dart

设计的时候确实没有考虑到大家的运行环境可能并不满足要求，无形之中增加了难度。

ThTsOd大佬的题解利用CE+IDA动态调试的方式简单快捷优雅地解出了题目

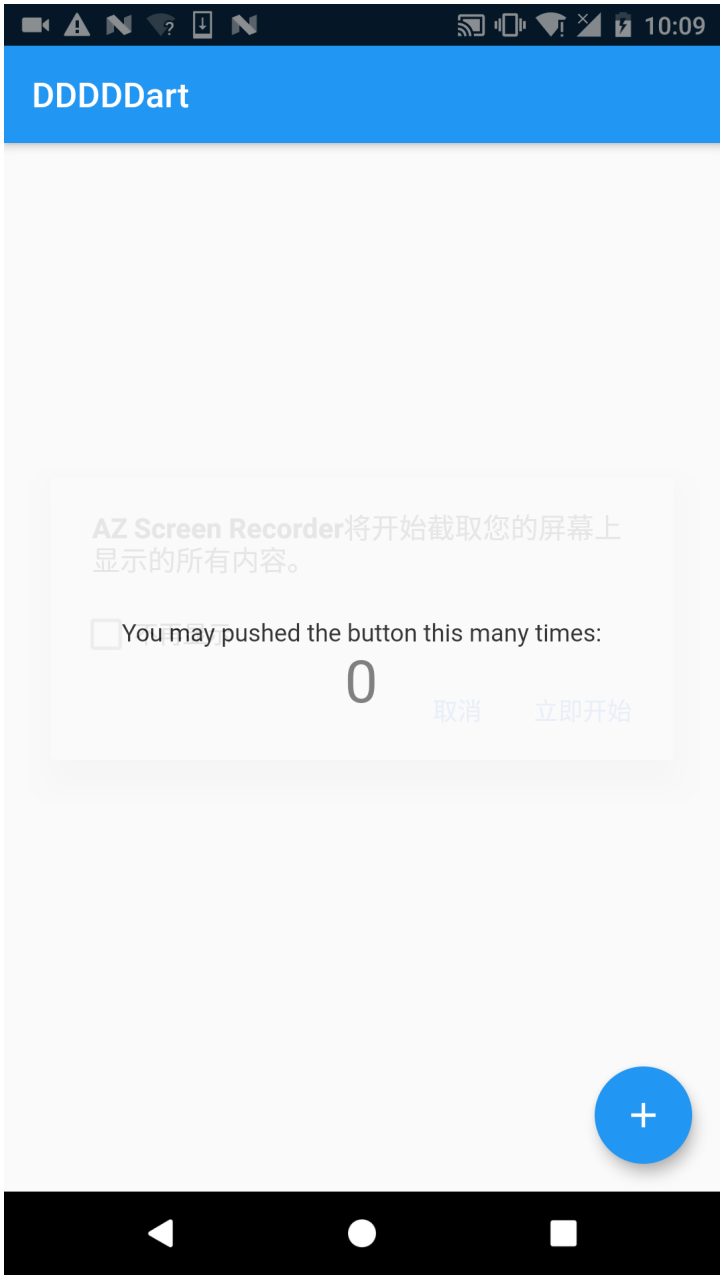
本题难度不大，解法很多，这里仅展示一种，欢迎大家一起讨论交流

Step 1 Init

安装软件

```
minSdkVersion 25
ndk arm64-v8a
```

有些选手反应这个配置加高了解题门槛，实在抱歉



> 这里的app和flutter sample程序很像，可以build一个作为参考（事实上就是除了核心的算法没有区别）

没有输入框，唯一按键可以增加数值大小，该按键可以作为切入口

Step 2 HINT

利用`reflutter`可以观察被加载的dart代码，搜索`main.dart`，可以观察到可能的关键代码

> 附件release.RE-aligned-debugSigned.apk

```
:       }
: reflutter:
:   Library:'package:whats_todays_date/main.dart' Class: RawKeyEventDataWindow extends StatefulWidget {
:
: Function 'createState':. null {
:
:                 }
:
:       }
```

`libflutter.so`查看版本`2.12.0`

```
.rodata:00000000000F8BB8                                        ; DATA XREF: .data.rel.ro:0000000000
.rodata:00000000000F8BD3 a2120StableThuF DCB "2.12.0 (stable) (Thu Feb 25 19:50:53 2021 +0100)",0
```

满足使用`Doldrums`

HINT 1：

rscloura/Doldrums: A Flutter/Dart reverse engineering tool

安装工具并使用，生成导出类及其绝对代码偏移量

```
28324   class :: {
28325   }
28326
28327   class :: {
28328   }
28329
28330   class _RawKeyEventDataWindow@563477602 extends State {
28331
28332       Null RawKeyIncrease() {
28333           Code at absolute offset: 0xecc84
28334       }
28335
28336       Null __RawKeyEventDataMisbehave@563477602() {
28337           Code at absolute offset: 0xecd00
28338       }
28339
28340       Null __RawKeyEventDataAlgorithm@563477602() {
28341           Code at absolute offset: 0xec4a4
28342       }
28343
28344       Null __RawKeyEventDataCounter@563477602() {
28345           Code at absolute offset: 0xeb0c0
28346       }
28347
28348       Null build() {
28349           Code at absolute offset: 0xead7c
28350       }
28351
28352       Null get:__RawKeyEventDataCounter@563477602() {
28353           Code at absolute offset: 0x1253c
28354       }
28355   }
28356
28357   class RawKeyEventDataWindow extends StatefulWidget {
28358
```

借助导出类在IDA中还原基本代码

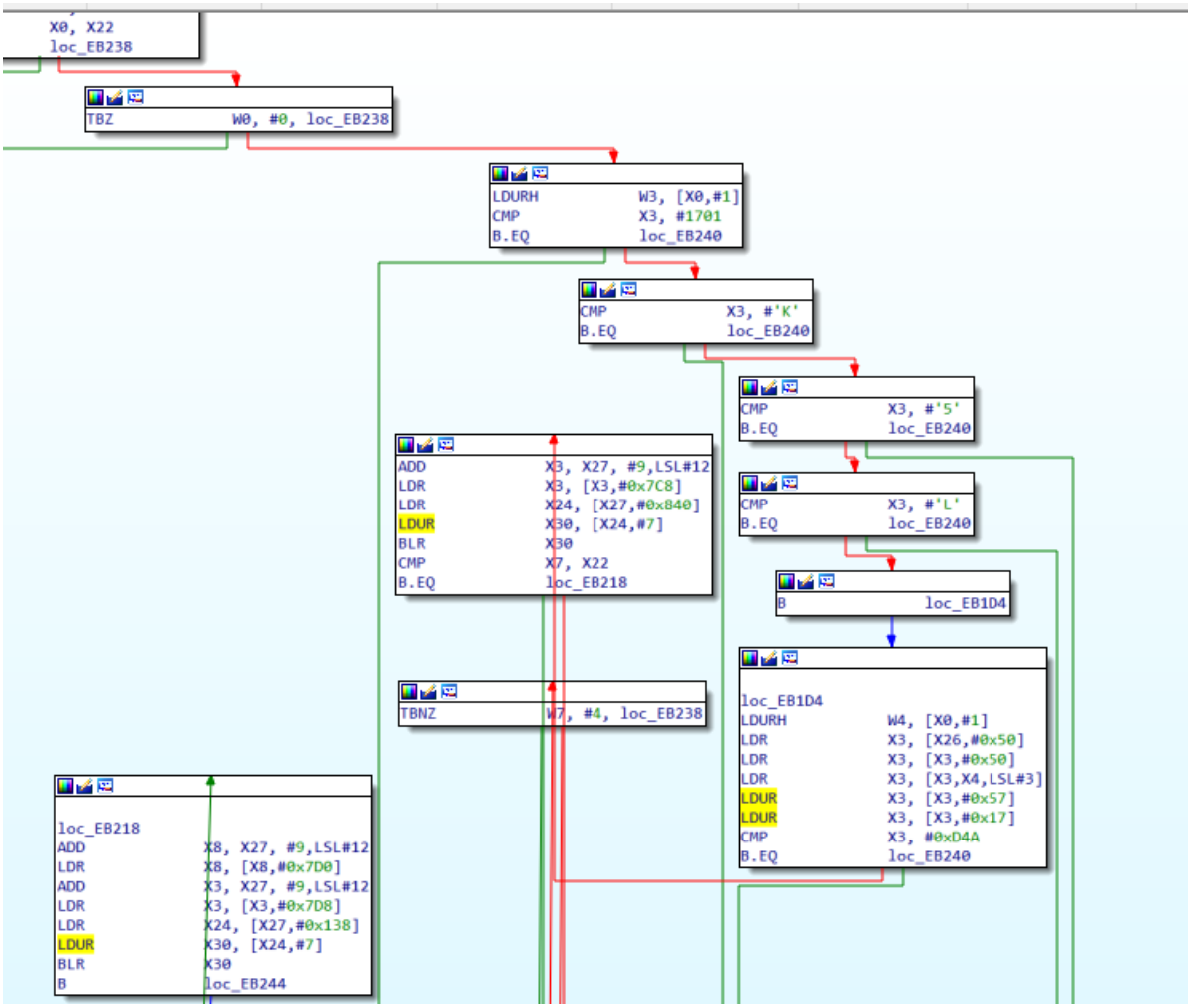
```
  *(_QWORD *)(v9 - 8) = v2;
  v10 = Random_();
  v11 += 8LL;
  *(_QWORD *)(v6 - 144) = v10;
  *(_QWORD *)(v6 - 136) = 4294967294LL;
  *(_QWORD *)(v11 - 16) = 4294967294LL;
  *(_QWORD *)(v11 - 8) = v10;
  v12 = random_nextInt(v10);
  v13 += 16LL;
  *(_QWORD *)(v6 - 152) = v12;
  *(_QWORD *)(v13 - 8) = *(_QWORD *)(v6 - 144);
  *(_QWORD *)(v13 - 16) = *(_QWORD *)(v6 - 136);
  v14 = random_nextInt(v12);
  v15 += 16LL;
  *(_QWORD *)(v6 - 160) = v14;
  *(_QWORD *)(v15 - 8) = *(_QWORD *)(v6 - 144);
  *(_QWORD *)(v15 - 16) = *(_QWORD *)(v6 - 136);
  v16 = random_nextInt(v14);
  v17 += 16LL;
  *(_QWORD *)(v6 - 168) = v16;
  *(_QWORD *)(v17 - 8) = *(_QWORD *)(v6 - 144);
  *(_QWORD *)(v17 - 16) = *(_QWORD *)(v6 - 136);
  *(_QWORD *)(v6 - 136) = random_nextInt(v16);
  *(_QWORD *)(v18 + 8) = *(_QWORD *)(v6 + 16);
  _RawKeyEventDataMisbehave();
}
```

HINT 2 :

dart 异常处理

dart的异常处理和c++还是有些明显的差别的，在函数`__RawKeyEventDataMisbehave`（`rethrow`的使用）与`__RawKeyEventDataCounter`均有涉及

反应到IDA中则是直接看F5的代码大概率是不全的，这里算一个坑点



不过这里代码不多，仅看汇编问题也不会很大

Step 3 Function

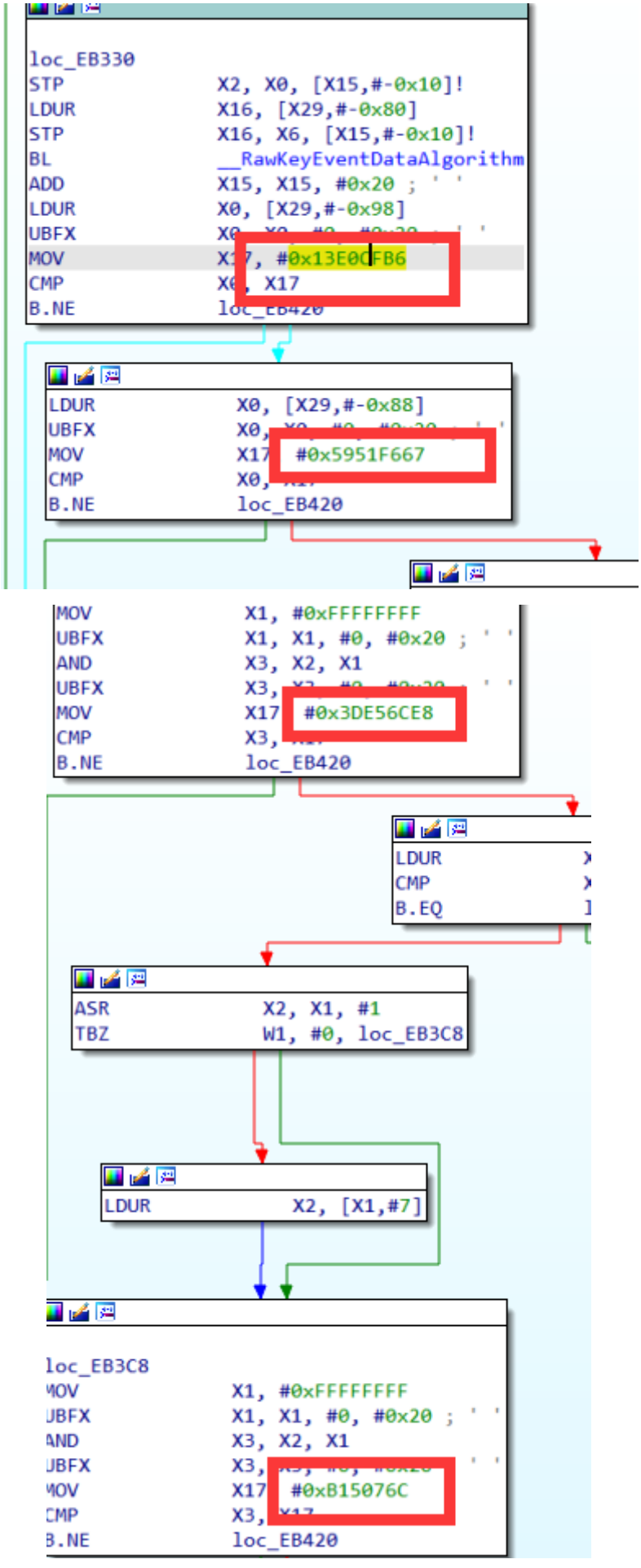
`__RawKeyEventDataCounter`

`__RawKeyEventDataCounter`主要逻辑为：

生成四个随机数，并赋值，进入异常处理逻辑，第一步运行`__RawKeyEventDataMisbehave`

`catch`上一步`throw`的`Exception`并保存后，运行`RawKeyIncrease`，计数后与常量`CMP`,运行两次`__RawKeyEventDataAlgorithm`，每次`__RawKeyEventDataAlgorithm`取出一开始生成的两个随机数，并与`Exception`共同作为函数参数

4个计算后的数值与常量比较



那么很明显我们需要求解的即为那四个随机生成的数值，使其经过计算后满足需求即可

`__RawKeyEventDataMisbehave`

```
  *(_QWORD *)(v17 - 8) = v16;
  v18 = ((__int64 (*)(void))RC4)();
  v19 += 8LL;
  v20 = v11[0xA9];
  *(_QWORD *)(v19 - 16) = v11[0x133E];
  *(_QWORD *)(v19 - 8) = v20;
  v21 = ((__int64 (__fastcall *)(__int64))encode)(v18);
  v22 += 16LL;
  v23 = *(_QWORD *)(v14 - 72);
  *(_QWORD *)(v22 - 16) = v21;
  *(_QWORD *)(v22 - 8) = v23;
  v24 = ((__int64 (*)(void))encodeBytes)();
  a1 = sub_182EF4(v24);
  __break(0);
LABEL_3:
  sub_184764(a1, a2, a3, a4, a5, a6, a7, a8, a9);
}
```

这里发现 `RC4` 算法，最后结果encode为Bytes数组

> 这里的RC4算法我修改了一个步骤，直接用Strings出来的KEY与明文无法获得正确的结果

`RawKeyIncrease`

没有异常处理

这个函数逻辑其实非常简单，计数器-1，后更新Window中显示的数值状态并计数器+1，build flutter sample会更方便理解这里的调用

最后返回值固定为1，在 `__RawKeyEventDataCounter` 中计数器+1

`__RawKeyEventDataAlgorithm`

这里的代码逻辑会稍显复杂，但还是可以通过，动态调试以及一些比较明显的位移操作能够看出这里是修改后的XTEA，这里的 `Delta` 与计数器比较的常数一致 `0x1243d64c`，且循环次数为33

```
*(v9 - 8) = v56;
*(v9 - 7) = v57 + 0x1243D64C;
*(_QWORD *)(v55 - 16) = 64LL;
if ( v21 >= 33 )
  break;
```

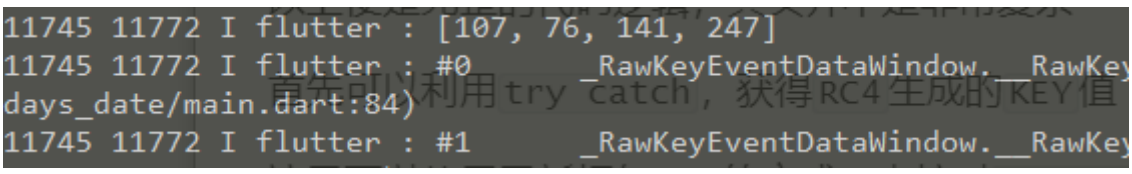
Step 3 RE

以上便是完整的代码逻辑，其实并不是非常复杂

首先可以利用 `try catch`，获得 `RC4` 生成的 `KEY` 值

这里可以使用重新打包APK的方式，直接对 `libapp.so` patch即可，将后序的异常处理步骤 `nop` 掉或者再次调用 `__RawKeyEventDataMisbehave`，即可在 `logcat` 中看到 `key` 值

> 附件[dddart_resigned.apk]



这样就可以编写脚本

```
from ctypes import *
def decrypt(v, key):
    v0 = c_uint32(v[0])
    v1 = c_uint32(v[1])
    summ = c_uint32(0x1243d64c*33)
    delta = 0x1243d64c
    for i in range(33):
        v1.value -= (((v0.value << 4) ^ (v0.value >> 5)) + v0.value) ^
(summ.value + key[(summ.value>>11)&3])
        summ.value -= delta
        v0.value -= (((v1.value << 4) ^ (v1.value >> 5)) + v1.value) ^
(summ.value + key[summ.value & 3])

    print("%#x %#x" % (v0.value, v1.value))

k = [107, 76, 141, 247]
enc = [[0x13E0CFB6, 0x5951F667],[0x3DE56CE8,0xB15076C]]
decrypt(enc[0],k)
decrypt(enc[1],k)
```

```
0xd33214ab 0x88c84d6f
0x70b7b428 0x6cc31f5
```

关键数值有了，之后获得flag的方式就很多了，FRIDA SCRIPT

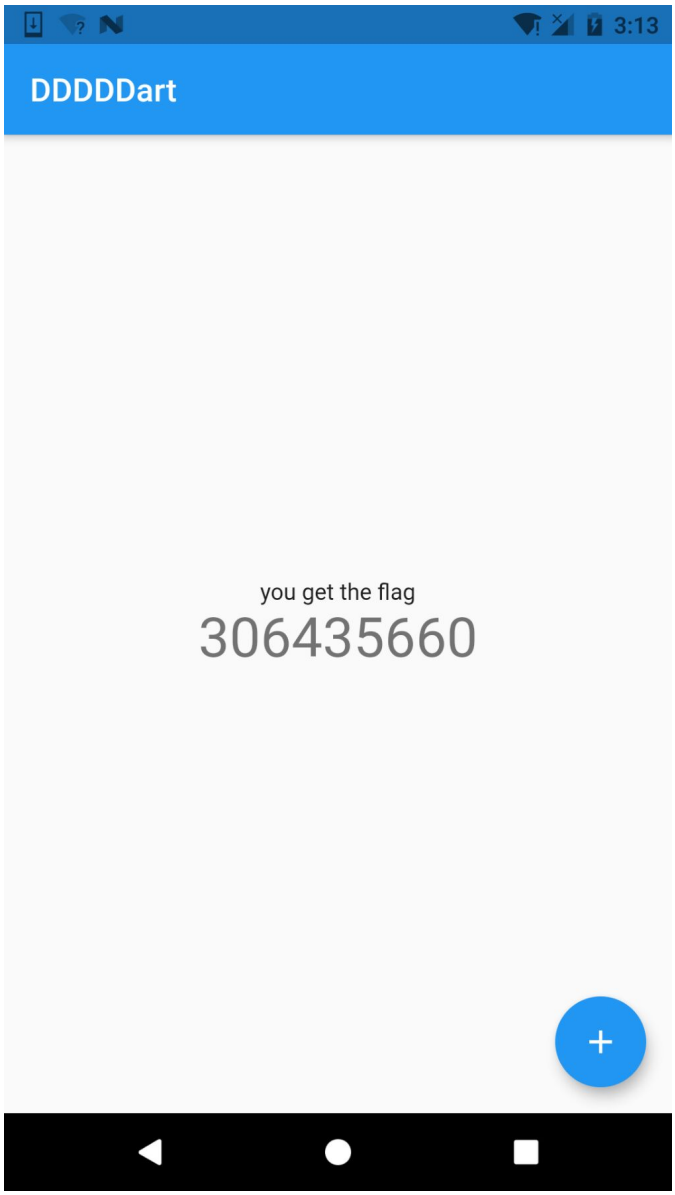
```
import frida
import sys
rdev = frida.get_remote_device()
processes = rdev.enumerate_processes()
pid = rdev.spawn(["com.whats.whats_todays_date"])
rdev.resume(pid)
session = rdev.attach(pid)
script_js = """
var str_name_so = "libapp.so";    //
var list = [0xd33214ab,0x88c84d6f,0x70b7b428,0x06cc31f5]
var cnt = 0
var counter_add = 0
var nextint_func = Module.findBaseAddress(str_name_so).add("0xed4fc");
var increase_func = Module.findBaseAddress(str_name_so).add("0xecd00");
console.log(str_name_so + " addr is ---" + Module.findBaseAddress(str_name_so));

Interceptor.attach(nextint_func, {
    onEnter: function(args){
        Memory.writeS64(counter_add,0x1243d64b); //0x1243d64c - 1
    },
    onLeave: function(retval){
        retval.replace(list[cnt%4])
        cnt++
    }
});

Interceptor.attach(increase_func, {
    onEnter: function(args){
       counter_add = ptr(Memory.readPointer(ptr(this.context.x15))).add(31)
```

```
    },
});

"""
script = session.create_script(script_js)
script.load()
sys.stdin.read()
```



依然是`logcat`中可以找到FLAG