Assignment 5 - Report - Attack Phase

Secure Systems Engineering

Shreyas Bargale CS22B016
Raadhes Chandaluru CS22B069

Note

In binary 65 the output of global flag will always be ZERO because the compute_gf is:

```
uint8_t compute_gf(uint8_t* eggs){
    return eggs[4] * eggs[3] * 32 + eggs[4];
}
```

We have broken this by running the compute_gf function in the binary with different eggs inputs. However in runtime eggs[4] is always ZERO because the 5th row of egg_params has (i,j) = (l,m), hence the xor will be taken on take state element to get eggs[4] which would be zero.
And due to this the compute_gf output is always zero as all its terms depend on eggs[4].

Table of submitted data/ section components

We have ensured that most submitted data components would be correct. However some may be wrong so we ask that you make sure to test each component individually for scoring when the output of executable generated from main.c submitted by us is not correct.
For example:

If key is wrongly submitted, but egg_params & compute_gf is correct in submission then simply running the a.out produced on compilation of our submitted main.c will generate wrong Cipher, egg[0] and global_flag output.

Binary	key	egg_params	compute_gf
1	Yes	Yes	Yes
2	Yes	Yes	Yes
4	Yes	Yes	Yes
5	Yes	Yes	Yes
7	Yes	Yes	Yes
12	Yes	Yes	Yes
13	Yes		
14	Yes	Yes	Yes
15	Yes	Yes	Yes

16	Yes	Yes	Yes
17	Yes	Yes	Yes
19	Yes	Yes	Yes
22			
24	Yes	Yes	Yes
25			
27	Yes		
31	Yes	Yes	Yes
33	Yes	Yes	Yes
35	Yes	Yes	Yes
37	Yes	Yes	Yes
39	Yes	Yes	Yes
42	Yes	Yes	Yes
45	Yes	Yes	Yes
49			
53	Yes	Yes	Yes
55	Yes	Yes	Yes
56	Yes		
60			
61	Yes	Yes	Yes
63	Yes	Yes	Yes
65	Yes	Yes	Yes
68	Yes	Yes	
69			
72			
77	Yes	Yes	Yes
79	Yes	Yes	Yes

83	Yes	Yes	Yes
85			
88	Yes	Yes	Yes
89			
91	Yes	Yes	Yes
99	Yes	Yes	Yes

Components we are most unsure about:

- We are unsure if egg_params of 42 is correct.
- We are unsure if compute_gf of 17 is correct and not even sure if the binary has implemented this properly.

Explanation of Common Techniques

- Key:
    - Can be obtained by checking the first 16 bytes of Roundkey in KeyExpansion or AddRoundKey(0,...) call in Cipher
    - Can also be obtained by taking xor of state before and after AddRoundKey(0,...) call. (This is done in case the function is obfuscated in some manner)
- Egg_params:
    - In easy binaries we are able to simply print the content of the egg_params array at runtime. For extremely minimal obfuscation binaries we can even just check the contents in the binary using ghidra.
    - Print egg_params once in local variables in cipher function using gdb if the storage is obfuscated.
- Compute_gf
    - In easy binaries we find some computing function which is minimally obfuscated or we find some line of code presented by ghidra which is clearly the compute_gf logic
    - In harder binaries with a compute_gf function we can use a method of setting eggs manually in gdb and allowing compute_gf to generate global_flag and checking the output to derive the function. Giving some simple inputs of eggs like (1,0,0,0,0),(0,1,0,0,0),(0,0,1,0,0) etc will help us understand which eggs contribute. We then also check for multiplication & coefficients by using different combinations of non-zero valued eggs. The probable function is verified with test cases after a good number of
- Some binaries the address of instructions shown by ghidra is not the same as runs in gdb. Generally text section is near address 0x555555555000 in these binaries and ghidra it may be near 0x101000. This is probably due to position independent code. To set breakpoints we check the memory mappings and locations of sections using:
    - info proc mapping
    - info files - shows location of sections

    - Afterwards we simply add the necessary offsets to set breakpoints at instructions.
- To anti debugging techniques where teams used ptrace function or other functions to check whether the binary is being run on GDB, we simply set some instructions to nops.
  We set the exit() in the branch to NOP instructions sometimes. Other times we changed a few instructions to redirect control flow. HxD editor was used.

Explanation of Attacked Binaries

We have left empty space beneath the headings of binaries in which we have not cracked any components.

1

The initial binary creates “unpacked” file.
We removed the unlink syscall in the binary so we can use the binary “unpacked”

Unpacked creates p1, p2 and p3. We also remove the unlink syscalls of these files so we can run these binaries in terminal.
In unpacked, the eggs[3] and eggs[2] is passed to p2.
P1 outputs the key. Just running it gives the key.
P3 spews the egg_params. Just running it gives the egg_params.
They used pipe to get these.

```
close(local_62c);
sprintf(local_498,"%d",(ulong)*(byte *)(param_1 + 3));
sprintf(local_428,"%d",(ulong)*(byte *)(param_1 + 2));
local_628 = "./p2";
local_620 = local_498;
local_618 = local_428;
local_610 = 0;
execv("./p2",&local_628);
```

Running p2 with some combinations of inputs of egg[2] and eggs[3] easily gives the compute_gf logic.

```
rad@DESKTOP-SLPVP4D:/mnt/c/Users/LEGION/OneDrive/Documents/raadhe
ses/CS6570/Assignments/A5/AttackPhase/Fortify/1$ ./p2 1 1
117
rad@DESKTOP-SLPVP4D:/mnt/c/Users/LEGION/OneDrive/Documents/raadhe
ses/CS6570/Assignments/A5/AttackPhase/Fortify/1$ ./p2 1 0
38
rad@DESKTOP-SLPVP4D:/mnt/c/Users/LEGION/OneDrive/Documents/raadhe
ses/CS6570/Assignments/A5/AttackPhase/Fortify/1$ ./p2 0 1
79
```

```
uint8_t compute_gf(uint8_t* eggs){
        return eggs[3] * 38 + eggs[2] * 79;
}
```

2

Applied the common techniques mentioned for cracking. All the components were simple and direct after opening in ghidra and using gdb.

4

FUN_00403250 is the cipher function.
Compute_gf:

```
    *param_1 = param_2[0x14] ^ *param_1;
    param_1[1] = uVar17 ^ param_1[1];
  }
  if ((cVar2 == '\x04') && (cVar1 == '\n')) {
    uVar17 = (ulong)DAT_004c5451;
    DAT_004c5451 = DAT_004c5451 + 1;
    (&DAT_004c64e0)[uVar17] =
         *(byte *)((long)param_1 + (ulong)bVar4 + (ulong)bVar3 * 4) ^
         *(byte *)((long)param_1 + (ulong)bVar6 + (ulong)bVar5 * 4);
  }
  DAT_004c4b80 = FUN_00403180(&DAT_004c3060);
  DAT_004c4b70 = DAT_004c64e0;
  DAT_004c5450 = DAT_004c4b80;
  if (lVar12 != *(long *)(in_FS_OFFSET + 0x28)) {
                /* WARNING: Subroutine does not return */
    FUN_0044e820();
  }
  return;
```

```
00403230 89 d0          MOV     EAX,EDX
00403232 89 d7          MOV     EDI,EDX
00403234 44 8d 04 12    LEA     R8D,[RDX + RDX*0x1]
00403238 c1 e0 06       SHL     EAX,0x6
0040323b c1 e7 05       SHL     EDI,0x5
0040323e 01 f8          ADD     EAX,EDI
00403240 44 29 c0       SUB     EAX,R8D
00403243 41 0f af c1    IMUL    EAX,R9D
00403247 29 d0          SUB     EAX,EDX
00403249 c3             RET
```

```
  *(undefined4 **)(puVar16 + 2) = puVar17;
  do {
    lVar15 = *(long *)(lVar15 + 8);
  } while (lVar15 != 0);
  return (uint)DAT_004c64e3 * 0x5e * (uint)DAT_004c64e4 * 0x12 - (uint)DAT_004c64e3;
}


_00402f40();
r15 = *(long *)(in_FS_OFFSET + 0x28);
yte *)puVar26 = (byte)*puVar26 ^ bRam00000000004c4cd3;
yte *)((long)puVar26 + 1) = *(byte *)((long)puVar26 + 1) ^ bRam00000000004c4cd5;
```

Below is not the key on verifying with test cases:

```
(gdb) b *0x00402ea3
Breakpoint 2 at 0x402ea3
(gdb) c
Continuing.

Breakpoint 2, Python Exception <type 'exceptions.NameError'> Installation error:
 gdb.execute_unwinders function is missing:
0x0000000000402ea3 in ?? ()
(gdb) x/16x 140737488346160
0x7fffffffdc30: 0x0c989133      0xd672ca20      0x5f13b769      0x7e00cbc0
0x7fffffffdc40: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdc50: 0x00000000      0x00000000      0x0000007c      0x00000077
0x7fffffffdc60: 0x0000006e      0x0000005b      0x00000000      0x00000000
(gdb)
```

The key is the XOR of below arrays. Why? Because this is value of state before and after RoundKey(0,...) is called. The key is somewhat stored weirdly so had to resort to this.

```
(gdb) p $rdi
$9 = 140737488345952
(gdb) x/16x 140737488345952
0x7fffffffdb60: 0xd3e8b09a      0x5b587211      0xbaa21726      0x574062bc
0x7fffffffdb70: 0x00000000      0x00000000      0xffffdd08      0x00007fff
0x7fffffffdb80: 0x6ab31d30      0x807063d0      0xdc40ae74      0xbd6cc050
0x7fffffffdb90: 0x004c0000      0x00000000      0x745bf400      0xa38dca0d
(gdb) b *0x004033f9
Breakpoint 8 at 0x4033f9
(gdb) c
Continuing.

Breakpoint 8, 0x00000000004033f9 in ?? ()
(gdb) x/16x 140737488345952
0x7fffffffdb60: 0x29be09ed      0xd8c92035      0x8c81f45d      0xba8aa96d
0x7fffffffdb70: 0x00000000      0x00000000      0xffffdd08      0x00007fff
0x7fffffffdb80: 0x6ab31d30      0x807063d0      0xdc40ae74      0xbd6cc050
0x7fffffffdb90: 0x004c0000      0x00000000      0x745bf400      0xa38dca0d
(gdb)
```

Egg_params:

```
do {
  local_e8 = (ulong)bVar25;
  lVar13 = (ulong)(uVar14 & 0xff) * 6;
  cVar1 = (&DAT_004c4c80)[lVar13];
  cVar2 = (&DAT_004c4b91)[lVar13];
  bVar3 = (&DAT_004c4ca2)[lVar13];
  bVar4 = (&DAT_004c4bb3)[lVar13];
  bVar5 = (&DAT_004c4bf4)[lVar13];
  bVar6 = (&DAT_004c4bd5)[lVar13];
```

```
004c4c80 01             ??          01h
004c4c81 02             ??          02h
004c4c82 02             ??          02h
004c4c83 02             ??          02h
004c4c84 02             ??          02h
```

DAT_004c4ca2

004c4ca2	03	??	03h
004c4ca3	00	??	00h
004c4ca4	03	??	03h
004c4ca5	00	??	00h
004c4ca6	05	??	05h

DAT_004c4b91

004c4b91	01	??	01h
004c4b92	01	??	01h
004c4b93	03	??	03h
004c4b94	01	??	01h
004c4b95	02	??	02h

004c4bb3	00	??	00h
004c4bb4	00	??	00h
004c4bb5	03	??	03h
004c4bb6	04	??	04h
004c4bb7	02	??	02h

004c4bf4	01	??	01h
004c4bf5	01	??	01h
004c4bf6	04	??	04h
004c4bf7	04	??	04h
004c4bf8	00	??	00h

DAT_004c4bd5

004c4bd5	01	??	01h
004c4bd6	05	??	05h
004c4bd7	03	??	03h
004c4bd8	02	??	02h
004c4bd9	00	??	00h

The data in binary is wrong.
I had to manually print register r15 after each load instruction for egg params 1-5. And print eax for 6th egg param

5

Cipher:

```
1
2 void FUN_00401c60(byte *param_1,undefined8 param_2)
3
```

AddRoundKey call in cipher: Get 16 bytes of Roundkey from param_3

```
  FUN_00401b60(0,param_1,param_2);
  uVar52 = (uint)DAT_00409176;
```

```
2 void FUN_00401b60(byte param_1,byte *param_2,long param_3)
3
```

```
(gdb) r 12
Starting program: /mnt/c/Users/LEGION/OneDr
s/CS6570/Assignments/A5/AttackPhase/Fortify

Breakpoint 2, 0x0000000000401c85 in ?? ()
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000401c85 in ?? ()
(gdb) b *0x00401b60
Breakpoint 3 at 0x401b60
(gdb) c
Continuing.
```

```
(gdb) p $rdx
$3 = 140737488345920
```

```
(gdb) x/16x 140737488345920
0x7fffffffdb40: 0x97f89b71      0x03a90c3d      0xafc9bddd      0x91b982d4
0x7fffffffdb50: 0xdf79cd63      0xdcd0c15e      0x73197c83      0xe2a0fe57
0x7fffffffdb60: 0x84e12dda      0x5831ec84      0x2b289007      0xc9886e50
0x7fffffffdb70: 0xd73ce941      0x8f0d05c5      0xa42595c2      0x6dadfb92
```

Egg_params:

```
(gdb) x/10x 0x407000
0x407000:       0x02030202      0x00000104      0x00030002      0x00000204
0x407010:       0x02020001      0x00000304      0x02010203      0x00000801
0x407020:       0x02020102      0x00000903
```

How it is stored:

As clearly visible LSB bits of local_80, local_48, uVar53, uVar51, uVar50 and uVar17 are the egg_params. Combination of understanding logic from the if statement and bit shifts!

```
uVar17 = *(ulong *)(&DAT_00407000 + (long)
uVar51 = uVar17 >> 0x10;
uVar53 = uVar17 >> 0x18;
bVar98 = (&DAT_00407040)[param_1[2]];
bVar3 = (&DAT_00407040)[param_1[9]];
uVar50 = uVar17 >> 8;
bVar4 = (&DAT_00407040)[param_1[0xd]];
bVar5 = (&DAT_00407040)[param_1[5]];
bVar6 = (&DAT_00407040)[param_1[0xe]];
bVar7 = (&DAT_00407040)[param_1[1]];
bVar8 = (&DAT_00407040)[param_1[6]];
bVar9 = (&DAT_00407040)[param_1[10]];
bVar10 = (&DAT_00407040)[param_1[7]];
bVar11 = (&DAT_00407040)[param_1[3]];
bVar12 = (&DAT_00407040)[param_1[0xb]];
bVar13 = (&DAT_00407040)[param_1[0xf]];
uVar66 = (&DAT_00407040)[param_1[8]];
local_48 = (char)(uVar17 >> 0x28);
uVar67 = (&DAT_00407040)[param_1[0xc]];
bVar56 = local_48 != (char)local_74;
local_80 = (char)(uVar17 >> 0x20);

if ((local_80 == '\x04') && (local_48 == '\n')) {
  uVar49 = (ulong)DAT_00409176;
  DAT_00409176 = DAT_00409176 + 1;
  (&DAT_00409171)[uVar49] =
       param_1[(uVar17 & 0xff) + (uVar50 & 0xff) * 4] ^
       param_1[(uVar51 & 0xff) + (uVar53 & 0xff) * 4];
}
```

Compute_gf:
Eggs array is at 0x409171. The second thing is the compute_gf

```
if ((local_80 == '\x04') && (local_48 == '\n')) {
  uVar49 = (ulong)DAT_00409176;
  DAT_00409176 = DAT_00409176 + 1;
  (&DAT_00409171)[uVar49] =
       param_1[(uVar17 & 0xff) + (uVar50 & 0xff) * 4] ^
       param_1[(uVar51 & 0xff) + (uVar53 & 0xff) * 4];
}
DAT_00409170 = DAT_00409173 * 'a' + DAT_00409172 * -0x43;
```

7

```
void FUN_00401785(long param_1,long param_2)

{
  uint uVar1;
  uint uVar2;
  byte local_1c;
  byte local_1b;
  byte local_1a;
  byte local_19;
  uint local_c;

  for (local_c = 0; local_c < 4; local_c = local_c + 1) {
    *(undefined1 *)(param_1 + (ulong)(local_c << 2)) =
         *(undefined1 *)(param_2 + (ulong)(local_c << 2));
    *(undefined1 *)(param_1 + (ulong)(local_c * 4 + 1)) =
         *(undefined1 *)(param_2 + (ulong)(local_c * 4 + 1));
    *(undefined1 *)(param_1 + (ulong)(local_c * 4 + 2)) =
         *(undefined1 *)(param_2 + (ulong)(local_c * 4 + 2));
    *(undefined1 *)(param_1 + (ulong)(local_c * 4 + 3)) =
         *(undefined1 *)(param_2 + (ulong)(local_c * 4 + 3));
  }
```

Break by printing the key (param2 : rdi).

Egg_params by printing:

```
while( true ) {
  cVar1 = (&DAT_004ad340)[(long)(int)(uint)DAT_004ad330 * 6];
  cVar2 = (&DAT_004ad341)[(long)(int)(uint)DAT_004ad330 * 6];
  bVar3 = (&DAT_004ad342)[(long)(int)(uint)DAT_004ad330 * 6];
  bVar4 = (&DAT_004ad343)[(long)(int)(uint)DAT_004ad330 * 6];
  bVar5 = (&DAT_004ad344)[(long)(int)(uint)DAT_004ad330 * 6];
  bVar6 = (&DAT_004ad345)[(long)(int)(uint)DAT_004ad330 * 6];
```

```
Breakpoint 2, 0x0000000000402988 in ?? ()
(gdb) x/16x 0x4ad340
0x4ad340:       0x00020101      0x04020101      0x00000203      0x02020303
0x4ad350:       0x04080002      0x00010102      0x03010309      0x00000103
0x4ad360:       0x00000001      0x00000000      0x00350000      0x00000000
0x4ad370:       0x00350000      0x00000000      0x00000000      0x00000000
```

Compute_gf:
As we can see the compute_gf only depends on eggs[1] (0x4ad332) and eggs[2] (0x4ad333).
Additionally, the multiplier of eggs[1] seems to be -91.
Additionally, the multiplier of eggs[2] seems to be 26.

```
(gdb) set {int}0x4ad331 = 0x0
(gdb) set {char}0x4ad335 = 0x0
(gdb) set {char}0x4ad331 = 0x1
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000402ce6
(gdb) p $al
$5 = 0
```

```
(gdb) set {int}0x4ad331 = 0x0
(gdb) set {char}0x4ad335 = 0x0
(gdb) set {char}0x4ad332 = 0x1
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000402ce6 in ?? ()
(gdb) p $al
$4 = -91
```

```
(gdb) set {char}0x4ad335 = 0x0
(gdb) set {int}0x4ad331 = 0x0
(gdb) set {char}0x4ad332 = 0x2
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000402ce6
(gdb) p $al
$6 = 74
```

```
Breakpoint 1, 0x0000000000402ce1 in ?? ()
(gdb) set {int}0x4ad331 = 0x0
(gdb) set {char}0x4ad335 = 0x0
(gdb) set {char}0x4ad333 = 0x1
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000402ce6 in ?? ()
(gdb) p $al
$3 = 26
```

```
(gdb) set {int}0x4ad331 = 0x0
(gdb) set {char}0x4ad335 = 0x0
(gdb) set {char}0x4ad333 = 0x2
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000402ce6 in ?? ()
(gdb) p $a
$7 = void
(gdb) p $al
$8 = 52
```

```
Breakpoint 1, 0x0000000000402ce1 in ?? ()
(gdb) set {int}0x4ad331 = 0x0
(gdb) set {char}0x4ad335 = 0x0
(gdb) set {char}0x4ad334 = 0x1
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000402ce6 in ?? ()
(gdb) p $al
$2 = 0
```

```
Breakpoint 1, 0x0000000000402ce1 in ?? ()
(gdb) set {int}0x4ad331 = 0x0
(gdb) set {char}0x4ad335 = 0x1
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000402ce6 in ?? ()
(gdb) p $al
$1 = 0
```

Final:

```
uint8_t compute_gf(uint8_t* eggs){
        return eggs[1] * (-91) + eggs[2] * 26;
}
```

12

```
Files      gs "test"
(gdb) b *0x804a148
Breakpoint 1 at 0x804a148
(gdb) run
```

```
gs             0x63     99
(gdb) set $eax= 0x0
Python Exception <type 'ex
```

Bypass the ptrace by breaking before it and setting eax to 0

Key

x/16b 0xffffcf54

0xffffcf54: 0x9e 0xa4 0x21 0xb2 0xb1 0xa9 0xc1 0xe7
0xffffcf5c: 0x36 0xbd 0x37 0x41 0xd6 0x67 0xdd 0x6b

Check for keys in addroundkey(0)

For **egg params**
Add watch points for eggs and access from global

```
(gdb) x/6b 0x0804d0dc+18
0x804d0ee:      0x07    0x02    0x03    0x03    0x00    0x01
(gdb) c
Continuing.

Thread 1 "safe_main" hit Hardware watchpoint 7: {char} 0x0804d108

Old value = 0 '\000'
New value = -18 '\356'
Python Exception <type 'exceptions.NameError'> Installation error: gdb.execut
hwinders function is missing:
0x080498ac in ?? ()
(gdb) x/6b 0x0804d0dc+24
0x804d0f4:      0x08    0x01    0x00    0x01    0x00    0x03
(gdb)
```

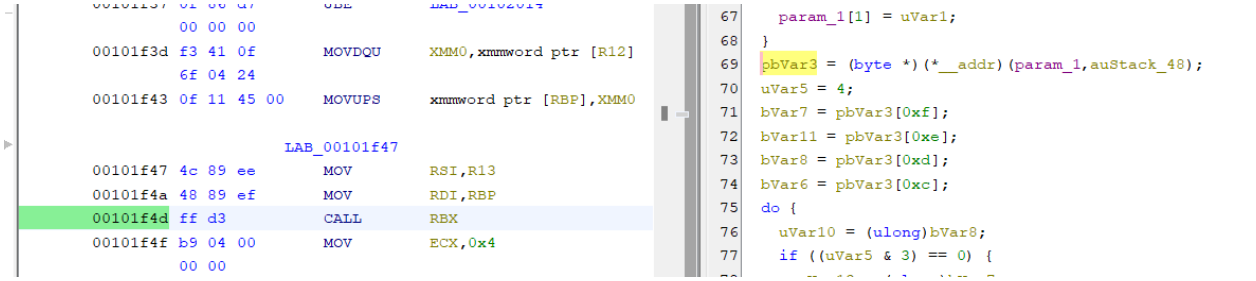
```
uint8_t egg_params[5][6] =
    {{ 2, 4, 3,2, 3, 0},
     {4, 3, 1, 2, 3, 2},
     {6, 1, 3, 3, 3, 2},       //6, ?
     {7, 2, 3, 3, 0, 1},
     {8, 1, 0, 1, 0, 3}};
```

For **compute_gf**
Set each egg to 0 or 1 when watch point triggers and then figure out the coeffs

```
uint8_t compute_gf(uint8_t* eggs) {


    return (uint8_t)(eggs[1] * 0x75 + eggs[3]);
}
```

13

Get pbVar3 and print 16 bytes, this is roundkey.



```
00 00 00
00101f3d f3 41 0f 6f 04 24   MOVDQU   XMM0,xmmword ptr [R12]
00101f43 0f 11 45 00         MOVUPS   xmmword ptr [RBP],XMM0

                     LAB_00101f47
00101f47 4c 89 ee            MOV      RSI,R13
00101f4a 48 89 ef            MOV      RDI,RBP
00101f4d ff d3               CALL     RBX
00101f4f b9 04 00 00 00      MOV      ECX,0x4
```

```
  param_1[1] = uVar1;
}
pbVar3 = (byte *)(*__addr)(param_1,auStack_48);
uVar5 = 4;
bVar7 = pbVar3[0xf];
bVar11 = pbVar3[0xe];
bVar8 = pbVar3[0xd];
bVar6 = pbVar3[0xc];
do {
  uVar10 = (ulong)bVar8;
  if ((uVar5 & 3) == 0) {
```

```
Breakpoint 6, 0x0000555555555f4f in ?? ()
(gdb) p $rax
$7 = 140737488345680
(gdb) x/4x 140737488345680
0x7fffffffda50: 0x3f25088b      0x1b21ddcb      0x0a9d9f5e      0xbc004259
(gdb)
```

This is the KeyExpansion, we can see pbVar3 being set here.

```
  uVar10 = (ulong)bVar8;
  if ((uVar5 & 3) == 0) {
    uVar13 = (ulong)bVar7;
    bVar7 = (&DAT_00104060)[
    bVar8 = (&DAT_00104060)[
    bVar11 = (&DAT_00104060)
    if ((uint)uVar5 < 0x18)
      bVar6 = (&DAT_0010405(
    }
    else {
      bVar6 = (&DAT_0010404(
    }
    bVar6 = bVar6 ^ (&DAT_0(
  }
  bVar6 = bVar6 ^ *pbVar3;
  bVar8 = bVar8 ^ pbVar3[1];
  uVar4 = (uint)uVar5 + 1;
  uVar5 = (ulong)uVar4;
  bVar11 = bVar11 ^ pbVar3[2
  bVar7 = bVar7 ^ pbVar3[3];
  pbVar3[0x10] = bVar6;
  pbVar3[0x11] = bVar8;
  pbVar3[0x12] = bVar11;
  pbVar3[0x13] = bVar7;
  pbVar3 = pbVar3 + 4;
} while (uVar4 != 0x2c);
```

14

There is a chain of files creates and execv called by process. Copied the final process into my directory to use with ghidra from /tmp/tmpbinvMEca0.

```
nt/c/Users/LEGION/OneDrive/Documents/raadhes/iitm/Courses/CS6570/Assignments
/A5/AttackPhase/Fortify/14$ strace ./safe_main 12
```

```
brk(0x564ecdbfc000)                     = 0x564ecdbfc000
openat(AT_FDCWD, "/tmp/tmpbinzxWvBw", O_RDWR|O_CREAT|O_EXCL, 0600) = 3
write(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\200\21\0\0\0\0\0\0"
..., 80384) = 80384
close(3)                                = 0
chmod("/tmp/tmpbinzxWvBw", 0755)        = 0
execve("/tmp/tmpbinzxWvBw", ["/tmp/tmpbinzxWvBw", "12"], 0x7ffebe973580 /* 2
```

```
openat(AT_FDCWD, "/tmp/tmpbin2LcHqy", O_RDWR|O_CREAT|O_EXCL, 0600) = 3
write(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 \22\0\0\0\0\0\0"..
, 62864) = 62864
close(3)                                = 0
chmod("/tmp/tmpbin2LcHqy", 0755)        = 0
execve("/tmp/tmpbin2LcHqy", ["/tmp/tmpbin2LcHqy", "12"], 0x7ffd85294d70 /*
8 vars */) = 0
brk(NULL)                               = 0x55ed88d7b000
```

```
openat(AT_FDCWD, "/tmp/tmpbinvMEca0", O_RDWR|O_CREAT|O_EXCL, 0600) = 3
write(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\340\21\0\0\0\0\0\0"
..., 39968) = 39968
close(3)                                = 0
chmod("/tmp/tmpbinvMEca0", 0755)        = 0
execve("/tmp/tmpbinvMEca0", ["/tmp/tmpbinvMEca0", "12"], 0x7ffcda882bb0 /* 2
8 vars */) = 0
```

Also had to NOP an exit, Was checking gdb.

Key:

Channa_mereya is the function of same functionality as KeyExpansion in the final executable that was generated. Just checked the first 16 bytes of roundkey at the end of this function.

```
Breakpoint 1, 0x0000555555555548c in channa_mereya ()
(gdb) p $rsi
$5 = 93824992269024
(gdb) p $rdi
$6 = 140737488345840
(gdb) finish
Run till exit from #0  0x0000555555555548c in channa_mereya ()
0x0000555555555882 in abhi_na_jao_ctx ()
(gdb) x/16x 140737488345840
0x7fffffffdaf0: 0xa1eb146c      0x17f36abe      0x53091992      0x4f89207a
0x7fffffffdb00: 0x7b6fb3da      0x6c8cd96b      0x3f95c0f6      0x701ce08c
```

Egg_params:

The instruction 88 45 fa puts the uVar1 value at $rbp - 6.

```
00101e31 89 c7           MOV        EDI,EAX
00101e33 e8 38 fe        CALL       abhi_decode
         ff ff
00101e38 88 45 fa        MOV        byte ptr [RBP + local_e],AL
00101e3b 0f b6 05        MOVZX      EAX,byte ptr [k]
         fe 71 00 00
00101e42 0f b6 c0        MOVZX      EAX,AL
```

```
16  while( true ) {
17    cVar1 = abhi_decode(lizard[(long)(int)(uint)k * 6]);
18    cVar2 = abhi_decode(lizard[(long)(int)(uint)k * 6 + 1]);
19    bVar3 = abhi_decode(lizard[(long)(int)(uint)k * 6 + 2]);
20    bVar4 = abhi_decode(lizard[(long)(int)(uint)k * 6 + 3]);
21    bVar5 = abhi_decode(lizard[(long)(int)(uint)k * 6 + 4]);
22    bVar6 = abhi_decode(lizard[(long)(int)(uint)k * 6 + 5]);
```

Let us check these values immediately after all 6 are set in each iteration.

```
Breakpoint 4, 0x0000555555555933 in abhi_sub_bytes ()
(gdb) finish
Run till exit from #0  0x0000555555555933 in abhi_sub_bytes ()
0x0000555555555f4b in Cipher ()
(gdb) x/6b $rbp-6
0x7fffffffda0a: 0x01    0x01    0x02    0x00    0x03    0x01
(gdb) c
Continuing.

Breakpoint 4, 0x0000555555555933 in abhi_sub_bytes ()
(gdb) finish
Run till exit from #0  0x0000555555555933 in abhi_sub_bytes ()
0x0000555555555f4b in Cipher ()
(gdb) x/6b $rbp-6
0x7fffffffda0a: 0x02    0x01    0x00    0x00    0x00    0x02
(gdb) c
Continuing.

Breakpoint 4, 0x0000555555555933 in abhi_sub_bytes ()
(gdb) finish
Run till exit from #0  0x0000555555555933 in abhi_sub_bytes ()
0x0000555555555f4b in Cipher ()
(gdb) x/6b $rbp-6
0x7fffffffda0a: 0x03    0x02    0x03    0x02    0x00    0x01
(gdb) c
Continuing.
```

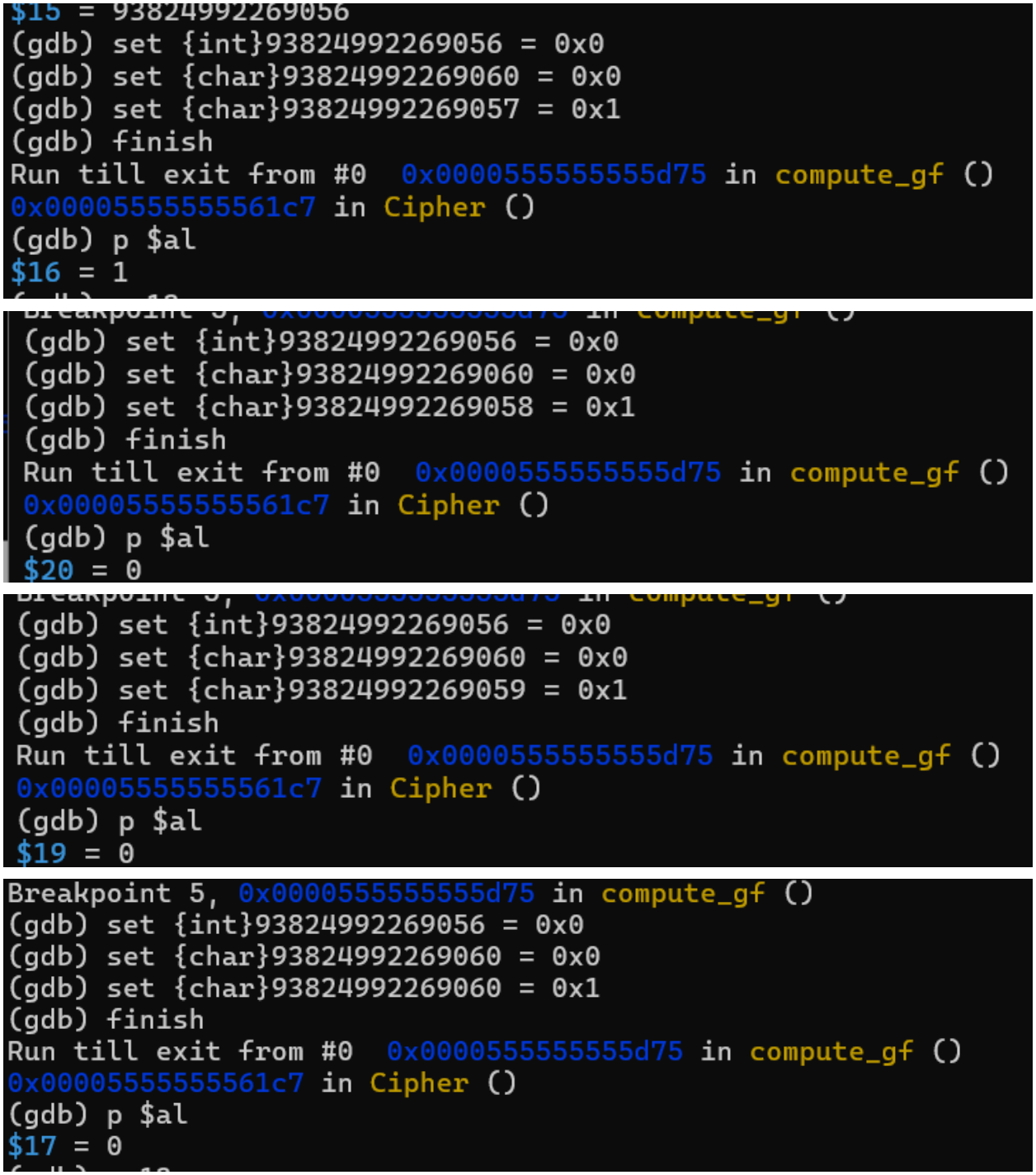
```
Breakpoint 4, 0x0000555555555933 in abhi_sub_bytes ()
(gdb) finish
Run till exit from #0  0x0000555555555933 in abhi_sub_bytes ()
0x00005555555555f4b in Cipher ()
(gdb) x/6b $rbp-6
0x7fffffffda0a: 0x05    0x03    0x01    0x02    0x00    0x01
(gdb) c
Continuing.

Breakpoint 4, 0x0000555555555933 in abhi_sub_bytes ()
(gdb) finish
Run till exit from #0  0x0000555555555933 in abhi_sub_bytes ()
0x00005555555555f4b in Cipher ()
(gdb) x/6b $rbp-6
0x7fffffffda0a: 0x05    0x03    0x01    0x02    0x00    0x01
(gdb) c
Continuing.

Breakpoint 4, 0x0000555555555933 in abhi_sub_bytes ()
(gdb) finish
Run till exit from #0  0x0000555555555933 in abhi_sub_bytes ()
0x00005555555555f4b in Cipher ()
(gdb) x/6b $rbp-6
0x7fffffffda0a: 0x08    0x01    0x00    0x00    0x02    0x01
(gdb) c
Continuing.
```

Compute_gf:
Rdi has the address of eggs in the call of compute_gf

```
}
global_flag = compute_gf(&eggs);
```

```
Cannot access memory a
(gdb) p $rdi
$15 = 938249922690 56
```

Likely that eggs[1] term present in compute_gf.



More checks:
Possible that eggs[1] + 20 * eggs[1] * eggs[4] is the compute_gf based on below patterns.

```
Breakpoint 5, 0x0000555555555d75 in compute_gf ()
(gdb) set {char}93824992269057 = 0x1
(gdb) set {char}93824992269058 = 0x2
(gdb) set {char}93824992269059 = 0x3
(gdb) set {char}93824992269060 = 0x4
(gdb) finish
Run till exit from #0  0x0000555555555d75 in compute_gf ()
0x00005555555561c7 in Cipher ()
(gdb) p $al
$22 = 81
```

```
Breakpoint 5, 0x0000555555555d75 in compute_gf ()
(gdb) set {char}93824992269057 = 0x4
(gdb) set {char}93824992269058 = 0x3
(gdb) set {char}93824992269059 = 0x2
(gdb) set {char}93824992269060 = 0x1
(gdb) finish
Run till exit from #0  0x0000555555555d75 in compute_gf ()
0x00005555555561c7 in Cipher ()
(gdb) p $al
$23 = 84
```

```
(gdb) set {char}93824992269057 = 0x2
(gdb) set {char}93824992269058 = 0x3
(gdb) set {char}93824992269059 = 0x4
(gdb) set {char}93824992269060 = 0x5
(gdb) finish
Run till exit from #0  0x0000555555555d75 in compute_
0x00005555555561c7 in Cipher ()
(gdb) p (uint8_t)($al)
$29 = 202 '\312'
(gdb)
```

15

Key found from addroundkey (0), ctx/roundkey passed as param first 16 bytes.

```
uint8_t key[16] = {
      165, 229, 242, 78, 100, 91, 203, 104,
  121, 158, 68, 47, 112, 207, 109, 58


  };
```

```
uint8_t compute_gf(uint8_t* eggs) {

   return (-4 * eggs[1] +1) * eggs[3];
}
uint8_t egg_params[5][6] =
   {{ 1, 1, 3,0, 0, 3},
    {3, 1, 2, 0, 0, 0},
    {4, 3, 0, 1, 3, 3},      //6, ?
    {6, 2, 3, 3, 3, 2},
    {8, 3, 3, 3, 2, 3}};
```

Egg params found by putting watch on eggs[k] and then looking at the global variable when egg gets updated.

Compute_gf:
By observation by setting eggs.
Also note that the global flag is being finally set in this function which is called in main. There is a decoy compute_gf in the cipher function.

```
2  void FUN_004022da(undefined8 param_1,long param_2,ulong param_
3
4  {
5    char cVar1;
6    ulong uVar2;
7    ulong local_10;
8
9    uVar2 = (ulong)DAT_004050bc;
10   DAT_004050bc = DAT_004050bc + 1;
11   printf("%s: ",param_1,uVar2);
12   for (local_10 = 0; local_10 < param_3; local_10 = local_10 +
13     printf("%02x ",(ulong)*(byte *)(local_10 + param_2));
14   }
15   putchar(10);
16   if (DAT_004050bc == 2) {
17     cVar1 = FUN_00401c43(&DAT_004050b5);
18     DAT_004050ba = cVar1 * DAT_004050b8;
19   }
20   return;
21 }
22
```

```
FUN_004022da("Ciphertext:",&local_f8,0x10);
printf("Egg 0 : 0x%02x\n",(ulong)DAT_004050b5);
printf("Global Flag: 0x%02x\n",(ulong)DAT_004050ba);
```

The function FUN_00401c43 seems to return eggs[1] * -4 + 1. This is then multiplied by eggs[3] which is DAT_004050b8. Some of working to find FUN_00401c43.:

```
Breakpoint 2 at 0x40236f
(gdb) set {int} 4050b5 = 0x0
Invalid number "4050b5".
(gdb) set {int} 0x4050b5 = 0x0
(gdb) set {char} 0x4050b9 = 0x0
(gdb) c
Continuing.

Breakpoint 2, 0x000000000040236f in ??
(gdb) p $al
$1 = 1
```

```
Breakpoint 1, 0x0000000000402365 in ?
(gdb) set {int} 0x4050b5 = 0x0
(gdb) set {char} 0x4050b9 = 0x0
(gdb) set {char} 0x4050b6 = 0x1
(gdb) c
Continuing.

Breakpoint 2, 0x000000000040236f in ?
(gdb) p $al
$2 = -3
(gdb) r 12
```

16

Key found easly from addrounkey(0)

```
uint8_t key[16] = { 182, 138, 5, 196, 189, 16, 138, 17,
    222, 13, 63, 174, 65, 129, 70, 12};
```

Visible easily in code/data

```
uint8_t egg_params[5][6] =
    {{ 4, 1, 1,2, 0, 2},
     {5, 2, 2, 2, 3, 0},
     {6, 4, 0, 0, 3, 2},
     {8, 4, 3, 2, 2, 2},
     {9, 3, 3, 1, 1, 0}};
uint8_t compute_gf(uint8_t* eggs) {

    return (uint8_t)(eggs[4] * eggs[3] * 0x6B - eggs[2]);
}
```

Compute_gf also found in decompiled code

17

Key: From keyExpansion.

```
Breakpoint 1, 0x00000000004017e0 in KeyExpansion ()
(gdb) p $rdi
$1 = 140737488345680
(gdb) p $rsi
$2 = 140737488345632
(gdb) x/16x 140737488345632
0x7fffffffda20: 0x012e52fc      0x36b67171      0x1a86587a      0xd1c6baa2
```

```
uint8_t key[16] = {252, 82, 46, 1, 113, 113, 182, 54,
    122, 88, 134, 26, 162, 186, 198, 209};
```

Egg_params:
There seems to not be direct array perhaps. We print the value returned by the get_egg_param function. Screenshot for first row gdb printing is shown.

```
cVar47 = get_egg_param(uVar55,0);
cVar48 = get_egg_param(uVar55 & 0xffffffff,1);
bVar49 = get_egg_param(uVar55 & 0xffffffff,2);
uVar62 = (ulong)bVar49;
bVar49 = get_egg_param(uVar55 & 0xffffffff,3);
uVar56 = (ulong)bVar49;
bVar49 = get_egg_param(uVar55 & 0xffffffff,4);
uVar64 = (ulong)bVar49;
bVar49 = get_egg_param(uVar55 & 0xffffffff,5);
```

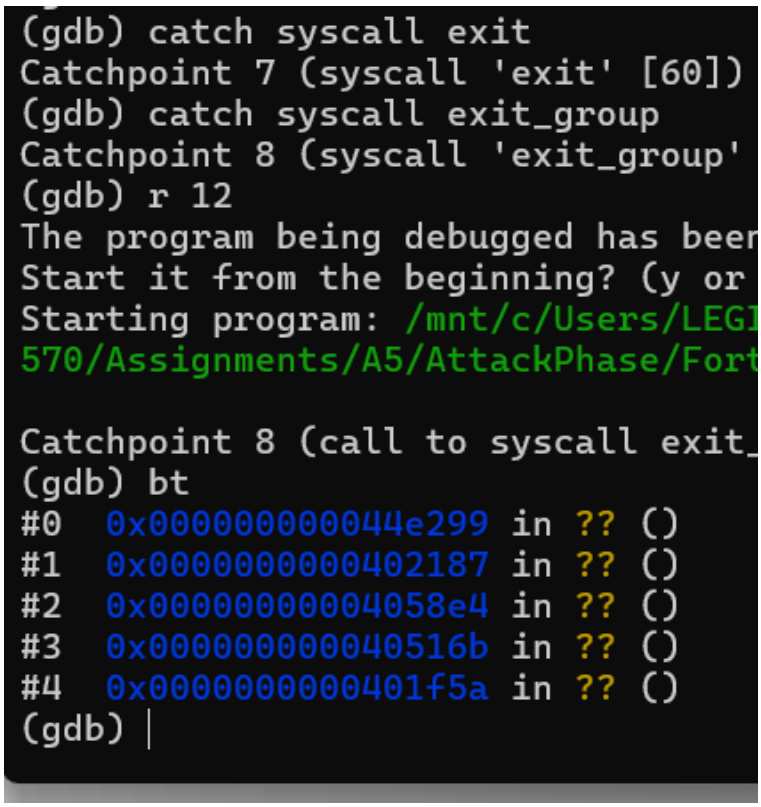
```
Breakpoint 1, 0x0000000000401a80 in get_egg_param ()
(gdb) finish
Run till exit from #0  0x0000000000401a80 in get_egg_param ()
0x0000000000401bfc in Cipher ()
(gdb) p $al
$7 = 1
(gdb) c
Continuing.

Breakpoint 1, 0x0000000000401a80 in get_egg_param ()
(gdb) finish
Run till exit from #0  0x0000000000401a80 in get_egg_param ()
0x0000000000401c0d in Cipher ()
(gdb) p $al
$8 = 3
(gdb) c
Continuing.

Breakpoint 1, 0x0000000000401a80 in get_egg_param ()
(gdb) finish
Run till exit from #0  0x0000000000401a80 in get_egg_param ()
0x0000000000401c1c in Cipher ()
(gdb) p $al
$9 = 2
(gdb) c
Continuing.

Breakpoint 1, 0x0000000000401a80 in get_egg_param ()
(gdb) finish
Run till exit from #0  0x0000000000401a80 in get_egg_param ()
0x0000000000401c2d in Cipher ()
(gdb) p $al
$10 = 2
(gdb) c
Continuing.

Breakpoint 1, 0x0000000000401a80 in get_egg_param ()
(gdb) finish
Run till exit from #0  0x0000000000401a80 in get_egg_param ()
0x0000000000401c3d in Cipher ()
(gdb) p $al
$11 = 2
(gdb) c
Continuing.

Breakpoint 1, 0x0000000000401a80 in get_egg_param ()
(gdb) finish
Run till exit from #0  0x0000000000401a80 in get_egg_param ()
0x0000000000401c4e in Cipher ()
(gdb) p $al
$12 = 1
(gdb) c
```

Compute_gf:

```
int compute_gf(byte *param_1)

{
  byte bVar1;
  byte bVar2;

  bVar1 = param_1[4];
  bVar2 = param_1[3];
  if ((((uint)(*param_1 ^ param_1[1]) + (uint)bVar1 & 1) == 0) && ((bVar2 & 1) == 0)) {
    return (bVar2 + 0x11) * (param_1[2] - 8) - (uint)bVar1;
  }
  return (uint)bVar2 * 0x23 * (uint)param_1[2] * 0x5c - (uint)bVar1;
}
```

19

Dealing with gdb. The gdb trap is hidden somewhere before main! The process crashes even if we set breakpoint as main and run.



Place of exit:
We remove this call by setting the instructions to NOPS.

```
void FUN_0044e270(void)

{
  syscall();
  syscall();
  do {
                    /* W
  } while( true );
}
```

```
                    if ((local_c != 0) && (local_10 != 0)) {
                                        /* WARNING: Subroutine d
                      FUN_0044e270(1);
                    }
```

Go into FUN_004042e0

```
    local_e7 = 0;
    FUN_00402540(local_d8,&local_108);
    FUN_00402630(local_d8,local_f8);
                    /* WARNING: Subroutine does n
    FUN_004042e0(param_1,param_2);
  }
  FUN_00412980("Invalid Usage!!");
  FUN_00412980("Usage: ./encrypt <plain_text>");
  return 1;
```

KeyExpansion is FUN_004021A0 and Cipher is FUN_0040360

```
  FUN_00412b60(10);
  FUN_004021a0(local_e8,&local_118);
  FUN_00403600(local_e8,puVar3);
  FUN_0040bf70(&DAT_0048701f,"Ciphertext:");
  do {
    uVar1 = *puVar3;
    puVar3 = puVar3 + 1;
    FUN_0040bf70("%02x ",uVar1);
  } while (puVar3 != local_f8);
  FUN_00412b60(10);
  FUN_0040bf70("Egg 0 : 0x%02x\n",DAT_004b323f);
  FUN_0040bf70("Global Flag: 0x%02x\n",DAT_004b323e);
```

Key:

```
Breakpoint 6, 0x0000000000403600 in ?? ()
(gdb) p $rsi
$5 = 140737488345504
(gdb) p $rdi
$6 = 140737488345536
(gdb) x/16x 140737488345536
0x7fffffffd9c0: 0x9043cd1f      0xbdace3fb      0x2589d190      0x7ec810d2
```

Egg_params:

```
cVar2 = (&DAT_004b3220)[lVar1];
local_a2 = (&DAT_004b3221)[lVar1];
bVar3 = (&DAT_004b3222)[lVar1];
bVar4 = (&DAT_004b3223)[lVar1];
bVar5 = (&DAT_004b3224)[lVar1];
bVar6 = (&DAT_004b3225)[lVar1];
```

```
(gdb) x/8x 0x4b3220
0x4b3220:       0x01010301      0x01020101      0x03020003      0x01030303
0x4b3230:       0x01040003      0x00030002      0x01010105      0x00000202
```

Compute_gf

```
if ((local_a2 == '\x04') && (cVar2 == '\n')) {
  uVar13 = (ulong)DAT_004b3210;
  DAT_004b3210 = DAT_004b3210 + 1;
  (&DAT_004b323f)[uVar13] =
       param_2[(ulong)bVar6 + (ulong)bVar5 * 4]
       ;
}
DAT_004b323e = FUN_00402580(&DAT_004b323f);
```

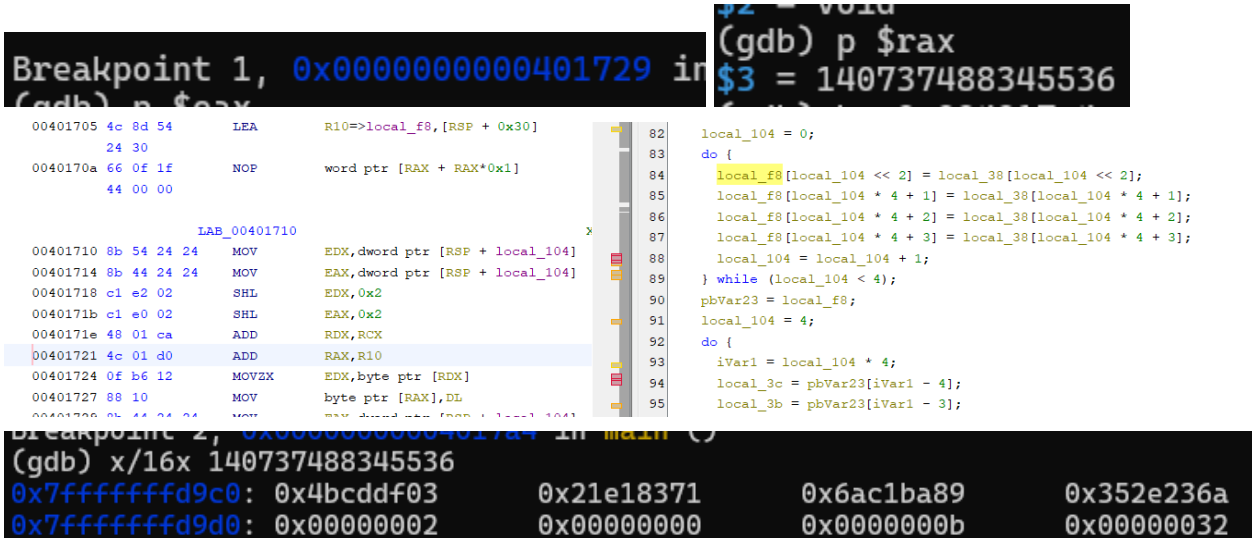
```
int FUN_00402580(long param_1)

{
  return (uint)*(byte *)(param_1 + 2) * 0x15 + (uint)*(byte *)(param_1 + 3) * 0x1c;
}
```

22

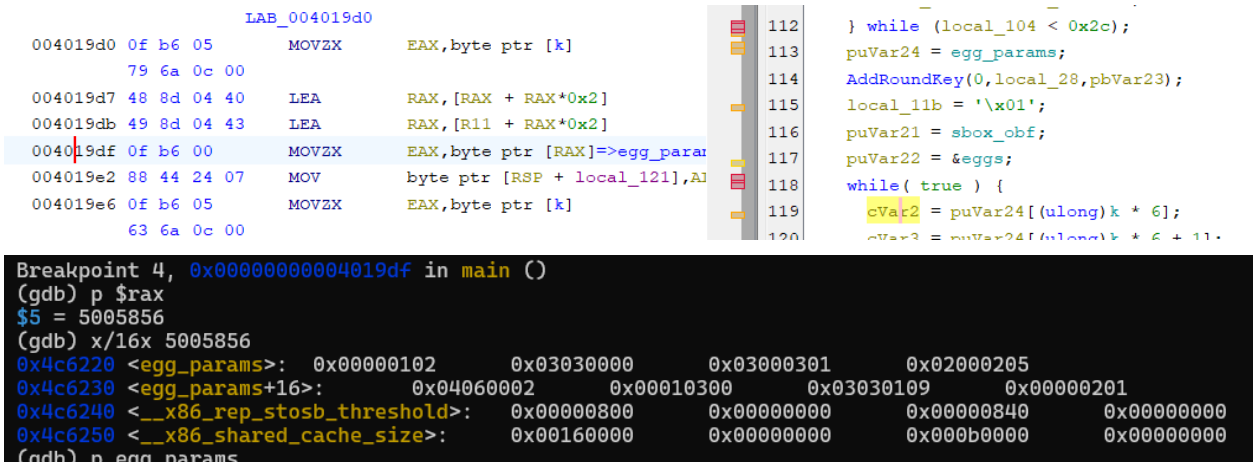
24

To get key:

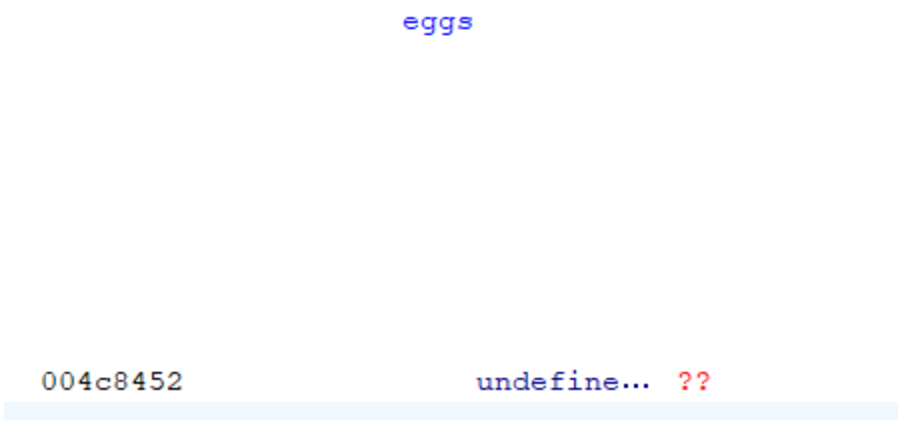
First find address local_38 which is obviously the key.



Egg_params:
Easily print egg_params like this



Compute_gf:



```
global_flag = (DAT_004c8453 * '\v' - DAT_004c8456) + DAT_004c8455 * -0x44;
```

Clear to get the function

25

27

[I changed jnz in a if statement to jmp near debugger check also so that binary runs in gdb]
Main function:

```
2 undefined8 FUN_0040102f(int param_1,long param_2)
```

Key:

```
Time of execution: 0.000038 seconds
[Inferior 1 (process 1545) exited normally]
(gdb) b *0x00401890
Breakpoint 1 at 0x401890
(gdb) r 12
Starting program: /mnt/c/Users/LEGION/OneDrive/Documents/raadhes/iitm/Courses/
CS6570/Assignments/A5/AttackPhase/Fortify/27/safe_main1 12
Plaintext :: 31 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Breakpoint 1, 0x0000000000401890 in ?? ()
(gdb) p $edi
$1 = -9216
(gdb) p $rdi
$2 = 140737488346112
(gdb) x/16x 140737488346112
0x7fffffffdc00: 0x3f693c2e      0x744ded5c      0x6f7150f3      0xd954ada8
0x7fffffffdc10: 0x000003e8      0x00000000      0xffffdf79      0x00007fff
0x7fffffffdc20: 0x1f8bfbff      0x00000000      0x00000064      0x00000000
0x7fffffffdc30: 0x00000000      0x00000000      0x00000000      0x00000000
(gdb) c
```

31

Key:

```
Breakpoint 1, 0x0000000000400ef6 in AddRoundKey ()
(gdb) p $rdx
$1 = 140737488345856
(gdb) x/16x 140737488345856
0x7fffffffdb00: 0xa28aef82      0xd07795f2      0x15eac7e2      0xe69068f5
0x7fffffffdb10: 0x44048fc6      0x94731a34      0x8199ddd6      0x6709b523
0x7fffffffdb20: 0x62818e11      0xf6f29425      0x776b49f3      0x1062fcd0
0x7fffffffdb30: 0x124b24a5      0xe4b9b080      0x93d2f973      0x83b005a3
(gdb) c
```

Egg_params

```
Breakpoint 1, 0x0000000000401bac in Cipher ()
(gdb) p $rax
$1 = 7127184
(gdb) x/16x 7127184
0x6cc090 <egg_params>:  0x00000201      0x04040202      0x03010002      0x03020306
0x6cc0a0 <egg_params+16>:       0x02080300      0x01030300      0x00030409      0x00000203
0x6cc0b0 <_dl_tls_static_size>: 0x00001180      0x00000000      0x004a2fb8      0x00000000
0x6cc0c0 <__exit_funcs>:        0x006ce180      0x00000000      0x00000000      0x00000000
(gdb)
```

Compute_gf:

Values of iVar4, iVar1(index), iVar5, iVar2(index) and iVar4.

```
Breakpoint 3, 0x0000000000401690 in compute_gf ()
(gdb) p $rax
$3 = 34
(gdb) p $eax
$4 = 34
(gdb) b *0x0040169c
Breakpoint 4 at 0x40169c
(gdb) c
Continuing.

Breakpoint 4, 0x000000000040169c in compute_gf ()
(gdb) p $eax
$5 = 1
(gdb) b *0x004016ba
Breakpoint 5 at 0x4016ba
(gdb) c
Continuing.

Breakpoint 5, 0x00000000004016ba in compute_gf ()
(gdb) p $eax
$6 = 64
(gdb) b *x004016cc
No symbol table is loaded.  Use the "file" command.
(gdb) b *0x004016cc
Breakpoint 6 at 0x4016cc
(gdb) c
Continuing.

Breakpoint 6, 0x00000000004016cc in compute_gf ()
(gdb) p $eax
$7 = 3
(gdb) b *0x004016eb
Breakpoint 7 at 0x4016eb
(gdb) c
Continuing.
```

```
Breakpoint 7, 0x00000000004016eb in compute_gf ()
(gdb) p $eax
$8 = 1
```

```
int compute_gf(long param_1)

{
  byte bVar1;
  byte bVar2;
  byte bVar3;
  int iVar4;
  int iVar5;

  iVar4 = _____();
  bVar1 = ____();
  bVar1 = *(byte *)(param_1 + (ulong)bVar1);
  iVar5 = _();
  bVar2 = ___(3,3);
  bVar2 = *(byte *)(param_1 + (ulong)bVar2);
  bVar3 = ____();
  return (uint)*(byte *)(param_1 + (ulong)bVar3) + iVar4 * (uint)bVar1 + iVar5 * (uint)bVar2;
}
```

33

Key found

```
uint8_t key[16] = {
        28, 120, 25, 88, 167, 237, 130, 112, 127, 167, 102, 121, 97, 192, 12, 155


  };
```

Break just before 44 wala loop after looking at ghidra

For egg params theres loop of 1->4 and 4->7 one repeats
Set breaks after xor command for all locala,b,c,d,e,f

```
uint8_t egg_params[5][6] =
   {{ 1, 4, 1,2, 2, 1},
    {2, 2, 1, 3, 2, 3},
    {5, 2, 1, 3, 3, 0},
    {7, 3, 3, 3, 3, 3},
    {7, 3, 3, 3, 3, 3}};
```

Use set {char} 0x6d0e98 and the next 5 addr to set them one by one and check coeffs

```
uint8_t compute_gf(uint8_t* eggs) {

    return (uint8_t)(eggs[1] * 8);
}
```

35

```
uint8_t key[16] = {
            183,35,137,67,53,104,61,203,204,111,208,141,126,124,155,242
    };
Found from cipher &ctx first 16 bytes

Egg params found from global variable

Compute flag not hidden
```

```
uint8_t egg_params[5][6] =
    {
    {1, 4, 3, 0, 1, 0},
    {3, 2, 3, 2, 0, 3},
    {4, 3, 0, 0, 0, 2},
    {7, 1, 0, 1, 2, 2},
    {9, 3, 0, 0, 1, 1}
};
uint8_t compute_gf(uint8_t* eggs) {

    return ( eggs[1] * '['  + eggs[3] *(0-0x49) - eggs[4]);
}
```

37

Breakpoint 6 is the start of cipher. I print the array location of state array.
Breakpoint 4 is before the AddRoundkey(0, …) call. Print the state before this.
Breakpoint 5 is sometime after this, but before any other state changes. The xor of these values is the key.

```
Breakpoint 6, 0x0000000000403e8f in ?? ()
(gdb) p $rsi
$6 = 140737488345808
(gdb) p $rdi
$7 = 140737488345760
(gdb) c
Continuing.

Breakpoint 4, 0x0000000000403eb1 in ?? ()
(gdb) x/4x 140737488345760
0x7fffffffdaa0: 0x00003231      0x00000000      0x00000000      0x00000000
(gdb) c
Continuing.

Breakpoint 5, 0x0000000000403fc4 in ?? ()
(gdb) x/4x 140737488345760
0x7fffffffdaa0: 0xf79b71fa      0xa0ae33cd      0xc4d51a36      0x7b55ab33
(gdb)
```

Egg_params:

Interesting.

Egg[0] is calculated into this array. Hence the first row of egg_params seems to be correct.

```
cVar1 = (&DAT_004c3170)[(long)(int)(uint)DAT_004c5310 * 6];
cVar2 = (&DAT_004c3171)[(long)(int)(uint)DAT_004c5310 * 6];
bVar3 = (&DAT_004c3172)[(long)(int)(uint)DAT_004c5310 * 6];
bVar4 = (&DAT_004c3173)[(long)(int)(uint)DAT_004c5310 * 6];
bVar5 = (&DAT_004c3174)[(long)(int)(uint)DAT_004c5310 * 6];
bVar6 = (&DAT_004c3175)[(long)(int)(uint)DAT_004c5310 * 6];
FUN_0040397f(param_1);
```

```
(gdb) x/6b 0x4c3170
0x4c3170:       0x01    0x02    0x01    0x02    0x02    0x02
(gdb)
```

However the other rows of egg_params seem to be wrong.

```
FUN_00413230("Egg 0 : 0x%02x\n",DAT_004c64d0);
```

```
cVar1 = (&DAT_004c3170)[(long)(int)(uint)DAT_004c5310 * 6];
cVar2 = (&DAT_004c3171)[(long)(int)(uint)DAT_004c5310 * 6];
bVar3 = (&DAT_004c3172)[(long)(int)(uint)DAT_004c5310 * 6];
bVar4 = (&DAT_004c3173)[(long)(int)(uint)DAT_004c5310 * 6];
bVar5 = (&DAT_004c3174)[(long)(int)(uint)DAT_004c5310 * 6];
bVar6 = (&DAT_004c3175)[(long)(int)(uint)DAT_004c5310 * 6];
FUN_0040397f(param_1);
if ((cVar2 == '\x01') && (cVar1 == DAT_004c63c0)) {
  uVar7 = (uint)DAT_004c5310;
  DAT_004c5310 = DAT_004c5310 + 1;
  (&DAT_004c64d0)[(int)uVar7] =
       *(byte *)((long)(int)(uint)bVar5 * 4 + param_1 + (long)(int)(uint)bVar6) ^
       *(byte *)((long)(int)(uint)bVar3 * 4 + param_1 + (long)(int)(uint)bVar4);
}
```

On further observation, we found compute_gf being called at the end of Cipher:
This means that other eggs seem to be stored somewhere else! Where are they being calculated?

```
DAT_004c5311 = FUN_00403d99(&DAT_004c64d0);
```

```
7 int FUN_00403d99(void)
8
9 {
0   int local_c;
1   int local_8;
2
3   for (local_8 = 0; local_8 < 3; local_8 = local_8 + 1) {
4   }
5   for (local_c = 0; local_c < 5; local_c = local_c + 1) {
6   }
7   return (uint)DAT_004c3168 * -0x45 + (uint)DAT_004c3161 * 0x14;
8 }
```

On further inspection where actual eggs are being set. These are being set in the functions represented as Subbytes, ShiftRows and MixColumns.
Here we have 0x4c3168 being set based on state[0][2]^state[1][3] in round 9. As this is just after subbytes functionality the op = 1.

```
void FUN_0040397f(long param_1)

{
  byte local_2;
  byte local_1;

  for (local_1 = 0; local_1 < 4; local_1 = local_1 + 1) {
    for (local_2 = 0; local_2 < 4; local_2 = local_2 + 1) {
      *(undefined *)((long)(int)(uint)local_2 * 4 + param_1 + (lon
           (&DAT_00498020)
           [(int)(uint)*(byte *)((long)(int)(uint)local_2 * 4 + pa
           )]
      ;
    }
  }
  if (DAT_004c63c0 == '\t') {
    DAT_004c3168 = *(byte *)(param_1 + 2) ^ *(byte *)(param_1 + 7);
  }
  return;
}
```

Here we have 0x4c3161 being set based on state[3][1]]^state[0][2] in round 2. As this is just after ShiftRows functionality the op = 2.
Here we have 0x4c3162 being set based on state[0][2]^state[0][0] in round 7. As this is just after ShiftRows functionality the op = 2.

```
void FUN_00403a27(byte *param_1)

{
  byte bVar1;

  bVar1 = param_1[1];
  param_1[1] = param_1[5];
  param_1[5] = param_1[9];
  param_1[9] = param_1[0xd];
  param_1[0xd] = bVar1;
  bVar1 = param_1[2];
  param_1[2] = param_1[10];
  param_1[10] = bVar1;
  bVar1 = param_1[6];
  param_1[6] = param_1[0xe];
  param_1[0xe] = bVar1;
  bVar1 = param_1[3];
  param_1[3] = param_1[0xf];
  param_1[0xf] = param_1[0xb];
  param_1[0xb] = param_1[7];
  param_1[7] = bVar1;
  if (DAT_004c63c0 == '\x02') {
    DAT_004c3161 = param_1[0xd] ^ param_1[2];
  }
  if (DAT_004c63c0 == '\a') {
    DAT_004c3162 = param_1[2] ^ *param_1;
  }
```

Here we have 0x4c3167 being set based on state[2][3]^state[2][2] in round 8. As this is just after ShiftRows functionality the op = 3.

```
FUN_00403b6b                                    XR
     ENDBR64
     SUB        RSP,0x18
     MOV        qword ptr [RSP]=>local_18,RDI
     MOV        byte ptr [RSP + local_1],0x0
     JMP        LAB_00403d65

LAB_00403b81                                    XR
     MOVZX      EDX,byte ptr [RSP + local_1]
```

```
    *(byte *)(param_1 + 1 + (long)(int)(uint)local_1 * 4) =
         *(byte *)(param_1 + 1 + (long)(int)(uint)local_1 * 4) ^ bVar2
    bVar2 = FUN_00402f48(*(byte *)(param_1 + 3 + (long)(int)(uint)loca
                         *(byte *)(param_1 + 2 + (long)(int)(uint)loca
    *(byte *)(param_1 + 2 + (long)(int)(uint)local_1 * 4) =
         *(byte *)(param_1 + 2 + (long)(int)(uint)local_1 * 4) ^ bVar2
    bVar3 = FUN_00402f48(*(byte *)(param_1 + 3 + (long)(int)(uint)loca
    *(byte *)(param_1 + 3 + (long)(int)(uint)local_1 * 4) =
         *(byte *)(param_1 + 3 + (long)(int)(uint)local_1 * 4) ^ bVar3
  }
  if (DAT_004c63c0 == '\b') {
    DAT_004c3167 = *(byte *)(param_1 + 0xb) ^ *(byte *)(param_1 + 10);
  }
```

Serializing in order of rounds we have:

```
uint8_t egg_params[5][6]
{{1 ,2, 1, 2, 2, 2},
{2, 2, 0, 2, 3, 1},
{7, 2, 0, 0, 0, 2},
{8, 3, 2, 3, 2, 2},
{9, 1, 0, 2, 1, 3}};
```

Compute_gf:
The first one is eggs[4] and the second mem location is eggs[1] !

```
7 int FUN_00403d99(void)
8
9 {
0   int local_c;
1   int local_8;
2
3   for (local_8 = 0; local_8 < 3; local_8 = local_8 + 1) {
4   }
5   for (local_c = 0; local_c < 5; local_c = local_c + 1) {
6   }
7   return (uint)DAT_004c3168 * -0x45 + (uint)DAT_004c3161 * 0x14;
8 }
```

39

Creates a file called add and likely calls it:

```
mprotect(0x4b4000, 20480, PROT_READ)    = 0
openat(AT_FDCWD, "add", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
fstat(3, {st_mode=S_IFREG|0777, st_size=0, ...}) = 0
```

```
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITE
unlink("add")                           = 0
exit_group(0)                           = ?
```

Remove the unlink so we can work with “add” file.
Change this to NOP instructions in safe_main.

```
0040195a 48 89 df       MOV     RDI=>DAT_00488047,RBX
0040195d e8 0e 6f       CALL    FUN_00428870
         02 00
00401962 31 c0          XOR     EAX,EAX
00401964 48 81 c4       ADD     RSP,0x100278
         78 02 10 00
```

```
22      FUN_00411770(uVar2);
23      FUN_004284d0(&DAT_00488047,0x1ed);
24      FUN_0040bf30(&stack0xfffffffffeffde8);
25      FUN_00428870(&DAT_00488047);
26      return 0;
27  }
```

```
Catchpoint 4 (call to syscall unlink), 0x000000000042887b in ?? ()
(gdb) bt
#0  0x000000000042887b in ?? ()
#1  0x0000000000401962 in ?? ()
#2  0x0000000000408dd8 in ?? ()
#3  0x000000000040afa0 in ?? ()
#4  0x0000000000401735 in ?? ()
(gdb)
```

Key:

Print the Roundkey 16 bytes from param_2 of Cipher function.

```
void FUN_00101350(long param_1,undefined8 param_2)

{
  ulong uVar1;
  byte bVar2;
  byte bVar3;
  byte bVar4;
  byte bVar5;
  byte bVar6;
  byte bVar7;
  byte bVar8;

  FUN_00101750(0,param_1);
```

```
Breakpoint 3, 0x0000555555555350 in ?? ()

(gdb) p $rdi

$1 = 140737488345856
(gdb) p $rsi

$2 = 140737488345904
(gdb) x/4x 140737488345904

0x7fffffffdb30: 0x0aede374      0x67cc63c6      0x41937c9d      0xf41b502f
```

Egg_params:

The value stored in DAT_00104040 is 27.

```
00101396 48 8d 0d e3 2c 00 00    LEA         RCX,[DAT_00104080]
```

```
uVar1 = (ulong)DAT_00104070;
bVar8 = (&DAT_00104080)[uVar1 * 6] ^ DAT_00104040;
bVar7 = (&DAT_00104081)[uVar1 * 6] ^ DAT_00104040;
bVar2 = (&DAT_00104082)[uVar1 * 6] ^ DAT_00104040;
bVar5 = (&DAT_00104083)[uVar1 * 6] ^ DAT_00104040;
bVar3 = (&DAT_00104084)[uVar1 * 6] ^ DAT_00104040;
bVar6 = DAT_00104040 ^ (&DAT_00104085)[uVar1 * 6];
```

```
(gdb) p $rcx
$2 = 93824992247936
(gdb) x/16x 93824992247936
0x555555558080: 0x24272526      0x23242424      0x26252426      0x26272622
0x555555558090: 0x25212727      0x27272426      0x2724232e      0x00002425
0x5555555580a0: 0x00000000      0x00000000      0x00000000      0x00000000
```

Compute_gf:
This is the function given by Ghidra.
The return value is a 8 bit integer. So just the LSB 8 bits matter.
CONCAT31 is an operation that Concatenates 3 bytes of its first parameter and first byte of second parameter.

```
undefined4 FUN_00101310(long param_1)

{
  int iVar1;

  if (*(byte *)(param_1 + 1) != 0) {
    _DAT_00104074 = _DAT_00104074 + 10;
  }
  iVar1 = (uint)*(byte *)(param_1 + 1) * 0x59;
  return CONCAT31((int3)((uint)iVar1 >> 8),
                  ((char)iVar1 + *(char *)(param_1 + 2) * -0x33) - *(char *)(param_1 + 3));
}
```

Observing from above the compute_gf can be simpified to:

```
uint8_t compute_gf(uint8_t* eggs){
    return (eggs[1] * 0x59 + eggs[2] * (-0x33)) - eggs[3];
}
```

42

Remove gdb check by removing jz instruction.

```
0040f1b5 e8 56 c4        CALL        FUN_0044b610
         03 00
0040f1ba 48 83 f8 ff     CMP         RAX,-0x1
0040f1be 0f 84 44        JZ          LAB_0040f708
         05 00 00
0040f1c4 c6 44 24        MOV         byte ptr [RSP + local_d8],
         20 1f
0040f1c9 c6 44 24        MOV         byte ptr [RSP + local_d7],
```

```
34  lVar2 = FUN_0044b610(0,0,1,0);
35  if (lVar2 == -1) {
36    uStack_100 = 0x40f714;
37    FUN_00417090("Running inside GDB");
38    uStack_100 = 0x40f71e;
39    FUN_00415c30(0);
40  }
```

The Cipher function seems to be the FUN_0040b971. This can be found by checking the references to egg[0] and global_flag which are visible in main on ghidra application.

Key:
I have added breakpoint at the cipher function, however it seems it is called 50 times with different RoundKeys.
Was not sure how to crack so just tried checking first 16 bytes of Roundkey in each all. The 9th call matched the ciphertext.

Egg_params: - only first seems to be correct
In the same iteration of cipher above we checked the values of egg_params here.

```
while( true ) {
  uVar11 = (ulong)DAT_004b7331;
  cVar1 = (&DAT_004b7370)[uVar11 * 6];
  cVar2 = (&DAT_004b7371)[uVar11 * 6];
  bVar3 = (&DAT_004b7372)[uVar11 * 6];
  bVar4 = (&DAT_004b7373)[uVar11 * 6];
  bVar5 = (&DAT_004b7374)[uVar11 * 6];
  bVar6 = (&DAT_004b7375)[uVar11 * 6];
  pbVar9 = param_1;
```

```
(gdb) x/16x 0x4b7370
0x4b7370:       0x02030401      0x04030000      0x00030201      0x03010204
0x4b7380:       0x02060100      0x00010302      0x00010109      0x00000302
0x4b7390:       0x00000000      0x00000000      0x00000000      0x00000000
0x4b73a0:       0x00000040      0x00000000      0x00000000      0x00000000
(gdb)
```

Compute_gf:
Used method of setting eggs before the function and checking value after. Last call to compute_gf likely in main just before printing global_flag.
On setting only one egg[i] = 1 and checking for eggs[1…4] we find that only egg[4] has an affect of (-1) coefficient.
This may mean the other eggs are in multiplication. A bit of experimenting gives:

```
uint8_t compute_gf(uint8_t* eggs){
        return 64 * eggs[1] * eggs[3] - eggs[4];
}
```

45

KeyExpansion function: void FUN_00401c7c. Key is taken from here.
Cipher function: void FUN_004045f4. This is not the actual cipher function.

ACTUAL cipher: FUN_00403de0
Called from void FUN_00404e77 at bottom.

Set break here and get each row of egg_params
0x0000000000404222 in ?? ()
(gdb) x/16x $rbp-0x78
0x7fffffffdb18: **0x02000301** 0x01bb**0202** 0x00000008 0x00000000
Compute_gf

```
int FUN_0040262f(long param_1)

{
  return (uint)*(byte *)(param_1 + *DAT_004b1b90) * *DAT_004b1b38 * *DAT_004b1ba8 *
         (uint)*(byte *)(param_1 + *DAT_004b1bc0) - (uint)*(byte *)(param_1 + *DAT_004b1bd0);
}
```

49

53

This binary is very similar to 55. Key is derived by printing the roundkey in expansion.
Eggs is at 0x4c5400
Compute_gf.

```
    }
    DAT_004c4370 = (DAT_004c5402 * 'G' + DAT_004c5401 * -0x4e) - DAT_004c5403;
    return;
```

55

Round == (cvar2 - 1) something, will need to decrement this egg_param finally.
Subtracts other values from egg_params too. → Simple.

Eggs stored with xored with 0x5a.
For some weird reason AddRoundKey function is inlined into cipher.
Compute_gf:

```
 }
 DAT_00605440 = (DAT_00605472 ^ 0x5a) * '_' + (DAT_00605474 ^ 0x5a) * -4;
 return;
```

3 function calls that detect gdb are set to NOP so that we can run on gdb.

```
void FUN_004026d0(void)

{
  long lVar1;

  lVar1 = ptrace(PTRACE_TRACEME,0,0,0);
  if (lVar1 != -1) {
    return;
  }
                    /* WARNING: Subroutine does not return */
  exit(1);
}
```

Roundkey start:
0x7fffffffdcb0: 0x106ecd63 0x5e966d17 0xc6d9f5fc 0x04a8420d
There are weird if statements that always evaluate to false.

56

Key:
Simply take XOR of state before and after AddRoundKey(0,...) is called.

```
2  void FUN_0040390a(long param_1,undefined8 param_2)
3
4  {
5    char cVar1;
6    char cVar2;
7    byte bVar3;
8    byte bVar4;
9    byte bVar5;
10   byte bVar6;
11   uint uVar7;
12   char local_f;
13
14   FUN_00402be3();
15   FUN_00402721(0,param_1,param_2);
16   local_f = '\x01';
```

```
Breakpoint 14, 0x0000000000403934 in ?? ()
(gdb) x/16x $rsi
0x7fffffffdb40: 0x00003231      0x00000000      0x00000000      0x00000000
```

```
Breakpoint 16, 0x0000000000403939 in ?? ()
(gdb) x/16x $rsi
0x7fffffffdb40: 0xdcef577b      0xff4cde43      0x9f959b0c      0x750d9ae6
```

61

keys not hidden

```
uint8_t key[16] = {
    108, 20, 105, 255, 176, 113, 178, 130, 167, 197, 67, 116, 5, 102, 18, 118
    };
```

Egg params not hidden

```
uint8_t egg_params[5][6] =
{{1,4,1,1,0,3},
{2,2,2,1,2,2}
,{4,2,0,2,2,0},
{6,2,0,3,1,0}
,{7,4,3,3,1,0}};
```

```
uint8_t compute_gf(uint8_t* eggs) {

    return eggs[1] + eggs[4] * 0xA8;
}
```

```
uint compute_gf(byte *param_1)

{
  uint uVar1;

  if (opaque_value == 0) {
    uVar1 = (uint)(param_1[1] ^ *param_1);
  }
  else {
    uVar1 = (uint)param_1[1] + (uint)param_1[4] * -0x58;
  }
  return uVar1;
}
```

63

Looking at function parameter in init_ctx
Rsi 0x7fffffffddb0 140737488346544
rdi 0x7fffffffdd00

0x7fffffffddb0: 0x5f 0x60 0x7c 0xce 0xdf 0x8e 0x26 0x56
0x7fffffffddb8: 0xf5 0x62 0x94 0xb4 0x7a 0x24 0x3b 0xb6

```
uint8_t key[16] = {
  0x5f ,0x60 ,0x7c ,0xce ,0xdf ,0x8e ,0x26 ,0x56 ,0xf5 ,0x62 ,0x94 ,0xb4 ,0x7a ,0x24 ,0x3b ,0xb6

  };
```

Egg params not hidden

```
uint8_t egg_params[5][6] =
{{1,1,0,3,3,3},
{2,4,3,1,1,2}
,{5,1,1,0,1,3},
{6,3,2,3,0,0}
,{8,4,3,1,2,3}};
```

Compute gf not hidden

```
uint8_t compute_gf(uint8_t* eggs){
        return eggs[1] +eggs[2]* 34 + eggs[4]*17 + eggs[4]*2;
}
```

65

Note: The Cipher function which is visible easily seems to be a decoy.
This is the function reached by looking at normal cipher function pathway:

```
2 void FUN_00103785(undefined8 *param_1,undefined8 *param_2)
```

However the real cipher function seems to be this: (It is called in the above function).
We found it by checking references to the global_flag which is printed in main.

```
  else if (local_1a == 7) {
    FUN_001027d8(param_1);
  }
```

Key:
Take XOR of state before and after functionality of AddRoundKey(0,....)

```
(gdb) p $rsi
$2 = 3
(gdb) x/16x 140737488345696
0x7fffffffda60: 0x00003231      0x00000000      0x00000000      0x00000000
0x7fffffffda70: 0xffff0000      0x00007fff      0xf7fdeddb      0x00007fff
0x7fffffffda80: 0x293e7215      0x00591f95      0xbe3800de      0x0061002a
0x7fffffffda90: 0x293e7215      0x00591f95      0xbe3800de      0x0061002a
(gdb) b *0x0000555555556a53
Breakpoint 3 at 0x555555556a53
(gdb) c
Continuing.

Breakpoint 3, 0x0000555555556a53 in ?? ()
(gdb) x/16x 140737488345696
0x7fffffffda60: 0xb9ef79d4      0x7a01b230      0xa6c2cd37      0x41fce8d4
0x7fffffffda70: 0xffff0000      0x00007fff      0xf7fdeddb      0x00007fff
0x7fffffffda80: 0x293e7215      0x00591f95      0xbe3800de      0x0061002a
0x7fffffffda90: 0x293e7215      0x00591f95      0xbe3800de      0x0061002a
```

Egg_params:
Print the contents of array pointed to by local_f8 just after the FUN_001025BB call.

```
  FUN_001025bb(local_f8);
  local_11 = 1;
  local_12 = 0;
LAB_00102977:
  local_25 = local_f8[(long)(int)(uint)local_12 * 6];
  local_26 = local_f8[(long)(int)(uint)local_12 * 6 + 1];
  local_13 = local_f8[(long)(int)(uint)local_12 * 6 + 2];
  local_14 = local_f8[(long)(int)(uint)local_12 * 6 + 3];
  local_27 = local_f8[(long)(int)(uint)local_12 * 6 + 4];
  local_28 = local_f8[(long)(int)(uint)local_12 * 6 + 5];
```

```
Breakpoint 5, 0x000055555556696f in ?? ()
(gdb) x/16x 140737488344992
0x7fffffffd7a0: 0x03020201      0x04030203      0x02020001      0x03010404
0x7fffffffd7b0: 0x04070202      0x00020000      0x05030309      0xdccd0503
```

Compute_gf:
Always zero, however this is due to values of egg_params.
We can infer probable compute_gf by running it with different egg inputs.

```
Breakpoint 6, 0x0000555555557650 in ??
(gdb) set {int} 93824992260658 = 0x0
(gdb) set {char} 93824992260662 = 0x1
(gdb) c
Continuing.

Breakpoint 7, 0x0000555555557655 in ??
(gdb) p 4al
Invalid number "4al".
(gdb) p $al
$12 = 1
```

```
Breakpoint 6, 0x0000555555557650 in ??
(gdb) set {int} 93824992260658 = 0x0
(gdb) set {char} 93824992260661 = 0x1
(gdb) set {char} 93824992260662 = 0x1
(gdb) c
Continuing.

Breakpoint 7, 0x0000555555557655 in ??
(gdb) p $al
$11 = 33
```

```
Breakpoint 6, 0x0000555555557650 in ?? ()
(gdb) set {int} 93824992260658 = 0x0
(gdb) set {char} 93824992260661 = 0x1
(gdb) set {char} 93824992260662 = 0x3
(gdb) c
Continuing.

Breakpoint 7, 0x0000555555557655 in ?? ()
(gdb) p $al
$15 = 99
```

Probable.

```
uint8_t compute_gf(uint8_t* eggs){
        return eggs[4] * eggs[3] * 32 + eggs[4];
}
```

68

Key found at the end of function and putting break at ret and looking at memory from registers.

```
uint8_t key[16] =
          {
    107, 255, 125, 233, 99, 27, 138, 127,
    237, 190, 191, 172, 57, 9, 111, 247
};
```

Found in data section

```
uint8_t egg_params[5][6] =
    {
    {1, 4, 1, 3, 1, 0},
```

```
    {3, 3, 0, 0, 2, 3},
    {7, 4, 1, 2, 3, 0},
    {8, 4, 1, 1, 1, 0},
    {9, 3, 1, 0, 2, 3}
};
```

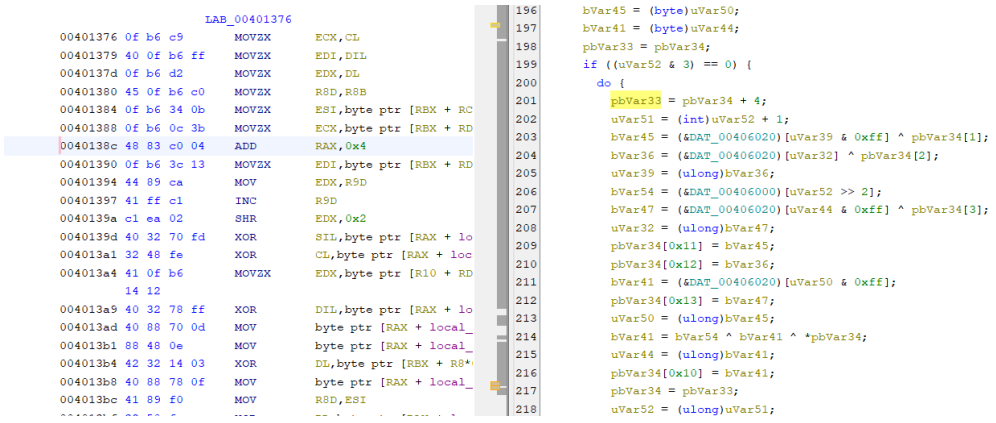
69

72

77

Roundkey is from here.

```
(gdb) b *0x40138c
Breakpoint 2 at 0x40138c
(gdb) c
Continuing.

Breakpoint 2, 0x000000000040138c in ?? ()
(gdb) p $rax
$3 = 140737488346080
(gdb) si
0x0000000000401390 in ?? ()
(gdb) p $rax
$4 = 140737488346084
(gdb) x/16x 140737488346080
0x7fffffffdbe0: 0x6c494fb2      0x2b57a475      0x53340fc4      0xa38100d5
0x7fffffffdbf0: 0x00000000      0x00000000      0x00401d5e      0x00000000
0x7fffffffdc00: 0x00000000      0x00000000      0x000003e8      0x00000000
0x7fffffffdc10: 0x000003e8      0x00000000      0x000003e8      0x00000000
(gdb) q
A debugging session is active.
```



Egg_params:

```
Breakpoint 1, 0x000000000040 1a3a in ?? ()
(gdb) x/16b 0x4081a0
0x4081a0:       1       4       1       2       0       1       2       4
0x4081a8:       1       0       3       2       6       4       2       2
(gdb) x/32b 0x4081a0
0x4081a0:       1       4       1       2       0       1       2       4
0x4081a8:       1       0       3       2       6       4       2       2
0x4081b0:       3       3       7       1       2       3       1       7
0x4081b8:       8       4       0       3       3       3       0       0
(gdb)
```

Observe bvar59, cvar2 to cvar5

```
cVar2 = (&DAT_004081a1)[lVar1];
bVar29 = (&DAT_00406020)[local_179];
local_174 = CONCAT31(local_174._1_3_,bVar45);
bVar27 = (&DAT_00406020)[uVar32 & 0xff];
uVar21 = local_170._1_7_;
local_170 = CONCAT71(local_170._1_7_,bVar54);
bVar3 = (&DAT_004081a2)[lVar1];
bVar4 = (&DAT_004081a3)[lVar1];
bVar5 = (&DAT_004081a4)[lVar1];
bVar6 = (&DAT_004081a5)[lVar1];
uVar22 = local_168._1_7_;
local_168 = (undefined1 *)CONCAT71(local_168._1_7_,bVar38)
bVar7 = (&DAT_00406020)[local_14d];
bVar48 = (&DAT_00406020)[(byte)local_14c];
bVar28 = (&DAT_00406020)[uVar46 & 0xff];
local_180 = CONCAT31(local_180._1_3_,bVar48);
local_188 = CONCAT31(local_188._1_3_,bVar28);
auVar60 = vpinsrb_avx(ZEXT416((uint)local_168),(uint)bVar7
auVar14 = vpinsrb_avx(ZEXT416((uint)local_170),(uint)bVar4
bVar8 = (&DAT_00406020)[uVar50 & 0xff];
auVar15 = vpinsrb_avx(ZEXT416(local_174),(uint)bVar27,1);
auVar16 = vpinsrb_avx(ZEXT416(local_178),(uint)bVar29,1);
bVar59 = (&DAT_004081a0)[lVar1] == local_14e;
```

Compute_gf:

```
                    DAT_00408180

00408180                undefine... ??

                    EGGS

00408181                undefine... ??

                    DAT_00408182
00408182                undefine... ??

                    DAT_00408183
00408183                undefine... ??
00408184                ??          ??
00408185                ??          ??
00408186                ??          ??
```

```
476       uStack_113._0_1_ = local_179;
477       bStack_111 = local_18c;
478       if ((cVar2 == '\x04') && (bVar59)) {
479         local_170 = (ulong)bVar6;
480         uVar40 = (ulong)bVar36;
481         bVar36 = bVar36 + 1;
482         local_168 = auStack_40 + (ulong)bVar3 * 4;
483         (&EGGS)[uVar40] = local_168[(ulong)bVar4 - 0xe0] ^ state
484       }
485       local_14e = local_14e + '\x01';
486       pbVar34 = pbVar34 + 0x10;
487     }
488     _state = vpxorq_avx512vl(_state,local_60);
489     DAT_00408180 = DAT_00408182 * '8' + DAT_00408183 * -0xc;
490     FUN_00401e80("Ciphertext:",state);
491     FUN_004024c9("Egg 0 : 0x%02x\n",EGGS);
492     FUN_004024c9("Global Flag: 0x%02x\n",DAT_00408180);
493     uVar35 = 0;
494   }
495   return uVar35;
496 }
497
```

```
int malloc_sys_time(long param_1)

{
  return (uint)DAT_004ae190 * (uint)*(byte *)(param_1 + (ulong)DAT_004ae193) +
         (uint)strspnng * (uint)*(byte *)(param_1 + (ulong)DAT_004ae191) +
         (uint)DAT_004ae18f * (uint)*(byte *)(param_1 + (ulong)DAT_004ae192);
}
```

79

Roundkey being set here.
I am running 79 on wsl. The addresses are modified for some reason.
0x100d38 is being shown as 0x0000555555400d38 for gdb purposes.

```
00 00 00
00100d38 0f b6 15       MOVZX      EDX,byte ptr [DAT_00303
         20 23 20 00
00100d3f 88 84 24       MOV        byte ptr [RSP + ROUNDKE
         90 00 00 00
00100d46 0f b6 05       MOVZX      EAX,byte ptr [DAT_00303
         04 23 20 00
00100d4d 44 88 84       MOV        byte ptr [RSP + Stack[-
         24 9c 00
         00 00
00100d55 40 88 bc       MOV        byte ptr [RSP + Stack[-
         24 9d 00
         00 00
00100d5d 40 88 b4       MOV        byte ptr [RSP + Stack[-
         24 9e 00
         00 00
00100d65 88 94 24       MOV        byte ptr [RSP + Stack[-
         9f 00 00 00
00100d6c 88 84 24       MOV        byte ptr [RSP + ROUNDKE
         91 00 00 00
00100d73 0f b6 05       MOVZX      EAX,byte ptr [DAT_00303
         d8 22 20 00
00100d7a 88 84 24       MOV        byte ptr [RSP + ROUNDKE
         92 00 00 00
00100d81 0f b6 05       MOVZX      EAX,byte ptr [DAT_00303
```

```
150   prctl(4,0);
151   ROUNDKEYONLY = ROUNDKEYONLYONLY;
152   uVar48 = 4;
153   ROUNDKEYONLYONLY[0] = DAT_00303050;
154   bStack_dc = DAT_0030305c;
155   bStack_db = DAT_0030305d;
156   bStack_da = DAT_0030305e;
157   bStack_d9 = DAT_0030305f;
158   ROUNDKEYONLYONLY[1] = DAT_00303051;
159   ROUNDKEYONLYONLY[2] = DAT_00303052;
160   ROUNDKEYONLYONLY[3] = DAT_00303053;
161   ROUNDKEYONLYONLY[4] = DAT_00303054;
162   ROUNDKEYONLYONLY[5] = DAT_00303055;
163   ROUNDKEYONLYONLY[6] = DAT_00303056;
164   uStack_e1 = DAT_00303057;
165   uStack_e0 = DAT_00303058;
166   uStack_df = DAT_00303059;
167   uStack_de = DAT_0030305a;
168   uStack_dd = DAT_0030305b;
169   RoundKey = ROUNDKEYONLY;
170   bVar34 = DAT_0030305f;
171   bVar39 = DAT_0030305e;
172   loopindex = DAT_0030305d;
173   bVar49 = DAT_0030305c;
174   do {
```

```
Plaintext :: 31 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Breakpoint 4, 0x0000555555400d1c in ?? ()
(gdb) si
0x0000555555400d24 in ?? ()
(gdb) p $rcx
$3 = 140737488345872
(gdb) x/16x 0x0000555555603050
0x555555603050: 0x0144ccf0      0xc9ac6787      0x2d16f298      0xbda755d7
0x555555603060: 0x00000000      0x00000000      0x00000000      0x00000000
0x555555603070: 0x00000000      0x00000000      0x00000000      0x00000000
0x555555603080: 0x00000000      0x00000000      0x00000000      0x00000000
(gdb) q
A debugging session is active.
```

Eggs_params:

```
local_112 = 0;
local_118 = 0x64642223;
lVar26 = uVar48 * 6;
cVar1 = (&DAT_00303020)[lVar26];
cVar45 = (&DAT_00303021)[lVar26];
bVar2 = (&DAT_00303022)[lVar26];
bVar3 = (&DAT_00303023)[lVar26];
bVar4 = (&DAT_00303024)[lVar26];
bVar5 = (&DAT_00303025)[lVar26];
local_114 = 0x6e65;
puVar33 = &local_118;
do {
```

```
.
(No debugging symbols found in ./safe_main1)
(gdb) b *0x0000555555400d38
Breakpoint 1 at 0x555555400d38
(gdb) r 12
Starting program: /mnt/c/Users/LEGION/OneDrive/Documents/raadhes/iitm
s/CS6570/Assignments/A5/AttackPhase/Fortify/79/safe_main1 12
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.s
Plaintext :: 31 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Breakpoint 1, 0x0000555555400d38 in ?? ()
(gdb) x/16b 0x0000555555603020
0x555555603020: 1       3       1       3       2       0       2
0x555555603028: 0       2       0       1       6       3       3
(gdb) x/30b 0x0000555555603020
0x555555603020: 1       3       1       3       2       0       2
0x555555603028: 0       2       0       1       6       3       3
0x555555603030: 2       1       7       2       1       2       2
0x555555603038: 8       2       0       3       1       3
(gdb)
```

Compute_gf:

```
eggs3copy = (uint)EGGS3;
FUN_00101dd0();
FUN_00101d90();
FUN_00101dd0();
FUN_00101d90();
eggs4copy = EGGS4;
```

```
FUN_00101d90();
eggs3copycopy = (char)eggs3copy;
cVar1 = eggs3copycopy * -2;
FUN_00101dd0();
FUN_00101d90();
FUN_00101dd0();
FUN_00101d90();
FUN_00101dd0();
FUN_00101d90();
BAKABAKABAKA = CONCAT31(BAKABAKABAKA._1_3_,
                        ((char)(eggs3copy << 4) + cVar1) * eggs4copy * 'F' + eggs3copycopy);
FUN_00101d80();
FUN_00101dd0();
FUN_00101d90();
DAT_0030303f = (undefined1)BAKABAKABAKA;
```

Observe above that the higher bits of BAKABAKABAKA ARE USELESS. Just stupid obfuscation. We can write this compute_gf: (can be simplified a bit more, will do this for final submission)

```
uint8_t compute_gf(uint8_t* eggs){
    return (eggs[3] * 16 + eggs[3] * (-2)) *
    eggs[4] * ((uint8_t)('F')) + eggs[3];
}
```

83

We can see execv is being called on a file.

```
snprintf(local_58,0x40,"/proc/self/fd/%d",(ulong)__fd)
__argv = (char **)malloc((long)(param_1 + 1) << 3);
if (__argv == (char **)0x0) {
  perror("malloc failed");
  close(__fd);
}
else {
  *__argv = local_58;
  for (local_74 = 1; local_74 < param_1; local_74 = loc
    __argv[local_74] = *(char **)(param_2 + (long)local
  }
  __argv[param_1] = (char *)0x0;
  execv(local_58,__argv);
  perror("execv failed");
```

Check the syscalls done for the file.

```
rad@DESKTOP-SLPVP4D:/mnt/c/Users/LEGION/OneDrive/Documents/raadhes/iitm/Cour
gnments/A5/AttackPhase/Fortify/83$ strace ./safe_main
execve("./safe_main", ["./safe_main"], 0x7ffd8168a050 /* 28 vars */) = 0
```

```
munmap(0x7f87cd0e3000, 806912)          = 0
execve("/proc/self/fd/3", ["/proc/self/fd/3"], 0x7ffda36e5988 /* 28 vars */) = 0
brk(NULL)                               = 0x5164000
```

Get the file from filesystem:

```
(No debugging symbols found in ./safe_ma
(gdb) b execv
Breakpoint 1 at 0x727610
(gdb) r 12
Starting program: /mnt/c/Users/LEGION/O
ments/A5/AttackPhase/Fortify/83/safe_ma

Breakpoint 1, 0x0000000000727610 in exec
(gdb) info inferiors
  Num  Description       Connection
* 1    process 13247     1 (native)
s/iitm/Courses/CS6570/Assignments/A5/Att
(gdb)
```

```
rad@DESKTOP-SLPVP4D:/mnt/c/Users/LEGION/OneDrive/Documents/raadhes/iitm/
ignments/A5/AttackPhase/Fortify/83$ cp /proc/13247/fd/3 ./image3
rad@DESKTOP-SLPVP4D:/mnt/c/Users/LEGION/OneDrive/Documents/raadhes/iitm/
```

The file is somewhat corrupted, change the sh_entsize to non-zero value. This is also causing some error in ghidra with division. Set this field to non-zero value:

printf '\x18\x00\x00\x00' | dd of=./image3_repaired bs=1 seek=$((0xc463c)) conv=notrunc.

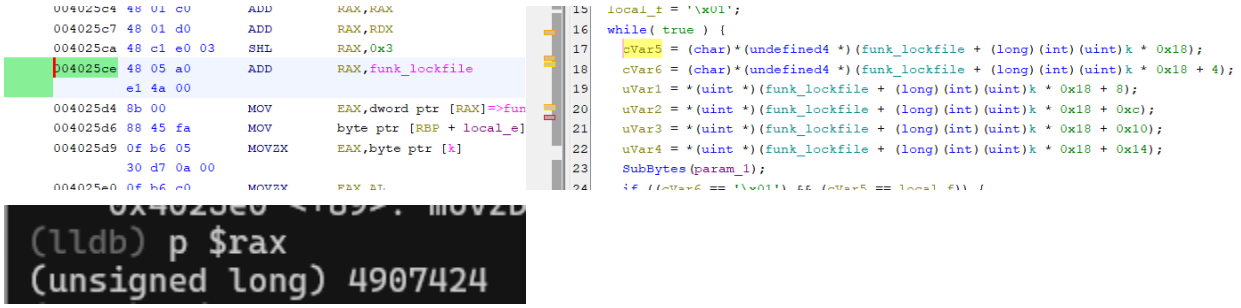
```
rad@DESKTOP-SLPVP4D:/mnt/c/Users/LEGION/OneDrive/Documents/raadhes/iitm
gnments/A5/AttackPhase/Fortify/83$ readelf -h image3_repaired
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 03 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - GNU
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x401b70
  Start of program headers:          64 (bytes into file)
  Start of section headers:          803712 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         10
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
readelf: Error: Section 10 has invalid sh_entsize of 0
```

Key:
The binary does not run on gdb, use the second most popular debugger I could find. Lldb.
Print the key in the KeyExpansion function. Rsi register contains the address of key at the start of the KeyExpansion function - param_2. It seems the key bytes of spaced bu 8 bytes.

```
(lldb) p $rsi
(unsigned long) 4907264
```

```
(lldb) x/32x 4907264
0x004ae100: 0x000000da 0x00000000 0x000000e5 0x00000000
0x004ae110: 0x000000d8 0x00000000 0x00000050 0x00000000
0x004ae120: 0x000000d9 0x00000000 0x000000b8 0x00000000
0x004ae130: 0x00000002 0x00000000 0x0000008e 0x00000000
0x004ae140: 0x000000c3 0x00000000 0x000000f9 0x00000000
0x004ae150: 0x00000073 0x00000000 0x00000040 0x00000000
0x004ae160: 0x000000b8 0x00000000 0x0000005c 0x00000000
0x004ae170: 0x00000083 0x00000000 0x00000084 0x00000000
```

Egg_params:
Print the values present at the address of funk_lockfile. Note that the value are located in spacing of 4 bytes.




```
(lldb) p $rax
(unsigned long) 4907424
```

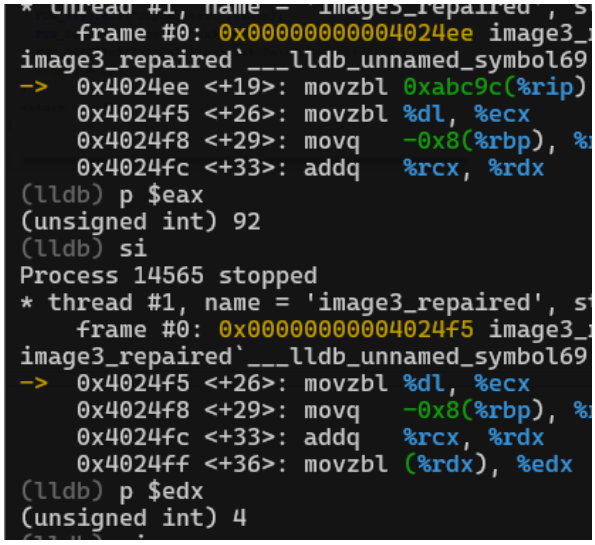
```
(lldb) x/30x 4907424
0x004ae1a0: 0x00000003 0x00000004 0x00000002 0x00000000
0x004ae1b0: 0x00000001 0x00000003 0x00000006 0x00000004
0x004ae1c0: 0x00000001 0x00000003 0x00000002 0x00000001
0x004ae1d0: 0x00000007 0x00000001 0x00000000 0x00000000
0x004ae1e0: 0x00000002 0x00000002 0x00000008 0x00000001
0x004ae1f0: 0x00000001 0x00000000 0x00000001 0x00000003
0x004ae200: 0x00000009 0x00000002 0x00000001 0x00000002
0x004ae210: 0x00000003 0x00000003
```

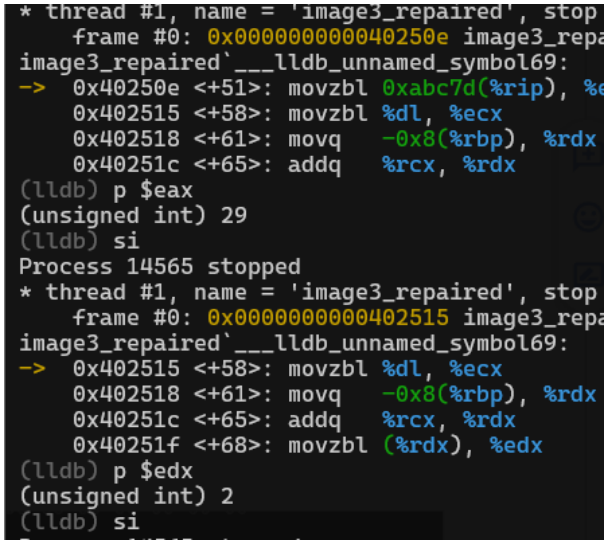
Compute_gf:

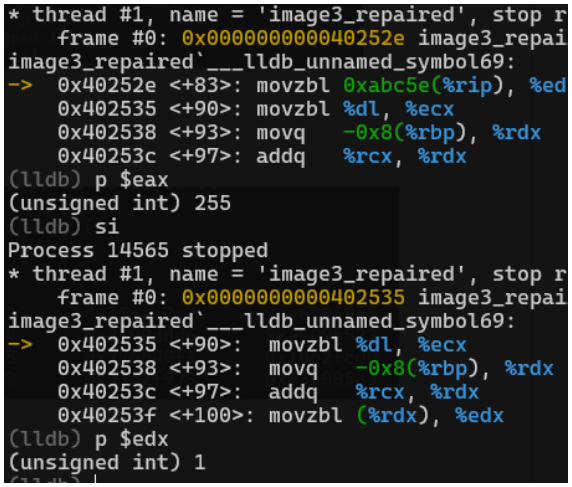
Printed the values of the variables using debugger.

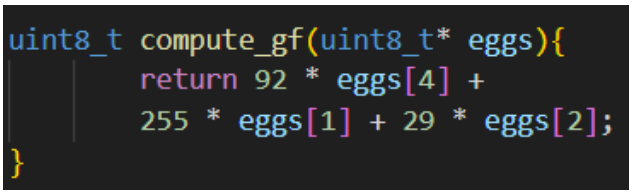
```
{
  return (uint)DAT_004ae190 * (uint)*(byte *)(param_1 + (ulong)DAT_004ae193) +
         (uint)strspnng * (uint)*(byte *)(param_1 + (ulong)DAT_004ae191) +
         (uint)DAT_004ae18f * (uint)*(byte *)(param_1 + (ulong)DAT_004ae192);
}
```

The values of the variables are not the ones that are present in the binary before runtime. They seem to change, so we had to print the values in the debugger at runtime. The final compute_gf we constructed is also present below.







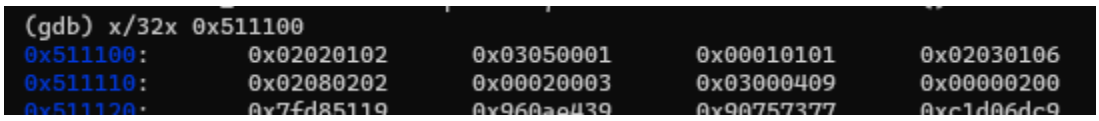


85

This is our binary!

88

Possible cipher function:
FUN_00401cf7
sbox-DAT_004df060
Egg_params look easy At 0x511100. Not dynamic kek.



He is using 4 arrays for state auStack48, 58, 68, 78 for different control flows. meh

Roundkey has key so get it:

```
Thread 1 "safe_main" hit Breakpoint 1, 0x0000000000401cf7 in ?? ()
(gdb) b *0x0401e30
Breakpoint 2 at 0x401e30
(gdb) c
Continuing.

Thread 1 "safe_main" hit Breakpoint 2, 0x0000000000401e30 in ?? ()
(gdb) x/32x 0x511120
0x511120:       0x7fd85119      0x960ae439      0x90757377      0xc1d06dc9
0x511130:       0xa2a02124      0x34aac51d      0xa4dfb66a      0x650fdba3
0x511140:       0xa8ed579f      0x9c479282      0x389824e8      0x5d97ff4b
0x511150:       0x1ba1df8d      0x87e64d0f      0xbf7e69e7      0xe2e996ac
0x511160:       0x8a39c115      0x0ddf8c1a      0xb2a1e5fd      0x50487351
0x511170:       0x5b6a938a      0x56b51f90      0xe414fa6d      0xb45c893c
```

Compute_gf

```
int FUN_00401a25(long param_1)

{
  long in_FS_OFFSET;

  if (*(long *)(in_FS_OFFSET + 0x28) != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    FUN_00428fe0();
  }
  return ((uint)*(byte *)(param_1 + 3) * 0x31 + (uint)*(byte *)(param_1 + 1) * -0x61) -
         (uint)*(byte *)(param_1 + 1);
}
```

89

91

Present in main:
obfuscated key part 1->e1 10 79 f8 05 bd 7b e1
obfuscated key part 2->49 bd 8f b6 e5 60 03 f
part1 = bytes([0xe1, 0x10, 0x79, 0xf8, 0x05, 0xbd, 0x7b, 0xe1])
part2 = bytes([0x49, 0xbd, 0x8f, 0xb6, 0xe5, 0x60, 0x03, 0xf1])

# XOR each byte with 0x41
real_key = bytes([b ^ 0x41 for b in part1 + part2])

uint8_t key[16] = {
    0xa0, 0x51, 0x38, 0xb9,
    0x44, 0xfc, 0x3a, 0xa0,
    0x08, 0xfc, 0xce, 0xf7,
    0xa4, 0x21, 0x42, 0xb0

};
Computegf

int f15(long param_1)

{
  return (uint)*(byte *)(param_1 + 4) * (uint)*(byte *)(param_1 + 3) * 7 -
         (uint)*(byte *)(param_1 + 4);
}

Egg params in simple function

```
  f5(0,param_1,param_2);
  k = 0;
  local_a8 = '\x01';
  while( true ) {
    cVar1 = f14(k,0);
    cVar2 = f14(k,1);
    bVar3 = f14(k,2);
    bVar4 = f14(k,3);
    bVar5 = f14(k,4);
    bVar6 = f14(k,5);
    f6(param_1);
    if ((cVar2 == '\x01') && (cVar1 == local_a8)) {
      uVar7 = (uint)k;
      k = k + 1;
      (&eggs)[(int)uVar7]
```

99

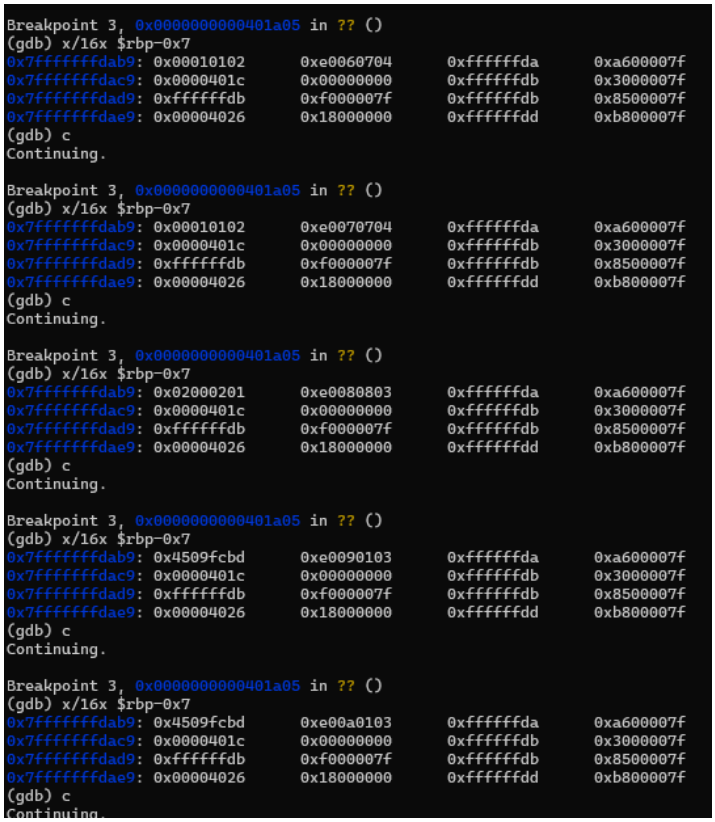
Key:

```
0x7fffffffdb00: 0x00003231      0x00000000      0x00000000      0x00000000
(gdb) b *0x401d4c
Note: breakpoint 1 also set at pc 0x401d4c.
Breakpoint 2 at 0x401d4c
(gdb) r 12
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /mnt/c/Users/LEGION/OneDrive/Documents/raadhes/iitm/Courses/CS
/A5/AttackPhase/Fortify/99/safe_main 12
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Plaintext :: 31 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Breakpoint 1, 0x0000000000401d4c in ?? ()
(gdb) p $rsi
$4 = 140737488345808
(gdb) x/16x 140737488345808
0x7fffffffdad0: 0x16ab232f      0xd89f7399      0x51af7519      0x7b1eef4f
0x7fffffffdae0: 0xffffdbf0      0x00007fff      0x00402653      0x00000000
0x7fffffffdaf0: 0xffffdd18      0x00007fff      0xffffdbb8      0x00000002
0x7fffffffdb00: 0x00003231      0x00000000      0x00000000      0x00000000
```

Go through FUN_00401c83 for cipher

Egg params: local params from rbp - 0x2(var1) to rbp-0x7(var6)

```
Breakpoint 3, 0x0000000000401a05 in ?? ()
(gdb) x/16x $rbp-0x7
0x7fffffffdab9: 0x03010201      0xe0010101      0xffffffda      0xa600007f
0x7fffffffdac9: 0x0000401c      0x00000000      0xffffffdb      0x3000007f
0x7fffffffdad9: 0xffffffdb      0xf000007f      0xffffffdb      0x8500007f
0x7fffffffdae9: 0x00004026      0x18000000      0xffffffdd      0xb800007f
(gdb) c
Continuing.

Breakpoint 3, 0x0000000000401a05 in ?? ()
(gdb) x/16x $rbp-0x7
0x7fffffffdab9: 0x01000301      0xe0020203      0xffffffda      0xa600007f
0x7fffffffdac9: 0x0000401c      0x00000000      0xffffffdb      0x3000007f
0x7fffffffdad9: 0xffffffdb      0xf000007f      0xffffffdb      0x8500007f
0x7fffffffdae9: 0x00004026      0x18000000      0xffffffdd      0xb800007f
(gdb) c
Continuing.

Breakpoint 3, 0x0000000000401a05 in ?? ()
(gdb) x/16x $rbp-0x7
0x7fffffffdab9: 0x01000303      0xe0030401      0xffffffda      0xa600007f
0x7fffffffdac9: 0x0000401c      0x00000000      0xffffffdb      0x3000007f
0x7fffffffdad9: 0xffffffdb      0xf000007f      0xffffffdb      0x8500007f
0x7fffffffdae9: 0x00004026      0x18000000      0xffffffdd      0xb800007f
(gdb) c
Continuing.

Breakpoint 3, 0x0000000000401a05 in ?? ()
(gdb) x/16x $rbp-0x7
0x7fffffffdab9: 0x01000303      0xe0040401      0xffffffda      0xa600007f
0x7fffffffdac9: 0x0000401c      0x00000000      0xffffffdb      0x3000007f
0x7fffffffdad9: 0xffffffdb      0xf000007f      0xffffffdb      0x8500007f
0x7fffffffdae9: 0x00004026      0x18000000      0xffffffdd      0xb800007f
(gdb) c
Continuing.

Breakpoint 3, 0x0000000000401a05 in ?? ()
(gdb) x/16x $rbp-0x7
0x7fffffffdab9: 0x00010102      0xe0050704      0xffffffda      0xa600007f
0x7fffffffdac9: 0x0000401c      0x00000000      0xffffffdb      0x3000007f
0x7fffffffdad9: 0xffffffdb      0xf000007f      0xffffffdb      0x8500007f
0x7fffffffdae9: 0x00004026      0x18000000      0xffffffdd      0xb800007f
(gdb) c
```



compute_gf is easy. Ghidra shows this function: Clearly compute_gf.

```
int FUN_00401801(long param_1)

{
  return ((uint)*(byte *)(param_1 + 4) * 9 + (uint)*(byte *)(param_1 + 4) * 2 +
          (uint)*(byte *)(param_1 + 3) * 0x45) - (uint)*(byte *)(param_1 + 1);
}
```

Resources Used , Commands & Acknowledgements:

- Objdump, GDB, lldb, radare
- Readelf, strace
- Ghidra

Contributions

We both worked collaboratively and chose binaries to crack independently. Sometimes we split binaries so we do not work on overlapping binaries.