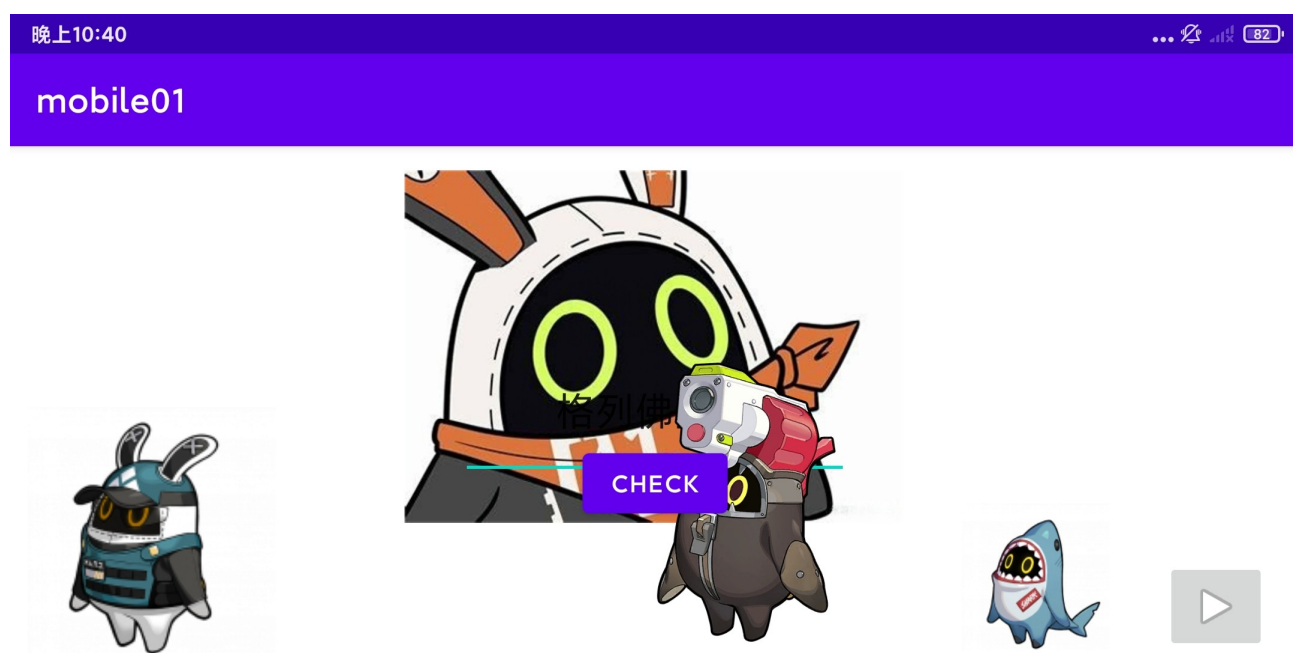


ISCC 练武初赛re+mobile wp-先知社区

mobile

ISCC mobile 邦布出击

安装apk



点击右下角的按钮，进入图鉴界面，百度各种邦布的种类，一个一个试，可以得到三段base64加密的文本
[邦布图鉴 - 绝区零WIKI_BWIKI_哔哩哔哩](#)



邦布图鉴提交处

BBTUJIAN

名字： 鲨牙布

级别： S

SUBMIT

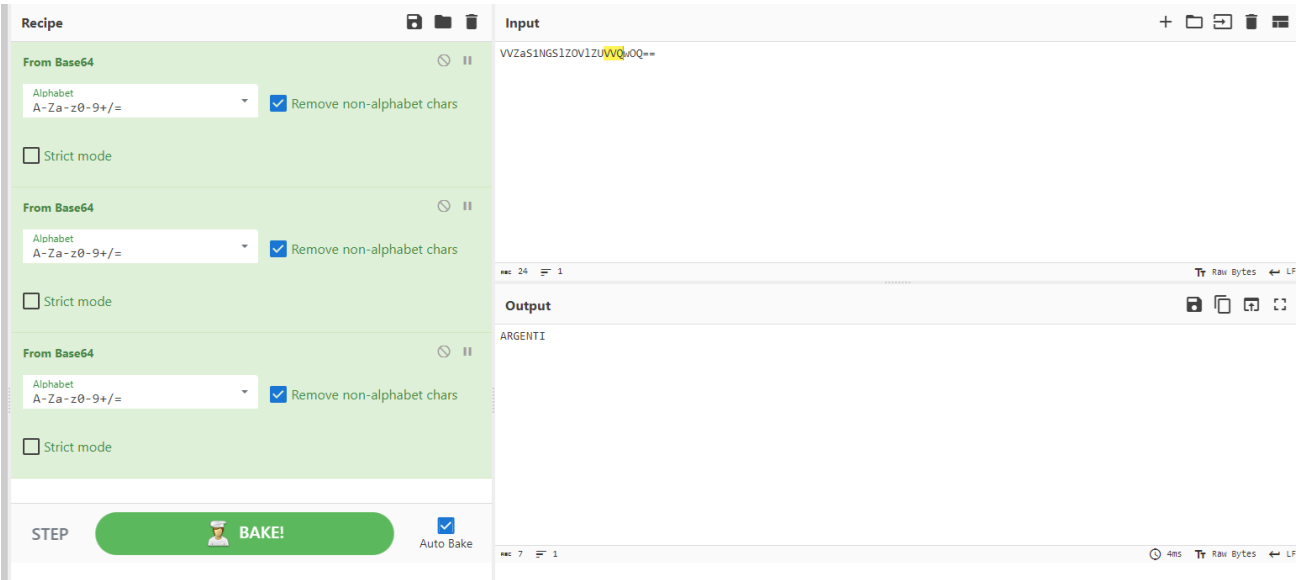
VVZaS1NGS



哔哩

```
VVQwOQ==
1Z0V1ZU
VVZaS1NGS
```

然后将三段base64拼接起来，循环解码三次base64



得到一串明文

尝试打开解压得到的db文件，提示非数据库文件，经查询是经过sqlcipher加密，那么此前得到的明文应该就是解密的key

```
>sqlcipher enflag.db
SQLCipher version 3.8.0.2 2013-09-03 17:11:13
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> PRAGMA key = 'ARGENTI';
sqlite> ATTACH DATABASE 'plaintext.db' AS plaintext KEY '';
sqlite> SELECT sqlcipher_export('plaintext');

sqlite> DETACH DATABASE plaintext;
sqlite> .q
```

id		name	value	info
1	1	flag!	102;108;97;103;123;121;111;11...	Congratulationo.0♦♦&#♦♦
2	2	key?	CdEfGhIjKIMnOpQr	!blowfish!
3	3	():flag?KEY	\u0074\u0068\u0065\u0020\u...	something crucial

flag是假的，实际应该留意的是key以及info中的blowfish（一种加密方式）
使用jadx打开apk

```

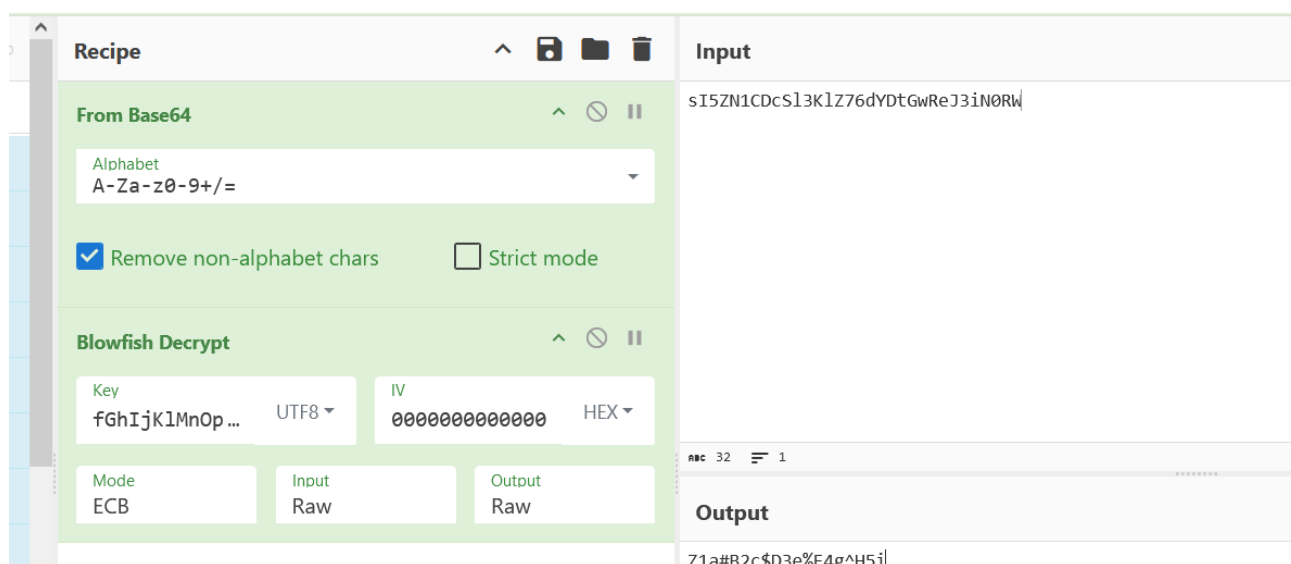
public boolean Jformat(String str) {
    if (str.length() < 7 || !str.substring(0, 5).equals("ISCC{") || str.charAt(str.length() - 1) != '}') {
        return false;
    }
    try {
        String a = a.a();
        Log.d("str1", "des加密明文: " + a);
        try {
            String encrypt = new DESHelper().encrypt(a, "Whitenet", getiv());
            Log.d("DEBUG_RES", "加密结果 res: " + encrypt);
            return str.substring(5, str.length() - 1).equals(encrypt);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    } catch (Exception e2) {
        throw new RuntimeException(e2);
    }
}

public class b {
    private static String hiddenString = "sI5ZN1CDcS13K1Z76dYDtGwReJ3iN0RW";

    public static String b() {
        try {

```

将上图中的密文通过blowfish解密之后得到的内容就是DES的明文



根据apk的逻辑，只有当该明文DES加密的结果和输入内容去掉flag格式后的内容相同才正确
已知明文、key、加密方式，那么对于DES加密，还需要具备的就是iv，但是iv是通过native函数生成的

```
public native String getiv();
```

方法一：分析so文件iv的生成逻辑 -- 生成逻辑比较复杂，放弃

方法二：hook native function，在调用getiv时输出iv

这里使用frida hook（要在手机上先运行frida-server）

```

Java.perform(function () {
    try {
        var cls = Java.use("com.example.mobile01.MainActivity");
        cls.getiv.implementation = function () {
            var iv_val = this.getiv();
            console.log("[*] MainActivity.getiv() called, returned: " + iv_val);
            return iv_val;
        };
        console.log("[+] Hooked com.example.mobile01.MainActivity.getiv()");
    } catch (err) {
        console.error("[-] Failed to hook MainActivity.getiv: " + err);
    }
});

```

```
frida -U -f 进程名 -l hook.js
```

```
[*] MainActivity.getiv() called, returned: Jhuadlhykvdufpssbzipvu
Jhuadlhykvdufpssbzipvu
```

Recipe

DES Encrypt

Key

Whitenet

UTF8

IV

Jhuadlhy

UTF8

Mode

CBC

Input

Raw

Output

Hex

Input

Z1a#B2c\$D3e%F48^H5i

Output

baecf9fd2e97fcb0bed938234e5d6f1ec44e0276c857b8e5

ISCC mobile detective

附件是一个apk文件，用jadx打开

```

hide // androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity, android.app.Activity
24 ct void onCreate(Bundle bundle) {
25     super.onCreate(bundle);
26     ctivityMainBinding.inflate = ActivityMainBinding.inflate(getLayoutInflater());
27     his.binding = inflate;
28     etContentView.inflate(getRoot());
29     his.flagEditText = this.binding.editTextFlag;
30     utton button = this.binding.button;
31     his.submitButton = button;
32     utton.setOnClickListener(new View.OnClickListener() { // from class: com.example.detective.MainActivity.1
33         @Override // android.view.View.OnClickListener
34         public void onClick(View view) {
35             if (MainActivity.this.Jformat(MainActivity.this.flagEditText.getText().toString())) {
36                 Toast.makeText(MainActivity.this, "Congratulations, you are right!", 1).show();
37             } else {
38                 Toast.makeText(MainActivity.this, "PITY", 0).show();
39             }
40         }
41     });
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
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 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }
1001 }
1002 }
1003 }
1004 }
1005 }
1006 }
1007 }
1008 }
1009 }
1010 }
1011 }
1012 }
1013 }
1014 }
1015 }
1016 }
1017 }
1018 }
1019 }
1020 }
1021 }
1022 }
1023 }
1024 }
1025 }
1026 }
1027 }
1028 }
1029 }
1030 }
1031 }
1032 }
1033 }
1034 }
1035 }
1036 }
1037 }
1038 }
1039 }
1040 }
1041 }
1042 }
1043 }
1044 }
1045 }
1046 }
1047 }
1048 }
1049 }
1050 }
1051 }
1052 }
1053 }
1054 }
1055 }
1056 }
1057 }
1058 }
1059 }
1060 }
1061 }
1062 }
1063 }
1064 }
1065 }
1066 }
1067 }
1068 }
1069 }
1070 }
1071 }
1072 }
1073 }
1074 }
1075 }
1076 }
1077 }
1078 }
1079 }
1080 }
1081 }
1082 }
1083 }
1084 }
1085 }
1086 }
1087 }
1088 }
1089 }
1090 }
1091 }
1092 }
1093 }
1094 }
1095 }
1096 }
1097 }
1098 }
1099 }
1100 }
1101 }
1102 }
1103 }
1104 }
1105 }
1106 }
1107 }
1108 }
1109 }
1110 }
1111 }
1112 }
1113 }
1114 }
1115 }
1116 }
1117 }
1118 }
1119 }
1120 }
1121 }
1122 }
1123 }
1124 }
1125 }
1126 }
1127 }
1128 }
1129 }
1130 }
1131 }
1132 }
1133 }
1134 }
1135 }
1136 }
1137 }
1138 }
1139 }
1140 }
1141 }
1142 }
1143 }
1144 }
1145 }
1146 }
1147 }
1148 }
1149 }
1150 }
1151 }
1152 }
1153 }
1154 }
1155 }
1156 }
1157 }
1158 }
1159 }
1160 }
1161 }
1162 }
1163 }
1164 }
1165 }
1166 }
1167 }
1168 }
1169 }
1170 }
1171 }
1172 }
1173 }
1174 }
1175 }
1176 }
1177 }
1178 }
1179 }
1180 }
1181 }
1182 }
1183 }
1184 }
1185 }
1186 }
1187 }
1188 }
1189 }
1190 }
1191 }
1192 }
1193 }
1194 }
1195 }
1196 }
1197 }
1198 }
1199 }
1200 }
1201 }
1202 }
1203 }
1204 }
1205 }
1206 }
1207 }
1208 }
1209 }
1210 }
1211 }
1212 }
1213 }
1214 }
1215 }
1216 }
1217 }
1218 }
1219 }
1220 }
1221 }
1222 }
1223 }
1224 }
1225 }
1226 }
1227 }
1228 }
1229 }
1230 }
1231 }
1232 }
1233 }
1234 }
1235 }
1236 }
1237 }
1238 }
1239 }
1240 }
1241 }
1242 }
1243 }
1244 }
1245 }
1246 }
1247 }
1248 }
1249 }
1250 }
1251 }
1252 }
1253 }
1254 }
1255 }
1256 }
1257 }
1258 }
1259 }
1260 }
1261 }
1262 }
1263 }
1264 }
1265 }
1266 }
1267 }
1268 }
1269 }
1270 }
1271 }
1272 }
1273 }
1274 }
1275 }
1276 }
1277 }
1278 }
1279 }
1280 }
1281 }
1282 }
1283 }
1284 }
1285 }
1286 }
1287 }
1288 }
1289 }
1290 }
1291 }
1292 }
1293 }
1294 }
1295 }
1296 }
1297 }
1298 }
1299 }
1300 }
1301 }
1302 }
1303 }
1304 }
1305 }
1306 }
1307 }
1308 }
1309 }
1310 }
1311 }
1312 }
1313 }
1314 }
1315 }
1316 }
1317 }
1318 }
1319 }
1320 }
1321 }
1322 }
1323 }
1324 }
1325 }
1326 }
1327 }
1328 }
1329 }
1330 }
1331 }
1332 }
1333 }
1334 }
1335 }
1336 }
1337 }
1338 }
1339 }
1340 }
1341 }
1342 }
1343 }
1344 }
1345 }
1346 }
1347 }
1348 }
1349 }
1350 }
1351 }
1352 }
1353 }
1354 }
1355 }
1356 }
1357 }
1358 }
1359 }
1360 }
1361 }
1362 }
1363 }
1364 }
1365 }
1366 }
1367 }
1368 }
1369 }
1370 }
1371 }
1372 }
1373 }
1374 }
1375 }
1376 }
1377 }
1378 }
1379 }
1380 }
1381 }
1382 }
1383 }
1384 }
1385 }
1386 }
1387 }
1388 }
1389 }
1390 }
1391 }
1392 }
1393 }
1394 }
1395 }
1396 }
1397 }
1398 }
1399 }
1400 }
1401 }
1402 }
1403 }
1404 }
1405 }
1406 }
1407 }
1408 }
1409 }
1410 }
1411 }
1412 }
1413 }
1414 }
1415 }
1416 }
1417 }
1418 }
1419 }
1420 }
1421 }
1422 }
1423 }
1424 }
1425 }
1426 }
1427 }
1428 }
1429 }
1430 }
1431 }
1432 }
1433 }
1434 }
1435 }
1436 }
1437 }
1438 }
1439 }
1440 }
1441 }
1442 }
1443 }
1444 }
1445 }
1446 }
1447 }
1448 }
1449 }
1450 }
1451 }
1452 }
1453 }
1454 }
1455 }
1456 }
1457 }
1458 }
1459 }
1460 }
1461 }
1462 }
1463 }
1464 }
1465 }
1466 }
1467 }
1468 }
1469 }
1470 }
1471 }
1472 }
1473 }
1474 }
1475 }
1476 }
1477 }
1478 }
1479 }
1480 }
1481 }
1482 }
1483 }
1484 }
1485 }
1486 }
1487 }
1488 }
1489 }
1490 }
1491 }
1492 }
1493 }
1494 }
1495 }
1496 }
1497 }
1498 }
1499 }
1500 }
1501 }
1502 }
1503 }
1504 }
1505 }
1506 }
1507 }
1508 }
1509 }
1510 }
1511 }
1512 }
1513 }
1514 }
1515 }
1516 }
1517 }
1518 }
1519 }
1520 }
1521 }
1522 }
1523 }
1524 }
1525 }
1526 }
1527 }
1528 }
1529 }
1530 }
1531 }
1532 }
1533 }
1534 }
1535 }
1536 }
1537 }
1538 }
1539 }
1540 }
1541 }
1542 }
1543 }
1544 }
1545 }
1546 }
1547 }
1548 }
1549 }
1550 }
1551 }
1552 }
1553 }
1554 }
1555 }
1556 }
1557 }
1558 }
1559 }
1560 }
1561 }
1562 }
1563 }
1564 }
1565 }
1566 }
1567 }
1568 }
1569 }
1570 }
1571 }
1572 }
1573 }
1574 }
1575 }
1576 }
1577 }
1578 }
1579 }
1580 }
1581 }
1582 }
1583 }
1584 }
1585 }
1586 }
1587 }
1588 }
1589 }
1590 }
1591 }
1592 }
1593 }
1594 }
1595 }
1596 }
1597 }
1598 }
1599 }
1600 }
1601 }
1602 }
1603 }
1604 }
1605 }
1606 }
1607 }
1608 }
1609 }
1610 }
1611 }
1612 }
1613 }
1614 }
1615 }
1616 }
1617 }
1618 }
1619 }
1620 }
1621 }
1622 }
1623 }
1624 }
1625 }
1626 }
1627 }
1628 }
1629 }
1630 }
1631 }
1632 }
1633 }
1634 }
1635 }
1636 }
1637 }
1638 }
1639 }
1640 }
1641 }
1642 }
1643 }
1644 }
1645 }
1646 }
1647 }
1648 }
1649 }
1650 }
1651 }
1652 }
1653 }
1654 }
1655 }
1656 }
1657 }
1658 }
1659 }
1660 }
1661 }
1662 }
1663 }
1664 }
1665 }
1666 }
1667 }
1668 }
1669 }
1670 }
1671 }
1672 }
1673 }
1674 }
1675 }
1676 }
1677 }
1678 }
1679 }
1680 }
1681 }
1682 }
1683 }
1684 }
1685 }
1686 }
1687 }
1688 }
1689 }
1690 }
1691 }
1692 }
1693 }
1694 }
1695 }
1696 }
1697 }
1698 }
1699 }
1700 }
1701 }
1702 }
1703 }
1704 }
1705 }
1706 }
1707 }
1708 }
1709 }
1710 }
1711 }
1712 }
1713 }
1714 }
1715 }
1716 }
1717 }
1718 }
1719 }
1720 }
1721 }
1722 }
1723 }
1724 }
1725 }
1726 }
1727 }
1728 }
1729 }
1730 }
1731 }
1732 }
1733 }
1734 }
1735 }
1736 }
1737 }
1738 }
1739 }
1740 }
1741 }
1742 }
1743 }
1744 }
1745 }
1746 }
1747 }
1748 }
1749 }
1750 }
1751 }
1752 }
1753 }
1754 }
1755 }
1756 }
1757 }
1758 }
1759 }
1760 }
1761 }
1762 }
1763 }
1764 }
1765 }
1766 }
1767 }
1768 }
1769 }
1770 }
1771 }
1772 }
1773 }
1774 }
1775 }
1776 }
1777 }
1778 }
1779 }
1780 }
1781 }
1782 }
1783 }
1784 }
1785 }
1786 }
1787 }
1788 }
1789 }
1790 }
1791 }
1792 }
1793 }
1794 }
1795 }
1796 }
1797 }
1798 }
1799 }
1800 }
1801 }
1802 }
1803 }
1804 }
1805 }
1806 }
1807 }
1808 }
1809 }
1810 }
1811 }
1812 }
1813 }
1814 }
1815 }
1816 }
1817 }
1818 }
1819 }
1820 }
1821 }
1822 }
1823 }
1824 }
1825 }
1826 }
1827 }
1828 }
1829 }
1830 }
1831 }
1832 }
1833 }
1834 }
1835 }
1836 }
1837 }
1838 }
1839 }
1840 }
1841 }
1842 }
1843 }
1844 }
1845 }
1846 }
1847 }
1848 }
1849 }
1850 }
1851 }
1852 }
1853 }
1854 }
1855 }
1856 }
1857 }
1858 }
1859 }
1860 }
1861 }
1862 }
1863 }
1864 }
1865 }
1866 }
1867 }
1868 }
1869 }
1870 }
1871 }
1872 }
1873 }
1874 }
1875 }
1876 }
1877 }
1878 }
1879 }
1880 }
1881 }
1882 }
1883 }
1884 }
1885 }
1886 }
1887 }
1888 }
1889 }
1890 }
1891 }
1892 }
1893 }
1894 }
1895 }
1896 }
1897 }
1898 }
1899 }
1900 }
1901 }
1902 }
1903 }
1904 }
1905 }
1906 }
1907 }
1908 }
1909 }
1910 }
1911 }
1912 }
1913 }
1914 }
1915 }
1916 }
1917 }
1918 }
1919 }
1920 }
1921 }
1922 }
1923 }
1924 }
1925 }
1926 }
1927 }
1928 }
1929 }
1930 }
1931 }
1932 }
1933 }
1934 }
1935 }
1936 }
1937 }
1938 }
1939 }
1940 }
1941 }
1942 }
1943 }
1944 }
1945 }
1946 }
1947 }
1948 }
1949 }
1950 }
1951 }
1952 }
1953 }
1954 }
1955 }
1956 }
1957 }
1958 }
1959 }
1960 }
1961 }
1962 }
1963 }
1964 }
1965 }
1966 }
1967 }
1968 }
1969 }
1970 }
1971 }
1972 }
1973 }
1974 }
1975 }
1976 }
1977 }
1978 }
1979 }
1980 }
1981 }
1982 }
1983 }
1984 }
1985 }
1986 }
1987 }
1988 }
1989 }
1990 }
1991 }
1992 }
1993 }
1994 }
1995 }
1996 }
1997 }
1998 }
1999 }
2000 }
2001 }
2002 }
2003 }
2004 }
2005 }
2006 }
2007 }
2008 }
2009 }
2010 }
2011 }
2012 }
2013 }
2014 }
2015 }
2016 }
2017 }
2018 }
2019 }
2020 }
2021 }
2022 }
2023 }
2024 }
2025 }
2026 }
2027 }
2028 }
2029 }
2030 }
2031 }
2032 }
2033 }
2034 }
2035 }
2036 }
2037 }
2038 }
2039 }
2040 }
2041 }
2042 }
2043 }
2044 }
2045 }
2046 }
2047 }
2048 }
2049 }
2050 }
2051 }
2052 }
2053 }
2054 }
2055 }
2056 }
2057 }
2058 }
2059 }
2060 }
2061 }
2062 }
2063 }
2064 }
2065 }
2066 }
2067 }
2068 }
2069 }
2070 }
2071 }
2072 }
2073 }
2074 }
2075 }
2076 }
2077 }
2078 }
2079 }
2080 }
2081 }
2082 }
2083 }
2084 }
2085 }
2086 }
2087 }
2088 }
2089 }
2090 }
2091 }
2092 }
2093 }
2094 }
2095 }
2096 }
2097 }
2098 }
2099 }
2100 }
2101 }
2102 }
2103 }
2104 }
2105 }
2106 }
2107 }
2108 }
2109 }
2110 }
2111 }
2112 }
2113 }
2114 }
2115 }
2116 }
2117 }
2118 }
2119 }
2120 }
2121
```

可以看到关键是这个stringFromJNI函数，跟进之后发现是native函数，因此用IDA打开so文件

```
37  intext2 = (char *)v22 + 1;
38  LOBYTE(v22[0]) = 2 * len;
39  if ( len )
40 LABEL_5:
41  memmove(intext2, intext, len2);
42  intext2[len2] = 0;
43  (*a1)->ReleaseStringUTFChars(a1, (jstring)input, intext);
44  v20 = 16;
45  strcpy(v21, "Sherlock");
46  xorEncrypt((__int64)v22, &v20, (__int64)&v17);
47  v15[0] = 0LL;
48  v15[1] = 0LL;
49  if ( (v17 & 1) != 0 )
50  v10 = v19;
51  else
52  v10 = v18;
53  v16 = 0LL;
```

关键是这个xorEncrypt函数

```
34 }
35 v9 = *a2;
36 v10 = v3 >> 1;
37 if ( (v9 & 1) != 0 )
38 v11 = *((_QWORD *)a2 + 1);
39 else
40 v11 = v9 >> 1;
41 if ( !(_DWORD)v8 )
42 v10 = *((_QWORD *)v5 + 1);
43 if ( v10 )
44 {
45 v12 = 0LL;
46 do
47 {
48 v13 = (unsigned __int8 *)((_QWORD *)v5 + 2);
49 v14 = (v8 & 1) == 0;
50 v15 = (unsigned __int8 *)((_QWORD *)a2 + 2);
51 if ( !v14 )
52 v13 = v5 + 1;
53 if ( (*a2 & 1) == 0 )
54 v15 = a2 + 1;
55 v16 = v15[v12 % v11];
56 v17 = v13[v12];
57 if ( (*(_BYTE *)a3 & 1) != 0 )
58 v18 = *((_QWORD *)a3 + 16);
59 else
60 v18 = a3 + 1;
61 *(_BYTE *)v18 + v12++ = v16 ^ v17;
62 v8 = *v5;
63 v19 = *((_QWORD *)v5 + 1);
64 v20 = v8 >> 1;
65 LOBYTE(v8) = (v8 & 1) == 0;
66 if ( (_BYTE)v8 )
67 v19 = v20;
68 }
69 while ( v12 < v19 );
70 return result;
71 }
72 }

1  __int64 __usercall xorEncrypt@<X0>(__int64 result@<X0>, unsigned __int8 *a2@<X1>, __int64 a3@<X2>)
2 {
3  unsigned __int8 v3; // w9
4  unsigned __int8 *v5; // x20
5  __int64 v7; // x8
6  unsigned __int64 v8; // x11
7  unsigned __int64 v9; // x8
8  __int64 v10; // x9
9  unsigned __int64 v11; // x8
10 unsigned __int64 v12; // x9
11 unsigned __int8 *v13; // x15
12 bool v14; // zf
13 unsigned __int8 *v15; // x11
14 unsigned __int8 v16; // w11
15 unsigned __int8 v17; // w14
16 __int64 v18; // x15
17 unsigned __int64 v19; // x14
18 unsigned __int64 v20; // x15
19
20 v3 = *(_BYTE *)result;
21 v5 = (unsigned __int8 *)result;
22 if ( (*(_BYTE *)result & 1) != 0 )
23 {
24 result = sub_62B70(a3, *((_QWORD *)result + 16), *((_QWORD *)result + 8));
25 v3 = *v5;
26 LODWORD(v8) = (*v5 & 1) == 0;
27 }
28 else
29 {
30 v7 = *((_QWORD *)result + 16);
31 LODWORD(v8) = 1;
32 *(_QWORD *)a3 = *((_QWORD *)result);
33 *(_QWORD *)a3 + 16 = v7;
34 }
35 v9 = *a2;
36 v10 = v3 >> 1;
37 if ( (v9 & 1) != 0 )
38 v11 = *((_QWORD *)a2 + 1);
39 else
```

通过分析代码可知，该函数先将字符串转换为十六进制，再将输入与key异或之后转为字符串，然后从每4个字符中提取前2个字符，然后再根据一定规律打乱字符串的位置信息，最后替换特定位置的字符

```
import re
from functools import reduce
import binascii

class CryptoSolver:
    @staticmethod
    def extract_alternate_chars(encoded_text):
        """提取每4个字符中的前2个字符"""
```

```

        if len(encoded_text) % 4 == 2:
            encoded_text += '00'
        return ''.join([encoded_text[i:i+2] for i in range(0, len(encoded_text), 4)])

    @staticmethod
    def hex_encode_chars(text):
        """将字符串转为十六进制表示"""
        hex_representation = ''.join([f'{ord(c):04x}' for c in text])
        return hex_representation[2:] if hex_representation.startswith('00') else hex_representation

    @staticmethod
    def process_pattern_swaps(text):
        """处理特定模式的字符交换"""
        result = []
        pattern = re.compile(r'(..)(..)')
        i = 0

        while i < len(text):
            if i + 3 < len(text) and text[i+2:i+4] == '21':
                result.append(text[i+1])
                result.append(text[i])
                i += 4
            else:
                result.append(text[i:i+2])
                i += 2

        return ''.join(result)

    @staticmethod
    def interleave_with_substitution(text):
        """分割、替换和交错合并处理"""
        mid_point = len(text) // 2
        first_half = list(text[:mid_point])
        second_half = list(text[mid_point:])

        # 替换特定位置的'3'为'0'
        for i in range(len(second_half)):
            if second_half[i] == '3' and (i == 0 or i % 3 == 0):
                second_half[i] = '0'

        for i in range(len(first_half)):
            if first_half[i] == '3' and (i == 1 or (i-1) % 3 == 0):
                first_half[i] = '0'

        # 交错合并
        merged = []
        for i in range(len(text)):
            if i % 2 == 0 and i//2 < len(second_half):
                merged.append(second_half[i//2])
            elif i % 2 == 1 and i//2 < len(first_half):
                merged.append(first_half[i//2])

        return ''.join(merged)

    @staticmethod
    def decode_to_chars(encoded):
        """将十六进制编码转换为字符"""
        chars = []
        index = 0

        while index < len(encoded):
            if encoded[index] == '0' and index + 2 < len(encoded):
                # 处理三位编码

```

```

        hex_val = encoded[index+1:index+3]
        chars.append(chr(int(hex_val, 16)))
        index += 3
    else:
        # 处理四位编码
        end = min(index+4, len(encoded))
        hex_val = encoded[index:end]
        if end - index == 4: # 确保有足够的字符
            chars.append(chr(int(hex_val, 16)))
            index += 4

    return ''.join(chars)

class XorDecoder:
    def __init__(self, key="Sherlock"):
        self.key = key.encode('utf-8')
        self.solver = CryptoSolver()

    def xor_bytes(self, hex_string):
        """XOR解密十六进制字符串"""
        # 将十六进制字符串转换为字节列表
        bytes_data = [int(hex_string[i:i+2], 16) for i in range(0, len(hex_string), 2)]

        # 应用XOR操作
        xor_result = []
        for i, byte in enumerate(bytes_data):
            key_byte = self.key[i % len(self.key)]
            xor_result.append(byte ^ key_byte)

        # 转换为UTF-8字符串
        try:
            return bytes(xor_result).decode('utf-8')
        except UnicodeDecodeError:
            # 处理解码错误
            return ''.join([chr(b) for b in xor_result])

    def process_layers(self, input_text):
        """应用多层处理"""
        layer1 = self.solver.hex_encode_chars(input_text)
        layer2 = self.solver.extract_alternate_chars(layer1)
        layer3 = self.solver.process_pattern_swaps(layer2)
        layer4 = self.solver.interleave_with_substitution(layer3)
        return self.solver.decode_to_chars(layer4)

    def decrypt(self, encrypted_hex):
        """完整解密流程"""
        # 先XOR解密
        intermediate = self.xor_bytes(encrypted_hex)
        # 然后应用多层解码
        return self.process_layers(intermediate)

def main():
    # 示例加密数据
    encrypted = "xxxxxxxxxxxxxxxxxxx"

    # 创建解码器并解密
    decoder = XorDecoder()
    result = decoder.decrypt(encrypted)
    print(f"解密结果: {result}")

    # 直接使用XOR解密检查中间结果

```

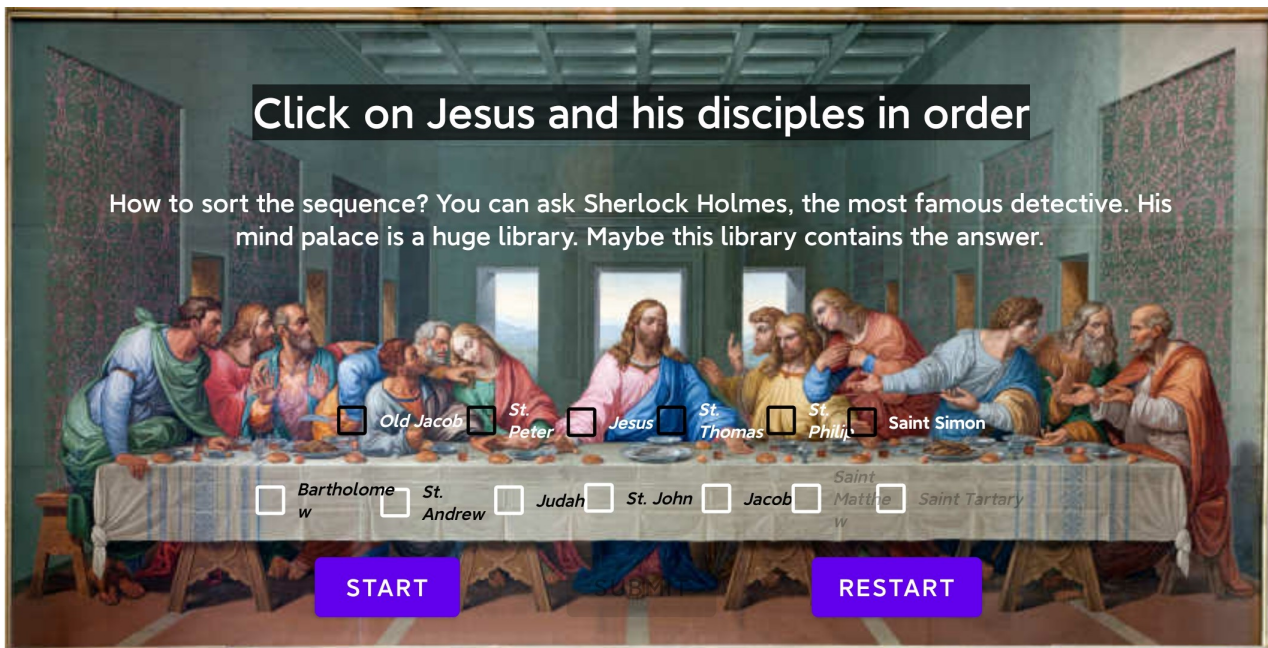


```
xor_result = decoder.xor_bytes(encrypted)
print(f"XOR中间结果: {xor_result}")
```

```
if __name__ == "__main__":
    main()
```

HolyGrail

附件为apk安装包



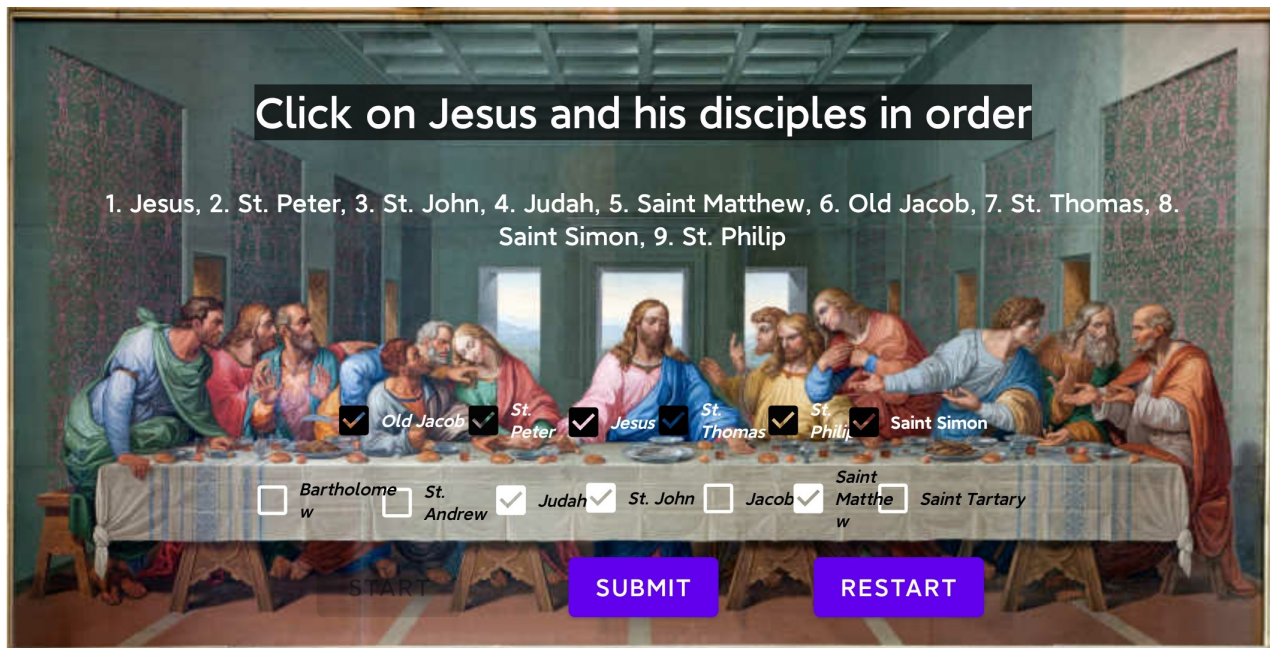
使用jadx打开apk，发现其中有许多checkbox，点击checkbox的响应如下

```
/* JADX INFO: Access modifiers changed from: private */
/* renamed from: onCheckBoxClicked, reason: merged with bridge method [inline-methods] */
119 public void m51lambda$enableCheckBoxes$$comexampleholylgrailMainActivity(CheckBox checkBox) {
120     String resourceEntryName = getResources().getResourceEntryName(checkBox.getId());
121     if (checkBox.isChecked()) {
122         this.userSequence.add(resourceEntryName);
123     } else {
124         this.userSequence.remove(resourceEntryName);
125     }
126     updateSelectedOrderTextView();
127 }
```

每点击一个checkbox就会在userSequence末尾添加当前checkbox的资源名称

```
private void submitSequence() {
    CipherDataHandler.saveCipherText(this, CipherDataHandler.getCipherText(this.userSequence));
    GameData.userSequence.clear();
    GameData.userSequence.addAll(this.userSequence);
    goToNextPage();
}
```

而根据app的提示，需要按照特定顺序点击checkbox，才能进入验证flag的页面，并且返回在native层加密后的密文关于顺序，可以自行百度，也可以问ai，最终顺序如下



如何获得密文：通过frida hook，手动传入特定顺序的参数（每个checkbox的参数也需要通过frida hook得到），然后输出返回的密文

```
var cipher = Java.use("com.example.holygrail.CipherDataHandler");
var args = Java.array("java.lang.String", ["checkBox8", "checkBox6", "checkBox7", "checkBox5", "checkBox12", "checkBox11", "checkBox9", "checkBox10", "checkBox13"]);
console.log(cipher.generateCipherText(args));
```

然后分析验证flag的页面

```

private void submitSequence() {
    String trim = this.flagInput.getText().toString().trim();
    if (!isCorrectFormat(trim)) {
        Toast.makeText(this, "Wrong flag format", 0).show();
        return;
    }
    String substring = trim.substring(5, trim.length() - 1);
    String string = this.sharedPreferences.getString("cipherText", "");
    String validateFlag = a.validateFlag(this, substring);
    if (validateFlag != null && validateFlag.equals(string)) {
        if ("af5c66387436b0c8cfa537be5751c4629c9e288966315c41ec07bf91658a32f4".equalsIgnoreCase(sha256(substring))) {
            Toast.makeText(this, "Success", 0).show();
            return;
        } else {
            Toast.makeText(this, "Correctly matched but in the wrong order.", 0).show();
            return;
        }
    }
    Toast.makeText(this, "Wrong flag", 0).show();
}

private boolean isCorrectFormat(String str) {
    return str.startsWith("ISCC{") && str.endsWith("}");
}

private String sha256(String str) {
    try {
        byte[] digest = MessageDigest.getInstance("SHA-256").digest(str.getBytes("UTF-8"));
        StringBuilder sb = new StringBuilder();
        for (byte b : digest) {
            String hexString = Integer.toHexString(b & UByte.MAX_VALUE);
            if (hexString.length() == 1) {
                sb.append('0');
            }
            sb.append(hexString);
        }
        return sb.toString();
    } catch (Exception unused) {
        return "";
    }
}
}

```

首先检查flag格式，然后调用a类的validateFlag方法

```

public class a {
    public static native String processWithNative(String str);

    static {
        System.loadLibrary("holygrail");
    }

    public static String validateFlag(Context context, String str) {
        return b.a(processWithNative(b(context, str)));
    }

    private static String b(Context context, String str) {
        return vigenereEncrypt(str, getEncryptionKey(context));
    }

    private static String getEncryptionKey(Context context) {
        String str = "";
        try {
            BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(context.getResources().openRawResource(R.raw.key)));
            str = bufferedReader.readLine();
            bufferedReader.close();
            return str;
        } catch (Exception e) {
            e.printStackTrace();
            return str;
        }
    }

    private static String vigenereEncrypt(String str, String str2) {
        StringBuilder sb = new StringBuilder();
        int length = str2.length();
        int i = 0;
        for (int i2 = 0; i2 < str.length(); i2++) {
            char charAt = str.charAt(i2);
            if (Character.isLetter(charAt)) {
                char c = Character.isLowerCase(charAt) ? 'a' : 'A';
                char charAt2 = str2.charAt(i % length);
                charAt = (char) (((charAt - c) + (charAt2 - (Character.isUpperCase(charAt2) ? 'A' : 'a'))) % 26) + c;
                i++;
            }
            sb.append(charAt);
        }
        return sb.toString();
    }
}

```

大概流程

- getEncryptionKey
- vigenereEncrypt
- processWithNative
- b.a

```
public class b {
    public static String a(String str) {
        StringBuilder sb = new StringBuilder();
        if (str.length() % 2 != 0) {
            Log.e("b", "Invalid input: Length must be even");
            return "Invalid input: Length must be even";
        }
        int i = 0;
        while (i < str.length()) {
            int i2 = i + 2;
            sb.append((char) Integer.parseInt(str.substring(i, i2), 16));
            i = i2;
        }
        return sb.toString();
    }
}
```

由于processWithNative是JNI函数，因此尝试frida hook该函数，尝试传入不同的值，发现每个字符对应的加密结果和顺序无关，因此可以直接生成所有字符加密的结果，再对目标字符串进行匹配

解密思路

- 转十六进制
- 字符替换
- 字符偏移

exp

```
from collections import defaultdict
import hashlib
import binascii
from rich.progress import track

CHARACTER_SET = r"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!#$%&'()*+,-./:;<=>?@[\\]^_`{|}~"

def build_mapping_table():
    raw_data = "39213A213B213C21402141214221432144214521464748494A4B4C505152535455565758595A5B5C60616263646550215"
    mapping = []

    idx = 0
    while idx < len(raw_data):
        if idx + 3 < len(raw_data) and raw_data[idx+2:idx+4] == "21":
            mapping.append(raw_data[idx:idx+4].lower())
            idx += 4
        else:
            mapping.append(raw_data[idx:idx+2].lower())
            idx += 2

    return mapping
```

```

class VigenereCipher:
    def __init__(self, key):
        self.key = key.lower()
        self.key_length = len(key)

    def decrypt(self, text):
        result = []
        key_position = 0

        for character in text:
            if not character.isalpha():
                result.append(character)
                continue

            base = ord('a') if character.islower() else ord('A')

            key_char = self.key[key_position % self.key_length]
            key_shift = ord(key_char) - ord('a')

            char_code = ord(character) - base
            decrypted_code = (char_code - key_shift) % 26
            result.append(chr(decrypted_code + base))

            key_position += 1

        return ''.join(result)

def compute_hash(content):
    """计算内容的SHA-256哈希值"""
    return hashlib.sha256(content.encode('utf-8')).hexdigest()

class CryptSolver:
    def __init__(self, cipher_mapping, charset):
        self.cipher_mapping = cipher_mapping
        self.charset = charset
        self.vigenere = VigenereCipher("TheDaVinciCode")

    def chunk_hexstring(self, hex_string):
        """将十六进制字符串分割为块"""
        chunks = []
        position = 0

        while position < len(hex_string):
            if (position + 3 < len(hex_string) and
                hex_string[position+2:position+4] == "21"):
                chunks.append(hex_string[position:position+4])
                position += 4
            else:
                chunks.append(hex_string[position:position+2])
                position += 2

        return chunks

    def decrypt_message(self, encrypted_bytes):
        """解密消息的主流程"""
        hex_data = binascii.hexlify(encrypted_bytes).decode() if isinstance(encrypted_bytes, bytes) else encrypted_bytes

        hex_chunks = self.chunk_hexstring(hex_data)
        print(f"解析后的块: {hex_chunks}")

        translated = []
        for chunk in hex_chunks:
            try:

```

```
        index = self.cipher_mapping.index(chunk.lower())
        translated.append(self.charset[index])
    except ValueError:
        translated.append('?')

    intermediate = ''.join(translated)
    print(f"中间结果: {intermediate}")

    plaintext = self.vigenere.decrypt(intermediate)
    print(f"解密结果: {plaintext}")

    return plaintext

def main():
    cipher_mapping = build_mapping_table()

    solver = CryptSolver(cipher_mapping, CHARACTER_SET)

    encrypted = b"xxxxxxxxxxxxxxxx"
    result = solver.decrypt_message(encrypted.hex())

    return result

if __name__ == "__main__":
    main()
```

whereisflag

下午4:53

... 88

whereisflag

请找到正确的flag并验证

验证flag

验证

jadx打开apk可以看到具体逻辑


```

public void onClick(View view) {
    String obj = MainActivity.this.flagEditText.getText().toString();
    if (obj.length() != 16) {
        Toast.makeText(MainActivity.this, "flag长度错误, 请继续寻找", 0).show();
    } else if (new a().b(obj)) {
        Toast.makeText(MainActivity.this, "恭喜你找到了正确的flag", 1).show();
    } else {
        Toast.makeText(MainActivity.this, "flag错误, 请继续寻找", 0).show();
    }
}
}

```

分析之后发现核心函数是native函数

Native 函数基本介绍

- 定义: Native 函数通过 `native` 关键字在 Java 中声明, 实际代码编译在 `.so` 动态库 (ELF 格式) 中。
- JNI 桥梁: Java 层通过 JNI (Java Native Interface) 调用 Native 函数, 函数名和参数需遵循 JNI 规范。

```

public class a {
    public native String compute(String str);

    public boolean b(String str) {
        if (str.startsWith("ISCC{") && str.endsWith("}")) {
            return compute(str.substring(5, 15)).equals("iB3A7kSISR");
        }
        return false;
    }
}

```

用解压软件直接解压apk文件, 然后进入 `\lib rm64-v8a` 目录找到so文件, 使用IDA64打开so文件, 在其中找到 `Java_` 开头的函数便是native导出函数

在加密函数中首先将输入倒序

```

if ( v11 )
{
    v13 = &v12[v11 - 1];
    if ( v13 > v12 )
    {
        v14 = v12 + 1;
        do
        {
            v15 = *(v14 - 1);
            *(v14 - 1) = *v13;
            *v13-- = v15;
        }
        while ( v14++ < v13 );
        v7 = *((__QWORD *)&v20 + 1);
        v8 = (unsigned __int8 *)v21;
        v9 = (unsigned __int64)(unsigned __int8)v20 >> 1;
        v10 = v20 & 1;
    }
}

```

然后根据字符表查找输入的字符

```

    *(_QWORD *)v21 = v16 + 2;
    goto LABEL_5;
}
v15 = (char *)v21 + 1;
LOBYTE(v21[0]) = 2 * v6;
if ( v6 )
LABEL_5:
    memmove(v15, v5, (size_t)v14);
    *(( BYTE *)v14 + ( QWORD)v15) = 0;
    encrypt((int)v21, v7, v8, v9, v10, v11, v12, v13, v20, v21[0], v22);
    if ( (v21[0] & 1) != 0 )
        operator delete(v23);
    (*(void (__fastcall **)(__int64, __int64, const char *)))(_QWORD *)a1 + 1360LL)(a1, a3, v5);
    if ( (v24 & 1) != 0 )
        v17 = v26;
    else

```

字符表需要动态调试得到

```

1  __int64 __fastcall charToIndex(unsigned int a1)
2  {
3      __int64 v1; // x0
4
5      v1 = std::string::find(&xmmword_52DF0, a1, 0LL);
6      if ( v1 == -1 )
7          return 0xFFFFFFFFLL;
8      else
9          return (unsigned int)(v1 + 1);
10 }

```

```

1  __int64 __fastcall indexToChar(int a1)
2  {
3      unsigned __int64 v1; // x9
4      char *v3; // x8
5
6      if ( a1 < 1 )
7          return 0LL;
8      v1 = *((_QWORD *)&xmmword_52DF0 + 1);
9      if ( (xmmword_52DF0 & 1) == 0 )
10         v1 = (unsigned __int64)(unsigned __int8)xmmword_52DF0 >> 1;
11         if ( v1 < (unsigned int)a1 )
12             return 0LL;
13         if ( (xmmword_52DF0 & 1) != 0 )
14             v3 = (char *)qword_52E00;
15         else
16             v3 = (char *)&xmmword_52DF0 + 1;
17         return (unsigned __int8)v3[a1 - 1];
18 }

```

而根据encrypt、charToIndex、indexToChar函数的逻辑，可以看到在索引转换时有固定偏移，为2
从jadx反编译的结果得到目标密文 `iB3A7kSISR`，解密

exp


```
s = "WHEReISFLAGBCDJKMNOPQTUVXYZabcdghijklmnopqrstuvwxyz01234567890"

ss = "iB3A7kSISR"

print("".join([s[(s.index(i)-2)%len(s)] for i in ss][::-1]))
```

RE

打出flag

从可执行程序的图标判断为pyinstaller编译的程序，使用pyinstxtractor反编译

```
python pyinstxtractor.py asd.exe
```

然后打开反编译的文件夹，打开同名pyc文件，反编译（uncompyle6或者在线）

[python反编译 - 在线工具](#)

```
#!/usr/bin/env python
# visit https://tool.lu/pyc/ for more information
# Version: Python 3.8

import lzma
import base64
exec(lzma.decompress(base64.b64decode('/Td6WFOAAATm1rRGAGAhARYAAAB0L+Wj4EzVCRVdADSbSme4Ujxz7+Hf1941j8gw1Q3vdmpD9b
```

可以将decompress之后的内容写入文件（以下为部分）

```

import base64
總碁碁碁碁碁碁碁=base64.b64encode
總碁碁碁碁碁碁碁碁( )
總碁碁碁碁碁碁碁碁680=;
總碁碁碁碁碁碁碁碁=800
總碁碁碁碁碁碁碁碁=總碁碁碁碁碁碁碁碁.set_mode((總碁碁碁碁碁碁碁碁,總碁碁碁碁碁碁碁碁))
總碁碁碁碁碁碁碁碁.set_caption("打出flag")
總碁碁碁碁碁碁碁碁255,255,255)=
總碁碁碁碁碁碁碁碁0,0,0)=)
總碁碁碁碁碁碁碁碁=(255,0,0)
總碁碁碁碁碁碁碁碁=(0,255,0)
總碁碁碁碁碁碁碁碁=(128,128,128)
總碁碁碁碁碁碁碁碁=總碁碁碁碁碁碁碁碁+總碁碁碁碁碁碁碁碁+總碁碁碁碁碁碁碁碁
總碁碁碁碁碁碁碁碁=30
總碁碁碁碁碁碁碁碁= 'ZpmDBMytVs5Bi0NvBYN4CoA+AXV5AMR0EBp8BYy9 '
總碁碁碁碁碁碁碁碁=5
def 總碁碁碁碁碁碁碁碁(text,shift):
    總碁碁碁碁碁碁碁碁=""
    for 總碁碁碁碁碁碁碁碁 in text:
        if 'A'<=總碁碁碁碁碁碁碁碁<='Z':
            總碁碁碁碁碁碁碁碁+=總碁碁碁碁碁碁碁碁90)%(總碁碁碁碁碁碁碁碁(總碁碁碁碁碁碁碁碁65-(碁))
        elif 'a'<=總碁碁碁碁碁碁碁碁<='z':
            總碁碁碁碁碁碁碁碁+=總碁碁碁碁碁碁碁碁122)%(總碁碁碁碁碁碁碁碁(總碁碁碁碁碁碁碁碁97-(碁))
        else:
            總碁碁碁碁碁碁碁碁+=總碁碁碁碁碁碁碁碁
    總碁碁碁碁碁碁碁碁=總碁碁碁碁碁碁碁碁碁(總碁碁碁碁碁碁碁碁碁.encode()).decode()
    總碁碁碁碁碁碁碁碁=""
    for 總碁碁碁碁碁碁碁碁 in 總碁碁碁碁碁碁碁碁:
        if 'A'<=總碁碁碁碁碁碁碁碁<='Z':
            總碁碁碁碁碁碁碁碁[碁]=總碁碁碁碁碁碁碁碁%(總碁碁碁碁碁碁碁碁(總碁碁碁碁碁碁碁碁65-(碁+shift)%26+65)
            總碁碁碁碁碁碁碁碁+=總碁碁碁碁碁碁碁碁
        elif 'a'<=總碁碁碁碁碁碁碁碁<='z':
            總碁碁碁碁碁碁碁碁[碁]=總碁碁碁碁碁碁碁碁%(總碁碁碁碁碁碁碁碁(總碁碁碁碁碁碁碁碁97-(碁+shift)%26+97)
            總碁碁碁碁碁碁碁碁+=總碁碁碁碁碁碁碁碁
        else:
            總碁碁碁碁碁碁碁碁+=總碁碁碁碁碁碁碁碁
    return 總碁碁碁碁碁碁碁碁
class 總碁碁碁碁碁碁碁碁(總碁碁碁碁碁碁碁碁.Sprite):
    def __init__(總碁碁碁碁碁碁碁碁):

```

叫AI写个脚本去混淆

```
import base64

def decrypt(encrypted_text, shift):
    # 逆向凯撒移位
    decrypted_caesar = []
    for c in encrypted_text:
        if 'A' <= c <= 'Z':
            shifted = (ord(c) - ord('A') - shift) % 26
            decrypted_caesar.append(chr(shifted + ord('A')))
        elif 'a' <= c <= 'z':
            shifted = (ord(c) - ord('a') - shift) % 26
            decrypted_caesar.append(chr(shifted + ord('a')))
        else:
            decrypted_caesar.append(c)
    decrypted_caesar_str = ''.join(decrypted_caesar)

    # Base64解码
    decoded_bytes = base64.b64decode(decrypted_caesar_str)
    decoded_str = decoded_bytes.decode('utf-8')

    # 字符反转
    reversed_str = []
    for c in decoded_str:
        if 'A' <= c <= 'Z':
            reversed_char = chr(ord('Z') - (ord(c) - ord('A')))
        elif 'a' <= c <= 'z':
            reversed_char = chr(ord('z') - (ord(c) - ord('a')))
        else:
            reversed_char = c
        reversed_str.append(reversed_char)
    return ''.join(reversed_str)

target = "ZpmDBMytVs5Bi0NvBYN4CoA+AXV5AMR0EBp8BYy9"
flag = decrypt(target, 5)
print(flag)
```

有趣的小游戏

附件是一个exe和两个txt，其中txt内容为非打印字符
main函数中定义了许多常量

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     _QWORD *v3; // rax
4     _QWORD *v4; // rax
5     _QWORD *v5; // rax
6     __int64 v7[3]; // [rsp+20h] [rbp-60h] BYREF
7     char v8; // [rsp+3Fh] [rbp-41h] BYREF
8     int v9[24]; // [rsp+40h] [rbp-40h] BYREF
9     char v10[32]; // [rsp+A0h] [rbp+20h] BYREF
10    int v11[31]; // [rsp+C0h] [rbp+40h] BYREF
11    char v12; // [rsp+13Fh] [rbp+BFh] BYREF
12    char v13[48]; // [rsp+140h] [rbp+C0h] BYREF
13
14    sub_40B270(argc, argv, envp);
15    v11[0] = 0x18A550A;
16    v11[1] = 0x840630DB;
17    v11[2] = 0x3EC0C129;
18    v11[3] = 0x175BDB99;
19    v11[4] = 0x7FD5E3DB;
20    v11[5] = 0xF99F6912;
21    v11[6] = 0x199B32C1;
22    v11[7] = 0x836C22BB;
23    v11[8] = 0x440E4880;
24    v11[9] = 0xE4EC8310;
25    v11[10] = 0x2F00227A;
26    v11[11] = 0xAB294A2A;
27    v11[12] = 0x8EDB89F1;

```

通过查看附近函数，发现其他地方也定义了常数

```

55    sub_48EC30(a1);
56    *(_DWORD *)(a1 + 48) = 1000;
57    *(_DWORD *)(a1 + 52) = 0;
58    sub_48F820(a1 + 56);
59    *(_DWORD *)(a1 + 80) = 0x12345678;
60    *(_DWORD *)(a1 + 84) = 0x9ABCDEF0;
61    *(_DWORD *)(a1 + 88) = 0xFEDCBA98;
62    *(_DWORD *)(a1 + 92) = 0x76543210;
63    v2 = time64(0i64);
64    srand(v2);
65    qmemcpy(v32, "#####", sizeof(v32));
66    nullsub_4(&v33);
67    v20.m128i_i64[0] = (__int64)v32;
68    v20.m128i_i64[1] = 10i64;
69    sub_48F050(v21, &v20, &v33);
70    qmemcpy(v34, "# # #", sizeof(v34));
71    nullsub_4(&v35);
72    v20.m128i_i64[0] = (__int64)v34;
73    v20.m128i_i64[1] = 10i64;
74    sub_48F050(&v22, &v20, &v35);
75    qmemcpy(v36, "# # #### #", sizeof(v36));
76    nullsub_4(&v37);
77    v20.m128i_i64[0] = (__int64)v36;
78    v20.m128i_i64[1] = 10i64;

```

查看字符串表，可以在其中找到两个txt的文件名，交叉引用查看

```

.S .data:00... 00000001 C
.S .data:00... 00000005 C
.S .data:00... 00000005F C
.S .rdata:0... 0000000A C file1.txt
.S .rdata:0... 0000000A C file2.txt
.S .rdata:0... 00000006 C pause
.S .rdata:0... 0000001E C terminate called recursively\n
.S .rdata:0... 00000031 C terminate called after throwing an instance of '
.S .rdata:0... 0000002E C terminate called without an active exception\n
.S .rdata:0... 0000000C C what():
.S .rdata:0... 00000024 C __gnu_cxx::__concurrency_lock_error
.S .rdata:0... 00000026 C __gnu_cxx::__concurrency_unlock_error
.S .rdata:0... 00000015 C basic_string::append
.S .rdata:0... 00000031 C locale::_S_normalize_category category not found
.S .rdata:0... 00000020 C locale::_Impl::_M_replace_facet
.S .rdata:0... 00000024 C __gnu_cxx::__concurrency_lock_error

```

```

1 void __fastcall process(__int64 a1, int a2, __int64 a3)
2 {
3     __int64 v3; // [rsp+20h] [rbp-20h]
4     void (__fastcall *v4)(__int64, _QWORD, __int64); // [rsp+28h] [rbp-18h]
5     __int64 v5; // [rsp+30h] [rbp-10h]
6     void (__fastcall *lpAddress)(__int64, _QWORD, __int64); // [rsp+38h] [rbp-8h]
7
8     if ( a2 <= 1 )
9     {
10         if ( a2 < -1 )
11         {
12             v4 = (void (__fastcall *)(__int64, _QWORD, __int64))sub_41C090("file2.txt");
13             if ( v4 )
14             {
15                 v3 = sub_48F610(a1);
16                 v4(v3, (unsigned int)a2, a3);
17                 VirtualFree(v4, 0i64, 0x8000u);
18             }
19         }
20     }
21     else
22     {
23         lpAddress = (void (__fastcall *)(__int64, _QWORD, __int64))sub_41C090("file1.txt");
24         if ( lpAddress )
25         {
26             v5 = sub_48F610(a1);
27             lpAddress(v5, (unsigned int)a2, a3);

```

其中process是我重命名的结果

可以看到其中比较奇怪的一点是程序将文件的内容作为函数执行，也就是说原本内容不可见的txt其实是函数的二进制数据，要想知道该函数的具体逻辑，需要动态调试，在此处下断点，触发断点之后在汇编步进就可以看到其中逻辑

```

debug032:0000000000190000 sub     rsp, 80h
debug032:0000000000190004 mov     [rsp+28h], r8
debug032:0000000000190009 mov     [rsp+24h], edx
debug032:000000000019000D mov     [rsp+18h], rcx
debug032:0000000000190012 xor     eax, eax
debug032:0000000000190014 sub     eax, [rsp+24h]
debug032:0000000000190018 mov     [rsp+24h], eax
debug032:000000000019001C mov     eax, 34h ; '4'
debug032:0000000000190021 cdq
debug032:0000000000190022 idiv    dword ptr [rsp+24h]
debug032:0000000000190026 add     eax, 6
debug032:0000000000190029 mov     [rsp+4], eax
debug032:000000000019002D imul    eax, [rsp+4], 9E3779B9h
debug032:0000000000190035 mov     [rsp+0Ch], eax
debug032:0000000000190039 mov     rax, [rsp+18h]
debug032:000000000019003E mov     eax, [rax]
debug032:0000000000190040 mov     [rsp+14h], eax
debug032:0000000000190044 loc_190044: ; CODE XREF: debug032:000000000019017A!j
debug032:0000000000190044 mov     eax, [rsp+0Ch]

```

可以将汇编扔给ai判断函数逻辑

deekseek: “这段汇编代码实现的是 **XXTEA (eXtended TEA)** 算法的解密过程.....”

于是知道了加解密逻辑，并且根据xtea的密钥格式可以判断先前的两处常量中位数较短的是key，而位数较长的是密文

接下来有两种解题方式：

1. 手动分析解密逻辑，自己编写代码
2. 交给ai

xtea的加解密逻辑网上有很多就不细说了，直接给出解密脚本

```

import base64
import struct
from typing import List

def mask_32bit(value):
    """Handle 32-bit unsigned integer overflow"""
    return value & 0xFFFFFFFF

def tea_decrypt(data: List[int], key: List[int]) -> List[int]:
    """
    Alternative implementation of XXTEA decryption

    Args:
        data: List of encrypted 32-bit integers
        key: Decryption key as 4x 32-bit integers

    Returns:
        List of decrypted 32-bit integers
    """
    data_len = len(data)
    rounds = 6 + 52 // data_len

    # Convert input data to a list that we can modify
    result = data.copy()

    # Initialize the sum value
    magic_constant = 0x9E3779B9
    accumulated_sum = mask_32bit(magic_constant * rounds)

    # Main decryption loop
    for _ in range(rounds):
        # Calculate the feistel key index
        mix_index = (accumulated_sum >> 2) & 3

```

```

# Process the data from end to beginning (except first element)
for i in range(data_len - 1, 0, -1):
    # Get the values for the current operation
    current = result[i]
    previous = result[i - 1]
    next_val = result[0] if i == data_len - 1 else result[i + 1]

    # Calculate the mix value
    mx1 = mask_32bit((previous >> 5) ^ (next_val << 2))
    mx2 = mask_32bit((next_val >> 3) ^ (previous << 4))
    mx_sum = mask_32bit(mx1 + mx2)

    # Calculate the key part
    key_index = (i & 3) ^ mix_index
    key_mx = mask_32bit((accumulated_sum ^ next_val) + (key[key_index] ^ previous))

    # Apply the decryption transformation
    result[i] = mask_32bit(current - (mx_sum ^ key_mx))

# Process the first element separately
current = result[0]
previous = result[data_len - 1]
next_val = result[1]

mx1 = mask_32bit((previous >> 5) ^ (next_val << 2))
mx2 = mask_32bit((next_val >> 3) ^ (previous << 4))
mx_sum = mask_32bit(mx1 + mx2)

key_index = (0 & 3) ^ mix_index
key_mx = mask_32bit((accumulated_sum ^ next_val) + (key[key_index] ^ previous))

result[0] = mask_32bit(current - (mx_sum ^ key_mx))

# Update the accumulated sum for next round
accumulated_sum = mask_32bit(accumulated_sum - magic_constant)

return result

def decrypt_and_check(encrypted_data, encryption_key, max_iterations=10000):
    """
    Repeatedly decrypt data and check for readable output

    Args:
        encrypted_data: List of encrypted 32-bit integers
        encryption_key: List of 4 32-bit integers
        max_iterations: Maximum number of decryption iterations
    """
    current_data = encrypted_data.copy()

    for iteration in range(max_iterations):
        # Decrypt one round
        current_data = tea_decrypt(current_data, encryption_key)

        # Try to interpret as text in different ways
        raw_bytes = b''.join([struct.pack("<I", val) for val in current_data])

        # Method 1: Try to decode the entire output as UTF-8
        try:
            decoded_text = raw_bytes.decode('utf-8')
            print(f"Iteration {iteration + 1}: Found valid UTF-8!")
            print(decoded_text)
        except UnicodeDecodeError:
            pass

```

```

# Method 2: Try to extract first byte of each word
try:
    first_bytes = bytes([raw_bytes[i] for i in range(0, len(raw_bytes), 4)])
    decoded_first = first_bytes.decode('utf-8')
    if any(c.isprintable() for c in decoded_first):
        print(f"Iteration {iteration + 1}: First bytes as text: {decoded_first}")
except UnicodeDecodeError:
    pass

# Main execution
def main():
    # Same key and encrypted data as original
    encryption_key = [0x12345678, 0x9ABCDEF0, 0xFEDCBA98, 0x76543210]
    encrypted_data = [
        0x018A550A, 0x840630DB, 0x3EC0C129, 0x175BDB99,
        0x7FD5E3DB, 0xF99F6912, 0x199B32C1, 0x836C22BB,
        0x440E4880, 0xE4EC8310, 0x2F00227A, 0xAB294A2A,
        0x8EDB89F1, 0x28099186, 0xD04F421F, 0x23E7FD1C,
        0x6F48B862, 0x61796B6A, 0x857587A7, 0x33254C3A,
        0x06AAB088, 0x568A0B78, 0xAC64D9CF, 0xFB40A2C6,
        0x9082056A, 0x4FAAB834, 0x5D033C8B, 0x7D570A1C,
        0xCC81E29B, 0xCE1DE040
    ]

    decrypt_and_check(encrypted_data, encryption_key)

if __name__ == "__main__":
    main()

```

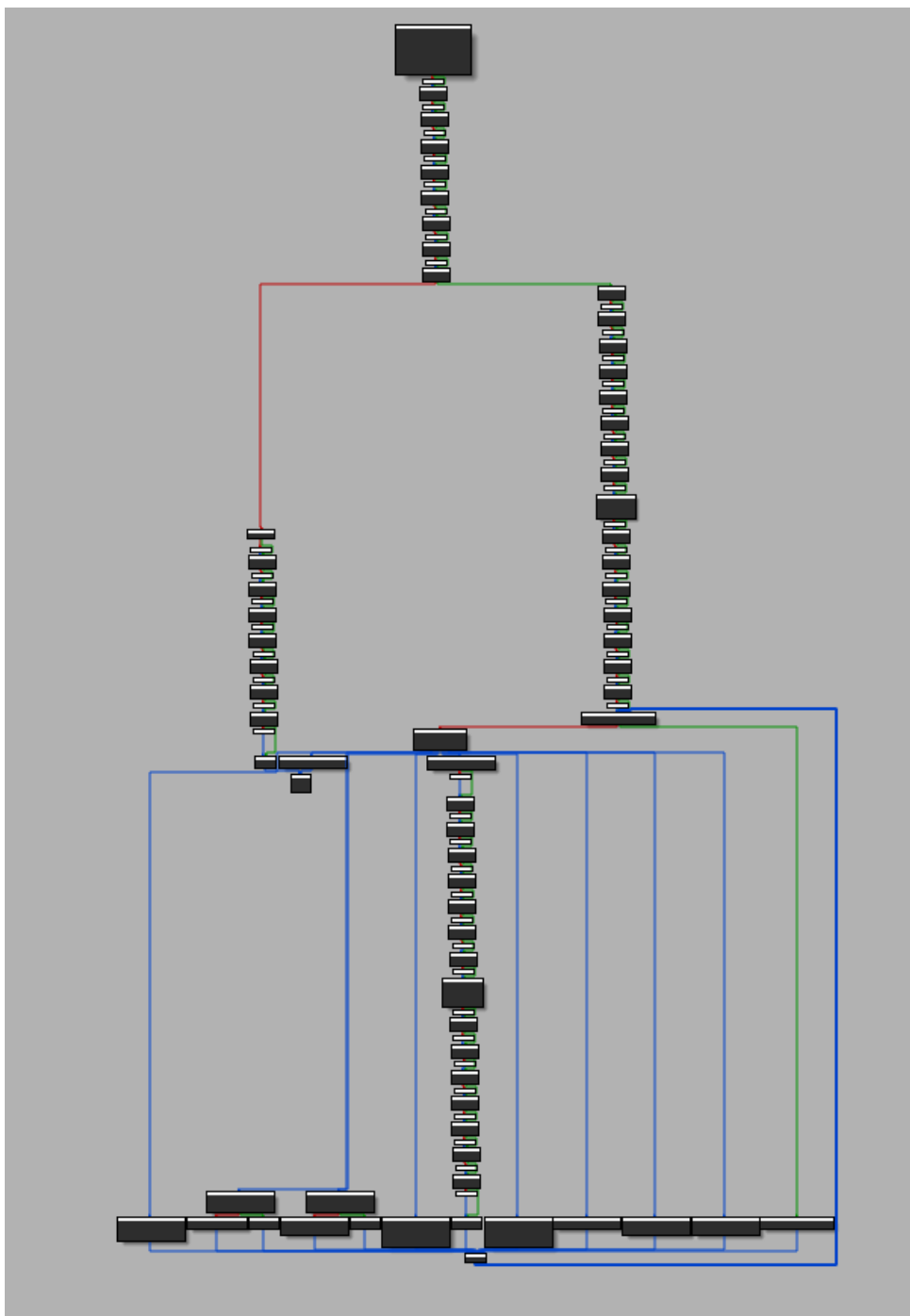
真？复杂

题目附件是一个raw文件，010editor查看发现JFIF文件头，提取图片

8C 8C 53 F2	21 12 63 FD	9D 55 CB 5A	08 2D 53 8C	ÆESò!.cý.UËZ.-SE
F5 0F F6 A3	9D 0B A6 F9	8C FC AC 99	A3 9F F8 79	ö.ö£... ùEü-™£Yøÿ
16 7B AE 70	97 EF 93 8D	FC DC 5B 1B	47 53 84 F0	.{@p-í".üÜ[.GS,,ð
4A 70 B5 9D	01 AA 9F 38	16 F1 08 60	B7 98 F2 7C	Jpµ..ªÿ8.ñ.`~ò
87 A2 E0 97	3E B6 39 A7	00 70 12 17	BC CF E7 00	+çà->¶9\$.p.¼İç.
D8 FF E0 00	10 4A 46 49	46 00 01 01	01 00 78 00	øÿà. JFIF.....x.
78 00 00 FF	DB 00 84 00	06 06 06 06	07 06 07 08	x.ÿÜ.,,.....
08 07 0A 0B	0A 0B 0A 0F	0E 0C 0C 0E	0F 16 10 11
10 11 10 16	22 15 19 15	15 19 15 22	1E 24 1E 1C"......".\$. ..
1E 24 1E 36	2A 26 26 2A	36 3E 34 32	34 3E 4C 44	..\$.6*&&*6>424>LD
44 4C 5F 5A	5F 7C 7C A7	01 06 06 06	06 07 06 07	DL_Z_ \$.
08 08 07 0A	0B 0A 0B 0A	0F 0E 0C 0C	0E 0F 16 10
11 10 11 10	16 22 15 19	15 15 19 15	22 1E 24 1E"......".\$. ..
1C 1E 24 1E	36 2A 26 26	2A 36 3E 34	32 34 3E 4C	..\$.6*&&*6>424>L
44 44 4C 5F	5A 5F 7C 7C	A7 FF C2 00	11 08 00 A2	DDL_Z_ \$ÿÂ....ç
02 37 03 01	22 00 02 11	01 03 11 01	FF C4 00 1C	.7.."......ÿÄ..



然后使用cyberchef解密，解密之前要先把原raw文件中附加的图片信息删除
解密之后得到压缩包一个，解密得exe文件和enc文件各一个



虽然流程图长这个样，但是是可以手动去除的

```

65     case 1:
66         if ( (unsigned int)sub_4015DE() )
67             sub_401550();
68         if ( (unsigned int)sub_401682() )
69             sub_40172A();
70         if ( (unsigned int)sub_40172A() )
71             sub_4015DE();
72         if ( (unsigned int)sub_4015DE() )
73             sub_401550();
74         if ( (unsigned int)sub_401550() )
75             sub_401682();
76         if ( (unsigned int)sub_40172A() )
77             sub_401550();
78         if ( (unsigned int)sub_401550() )
79             sub_40172A();
80         if ( (unsigned int)sub_401682() )
81             sub_40172A();
82         if ( (rand() & 1) != 0 )
83         {
84             if ( (rand() & 1) != 0 )
85             {
86                 if ( (v4 & 1) != 0 )
87                     v5 = 15;
88                 else
89                     v5 = 2;
90             }
91             else if ( (v4 & 1) != 0 )
92             {
93                 v5 = 15;
94             }
95             else
96             {
97                 v5 = 2;
98             }
99         }
100         else if ( (rand() & 1) != 0 )
101         {
102             if ( (v4 & 1) != 0 )

```

第一种方法：（直接忽略和输入无关的语句和函数，对于涉及到修改输入的语句统统下断点）

第二种方法：直接分析加密函数的switch逻辑，可以发现是对奇偶索引的字符做不同的变换，核心变量为v4（索引）和v5（控制跳转的case），通过 `v4&1` 的操作判断奇偶

通过分析exe文件可知原本逻辑是给定flag.txt，用exe加密得到enc文件，而现在只有enc文件，故需要逆向推解密逻辑

通过分析得到解密脚本

```

with open('flag.txt.enc', 'rb') as f:
    encrypted = f.read()

key = [0x88, 0x83, 0xA3, 0x7E, 0xEA, 0xA1, 0xBA, 0x25, 0x72, 0xCF, 0x1D, 0x6E, 0x79, 0x50, 0x17, 0x50]
decrypted = []
for v4, byte in enumerate(encrypted):
    if v4 % 2 == 0: # 偶数索引处理
        temp = (~byte) & 0xFF # 取消取反
        temp = (temp + v4) % 256 # 逆向减法
        temp ^= key[v4 % 16] # 异或密钥
        orig = (temp - v4) % 256 # 逆向加法
    else: # 奇数索引处理
        temp = byte ^ v4 # 取消异或v4
        temp = (temp - v4) % 256 # 逆向加法
        temp ^= key[v4 % 16] # 异或密钥
        orig = (temp + v4) % 256 # 逆向减法
    decrypted.append(orig)

# 输出可打印字符（避免解码错误）
print(''.join([chr(b) if 32 <= b <= 126 else '.' for b in decrypted]))

```

题目附件: faze.exe

使用IDA打开附件

```
text:0000000000401DBE loc_401DBE: ; CODE XREF: main+CD1j
text:0000000000401DBE lea rax, [rbp+390h+var_3B0]
text:0000000000401DC2 mov rcx, rax
text:0000000000401DC5 call _Z3xP9PA10_i ; xP9(int (*)(10))
text:0000000000401DCA lea rax, [rbp+390h+var_3BD]
text:0000000000401DCE mov rcx, rax
text:0000000000401DD1 call _Z3mS2Pc ; mS2(char *)
text:0000000000401DD6 lea rdx, [rbp+390h+var_3BD]
text:0000000000401DDA lea rax, [rbp+390h+var_3D0]
text:0000000000401DDE mov r8, rdx
text:0000000000401DE1 lea rdx, aIsccS ; "ISCC{%s}"
text:0000000000401DE8 mov rcx, rax
text:0000000000401DEB call _Z9sprintf_sIly19EEiRAT_cPKcz ; sprintf_s<19ull>(char (&)[19ull],char const*,...)
text:0000000000401DF0 lea rax, [rbp+390h+var_3F0]
text:0000000000401DF4 mov rcx, rax
text:0000000000401DF7 call _ZNSt7_cxx1112basic_stringIcSt11char_traitsIcESaIcEEC1Ev ; std::__cxx11::basic_string<char,std::cha
text:0000000000401DFC lea rdx, aEnterFlag ; "Enter flag:"
text:0000000000401E03 mov rcx, cs:_refptr_ZSt4cout
text:0000000000401E0A ; try {
text:0000000000401E0A call _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc ; std::operator<<(std::char_traits<char>>(st
text:0000000000401E0F rdx, cs:_refptr_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_0_ES6_
text:0000000000401E16 mov rcx, rax
text:0000000000401E19 call _ZNSo1sEPFRSoS_E ; std::ostream::operator<<(std::ostream & (*)(std::ostream &))
text:0000000000401E1E lea rax, [rbp+390h+var_3F0]
text:0000000000401E22 mov rdx, rax
text:0000000000401E25 mov rcx, cs:_refptr_ZSt3cin
text:0000000000401E2C call _ZSt7getlineIcSt11char_traitsIcESaIcEERSt13basic_istreamIT_0_ES7_RNSt7_cxx1112basic_stringIS4_S5_T
text:0000000000401E31 rdx, [rbp+390h+var_3D0]
text:0000000000401E35 lea rax, [rbp+390h+var_3F0]
text:0000000000401E39 mov rcx, rax
text:0000000000401E3C call _ZSteqIcSt11char_traitsIcESaIcEERKNSt7_cxx1112basic_stringIT_0_T1_EEPKS5_ ; std::operator==(char>
text:0000000000401E41 test al, al
text:0000000000401E43 jz short loc_401E69
text:0000000000401E45 lea rdx, aCorrect ; "Correct!"
text:0000000000401E4C mov rcx, cs:_refptr_ZSt4cout
```

一眼C++, 通过判断代码可以发现目标字符串在用户输入之前 (getline) 已经完成了目标字符串的初始化, 所以这
里有多种解法

1. 在sprintf上下断点, 直接查看写入目标字符串的内容
2. 在比较的时候 (operator==) 下断点, 查看比较的数据
这里选择前者, 在程序暂停时跳转到rcx所在地址

```
003C60]:000000000079FA62 db 0
003C60]:000000000079FA63 db 28h ; (
003C60]:000000000079FA64 db 3Ah ; :
003C60]:000000000079FA65 db 26h ; &
003C60]:000000000079FA66 db 5Bh ; [
003C60]:000000000079FA67 db 47h ; G
003C60]:000000000079FA68 db 42h ; B
003C60]:000000000079FA69 db 5Ch ; \
003C60]:000000000079FA6A db 58h ; X
003C60]:000000000079FA6B db 72h ; r
003C60]:000000000079FA6C db 46h ; F
003C60]:000000000079FA6D db 61h ; a
003C60]:000000000079FA6E db 29h ; )
003C60]:000000000079FA6F db 0
```

greeting

首先IDA打开可执行文件, 会发现有些函数反编译的结果不正确, 且提示错误, 因此可以查看目标函数附近的汇编
代码, 找到类似加密逻辑的代码

```

.text:00000001400016B0 loc_1400016B0: ; CODE XREF: sub_140001220+4D0↓j
.text:00000001400016B0 mov     r8, [rbp+70h+var_60]
.text:00000001400016B4 loc_1400016B4: ; CODE XREF: sub_140001220+4C6↓j
.text:00000001400016B4 mov     [r8+rsi], r12b
.text:00000001400016B8 inc     rsi
.text:00000001400016BB mov     [rbp+70h+var_58], rsi
.text:00000001400016BF cmp     r14, rsi
.text:00000001400016C2 jz      short loc_1400016F2
.text:00000001400016C4 loc_1400016C4: ; CODE XREF: sub_140001220+482↑j
.text:00000001400016C4 mov     rax, rsi
.text:00000001400016C7 mul     r15
.text:00000001400016CA shr     dl, 2
.text:00000001400016CD movzx   eax, dl
.text:00000001400016D0 lea     eax, [rax+rax*4]
.text:00000001400016D3 lea     r12d, [rsi+5Ah]
.text:00000001400016D7 xor     r12b, [rbx+rsi]
.text:00000001400016DB mov     ecx, esi
.text:00000001400016DD sub     ecx, eax
.text:00000001400016DF rol     r12b, cl
.text:00000001400016E2 cmp     rsi, [rbp+70h+var_68]
.text:00000001400016E6 jnz     short loc_1400016B4
.text:00000001400016E6 ; -----
.text:00000001400016E8 db 48h
.text:00000001400016E8 ; } // starts at 140001605
.text:00000001400016E9 ; -----

```

明显的异或和循环左移操作，大概率是加密逻辑

通过分析可知，代码首先是计算一个偏移，然后将目标数据对应索引的字节在异或 $i+0x5a$ 之后（esi为索引）循环左移该计算出来的偏移，因此目标可以分为两步：

1. 分析该偏移的计算方式
2. 反推整个加密逻辑

这里的r15其实是一个固定的值

```

loc_140001689: ; CODE XREF: sub_140001220+34D↑j
add     rbx, rax
mov     r8d, 1
xor     esi, esi
mov     r15, 0CCCCCCCCCCCCCDh
lea     rdi, [rbp+70h+var_68]
jmp     short loc_1400016C4

```

关于偏移量的计算

- 通过手动分析

- `mul r15` 和 `shr dl, 2` 的组合实际上执行的是整数除法 $i / 5$
- `lea eax, [rax+rax*4]` 计算的是 $(i/5)*5$
- `sub ecx, eax` 计算的是 $i - (i/5)*5$
- 以上逻辑等价于 $i\%5$

- 直接动态调试可以发现rol操作中cl的取值是0、1、2、3、4、0.....，所以其实偏移的计算方式是索引对5取余

然后就是逆向整个加密逻辑，有了偏移的计算方式，解密的逻辑很好推，就是对每个字节先循环右移再异或 $(i+0x5a)$

对于密文，通过交叉引用和人肉分析等方式最终可以找到位于 `0x014001B390`

因此完整的解密脚本如下

```

def encrypt(input_bytes):
    output = bytearray(len(input_bytes))
    for i in range(len(input_bytes)):
        div_result = (i // 10) * 5

        value = (i + 0x5A) & 0xFF
        value ^= input_bytes[i]

        rot_amount = (i - div_result) & 0x7
        value = ((value << rot_amount) | (value >> (8 - rot_amount))) & 0xFF

        output[i] = value

    return output

def decrypt(encrypted_bytes):
    output = ""
    for i in range(len(encrypted_bytes)):
        rot_amount = i % 5

        value = encrypted_bytes[i] & 0xFF
        value = ((value >> rot_amount) | (value << (8 - rot_amount))) & 0xFF

        value ^= (i + 0x5A)

        output += chr(value & 0xFF)

    return output

def main():
    encrypted_hex = "xxxxxxxxxx"
    encrypted_bytes = bytes.fromhex(encrypted_hex)

    decrypted = decrypt(encrypted_bytes)
    print("Decrypted:", decrypted)

if __name__ == "__main__":
    main()

```