Physical Attacks on Secure Systems (NWI-IMC068)

Tutorial 4: Leakage assessment

Omid Bazangani (omid.bazangani@ru.nl)

Goals: After completing this tutorial successfully, you should understand how to conduct a TVLA test for a leakage assessment, choose a fixed-set and random-set, record the dataset, and how interpret the result.

Before you start: To complete this tutorial, your teacher will provide you with a package that contains the Chipwhisper-lite board. You will collect the traces with Chipwhisperer as you did in the previous tutorials.

1 Test Vector Leakage Assessment (TVLA)

Leakage assessment can be done independently from any specific intermediate values and only depends on key or input. This test is called the Non-Specific TVLA test. In contrast, Specific TVLA evaluates a specific intermediate value. You can find more information here. This tutorial will apply a Non-Specific TVLA test on the TinyAES implementation running on an ARM Cortex-M3 microcontroller (`STM32F303`) using the ChipWhisperer framework. In this assessment, we investigate if an adversary can learn about the key by analyzing the power traces for fixed data compared to random data.

1.1 Test Requirments

Using the ChipWhisperer framework gives you access to an implemented TinyAES firmware ready to be run on the target. A Jupyter notebook file (`.ipynb`) is provided to guide you to conduct the test step by step. You can download the file from the link. After downloading the file, you need to place it in the ChipWhisperer directory. For example, in Windows, it can be like: "`C:\Users\USER\ChipWhisperer_xx\cw\home\portable\chipwhisperer\jupyter\experiments`." [1]

This file consists of five Steps as follows.

1. Configure platform and crypto algorithm and compile the firmware

[1] In Linux or MAC, the path can be different. You must put the notebook file in the correct directory based on the operating system.

2. Configure and connect to the ChipWhisperer
3. Configure the programmer and flash the target
4. Trace record and dataset creation (needs to be completed by you)
5. Apply the TVLA test (needs to be completed by you)

1.2 Step1:

In this step, we will select the firmware that we need (TinyAES), the target board (CWLITEARM), and the ChipWhisperer scope.

```
# Config Platform, Crypto Algorithm and Scope

SCOPETYPE = "OPENADC"
PLATFORM  = "CWLITEARM"
CRYPTO_TARGET  = "TINYAES128C"
```

The following cell of the Jupyter Notebook file is responsible for compiling the code and making the firmware.[2]

```
# Compile the firmware for the introduced target

%%bash -s "$PLATFORM" "$CRYPTO_TARGET"
cd  ../../hardware/victims/firmware/simpleserial-aes
make PLATFORM=$1 CRYPTO_TARGET=$2
```

1.3 Step2:

This step checks if the ChipWhisperer is connected to the USB port, and then it connects to it. Otherwise, you better check the USB port and make sure the driver of your ChipWhisperer is appropriately installed.

[2]The provided bash script is a Windows-compatible command. You must replace it with proper commands based on your OS.

```
import chipwhisperer as cw
import usb

try:
    try:
        if not: scope.connectStatus:
            scope.con()
    except: NameError:
        scope = cw.scope()

    try:
        if SS_VER == "SS_VER_2_0":
            target_type = cw.targets.SimpleSerial2
        else:
            target_type = cw.targets.SimpleSerial
    except:
        SS_VER="SS_VER_1_1"
        target_type = cw.targets.SimpleSerial

    try:
        target = cw.target(scope, target_type)
    except IOError:
        print("INFO: Caught exception on reconnecting to target attempting
               to reconnect to scope first.")
        print("INFO: This is a work-around when USB has died without Python
              knowing. Ignore errors above this line.")
        scope = cw.scope()
        target = cw.target(scope, target_type)
except:
    if usb.__version__ < '1.1.0':
        print("-----------------------------------")
        print("Unable to connect to chipwhisperer. pyusb  detected
        (>= 1.1.0 required)".format(usb.__version))
        print("-----------------------------------")
    raise

print("INFO: Found ChipWhisperer")
```

1.4 Step3:

Before starting trace collection, you must program the chip with the compiled firmware. The following two cells show how the ChipWhisperer configs the built-in programmer, and the third cell defines the firmware path followed by the chip programming command.

```
# Choosing the proper programmer based on the platform

if "STM" in PLATFORM or PLATFORM == "CWLITEARM" or PLATFORM == "CWNANO":
    prog = cw.programmers.STM32FProgrammer
elif PLATFORM == "CW303" or PLATFORM == "CWLITEXMEGA":
    prog = cw.programmers.XMEGAProgrammer
else:
    prog = None
```

```
# GPIO pin configuration act as the trigger signal for Oscope

import time
time.sleep(0.05)
scope.default_setup()
def reset_target(scope):
    if PLATFORM == "CW303" or PLATFORM == "CWLITEXMEGA":
        scope.io.pdic = "low"
        time.sleep(0.1)
        scope.io.pdic = "high_z"  #XMEGA doesn't like pdic driven high
        time.sleep(0.1)           #xmega needs more startup time
    else:
        scope.io.nrst = "low"
        time.sleep(0.05)
        scope.io.nrst = "high_z"
        time.sleep(0.05)
```

```
# program the chip with the compiled firmware

fw_path = '../../hardware/victims/firmware/simpleserial-aes/
          simpleserial-aes-.hex'.format(PLATFORM)
cw.program_target(scope, prog, fw_path)
```

1.5 Step4:

This section aims to capture a trace set to conduct a Non-Specific TVLA test on it. We need to capture a fixed and random dataset based on a fixed and random plain text fed to the target while the key is fixed (Fig 1).
Statistically, it's important how the fixed and random plain text and the fixed key are chosen, and there are specific rules for it. For the AES algorithm, it has been suggested to choose the fixed key (K_{Fixed}) as follows:

- 0x0123456789abcdef123456789abcdef0 for AES-128
- 0x0123456789abcdef123456789abcdef023456789abcdef01 for AES-192
- 0x0123456789abcdef123456789abcdef023456789abcdef013456789abcdef012 for AES-256

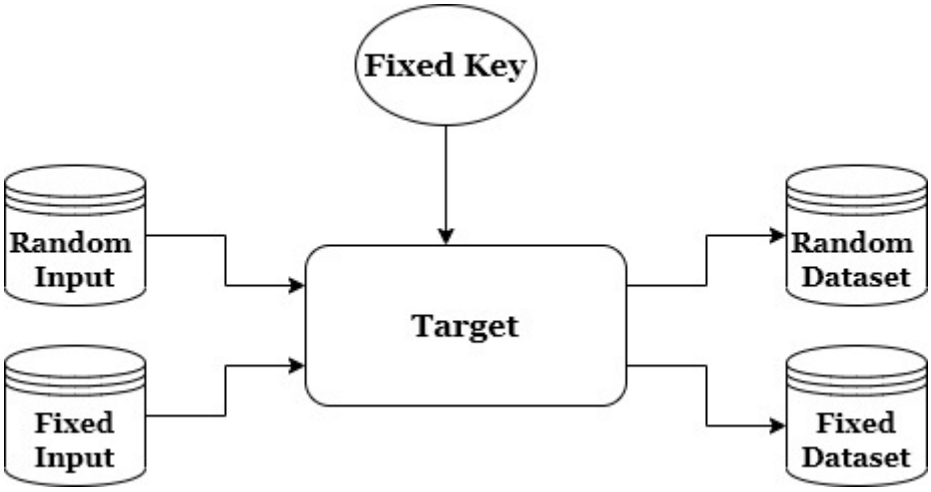

Figure 1: TVLA dataset construction

Setting the fixed key, for the fixed data datasets, perform n encryptions with the fixed plaintext (P_{Fixed}) set as follows:

- 0xda39a3ee5e6b4b0d3255bfef95601890 for AES-128
- 0xda39a3ee5e6b4b0d3255bfef95601888 for AES-192
- 0xda39a3ee5e6b4b0d3255bfef95601895 for AES-256

Constructing the random dataset needs a random set of plaintext (P_{Rand} which is randomized by encryption of the key and previous plaintext as follows:

- I_0 = 0x00...0 (16 bytes of zeros)
- I_{j+1} = AES(K_{gen}, I_j) for 0 ≤ j < n

Although in the ChipWhisperer framework, the required key and plaintexts are handled by AcqKeyTextPattern_TVLATTest class, You can find more information about it in a paper provided by Rambus Company here.
Having the fixed key and plaintext set raises the question of "in which order should the trace be collected?". The answer is, the two datasets should be recorded randomly interspersed to avoid any systematic bias due to the order of collection. It means we must collect one fixed trace followed by a random trace, and so on (Fig2). You will see it would be done easily by using `AcqKeyTextPattern_TVLATTest` class in the ChipWhisperer framework.
In order to construct the dataset, you need to complete the `Jupyter Notebook` code. It's as follows:

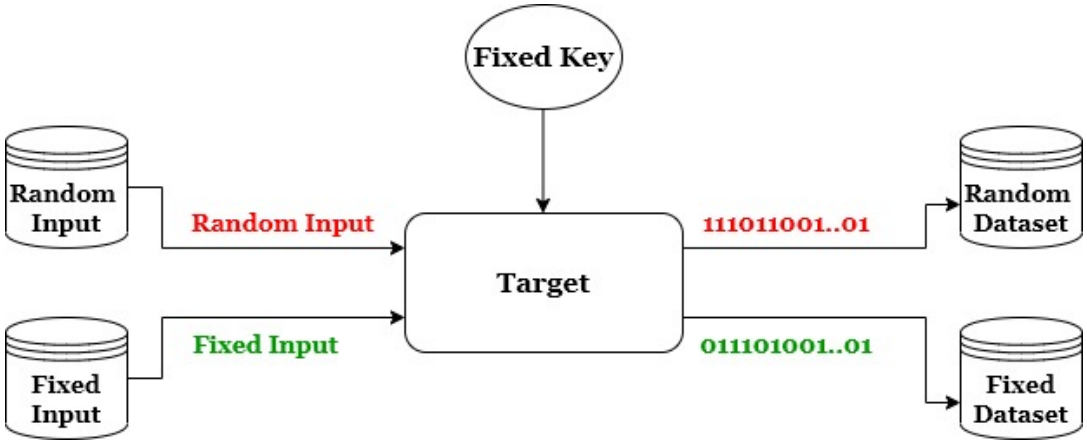

Figure 2: TVLA dataset order

```
 #Capture Traces

from tqdm import trange
import numpy as np

N = 10000  # Number of traces (50% Fixed 50% Random)
ktp = cw.ktp.TVLATTest()
ktp.init(N)          # init with the number of traces you plan to

traces = []
Fix_text = bytearray.fromhex("da39a3ee5e6b4b0d3255bfef95601890")

Random_project = cw.create_project("RandomSet.cwp")
Fix_project = cw.create_project("FixSet.cwp")

for i in trange(N, desc='Capturing traces'):
    key, text = ktp.next() # Select the Next key and plaintext

    trace = ...    #Code needs to be completed

     #You need to append traces either in the Random or the Fixed project
     based on their fed text!
```

1.6 Step5:

Having the required dataset for the test[3], we can apply the TVLA test in Python using **ttest_ind** function from **statistics library**.

[3]Note: if you cannot capture traces using ChipWhisperer, You can download the provided dataset from this link. The provided dataset consists of two different power trace files in `.npy` format, one for fixed data and the other for random data.

ttest_rslt, pvalue_rslt = stats.ttest_ind(Fixed_dataset [ ][ ], Random_dataset [ ][ ], equal_var=False)
In order to use this function, you need to import **stats** from **scipy** library (from scipy import stats).

Some useful tips.

- To find the samples which failed the test, compare ttest_rslt with the threshold $(+/-4.5)$.
- Argument **equal_var** should be False (to apply Welch's t-test).
- Do not forget about the negative part of the threshold value.
- Plot the result of the t-test.

Considering the aforementioned tips, you must complete the following code in the Jupyter Notebook provided.

```
#Apply TVLA test on the provided Dataset

from scipy import stats
import matplotlib.pyplot as plt
ttest_rslt, pvalue_rslt = stats.ttest_ind(..., equal_var=False)

#Code needs to be completed
...

print(f"Number of leaky points are: ...")

#Plot the TVLA Graph
plt.figure(figsize=(9, 3))
plt.plot(ttest_rslt)
plt.xlabel("Samples")
plt.ylabel("t-test value: ")
plt.axhline(y=4.5, color='r', linestyle=':')
plt.axhline(y=-4.5, color='r', linestyle=':')
plt.ylim((-20, 20))
plt.xlim((0, len(ttest_rslt)))
plt.title('TinyAES Implementation')
plt.savefig("TinyAes_TVLA.jpg")
plt.tight_layout()
```

2 Questions

After successfully conducting the test, you can challenge yourself by answering the following questions.

1. How can you interpret the TVLA graph?
2. How many leaky points did you find?
3. Is there any relation between the number of traces and the number of leaky points?
4. Can you think of any countermeasure against the TVLA test?
5. How do you interpret a high-value compared to a low-value for test results?
6. Can you explain how we can trace back from the power trace to the assembly instructions (software code) to pinpoint the cause of leakage?