nthu hardware security hw2

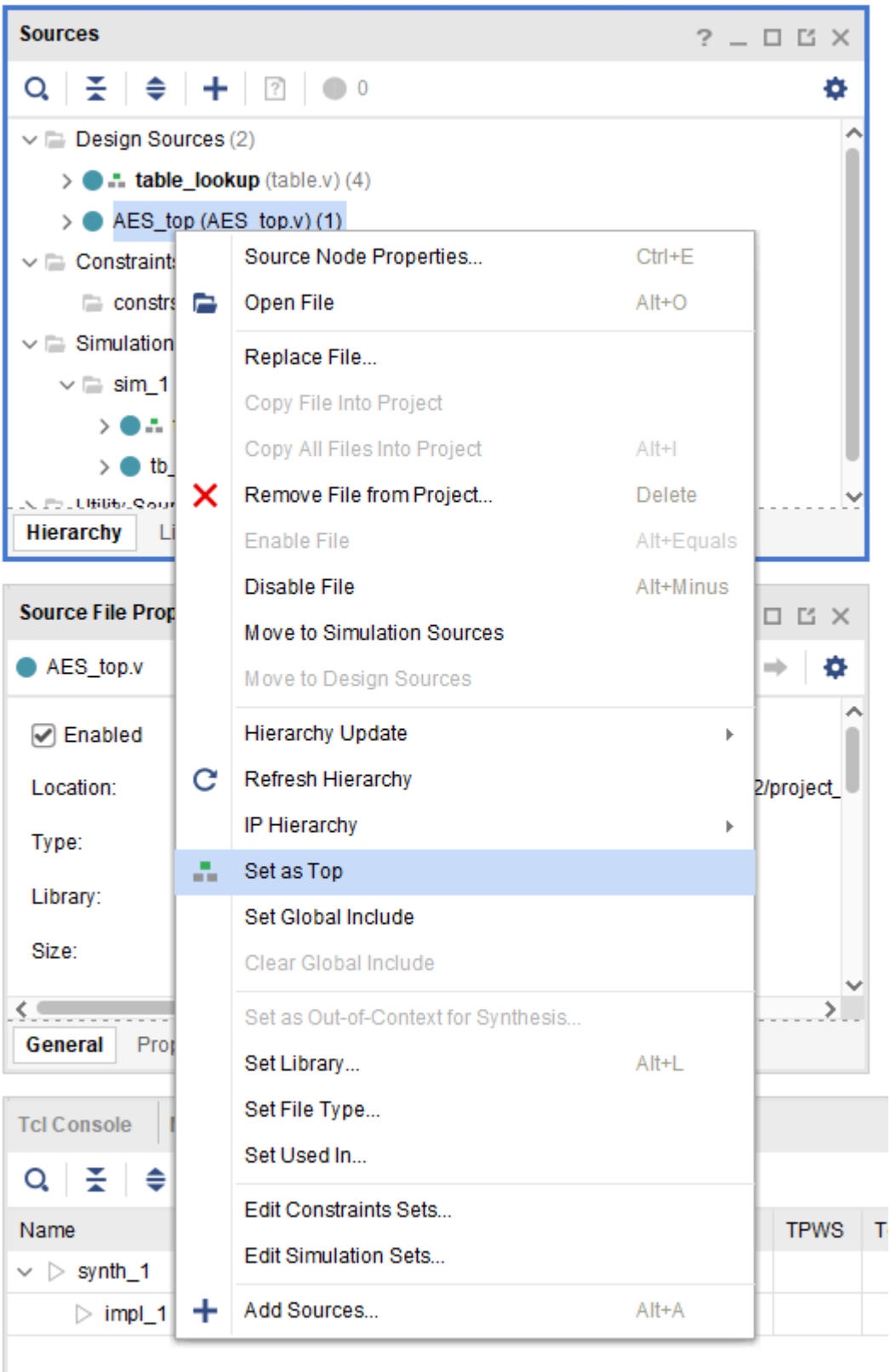
109062233 蘇裕恆

How to compile and execute your program

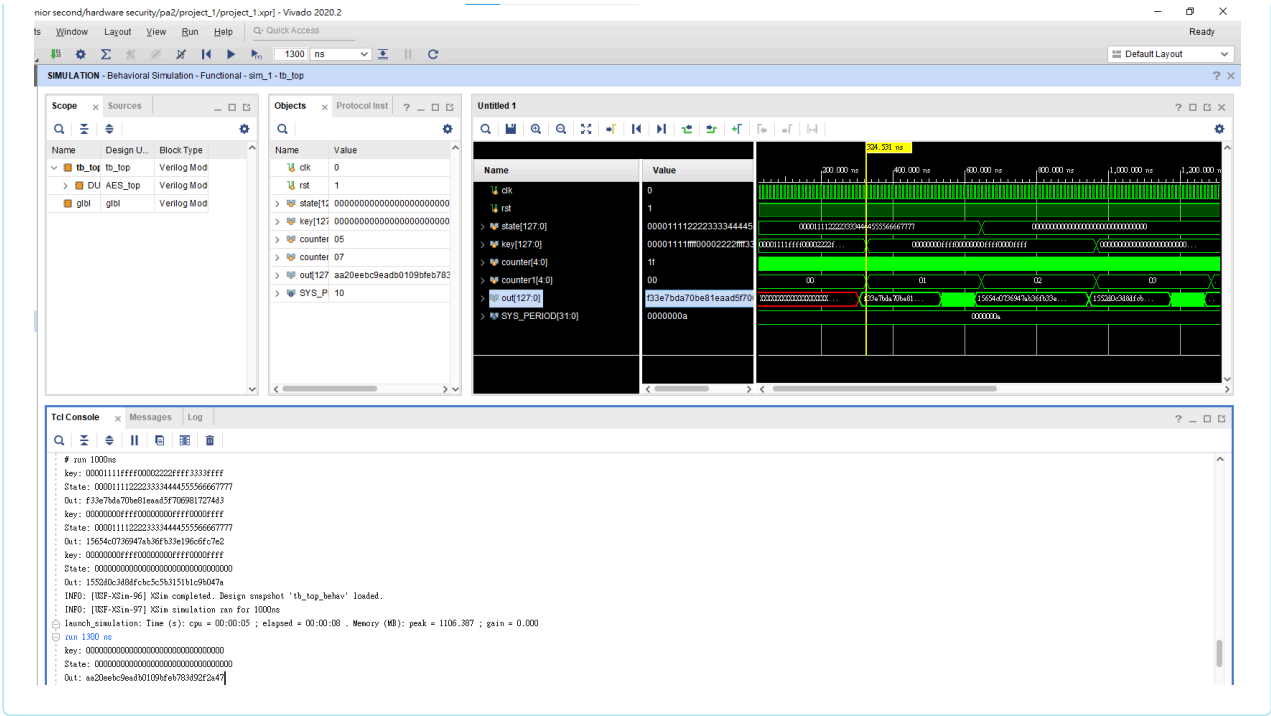
dicectory structure

```
./109062233_PA2_AES
├── aes_128.v # main of aes
├── AES_top.v # top module
├── expand_key_128.v # key generation for round keys
├── round.v # main changes
├── table.v # utility functions
└── tb_top.v # testbench

./109062233_PA2_HT
├──  sample_HT
│   └── aes_128.v # main of aes
│   └── AES_top.v # top module (changed)
│   └── expand_key_128.v # key generation for round keys
│   └── round.v # same as pa2_aes
│   └── table.v # utility functions
│   └── tb_top.v # testbench (changed)
├── reference_HT
│   └── aes_128.v # main of aes
│   └── AES_top.v # top module
│   └── expand_key_128.v # key generation for round keys
│   └── round.v # changed
│   └── table.v # utility functions
└   └── tb_top.v # testbench (same as aes 128)

./revert
├── aeskeyschedule.py # main util for reverse key
├── main.py # top module
└── test_aeskeyschedule.py # test for util


```

Remember to set the AES_top as top (Also the testbench)



Then, run the simulation. Set the ns to be **1300** to see full output.



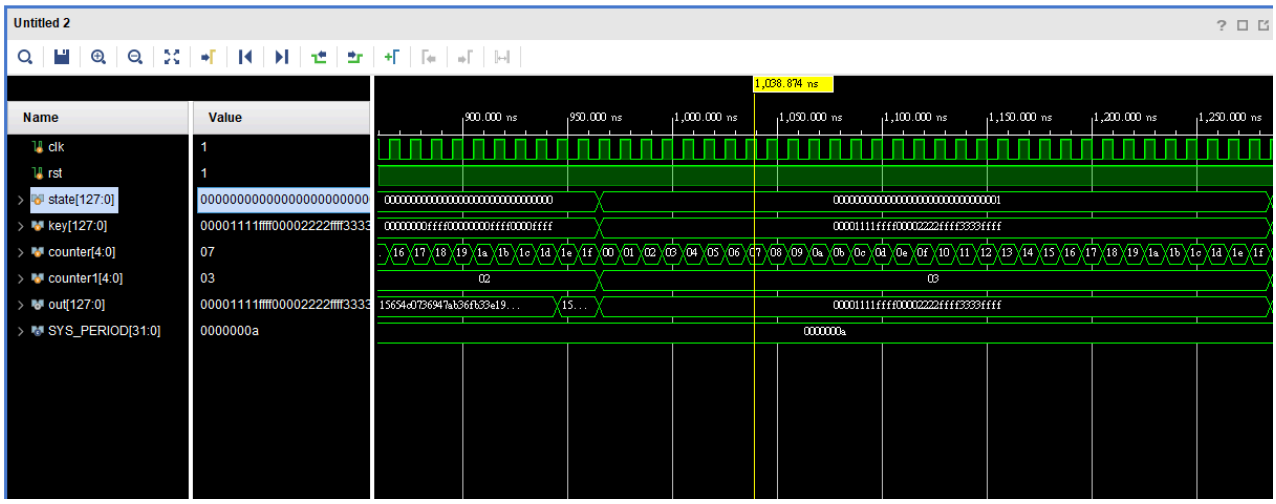
The completion of the assignment

baseline aes

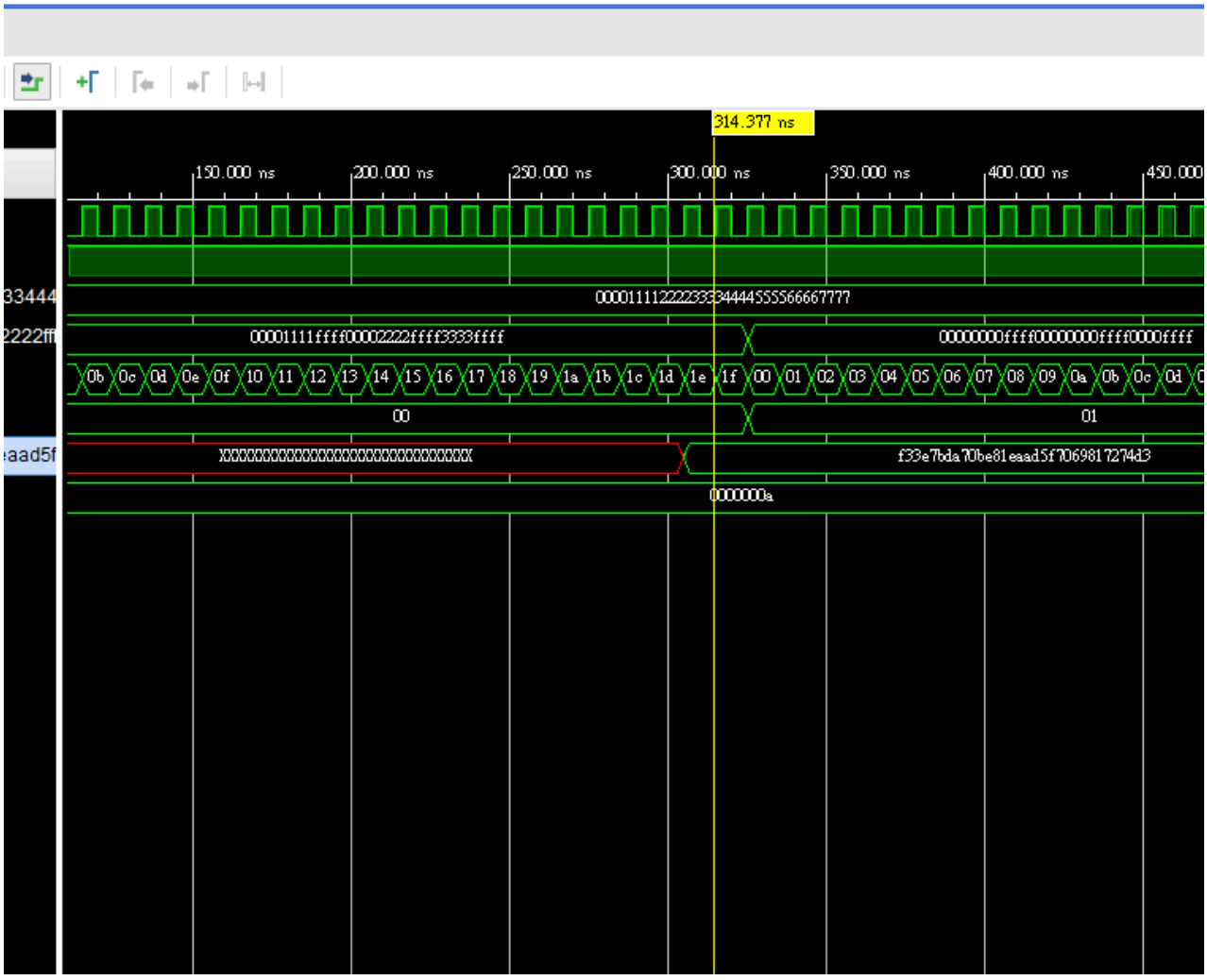
```
key: 00001111ffff00002222ffff3333ffff
State: 0000111122223333444455556666777 7
Out: f33e7bda70be81eaad5f7069817274d3
key: 00000000ffff00000000ffff0000ffff
State: 0000111122223333444455556666777 7
Out: 15654c0736947ab36fb33e196c6fc7e2
key: 00000000ffff00000000ffff0000ffff
State: 0000000000000000000000000000000 0
Out: 1552d0c3d8dfcbc5c5b3151b1c9b047a
key: 00000000000000000000000000000000
State: 0000000000000000000000000000000 0
Out: aa20eebc9eadb0109bfeb783d92f2a47
```

sample ht:

For state == 1, we will simply output the key.

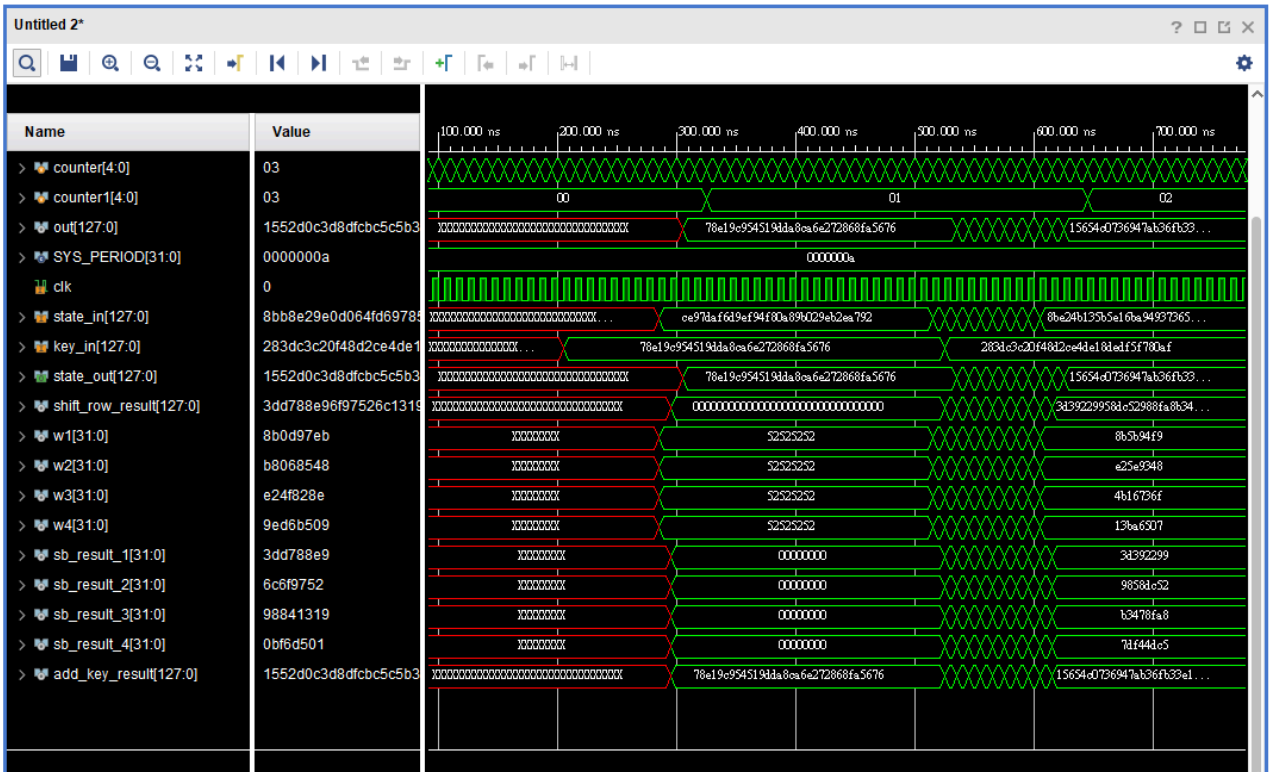


For the rest, the behavior will be the same as the eas 128 before.

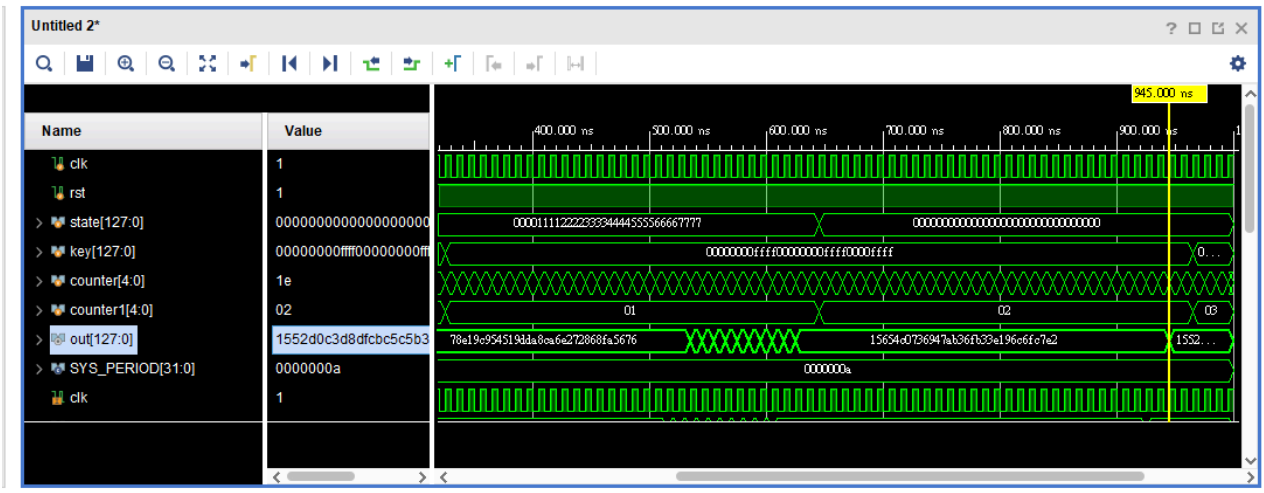


reference ht:

For some specific input(state), it will generate the last key.(Explained downwards)



For others, it will generate the same output as the baseline



The hardware trojan you design

我更改的是從**A trojan framework in AES core to evade state-of-the-art HT detection schemes** 所論述的方法。
首先，trigger rate是1/1024。

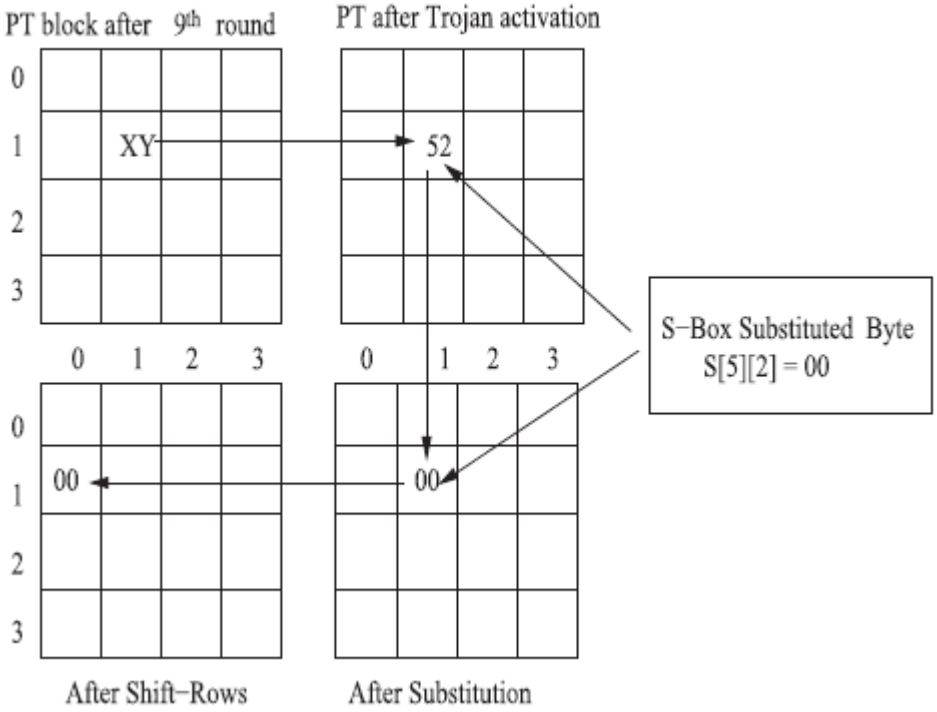
```
assign {w1} = (state_in & 10'b00_0110_1101 ) ? {byte_16, byte_12, byte_8,
assign {w2} = (state_in & 10'b00_0110_1101 ) ? {byte_15, byte_11, byte_7,
assign {w3} = (state_in & 10'b00_0110_1101 ) ? {byte_14, byte_10, byte_6,
assign {w4} = (state_in & 10'b00_0110_1101 ) ? {byte_13, byte_9, byte_5,
```

trigger

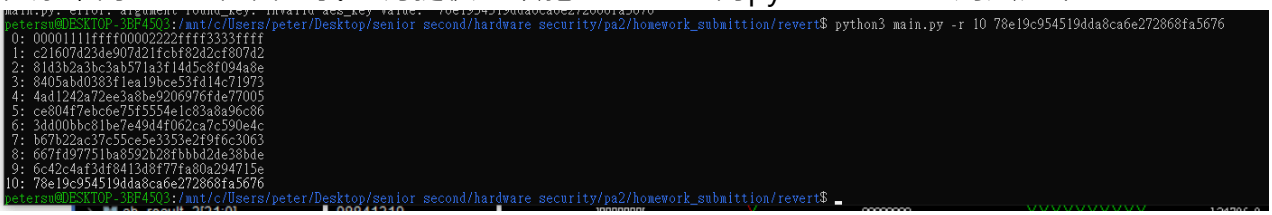
可以看到 當他在最後一個state傳進去的時候，若他的值最後幾個為11_1001_0010的話，這樣才會trigger trojan。以上面的例子來看，因為有一個testcase傳進去state在最後一個round之中的suffix為792，剛好符合suffix為11_1001_0010這個條件，因此，他就會trigger到hardware trojan。

payload

因為要在有限的testbench裡面完成，所以我將整個paper的作法只取核心(?) 的一小部分(?)
首先，他的機制在於當最後個round的時候，因為他不會做mixcolumn，因此，假設我們今天把其中一個block(byte)換成52，他在substitude bytes的時候，就會將它變成00



眾所周知，00 xor 任意兩個bits的東西都會顯示成原本的東西，因此，他就會洩漏一部份的key。而我們就可以藉由final key來得到最初的key。
而我們的作法是將全部的value都改成52，這樣我們就不需要慢慢一個一個拚成原本的，而是可以在快速的時間內得到原本的key (round 10's key)。
在原本的folder裡面，我也有提供可以把它reverse的python tool，用法如下




```
# in the python folder
python3 main.py -r 10 78e19c954519dda8ca6e272868fa5676 (key for 10th roun
```

The hardness of this assignment and how you overcome it

not much xd

Any suggestions about this programming assignment?

我覺得這是一個很棒的作業喔!