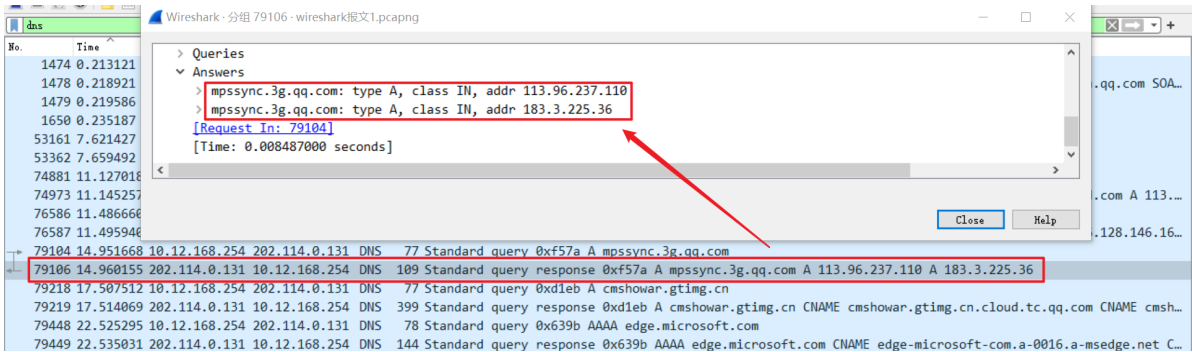


# 任务1：流量分析

- 借助Wireshark抓取Android模拟器中“QQ同步助手”登录和同步数据时的流量，回答以下问题：
- （1）筛选流量中，对应域名“mpssync.3g.qq.com”的IP地址；
  - （2）同步报文的TCP流量源IP：端口，目的IP：端口；
  - （3）分析同步流量的数据特征，并根据这些特征，能否获取报文的一些信息，例如密文长度信息；
  - （4）保存并上传流量分组文件。

首先登录QQ，然后再打开QQ同步助手，此时打开Wireshark对WLAN网卡进行抓包，随后点击同步按钮利用QQ登录。随后对抓取到的报文进行如下分析：

## “mpssync.3g.qq.com”的IP地址



观察到“mpssync.3g.qq.com”对应的IP地址为“113.96.237.110”和“183.3.225.36”。可以得知，为了保证访问流量的均衡，采取了为DNS服务器配置一个IP多个A条目的方式。

## 同步报文的源、目的信息

由于“mpssync.3g.qq.com”所对应的地址即为QQ同步助手同步数据所用的服务器地址，因此筛选IP地址为“113.96.237.110”的报文，得到如下结果：

No.	Time	Source	Destination	Protcl	Len	Info
79107	14.961494	10.12.168.254	113.96.237.110	TCP	66	53965 → 14000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
79108	14.992643	113.96.237.110	10.12.168.254	TCP	66	14000 → 53965 [SYN, ACK] Seq=0 Ack=1 Win=13600 Len=0 MSS=1360 SACK_PERM=1 WS=1024
79109	14.992772	10.12.168.254	113.96.237.110	TCP	54	53965 → 14000 [ACK] Seq=1 Ack=1 Win=131840 Len=0
79110	15.006169	10.12.168.254	113.96.237.110	TCP	294	53965 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=131840 Len=240
79114	15.035989	113.96.237.110	10.12.168.254	TCP	60	14000 → 53965 [ACK] Seq=1 Ack=241 Win=15360 Len=0
79119	15.074839	113.96.237.110	10.12.168.254	TCP	310	14000 → 53965 [PSH, ACK] Seq=1 Ack=241 Win=15360 Len=256
79120	15.115027	10.12.168.254	113.96.237.110	TCP	54	53965 → 14000 [ACK] Seq=241 Ack=257 Win=131584 Len=0

观察到，同步报文的源、目的信息如下：

- 源：10.12.168.254:53965
- 目的：113.96.237.110:14000

## 分析同步流量数据特征

对该同步数据流进行跟踪，得到对应的流信息，以十六进制查看得到如下结果：



根据前四个字节所反映的数据大小，观察到收发报文的长度分别为 0xfc 和 0xec，均为8的倍数，因此猜测该加密方式为分组加密方式。

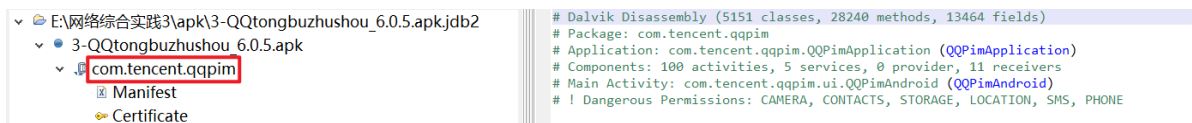
## 任务2：函数调用栈动态跟踪

借助AndroidKiller分析和DDMS的“Method Profiling”功能，分析“QQ手机助手”登录和同步时的函数调用栈，回答以下问题：

- (1) 程序的包名是什么？
- (2) 在DDMS中对“QQ手机助手”进行动态跟踪时，PC端与手机（或模拟器）端连接的端口是多少，并参考实验手册中6.2节中的示例，附上截图。
- (3) trace文件中，列举若干涉及加解密函数的调用栈（附上截图）。
- (4) 上传trace文件

## 程序的包名

使用 JEB Pro 对QQ同步助手的 apk 包进行反编译，得到如下结果：



观察到，该程序的包名为“com.tencent.qqim”。

## 使用DDMS跟踪并查看连接端口号

首先启动“SDK/tools/monitor.bat”文件，并设置相关参数调整最大保存文件大小，查看DDMS为QQ同步助手分配的端口号，如下：

com.android.defcontainer	3614	8621	
com.tencent.qqim	9899	8612 / 8700	
com.tencent.securedownload.service	10069	8620	

观察到，模拟器上给QQ同步助手分配的可调试端口号为8612，本地监听端口号为8700。（DDMS从8600开始给进程分配调试端口号，并在8700端口监听。）

随后利用DDMS进行动态跟踪，点击“Stop Method Profiling”按钮后登录QQ同步助手并进行同步操作，得到的trace文件被存储在 C:\Users\lyg\AppData\Local\Temp 文件夹下。

## 列举涉及加解密函数的调用栈

利用“SDK/tools/traceview.bat”分析获得的trace文件，报错如下：

```
-Djava.ext.dirs=lib\x86_64;lib is not supported. Use -classpath instead.  
Error: Could not create the Java Virtual Machine.  
Error: A fatal exception has occurred. Program will exit.
```

这是由于JDK12不支持“-Djava.ext.dirs”，希望在不进行JDK版本回退的情况下解决问题，故将“traceview.bat”文件中的“-Djava.ext.dirs”改为建议的“-classpath”，仍会报错如下：

```
Unrecognized option: -classpath=lib\x86_64;lib  
Error: Could not create the Java Virtual Machine.  
Error: A fatal exception has occurred. Program will exit.
```

查看JDK12的命令后将其更改为“--class-path”后解决lib错误问题，但出现如下新问题：

```
错误: 无法初始化主类 com.android.traceview.MainWindow  
原因: java.lang.NoClassDefFoundError: org/eclipse/swt/widgets/Control
```

因此同时考虑到“SDK/tools/monitor.bat”查看trace文件时无法使用查找，最终还是选择了将JDK版本回退至JDK8的方式，回退后再次使用traceview查看trace文件，利用查找对包含关键词 `encrypt` 和 `decrypt` 的调用进行检索。结果如下：

1894 com.tencent.tccsync.TccTeaEncryptDecrypt.encrypt ((B)[B])  
Parents  
2075 com.tencent.wscl.wslib.platform.e.a ((B)[B])  
2178 com.tencent.wscl.wslib.platform.e.a (Ljava/lang/String;)B

2590 com.tencent.tccsync.TccTeaEncryptDecrypt.tccXXTeaDecrypt ((B)[B])  
Parents  
2527 com.tencent.wscl.wslib.platform.e.c ((B)[B])B

3913 com.tencent.tccsync.TccTeaEncryptDecrypt.getXXTccTeaEncryptDecryptKey ()B  
Parents  
3887 com.tencent.wscl.wslib.platform.e.a ()B

5291 com.android.org.conscrypt.NativeCrypto.RSA\_public\_encrypt ((B)[B]com/android/org/conscrypt/NativeRef\$EVP\_PKEY;I)I  
Parents  
4732 com.android.org.conscrypt.OpenSSLCipherRSA.engineDoFinal ((B)I)B

215 oicq.wlogin\_sdk.tools.RSACrypt.DecryptData ((B)[B])B  
Parents  
207 oicq.wlogin\_sdk.request.e.a (Landroid/content/Intent;)Loicq/wlogin\_sdk/request/WUserSigInfo;  
Children  
self  
220 oicq.wlogin\_sdk.tools.RSACrypt.decryptdata ((B)[B])B  
738 oicq.wlogin\_sdk.tools.j.z (Landroid/content/Context;)B

220 oicq.wlogin\_sdk.tools.RSACrypt.decryptdata ((B)[B])B  
Parents  
215 oicq.wlogin\_sdk.tools.RSACrypt.DecryptData ((B)[B])B  
Children  
self  
(context switch)

3954 com.tencent.tccsync.TccTeaEncryptDecrypt.decrypt ((B)[B])  
Parents  
3832 com.tencent.wscl.wslib.platform.e.b ((B)[B])

5618 com.tencent.tccsync.TccTeaEncryptDecrypt.tccXXTeaEncrypt ((B)[B])B  
Parents  
5324 com.tencent.wscl.wslib.platform.e.b ((B)[B])B

以上即为所有包含 `encrypt` 或 `decrypt` 字段的调用栈情况。

## 任务3：函数参数动态跟踪

安装Frida工具，编写Python脚本，基于动态跟踪结果，回答以下问题：

(1) 基于DDMS动态跟踪结果，列举同步过程中，调用了哪些加解密函数，并分析函数参数列表中的参数含义，比如，明文、密文和密钥。

(2) 针对每一加解密函数，编写Python脚本，跟踪函数的输入和输出，分析可能采用的密钥、明文或密文数组。

(3) 在跟踪函数参数时，同时利用Wireshark抓取同步时的报文，将报文与跟踪的函数参数（密文）进行比对。

(4) 上传相关脚本和跟踪结果文件。

## frida安装配置

安装frida并查看frida版本和模拟器信息，命令如下：

```
pip install frida frida-tools # 安装
frida --version # 查看frida版本
# 查看模拟器版本
adb shell
getprop ro.product.cpu.abi # 进入shell后执行该命令
```

```
E:\网络综合实践3\apk>frida --version
14.2.18
```

```
E:\网络综合实践3\apk>adb shell
aosp:/ # getprop ro.product.cpu.abi
x86
aosp:/ #
```

随后在<https://github.com/frida/frida/releases>下载对应版本的 frida-server-14.2.18-android-x86 将其上传至模拟器的“/data/local/tmp”目录下并赋予执行权，如下：

```
E:\网络综合实践3\apk>adb push frida-server-14.2.18-android-x86 /data/local/tmp
frida-server-14.2.18-android-x86: 1 file pushed, 0 skipped. 8.3 MB/s (42958488 bytes in 4.927s)
```

```
E:\网络综合实践3\apk>adb shell
aosp:/ # cd /data/local/tmp
aosp:/data/local/tmp # chmod 777 frida-server-14.2.18-android-x86
aosp:/data/local/tmp #
```

## 动态跟踪、Wireshark抓包结果

【需要特别说明的是，一下数据是同一次登录所抓取的对应的加解密函数的信息】

编写python程序如下，代码具体作用已在注释中说明：

```
import frida
import sys

# 获取远程调试设备
device = frida.get_usb_device()
print(device)
print(device.get_frontmost_application())
# attach传入进程名称
session = device.attach("com.tencent.qqvim")

# hook逻辑脚本
jrcode = """
Java.perform(function() {
    var tted = Java.use("com.tencent.tccsync.TccTeaEncryptDecrypt");
    tted.tccXXTeaDecrypt.implementation = function(arg1, arg2) {
        send("Hook start ...");
        send("p1: ");
        send(arg1);
        send("p2: ");
```

```

        var ss = [];
        for (var i=0; i<arg2.length; i++) {
            ss.push(String.fromCharCode(arg2[i]));
        }
        send(ss.toString());
        var rtn = this.tccXXTeaDecrypt(arg1, arg2);
        send("rtn: ");
        send(rtn);
        return rtn;
    };
});
"""

```

# 接收脚本的回调函数，一般为固定形式

```
def on_message(message, data):
```

```
    print(message)
```

# 注入进程

```
script = session.create_script(jrcode)
```

# 设置注入脚本的回调函数

```
script.on('message', on_message)
```

# 加载hook脚本

```
script.load()
```

# 保持主线程不结束

```
sys.stdin.read()
```

随后运行该脚本得到具体结果见上传文件 refDecrypt.txt 中，部分结果如下图：

```

E:\网络综合实践3>python tccXXTeaDecrypt.py
Device(id="emulator-5554", name="Android Emulator 5554", type='usb')
Application(identifier="com.tencent.qqim", name="QQ同步助手", pid=13260)
{'type': 'send', 'payload': 'Hook start ...'}
{'type': 'send', 'payload': 'p1: '}
{'type': 'send', 'payload': '[-66, -88, -45, -27, -69, 20, -89, 27, -58, -20, -50, 31, -28, 57, 20, -90, -40, -34, 120, 88]'}
{'type': 'send', 'payload': 'p2: '}
{'type': 'send', 'payload': '3,1,3,6,8,0,8,2,9,0,&,C,0,M,N,: ,3,5,2,7,4,6,0,2,3,4,3,0,6,0,8,&,2,7,7,6,5,1,7,4,0,&,1,6,2,1,9,4,1,2,6,8'}
}
{'type': 'send', 'payload': 'rtn: '}
{'type': 'send', 'payload': '[8, 114, 73, 29, 124, -46, 76, -104, -46, -80, 68, 70, -110, 22, 5, 4]'}
{'type': 'send', 'payload': 'Hook start ...'}
{'type': 'send', 'payload': 'p1: '}
{'type': 'send', 'payload': '[-91, 27, 34, -53, -19, -102, 21, -3, -41, 74, -16, 57, -73, 80, 23, 58, 25, -80, 81, -51, 84, -92, -53, 66, 119, -110, 42, -96, -120, 14, 35, -99, -73, -122, 71, -88, 85, 23, 65, -95, -14, -41, 61, -64, -116, -48, -118, -28, -121, -47, -34, -47, 88, 57, -37, 36, 124, -59, 65, -81, 25, -89, 42, 35, 76, -92, -110, -70, 67, -128, 27, 92, -107, -74, 92, 44]'}
{'type': 'send', 'payload': 'p2: '}
{'type': 'send', 'payload': '3,1,3,6,8,0,8,2,9,0,&,C,0,M,N,: ,3,5,2,7,4,6,0,2,3,4,3,0,6,0,8,&,2,7,7,6,5,1,7,4,0,&,1,6,2,1,9,4,1,2,6,8'}
}
{'type': 'send', 'payload': 'rtn: '}
{'type': 'send', 'payload': '[69, 65, 69, 116, 65, 65, 65, 119, 47, 98, 113, 104, 77, 118, 109, 48, 48, 88, 73, 108, 84, 67, 57, 53, 100, 114, 121, 73, 108, 106, 110, 82, 55, 111, 81, 75, 69, 76, 98, 55, 99, 66, 76, 106, 86, 82, 55, 100, 117, 119, 74, 52, 118, 87, 87, 79, 89, 119, 50, 52, 76, 109, 84, 75, 77, 55, 79, 76, 121, 119, 84, 77]'}
{'type': 'send', 'payload': 'Hook start ...'}
{'type': 'send', 'payload': 'p1: '}
{'type': 'send', 'payload': '[99, 12, -107, -36, -24, 105, -106, 117, -14, 59, 118, -1, 78, -103, -15, 10, -17, -90, 38, 127]'}
{'type': 'send', 'payload': 'p2: '}
{'type': 'send', 'payload': 'D,F,G,#,$,%,^,#,%, $,R,G,H,R,(,&*,M,<,>,<' }
{'type': 'send', 'payload': 'rtn: '}
{'type': 'send', 'payload': '[120, -100, 99, 100, 88, 44, -58, 0, 0, 2, 31, 0, -69]'}
{'type': 'send', 'payload': 'Hook start ...'}
{'type': 'send', 'payload': 'p1: '}

```

另外，根据任务2中DDMS截获到的数据，对其他函数参数进行截获，例如对上述函数对应的加密函数 `com.tencent.tccsync.TccTeaEncryptDecrypt.tccXXTeaEncrypt()` 进行参数跟踪，其hook部分脚本如下：

```

jrcode = """
Java.perform(function() {
    var tted = Java.use("com.tencent.tccsync.TccTeaEncryptDecrypt");
    tted.tccXXTeaEncrypt.implementation = function(arg1, arg2) {
        send("Hook start ...");
        send("p1: ");
        send(arg1);
        send("p2: ");
        var ss = [];

```

```

        for (var i=0; i<arg2.length; i++) {
            ss.push(String.fromCharCode(arg2[i]));
        }
        send(ss.toString());
        var rtn = this.tccXXTeaEncrypt(arg1, arg2);
        send("rtn: ");
        send(rtn);
        return rtn;
    };
});

```

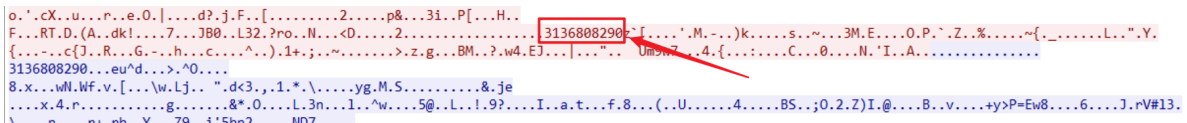
随后运行该脚本得到具体结果见上传文件 `refEncrypt.txt` 中，部分结果如下图：

```

E:\网络综合实践3>python tccXXTeaEncrypt.py
Device(id="emulator-5554", name="Android Emulator 5554", type='usb')
Application(identifier="com.tencent.qqim", name="QQ同步助手", pid=13260)
{'type': 'send', 'payload': 'Hook start ...'}
{'type': 'send', 'payload': 'p1: '}
{'type': 'send', 'payload': '[100, 51, 54, 57, 49, 100, 51, 100, 51, 57, 97, 55, 102, 99, 99, 101, 55, 102, 48, 53, 55, 51, 54, 54, 98, 55, 53, 97, 99, 100, 102, 57, 97, 50, 52, 99, 50, 56, 102, 50, 48, 52, 57, 52, 56, 100, 100, 101, 52, 56, 52, 100, 55, 52, 52, 49, 102, 97, 55, 50, 49, 49, 51]}
{'type': 'send', 'payload': 'p2: '}
{'type': 'send', 'payload': '3,5,2,7,4,6,0,2,3,4,3,0,6,0,8,h,^,j,9,o,`'}
{'type': 'send', 'payload': 'rtn: '}
{'type': 'send', 'payload': '[-27, -20, -93, -40, 34, -16, 97, 9, 86, -72, 86, 29, -56, -125, -80, -25, 96, -89, 9, 71, 26, 35, 2, 11, 120, 15, 26, 49, 126, -65, -83, 117, -28, -73, -44, -56, 40, 94, -84, 80, -27, -70, 127, -70, 98, -100, 124, -70, -13, 42, 112, 124, -63, 3, 11, 83, 60, -22, -109, 85, 12, -106, -59, 24, -31, 81, 96, -99]}
{'type': 'send', 'payload': 'Hook start ...'}
{'type': 'send', 'payload': 'p1: '}
{'type': 'send', 'payload': '[120, -38, 5, -63, -71, 1, -128, 48, 8, 0, -64, -40, 56, -126, -75, 35, -124, 39, 16, 58, 7, 112, 4, 27, -124, -80, -105, -101, 122, -73, 95, 73, 98, -112, -108, 100, -82, 21, -79, -76, -6, 80, 18, 121, 117, 120, 100, -103, 59, 114, -32, 44, -20, 108, 60, 51, 23, 79, 78, 101, -122, 114, 69, 0, 58, -38, -71, 65, -5, -18, -25, 7, 2, -67, 19, -120]}
{'type': 'send', 'payload': 'p2: '}
{'type': 'send', 'payload': 'D,F,G,#,$,%,^,#,%,R,G,H,R,(,&,*M,<,>,<'}
{'type': 'send', 'payload': 'rtn: '}
{'type': 'send', 'payload': '[-60, -36, 73, 6, -115, 123, 121, 34, 28, -8, -30, 16, -6, -120, 120, 50, -51, 55, -72, 6, 75, 104, -96, -39, 62, -10, -111, 17, -35, -62, 81, -111, 11, 63, 39, 63, 11, -36, 37, 102, 111, -26, 31, 47, -11, -71, -106, -2, 58, -49, -28, 36, -119, 65, 14, 26, 83, -74, 10, 120, 95, -43, -36, 12, 122, 14, -29, 1, 61, -3, 9, -88, -30, -44, -50, 34, 91, -1, 10, -48]}
{'type': 'send', 'payload': 'Hook start ...'}
{'type': 'send', 'payload': 'p1: '}
{'type': 'send', 'payload': '[10, 12, 16, 17, 32, 1, 48, 1, 74, 6, 10, 51, 49, 51, 54, 56, 48, 56, 50, 57, 48, 16, 1, 11, 86, 20, 67, 79, 77, 78, 58, 51, 53, 50, 55, 52, 54, 48, 50, 51, 52, 51, 48, 54, 48, 56, 102, 0, 118, 16, 69, 48, 49, 49, 55, 56, 66, 52, 54, 56, 53, 66, 54, 50, 48, 49, -127, 3, -18, -99, 0, 0, 72, 69, 65, 69, 116, 65, 65, 65, 119, 47, 98, 113, 104, 77, 118, 109, 48, 48, 88, 73, 108, 84, 67, 57, 53, 100, 114, 121, 73, 108, 106, 110, 82, 55, 111, 81, 75, 69, 76, 98, 55, 99, 66, 76, 106, 86, 82, 55, 100, 117, 119, 74, 52, 118, 87, 87, 79, 89, 119, 50, 52, 76, 109, 84, 75, 77, 55, 79, 76, 121, 119, 84, 77, -90, 0, 11, 29, 0, 0, 80, -60, -36,

```

同时，对该次登录进行抓包，得到具体结果见上传文件 `加解密函数报文.pcapng` 中，部分信息如下图：



和Decrypt的参数进行比较，观察到Decrypt函数中也出现了该数据，即本人的QQ号，如下图：

```

{'type': 'send', 'payload': '[-66, -88, -45, -27, -69, 20, -89, 27, -58, -20, -50, 31, -28, 57, 20, -90, -40, -34, 120, 88]}
{'type': 'send', 'payload': 'p2: '}
{'type': 'send', 'payload': '3,1,3,6,8,0,8,2,9,0,&C,O,M,N,:3,5,2,7,4,6,0,2,3,4,3,0,6,0,8,&2,7,7,6,5,1,7,4,0,&1,6,2,1,9,4,1,2,6,8'}
{'type': 'send', 'payload': 'rtn: '}

```

该现象证明了Wireshark抓包结果与动态参数获取结果具有相关联系，且结果保持一致。

## 明密文、密钥等信息分析

在解密函数中，针对参数2，一直重复着以下信息：

```

3,5,2,7,4,6,0,2,3,4,3,0,6,0,8,h,^,j,9,o,`
D,F,G,#,,,R,G,H,R,(,&,*M,<,>,<
3,1,3,6,8,0,8,2,9,0,&C,O,M,N,:3,5,2,7,4,6,0,2,3,4,3,0,6,0,8,&2,7,7,6,5,1,7,4,0,&1,6,2,1,9,4,1,2,6,8
8

```

在加密函数中，针对参数2，一直重复着以下信息：

```

3,5,2,7,4,6,0,2,3,4,3,0,6,0,8,h,^,j,9,o,`
D,F,G,#,,,R,G,H,R,(,&,*M,<,>,<

```



因此猜测加密函数中的两个信息，可能与密钥有关。

另外，针对第一个数据（可能的密钥），其得到的返回值结果均为正值且均小于128，因此猜测加密结果可能对128进行了取余操作，如下图：

```
{'type': 'send', 'payload': 'p2: '}  
{'type': 'send', 'payload': '3,5,2,7,4,6,0,2,3,4,3,0,6,0,8,h,^,J,9,o,\''}  
{'type': 'send', 'payload': 'rtn: '}  
{'type': 'send', 'payload': [99, 111, 109, 46, 100, 101, 110, 115, 119, 46, 80, 105, 99, 70, 111, 108, 100, 101, 114, 46, 100, 111, 17, 100, 46, 115, 121, 110, 99, 59, 99, 111, 109, 46, 109, 111, 110, 101, 121, 59, 99, 111, 109, 46, 115, 107, 121, 112, 101, 46, 99, 111, 110, 116, 97, 99, 116, 115, 46, 115, 121, 110, 99, 59, 99, 111, 109, 46, 116, 97, 111, 98, 97, 111, 59, 99, 111, 109, 46, 116, 101, 110, 99, 101, 110, 116, 46, 109, 109, 46, 97, 99, 99, 111, 117, 110, 116, 59, 99, 111, 109, 46, 116, 101, 110, 99, 101, 110, 116, 46, 109, 111, 98, 105, 108, 101, 113, 113, 46, 97, 99, 99, 111, 117, 110, 116, 59, 99, 111, 109, 46, 121, 121, 46, 121, 121, 109, 101, 101, 116, 46, 99, 111, 110, 116, 97, 99, 116, 59, 118, 110, 100, 46, 115, 101, 99, 46, 99, 111, 110, 116, 97, 99, 116, 46, 115, 105, 109, 59]}  
{'type': 'send', 'payload': 'Hook start ...'}  
{'type': 'send', 'payload': 'p1: '}  
{'type': 'send', 'payload': [-96, 89, -11, -67, 15, -83, -119, 36, -67, -86, 53, 21, 56, 72, 28, -94, 44, 121, 122, -128, -110, -42, 93, -118, 98, 21, 47, 61, 67, 4, -70, 126, -107, -11, 126, -20, -21, 1, -64, 38, 7, 67, -125, 56]}  
{'type': 'send', 'payload': 'p2: '}  
{'type': 'send', 'payload': '3,5,2,7,4,6,0,2,3,4,3,0,6,0,8,h,^,J,9,o,\''}  
{'type': 'send', 'payload': 'rtn: '}  
{'type': 'send', 'payload': [32, 59, 32, 59, 32, 59, 32, 59, 32, 59, 32, 59, 32, 59, 112, 114, 105, 109, 97, 114, 121, 46, 115, 105, 109, 46, 97, 99, 111, 117, 110, 116, 95, 110, 97, 109, 101, 59]}  
}
```

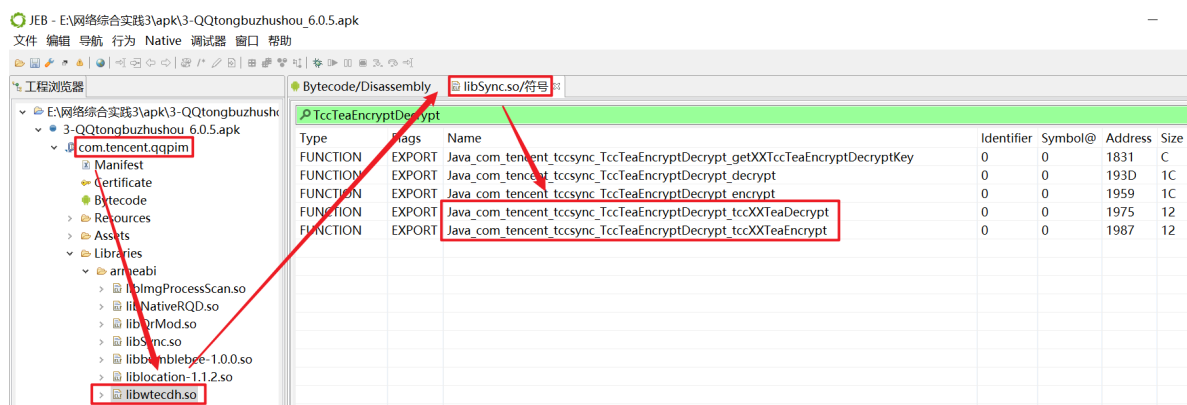
## 任务4：二进制代码分析

借助IDA Pro分析“QQ手机助手”中与同步相关的so文件，回答以下问题：

- （1）与同步操作有关的文件名称；
- （2）比较Java代码（借助AndroidKiller分析）与二进制代码中对应函数参数列表的差异；
- （3）还原与加密密钥相关的二进制代码，以C代码形式呈现；
- （4）还原二进制代码中核心的加解密代码，以C代码形式呈现；
- （5）上传还原的C代码（说明其主要功能）。

## 确定与同步操作有关的文件

利用JEB打开QQ同步助手的apk文件，在“Libraries”、“armeabi”文件夹下遍历每一个so文件进行搜索，最终在“libSync.so”文件中找到加解密有关的函数，如下图：



随后利用JEB抽取该文件。

## Java代码和二进制代码间参数列表差异

利用JEB观察解密函数的二进制代码，如下图：

```

; =====
; ROUTINE: Java_com_tencent_tccsync_TccTeaEncryptDecrypt_tccXXTeaDecrypt
;
; Signature: unsigned int __cdecl Java_com_tencent_tccsync_TccTeaEncryptDecrypt_tccXXTeaDecrypt(unsigned int, unsigned int, unsigned int)
LOAD.text:00001974 Java_com_tencent_tccsync_TccTeaEncryptDecrypt_tccXXTeaDecrypt proc
LOAD.text:00001974
LOAD.text:00001974          PUSH    {R7, LR}
LOAD.text:00001976          ADD     R7, SP, #0
LOAD.text:00001978          ADDS    R1, R2, #0
LOAD.text:0000197A          ADDS    R2, R3, #0
LOAD.text:0000197C          MOVS    R3, #0
LOAD.text:0000197E          BL      sub_183C
LOAD.text:00001982          MOV     SP, R7
LOAD.text:00001984          POP     {R7, PC}
LOAD.text:00001984 Java_com_tencent_tccsync_TccTeaEncryptDecrypt_tccXXTeaDecrypt endp
; -----

```

观察到其通过 BL sub\_183C 指令跳转至sub\_183C()函数执行相关代码，该函数的Java代码如下图所示：

```

void* sub_183C(void* param0, void* param1, void* param2, void* param3, unsigned int par00) {
    void* ptr0 = param3;
    param3 = (*param0 + 736);
    void* ptr1 = param1, ptr2 = param2, ptr3 = param0;
    param0 = {((unsigned char)(((int)param3)))}(param0, param2, 0, param3, ptr7);
    param3 = *ptr3;
    void* result = param0;
    param3 = *((unsigned int)(((int)gvar_2AC) + ((int)param3)));
    param0 = {((unsigned char)(((int)param3)))}(ptr3, param2, param2, param3, ptr7);
    param3 = *ptr3;
    void* ptr4 = param0;
    param3 = *(param3 + 736);
    param0 = {((unsigned char)(((int)param3)))}(ptr3, param1, 0, param3, ptr7);
    param3 = *ptr3;
    void* ptr5 = param0;
    param3 = *((unsigned int)(((int)gvar_2AC) + ((int)param3)));
    param0 = {((unsigned char)(((int)param3)))}(ptr3, param1, param2, param3, ptr7);
    param2 = ptr0;
    void* ptr6 = param0;
    size_t __byte_count = param0 + 8;

    if(((unsigned char)(((int)ptr0) != 0))) {
        __byte_count = ((__byte_count - 5) & -4) + 12;
    }

    param0 = _ptr_malloc(__byte_count);
    param3 = ptr0;
    void* __ptr = param0;
    ptr7 = param0;
    size_t v0 = __byte_count;
    param0 = ptr5;
    param1 = ptr6;
    param2 = result;
    param0 = !((unsigned char)(((int)ptr0) != 0)) ? sub_C49C(param0, ptr6, result, ptr4, ptr7, __byte_count): sub_C62C(param0, ptr6, result, ptr4, ptr7, __byte_count);
    __byte_count = (*ptr3 + &gvar_300);
    ptr6 = param0;
    __byte_count}(ptr3, ptr2, result, 2, ptr7);
    __byte_count = (*ptr3 + &gvar_300);
    __byte_count}(ptr3, ptr1, ptr5, 2, ptr7);
}

```

综上，对二进制代码和Java代码的参数列表进行比较，观察到从二进制代码中由于没有对栈的直接操作，因此无法直接分析出参数个数。而根据Java代码可以判断出该函数有五个参数，实际上是4个参数，第5个参数没有在代码中使用。

## 还原与加解密相关的二进制代码

使用IDA打开抽取“libSync.so”文件，找到加密函数并使用F5插件获得其对应的C语言代码如下图：

```

1 int __fastcall sub_183C(int a1, int a2, int a3, int a4)
2 {
3     int v4; // r5
4     int v5; // r0
5     int v6; // r4
6     void *v7; // r6
7     int v8; // r0
8     void *dest; // [sp+0h] [bp-8h]
9     int v11; // [sp+4h] [bp-4h]
10    int v12; // [sp+Ch] [bp+4h]
11    int v13; // [sp+10h] [bp+8h]
12    int v14; // [sp+14h] [bp+Ch]
13    int v15; // [sp+18h] [bp+10h]
14    void *src; // [sp+1Ch] [bp+14h]
15    int v17; // [sp+20h] [bp+18h]
16    int v18; // [sp+20h] [bp+18h]
17    int v19; // [sp+24h] [bp+1Ch]
18    int v20; // [sp+24h] [bp+1Ch]
19
20    v12 = a4;
21    v15 = a2;
22    v14 = a3;
23    v4 = a1;
24    v17 = (*int (**)(void))(*(_DWORD *)a1 + 736)();
25    v13 = (*int (__fastcall **)(int, int))(*(_DWORD *)v4 + 684)(v4, v14);
26    src = (void *)(*int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)v4 + 736)(v4, v15, 0);
27    v5 = (*int (__fastcall **)(int, int))(*(_DWORD *)v4 + 684)(v4, v15);
28    v19 = v5;
29    v6 = v5 + 8;
30    if ( v12 )

```

观察到代码中存在大量的函数指针，为了使这些函数具象化，需要修改这些函数都使用到的 a1 值（或 v4值，其中v4=a1）。在变量 a1 处，点击右键选择 Set lvar type，将变量设置为 JNIEnv \*，随后得到相关函数调用如下图：



```
IDA View-A Pseudocode-A Hex View-1 Structures Enums Imports Exports
24 v17 = ((int (*)(void))(*v4)->GetByteArrayElements)();
25 v13 = ((int (__fastcall *)(JNIEnv *, int))(*v4)->GetArrayLength)(v4, v14);
26 src = (void *)((int (__fastcall *)(JNIEnv *, int, _DWORD))(*v4)->GetByteArrayElements)(v4, v15, 0);
27 v5 = ((int (__fastcall *)(JNIEnv *, int))(*v4)->GetArrayLength)(v4, v15);
28 v19 = v5;
29 v6 = v5 + 8;
30 if ( v12 )
31     v6 = ((v5 + 3) & 0xFFFFFFFF) + 12;
32 v7 = malloc(v6);
33 dest = v7;
34 v11 = v6;
35 if ( v12 )
36     v8 = sub_C62C(src, v19, v17, v13, v7, v6);
37 else
38     v8 = sub_C49C(src, v19);
39 v20 = v8;
40 ((void (__fastcall *)(JNIEnv *, int, int, signed int, void *, int))(*v4)->ReleaseByteArrayElements)(
41     v4,
42     v14,
43     v17,
44     2,
45     dest,
46     v11);
47 ((void (__fastcall *)(JNIEnv *, int, void *, signed int))(*v4)->ReleaseByteArrayElements)(v4, v15, src, 2);
48 if ( v20 >= 0 )
49 {
50     v18 = ((int (__fastcall *)(JNIEnv *, int))(*v4)->NewByteArray)(v4, v20);
51     if ( ((int (__fastcall *)(JNIEnv *))(*v4)->ExceptionOccurred)(v4) )
52     {
53         free(v7);
54     }
55 }
```

观察到代码中此时出现了两个关键函数 `sub_C62C()` 和函数 `sub_C49C()`，对这两个函数分别进行跟踪，观察到两者的核心函数分别为 `sub_C4D8()` 函数和 `sub_C340()` 函数，如下图：

```
1 int __fastcall sub_C49C(void *a1, int a2, int a3, int a4, char *dest, int a6)
2 {
3     int result; // r0
4     int v7; // r3
5
6     result = sub_C340(a1, a2, a3, a4, dest, a6);
7     if ( result > 0 && dest )
8     {
9         v7 = *((_DWORD *)&dest[result - 4]);
10        if ( v7 < 0 || result - 3 <= v7 )
11        {
12            result = -32227;
13        }
14        else
15        {
16            dest[v7] = 0;
17            result = v7;
18        }
19    }
20    return result;
21 }
```

```
1 int __fastcall sub_C62C(void *src, int a2, int a3, int a4, void *dest, int a6)
2 {
3     int v6; // r6
4     int v7; // r5
5     int i; // r1
6     int v10; // [sp+8h] [bp+0h]
7     int v11; // [sp+Ch] [bp+4h]
8
9     v10 = a4;
10    v6 = a2;
11    v11 = a3;
12    v7 = ((a2 + 3) & 0xFFFFFFFF) + 4;
13    if ( dest )
14    {
15        if ( a6 < v7 )
16        {
17            v7 = -1;
18        }
19        else
20        {
21            if ( dest != src )
22            {
23                memcpy(dest, src, a2);
24                for ( i = v6; i < v7; ++i )
25                    *((_BYTE *)dest + i) = 0;
26                *((_DWORD *)dest + (i >> 2) - 1) = v6;
27                v7 = sub_C4D8(dest, i, v11, v10, dest, i);
28            }
29        }
30    }
```

## 分析与加解密相关的核心代码

观察到两者均使用了XXTEA算法来进行加解密，同时由于需要考虑线程等应用效率问题，在代码中加入了部分与加解密无关的代码部分。以下是XXTEA算法的代码，用来作为参考：

```
#include <stdio.h>
#include <stdint.h>
#define DELTA 0x9e3779b9
#define MX (((z>>5^y<<2) + (y>>3^z<<4)) ^ ((sum^y) + (key[(p&3)^e] ^ z)))

void btea(uint32_t *v, int n, uint32_t const key[4]) {
    uint32_t y, z, sum;
    unsigned p, rounds, e;
    if (n > 1) { // Coding Part
        rounds = 6 + 52/n;
        sum = 0;
        z = v[n-1];
        do {
            sum += DELTA;
            e = (sum >> 2) & 3;
```

```

        for (p=0; p<n-1; p++) {
            y = v[p+1];
            z = v[p] += MX;
        }
        y = v[0];
        z = v[n-1] += MX;
    } while (--rounds);
}
else if (n < -1) { // Decoding Part
    n = -n;
    rounds = 6 + 52/n;
    sum = rounds * DELTA;
    y = v[0];
    do {
        e = (sum >> 2) & 3;
        for (p=n-1; p>0; p--) {
            z = v[p-1];
            y = v[p] -= MX;
        }
        z = v[n-1];
        y = v[0] -= MX;
        sum -= DELTA;
    } while (--rounds);
}
}

int main() {
    uint32_t v[2]= {1,2}; // v为要加密的数据是两个32位无符号整数
    uint32_t const k[4]= {2,2,3,4}; // k为加密解密密钥，为4个32位无符号整数，即密钥长度为128位
    int n= 2; //n的绝对值表示v的长度，取正表示加密，取负表示解密

    btea(v, n, k);
    return 0;
}

```

## sub\_C4D8()函数

通过IDA结合JEB得到如下还原结果，对每一部分的解释已在注释中说明：

```

void* sub_C4D8(void* param0, void* param1, void* i, void* param3, void* par00,
void* par04) {
    void* result, ptr0 = param0, ptr1 = param1, ptr2 = i, ptr3 = param3, ptr4 =
par00;

    if((((unsigned char)((((int)param1) <= 0)) || ((unsigned char)((((int)param0)
== 0)))) {
        result = 0;
    }
    else {
        result = (param1 + 3) & -4;

        if(!((unsigned char)((((int)par00) == 0)))) {

            if(!((unsigned char)((((int)result) > ((int)par04)))) {

                if((((unsigned char)((((int)param0) != ((int)par00)))) {

```

```

        _ptr_memcpy(ptr4, ptr0, ptr1);
    }

    for(i = ptr1; !((unsigned char)(((int)i) >= ((int)result)));
++i) {

        *((unsigned int*)((int)i) + ((int)ptr4))) = 0;
    }

    ptr1 = i;
    sub_c2c4(ptr2, ptr3, &v2);
    param3 = *ptr4;
    param1 = (void*)((int)ptr1 >> 2);
    i = param1 - 1;
    ptr0 = (void*)((unsigned int)(((int)i) * 4)) + ((int)ptr4));
    ptr2 = i;
    result = *ptr0;

    if(!((unsigned char)(((int)i) > 0))) {
        param1 = gvar_C624;
        i = &v2;

        do {
            ptr0 = *i;
            i += 4;
            param3 = ((unsigned int)(((unsigned int)
(__ror__(((int)param3), 16))) ^ ((int)param1))) + ((unsigned int)(((int)param1)
^ ((int)ptr0)));
        }
        while(!((unsigned char)(((int)i) == ((int)(&v4)))));

        *ptr4 = param3;
    }
    else {
        param0 = sub_144C0(52, param1, i, param3, v3);
        ptr3 = param0;
        param3 = 0;
        unsigned int v0 = (unsigned int)(((int)i) & 3);

        // XXTEA算法核心部分
        while(ptr3 + 12 != 0) {
            param1 = ptr4;
            param3 = (void*)(gvar_C624 + ((int)param3));
            unsigned int v1 = ((unsigned int)(((int)param3) >> 2)) &
3;

            i = 0;
            void* ptr5 = param3;

            do {
                param0 = *((unsigned int)((((unsigned int)(((int)i)
& 3)) ^ v1) * 4 + ((int)(&v2))));
                ++i;
                result = ((((*param1 + 4) * 4) ^ ((unsigned int)
(((int)result) >> 5))) + (((unsigned int)(((int)result) * 16)) ^ ((unsigned int)
(((int)(*param1 + 4)) >> 3)))) ^ (((unsigned int)(((int)(*param1 + 4)) ^
((int)ptr5))) + ((unsigned int)(((int)param0) ^ ((int)result)))) + *param1;
                param0 = ptr2;
                *param1 = result;
                param1 += 4;
            }
        }
    }
}

```

```

    }
    while(!((unsigned char)(((int)param0) <= ((int)i))));

    i = *ptr4;
    param3 = ptr5;
    result = ((unsigned int)(((int)(((unsigned int)
(((int)i) * 4)) ^ ((unsigned int)(((int)result) >> 5))) + (((unsigned int)
(((int)result) * 16)) ^ ((unsigned int)(((int)i) >> 3)))) ^ (((unsigned int)(*
((unsigned int)((v0 ^ v1) * 4 + ((int>(&v2)))) ^ ((int)result))) + ((unsigned
int)(((int)i) ^ ((int)ptr5)))))) + *ptr0;
    param1 = ptr3;
    *ptr0 = result;
    ptr3 = param1 - 1;
}
}

    result = ptr1;
}
else {
    result = gvar_C628;
}
}
}

return result;
}

```

## sub\_C304()函数

通过IDA结合JEB得到如下还原结果，对每一部分的解释已在注释中说明：

```

void* sub_C340(void* param0, void* param1, void* param2, void* param3, void*
par00, void* par04) {
    unsigned char v0;
    void* ptr0, ptr1 = param0, result = param1, ptr2 = param2, ptr3 = param3,
ptr4 = par00;

    if(((unsigned char)(((int)param1) <= 0)) || ((unsigned char)(((int)param0)
== 0))) {
        return 0;
    }
    else if(((unsigned int)(((int)param1) << 30)) != 0) {
        return 0xFFFF821D;
    }
    else if((unsigned char)(((int)par00) != 0)) {
        param3 = par04;

        if(((unsigned char)(((int)param1) > ((int)par04)))) {
            return 0xFFFF821D;
        }
        else {

            if(((unsigned char)(((int)param0) != ((int)par00)))) {
                _ptr_memcpy(ptr4, ptr1, result);
            }

            ptr1 = (void*)((int)result) >> 2;

```

```

sub_C2C4(ptr2, ptr3, &v4);
ptr2 = ptr1;
ptr3 = ptr1 - 1;
param0 = ptr1 - 1;
ptr1 = *ptr4;

if(!((unsigned char)((((int)param0) > 0))) {
    param3 = 3;
    ptr0 = 0x79B99E37;

    do {
        ptr1 = (void*)((__ror__(((unsigned int)((((int)ptr1) -
((unsigned int)(*((unsigned int*)((unsigned int)((((int)param3) * 4)) + ((int)
(&v4)))) ^ ((int)0x9E3779B9))))), 16)) ^ ((int)ptr0));
        v0 = __carry__(param3 + 1, -2);
        --param3;
    }
    while(v0 != 0);

    *ptr4 = ptr1;
}
else {
    param0 = sub_144C0(52, ptr2, &v4, param3, v5);
    void* ptr5 = param0;
    param3 = (void*)((param0 + 6) * ((int)0x9E3779B9));
    param0 = ptr2;
    ptr0 = result;
    param1 = 0x3FFFFFFF; // 用0x3FFFFFFF表示无穷大数，该数本身满足无穷大，
且满足加上任意数仍为无穷大的特征
    unsigned int v1 = (unsigned int)((((unsigned int)((((int)ptr3) *
4)) + ((int)ptr4)));
    unsigned int v2 = (((unsigned int)((((int)ptr2) +
((int)0x3FFFFFFF))) * 4;

    while(ptr5 + 12 != 0) {
        int v3 = (((unsigned int)((((int)param3) >> 2)) & 3;
        param2 = (unsigned int)((((int)ptr4) + v2);
        ptr2 = ptr3;

        // 以下部分是XXTEA算法核心部分
        do {
            param1 = param2 - 4;
            param0 = (((*param1 * 16) ^ (((unsigned int)((((int)ptr1)
>> 3))) + (((unsigned int)((((int)ptr1) * 4)) ^ (*param1 >> 5))) ^ ((*((unsigned
int*)((((unsigned int)((((int)ptr2) & 3)) ^ v3) * 4 + ((int)(&v4)))) ^ *param1) +
((unsigned int)((((int)param3) ^ ((int)ptr1)))));
            ptr1 = *param2;
            result = ptr2 - 1;
            --ptr2;
            ptr1 = (void*)((((int)ptr1) - ((int)param0)));
            *param2 = ptr1;
            param2 = param1;
        }
        while(!((unsigned char)((((int)result) == 0)));

        // 根据魔数0x61C8864判断出一下应该是ThreadLocal中的
        param2 = *v1;

```

```

        ptr1 = *ptr4 - (((((unsigned int)(((int)param2) * 16)) ^
((unsigned int)(((int)ptr1) >> 3))) + (((unsigned int)(((int)ptr1) * 4)) ^
((unsigned int)(((int)param2) >> 5)))) ^ (((unsigned int)(*((unsigned int*)(v3 *
4 + ((int>(&v4)))) ^ ((int)param2))) + ((unsigned int)(((int)param3) ^
((int)ptr1))))));
        param0 = 0x61c88647;
        param1 = ptr5 - 1;
        *ptr4 = ptr1;
        param3 += 0x61c88647;
        ptr5 = param1;
    }

    return ptr0;
}
}
}
}

```

## 任务5：报文还原测试（选做）

根据抓取的报文，跟踪的密钥和还原代码，测试对跟踪和分析结果的正确性。上传分析过程和结果。