T-Scope: Side-channel Leakage Assessment with a Hardware-accelerated Online TVLA Test

A Major Qualifying Project (MQP) Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements
for the Degree of Bachelor of Science in

Electrical and Computer Engineering

By:

Hao Wang
Andrew Malnicof

Project Advisors:

Patrick Schaumont

Date: April 2024

Abstract

Test vector leakage assessment (TVLA) is a generic and commonly-used approach to assessing a device's side-channel vulnerability based on measuring a large number power traces. However, the current use model of TVLA uses batch processing which separates trace acquisition from statistical analysis, making the method harder to use in online, continuous-monitoring scenarios. We propose T-scope, a TVLA optimized for online testing of real-time targets. We use a pipelined solution to efficiently store power traces in compact histogram structures, and then use FPGA-based hardware acceleration of Welch's t-test to compute the TVLA. By continuously updating the histograms with newly acquired power traces, T-scope visualizes real-time changes in the side-channel leakage characteristics of the target. Our FPGA-based hardware-accelerator, created using high-level synthesis, offers a 99.93X performance gain over a software-based solution. We present a hardware demonstrator of a real-time display of the TVLA assessment of a leaky implementation of the Advanced Encryption Standard.

Acknowledgements

We would like to thank Prof. Patrick Schaumont for his excellent guidance and generous contributions to this project. This project would have not been successful without his mentorship.

Authorship

Paper	All
Preliminary Experimentation	Andrew Malnicof
Histogram Library Implementation	Hao Wang
Histogram Scaling Implementation	Hao Wang
T-Test Core Implementation	Andrew Malnicof
T-Test Core Testing	Andrew Malnicof
T-Test Hardware Integration	Andrew Malnicof
T-Test Core Optimization	Hao Wang
T-Scope Application	Hao Wang
Software Integration and Build System	Hao Wang
Performance Testing	All

Contents

List of Figures

List of Tables

1 Introduction

The rapid proliferation of cheap IoT devices has cast a spotlight on a critical concern in hardware security. The development of these devices typically doesn't include comprehensive security validation, which could potentially result in leaking secrets via side channels during cryptographic functions [1]. Test vector leakage assessment (TVLA) provides a technique to assess whether a device is potentially susceptible to attacks such as Simple Power Analysis and Differential Power Analysis [2]. Unfortunately, current TVLA campaigns are formulated as batch processes that separate trace acquisition from leakage assessment analysis. This makes TVLA difficult to apply in scenarios that require continuous monitoring of online targets.

We present a novel approach (T-Scope) to TVLA campaigns, illustrated in Fig. 1, which pipelines the process and allows for continuous evaluation. We adapt current TVLA methodologies towards an online campaign, in which the number of traces is potentially infinite. The TVLA statistic is continuously updated, and we propose an efficient implementation of this process by using hardware acceleration. This online computation results in a trace of t-values that reflects the leakage characteristics of a target. This t-trace can be rendered as an oscilloscope trace, leading to T-scope.

1.1 Significance and Novelty

Traditionally, side-channel leakage assessment is applied during prototyping or during testing. For example, a security testing lab may evaluate the side-channel leakage of a hardware chip using TVLA as one of the steps that support security certification. However, side-channel leakage assessment is also useful outside of the domain of security test and certification. TVLA has the unique ability to characterize information leakage in an application-independent manner, and therefore it can be used in a broad range of scenarios

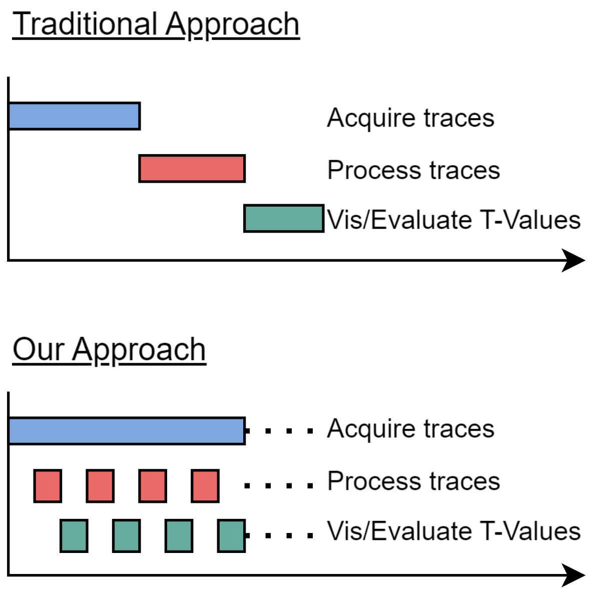

Figure 1: Traditional approaches to TVLA compared to our approach.

related to side-channel leakage. A first scenario is the online testing of new firmware on a deployed IoT device for rapid device prototyping. A second scenario is the detection of (potentially malicious) modifications to hardware and firmware in the field by recording the traces from a target responding to a known set of test inputs. This approach enables a self-monitoring system for system integrity.

Our proposed T-Scope is, to our knowledge, the first implementation of TVLA for this online, real-time scenario. Reference [3] proposes the online calculation of mean and variance for TVLA. However, it still operates in a batch based approach. Our methodology introduces a real-time aspect to online TVLA. We improve upon a known histogram-based assessment technique [4] by supporting a continuous stream of traces. We also provide an efficient, hardware-accelerated implementation that combines high-level synthesis with hardware-software co-design in an end-to-end demonstration.

2 Background

TVLA uses statistical analysis to provide a level of confidence to conclude if a device-under-test (DUT) has a data-dependent power consumption. This generic test can quickly and easily report if a DUT fails to provide proper side-channel security, independent of DUT architecture. However, TVLA reveals no information on the difficulty of an attack to exploit the leakage. Nevertheless, TVLA is widely used in rapid DUT testing applications to ensure device security [5].

Fig. 2 shows an overview of TVLA. Suppose an algorithm encrypts plain-texts based on a hidden DUT. Side-channel power consumption measurements on the DUT can be performed while the algorithm executes. A trace encapsulates the power measurements collected over a single algorithm iteration. Many traces can be collected while varying the plain-text input of the DUT and assembled into groups of traces. In a *non-specific* TVLA test, two groups of traces are gathered in a process known as fixed versus random testing: one group where the plaintext remains constant and the other group where the plaintext is randomized. Using t-tests, the groups of traces can be compared to determine when the two sets of traces are statistically different from one another. In TVLA, t-tests are performed on a time-sample basis, meaning that power-consumption data at same time are compared across groups of traces.

Let P_0 and P_1 indicate the power consumption measurements for each group at the same instant in the traces. Let μ_0 and μ_1 indicate the sample mean, s_0^2 and s_1^2 indicate the sample variances, and n_0 and n_1 indicate the cardinality of the sets. The t-test statistic is computed as follows [5].

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}} \tag{1}$$

Therefore, the number of t-tests computed corresponds to the number of samples in

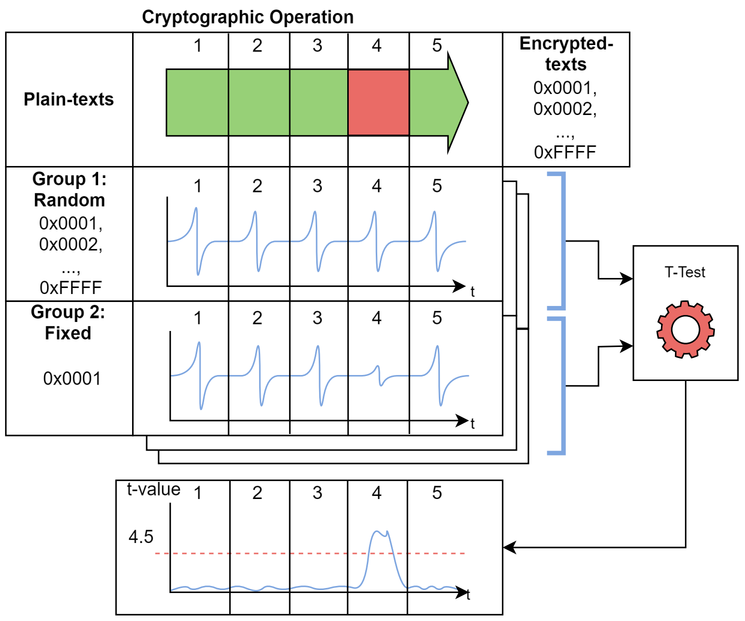

Figure 2: An overview of TVLA.

a trace. The t-test statistic calculated by (1) can be compared to a threshold of $|t| > 4.5$ to reject the null hypothesis, concluding significant difference. This t-statistic value corresponds to a confidence of greater than 0.99999 that the distributions are significantly different.

2.1 Basic TVLA Method

A basic method to perform TVLA involves gathering all the raw traces in storage, then using a two-pass algorithm to calculate the two t-test parameters, mean and variance, using all the available traces. This method may require gigabytes of storage for long TVLA campaigns, upwards of tens of thousands of traces, following (2).

$$storage = K * N * Q \tag{2}$$

Where *storage* represents the storage in bits, for K traces, N trace length, and Q bits per power-consumption measurement.

After collecting a batch of traces, the basic TVLA method requires $2N$ means and variances to compute N t-test statistics. These statistics must be recalculated upon the addition of more trace data.

2.2 Improved TVLA Methods

Different methodologies for TVLA have been proposed to reduce the TVLA's computation time and required storage. Schneider *et al.* propose a solution to address the efficiency of computations in traditional TVLA by using an incremental algorithm to update the t-test parameters [5]. In their method, traces can be incrementally collected, used to update the stored means and variances, then discarded. Such a characteristic allows for the use of TVLA on large sets of data, since trace data need not be stored, only the intermediate t-test parameters. Using incremental algorithms increases the computational efficiency and reduces the required storage compared to the basic TVLA method. Although more efficient, the algorithm may be more computationally intensive than the basic TVLA. The incremental method updates means and variances for each trace it processes compared to only computing the parameters once for a batch of traces. Thus, the speed at which the traces are gathered may be reduced since the system may be computationally bottle-necked from updating the means and variances.

Reparaz *et al.* propose a histogram paradigm to store traces [4]. The method promises orders of magnitude increase in performance compared to the traditional TVLA approach because the histograms enable a very efficient t-test computation. The histograms represent the distribution of power over the traces. Each histogram bin corresponds to the relative occurrence of a specific power level over the trace point. The entire set of traces is thereby reduced to a single trace of histograms. The histograms are created efficiently online, during the trace collection, by implementing each histogram bin as a counter that increments according to the number of times a specific power level is observed.

Equation (3) represents the memory required to store the traces in histograms for a single family of histograms.

$$storage = N * 2^Q * B \tag{3}$$

Where B represents a counter size of B bits and 2^Q represents the number of bins in each histogram based on the number of bits in each power-consumption measurement. Once the constants, N trace length, 2^Q bins, and B counter size have been determined, the required storage of histograms remains constant as it is independent of the number of traces being stored. While the memory required to store raw traces increases linearly according to (2), the memory required for two histogram families becomes more efficient for storage of more than $2^{Q+1}B/Q$ traces, with a maximum of 2^{B+1} traces able to be stored without risk of counter overflow.

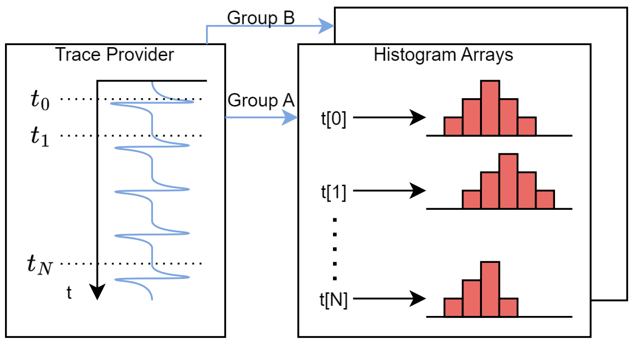

Figure 3: TVLA histogram storage overview.

Fig. 3 demonstrates the histogram method. For a TVLA test, two histograms are maintained per trace point, one for each data set. From the histogram pairs, the means and variances, and t-test statistics can be calculated using (1). It requires a low computational complexity to ingest and store traces and more efficient storage then storing raw traces. On the other hand, the mean, variance, and t-test static must be recalculated as more traces are gathered.

Neither the incremental method [5] nor the histogram method [4] are ideally suited for online TVLA computation. The incremental method copes with significant storage overhead, while the histogram method is not incremental. Therefore, we propose a method that combines both ideas into an online TVLA test. Our aim is the provide a TVLA test that dynamically updates as more traces become available. We will also allow aim at a continuous method; our methodology gives the more recently captured traces a higher weight when calculating the t-test values.

3 Methodology

To enable online TVLA campaigns, we propose a pipelined approach to TVLA which includes a dynamic-scaling histogram storage paradigm in addition to a hardware accelerator to increase the throughput of t-test trace computation.

3.1 Preliminary Experimentation

Experimentation was performed to refine the methodology to our novel approach. The experimentation helped to make design decisions in our proposed solution.

A simulation was constructed in order to determine the number of bits in each power-consumption measurement, Q. The intention of performing the simulation was to determine the minimum number of bits for each measurement while preserving accuracy in the calculated t-test statistic. For each iteration of the simulation, two datasets of normally distributed data with random mean and standard deviation within a certain range were generated. The t-test statistic, the real t-test statistic, was first calculated from the datasets without an reduction in precision, meaning the floating point values generated by the program were used. Then, the t-test statistics for $Q = 4$ to $Q = 12$ bit power-consumption measurements was calculated by scaling down the simulated datasets by the corresponding number of bits. The difference between the real t-test and the scaled histogram t-test statics was calculated for 100 iterations. Fig. 4 shows the results of the simulation. The error between the t-test statistic calculated from histograms with $Q = 8$ bits and the real t-statistic is less than 0.00001, with diminishing returns for the greater number of bits.

To determine the adequacy of a software solution for histogram binning to ensure the maximum frequency of trace collection, experimentation was performed. Using a ChipWhisperer Lite (CW), the trace collection tool in the pipeline, the maximum throughput of traces received was found to be 2 Mbps. This was found by constructing a scenario such

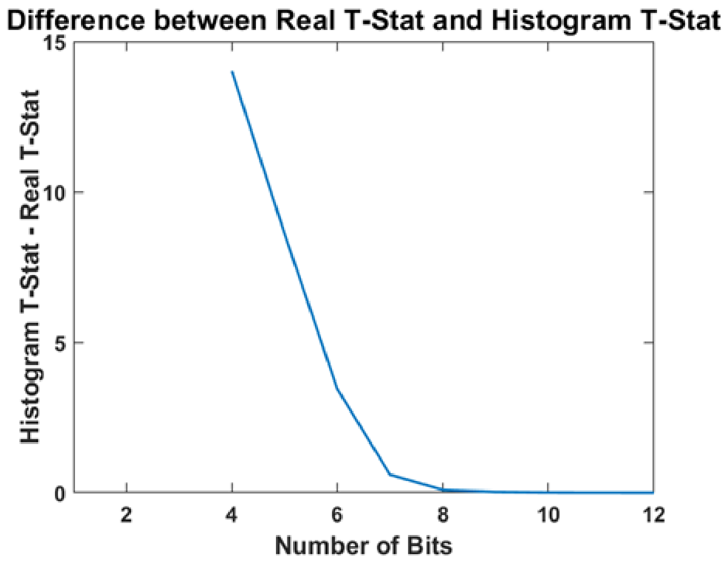

Figure 4: Power-consumption measurement scaling simulation.

that the CW collects traces as fast as possible. The CW's ADC samples at a rate of 105 MS/s and collects samples upon a rising edge trigger. A noise signal consisting of a rising edge trigger every other time sample was supplied to the CW to trigger collection. Then, a trace of length 3000 samples was collected and sent back to the host. Additionally, no serial control other then arming the CW was supplied, so likely the CW throughput during T-Scope operation becomes less than 2 Mbps.

To ensure the storing of traces into histograms does not limit the trace collection throughput, the throughput of the histogram binning program was measured. A scenario was constructed where 32 traces of 4096 samples were constructed then ingested into histograms in software. The 4096 histograms consisted of 256 bins with 32 bit counters. The procedure was performed for 4096 iterations and the average throughput was calculated to be 180 Mbps when run on the Pynq's SoC. Thus, a software solution for trace storage is sufficient in order to store traces collected from the CW. The limitation in the frequency of capturing and ingesting traces into T-Scope is determined by the ChipWhisperer throughput, not the T-Scope implementation.

Our choices in hardware were specific to implement our proposed methodology.

The ChipWhisperer Lite measurement board was used as it was a preexisting and relatively inexpensive solution to side-channel analysis. The board facilitates easy communication and power-consumption measurement of its accompanying target board using Python. The Pynq-Z1 FPGA board was chosen to interface with the ChipWhisperer Lite. The SOC on the Pynq board can use the ChipWhisperer Scope and Target Python APIs to communicate. Additionally, the t-test hardware accelerator can be programmed on the board's FPGA. The provided Pynq API allows for easy communication between the SOC and FPGA.

3.2 Real-Time Scope

We integrated the histogram storage and t-test core into a real-time scope application, leveraging a networked FPGA board, oscilloscope, and target board configured for AES encryption. Fig. 5 shows the layout and interconnections of the real-time scope.

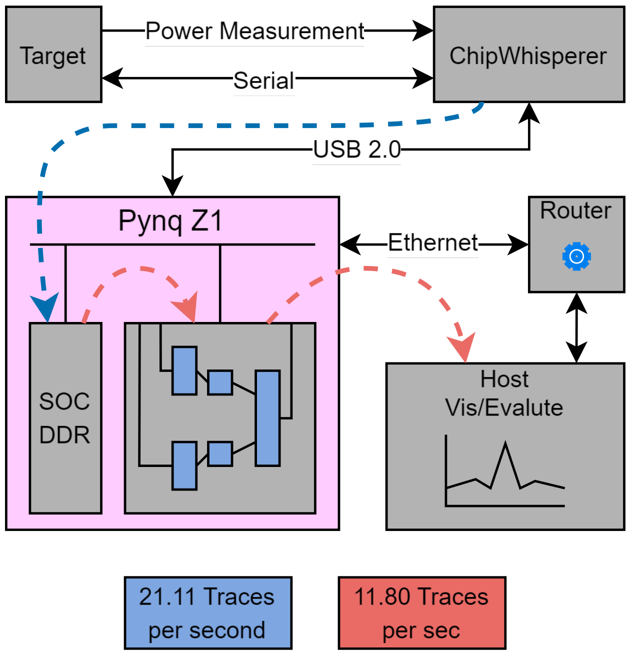

Figure 5: System structure of the real-time scope and throughput metrics between components.

The scope application comprises of two programs, one executed on the SOC of the FPGA board and the other on the host computer. The program running on the FPGA board encompasses two processes. Firstly, the ingest process acquires new traces from the oscilloscope and adds them into the histogram arrays. Additionally, the ingest process periodically invokes the scaling function on the histogram array, facilitating a sliding window like effect. Concurrently, the compute process continuously triggers the t-test core to compute the next t-value trace. Using high-performance ports, DMAs stream values between the t-test core and memory, to reduce SOC load. Upon completion, it transmits the t-values over the network to the host via a UDP stream. The transmission of the t-value trace is chunked into several datagrams. Each datagram consists of a header containing the offset into the t-value trace and how many samples follow. Then, the t-values are appended to the end. The two processes and the t-test core operate on the same shared memory array to avoid the need for message passing. The program executed on the host only listens for new packets and displays them in real-time on a plot.

Creating a SOC based pipeline to process traces rather then processing them on a host computer allows T-Scope to operate as a separate scope. It acts as an oscilloscope for TVLA, allowing changes to the DUT or cryptographic implementation to be monitored in real time. Additionally, using a separate pipeline enables the online aspect of T-Scope.

3.3 Histogram Storage

The histogram-based trace storage paradigm operates on a 2D array with memory usage corresponding to (3) [4]. Our choice in using the histogram method lies in its storage efficiency and simple memory organization. Histograms are efficient in streaming consecutive data into our hardware accelerator, as calculations are performed on values for a single time sample, instead of whole traces. If Q is smaller than the bit resolution of the trace samples, Q_T, scaling is done. Our scaling operation utilizes rounding to reduce truncation noise, and is modeled by (4).

$$Scaled = floor(\frac{sample + 2^{Q-Q_T-1}}{Q - Q_T}) \tag{4}$$

The procedure to ingest a trace is shown in algorithm 1. In the case where scaling is not needed, the operation becomes a series of read-increment-writes.

Algorithm 1 Procedure to ingest a power trace for a single histogram array.

```
$i \leftarrow 0$
while $i < N$ do
    $BinI \leftarrow \text{ScaleIfNeeded}(trace[i])$
    $H[\text{i}][BinI] \leftarrow H[\text{i}][BinI] + 1$
    $i \leftarrow i + 1$
end while
```

To adapt the histogram storage paradigm for online TVLA and for the t-test results to evolve over time, we need to remove the contribution of old traces. To produce a smooth windowing effect, we scale all bins in all histograms by $\frac{1}{2}$ by an interval of every S_I traces. In effect, the contribution of old traces decreases exponentially. Fig. 6 visually shows how old traces are phased out.

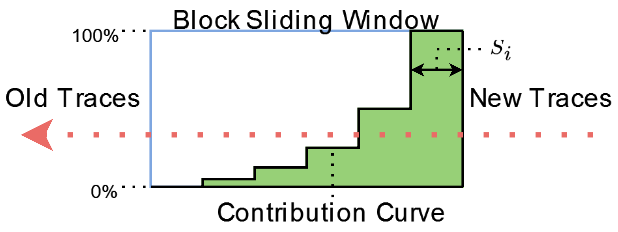

Figure 6: Visualization of how scaling the histogram bins arises to a block sliding window effect.

The procedure to scale the histograms bins is shown in algorithm 2. The operation becomes a series of read-shift-writes.

3.4 T-Test Core

To calculate the t-test traces from the histogram arrays, we implemented a hardware accelerator. Our core uses the standard t-test equation shown in (1). The means and

Algorithm 2 Procedure to scale a single histogram array.

```
$i \leftarrow 0$
while $i < N$ do
    $j \leftarrow 0$
    while $j < 2^Q$ do
        $H[\mathrm{i}][j] \leftarrow H[\mathrm{i}][j] >> 1$
        $j \leftarrow j + 1$
    end while
    $i \leftarrow i + 1$
end while
```

variances of the histogram arrays are computed using equations (5) and (6).

$$\mu[t] = \frac{\sum_{i=0}^{2^Q-1} i * h[t][i]}{n[t]} \tag{5}$$

$$\sigma[t] = \frac{\sum_{i=0}^{2^Q-1} (h[t][i] - \mu[t])^2}{(n[t] - 1)} \tag{6}$$

We split the computation of the t-values into several processes connected by first-in-first-out (FIFO) streams to optimize for resource usage and throughput, allowing for the pipelining of the process. We implemented the core assuming $Q = 8$ and $B = 32$. First the core simultaneously streams the two histogram arrays as pair-wise bins by packing two 32 bit values into a 64 bit packet. This allows the full use of the bandwidth of a 64 bit high performance port. It then computes the count (number of elements) and sum of each histogram, and stores the bins in to a parallel-in-parallel-out (PIPO) buffer. The histogram statistics are then used to compute the mean and inverses of counts. The mean and bin is then used to partially calculate the variance as follows.

$$VarSum[t] = \sum_{i=0}^{2^{Q-1}} (h[t][i] - \mu[t])^2 \tag{7}$$

Lastly, these intermediate products are used to compute the t-values which are then streamed out as a double-precision floating point.

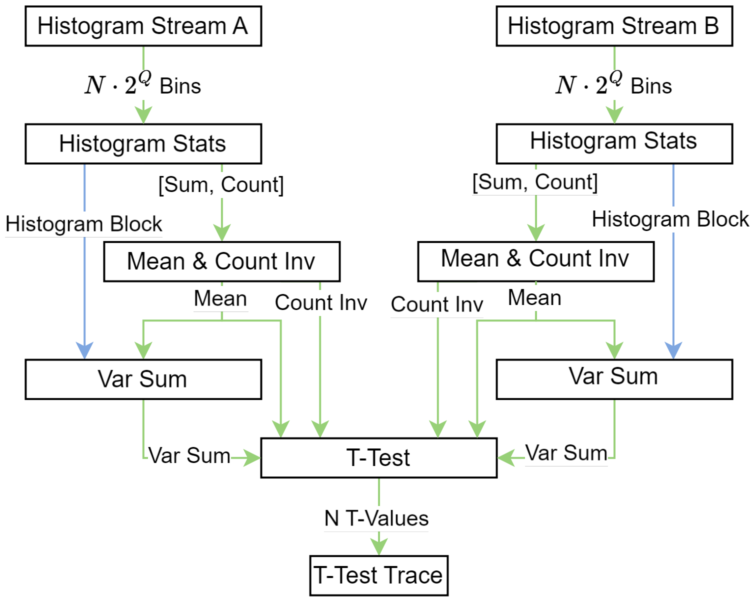

Figure 7: Architecture of the t-test core.

The architecture of the core is shown in Fig. 7. Due to hardware resource constraints floating point arithmetic needed to be avoided, in favor of using fixed point arithmetic. To avoid overflow within the core, we computed the maximum values and integer bit sizes for the intermediate products based on the bit depth of the histograms, Q, and the bit sizes of the bins, B. Table 1 shows the bit sizes of the intermediate products and their justification. These variables directly correspond to the FIFO streams in Fig. 7. Based on RTL simulation, we determined it was sufficient to use 24 fractional bits for the mean and variance, and 9 fractional bits for the variance sum to achieve 0.01 accuracy in the resulting t-values while not excessively increasing resource usage.

Multiple division operations are necessary throughout the core. To optimize for resource usage, we computed the inverse and then multiplied, rather than directly dividing. This change allowed us to use DSP cores for multiplication instead of LUTs.

We connected the core to the FPGA's SOC using 3 DMA controllers. This allows the core to operate independently from the SOC once triggered, and simplifies the core interface as it can then read and write to memory by a streaming interface instead of a full

Table 1: Variable sizes for intermediate products including justifications given the assumptions $Q = 8$ and $B = 32$.

Variable	**Bits (integer, fractional)**	**Justification**
Count	40,0	$log_2(2^Q 2^B) = 39.9$
Sum	47,0	$log_2(\sum_{i=0}^{2Q} i * 2^B) = 46.9$
Mean	8,24	Largest column is $2^Q - 1$
Count Inv	1,40	$log_2(\frac{1}{2^Q 2^B}) = -39.9$
Var	15,24	$log_2((\frac{2^Q}{2})^2) + 1 = 15$
Var Sum	55,9	$log_2((\frac{2^Q}{2})^2 * 2^Q 2^B) + 1 = 54.9$

AXI interface.

It is possible to further optimize for resource usage if desired by placing a bound on the count of the histograms. If the histogram arrays are scaled with a interval of s_i, and only one bin was incremented to model the worst case, the growth of the bin after scaling can be modeled using (8).

$$f(n) = \begin{cases} \frac{f(n-1)+s_i}{2}, & n > 1 \\ \frac{s_i}{2}, & n = 0 \end{cases} \tag{8}$$

Solving the recursive function we obtain (9).

$$f(n) = s_i(1 - 2^{-n}) \tag{9}$$

Then talking the limit, and adding the s_i to get the absolute max count of the bin, we obtain (10).

$$\lim_{x \to \infty} s_i(1 - 2^{-n}) + s_i = 2s_i \tag{10}$$

If s_i is small, the variable integer bit sizes can be significantly reduced, leading to a reduction in resource usage.

The core produces N t-values for a pair of $N2^Q$ histogram array. Given a trace acquisition frequency of f_t, and that the core is run every time a new trace is acquired, the minimum throughput of the core should be f_tN traces per second, and ingest the histogram arrays at $2f_tN2^QB$ bits per second.

4 Proposed Solution

We utilized a Pynq-Z1 FPGA board, ChipWhisperer Lite, and a CW303 XMEGA target board for the real-time scope hardware in our study. To demonstrate the system's functionality, we transmitted alternating sequences of random text and biased rounds to the target board for encryption. A biased round induces a state within the AES implementation wherein, during one of the rounds, many bits are set to 0, resulting in a noticeably low Hamming weight.

The t-test core was implemented using Vitis HLS and C++, and was loaded onto the FPGA fabric. Using Vitis HLS allowed us to rapidly prototype the core given it's complex operation. We targeted a clock rate of 10 ns and an initiation interval of 1 for each process. This results in a pipelined dataflow which is able to ingest a 64 bit input packet every 10 ns. Additionally, the updating and scaling of the histogram arrays was done using a C++ library run on the PYNQ's SoC.

The ingest and compute processes were configured to execute as fast as possible to maintain real-time monitoring and analysis. The system parameters were set according to Table 2.

Table 2: Chosen parameters for the real-time scope application.

Parameters	**Value**
N	8500 Samples
Q	8 Bits
Q_T	10 Bits
B	32 Bits
S_i	30 Traces

We selected $N = 8500$ samples to ensure that all rounds of the AES implementation could be captured effectively. A bit-width of $B = 32$ was opted for, ensuring that we were not limited by storage constraints for the histogram arrays, given that most TVLA campaigns do not exceed 2^{32} traces. For storing trace samples and computational purposes, we chose $Q = 8$ bits, which, based on the simulation in Section 3.1, provided sufficient resolution. Additionally, $Q_T = 10$ bits were set to align with the ADC of ChipWhisperer. Lastly, $S_i = 30$ traces were chosen to maintain manageable peak sizes in the real-time graph and ensure responsiveness to changes in the targeted biased round.

To allow for extensibility, a framework was created and was used for applications like the real-time scope. The framework, called pyTVLA, was implemented in python and contains the following sub-frameworks, data source, scheduler, memory, and engine. The data source provides the raw traces, which can be random data or from the ChipWhisperer. The chipwhisperer data source relies on the scheduler sub-framework, which provides the plain-text to send to the target board. The memory defines memory managers which are closely tied to the engine. Memory managers allocate and de-allocate shared memory used by all processes. Additionally the memory sub-framework defines a python wrapper for the histogram library. The wrapper and histogram library both operate on the shared memory from the memory manager. The engine defines classes that calculate the t-values. These classes are expected to ingest the histogram structures and write the t-values to shared memory. In addition to the Pynq based engine, a software engine based on NumPy was implemented for performance comparison. The source code is linked in appendix A.

5 Results

To demonstrate the performance of the proposed implementation, we implemented the Real-Time Scope, T-Scope, described in Section 3.2 with parameters from Section 3.4. Fig. 8 depicts the T-Scope demonstration setup, and Fig. 9 shows a snapshot of the real-time plot where one of the rounds is currently being targeted.

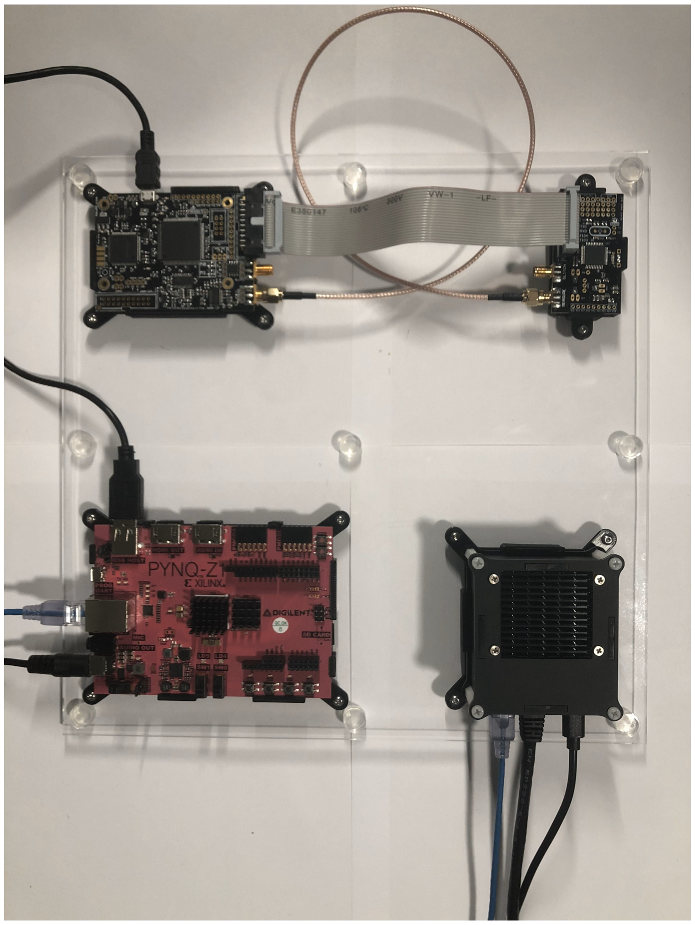

Figure 8: Real-time scope setup. Top Left: ChipWhisperer Lite Capture Board, to right: CW303 XMEGA target board, bottom left: Pynq-Z1 FPGA board, to right: Network Router.

Table 3 details the resource utilization of the t-test core when implemented on the Pynq-Z1. When RTL simulation was run with $N = 32$ samples, it showed an estimated runtime of 46060 ns or 1439.3 ns per pair of histograms.

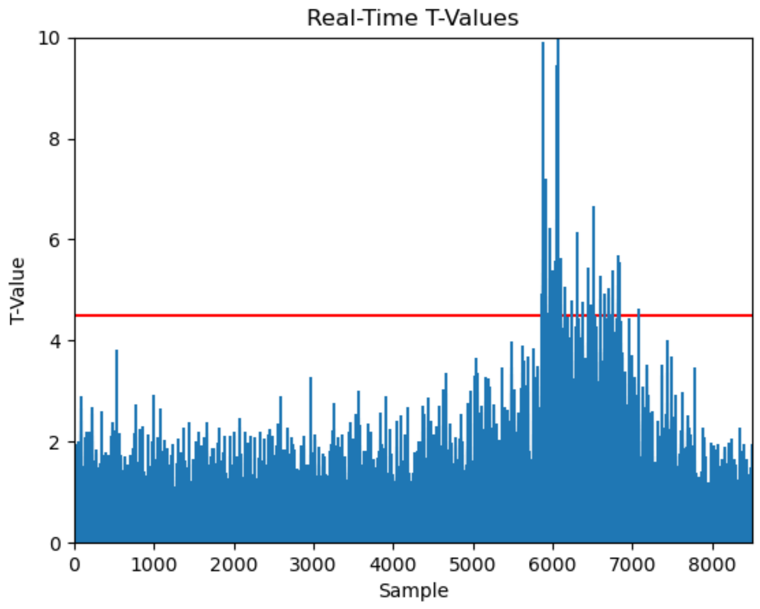

Figure 9: A snapshot of the real-time plot.

Table 3: T-Test Core resource utilization.

T-Test Core	**Raw**	**Percentage**
LUT	19163	36%
FF	33029	31%
DSP	17	35%
BRAM	16	5%

Additionally, we tested the data throughput of the proposed implementation experimentally. Table 4 describes the average frequency of the capture and ingest process and the calculation and upload process. It also compares the throughput of the tradition software solution and our hardware accelerated solution when only calculating the t-value trace. Overall, our hardware accelerated core shows a speed up of 99.93X.

Table 4: Experimental throughput.

	Software	**Core**
Capture & Ingest	24.50Hz	21.11Hz
Calc & Upload	0.40Hz	11.80Hz
Calc Throughput	65.37Mbps	6533.63Mbps
Calc Speed up		99.93X

6 Project Mechanics

The completion of this MQP was performed during A, C, and D terms of the 2023-24 academic year. The following timeline describes the structure of the project.

A term: The term was spent analyzing the problem. The original topic of the project was hardware-accelerated TVLA, and was open-ended as to our methodology and implementation. We performed experimentation to determine the feasibility and practicality of such a system. By the end of the term, we created a proposal involving our system architecture, our t-test hardware accelerator, and the histogram storage method.

C term: We refined our methodology and completed the implementation of T-Scope. The histogram library, scaling, t-test core, t-scope application, software integration, and build system were implemented to create the T-Scope system.

D term: The performance of the system was tested and evaluated. The hardware accelerated system was compared to a fully software based solution. We performed scholarly work, presenting a poster with demonstration at New England Hardware Security Day 2024 and submitting a paper to Midwest Symposium on Circuits and Systems (MWSCAS) 2024.

7 Conclusion

We presented our novel approach to online TVLA campaigns (T-Scope). By utilizing the histogram storage paradigm, scaling technique, and a hardware accelerator, we implemented and applied our methodology for real-time TVLA campaigns with an online target. Our methodology paves the way for applying TVLA to new applications that require low latency real-time results and continuous iteration.

References

[1] M. Randolph and W. Diehl, "Power side-channel attack analysis: A review of 20 years of study for the layman," Cryptography, vol. 4, p. 15, 2020.

[2] C. Carper, S. Olguin, J. Brown, C. Charlton and M. Borowczak, "Challenging Assumptions of Normality in AES s-Box Configurations under Side-Channel Analysis," Journal of Cybersecurity and Privacy, vol. 3, no. 4, pp. 844-857, 2023.

[3] S. Bhattacharya, S. Bhasin and D. Mukhopadhyay, "Online Detection and Reactive Countermeasure for Leakage from BPU Using TVLA," 2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID), Pune, India, 2018, pp. 155-160, doi: 10.1109/VLSID.2018.54.

[4] O. Reparaz, B. Gierlichs and I. Verbauwhede, "Fast leakage assessment," Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, pp. 387-399, 2017.

[5] T. Schneider and A. Moradi, "Leakage assessment methodology: A clear roadmap for side-channel evaluations," in Cryptographic Hardware and Embedded Systems–CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings 17, 2015.

Appendices

A Source Code

We provide a GitHub link to the code repository containing the Histogram library and T-Test core, https://github.com/arandomdev/T-Scope. A archive of the repository will also be uploaded to https://digital.wpi.edu/ as a part of the project submission.