CS577 : PROJECT PHASE 2

Gandhar Deshpande 224101019
Jasmin Borad 224101025
Ayush Mandloi 224101011
Kaimadam Antony Shejin 224101026
Bajirao Salunke 224101012

1. Approach:

In order to get the correct keys from locked code, we have used *SAT attack* using Z3 SAT solver in python.

SAT attack basically eliminates some set of wrong keys in each iteration. After some iterations no wrong keys were left to eliminate, so the remaining key is the correct key.
In order to perform SAT attack we need the Locked Code and the Oracle. Oracle is black box where it gives correct output for any given inputs.
We eliminate the wrong keys with the help of *Distinguishing Input Patterns*(DIPs). DIPs are the inputs for which different sets of keys give different outputs.

2. Method:

DIP = Distinguishing Input Patterns
LockedLogic = LL(I, K), Oracle = F(I)

Steps in SAT attack using SAT solver:

1. Write Locked logic (LL) using SAT solver.
2. Take two sets of keys K_A and K_B.
3. Find I_d(DIP) such that LL(I, K_A) != LL(I, K_B)
4. Find the output O_d for input I_d using oracle, i.e. O_d = F(I_d).
5. Now add constraints using oracle output O_d.
    a. LL(I_d, K_A) == O_d
    b. LL(I_d, K_B) == O_d
6. Repeat above steps until no DIP is left i.e the problem becomes unsat.
7. Now remove the constraint of DIP, and add constraint to make output and keys equal and call the SAT solver for the last time to get the correct values of keys.

3. Implementation:

Tool used : Z3 solver python

1. Write python code for using z3 library for the given locked code i.e. ByteSub_ShiftRow function from obfuscated.c file.

```
# function in obfuscated.c file implemented in z3
def ByteSub_ShiftRow(solver, statemt, Sbox, nb, n, key1, key2, key3,
key4, key5, key6, key7):
       ...
       ...
```

2. Take two sets of key

```
# Z3 Solver
s = Solver()

# variables inputs
nb = BitVec('nb', 32) # Int 32 Bit
n = BitVec('n', 32) # Int 32 Bit

# First set of keys
key11 = BitVec('key11', 32) # Int 32 Bit
key12 = BitVec('key12', 32) # Int 32 Bit
key13 = BitVec('key13', 32) # Int 32 Bit
key14 = BitVec('key14', 32) # Int 32 Bit
key15 = Bool('key15') # Boolean
key16 = Bool('key16') # Boolean
key17 = Bool('key17') # Boolean

#second set of keys
key21 = BitVec('key21', 32) # Int 32 Bit
key22 = BitVec('key22', 32) # Int 32 Bit
key23 = BitVec('key23', 32) # Int 32 Bit
key24 = BitVec('key24', 32) # Int 32 Bit
key25 = Bool('key25') # Boolean
key26 = Bool('key26') # Boolean
key27 = Bool('key27') # Boolean
```

3. Add constraints to find the DIP.

```
out1 = ByteSub_ShiftRow(s, statemt, Sbox, nb, n, key11, key12, key13,
key14, key15, key16, key17)
out2 = ByteSub_ShiftRow(s, statemt, Sbox, nb, n, key21, key22, key23,
key24, key25, key26, key27)
s.add(out1 != out2) # checking for DIP

# Checking if problem is sat or unsat
if (s.check() == sat):
      print("sat")
      print(s.model()) # prints all values of inputs and keys
else:
print("unsat")
```

For above constraints Z3 will give values of inputs n and nb.
We got value n = 4 and nb = 67108769

4. Find out the correct output values for above DIP using oracle

```
output = [67108804, 67108959, 67108839, 67108793, 67108965, 67108968,
67108966, 67108963, 67108968, 67108923, 67108835, 67108862, 67108994,
67108795, 67108773, 67108827]
```

5. Add constraints using the above output.

```
output = [67108804, 67108959, 67108839, 67108793, 67108965, 67108968,
67108966, 67108963, 67108968, 67108923, 67108835, 67108862, 67108994,
67108795, 67108773, 67108827]

# adding constraint for first set of keys
out_test = ByteSub_ShiftRow(s, statemt, Sbox, 4, 67108769, key11, key12,
key13, key14, key15, key16, key17)
for i in range(16):
      s.add(out_test[i] == output[i]) # adding constraints

# adding constraint for second set of keys
out_test = ByteSub_ShiftRow(s, statemt, Sbox, 4, 67108769, key21, key22,
key23, key24, key25, key26, key27)
for i in range(16):
      s.add(out_test[i] == output[i]) # adding constraints
```

6. Check again for DIP after adding constraints. Now we get unsat, so there is no DIP left.

```
# Checking if problem is sat or unsat
if (s.check() == sat):
	print("sat")
	print(s.model()) # prints all values of inputs and keys
else:
	print("unsat")
```

7. To find the keys change constraint out1 != out2 to out1 == out2 and make two set of keys equal

```
# making keys equal
s.add(key11 == key21)
s.add(key12 == key22)
s.add(key13 == key23)
s.add(key14 == key24)
s.add(key15 == key25)
s.add(key16 == key26)
s.add(key17 == key27)

out1 = ByteSub_ShiftRow(s, statemt, Sbox, nb, n, key11, key12, key13,
key14, key15, key16, key17)
out2 = ByteSub_ShiftRow(s, statemt, Sbox, nb, n, key21, key22, key23,
key24, key25, key26, key27)
# s.add(out1 != out2) # checking for DIP
s.add(out1 == out2) # checking after unsat final key value
```

8. Find the desired keys.

```
# Checking if problem is sat or unsat
if (s.check() == sat):
	print("sat")
	print(s.model()) # prints all values of inputs and keys
else:
	print("unsat")
```

As the model will be satisfied, it will print values of inputs and keys.

4. Final Keys found :

Key	Value
Key1	5
Key2	15
Key3	7
Key4	4
Key5	False
Key6	False
Key7	False