Correct");
        }
        else {
            this.n.setText("Wrong");
        }
    }

    protected void onCreate(Bundle arg2) {
        super.onCreate(arg2);
        this.setContentView(2130968602);
        this.m = this.findViewById(2131427413);
        this.n = this.findViewById(2131427415);
    }

    public native boolean stringFromJNI(String arg1) {
    }
}
```

可见关键验证函数都在 so 层，使用 IDA 打开 libhello-libs.so

找到 stringFromJNI 方法映射到 so 层的 Java_com_didictf_guesskey2018one_MainActivity_stringFromJNI 函数

```
int __fastcall Java_com_didictf_guesskey2018one_MainActivity_stringFromJNI(_JNIEnv *a1, int a2, char *key)
{
  int v3; // r5
  const char *cKey; // r4
  int *v5; // r0
  int v6; // r4
  int v8; // [sp+4h] [bp-28h]
  char v9; // [sp+8h] [bp-24h]
  int v10; // [sp+Ch] [bp-20h]
  int v11; // [sp+10h] [bp-1Ch]
  char v12; // [sp+14h] [bp-18h]
  char v13; // [sp+18h] [bp-14h]

  v3 = 0;
  cKey = a1->functions->GetStringUTFChars(&a1->functions, key, 0);
  j_GetTicks();
  do
    v11 = j_gpower(v3++);
  while ( v3 != 32 );
  j_GetTicks();
  sub_3133C(&v10, cKey, &v9);
  v5 = sub_309E0(&v8, &v10);
  v6 = j___aeabi_wind_cpp_prj(v5);              // 返回值取决于该函数
  sub_308BC(v8 - 12, &v13);
  sub_308BC(v10 - 12, &v12);
  return v6;
}
```

涉及的函数较多，暂时可以排除 sub_308BC 函数，开始动态调试辅助分析

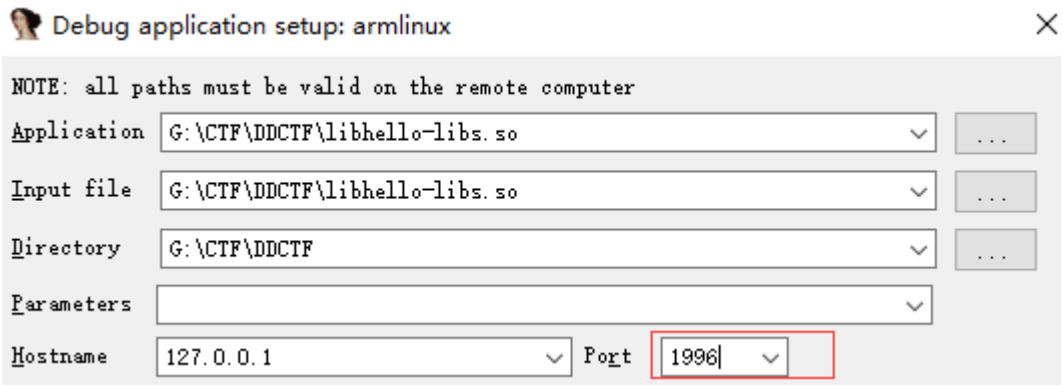
动态调试

```
root@mako:/ # ./data/local/tmp/as -p1996
./data/local/tmp/as -p1996
IDA Android 32-bit remote debug server(ST) v1.22. Hex-Rays (c) 2004-2017
Listening on 0.0.0.0:1996...
```

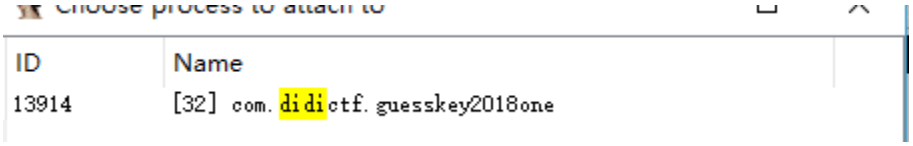
自定义端口启动 android_server

```
C:\Users\xiaobin>adb forward tcp:1996 tcp:1996
```

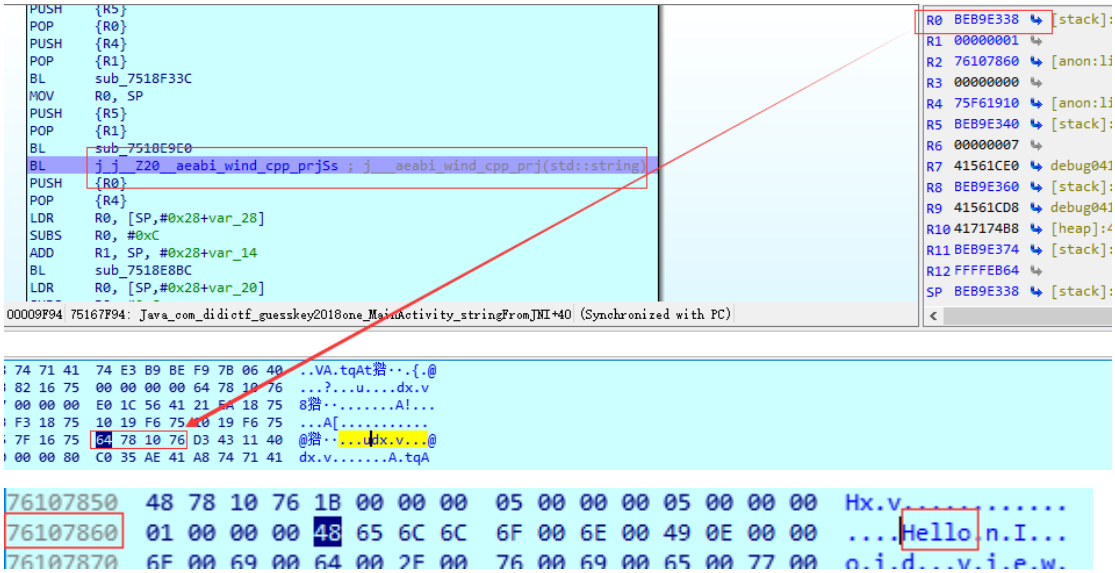
端口转发



设置 IDA process option



开始附加进程，接下来就可以愉快的动态调试了



起初浪费了一些时间分析 j_j__Z20__aeabi_wind_cpp_prjSs 函数之前的函数，最后发现 j_j__Z20__aeabi_wind_cpp_prjSs 的参数指向的地址就是我们输入的字符串，那么说明他之前

不过是一些对输入字符串的无关处理，直接忽略，只分析 j_j__Z20__aeabi_wind_cpp_prjSs 函数

```
v1 = *a1;
if ( *(v1 - 12) == 36 )
{
  v2 = 0;
  do
```

```
SUB     SP, SP, #0x74
LDR     R1, =(__stack_chk_guard_ptr - 0x75167C76)
ADD     R1, PC          ; __stack_chk_guard_ptr
LDR     R1, [R1]        ; __stack_chk_guard
LDR     R1, [R1]
STR     R1, [SP,#0x88+var_18]
MOVS    R1, #0xB
MVNS    R3, R1
STR     R0, [SP,#0x88+var_80]
LDR     R0, [R0]
LDR     R1, [R0,R3]
MOVS    R6, #0
CMP     R1, #0x24 ; '$'
BEQ     loc_75167C8C
```

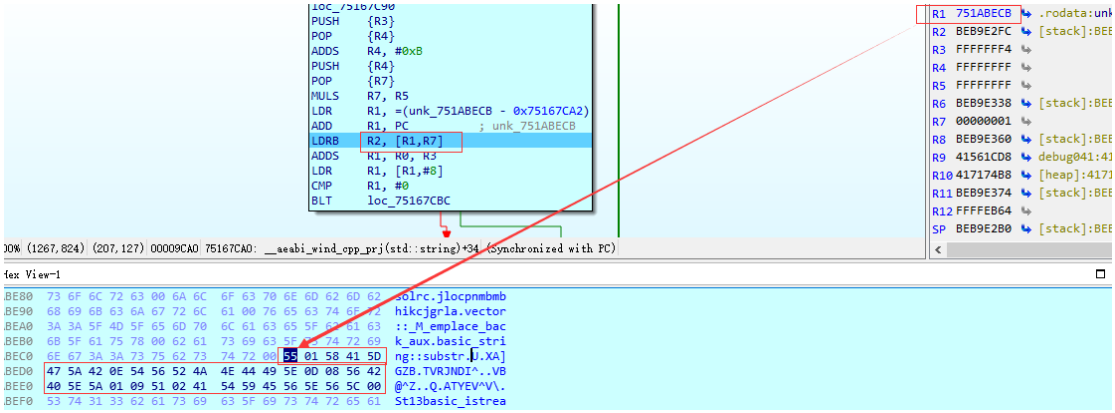
```
R0  76107864 [anon:lib
R1  00000005
R2  76107860 [anon:lib
R3  FFFFFFF4
R4  75F61910 [anon:lib
R5  BEB9E340 [stack]:B
R6  00000000
R7  41561CE0 debug041:
R8  BEB9E360 [stack]:B
R9  41561CD8 debug041:
R10 41717488 [heap]:41
```

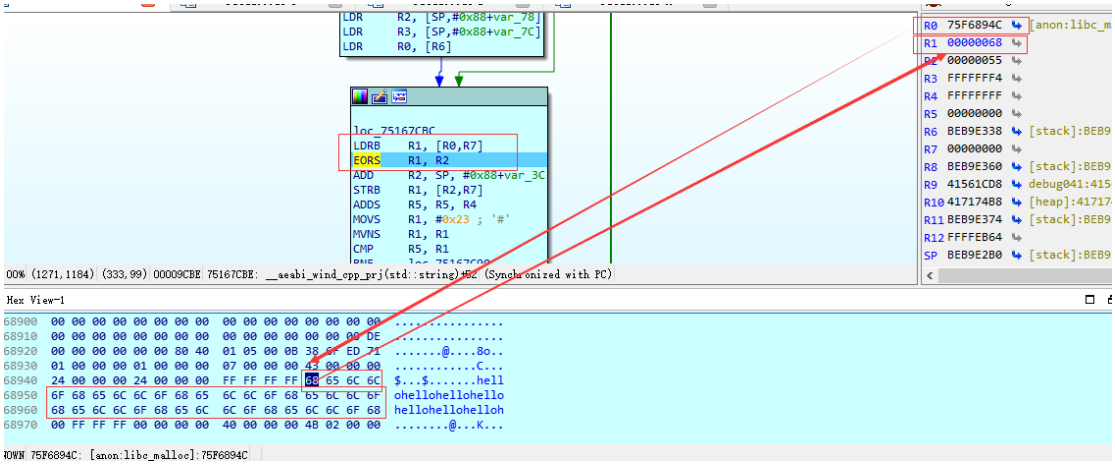
结合 F5 伪代码以及 ARM 汇编可以确定该处为字符串长度验证，输入字符串必须为 36 位

```
do
{
  v3 = *(&unk_751ABECB - v2);
  if ( *(v1 - 4) >= 0 )
  {
    v4 = *(&unk_751ABECB - v2);
    sub_7518DF64(v23);
    v3 = v4;
    v1 = *v23;
  }
  v39[-v2] = *(v1 - v2) ^ v3;
  --v2;
}
while ( v2 != -36 );
```

```
loc_75167CBC
LDRB    R1, [R0,R7]
EORS    R1, R2
ADD     R2, SP, #0x88+var_3C
```

第二段循环静态看一脸懵逼，使用动态调试重点关注被异或双方的由来



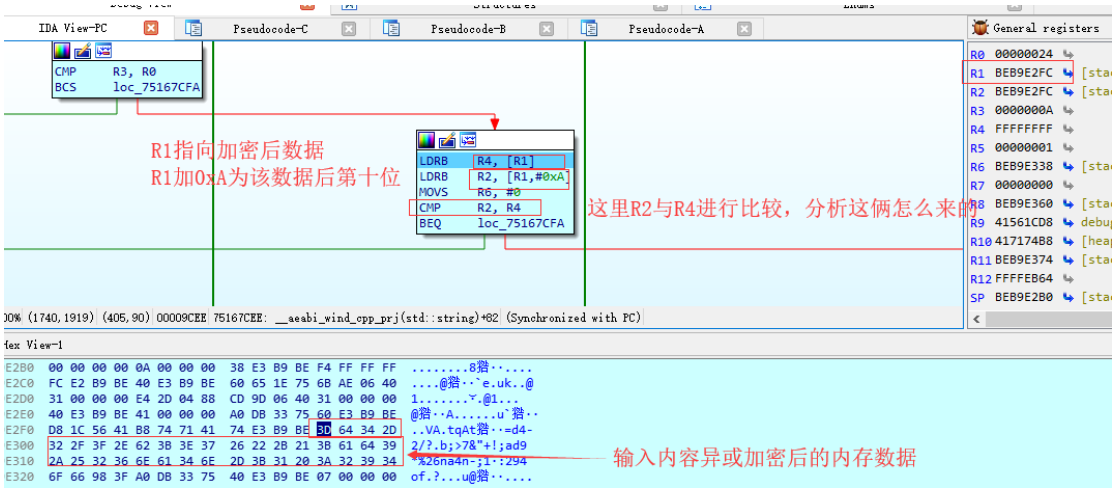


因此这一段循环将输入字符串的 ASCII 码值与某段数据（我们后面把其称为 enKey）异或

接下来看下一个验证

```
LABEL_7:
    v21 = v8;
    v25 = v6;
    while ( v9 < 1 || v8 >= v5 || v6[10] == *v6 )
    {
      ++v8;
      ++v6;
      if ( ++v7 >= 10 )
      {
        v8 = v21 + 10;
        v6 = v25 + 10;
        ++v9;
        v7 = 0;
        if ( v9 < 5 )
          goto LABEL_7;
```

静态分析依然难以理解，动态调试重点关注 cmp 指令

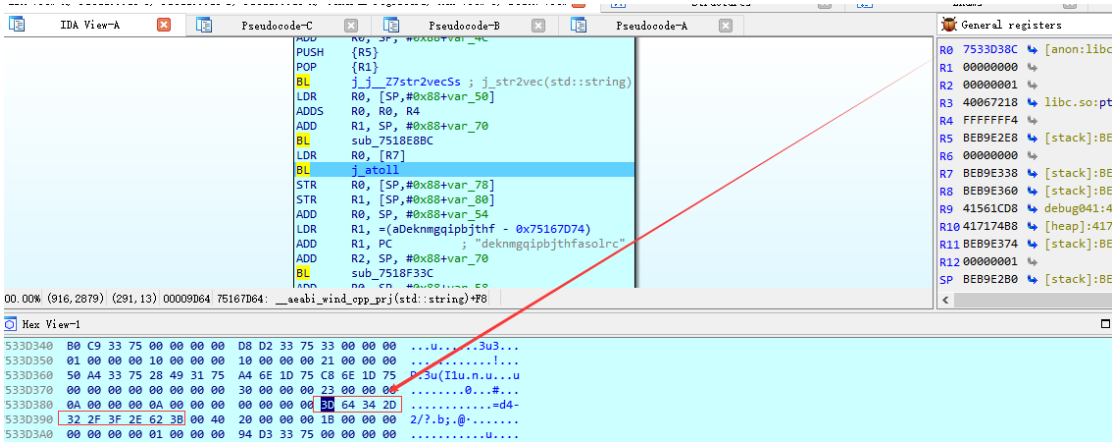


动态调试一次很容易发现这是一个字符串自身校验，即加密后字符串第 1、11、21、31 相等，

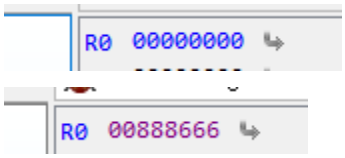
第 2、12、22、32 相等，依次类推

接下来跟踪到附近的可疑函数

```
v26 = j_atoll(*v23);
v24 = v11;
```



原来就是加密后字符串的前十位，通过上一轮检验后要转换为 long long 类型(我胡乱输得数是不满足上一轮检验的，这里为了分析整体逻辑直接 set ip 过来了)



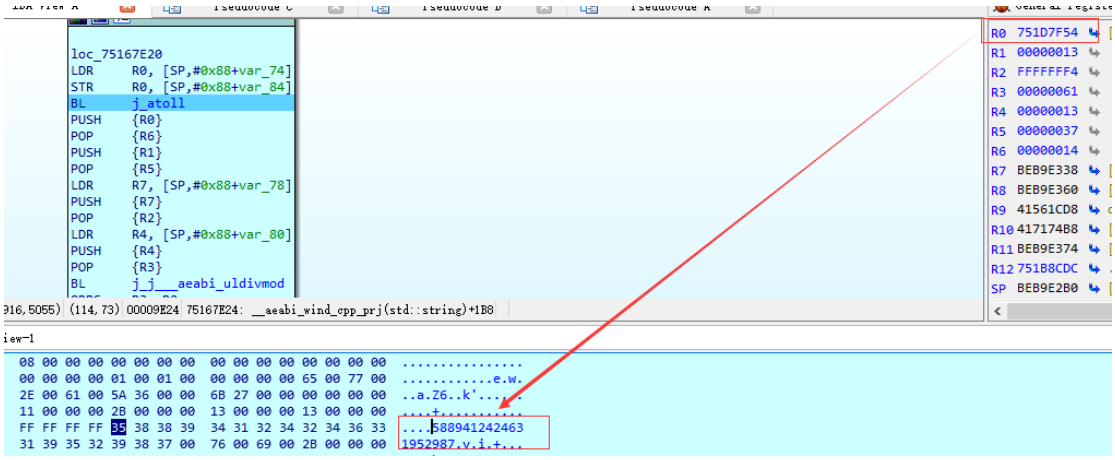
因此转换结果为 0，为了方便接下来分析时联想到这里，我手动把 r0 修改成别的值

```
v26 = j_atoll(*v23);
v24 = v11;
sub_7518F33C(&v34, "deknmgqipbjthfasolrc", &v28);
sub_7518F33C(&v33, "jlocpnmbmbhikcjgrla", &v28);
j___aeabi_memclr4(&v29, 20);
```

接下来是俩字符串加上两个复杂的循环，直接跟是不可能跟的，太麻烦了，直接循环后找个时机查看发生了什么

```
v22 = v27;
v17 = j_atoll(v27);
```

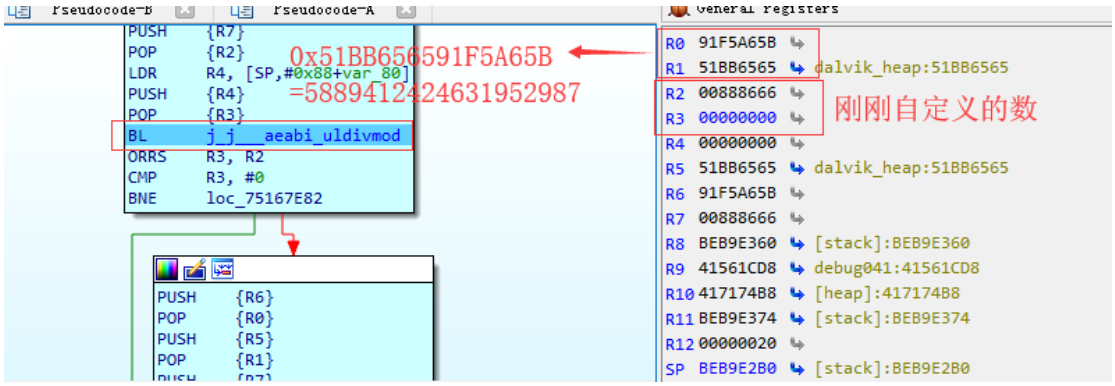
时机就在他一波乱七八糟的操作后的一个转换函数，直接在这里下个断点拦截数据



OK，直接获得核心数据 5889412424631952987

```
v19 = v18;
j_j___aeabi_uldivmod(v17, v18, v26, v24);
if ( !v20 )
```

接下来就是个取余运算了



0x51BB656591F5A65B=5889412424631952987

原来就是让 5889412424631952987 对加密后字符串的前十位取余，后面没什么好分析的，这里只要满足取余结果为 0 手机就会显示 Correct

本题和 RSA 加解密算法的关系就是名字听起来一样

逆推求解

思路一：找出一定范围满足被 5889412424631952987 取余为 0 的数，填充为 36 位，与 enKey 再次异或逆推出结果，但结果必须满足题目所说的 0-9 a-z 的数据

思路二：严格限定条件，我们输入的数据必须满足题目所说的 0-9 a-z 的范围，将其与 enKey 异或，异或后需满足是整数，并且满足自身的验证关系（每十位分别相等），得到所有满足的数据，与 5889412424631952987 取余为 0 就是我们所找的，最后恢复到原始输入

附上思路二的代码如下：

findNumber.py

```
data =
[48,49,50,51,52,53,54,55,56,57,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,
113,114,115,116,117,118,119,120,121,122]
enList = [0x55,0x01, 0x58,
0x41 ,0x5D ,0x47 ,0x5A ,0x42 ,0x0E ,0x54,0x56 ,0x52 ,0x4A ,0x4E ,0x44 ,0x49 ,0x5E ,0x0D ,0x0
8 ,0x56,0x42 ,0x40 ,0x5E ,0x5A ,0x01 ,0x09 ,0x51 ,0x02 ,0x41 ,0x54,0x59 ,0x45 ,0x56 ,0x5E ,0
x56 ,0x5C ]
result=[]
for cur in range(10):
    tmpList=[]
    for num in data:
        tmp = num^enList[cur]#寻找前十个元素
        if(tmp>57 or tmp<48):#加密后不为数字直接淘汰
            continue
        for num in data:
            tmp1 = num^enList[cur+10]#寻找 10-20 个元素
            if(tmp1>57 or tmp1<48):#加密后不为数字直接淘汰
                continue
            for num in data:
                tmp2 = num^enList[cur+20]#寻找 20-30 个元素
                if(tmp2>57 or tmp2<48):#加密后不为数字直接淘汰
                    continue
                for num in data:
                    if(cur<6):
                        tmp3 = num^enList[cur+30]#寻找 30-36 个元素
                        if(tmp3>57 or tmp3<48):#加密后不为数字直接淘汰
                            continue
                         #运行至此，完成初步筛选
                        if(tmp==tmp1 and tmp==tmp2 and tmp==tmp3):
                            print "第%s 个元素是"%(cur),chr(tmp)
                            tmpList.append(str(chr(tmp)))
                    else:
                        if(tmp==tmp1 and tmp==tmp2):
                            print "第%s 个元素是"%(cur),chr(tmp)
                            tmpList.append(str(chr(tmp)))
                            break
    result.append(tmpList)
print result
#接下来就是排列组合问题，简单粗暴，十层嵌套
for a in result[0]:
    for b in result[1]:
        for c in result[2]:
            for d in result[3]:
                for e in result[4]:
```

```
                for f in result[5]:
                    for g in result[6]:
                        for h in result[7]:
                            for i in result[8]:
                                for j in result[9]:
                                    number=long(a+b+c+d+e+f+g+h+i+j)
                                    if(588941242463195298 7%number==0):
                                        print number
```

recoveryData.py

```
number=1499419583#这是上一轮的结果
enList = [0x55,0x01, 0x58,
0x41 ,0x5D ,0x47 ,0x5A ,0x42 ,0x0E ,0x54,0x56 ,0x52 ,0x4A ,0x4E ,0x44 ,0x49 ,0x5E ,0x0D ,0x0
8 ,0x56,0x42 ,0x40 ,0x5E ,0x5A ,0x01 ,0x09 ,0x51 ,0x02 ,0x41 ,0x54,0x59 ,0x45 ,0x56 ,0x5E ,0
x56 ,0x5C ]
enStr = str(number)*4
result=""
for i in range(len(enList)):
    result+=chr(enList[i]^ord(enStr[i]))
print result
```

Hello Baby Dex

首先 JEB 静态分析

```
application android:allowBackup="true" android:icon="@mipmap/ic_launcher" android:label
   <activity android:name="cn.chaitin.geektan.crackme.MainActivity">
       <intent-filter>
           <action android:name="android.intent.action.MAIN" />
           <category android:name="android.intent.category.LAUNCHER" />
       </intent-filter>
```

来到程序入口类

```
   public void onClick(View arg9) {
       Toast v0_6;
       MainActivity v0_5;
       String v1_3;
       int v4 = 18;
       Object[] v0 = new Object[1];
       v0[0] = arg9;
       ChangeQuickRedirect v2 = cn.chaitin.geektan.crackme.MainActivity$1.changeQuickRedirect;
       Class[] v5 = new Class[1];
       Class v1 = View.class;
       v5[0] = v1;
       Class v6 = Void.TYPE;
       cn.chaitin.geektan.crackme.MainActivity$1 v1_1 = this;
       boolean v0_1 = PatchProxy.isSupport(v0, v1_1, v2, false, v4, v5, v6);
       if(v0_1) {
```

```
else {
    EditText v0_2 = this.val$input_text;
    Editable v0_3 = v0_2.getText();
    v0_1 = TextUtils.isEmpty(((CharSequence)v0_3));
    if(!v0_1) {
        v0_2 = this.val$input_text;
        v0_3 = v0_2.getText();
        String v0_4 = v0_3.toString();
        StringBuilder v1_2 = new StringBuilder();
        String v2_1 = "DDCTF{";
        v1_2 = v1_2.append(v2_1);
        v2_1 = this.val$result;
        v1_2 = v1_2.append(v2_1);
        v2_1 = "}";
        v1_2 = v1_2.append(v2_1);
        v1_3 = v1_2.toString();
        v0_1 = v0_4.equals(v1_3);
        if(v0_1) {
            v0_5 = MainActivity.this;
            v1_3 = "恭喜大佬！密码正确！";
            v0_6 = Toast.makeText(((Context)v0_5), ((CharSequence)v1_3), 0);
            v0_6.show();
            return;
        }
    }

    v0_5 = MainActivity.this;
    v1_3 = "大佬莫急！再试试！";
    v0 6 = Toast.makeText(((Context)v0 5), ((CharSequence)v1 3), 0);
```

那么关键逻辑就在这了，这么简单的吗？

```
    this.setContentView(v0_2);
    this.runRobust();
    String v0_3 = "1B:D0:4A:9D:B5:A9:84:93:7E:79:27:9C:6C:C4:14:AB:DD:B0:75:7F";
    SignCheck v10 = new SignCheck(this, ((Context)this), v0_3);
    v10.check();
    Debug.isDebuggerConnected();
```

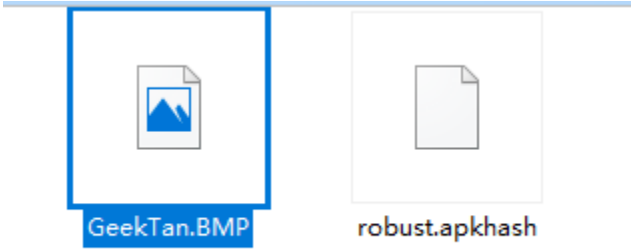
程序有签名验证，当然不排除还可能有其他的防篡改校验，这里直接用 IDA 动态调试 DEX 拦截 flag，得到的结果是错误的，而且似乎根本不会运行到本代码的“恭喜大佬…”和“大佬莫急….”这里。那么必然是其前面可以的代码作祟，推测是 DEX 热修复技术。

这里有很多思路，比如：

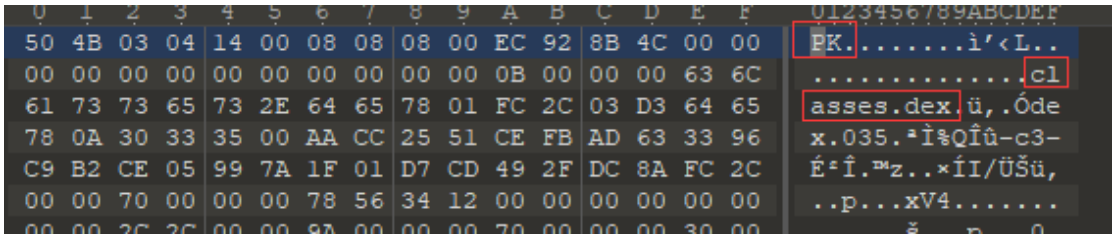
1. hook 这个类，直接拿到 this.val$result 字段

2. hook equals()方法，也应该能拿到

这里不使用 hook，看看安装包中有什么蹊跷



Assets 目录多了两个文件，非常可疑



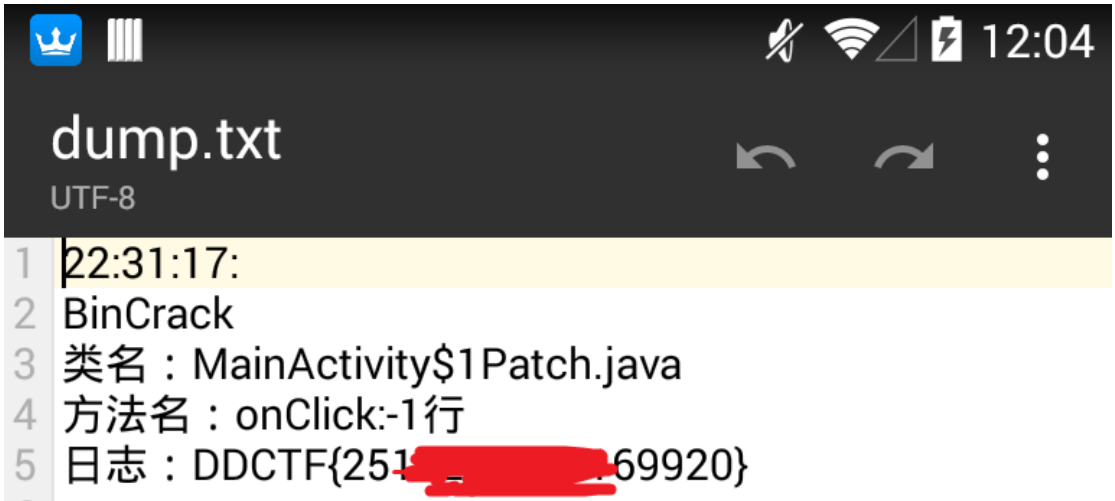
原来是个 zip 压缩文件，伪装成 BMP，里面至少有个 classes.dex 文件

```
    v0_2 = EnhancedRobustUtils.getFieldValue("this$0", v0, MainActivity$1.class);
    v2_4 = new Object[v11];
    v2_4[0] = v0_2;
    v2_4[1] = "恭喜大佬！密码正确！";
    v2_4[v9] = new Integer(0);
    v0_3 = this.getRealParameter(v2_4);
    v2_5 = Toast.class;
    v3_2 = new Class[v11];
    v3_2[0] = Context.class;
    v3_2[1] = CharSequence.class;
    v3_2[v9] = Integer.TYPE;
    v0_2 = EnhancedRobustUtils.invokeReflectStaticMethod("makeText", v2_5, v0_3, v3_2);
    if((((MainActivity$1Patch)v0_2)) == this) {
        v0 = ((MainActivity$1Patch)v0_2).originClass;
    }

    EnhancedRobustUtils.invokeReflectMethod("show", v0, new Object[0], v10, Toast.class);
}
else {
label_217:
    if((this instanceof MainActivity$1Patch)) {
        v0 = this.originClass;
    }
    else {
        v0_1 = this;
    }

    v0_2 = EnhancedRobustUtils.getFieldValue("this$0", v0, MainActivity$1.class);
    v2_4 = new Object[v11];
    v2_4[0] = v0_2;
    v2_4[1] = "大佬莫急！再试试！";
    v2_4[v9] = new Integer(0);
```

在里面也有个叫 onClick 的方法，使用了一堆 java 反射，好了，这就是真正运行的逻辑

接下来直接插入自己编写的 smali 代码扣出来 flag

插入代码还需要注意签名验证问题，我手机已经破解过了安卓核心，修改 apk 后不签名即可安装，绕过签名验证（当然可以 hook 绕过，或者修改签名验证函数）



Diffie-Hellman

JEB 静态分析

```
static {
    System.loadLibrary("hello-libs");
}

public MainActivity() {
    super();
}

public void onClickTest(View arg3) {
    this.n.setText("Empty Input");
    if(this.stringFromJNI(this.m.getText().toString())) {
        this.n.setText("Correct");
    }
    else {
        this.n.setText("Wrong");
    }
}

protected void onCreate(Bundle arg2) {
    super.onCreate(arg2);
    this.setContentView(2130968602);
    this.m = this.findViewById(2131427413);
    this.n = this.findViewById(2131427415);
}

public native boolean stringFromJNI(String arg1) {
}
```

和第一题长得真像

IDA 打开，找到关键函数

```
if ( !v10 )
{
  v11 = 2;
  j_j___aeabi_ldivmod(2);
  if ( v16 ^ 1 | v8 )
  {
    do
    {
      v13 = 1;
      if ( v11 >= v16 )
        v13 = 0;
      v14 = 1;
      if ( v11 >> 31 >= v8 )
        v14 = 0;
      if ( v11 >> 31 != v8 )
        v13 = v14;
      j_j___aeabi_ldivmod(2 * v12);
      ++v11;
    }
    while ( v13 );
  }
  v3 = 1;
  if ( mod_residual != v12 )
    v3 = 0;
}
```

关键逻辑就是这一堆东西，其实比起第一题已经简单了不知道多少了

利用循环一直对某个数取余，最后取余的结果要等于另外一个数才行，答案就是这个循环的次数

```
num=0x2
cur=0x2

def change(num):
    list=[]
    list.append(num%0x100000000)#低位
    list.append(num/0x100000000) #高位
    return list

while(cur<=0x1000000):
    temp = num%0x17A904F1B91290
    if(temp==0xDBDEE7AE5A90):
        break
    new = change(temp)
    r0 = new[0]>>31
    r1 = new[1]<<1
    print "              %x , %x"%(r0,r1)
    r1=r1|r0
    r0 = new[0]<<1

    num=r0%0x100000000+r1*0x100000000
    print "%x , %x"%(cur,num)
    cur+=1
```

很简单，直接贴上代码，运行到一定次数后就成功了

ECC

ECC

180

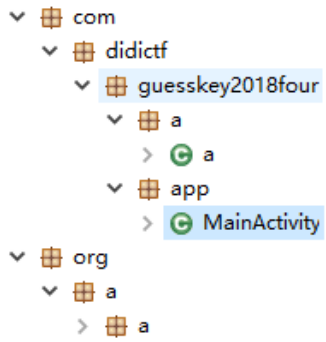
Android第4题：Elliptic Curve Cryptography(ECC)在信息安全、加密破解和数字货币中应用广泛。请阅读此网页，（https://arstechnica.com/information-technology/2013/10/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/2/），根据文中的dot算术规则根据乘积(public key)破解出multiplier乘数（private key）输入到app里。提示：答案是一个10位以内的正整数，例如提交的答案为DDCTF{012345}。

题目看起来很高大上，好像是让我们研究 ECC 的破解，提示给的链接还是个全英文的文档，对于我这样英语不咋滴的看不懂怎么办，那就直接不看了！

JEB 静态分析

```
public MainActivity() {
    super();
    this.m = "00C3632B69D3FC1DD8D80C288C44281B67F4828DC77E37EE338E830E66DC71972A008835BA3156353815DFEDEB4330B48B454F35A88D83DA6260C206E4A(
}

public void onClickTest(View arg9) {
    this.o.setText("Empty Input");
    String v0 = this.n.getText().toString();
    if(v0.length() == 0) {
        v0 = "1";
    }

    new a().a();
    n v0_1 = org.a.a.a.a.a("secp256k1").a().a(new BigInteger(v0.getBytes()));
    BigInteger v2 = v0_1.e().a();
    BigInteger v0_2 = v0_1.f().a();
    byte[] v3 = v2.toByteArray();
    byte[] v4 = v0_2.toByteArray();
    byte[] v5 = new byte[v3.length + v4.length];
    int v0_3;
    for(v0_3 = 0; v0_3 < v5.length; ++v0_3) {
        byte v2_1 = v0_3 < v3.length ? v3[v0_3] : v4[v0_3 - v3.length];
        v5[v0_3] = v2_1;
    }

    StringBuilder v2_2 = new StringBuilder();
    int v3_1 = v5.length;
    for(v0_3 = 0; v0_3 < v3_1; ++v0_3) {
        v2_2.append(String.format("%02X", Byte.valueOf(v5[v0_3])));
    }

    if(v2_2.toString().equals(this.m)) {
        this.o.setText("Correct");
```

看来还真是椭圆曲线加密，只有加密后的结果



看一眼类结构，其中 com.didictf.guesskey2018four.a 类初始化的一些字段都是没有被调用的，

推测是混淆视听的，直接无视

奇怪的就是为什么要有 org 的包，肯定不是自己写的代码!!!

同时在安装包中也发现了这个目录，原来是使用的 spongycastle 开源库

那么我们也下载一份，copy 到我们的工程中

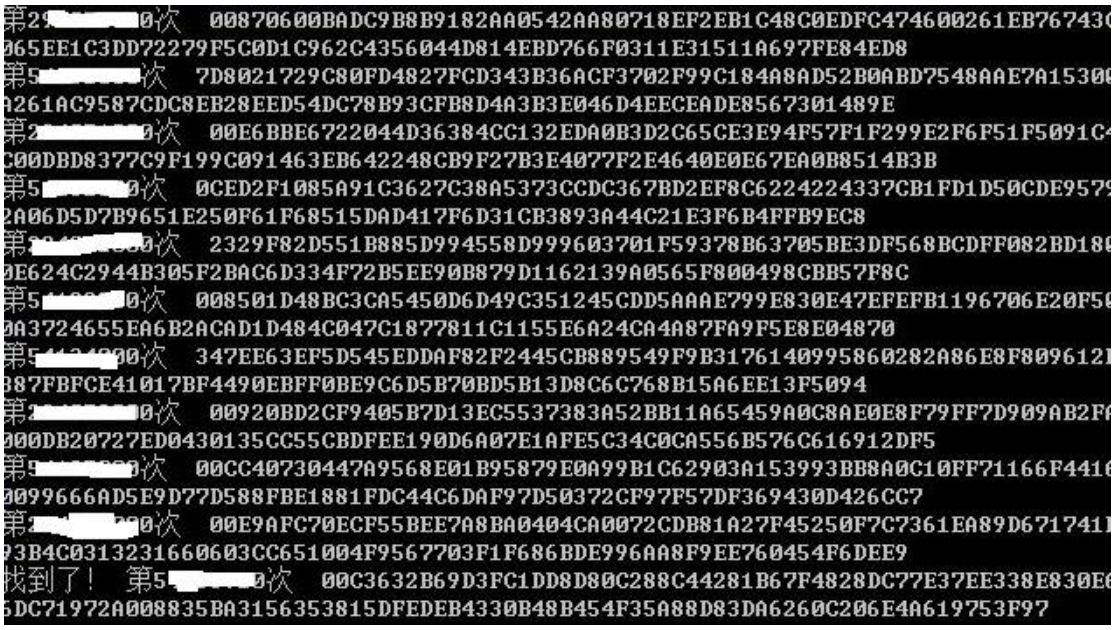
```
ECPoint abc = SECNamedCurves.getByName("secp256k1").getG().multiply(new BigInteger(""+i).getBytes());
BigInteger x = abc.getXCoord().toBigInteger();
BigInteger y = abc.getYCoord().toBigInteger();
```

还原出被混淆的方法，就可以直接暴力破解了

```
public static void calc(long j,long count){
      for (long i =j;i<=count;i++){
         ECPoint abc =
SECNamedCurves.getByName("secp256k1").getG().multiply(new
BigInteger((""+i).getBytes()));
         BigInteger x = abc.getXCoord().toBigInteger();
         BigInteger y = abc.getYCoord().toBigInteger();
         byte[] v3 = x.toByteArray();
          byte[] v4 = y.toByteArray();
          byte[] v5 = new byte[v3.length + v4.length];
          int v0_2;
          for(v0_2 = 0; v0_2 < v5.length; ++v0_2) {
             byte v2_1 = v0_2 < v3.length ? v3[v0_2] :
v4[v0_2 - v3.length];
             v5[v0_2] = v2_1;
          }

          StringBuilder v2_2 = new StringBuilder();
          int v3_1 = v5.length;
          for(v0_2 = 0; v0_2 < v3_1; ++v0_2) {
             v2_2.append(String.format("%02X",
Byte.valueOf(v5[v0_2])));
          }
          if(i%1000L==0)

            System.out.println("第"+i+"次  "+v2_2);

          if(m.equals(v2_2.toString())){
```

```
                System.out.println("找到了！ 第"+i+"次  "+v2_2);
                System.exit(0);
                break;
            }
        }
    }
```

当时很担心别真的是十位数了，找来四台电脑开多线程暴力破解。感谢出题人给了个八位数，睡了一觉第二天起来交 flag



破解密钥

个人觉得这个题大概是这五个中最简单的了，为什么难度在降低呢

JEB 静态分析

```
public void onClick(View arg4) {
    if(arg4.equals(this.validate)) {
        if(CtfLib.validate(String.valueOf(this.key.getText()).trim())) {
            Toast.makeText(((Context)this), "验证成功", 1).show();
        }
        else {
            Toast.makeText(((Context)this), "验证失败", 1).show();
        }
    }
}
```

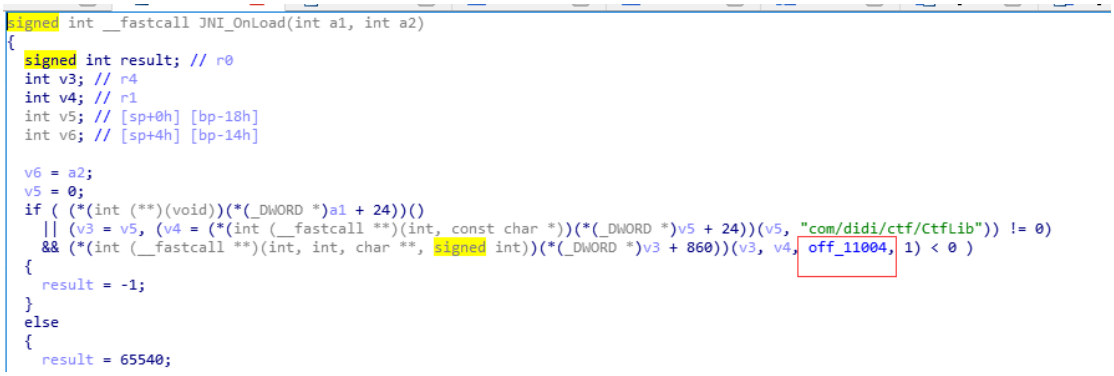
去寻找 CtfLib.validate 方法

```java
public class CtfLib {
    static {
        System.loadLibrary("didictf");
    }

    public CtfLib() {
        super();
    }

    public static native boolean validate(String arg0) {
    }
}
```

IDA 启动、分析 so

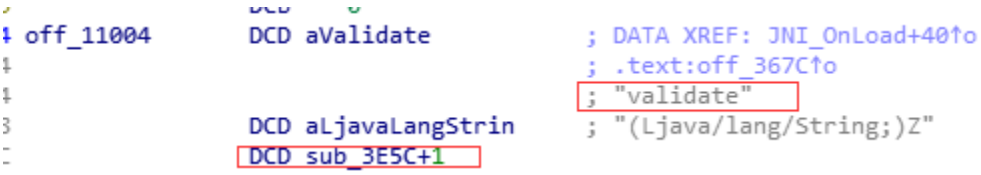
Name	Address	Ordinal
JNI_OnLoad	00003614	
_Unwind_Complete	00006B98	
_Unwind_DeleteException	00006B9C	
_Unwind_GetCFA	00006A3C	
_Unwind_GetDataRelBase	00007B24	

导出函数列表中找不到映射到 java 层的函数，那么必然是动态注册的了



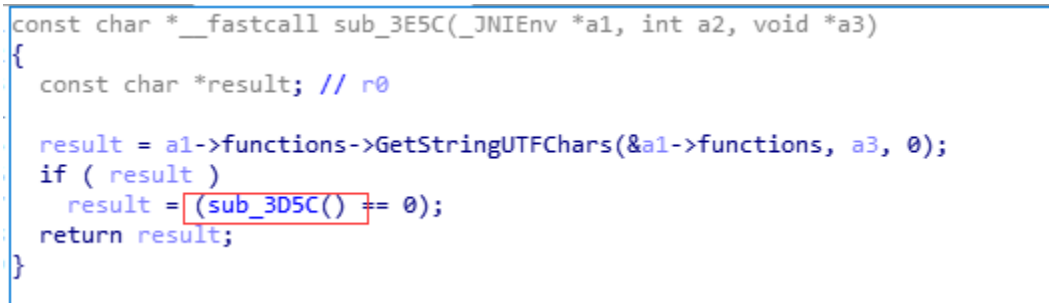
```
signed int __fastcall JNI_OnLoad(int a1, int a2)
{
  signed int result; // r0
  int v3; // r4
  int v4; // r1
  int v5; // [sp+0h] [bp-18h]
  int v6; // [sp+4h] [bp-14h]

  v6 = a2;
  v5 = 0;
  if ( (*(int (**)(void))(*(_DWORD *)a1 + 24))()
    || (v3 = v5, v4 = (*(int (__fastcall **)(int, const char *))(*(_DWORD *)v5 + 24))(v5, "com/didi/ctf/CtfLib")) != 0)
    && (*(int (__fastcall **)(int, int, char **, signed int))(*(_DWORD *)v3 + 860))(v3, v4, off_11004, 1) < 0 )
  {
    result = -1;
  }
  else
  {
    result = 65540;
  }
```

直接来到 Jni_Onload 点这个三元组就完事



```
off_11004       DCD aValidate           ; DATA XREF: JNI_OnLoad+40↑o
                                        ; .text:off_367C↑o
                                        ; "validate"
                DCD aLjavaLangStrin     ; "(Ljava/lang/String;)Z"
                DCD sub_3E5C+1
```

OK，定位到了 so 层的验证函数



```
const char *__fastcall sub_3E5C(_JNIEnv *a1, int a2, void *a3)
{
  const char *result; // r0

  result = a1->functions->GetStringUTFChars(&a1->functions, a3, 0);
  if ( result )
    result = (sub_3D5C() == 0);
  return result;
}
```

只需要分析红框中的函数即可

```
v1 = a1;
memset(&v6, 0, 0x20u);
v2 = dlopen("libc.so", 0);
v3 = v2;
if ( v2 )
{
  open = dlsym(v2, "open");
  close = dlsym(v3, "close");
  read = dlsym(v3, "read");
  strncmp = dlsym(v3, "strncmp");
  strstr = dlsym(v3, "strstr");
}
dword_110CC = 0;
sub_3804(&unk_11024);
sub_3C54();
sub_3B9C(&unk_11024, &v6);
if ( strlen(v1) != 32 )                    // 验证长度
  return -1;
v4 = 0;
do
{
  *(&v6 + v4) ^= v1[v4];
  ++v4;
}
while ( v4 != 32 );                        // 异或加密
return strncmp(&v6, &unk_7D6C);            // 字符串比较
}
```

其中还有好几处的调用，有一个好玩的地方是 sub_3C54（）

```
}
while ( v0 != 256 );
sub_3B9C(&v8, &unk_110D4);
result = open("/proc/self/status", 0);
v3 = result;
if ( result > 0 )
{
  while ( 1 )
  {
    v4 = sub_34E8(v9, 1024, v3);
    if ( v4 <= 0 )
      break;
    if ( !strncmp(v9, "TracerPid:") )
    {
      if ( v4 > 12 )
```

原来是个 TracerPid 的反调试，不禁会心一笑，我此前修改了安卓内核源码编译的安卓系统又能派上用场了，天生免疫反调试（猜测是个陷阱，如果被检测到，会导致 flag 是错的，不过管他呢，又检测不到我，无视了）

其他的一些函数就是初始化一个用来加密的数组了，动态调试一下把这个数组 dump 出来，再把&unk_7D6C 指向的数据 dump 出来，再异或一下轻松搞定

```
list1=[0xA9 ,0xDE ,0xAD ,0x01 ,0x83 ,0xBB ,0x4B ,0x31 ,0xB7 ,0x84 ,0x54 ,0xDD ,0x8F ,0x65 ,0
x0C ,0x44 ,0xBD ,0x13 ,0xAB ,0xB4 ,0xAE ,0x73 ,0x54 ,0xD9 ,0x69 ,0x64 ,0x32 ,0x0A ,0xAC ,0x3
6 ,0x0B ,0x97]
list2=[0xED ,0x9A ,0xEE ,0x55 ,0xC5 ,0xC0 ,0x0C ,0x5E ,0xD8 ,0xE0 ,0x1E ,0xB2 ,0xED ,0x49 ,0
x4F ,0x2B ,0xD3 ,0x74 ,0xD9 ,0xD5 ,0xDA ,0x06 ,0x38 ,0xB8 ,0x1D ,0x0D ,0x5D ,0x64 ,0xDF ,0x1
7 ,0x2A ,0xEA]
res=[]
```

```
for i in range(len(list1)):
    res.append(chr(list1[i]^list2[i]))
print "".join(res)
```