# Side-channel evaluation of Shakti's AES accelerators

Surya Prasad S (Guided by Chester Rebeiro)

https://github.com/GnosGnas/Side-Channel-Analysis

## Contents

# 1 Introduction

Power Analysis attacks have proven to be fatal to otherwise tamper-proof hardware. Non-invasive attacks like SPA and DPA are particularly dangerous as they can be easily reproduced and scaled. So, enter Test Vector Leakage Assessment Methodology (TVLA), a popular side-channel testing methodology as it can detect power leakages at any time of the operation irrespective of whether an attack can be successfully performed or not with that leakage. Test Vector Leakage Assessment (TVLA) aims to provide detection of information leakage using statistical analysis. We have done the TVLA test on Shakti's AES accelerators which can be integrated with the Shakti C-Class processor. The power consumption of our AES accelerators have been analysed extensively and optimisations are made to reduce power leakage.

# 2 Overview

## 2.1 AES Algorithm

Advanced Encryption Standard, or AES, is a popular symmetric algorithm for encryption. Symmetric algorithms are named because they use the same cryptographic key for encryption and decryption. This comes with both advantages and disadvantages. From an implementation point of view, symmetric algorithms process much quicker than asymmetric algorithms, and a typical hardware accelerator takes only 12 cycles to complete one process.

The algorithm takes in a 128-bit input and generates a 128-bit output. AES is capable of using keys of length 128, 192 or 256 bits. The algorithm encrypts or decrypts by repeatedly applying a set of operations to the input data. The number of rounds of a process is fixed for each length of key:

- 10 rounds for AES-128
- 12 rounds for AES-192
- 14 rounds for AES-256

Figure 1 shows the structure of AES algorithm and specifies the operations in each round.

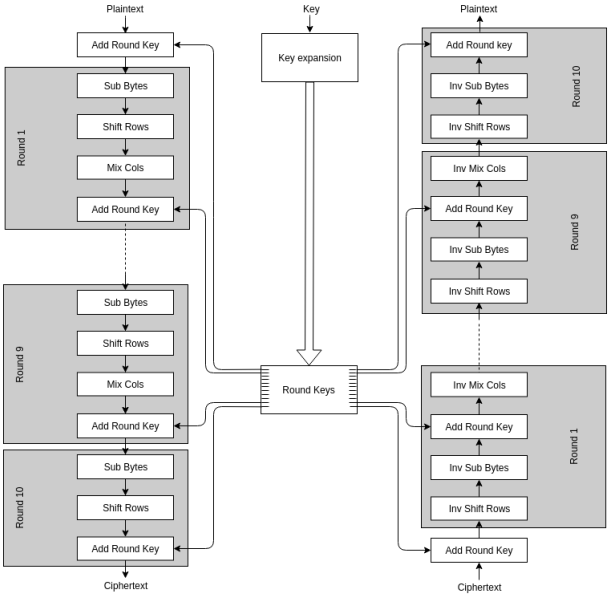



Figure 1: Algorithmic structure of 128-bit AES

*SubBytes* and *InvSubBytes* are two critical functions in encryption and decryption. They use the transformation function S-box, which consumes a great deal of power in the AES algorithm's hardware and software implementations.

### 2.1.1 SubBytes

*SubBytes* transformation [1] is a non-linear byte substitution that maps one byte to another in $GF(2^8)$. The mapping of each byte is done using an S-box which is defined over the irreducible polynomial, $R(x) = x^8 + x^4 + x^3 + x + 1$. The output of the S-box is defined as follows:

$$SBox(a) = \begin{cases} Ma^{-1} + N & if\ a \neq 0 \\ N & if\ a = 0 \end{cases}$$

where M is an $8x8$ matrix, $N \in GF(2^8)$ and $a^{-1}$ is the multiplicative inverse in $GF(2^8)$.

There exists an isomorphism between $GF(2^8)$ and composite field of the same cardinality, which we have taken advantage of in our implementation.

Table 1: AES S-box Table

| | | Y | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| X | 0 | 52 | 09 | 6A | D5 | 30 | 36 | A5 | 38 | BF | 40 | A3 | 9E | 81 | F3 | D7 | FB |
| | 1 | 7C | E3 | 39 | 82 | 9B | 2F | FF | 87 | 34 | 83 | 43 | 44 | C4 | DE | E9 | CB |
| | 2 | 54 | 7B | 94 | 32 | A6 | C2 | 23 | 3D | EE | 4C | 95 | 0B | 42 | FA | C3 | 4E |
| | 3 | 08 | 2E | A1 | 66 | 28 | D9 | 24 | B2 | 76 | 5B | A2 | 49 | 6D | 8B | D1 | 25 |
| | 4 | 72 | F8 | F6 | 64 | 86 | 68 | 98 | 16 | D4 | A4 | 5C | CC | 5D | 65 | B6 | 92 |
| | 5 | 6C | 70 | 48 | 50 | FD | ED | B9 | DA | 5E | 15 | 46 | 57 | A7 | 8D | 9D | 84 |
| | 6 | 90 | D8 | AB | 00 | 8C | BC | D3 | 0A | F7 | E4 | 58 | 05 | B8 | B3 | 45 | 06 |
| | 7 | D0 | 2C | 1E | 8F | CA | 3F | 0F | 02 | C1 | AF | BD | 03 | 01 | 13 | 8A | 6B |
| | 8 | 3A | 91 | 11 | 41 | 4F | 67 | DC | EA | 97 | F2 | CF | CE | F0 | B4 | E6 | 73 |
| | 9 | 96 | AC | 74 | 22 | E7 | AD | 35 | 85 | E2 | F9 | 37 | E8 | 1C | 75 | DF | 6E |
| | a | 47 | F1 | 1A | 71 | 1D | 29 | C5 | 89 | 6F | B7 | 62 | 0E | AA | 18 | BE | 1B |
| | b | FC | 56 | 3E | 4B | C6 | D2 | 79 | 20 | 9A | DB | C0 | FE | 78 | CD | 5A | F4 |
| | c | 1F | DD | A8 | 33 | 88 | 07 | C7 | 31 | B1 | 12 | 10 | 59 | 27 | 80 | EC | 5F |
| | d | 60 | 51 | 7F | A9 | 19 | B5 | 4A | 0D | 2D | E5 | 7A | 9F | 93 | C9 | 9C | EF |
| | e | A0 | E0 | 3B | 4D | AE | 2A | F5 | B0 | C8 | EB | BB | 3C | 83 | 53 | 99 | 61 |
| | f | 17 | 2B | 04 | 7E | BA | 77 | D6 | 26 | E1 | 69 | 14 | 63 | 55 | 21 | 0C | 7D |

### 2.1.2 InvSubBytes

*InvSubBytes* [1] is the inverse of the byte substitution transformation and maps one byte to another using the inverse S-box. Inverse S-box is defined as follows:

$$InvSBox(b) = \begin{cases} M^{-1}(b+N)^{-1} & if\ b \neq N \\ 0 & if\ b = N \end{cases}$$

where similar definitions are followed.
Here too we can utilize the isomorphism with composite field arithmetic.

Table 2: AES Inverse S-box Table

| | | Y | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| X | 0 | 63 | 7C | 77 | 7B | F2 | 6B | 6F | C5 | 30 | 01 | 67 | 2B | FE | D7 | AB | 76 |
| | 1 | CA | 82 | C9 | 7D | FA | 59 | 47 | F0 | AD | D4 | A2 | AF | 9C | A4 | 72 | C0 |
| | 2 | B7 | FD | 93 | 26 | 36 | 3F | F7 | CC | 34 | A5 | E5 | F1 | 71 | D8 | 31 | 15 |
| | 3 | 04 | C7 | 23 | C3 | 18 | 96 | 05 | 9A | 07 | 12 | 80 | E2 | EB | 27 | B2 | 75 |
| | 4 | 09 | 83 | 2C | 1A | 1B | 6E | 5A | A0 | 52 | 3B | D6 | B3 | 29 | E3 | 2F | 84 |
| | 5 | 53 | D1 | 00 | ED | 20 | FC | B1 | 5B | 6A | CB | BE | 39 | 4A | 4C | 58 | CF |
| | 6 | D0 | EF | AA | FB | 43 | 4D | 33 | 85 | 45 | F9 | 02 | 7F | 50 | 3C | 9F | A8 |
| | 7 | 51 | A3 | 40 | 8F | 92 | 9D | 38 | F5 | BC | B6 | DA | 21 | 10 | FF | F3 | D2 |
| | 8 | CD | 0C | 13 | EC | 5F | 97 | 44 | 17 | C4 | A7 | 7E | 3D | 64 | 5D | 19 | 73 |
| | 9 | 60 | 81 | 4F | DC | 22 | 2A | 90 | 88 | 46 | EE | B8 | 14 | DE | 5E | 0B | DB |
| | a | E0 | 32 | 3A | 0A | 49 | 06 | 24 | 5C | C2 | D3 | AC | 62 | 91 | 95 | E4 | 79 |
| | b | E7 | C8 | 37 | 6D | 8D | D5 | 4E | A9 | 6C | 56 | F4 | EA | 65 | 7A | AE | 08 |
| | c | BA | 78 | 25 | 2E | 1C | A6 | B4 | C6 | E8 | DD | 74 | 1F | 4B | BD | 8B | 8A |
| | d | 70 | 3E | B5 | 66 | 48 | 03 | F6 | 0E | 61 | 35 | 57 | B9 | 86 | C1 | 1D | 9E |
| | e | E1 | F8 | 98 | 11 | 69 | D9 | 8 E | 94 | 9B | 1E | 87 | E9 | CE | 55 | 28 | DF |
| | f | 8C | A1 | 89 | 0D | BF | E6 | 42 | 68 | 41 | 99 | 2D | 0F | B0 | 54 | BB | 16 |

## 2.2 Side-channel attacks

A side-channel attack attempts to gather information or influence the program execution of a system by measuring or exploiting the physical nature of the system or its hardware. There are various known attacks, and the most common attacks are:

- Timing attack
- Electromagnetic (EM) attack
- Simple Power Analysis (SPA)
- Differential Power Analysis (DPA)
- Template attack

In this section, we shall briefly cover SPA and DPA power attacks.

### 2.2.1 Simple Power Analysis (SPA)

Simple Power Analysis (SPA) is a side-channel attack that involves visual examination of graphs of the current used by the device over time. Variations in the graph allow the attacker to determine the secret key. Very few power traces are required, and usually, a single plaintext is passed multiple times, and the mean power trace is analysed. Consider a SPA attack on the RSA algorithm, a popular asymmetric algorithm. Multiplication is carried out when the $i_{th}$ bit of the private key is one else a square operation is performed. So if the power trace of multiplication and square is differentiable, the attacker will be able to reveal the $i_{th}$ bit and eventually the attacker will be able to reveal the whole key.

### 2.2.2 Differential Power Analysis (DPA)

Differential Power Analysis (DPA) attacks are among the most popular power analysis attacks. This is because the implementation details of the attacked device are not required in great detail. Furthermore, DPA can reveal the secret key even in the presence of a large amount of noise. In contrast to SPA attacks, DPA attacks would require a large number of traces. In a DPA attack [4], the attacker analyses how fixed time instances depend on the processed data. There are five steps involved:

1. Choose an intermediate result as a function $f(d, k)$, where $d$ is the data value (either plaintext or ciphertext) and $k$ is a byte of the key.
2. Measure the power consumption while running encryption or decryption on the device, and the data set can be represented as a 2-D array S$[N_D][N_T]$, where $N_D$ is the total number of data values and $N_T$ is the total length of the trace.
3. Construct a differential average trace T from the data set S.
4. The intermediate results are then mapped to the hypothetical power consumption values, H.
5. Finally, the hypothetical values H are compared with the measured power consumption values using statistical tools like correlation and R matrix is generated. The indices of the highest values of the matrix R are used to reveal the positions at which the chosen intermediate result has been processed and the key used by the device. If all the values are approximately equal, then more traces are needed.

### 2.2.3 NIST's Side-Channel Conformance Test

No testing program can guarantee resistance against all attacks, but a practical approach should be able to validate the security of the hardware implementation. The following are the requirements for an effective validation test [3]:

- Effectiveness of tests: Reproducible results and reasonable conditions of testing
- Ease and cost-effectiveness of testing: Should not take an excessive amount of time or require exceptional test operator skills

Test Vector Leakage Assessment (TVLA) is a popular conformance style testing methodology recommended by NIST due to its robustness and ease of implementing and integrating it for various crypto-implementations with existing methodologies.

### 2.2.4 Test Vector Leakage Assessment (TVLA)

TVLA uses Welch's t-test, a PASS/FAIL test that checks if t-value crosses a predefined threshold (proposed as $\pm 4.5$). If the t-value crosses the threshold, there is a possibility of data leakage at that particular instant of time. It is to be noted that this is not sufficient to objectively tell that the device leaks data, which can be confirmed only by performing side-channel attacks. This is particularly true with processors as they may have multiple power leakage sources unrelated to the actual crypto-code that might be running. The testing strategy can be summarised as follows.

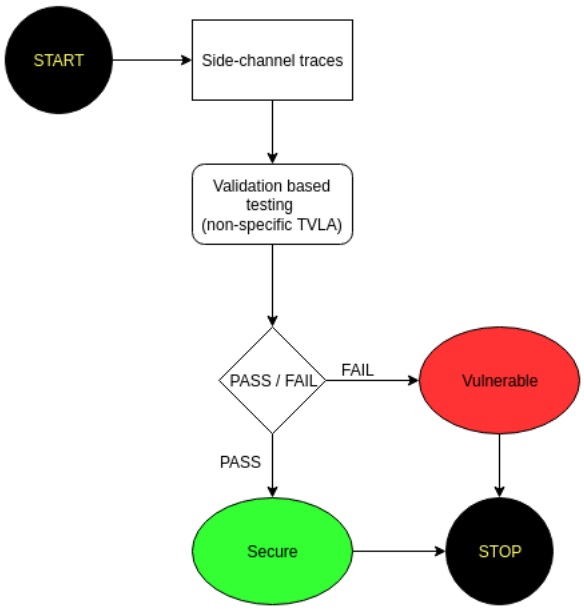



Figure 2: Testing methodology for non-specific TVLA

The ideal TVLA score (maximum t-value) should be 0, and this is the case when the mean of two sets of power traces corresponding to a fixed input and a random input are equal, which implies that the power consumed by different inputs is the same on an average at any point of time. The t-values are given using the following equation as shown below.

$t = \frac{\mu_A - \mu_B}{\sqrt{\frac{\sigma_A^2}{n_A} + \frac{\sigma_B^2}{n_B}}}$, where $\mu$ and $\sigma$ are mean and standard deviations of corresponding sets A and B.

Ideally, the variance for the fixed data should be 0, but there will be some variance due to physical randomness. High variance for fixed data will lead to a lower TVLA score which is expected as it implies higher unpredictability of identifying the immediate values. We also note the effect of the number of traces and theoritically the t-value increase proportional to $\sqrt{n}$, where $n$ is the number of traces.

There are two main categories of detection through TVLA. There is general, and there is specific detection. The general case involves feeding in fixed and random data set as inputs to the crypto-module and collecting traces for each cipher process. Specific detection usually is targetting an intermediate in a cryptographic algorithm. The target could be S-box outputs, round outputs, or an add-key operation. A random vs random data set is sufficient for this test. Both tests use a fixed key. We perform general detection on our AES accelerators with a fixed vs random data set and a fixed key for this project.

# 3 Shakti's AES Accelerator

There are three implementations of AES Accelerator made for the Shakti C-Class processor. These three give the same output for the same input, and they vary only in their implementation. This section will briefly discuss the differences and how we could interface them for side-channel analysis.

## 3.1 Lookup-table Based Implementation

This implementation uses an optimised implementation of Lookup-table based S-box for AES accelerator. Each input byte is mapped to an output byte explicitly in case constructs. LUT based S-box uses much higher number of LUTs on the FPGA and are not side-channel protected and can show significant differences in power consumption for different inputs. Figure 3 shows two traces for different inputs and they are fairly easy to distinguish visually.

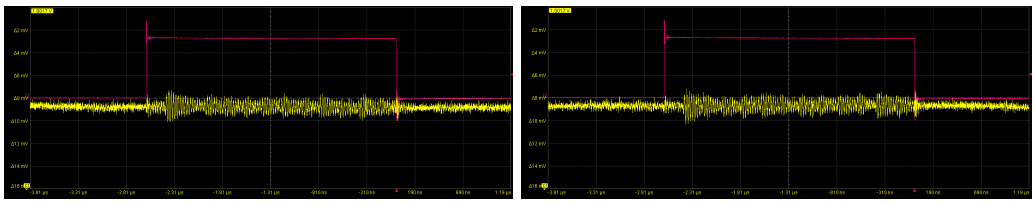


Figure 3: Plots of two traces of LUT based AES to show how the plots change for different inputs

## 3.2 Composite Field Implementation

As discussed earlier, here, we use composite field arithmetic to reduce the byte-to-byte mapping to simpler operations. This allows for modifications for side-channel resistance, and based on our previous research [1], we found it to occupy a lesser area. Our implementation of Forward S-box uses only 39 LUTs. The rest of the modules for both Composite Field and Lookup-table based implementations are the same.

In Composite Field theory, a higher dimension vector is projected to components of vectors in the lower dimensions. We can then derive efficient computations otherwise cumbersome in the original higher dimension space. $GF(2^8)$ involves expensive inverse multiplication and can be reduced to inverse operations in $GF(2^4)$. $GF(2^4)$ can be further reduced to $GF(2^2)$, but $GF(2^4)$ is more suitable for FPGAs for their 4/6-input LUTs and $GF(2^2)$ operations are more suited for ASICs.

For our composite field $GF((2^4)^2)$ [1], these are the irreducible polynomials used.

| SBox | Q(z) | P(y) | $g_1$ | $g_2$ | #LUTs |
|---|---|---|---|---|---|
| Forward | $z^4 + z + 1$ | $y^2 + y + 13$ | 16 | 31 | 39 |
| Inverse | $z^4 + z + 1$ | $y^2 + y + 13$ | 16 | 31 | 40 |

where P(y) and Q(z) are the polynomials and g1, g2 are generators for the composite and AES field. We can see from the plots in Figure 4 how similar the traces are for different inputs.

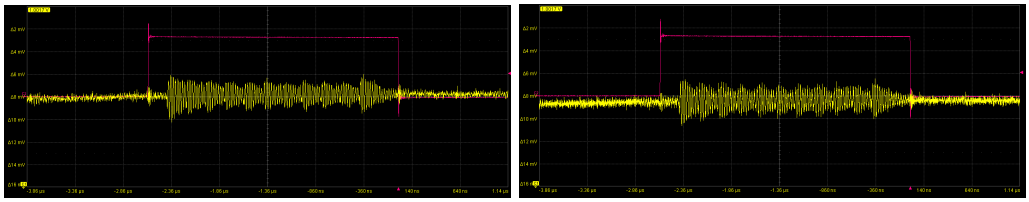


Figure 4: Plots of two traces of Composite AES to show how the plots change for different inputs

In the *SubBytes* stage, the 128-bit text is split into 16 bytes and each is fed into one S-box. S-boxes are also used in Key generation modules. In our implementation, Key generation is done separately before the cipher process as the same set of round keys are used for all the processes. The main component of the 11 peaks in the power trace is the 16 S-box transformations in each round.

### 3.3 Threshold Implementation

Threshold implementation [5] of AES is built to ensure more security of the S-box. Each input to the S-box, $X$, is split across $n$ shares such that they follow the following property:

$$\sum_{i=1}^{n} X_i = X \tag{1}$$

$$\sum_{i=1}^{n} F(X_i) = F(X) = Y \tag{2}$$

Additionally, even the output can be shared across $m$ shares which sum up to give $Y$.

$$\sum_{i=1}^{m} Y_i = Y \; where Y_i = F_i(X_1, X_2, X_3, ..., X_n) \tag{3}$$

The function $F_i$ is identified such that it is independent of the input $X_i$. This ensures that when each share is being executed a particular $X$ component is not revealed. This uniform masking makes the leaking signal almost a constant at all instants of time.

Plots are added below in Figure 5, but it is difficult to differentiate between them due to very low increase in power consumption. The low power consumption is because only one S-box runs in a cycle. The S-box for threshold implementation consumes more resources than the Composite Field based S-box, and hence this strategy was followed.

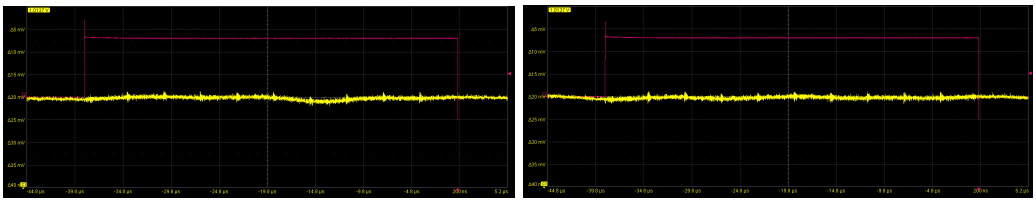


Figure 5: Plots of two traces of Threshold AES to show how the plots change for different inputs

## 3.4 Implementation details

In this section, we briefly describe the implementation aspects of the three accelerators. Among them, Lookup-table based and Composite Field implementations use typical AES top modules while Threshold implementatioon uses a different set of top modules. As discussed earlier, the only difference between the the top modules is that the typical version uses 16 S-boxes in parallel while the other uses only one. Typical AES accelerator can be embedded with either LUT based or Composite S-box.

The modules for typical AES accelerator have been modified for ease of usage so that only the imported module needs to be changed to switch between the S-box modules. The following figure shows the methods of the top modules which are used for one cipher process. "genKeys" is used to initiate key generation, "encrypt" is used to pass the plaintext for processing and "ret" returns the output of the cipher process. Other methods are also present which can be used to check the status of the process.

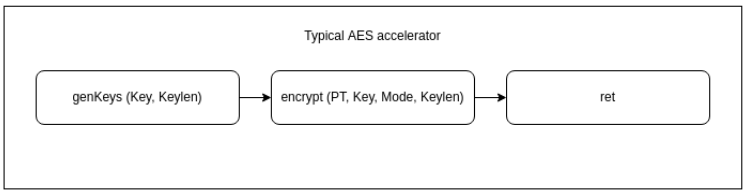



Figure 6: Flow diagram for Typical AES module

Threshold's top module is slightly different as here the "encrypt" method additionally checks if the Key received has been expanded before and then continues with the cipher process.

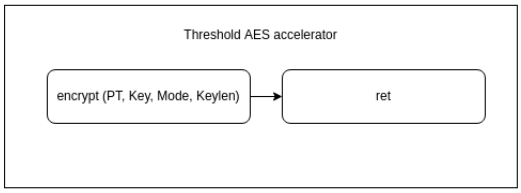



Figure 7: Flow diagram for Threshold AES module

Following table sums up the different physical aspects of each implementation. The hardware utilisations and maximum operable frequency of each accelerator was found by implementing the accelerators on our FPGA. The wrapper module used here is Wrapper2 and details about wrapper modules and FPGA are discussed in the next section.

| Accelerator | Time taken for encryption (in cycles) | Time taken for decryption (in cycles) | LUTs | FFs | Max frequency (in MHz) |
|---|---|---|---|---|---|
| LUT based AES | 12 | 13 | 5446 | 1216 | 197.03 |
| Composite AES | 12 | 13 | 3149 | 969 | 104.01 |
| Threshold AES | 181 | 193 | 2550 | 295 | 148.05 |

# 4 Side-channel evaluation of accelerators

This section covers the setup followed for the side-channel evaluation of our AES accelerator. Similar procedure can be followed for other accelerators too. The following are the steps involved:

- Make a wrapper module for the accelerators to feed the inputs to the crypto-module
- Configure the module for the FPGA with another Top file
- Configure the oscilloscope as required and program the FPGA with our module
- Collect the traces and evaluate the TVLA score using Python scripts

We begin by making a wrapper module for the accelerators discussed in the previous section. All the modules of the accelerator and wrapper are written in Bluespec SystemVerilog and Verilog files are generated from this. The top module for configuring the FPGA is written in Verilog HDL.

Two wrapper modules were made using different approaches. For Wrapper1, we have four stages as shown in Figure 8 and for Wrapper2, we have five stages as shown in Figure 9. Details about the wrappers have been given later.

We use the SASEBO-GIII board as the platform for evaluation and the Teledyne LeCroy HDO4104 oscilloscope for collecting traces. After that, the Python script is run to compute the TVLA score for each set of traces. More details are provided in the following subsections.

## 4.1 AES Wrapper modules

There are multiple ways to pass the inputs, and we have chosen to create a wrapper file for the accelerator to feed fixed and random inputs to the core module. Random inputs are generated by feeding the cipher output of the previous random input. This is in conformance with NIST standards. We have made two variants to compare the effects: in one (Wrapper1), the output of the encryption/decryption was stored in a 128-bit register within the trigger along with other register updates, and in the other (Wrapper2), the total number of flip-flop changes in the wrapper module was limited to 2 bits within the trigger.

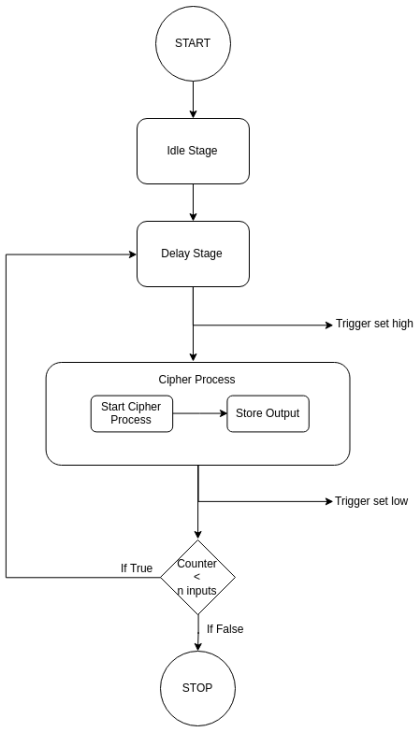



Figure 8: Working of Wrapper1 module

As can be seen from Figure 8, Wrapper1 is a simple module with four stages and three of them repeating till n inputs have been fed. The delay stage is very crucial as the oscilloscope needs some time between two traces to properly store the power values. In Figure 9, the working of Wrapper2 module is shown and the main difference is that the cipher process is repeated twice.

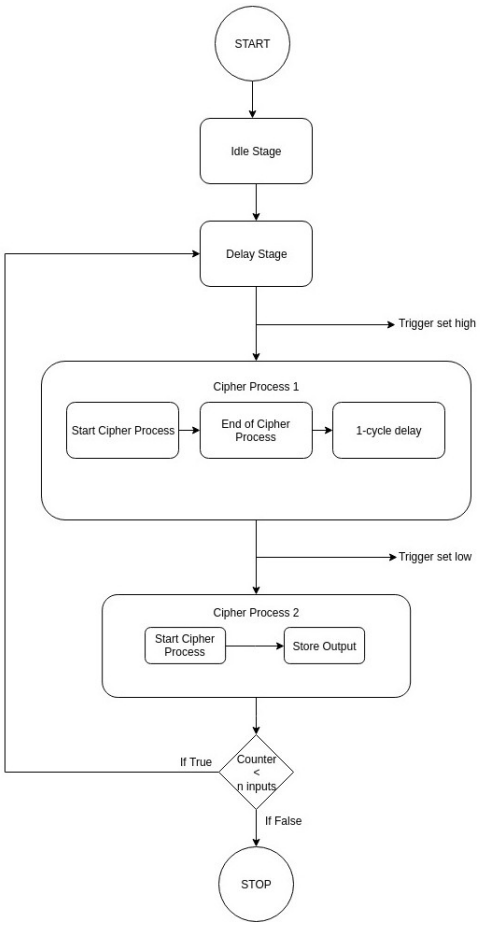



Figure 9: Working of Wrapper2 module

## 4.2 SASEBO-GIII board

The SASEBO-GIII is a successor of the SASEBO-GII board and is for further side-channel attack experimentation. The basic features of SASEBO-GIII are as follows:

- 200 mm x 150 mm x 1.6mm, FR-4, eight layers
- Two Xilinx FPGAs:
  - Cryptographic FPGA: Kintex-7 XC7K160T-1FBGC (part no.: xc7k160tfbg676-1)
  - Control FPGA: Spartan-6 XC6LX45-2FGG484C
- Differential clock at frequency of 200 Mhz.
- 1 Gigabit DDR3 SDRAM
- The external power source supplies the onboard power regulators and the FPGAs with 5.0 V
- A shunt resistor is provided to insert on the core VDD line of the cryptographic FPGA for measuring power traces.

The host PC controls and communicates with the board via the J-TAG port. The FPGA is default programmed with the vendor's Bit file. The default Bit file can be found on the company's website and can be used to collect traces using the C# project, SASEBO G-CHECKER.

For this project, Xilinx Vivado 2020.1 was used to program the Kintex-7 FPGA with our AES accelerator. The SASEBO Board uses a differential clock and it needs to be converted into single-ended clock using a clock wizard. In Vivado 2020.1, it can be found as the Xilinx IP, clk_wizard. The single-ended clock can be configured for frequencies from 4.7 MHz to 800 MHz. Differential clocks have the advantage that it is based on the difference between two signals and are less affected by noise, and require lower supply voltages.

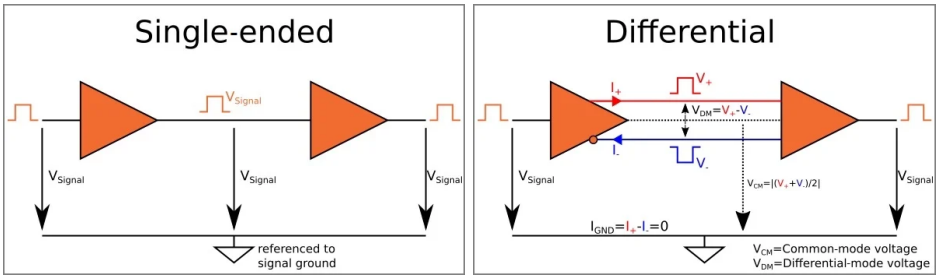



Figure 10: Distinctions between a single-ended and differential clock

## 4.3 Oscilloscope

Teledyne LeCroy HDO4104 was used for collecting the power traces from the FPGA. Important features of this oscilloscope are as follows:

- Bandwidth of $1GHz$
- Maximum sampling rate of $2.5GS/s$
- Resolution of 12 bits
- Four input channels
- Processor/CPU - Intel Celeron B810 Dual-Core, 1.6GHz
- Memory of 4GB RAM and 500GB disk
- Oscilloscope Operating Software - Teledyne LeCroy MAUI with OneTouch

Other features also exist, such as trigger types, filters for analysis and remote control setup. The oscilloscope uses an initial setup file, aes-initial-verilog--00000.lss. This uses default configurations and sets all the system variables.

For collecting traces, both C1 and C2 channels are used. C1 is connected to the measuring port of the FPGA and C2 collects the trigger signal from the board pins. Data is collected whenever the trigger is set high. We use a power probe for C1 and a single hook probe for C2.

We set the sampling rate to $10kS$, which is the number of samples to be collected in each trigger. This is not achievable in case of the typical AES accelerator (non-threshold AES), and instead, the oscilloscope will reduce the sampling rate on its own to around 5000-6000 samples per trigger based on the resolution. The sample size will be fixed for all traces provided we ensure in our design that the trigger is set for the same number of cycles for each trace. The oscilloscope can also synchronise triggers to give a clear and stable display of the power traces.

## 4.4 Code analysis of Python scripts

After collecting the traces, the python code TVLA_threaded.py can be used to compute the TVLA score. This code involves multithreading, and a corresponding sequential code was also made and can be found as TVLA_basic.py. The scores from the two codes differ only by 0.001% which is occuring due to floating point precision errors.

### 4.4.1 Libraries imported

The following libraries are imported:

```
import os
import pandas as pd
import numpy as np
import csv
import threading as th
from time import time
from joblib import Parallel, delayed
```

### 4.4.2 TVLA_basic.py

TVLA_basic.py is a simple sequential code that can read a file iteratively and accumulate the total sum and sum of squares for mean and variance computation. Power consumption at each instant of time is present in consecutive rows, and the t-value is found for each row separately. The following summations were used so that all the trace files need not be read and stored together for processing. This was done to prevent RAM from getting full and to also allow parallel processing of data which we utilize in the next script.

$$Mean_j(\mu_j) = \frac{1}{N}\sum_{k=1}^{N} row_k[j] \tag{4}$$

$$Variance_j(\sigma_j) = \frac{1}{N}(\sum_{k=1}^{N} row_k[j]^2) - \mu_j^2 \tag{5}$$

where N is the number of traces, k is the kth trace and j is jth instant of time during the trigger

This computation is done separately for dataset 1 and dataset 2. T-value is computed at every time instant, and the highest value is displayed in the end along with the time taken.

### 4.4.3 TVLA_threaded.py

This code is similar to the previous one but with the added advantage of threads. Threads are lightweight processes and allow parallel processing of data. If we are to use threads, we should try to ensure that the number of threads getting generated is some multiple of the number of cores we are using so that we get maximum efficiency. Threads are introduced using the Joblibs library.

To compute the total sum and sum of squares, we use reduction sum approach. This was found to be significantly faster than each thread returning values reading from a file. A function, *partial_accumulator()*, is defined which takes the parameters *csv_files*, loop variable *i*, *width* and number of traces. Using *Parallel()*, we generate threads, and each will perform this function.

We use a parameter "*width*" for which each thread will compute the partial sums. For good efficiency, the total number of traces by *width* should be a multiple of the number of cores. Other parameter in this code is the number of cores which ideally should be the number of cores in the computing system used.

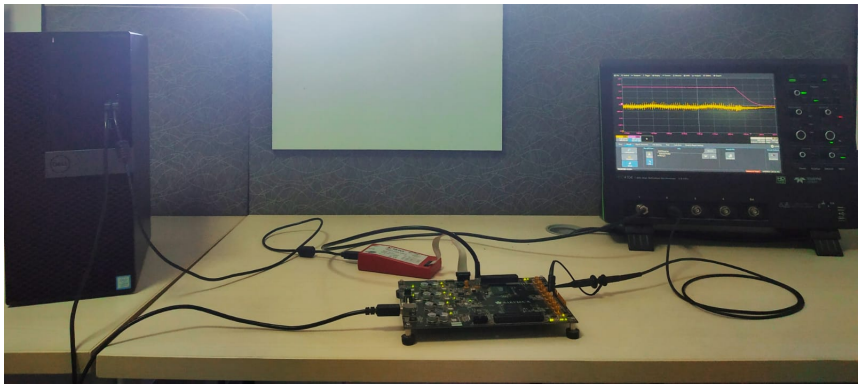
# 5 Evaluation of AES accelerators

Both encryption and decryption processes were evaluated in separate experiments for each of our AES accelerators. For the TVLA test, two sets of data [3] are to be passed by the Wrapper modules for both encryption and decryption.

1. Data-set 1:
   (a) 256-bit Constant Key is set to 0x0123456789abcdef123456789abcdef023456789abcdef013456789abcdef012 and Key length is set to Bit128
   (b) $n$ 128-bit fixed input is set to 0xda39a3ee5e6b4b0d3255bfef95601890 where $n$ is the number of times the fixed input is fed
2. Data-set 2:
   (a) 256-bit Constant Key is set to 0x0123456789abcdef123456789abcdef023456789abcdef013456789abcdef012 and Key length is set to Bit128
   (b) $n$ 128-bit inputs: $I_0$, $I_1$, ..., $I_n$, where $I_0 = 0x0$, $I_j = AES(Key, I_j, mode)$ for $0 < j \leq n$ where $n$ is the number of inputs and mode is for choosing between encryption and decryption

As it can be noted, the same Key is used for both the datasets and the same number of inputs were fed for both the datasets. NIST recommends dispersing the two datasets while emulating them, so we passed the inputs from both the datasets alternatively. We use only Kintex7 (part no.: xc7k160tfbg676-1) of the SASEBO-GIII board, and it was programmed using Xilinx Vivado 2020.1.

The following picture is the setup used for the above three accelerators.



Two connections were given from the FPGA to the oscilloscope, as shown in the above picture. C1 of the oscilloscope is connected to the power measuring port, and C2 of the oscilloscope collects the trigger signal from the board's pins. The Xilinx Platform Cable connects to the FPGA via the JTAG interface and is used for programming the FPGA using Vivado.

To feed the inputs to the accelerators we use wrapper modules. The wrapper files have the following ports:

```
input  CLK;
input  RST_N;

// value method trigger_pin
output trigger_pin;

// value method done_signal
output done_signal;
```

```
// value method output_fix
output [127 : 0] output_fix;
```

In the Top module above the wrapper file, we have given just the minimum required configurations required for the FPGA. To convert the differential clock of the FPGA to a single-ended clock, we added a Xilinx IP, clk_wizard. This will be present in the Top module which is written in Verilog HDL. The input frequency needs to be set to the SASEBO board's frequency of 200MHz, and the output frequency was set to 5MHz. Reset is set as active high.

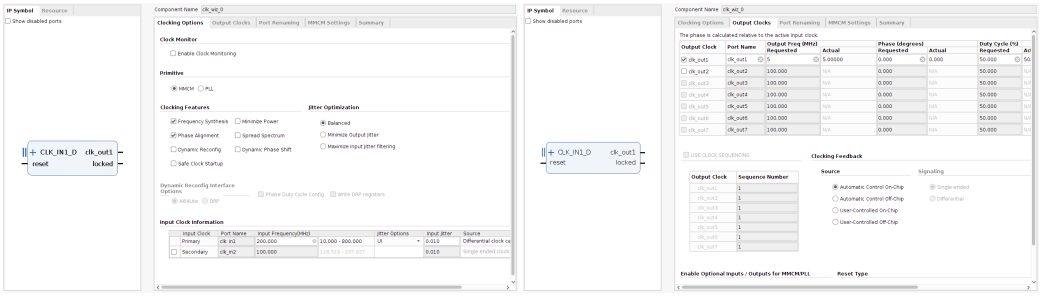


Figure 11: Clk_wizard configuration

The following constraint file was used for the FPGA. Vivado's default settings were used for synthesis and implementation.

```
## Clock configuration
set_property -dict {PACKAGE_PIN AA3 IOSTANDARD LVDS} [get_ports CLK_P]
set_property -dict {PACKAGE_PIN AA2 IOSTANDARD LVDS} [get_ports CLK_N]

## RESET configuration
set_property -dict {PACKAGE_PIN L23 IOSTANDARD LVCMOS25} [get_ports RST_N]

## GPIO configuration
set_property -dict {PACKAGE_PIN E23 IOSTANDARD LVCMOS25} [get_ports gpio0]
set_property -dict {PACKAGE_PIN N17 IOSTANDARD LVCMOS25} [get_ports gpio1]
set_property -dict {PACKAGE_PIN P24 IOSTANDARD LVCMOS25} [get_ports gpio2]

# General configuration
set_property BITSTREAM.General.UnconstrainedPins {Allow} [current_design]
```

The oscilloscope is set to sample $10kS$ points for each trigger, and the trigger mode is set to window trigger. For the proper collection of traces, a large delay of 600ms is given between two traces in the Wrapper module. This is about the time required for the oscilloscope to save the collected data before sampling the next one. The approximate time taken to collect $50K$ traces is 8 hours. After generating traces, it is suggested to move the files from one device to another in a zip folder to prevent any data loss during transfer. After collecting the data, we unzip the folder and set the appropriate parameters in the python script and execute the code.

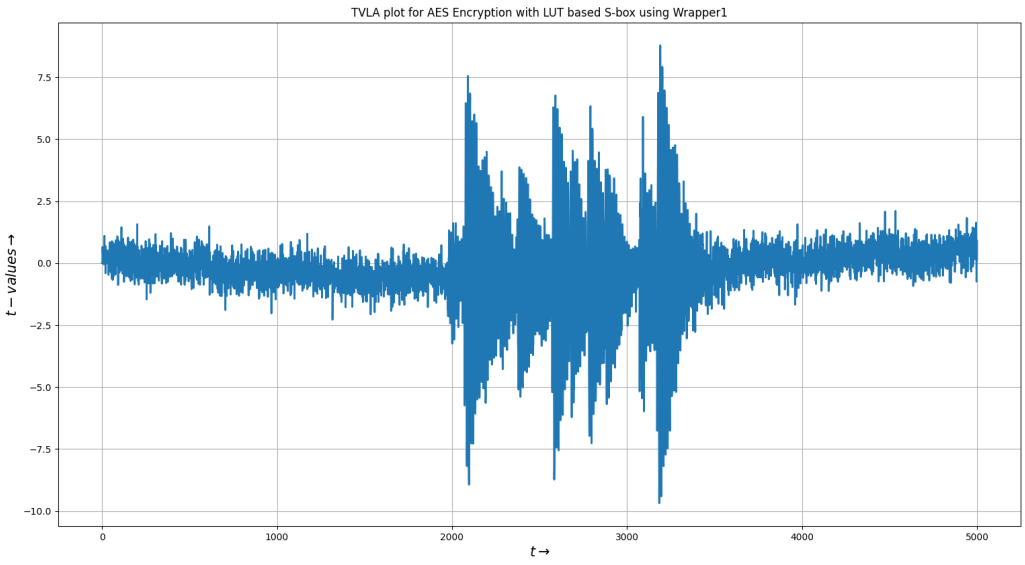
## 5.1 Evaluation of Lookup-table Based Implementation

For the typical AES accelerator with LUT based S-box, 25,000 inputs were given from each dataset. We perform four tests: encryption process with both the wrappers and decryption process for both the wrappers. The number of clock cycles one process takes is 12 for encryption and 13 for decryption and the FPGA's frequency is set at 5 MHz. Using our Teledyne Lecroy oscilloscope we set the maximum sampling rate at 10K samples per trigger. We set the oscilloscope to resolution of $2mV/div$ and $500ns/div$. For the python script, the number of cores is set to 10 and the width is set to 1000. The following table depicts the implementation details for comparing the two wrapper modules.

| Wrapper module | LUTs | FFs | Max frequency (in MHz) |
|---|---|---|---|
| Wrapper1 | 5562 | 1213 | 194.92 |
| Wrapper2 | 5446 | 1216 | 197.03 |

### Evaluation with Wrapper1

Following plots are the TVLA plots for encryption and decryption process using Wrapper1. The sampling rate was at 5001 samples per trigger for both the processes. Note that the x-axis is in sample units.



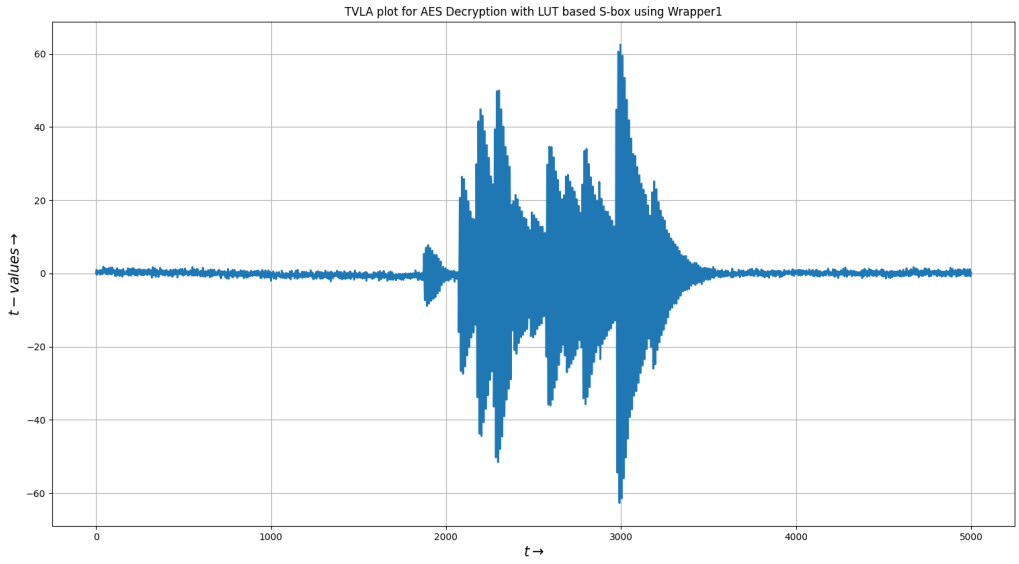



Figure 12: Plots for Encryption and Decryption process for LUT based S-box with Wrapper1

The same starting random input, $0x00$, was used for both the tests. We have also calculated the TVLA score for different number of traces.

Table 3: TVLA scores for encryption with LUT based S-box using Wrapper1

| S.no. | Number of Traces | Time taken (in sec) | TVLA Score |
|---|---|---|---|
| 1. | 100 | 5.836 | 3.000 |
| 2. | 500 | 32.685 | 2.608 |
| 3. | 1000 | 64.361 | 2.518 |
| 4. | 2000 | 79.695 | 3.176 |
| 5. | 5000 | 148.525 | 3.517 |
| 6. | 10000 | 189.343 | 3.635 |
| 7. | 20000 | 360.327 | 4.885 |
| 8. | 49998 | 1011.257 | 8.782 |

Table 4: TVLA scores for decryption with LUT based S-box using Wrapper1

| S.no. | Number of Traces | Time taken (in sec) | TVLA Score |
|---|---|---|---|
| 1. | 100 | 7.090 | 3.629 |
| 2. | 500 | 30.081 | 7.622 |
| 3. | 1000 | 53.970 | 9.505 |
| 4. | 2000 | 64.067 | 13.090 |
| 5. | 5000 | 90.483 | 19.197 |
| 6. | 10000 | 155.869 | 26.841 |
| 7. | 20000 | 351.947 | 38.167 |
| 8. | 50000 | 965.101 | 62.552 |

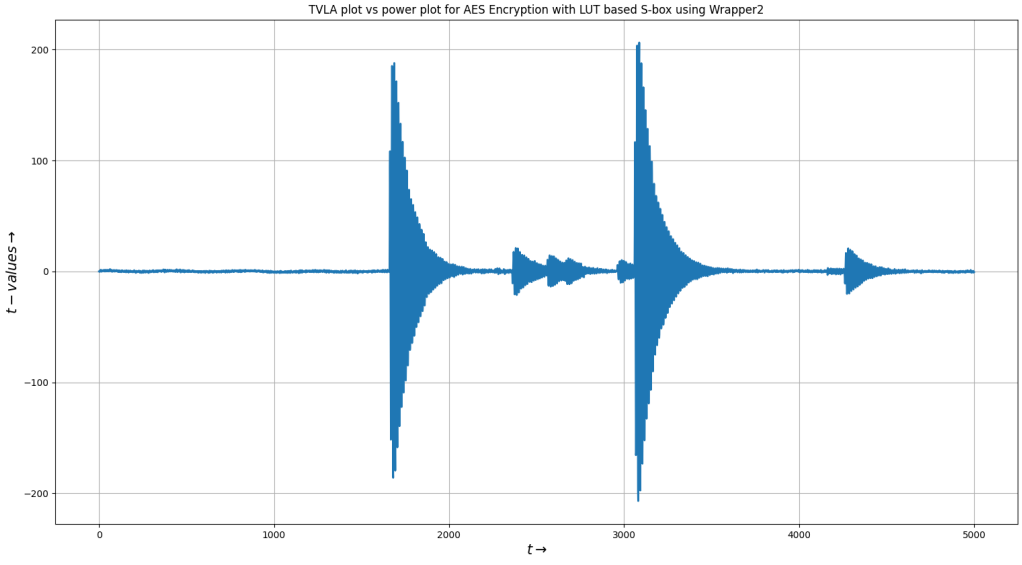
Additionally, we have run the TVLA scripts for different width (same number of cores) to see which is the most efficient. For this only one set of data with 50K traces, traces for encryption with LUT based S-box using Wrapper1, was used. Due to optimizations in the memory access patterns, the time taken values in Table 5 are smaller than in Table 3.

Table 5: Analysis of width vs time taken for 50K traces

| S.no. | Width | Time taken (in sec) |
|---|---|---|
| 1. | 200 | 546.707 |
| 2. | 500 | 522.640 |
| 3. | 1000 | 521.425 |
| 5. | 1250 | 570.895 |
| 6. | 2500 | 687.077 |
| 7. | 5000 | 723.229 |

**Evaluation with Wrapper2**

Following plots are the TVLA plots for encryption and decryption process using Wrapper2. The sampling rate was at 5001 and 10001 samples per trigger for encryption and decryption process respectively.



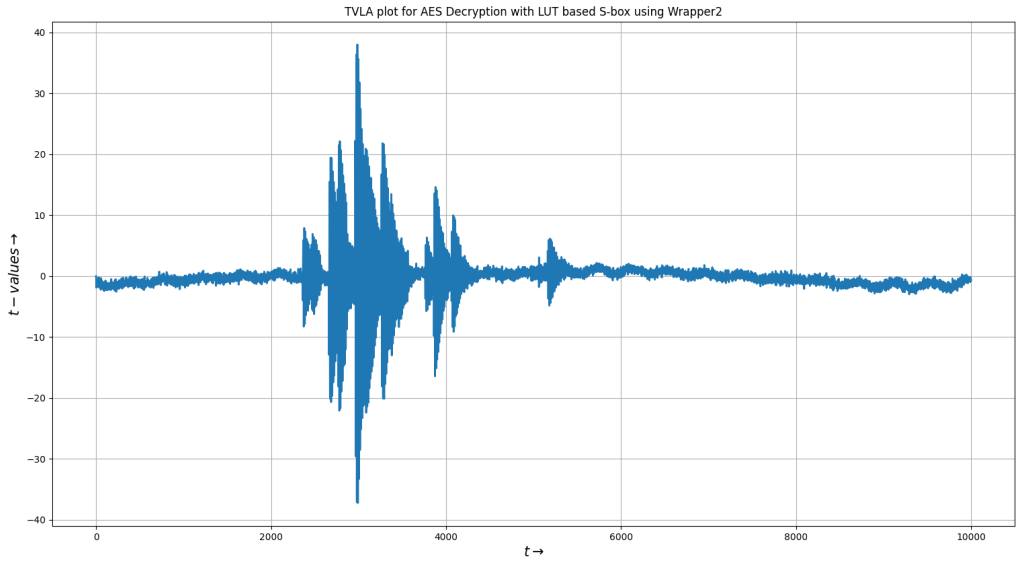



Figure 13: Plots for Encryption and Decryption process for LUT based S-box with Wrapper2

Similar to previous, the same starting random input, $0x00$, was used for both the tests and TVLA score was calculated for different number of traces.

Table 6: TVLA scores for encryption with LUT based S-box using Wrapper2

| S.no. | Number of Traces | Time taken (in sec) | TVLA Score |
|---|---|---|---|
| 1. | 100 | 8.051 | 4.738 |
| 2. | 500 | 33.238 | 16.905 |
| 3. | 1000 | 60.964 | 27.359 |
| 4. | 2000 | 78.048 | 40.734 |
| 5. | 5000 | 98.988 | 65.023 |
| 6. | 10000 | 144.303 | 94.032 |
| 7. | 20000 | 318.593 | 131.043 |
| 8. | 50000 | 882.578 | 206.301 |

Table 7: TVLA scores for decryption with LUT based S-box using Wrapper2

| S.no. | Number of Traces | Time taken (in sec) | TVLA Score |
|---|---|---|---|
| 1. | 100 | 12.137 | 3.420 |
| 2. | 500 | 49.048 | 7.031 |
| 3. | 1000 | 98.621 | 9.676 |
| 4. | 2000 | 99.773 | 13.730 |
| 5. | 5000 | 115.599 | 21.093 |
| 6. | 10000 | 190.690 | 30.530 |
| 7. | 20000 | 363.875 | 3.557 |
| 8. | 50000 | 918.135 | 38.018 |

Time analysis of workload for each thread is not done for these datasets as it gave similar results as the previous one.

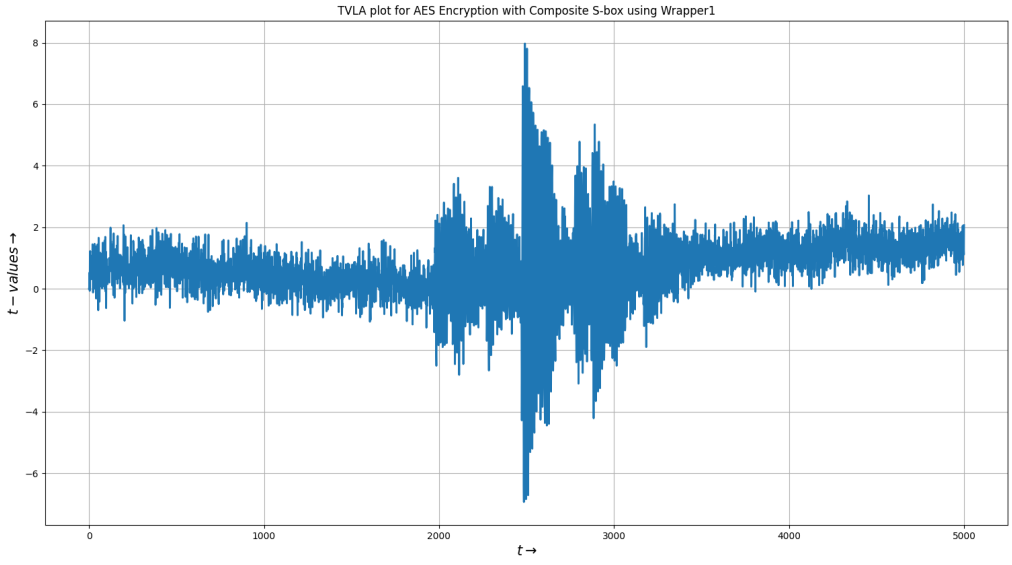
## 5.2 Evaluation of Composite Field Implementation

For the typical AES accelerator with Composite S-box too, we give 25,000 inputs from each dataset and separately conduct four tests for encryption and decryption processes with both the wrappers. This also takes the same number clock cycles as the previous. Using our Teledyne Lecroy oscilloscope we set the maximum sampling rate at 10K samples per trigger. We set the oscilloscope to resolution of $2mV/div$ and $500ns/div$. For the python script, the number of cores is set to 10 and the width is set to 1000. The following table depicts the implementation details for comparing the two wrapper modules.

| Wrapper module | LUTs | FFs | Max frequency (in MHz) |
|---|---|---|---|
| Wrapper1 | 3144 | 967 | 110.3 |
| Wrapper2 | 3149 | 969 | 104.01 |

### Evaluation with Wrapper1

Following plots are the TVLA plots for encryption and decryption process using Wrapper1. The sampling rate was at 5001 samples per trigger for both the processes.



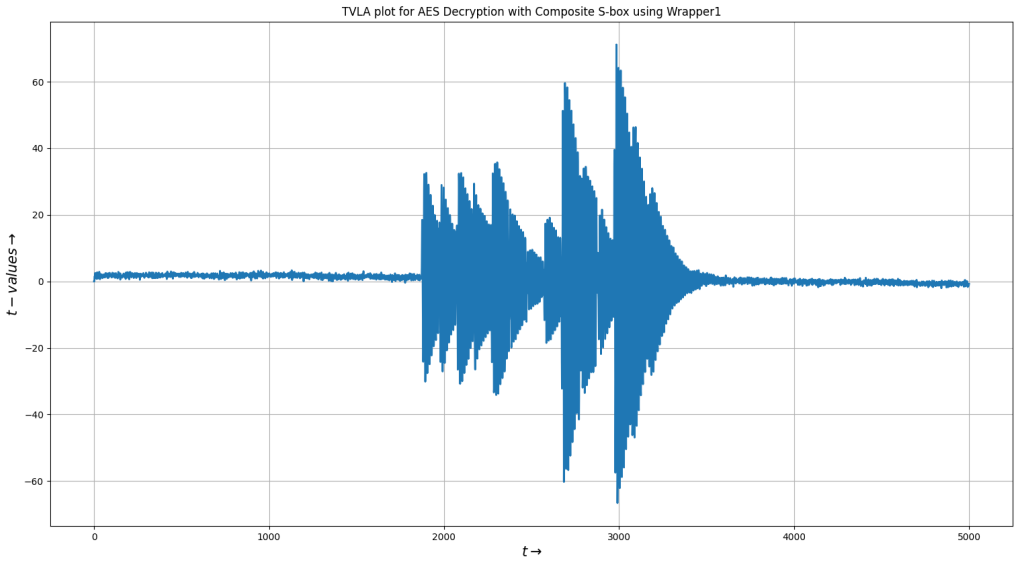



Figure 14: Plots for Encryption and Decryption process for Composite AES with Wrapper1

TVLA scores were also calculated for Wrapper1.

Table 8: TVLA scores for encryption with Composite AES using Wrapper1

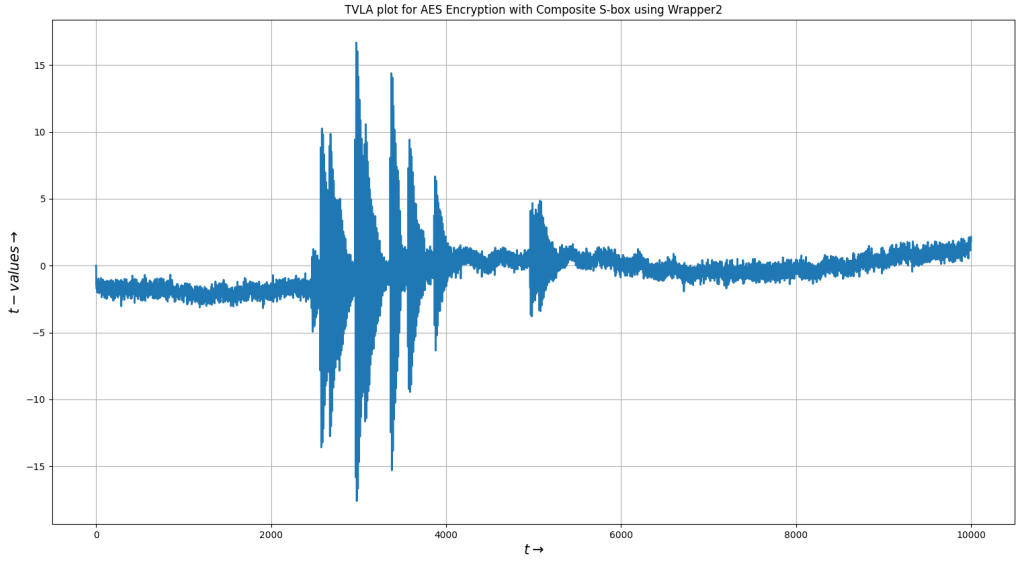
| S.no. | Number of Traces | Time taken (in sec) | TVLA Score |
|---|---|---|---|
| 1. | 100 | 7.265 | 4.109 |
| 2. | 500 | 32.266 | 3.819 |
| 3. | 1000 | 57.585 | 4.737 |
| 4. | 2000 | 70.652 | 6.027 |
| 5. | 5000 | 101.075 | 7.476 |
| 6. | 10000 | 156.559 | 1.851 |
| 7. | 20000 | 346.973 | 9.063 |
| 8. | 50000 | 999.713 | 7.971 |

Table 9: TVLA scores for decryption with Composite AES using Wrapper1

| S.no. | Number of Traces | Time taken (in sec) | TVLA Score |
|---|---|---|---|
| 1. | 100 | 7.334 | 3.366 |
| 2. | 500 | 32.393 | 7.673 |
| 3. | 1000 | 59.144 | 9.634 |
| 4. | 2000 | 70.229 | 14.222 |
| 5. | 5000 | 103.959 | 23.092 |
| 6. | 10000 | 157.114 | 31.912 |
| 7. | 20000 | 350.590 | 45.939 |
| 8. | 50000 | 1082.183 | 71.183 |

### Evaluation with Wrapper2

Following plots are the TVLA plots for encryption and decryption process using Wrapper2. The sampling rate was at 10001 samples per trigger for both the processes.



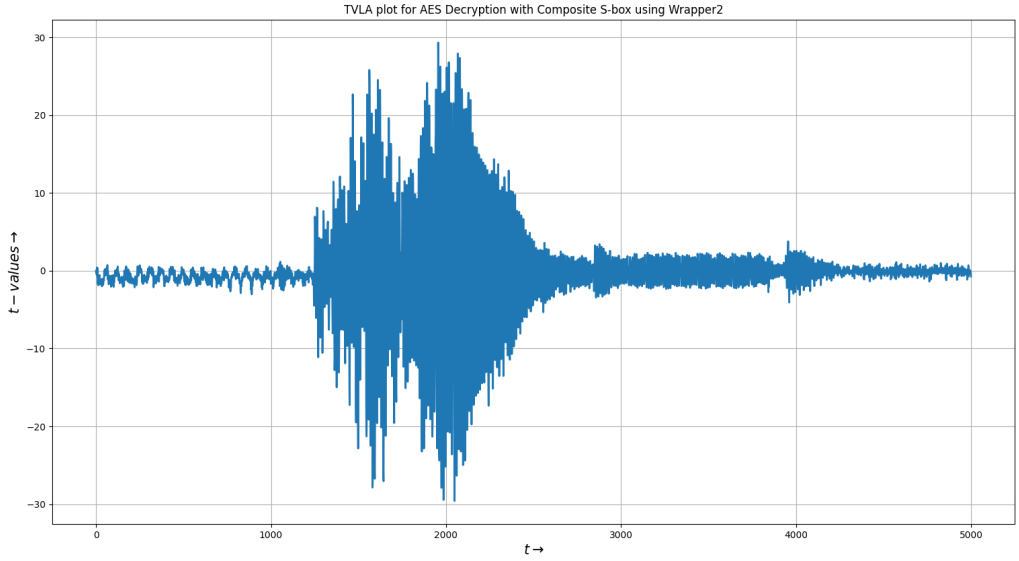




Figure 15: Plots for Encryption and Decryption process for Composite AES with Wrapper2

TVLA scores were also calculated for Wrapper2.

Table 10: TVLA scores for encryption with Composite AES using Wrapper2

| S.no. | Number of Traces | Time taken (in sec) | TVLA Score |
|---|---|---|---|
| 1. | 100 | 9.955 | 2.533 |
| 2. | 500 | 47.769 | 2.927 |
| 3. | 1000 | 97.101 | 2.503 |
| 4. | 2000 | 97.458 | 4.087 |
| 5. | 5000 | 99.738 | 5.625 |
| 6. | 10000 | 179.276 | 7.979 |
| 7. | 20000 | 369.946 | 10.116 |
| 8. | 50000 | 952.971 | 16.694 |

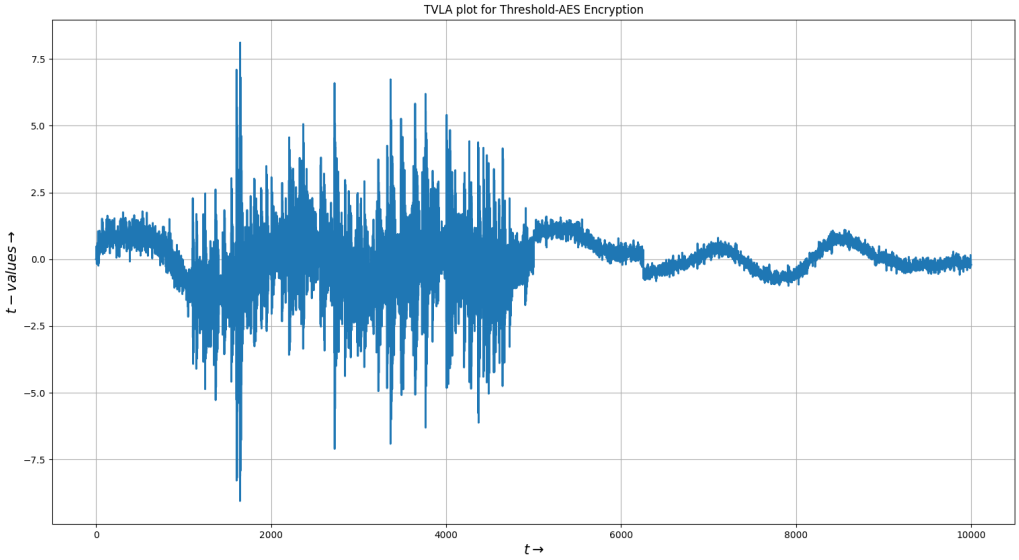
Table 11: TVLA scores for decryption with Composite AES using Wrapper2

| S.no. | Number of Traces | Time taken (in sec) | TVLA Score |
|---|---|---|---|
| 1. | 100 | 11.589 | 1.632 |
| 2. | 500 | 56.157 | 5.512 |
| 3. | 1000 | 107.000 | 4.695 |
| 4. | 2000 | 109.182 | 5.859 |
| 5. | 5000 | 142.772 | 3.812 |
| 6. | 10000 | 205.047 | 11.169 |
| 7. | 20000 | 487.546 | 22.246 |
| 8. | 49994 | 1229.611 | 29.331 |

## 5.3 Evaluation of Threshold Implementation

For the Threshold AES accelerator, approximately 5,00,000 traces were collected for each dataset. The oscilloscope has a limit of collecting a maximum of 1 lakh traces in each run and so we collected the output in 10 batches by feeding 50,000 inputs from each dataset. The starting random input for each batch was predetermined by simulating the Bluespec code. Unlike the typical AES accelerator, the Threshold implementation takes atleast 181 cycles for one AES process. We set the oscilloscope to resolution of $5mV/div$ and $5\mu s/div$ and set the maximum sampling rate at 10K samples per trigger. It was difficult to differentiate between the cipher process's power signal and noise but at this resolution some small peaks can be observed within the trigger range. From previous TVLA analysis, we can notice the major impact on TVLA score because of storing the AES output into registers within the trigger. Hence, for this accelerator only Wrapper2 module was used and it has given expected improvements. The implementation aspects can be found in Section 3.4. For the python script, the number of cores is set to 10 and the width is set to 10000.

Following plots are the TVLA plots for encryption and decryption process using Wrapper2.



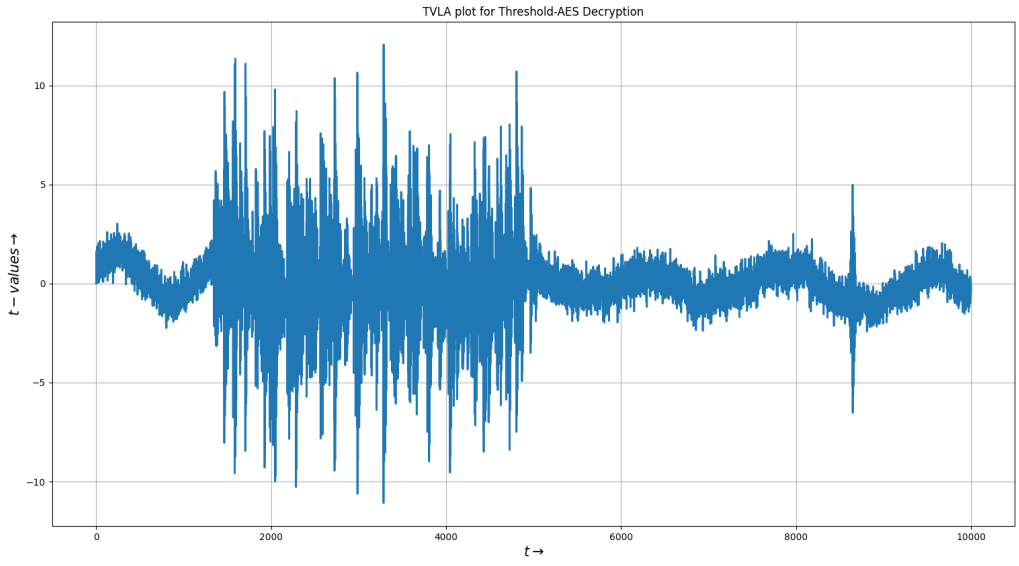




Figure 16: Plots for Encryption and Decryption process for Threshold AES with Wrapper2

TVLA scores were also calculated for Threshold AES accelerator with Wrapper2.

Table 12: TVLA scores for encryption with Threshold AES with Wrapper2

| S.no. | Number of Traces | Time taken (in sec) | TVLA Score |
|---|---|---|---|
| 1. | 100 | 20.778 | 2.634 |
| 2. | 500 | 74.766 | 3.148 |
| 3. | 1000 | 161.504 | 3.030 |
| 4. | 2000 | 284.405 | 3.731 |
| 5. | 5000 | 664.440 | 4.911 |
| 6. | 10000 | 1281.549 | 6.570 |
| 7. | 20000 | 2029.442 | 9.354 |
| 8. | 50000 | 1317.728 | 2.737 |
| 9. | 100000 | 2718.376 | 5.323 |
| 10. | 999190 | 34915.043 | 8.136 |

Table 13: TVLA scores for decryption with Threshold AES with Wrapper2

| S.no. | Number of Traces | Time taken (in sec) | TVLA Score |
|---|---|---|---|
| 1. | 100 | 18.261 | 3.944 |
| 2. | 500 | 65.443 | 4.063 |
| 3. | 1000 | 111.389 | 4.737 |
| 4. | 2000 | 223.350 | 6.189 |
| 5. | 5000 | 533.070 | 8.021 |
| 6. | 10000 | 1196.762 | 10.278 |
| 7. | 20000 | 1945.871 | 13.323 |
| 8. | 50000 | 1352.587 | 12.071 |
| 9. | 100000 | 3636.202 | 5.519 |
| 10. | 999154 | 63006.934 | 16.607 |

Additionally, we have run the TVLA scripts for different width (same number of cores) to see which is the most efficient. For this we have used only the traces collected during encryption. We can see from Table 14 that Width of around 1000 works most efficiently.

Table 14: Analysis of width vs time taken for ≈1 lakh traces

| S.no. | Width | Time taken (in sec) |
|---|---|---|
| 1. | 200 | 1143.268 |
| 2. | 500 | 1066.453 |
| 3. | 1000 | 1024.979 |
| 5. | 1250 | 1022.161 |
| 6. | 2500 | 1082.881 |
| 7. | 5000 | 1327.380 |
| 8. | 10000 | 1692.829 |

# 6 Results and Conclusion

Thus, we have evaluated three implementations of the AES accelerator using the Welch's t-test which is the Lookup-table Based Implementation, Composite Field Implementation and Threshold AES Implementation. From our previous research [1], Composite Field implementation is expected to be more secure than LUT based AES implementation. We got the result as expected when we used Wrapper2 but the results were ambiguous with Wrapper1.

Table 15: Results between LUT based and Composite Field implementations for 50K traces

| Wrapper type | LUT Based Implementation | | Composite Field Implementation | |
|---|---|---|---|---|
| | Encryption | Decryption | Encryption | Decryption |
| Wrapper1 | 8.782 | 62.552 | 7.971 | 71.183 |
| Wrapper2 | 206.301 | 38.018 | 16.694 | 29.331 |

In Wrapper1 module, the following registers are changing during the trigger:

- 2-bit *mod_state* changes during output stage and delay stage
- 32-bit *counter* increments once
- 1-bit *switcher*, to switch between fixed and random inputs, gets toggled
- 32-bit *delayer* (delay counter) is set to 0
- 128-bit *input_text* is updated with the cipher output if the input was from the fixed dataset
- 1-bit *trigger* bit is also set to low

In Wrapper2 module, the following registers change during the trigger:

- 1-bit *block* toggles once and is used to schedule the rules in the Cipher Process 1 in Figure 9
- 1-bit *trigger* bit is set to low

Practically, we can expect as many registers as in Wrapper1 to get updated in each cipher process but for TVLA we need to reduce the role played by these dynamic changes in the net power consumption to accurately evaluate the accelerator. Dynamic power consumption due to register changes is significantly higher than power consumption due to transistors [4]. Though practically we don't see much change in the power plots between the two wrappers, the effect on TVLA is very significant. Hence, only Wrapper2 was used for Threshold implementation and the result we have got is that it performs better than Composite Field implementation. So overall, Threshold implementation is the most secure among all three, followed by Composite Field implementation.

Table 16: Comparison of all three implementations using Wrapper2 and with 50K traces

| | LUT Based Implementation | Composite Field Implementation | Threshold Implementation |
|---|---|---|---|
| Encryption | 206.301 | 16.694 | 2.737 |
| Decryption | 38.018 | 29.331 | 12.071 |

We also conclude that the TVLA score increases with the number of traces with the following Log-log plots in Figure 17. Threshold AES will likely show more linearity at higher number of traces but we can observe the general increase of maximum t-value with the number of traces. With the values from the tables we can approximately show that $t - value \propto \sqrt{N}$.

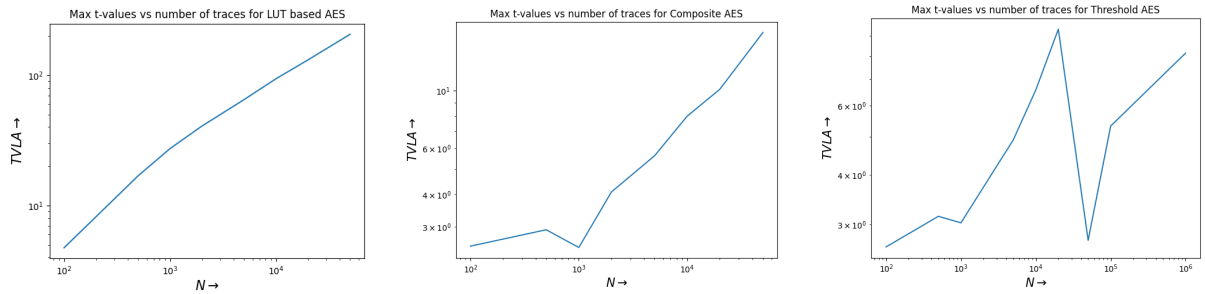



Figure 17: Log-log plots for the three implementations

# 7 Scope of improvement

1. **Remote setup for the oscilloscope for viewing the StartDSO application:** Instruments operation and trace data collection can be done through remote operation using TCP-IP, ActiveDSO, GP-IB, etc.

2. **Effect of routing algorithms used:** We have used only default synthesis and implementation for our analysis. The effect could be significant if, say, Speed Maximized synthesis option is used which has higher power consumption [2].

3. **Analysis of the dependence of TVLA on the frequency of the FPGA board:** Very low frequency was used because of various limitations as stated previously. Higher frequency implies higher power consumption and this could have its own implications.

4. **Practical assessment of accelerators by collecting traces when it is integrated with a CPU:** The Typical AES accelerator has been integrated with Shakti C-Class and system functions have been provided to allow setting of the trigger pin. The problem with using a CPU is that it will take more time to synthesise and implement as compared to the wrapper modules and modifications to the CPU might be necessary to limit the total number of LUT usage to less than 110,000 LUTs (total number of LUTs in the SASEBO-GIII platform).

5. **To improve threshold implementation:** Specific operations which are causing the leakage can be identified by detailed analysis of the TVLA plots and can be used to improve threshold implementation.

# 8 Appendix

Limitations of this study are as follows:

- TVLA value was not very steady and it is possible that the TVLA score could change significantly when a test is run multiple times: A large number of tests might be required to get a consistent value. The physical conditions of the setup could also play a role in the trace collection.
- Frequency of the FPGA board had to be set to minimum due to the limit on the maximum sampling rate: Large number of samples had to be collected for a small process which takes only 12 cycles. This causes some traces to be skipped in the beginning/end, and the process was repeated if sufficient number of traces were not recorded.
- The limited RAM capacity of the oscilloscope caused intermittent hanging of the set up, causing repetition of the validity of collected data and loss of time.

# References


[1] Aditya Pradeep, Vishal Mohanty, Adarsh Muthuveeru Subramaniam, and Chester Rebeiro (2019), 'Revisiting AES S-Box Composite Field Implementations for FPGAs', IEEE Embedded System Letters, Vol.11, No.3.

[2] Akhil Sai and Chester Rebeiro (2017), 'Influence of CAD Routing Algorithms on Cryptographic Side Channel Attacks', Department of Computer Science and Engineering, IIT Madras.

[3] Gilbert Goodwill, Benjamin Jun, Josh Jaffe and Pankaj Rohatgi, 'A testing methodology for side-channel resistance validation', Cryptography Research Inc.

[4] Stefan Mangard, Elisabeth Oswald and Thomas Popp, 'Power Analysis Attacks: Revealing the Secrets of Smart Cards', Textbook.

[5] Vishal Mohanty and Chester Reberio (2019), 'Efficient AES Threshold Implementation on FPGA', UGRC-II report, Department of Computer Science and Engineering, IIT Madras.