Algorithm-Agnostic System for Measuring Susceptibility of Cryptographic Accelerators to Power Side Channel Attacks

by

Brandon John

B.S. Electrical Science and Engineering,
Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author . . . .
Department of Electrical Engineering and Computer Science
May 6, 2022

Certified by . . . .
Mengjia Yan
Assistant Professor
Thesis Supervisor

Certified by . . . .
Brendon Chetwynd
Technical Staff, MIT Lincoln Laboratory
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Algorithm-Agnostic System for Measuring Susceptibility of Cryptographic Accelerators to Power Side Channel Attacks

by

Brandon John

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Many digital devices, from secure enclaves to generic processors, often handle encryption of sensitive data. Protecting this sensitive data is a significant challenge, with potential vulnerabilities extending from bugs in both software and hardware. One major class of vulnerabilities under active research is the use of Power Side Channels (PSCs), which involve precisely measuring the power consumption of a device over time. However, current research is fairly disjoint, without a standardized set of tools for quantifying protection techniques. This leads to the motivation of this project: to create a standardized baseline system for evaluating power side channels and their defenses.

This project makes several contributions to the power side channel community. First, it enables calibration of Signal to Noise Ratio (SNR) measurements to a common baseline, and thus easier comparison between various defense techniques. Second, it proposes a method of measuring SNR that requires a constant number of samples, as compared to some techniques that keep sampling until some reference amount of information is leaked. Third, it includes a case study of AES cores which yields a better understanding of how a PSC amplification technique (specifically using many identical cores in parallel) affects the PSC's signal "strength" and thus time to successfully extract the secret data. Finally, it makes public an ecosystem for quickly starting power side channel research without the significant effort of implementing everything from scratch before any research can begin.

Thesis Supervisor: Mengjia Yan
Title: Assistant Professor

Thesis Supervisor: Brendon Chetwynd
Title: Technical Staff, MIT Lincoln Laboratory

Acknowledgments

I would like to thank my advisors, Mengjia Yan and Brendon Chetwynd, for their support from the beginning to the end of my journey to write this thesis, and their patience as I procrastinated a bit too much. I'd also like to thank Kyle Ingols for going above and beyond in helping me in all sorts of ways, from personally delivering equipment to debugging broken servers.

I would also like to thank my partner, Cici, for supporting me throughout the year and especially during the crunch time at the end, where I became effectively nocturnal. And of course, I need to give a special shout out to my cat, Saturn, who taught me to lock my computer overnight lest I return to thousands of lines of a random letter typed wherever I had left off.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Power side channels (PSCs) are a well-known and widely-studied class of hardware vulnerabilities. PSCs can be used to non-invasively infer the internal state of a piece of hardware by measuring the power consumption over time. At a macro level, one possible PSC would be measuring the power consumption of an entire computer in order to determine when it is in use – periods of low power consumption may indicate inactivity or a sleep mode, while high power consumption indicates that the computer is actively processing data. Similarly, we can measure the power consumption at the chip level to extract information that correlates to individual clock cycles [10]. This thesis focuses on this class of PSC attacks, and specifically looks at PSCs targeting Advanced Encryption Standard (AES) encryption accelerators [11].

Current PSC research is quite varied, and has yielded various attack algorithms for extracting state information from power traces, defense techniques for reducing attack surfaces, and methods of measuring how much and how quickly information can be extracted from a particular PSC. One common testing methodology is the Test Vector Leakage Assessment (TVLA) test, which determines if a side channel has data dependent information that could potentially be exploited [1]. However, this is a binary metric, and thus not particularly useful for comparing PSCs. A similar method is to actually perform an attack using Correlation Power Analysis (CPA) or Differential Power Analysis (DPA), and determine how many traces are required to extract the secret key [2, 9]. However, doing this requires a leakage model, i.e. a way of predicting how much power the algorithm will draw given an arbitrary set of inputs [4]. This is especially problematic if the two implementations have different leakage models, as it is hard to guarantee that both leakage models are equally good. As such, it is generally difficult to rigorously compare the side channel leakage of two different algorithm implementations.

Along with the note of most PSC research being quite varied in breadth, this is also true for methods. While many FPGA PSC projects rely on the Sakura/Sasebo family of hardware, and there are some open source libraries for software PSC research (such as [12]), there is little commonality beyond this. As it is, most research still ends up being 1-off projects, be it a particular ASIC or a fully custom FPGA implementation. This has a few issues, namely that it is often hard to compare results from different researchers, and that there is a huge startup cost for each researcher who wants to investigate PSCs. In this work, we help alleviate this startup cost by introducing SCLAF, the Side Channel Leakage Assessment Framework.

SCLAF is a software and hardware toolchain built to support measuring the PSC leakage of FPGA-based cryptographic accelerator implementations. SCLAF focuses on automation, allowing for large rapid testing of various implementations, and allows for remote access for out-of-office efforts. SCLAF includes everything from the firmware running on the Sakura X's two FPGAs to the software to record power traces and the algorithm to compute the Signal to Noise Ratio (SNR) of the Algorithm Under Test (AUT). Specifically, SCLAF measures SNR as a ratio of variances, as described in Section 2.2 [14]. This definition does not require a leakage model, and as such, is implementation agnostic, allowing for automated direct comparisons of various algorithm implementations.

Along with introducing SCLAF, in this work, we perform a case study of two implementations of AES that differ in their S-Box implementation. The S-Box is a non-linear byte-size substitution table used several times during each round of encryption. The S-Box can be implemented in a few different ways, the first way we investigate is a simple lookup table (LUT) based approach, with all possible substitutions pre-computed. We call the second implementation “composite”, as it uses composite fields to dynamically compute S-Box substitutions. As the composite field variant has more combinational logic changing state during its operation, we expect to find that it leaks slightly more information via power side channels, and thus should have a higher SNR.

In this case study, we also investigate the effects of both core replication and added noise on SNR. For core replication, we instantiate up to 24 of a single core type and have them all operate on the same data in parallel. In doing this, we find that their contributions to the signal component of SNR are additive and outweigh additional noise components, thus showing that replicating cores in parallel increases SNR. When we add noise by instantiating circuitry that draws random amounts of power over time, the measured noise increases and thus SNR decreases.

Finally, this work begins the process of investigating the impacts of realistic use cases, where the AUT is on the same SoC as a processor and other accelerators. Such an arrangement is likely to significantly decrease the SNR of any given algorithm, and this work takes the first steps toward figuring that out. Specifically, we began porting the CEP, an open source RISC-V processor, to the Sakura X as the first step in the process of measuring the impact of a realistic use case.

Chapter 2

Design and Implementation

The following sections cover the design and implementation of SCLAF, our PSC measurement system. We have integrated tooling for automating the capture and analysis of power side channels with an implementation-agnostic side channel leakage metric to create this platform for systematically evaluating the vulnerability of various implementations of a particular algorithm to PSC attacks. We then performed a case study using the SCLAF to analyze the effects of replicated AES cores and added noise, then compared to a single core in a semi-realistic environment.

The overall architecture of SCLAF is described in detail in the following sections. First is Section 2.1, which covers the setup of the toolchain, from hardware to software. Section 2.2 details how we define and measure SNR to make it a useful cross-implementation definition. Finally, Section 2.3 covers how we will run the CEP on a Sakura X.

2.1 Toolchain

The physical setup of SCLAF including the Sakura X and other hardware is detailed in Section 2.1.1. Section 2.1.2 covers the FPGA firmwares, and the command and analysis software is described in Section 2.1.3.

2.1.1 Hardware Setup

SCLAF's physical setup has three major components: a Sakura X acting as the device under test (DUT), an oscilloscope measuring the power consumed by the Sakura X, and a computer coordinating the two. Additionally the oscilloscope and host computer are connected to a shared network storage server. Figures 2-1 and 2-2 include a block diagram and a photo of the setup.

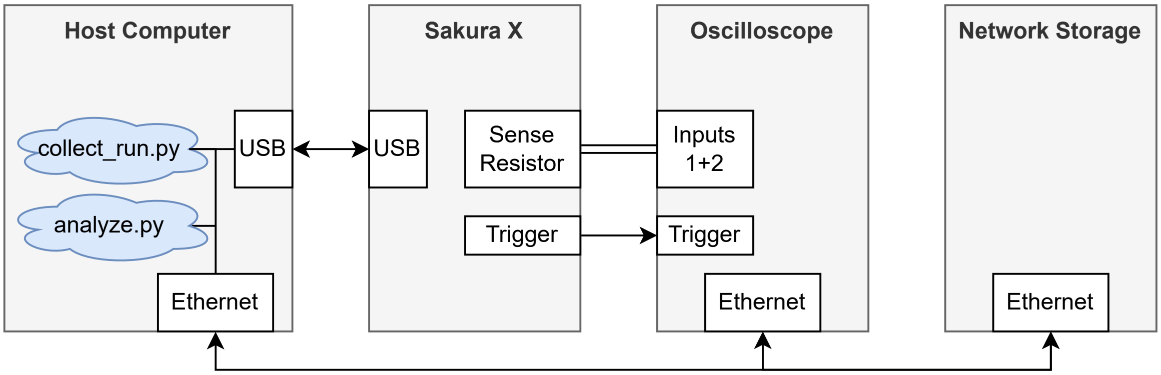


Figure 2-1: Block diagram of hardware setup, including host computer, Sakura X, oscilloscope, and network storage.

The primary component is the Sakura X, "a hardware security evaluation board" that includes a Kintex-7 (K7) series FPGA on an isolated power rail for power side channel attacks, and a Spartan-6 (S6) series fpga for command and control [7]. While we chose the Sakura X as the hardware platform for this research, it should be noted that there are many other viable options available - and we even used a custom setup as described in Section 2.3 for a portion of the case study. The main advantages of the Sakura X include the isolated power supply with integrated current sense resistor and probe points, dual FPGA architecture mentioned above allowing for cleaner data collection, and general commercial availability.

One of the first steps in any PSC attack is to find a way to measure the power being consumed by the DUT. This is easy on the Sakura X, as it provides a pair of SMA ports on either side of a $10\,\mathrm{m}\Omega$ sense resistor on the power rail feeding the V_{CORE} rail of the K7. Both of these SMA ports are then connected to the oscilloscope via a set of length matched SMA->BNC connectors. The oscilloscope is then configured

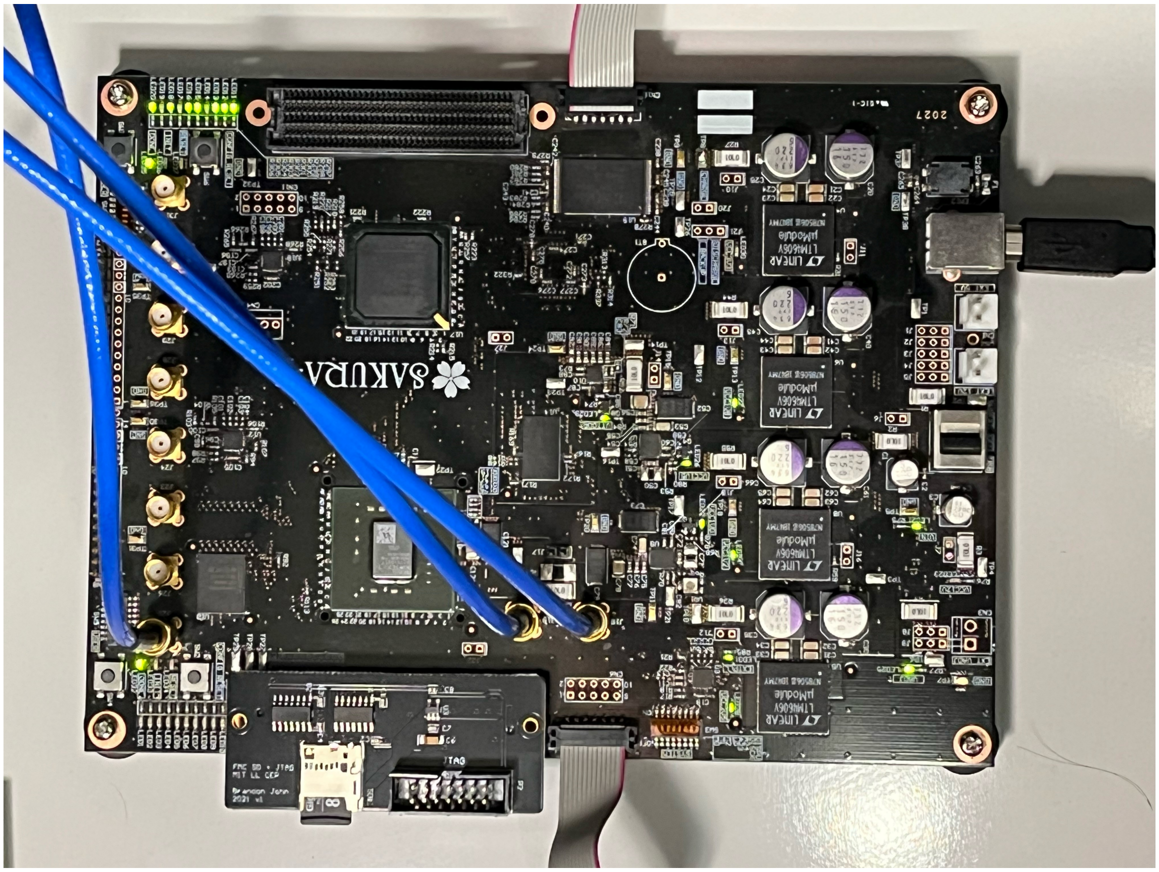


Figure 2-2: Photo of Sakura X inside the shielded isolation chamber including all used connections.

to calculate the difference in voltage between these two ports as the virtual MATH1 channel. This channel is then proportional to the current being consumed by the K7, and can thus be used as the input to any power side channel attack.

For a PSC attack to be possible, the device must be powered. The Sakura X includes a couple of different ways to be powered, namely via the USB port or a discrete connector. In our informal testing, we found no significant difference in the acquired signal based on which of the two power supplies we used, and thus decided to stick with power via the USB port for simplicity.

PSC attacks also require some form of trace synchronization. For the purposes of this research, all power traces were synchronized to the encryption process via the help of an oracle, namely an output of the K7 used as the trigger for the scope. The Sakura X has four such outputs. We connected one of these to the scope with another

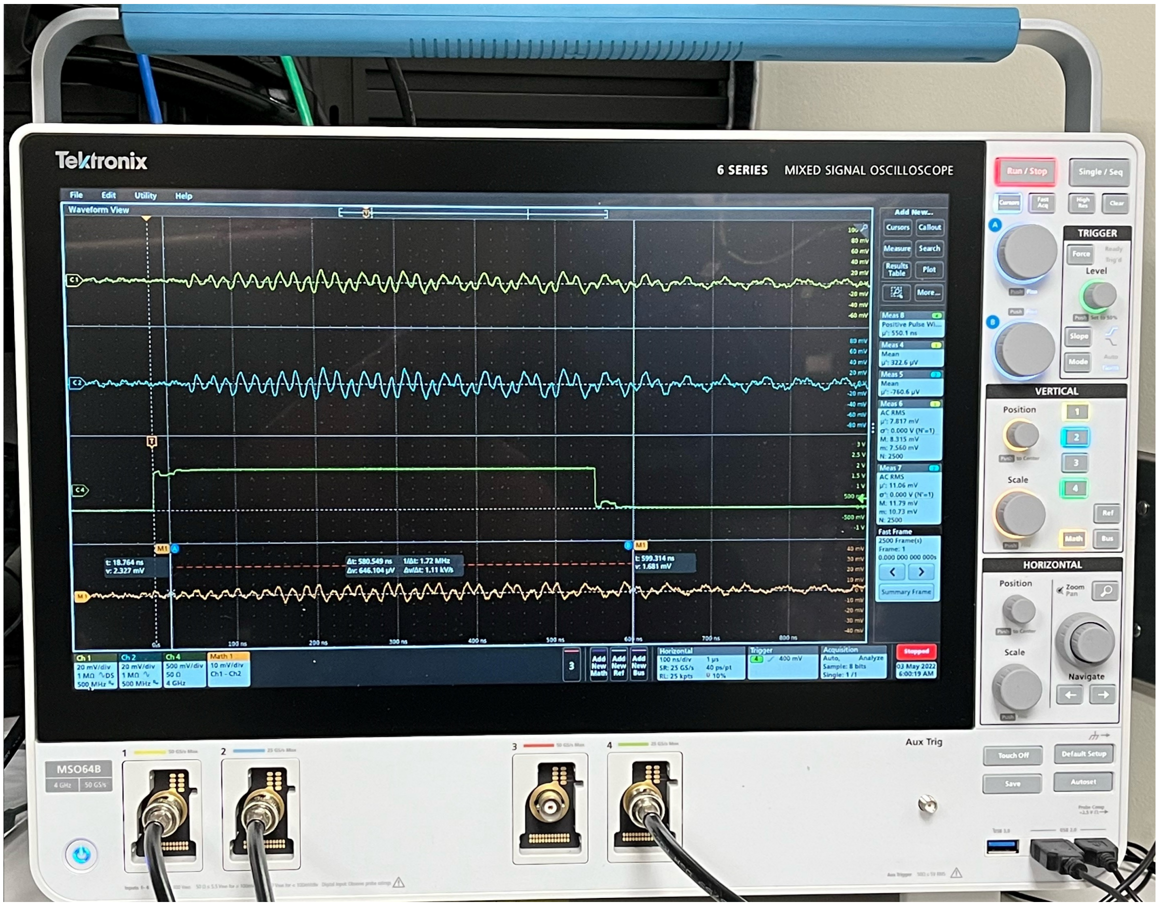


Figure 2-3: Photo of oscilloscope capturing a FastFrame sequence.

SMA->BNC cable.

While these few connections would technically be enough to perform side channel analysis, it would be an extremely manual process. This is one of the places where SCLAF shines - the thorough connectivity and control allow for automated testing and analysis. While these are all basic in a sense, they are also critically important. The Sakura X has two FPGAs, each with their own JTAG port, we hooked up to both with a pair of Xilinx Platform Cable USB II adapters, allowing both FPGAs to be easily reprogrammed. We required a very large quantity of power traces, so we connected the scope to the local network via ethernet to allow for automated data collection. To round out these base connections, the Sakura X is hooked up to the computer via another USB cable directly, for command and control operations as discussed in the next section.

Finally, we found that the recorded data was significantly affected by the ambient EM environment, as is discussed in Section 3.1. To mitigate these effects, the Sakura X was placed inside a shielded isolation chamber, with only a USB hub and the JTAG adapters inside. While the oscilloscope was kept outside of the chamber due to space limitations, the sense wires leaving the chamber were shielded thus reducing the impact of the scope's location. Overall, we found that this arrangement significantly improved the consistency of the results.

2.1.2 FPGA Firmware

The primary goal when designing the SCLAF was to create a platform from which we could effectively and accurately measure SNR, and this had major implications on the design of the firmware for the Sakura X's FPGAs. The information flow in the resulting physical layer of the SCLAF toolchain is summarized in Figure 2-4. In order to accurately measure SNR, the SCLAF must both minimize the amount of computation happening on the K7, and from there minimize the computations that can be temporally correlated to the AUT.

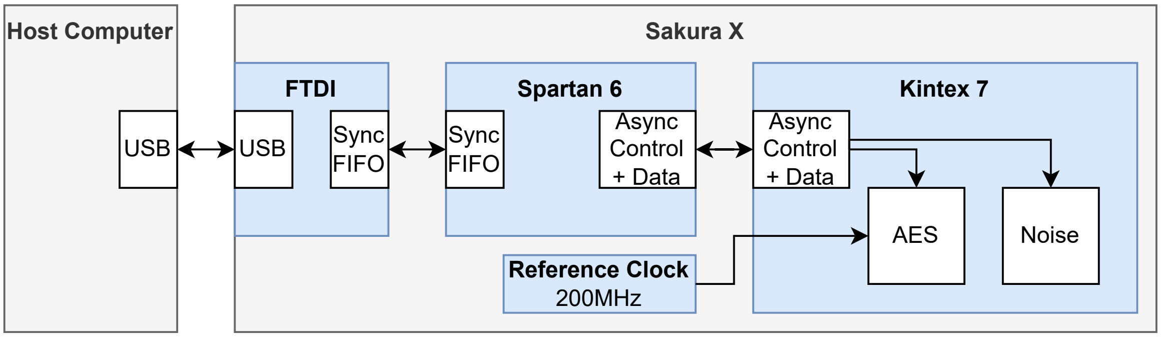


Figure 2-4: Block diagram of data paths within the Sakura X.

In order to minimize computation happening on the K7, the S6 on the Sakura X acts as a translator between the serialized commands coming from the host computer and a set of discrete parallelized asynchronous interfaces to the K7. This removes a significant chunk of processing from the power traces entirely as the S6 has its own power rail. This arrangement has the added benefit of allowing for a standardized control interface, meaning K7 bitstreams can be generated with minimal effort beyond the implementation of the AUT.

To minimize temporally correlated computations within the AUT, the K7 operates with two different clock domains generated with two separate crystal oscillators. The first is dedicated for support tasks such as communicating with S6 or generating noise, and the second domain is exclusively used for running the AUT. By isolating these clock domains, any fluctuations in power draw caused by the control circuitry is guaranteed to be data- and time-independent when referencing the AUT clock domain, and thus will only show up as noise in the SNR measurement.

2.1.2.1 FPGA Firmware: Spartan 6

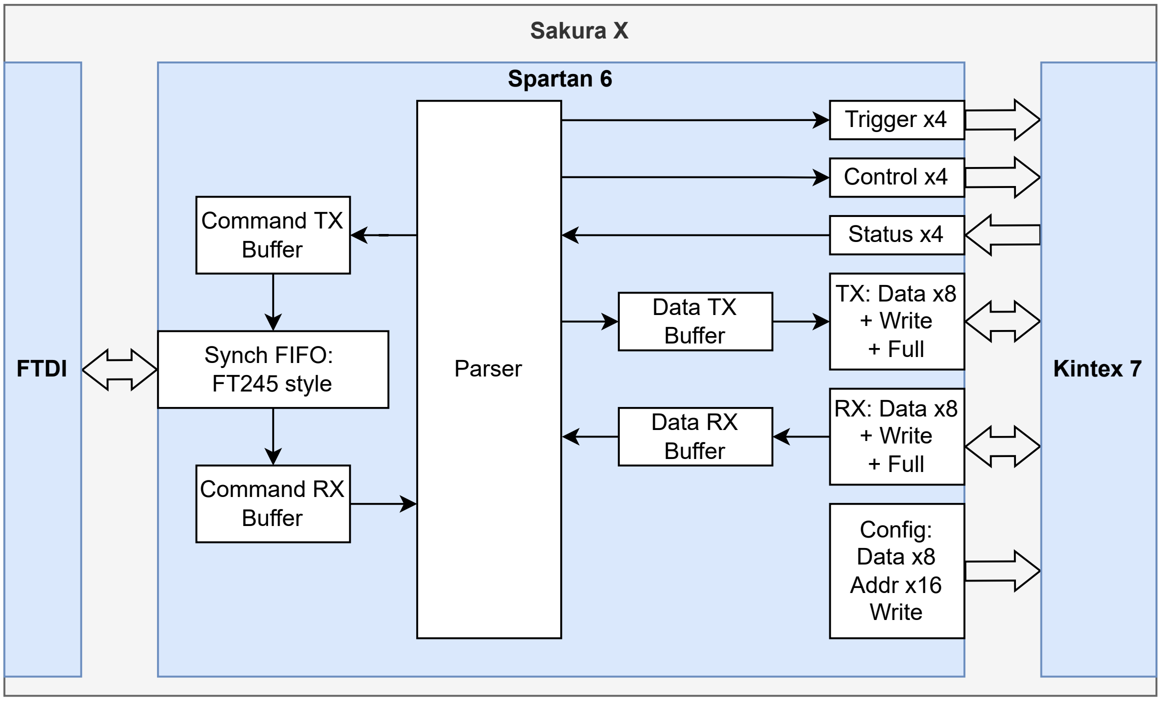


Figure 2-5: Block diagram of data paths within the Spartan 6.

Figure 2-5 focuses on the S6, and how it was designed to minimize peripheral logic within the K7. First, the S6 receives data from the FTDI USB adapter. Specifically, a FT245 style interface connects the FT2232H built into the Sakura X to the S6[5]. This interface was chosen as it allows for the maximum transfer bandwidth between the host computer and S6. Within the S6, data from the FT245 interface is held in the RX FIFO buffer to allow for a simplified command parsing system. The `parser` submodule takes a sequence of bytes from the RX buffer, and operates on them as described in Section 2.1.2.2 below. This typically involves sending a signal to the K7 via one of the discrete interfaces, or reporting data back to the host computer as requested.

The S6 then has a plethora of interfaces to the K7. This design allows for the K7 to use whatever interface best fits its requirements, typically as measured by the amount of decoding hardware required. The first interface is the 4 trigger wires, which allow the host to send a single cycle pulse over any of them to the K7. These are targeted for use as a remote reset control or to trigger a round of encryption on the K7. Next are the 4 control wires, which can each be set to high or low. These are most useful for low level, simple configurations, such as enabling a subsystem or as a debugging aid. Relatedly, there are 4 status wires which can be read back to the host computer as needed. These are also most commonly used as a debugging aid. Fourth is the pair of transmit and receive parallel ports, which provide a simple interface to read or write one byte at a time. These are great for providing a plaintext for encryption or reading back the results within the K7, and since they are parallel ports they require only a small amount of logic on the K7 to transmit and receive. Finally, the configuration port is excellent for writing a small set of data into registers on the K7. This configuration bus transmits 8 bits of data at a time to any of 2^{16} registers as defined by the 16 bit address. The data is considered valid on both the rising and falling edges of the write pin. This allows for a fairly simple synchronization circuit within the K7, and then lightweight decoding that can be distributed across the fpga as needed. This interface is best used for defining operating modes, setting encryption keys, and of course for debugging.

2.1.2.2 FPGA Firmware: Control Protocol

The SCLAF's DUT control protocol is a relatively simple packet based protocol, designed to maximize flexibility while keeping integration simple. Each packet starts with a command byte, followed by up to 2 address bytes and up to 128 data bytes depending on the command. This protocol has no error checking. The command definitions are listed in Table 2.1.2.2. The BULK_TX, CONFIG_K7, and CONFIG_S6 commands each encode the number of data bytes to receive into the command byte, while EXTERNAL_* commands each encode the entirety of the necessary data into the command byte. The remaining command bytes are reserved and ignored by the S6 if they are ever received.

Table 2.1: Control-byte bit definitions

B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0	Command
1	n	n	n	n	n	n	n	BULK_TX
0	0	0	0	n	n	n	n	CONFIG_K7
0	0	0	1	n	n	n	n	CONFIG_S6
0	1	0	0	d	d	d	d	EXTERNAL_TRIG
0	1	0	1	d	d	d	d	EXTERNAL_CONTROL
0	1	1	0	0	d	a	a	EXTERNAL_CONTROL_SINGLE

- n: Number of bytes in packet, minus 1.
- d: Value of control/trigger wire
- a: Selection of which control wire

BULK_TX transmits a sequence of data via the parallel TX port to the K7. The remainder of the packet (up to 128 bytes) is sent directly to the K7 via said port. The number of bytes to transmit is defined by the lower 7 bits of the BULK_TX command, and can be calculated as num_bytes = (command_byte % 128) + 1. Transmissions longer than 128 bytes are accommodated through multiple BULK_TX packets.

CONFIG_K7 and CONFIG_S6 commands are handled the same way, the only difference being whether the resulting parameters are sent to the K7's configuration bus or the S6's internal bus. CONFIG_* commands support up to 16 sequential registers to be written in a row, this quantity is defined similarly to BULK_TX by the formula num_bytes = (command_byte % 16) + 1. Following the command byte are the two address bytes in big-endian order, followed by num_bytes data bytes for sending to the sequence of addresses following the specified address.

EXTERNAL_TRIG and EXTERNAL_CONTROL are 1-byte packets that operate in a similar manner. One sends single clock pulses to the TRIGGER output port while the other sets the state of the CONTROL output port. In the case of triggers, bits 3 through 0 correspond to trigger wires 3 through 0, which are each set high for 1 clock cycle if the respective bit is a 1. The state of the control wires is directly set to the value of the respective bit.

EXTERNAL_CONTROL_SINGLE allows for updating a single control wire, if the state of the other wires is unknown or for simplicity of software development on the host. Bits 1 and 0 set the index of which control wire will be adjusted, and bit 2 holds the value that the wire will be set to.

SCLAF's DUT can also send data from the K7 to the host computer. This is normally in the form of directly passing on any data received on the parallel RX port, but can also include special injected packets. All injected packets start with the escape byte '@' = 0x40, followed by a command byte and up to 16 data bytes; to send a regular '@' from the K7 to the host computer involves simply sending the '@' byte twice.

Each command byte consists of a 4 bit command and 4 bits determining the data length. This allows each packet to transmit up to 16 bytes. The relevant commands are listed in Table 2.1.2.2.

Table 2.2: S6 Response Command Byte Definitions

Command Byte	Command Name	Description
0x0n	SEND_VERSION	Sending n+1 bytes describing the S6's current hardcoded version string; useful for compatibility verification by the host.
0x1n	SEND_STATUS	Sending n+1 bytes of the current "status" of the S6. This was implemented as a packet with the current state of the status input wires and the state of the physical buttons and switches connected to the S6 on the Sakura X.
0x40	SEND_ESCAPE	Special: no data, the sequence `[0x40, 0x40] = '@@'` replaces a single `'@'` received from the K7.

The initial mechanism for requesting a version or status packet involves sending a configuration packet to the S6's internal configuration bus. Setting address 0x0000 to 0x01 requests a version packet, and setting the same address to 0x02 requests a status packet.

2.1.2.3 FPGA Firmware: Kintex 7

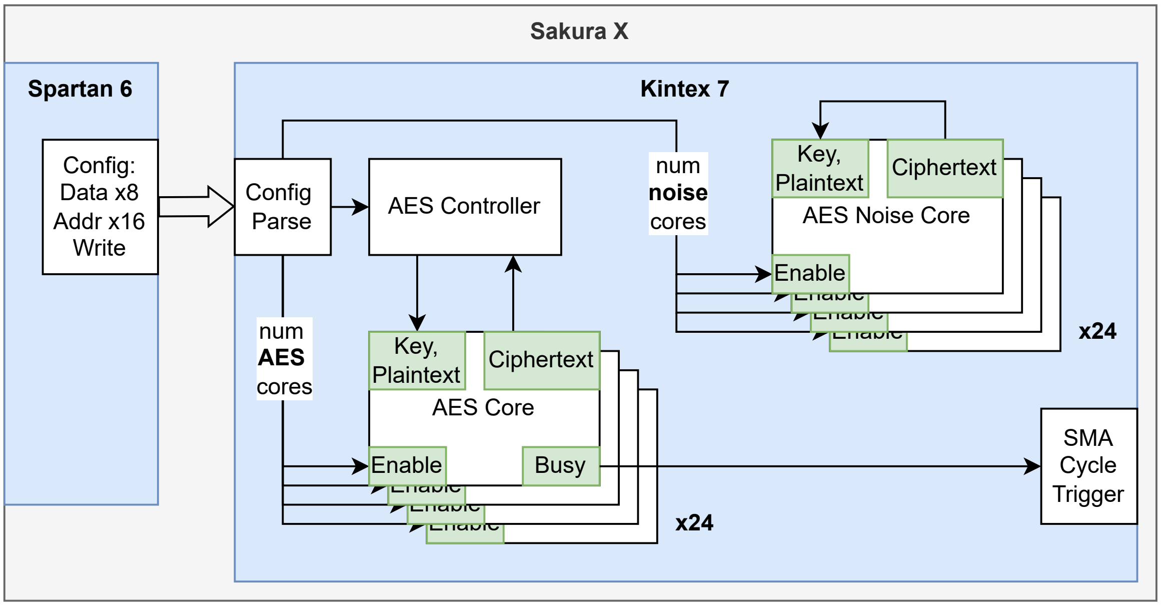


Figure 2-6: Block diagram of Kintex 7 as configured for AES core replication analysis.

As the Sakura X's K7 mainly holds the SCLAF's AUT, its design can vary widely based on the experiments being run. In this section we discuss the general requirements of the K7's design, and then look at the specifics involved for the AES core replication case study.

In order to effectively perform the analysis described in Section 2.2, the K7 must minimize any operations that can be temporally correlated to the computations of the AUT. To minimize these, the K7 operates with two different clock domains generated with two separate crystal oscillators. The first is dedicated for support tasks such as communicating with S6, while the second domain is exclusively used for running the AUT. By isolating these clock domains, any fluctuations in power draw caused by the control circuitry is guaranteed to be data- and time-independent when referencing the

AUT clock domain, and thus will only show up as noise in the SNR measurement.

For the purposes of the AES core replication case study, the K7 was designed as is shown in Figure 2-6. The main feature is a set of 24 parallel AES cores, configured to operate in unison. One of the configuration registers is then dedicated to controlling the number of AES cores that are enabled. This acts as a method of parametrically amplifying the signal of a single AES core - when all 24 cores are enabled, the current draw of each operation is nominally 24x higher, and thus any leaked information is 24x stronger. All of these cores are on the same dedicated clock domain. Of note, the first core is always enabled, and thus only that core's busy signal and ciphertext outputs are ever used. The rest are ignored.

There is a similar set of Composite AES Noise Cores, which can also be enabled parametrically. These operate on the configuration clock domain, and simply recycle the ciphertext as the next round's plaintext. This creates a simulated noise source which crucially is not temporally correlated to the AES core's operations.

Finally, the AES Controller module can be configured a few different ways. The main mode triggers another round of encryption every 255 clock cycles, giving the system time to settle between rounds. This is used extensively when collecting data for the SNR measurement. Alternatively, the controller can be configured in a loopback mode, which passes the last computed ciphertext in as the next plaintext, again with the 255 cycle loop rate. This mode is particularly useful for rapid data collection for CPA/DPA analysis. The final mode is a manual mode, allowing for the S6 to send discrete triggers to perform a single round of encryption.

2.1.3 Command and Analysis Software

In designing SCLAF, we had a few major goals for the host-side software toolchain (shown in Figure 2-7). Primarily, we needed it to meet the requirement of effectively and accurately measuring SNR. Beyond this, we strove to make it highly automated and robust - when analyzing a large design space for PSC leakage, it is often desirable to run a large number of tests, for which said goals can be a major benefit. Additionally, we needed SCLAF to be flexible, as different users may have different equipment

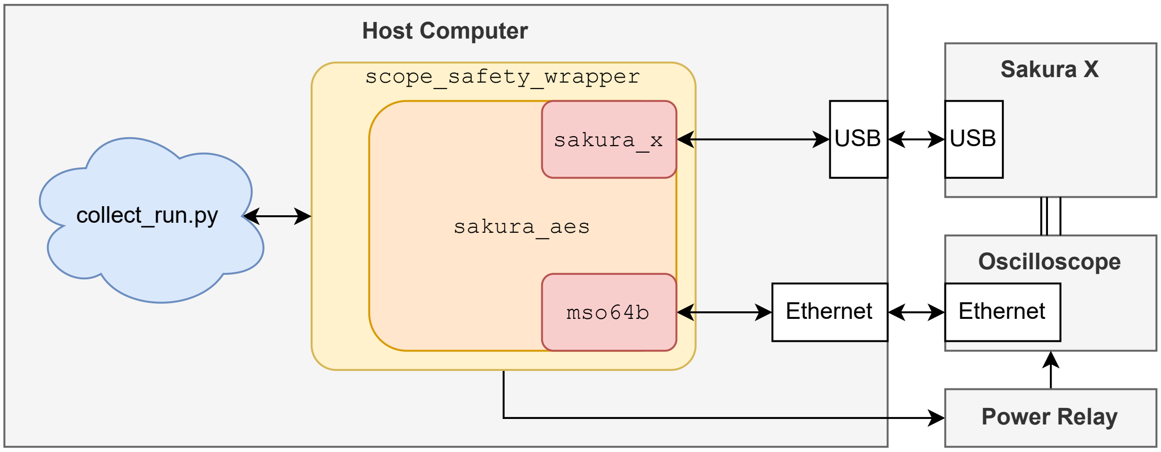


Figure 2-7: Block diagram of host computer software architecture.

available or different target devices to test.

The first major software goal is automation - as these tests require thousands of traces, manually collecting traces with an oscilloscope is infeasible. This naturally affects most aspects of this project, the first being control of the AUT/DUT. Section 2.1.2.2 covers the ability to control the DUT via a USB port, and we created a matching python module `sakura_x` to abstract away the serial protocol and provide a high level API to control the S6 and its interfaces to the K7. This module also handles parsing data returned from the Sakura X into the raw bytes sent by the K7 and the various packets that the S6 can inject.

The oscilloscope's data collection and handling needs to be automated in parallel, for the same reasons as above. We used a Tektronix MSO64b series oscilloscope for all data collection in this project; this scope has a VISA interface that allows complete control over the scopes operation [13]. We created a python module, `mso64b`, to interface with the VISA port and control the scope. This module abstracts away the VISA commands, providing interfaces such as `acq_run()` to initiate an acquisition sequence or `get_curve` to download the raw samples of a recorded waveform after the acquisition is complete.

Next is the `sakura_aes` module, a layer of abstraction that uses the `mso64b` and `sakura_x` interfaces to run individual tests of an experiment, by collecting waveforms of the AES core performing encryptions. For example, a call to `runTestsFastFrame()`

will first use the `sakura_x` module to configure the K7's AES core(s) to use the specified secret key and plaintext (key-text-pair or KTP) and start encrypting said KTP in an infinite loop. Then it will use the `mso64b` module to configure the oscilloscope to record a certain number of encryptions in the FastFrame mode, initiate the collection sequence, and finally download and return the collected waveforms.

One related advantage of modularizing both the `sakura_x` and `mso64b` controllers is that both can be relatively easily replaced if different hardware is in use. This same advantage applies to the `sakura_aes` module - when investigating an AUT other than AES, this module can be modified or replaced as necessary to support the needs of the new algorithm without having to rewrite any of the hardware interface code. For example, upon updating the K7's firmware this is often the only software module that needs to be updated with new configuration parameter definitions.

With automation mostly handled, the next major goal was to make SCLAF robust, meaning that data collection runs should be reliable and unlikely to crash. Throughout the course of this work, we had significant issues with long running data collections failing part way through. This was generally attributed to one of three things: random network errors, crashing the oscilloscope, or crashing the K7 on the Sakura X. The solution to the random network errors is to simply try the failed command again a few seconds later, or at worst rerun the current test. Similarly, crashing the K7 generally only required the current test be rerun. However, the oscilloscope we used had more significant issues. In certain cases, its response time to VISA commands would slow down so significantly that network operations would time out, while in other cases it would stop responding to all input and require a reboot. For the former, we found that reloading the setup file before it slowed to the point of timing out would often fix this issue, while the hard crashes necessitated installing a network-controlled power strip to force the complete power cycle.

We created `collect.py` to handle these errors. This module is effectively a wrapper for the sakura_aes's test procedures, but it adds multiple levels of error checking. Whenever a test fails, `collect.py`'s wrapper functions first try simply running the test again a few seconds later. This tends to resolve any failures due to intermittent network faults or K7 crashes. Should this second attempt also fail, `collect.py`'s `scope_safety_wrapper` method takes over. This method was designed to handle all issues we found with our oscilloscope. Upon a test failing twice, the `scope_safety_wrapper` attempts to reconfigure the oscilloscope by reloading the current test's setup file, and then runs the test again. This soft reset typically resolves any issues related to the scope slowing down, and is thus an efficient way of resolving those issues. Should this also fail, the `scope_safety_wrapper` will force the oscilloscope to reboot by power cycling it. This is a slow process, as our oscilloscope took around 10 minutes to power on, but was quite reliable in resolving the hard crashes. Once the oscilloscope is back online, `scope_safety_wrapper` attempts to run the test one last time.

2.2 SNR Calculation

2.2.1 SNR Algorithm

Yano et al. propose Equation 2.1 as a way to efficiently calculate SNR, and show that this value can be used as a design criteria when evaluating power side channel defense mechanisms. Specifically, they find that this method can replace both of the two other metrics that were commonly used for specifying SCA resistance criteria: maximum correlation coefficient and minimum number of traces needed to disclose the secret key [14]. Since decreasing SNR decreases correlation coefficients and increases the number of trials required to extract a secret key via CPA, maximum SNR can be used as a replacement design criteria.

$$\mathrm{SNR_i} = \frac{\mathrm{Var}(V_{signal})}{\mathrm{Var}(V_{noise})} \tag{2.1}$$

Equation 2.1 is a measurement of the power side channel's noise at any given time within the AUT's operation, and as such yields a curve representing SNR given a set of traces measuring the AUT's signal and noise components. The noise component is collected by setting both the key and plaintext input to `0`, and collecting a set of traces of the AUT repeatedly operating on that KTP. This allows the calculation of Var(V_{noise}) to only detect the random noise inherent to the system, and exclude any variation that may be data dependent (as the KTP is constant) or process dependent (as that is constant for each sample offset).

Measuring Var(V_{signal}) is similar, but requires an extra step to isolate the signal component. For each trace, first select a random KTP, then collect a set of waveforms using said KTP, and finally average together that set of waveforms on a per-sample basis to generate the trace. Any random noise is thus removed from each trace by collecting enough waveforms. Computing the variance of this set of traces removes process-dependent variations, and only leaves a measurement of how data-dependent the core's power draw is.

One major advantage of using Yano's definition of SNR over other leakage measurement techniques are that it does not use a leakage model. Metrics that use leakage models can struggle to compare differing algorithms - a suboptimal leakage model can inaccurately indicate that said AUT is secure. As such, excluding a leakage model makes this technique algorithm-agnostic.

Another benefit of using Equation 2.1 is that it does not require a huge and variable number of traces to accurately measure. Measuring low-leakage designs often requires hundreds of thousands of traces to extract information about the key and thus can take an extremely long time to perform. With Yano's design, increasing the number of traces can shrink the SNR's confidence interval, but otherwise doesn't affect the measured value. As such, Yano's design does not require a huge number of iterations.

2.2.2 SNR Measurement Procedure

Our calculation of SNR is based on Yano's, with some minor modifications to the procedure. The biggest difference between our use of Equation 2.1 and Yano's is that when measuring Var(V_{signal}), we kept the number of averaged traces constant at 512 instead of just 50. This allows us to make an ideal signal strength measurement without any noise added by our measurement system. We selected 512 traces as this was enough to remove effectively all noise from our signal measurements, see Section

3.1 to see how we found this value.

Another minor change is that we only collected 4000 signal and 4000 noise traces, instead of 10,000 of each. Once again, we found that increasing the number of collected traces only affected our confidence interval, and that 4000 traces was plenty for our purposes.

Finally, our goal was to return a single number representing SNR of an AUT, not a set of values representing the SNR at each timestep within the trace. To do this, we decided to simply return the maximum measured SNR (as in Equation 2.2). This aligns with attack algorithms (such as CPA) only operating on the sample with maximum correlation. We further believe this to be valid as [6] also used the maximum measured variance as their final output, though we will note that they used a slightly different definition of SNR.

$$\text{SNR} = \max(\frac{\text{Var}(V_{signal})}{\text{Var}(V_{noise})}) \tag{2.2}$$

2.2.3 SNR Measurement Implementation

Measuring $\text{Var}(V_{noise})$ involves measuring 4000 traces, which we can do quite efficiently using SCLAF. Specifically, the `runTestsFastFrame` function is designed to gather a large set of waveforms in a row, by utilizing the oscilloscope's FastFrame mode. Once we have this set of traces, we can simply compute the per-sample variances, which will yield the desired $\text{Var}(V_{noise})$.

Measuring $\text{Var}(V_{signal})$ is a little harder, as it involves measuring 4000 averaged traces, which are each a set of 512 individual waveforms. While this certainly could be done in the same manner as $\text{Var}(V_{noise})$, this would not be particularly efficient due to the large amounts of data that would have to be sent over the network from the oscilloscope to the host computer. Instead, we can use another of the oscilloscope's modes, this time the FastAcq Average mode. This will create a running average of the captured waveforms, allowing averaged traces to be downloaded directly. Once again, we can simply compute the per-sample variances of this set of traces, which will yield the desired $\text{Var}(V_{signal})$.

Finally, we calculate the SNR by following Equation 2.2, namely by selecting the maximum of the per-sample SNR ratios.

2.3 Modifications for Testing with the CEP

A limitation of the methods used so far in this work is that they are all focused on a device with only a single cryptographic accelerator. However, in the real world most devices with cryptographic accelerators tend to have more circuitry inside them operating in parallel. We have attempted to account for this by adding the random noise cores to the case study. However, they are of inherently limited use as they are only a rough simulation of what the remaining circuitry could look like. One way of improving realism of the SNR measurements is to run them while integrated into a representative processor. We have thus begun laying the groundwork for integration of the Common Evaluation Platform (CEP) into SCLAF, and integration of SCLAFinto the CEP [3]. The CEP is an open source RISC-V based System on a Chip, and is designed to be a common platform for testing various tools and techniques in a system representative of real world use. As such, it is an excellent target for combining with SCLAF.

2.3.1 CEP on Sakura X: Hardware

The CEP has certain hardware requirements that the Sakura X does not meet out of the box. Namely, the CEP requires an SD card to boot Linux, and it requires a jtag port to debug the processor. Fortunately the Sakura X includes a FMC port which allows for easy expansion of its IO, which allows us to relatively easily add the necessary ports. Appendix A shows a schematic of the board, the only interesting detail is the need for logic level converters U1 and U2. These are necessary due to the K7's io operating at 2.5v, while the SD card requires 3.3v. Figure 2-8 shows the final board installed on a Sakura X.

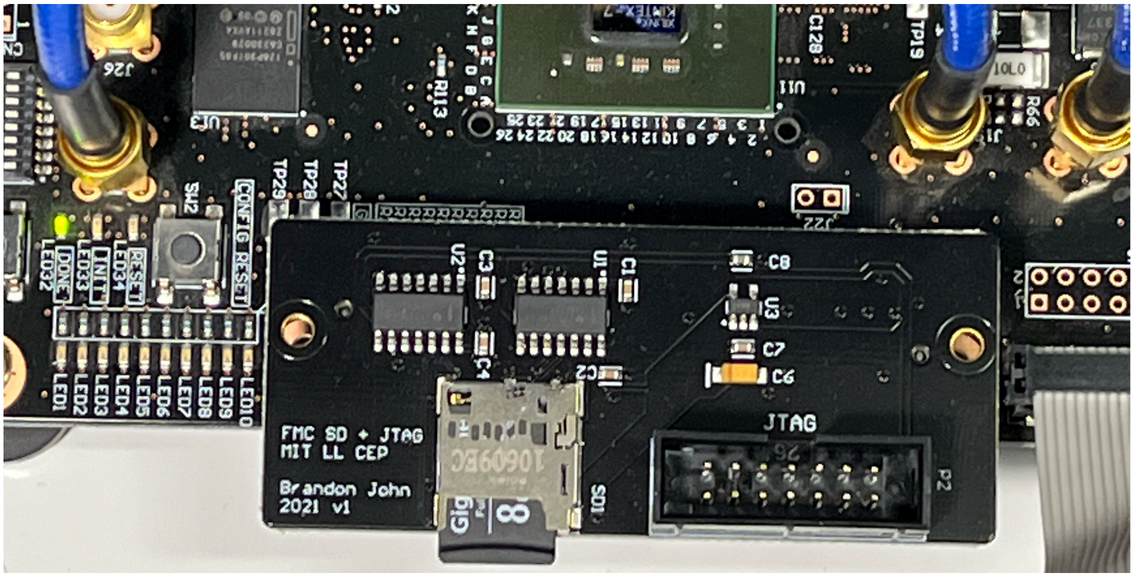


Figure 2-8: Picture of the final FMC adapter.

2.3.2 CEP on Sakura X: Firmware

The CEP was originally designed to be run on a VC707 [8], which is a much more capable FPGA than the K7 in a Sakura X. As such, porting the CEP to fit in the K7 is a non-trivial task which we are still working on as of this publishing.

Chapter 3

Validation

We evaluated the SCLAF by the following three metrics:

1. Functionality
2. Repeatability
3. Efficacy

All tests were performed with the entire SCLAF toolchain, except as noted below. We used an MSO64B series oscilloscope configured to sample at 25 GHz for 1 µs per waveform. Inputs 1 and 2 were both set to AC coupling, 1 MΩ impedance, and 20 mV/div.

The Sakura X is configured as described in Section 2.1.1 and placed in a shielded test enclosure. Its K7 is loaded with either the LUT based or composite AES core, with configurable parallelism of up to 24 parallel AES cores and 24 noise cores.

3.1 SCLAF Evaluation

One of the first tests we performed on SCLAF was to validate that we could successfully perform a standard CPA attack on a sample LUT-AES core. For this test we were not concerned with efficiency or quantifying the results, but only proving that the SCLAF was correctly set up as a representative device for further analysis. As such, the AUT is configured as a single core with no noise added. We collected 15,000 individual traces, and used the ChipWhisperer python library to analyze them [12]. Specifically, we used the `last_round_state_diff` leakage model to run the CPA attack. We found that this set of data paired with this leakage model was sufficient to extract all 16 subkeys. The correct keyguesses had an average correlation of 0.108 while the per-subkey 2nd best guess correlations averaged 0.035. This indicates a very high confidence that the key is correctly guessed, and proves that the SCLAF was capable of collecting usable data.

With this basic functionality confirmed, the next tests focused on repeatability - we needed to make sure that for a given AUT, SCLAF yields a consistent SNR. For this, we started by investigating the measurement of noise, $\mathrm{Var}(V_{noise})$, and how much it was impacted by environmental factors. The first iteration of this experiment had the entire system sitting on a bench, with no attempts to shield it from the environment. We performed a sequence of identical trials spaced out by 40 minutes each over the course of a couple days. For each trial we collected 4000 traces (without tracewise averaging), then calculated the variance curve of each trial in the same manner as we do for SNR calculations. Finally we computed the average value of the variance curve for each trial, and plotted them by time of sample collection (see Figure 3-1). When we did so, we found that there was a massive spike in measurable noise during business hours, with the maximum average variance almost 225% above the nighttime average. Such a large variance in the variances was clearly not acceptable, so we migrated the Sakura X into a shielded test chamber. This was a massive improvement, as the peak average variance was reduced to merely 22% higher than the minimum. We also discovered that this delta was reduced to only 1% over the weekend when no one was working in the lab. As such, we decided to run all further experiments at night or on weekends.

The next test was designed to determine the amount of tracewise averaging needed for the later SNR analysis. For this test we used a similar setup to the one used for the CPA test, but we used the composite AES core instead and only collected 2000 traces per experiment. Additionally, for each trial in this experiment, we varied

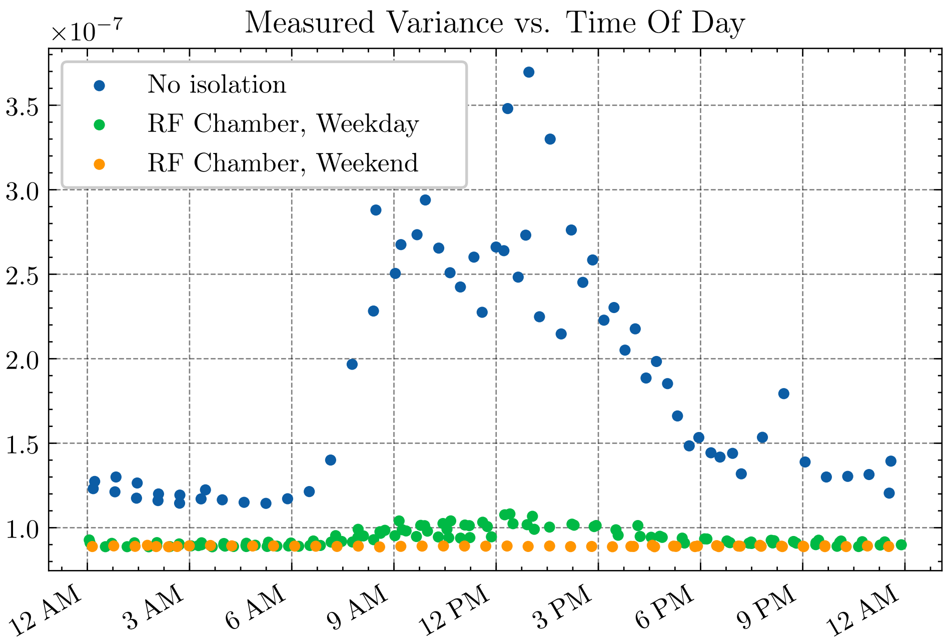


Figure 3-1: Plot of each trial's average variance vs time of day.

the amount of tracewise averaging between 2 and 2048. This means that instead of collecting a single trace for each KTP, we actually collected a number of traces and averaged said traces together. This is the same mechanism that is used for the signal variance portion of the SNR algorithm. We then ran CPA on the resulting set of averaged traces, and recorded the average correlation of the correct subkey guesses. The resulting data is shown in Figure 3-2. In this plot we can see that the correlation increases as we increase the number of averaged traces per KTP, until it plateaus around 250 traces per KTP. This tells us that for our particular setup, we need to average a minimum of 256 traces per KTP when calculating the signal variance as part of the SNR measurement. We therefore decided to average 512 traces to give a decent factor of safety.

3.2 AES Case Study

To validate the efficacy of SCLAF, we performed a case study measuring AES's SNR across three independent variables: S-Box implementation, core parallelism, and added noise. The two S-Box implementations were the LUT and Composite models,

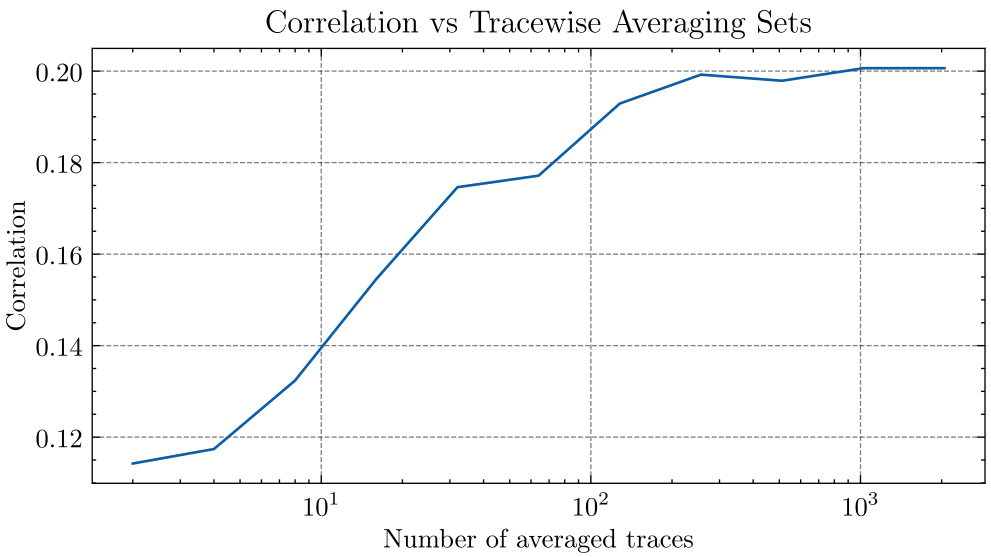


Figure 3-2: Plot of the average correct-subkey correlation in a CPA attack vs number of averaged traces per trace set.

specifically those provided with the Sakura X. We used a single bitstream for all LUT trials, and a second bitstream for all Composite trials. In both cases, the number of noise and AES cores enabled was set via a configuration register through the S6 interface, allowing for rapid testing without needing to compile a large number of bitstreams. The AES and noise cores were configured as described in Section 2.1.2.3.

Figure 3-3 shows the results of this experiment. As expected, increasing the number of signal cores increases the SNR, while increasing the number of noise cores decreases the SNR. The SNR of the LUT and Composite designs were fairly similar, but on average Composite AES cores tended to have a slightly higher SNR than their LUT counterparts. This generally aligns with our expectations, though we note that the difference in SNR between them is quite small and so is not entirely conclusive.

One limitation during this experiment is that we couldn't actually reliably enable all 24 signal and noise cores simultaneously, the Sakura X would often reboot as soon as we enabled this mode. We suspect that total power draw was either too much for the Sakura X's power supply, or otherwise had too large of a slew rate and browned out the entire K7 fpga.

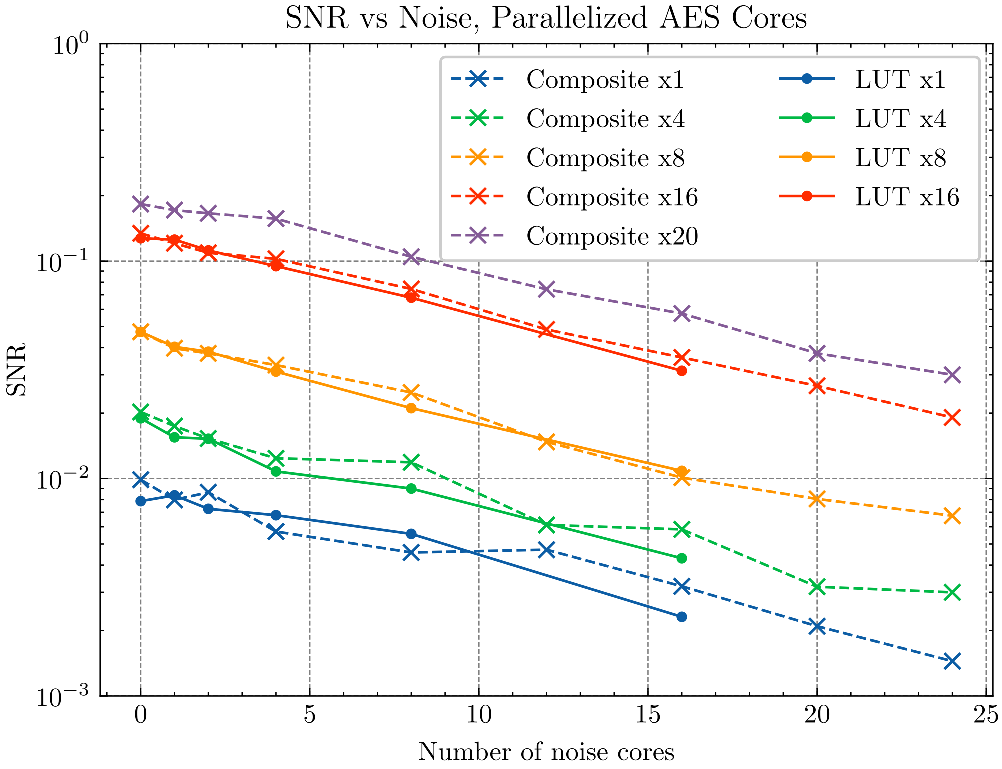


Figure 3-3: Plot of SNR vs number of AES cores enabled and number of noise cores enabled.

3.2.1 CEP on Sakura X

While running the full CEP on a Sakura X is still a work in progress, we were able to validate the hardware designed in Section 2.3.1. To do this we created a custom bitstream for the K7 to read and write individual blocks on an SD card. We then used the SCLAF toolchain to control the K7 to read and write data. We successfully wrote initialized the SD card, wrote data to it, and read the correct data back, showing that the hardware portion is ready to go.

Chapter 4

Discussion

While we have shown that SCLAF is a useful tool for rapidly analyzing AUTs for leakage, it is certainly far from a perfect tool. In this chapter we discuss some of the current limitations and how they can be mitigated in future versions of this tool.

4.1 Limitations

One major limitation of the current design is that it requires control over the input KTP, and an oracle to trigger data collection by the oscilloscope. This means that for the most part, SCLAF can only support DUTs that the investigator has full control over. Devices found in the real world are unlikely to have convenient AUT synchronization trigger ports available, significantly complicating the sample collection step. KTP control may also be a challenge with arbitrary DUTs as this method requires thousands of samples of each of a set of KTPs. However, this is still better than some other methods of comparing leakage where the KTP must be known - the investigator only needs the ability to replay an encryption sequence, and doesn't have to know the key or data being operated on to measure the DUT's SNR.

Beyond operational challenges, the method's SNR calculation has some nuanced limitations. Namely, it currently selects the largest instantaneous value from the SNR curve and uses that as the SNR of the device. However, there could be devices where multiple samples per trace could be used together to achieve better results than a single sample. The current method has no way of accounting for this.

4.2 Future Work

Looking forward, there is certainly plenty more work that can be done to improve the SCLAF. One possible improvement is to upgrade the current probing scheme - perhaps a local low-voltage differential probe could work to reduce the noise floor beyond what is currently possible with several feet of cable and the math channel subtracting the two sides.

There is also plenty of room to optimize the current data collection process. For example, the scope is currently configured to sample at 25 GHz, but the bandwidth of the sense resistor probes are limited to 500 MHz. Thus a reduction in the sampling rate could correlate to an equivalent reduction in the amount of data that needs to be processed, without any signal degradation. Likewise the recorded waveforms for this thesis all began 0.1 µs before the AES encryption even began, and continued for a while after the encryption round completed. SCLAF could be improved to allow for cropping the waveforms to the relevant locations, which can speed up the analysis step by reducing necessary computation proportionally to the reduction in data.

Another challenge involves comparing SNRs between devices. While we have shown that this method can consistently measure SNR within a particular DUT, we should investigate methods of calibrating across separate DUTs and measuring equipment. It remains to be seen how factors such as sampling rate, scope proximity, and even probing affect the SNR.

More broadly, the SNR calculation currently relies on the understanding that $\max(SNR_{trace})$ is a useful model-agnostic test. While this appears to be true given the data collected in this thesis, this claim probably deserves further attention with a broad array of AUTs.

Finally, we would like to see a proper comparison between isolated accelerator cores and those integrated into a realistic processor. While the work in this paper began this investigation using the CEP, the unfortunate reality is there were too many uncontrolled variables to make a proper comparison. The next steps here will likely involve porting the CEP onto the Sakura X to eliminate most of the variables, or otherwise finding a good method of calibrating SNR measurements.

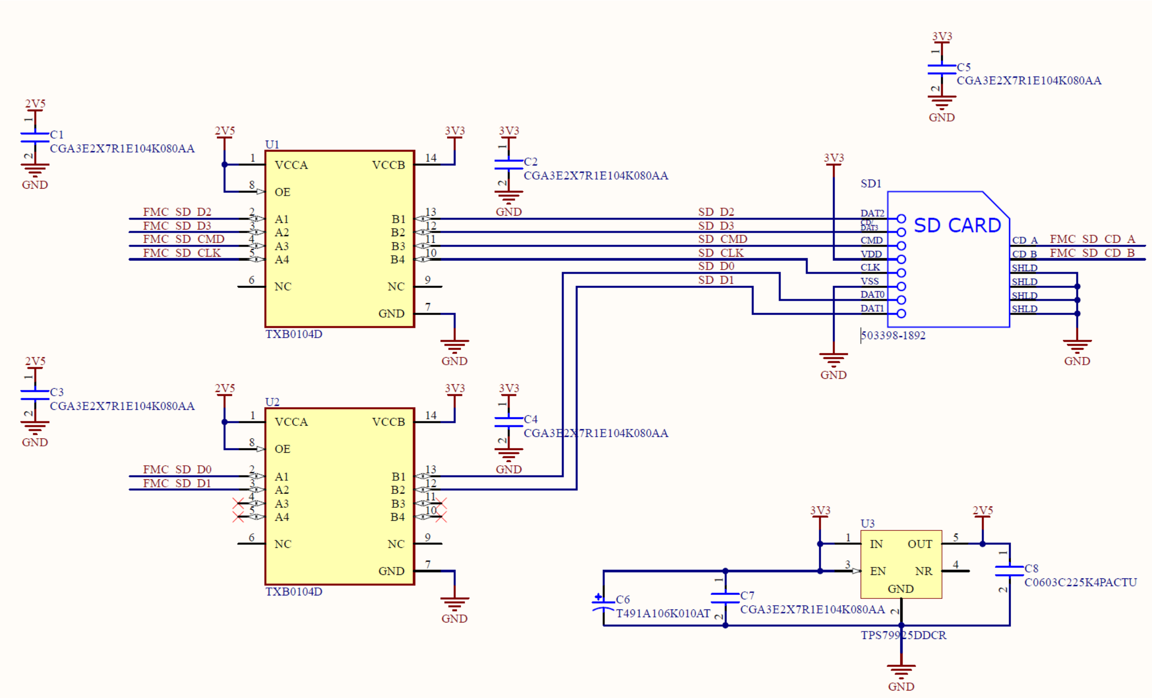
Chapter 5

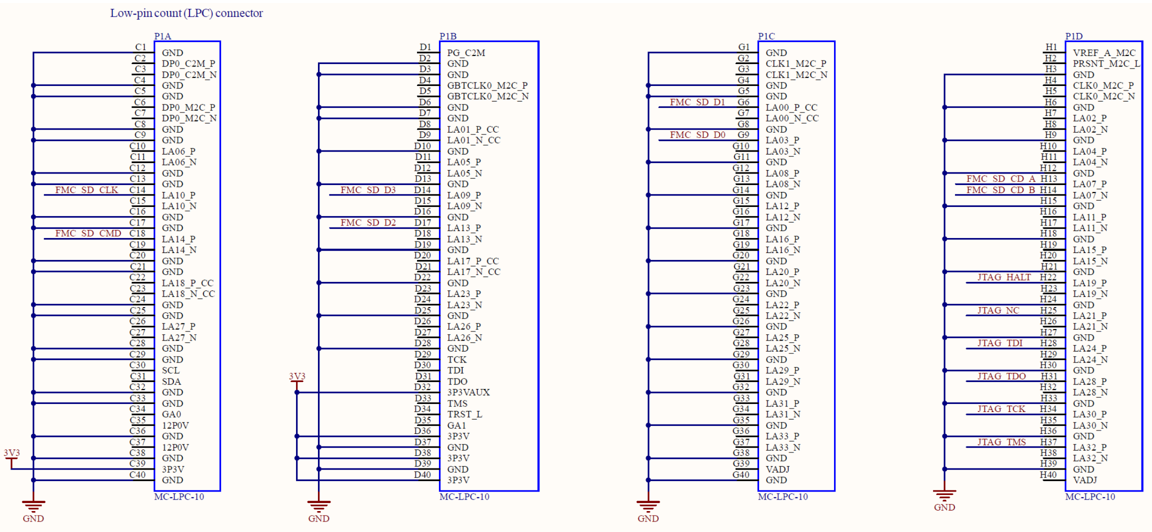
Conclusion

Physical side channels are real and dangerous, and the security community must focus on reducing their impact. While current research shows many potential avenues for reducing their impact, it is often hard to directly compare the results of different side channel prevention techniques. In that regard, SCLAF has significant potential to help simplify the comparison of PSC defenses. First, it provides a common toolchain to reduce the startup effort for new investigators, making it easier to have more people begin studying PSC defenses. Second, it provides comprehensive automation, allowing for thorough testing of various search parameters with minimal manual involvement. Finally, the use of variance-based SNR allows for algorithm-agnostic comparisons of various algorithms.

Appendix A

FMC SD Adapter Schematic




Low-pin count (LPC) connector
P1A
C1 GND
C2 DP0_C2M_P
C3 DP0_C2M_N
C4 GND
C5 GND
C6 DP0_M2C_P
C7 DP0_M2C_N
C8 GND
C9 GND
C10 LA06_P
C11 LA06_N
C12 GND
C13 GND
C14 LA10_P
C15 LA10_N
C16 GND
C17 GND
C18 LA14_P
C19 LA14_N
C20 GND
C21 GND
C22 LA18_P_CC
C23 LA18_N_CC
C24 GND
C25 GND
C26 LA27_P
C27 LA27_N
C28 GND
C29 GND
C30 SCL
C31 SDA
C32 GND
C33 GND
C34 GA0
C35 12P0V
C36 GND
C37 12P0V
C38 GND
C39 3P3V
C40 GND
FMC_SD_CLK
FMC_SD_CMD
3V3
GND
MC-LPC-10
P1B
D1 PG_C2M
D2 GND
D3 GND
D4 GBTCLK0_M2C_P
D5 GBTCLK0_M2C_N
D6 GND
D7 GND
D8 LA01_P_CC
D9 LA01_N_CC
D10 GND
D11 LA05_P
D12 LA05_N
D13 GND
D14 LA09_P
D15 LA09_N
D16 GND
D17 LA13_P
D18 LA13_N
D19 GND
D20 LA17_P_CC
D21 LA17_N_CC
D22 GND
D23 LA23_P
D24 LA23_N
D25 GND
D26 LA26_P
D27 LA26_N
D28 GND
D29 TCK
D30 TDI
D31 TDO
D32 3P3VAUX
D33 TMS
D34 TRST_L
D35 GA1
D36 3P3V
D37 GND
D38 3P3V
D39 GND
D40 3P3V
FMC_SD_D3
FMC_SD_D2
3V3
GND
MC-LPC-10
P1C
G1 GND
G2 CLK1_M2C_P
G3 CLK1_M2C_N
G4 GND
G5 GND
G6 LA00_P_CC
G7 LA00_N_CC
G8 GND
G9 LA03_P
G10 LA03_N
G11 GND
G12 LA08_P
G13 LA08_N
G14 GND
G15 LA12_P
G16 LA12_N
G17 GND
G18 LA16_P
G19 LA16_N
G20 GND
G21 LA20_P
G22 LA20_N
G23 GND
G24 LA22_P
G25 LA22_N
G26 GND
G27 LA25_P
G28 LA25_N
G29 GND
G30 LA29_P
G31 LA29_N
G32 GND
G33 LA31_P
G34 LA31_N
G35 GND
G36 LA33_P
G37 LA33_N
G38 GND
G39 VADJ
G40 GND
FMC_SD_D1
FMC_SD_D0
GND
MC-LPC-10
P1D
H1 VREF_A_M2C
H2 PRSNT_M2C_L
H3 GND
H4 CLK0_M2C_P
H5 CLK0_M2C_N
H6 GND
H7 LA02_P
H8 LA02_N
H9 GND
H10 LA04_P
H11 LA04_N
H12 GND
H13 LA07_P
H14 LA07_N
H15 GND
H16 LA11_P
H17 LA11_N
H18 GND
H19 LA15_P
H20 LA15_N
H21 GND
H22 LA19_P
H23 LA19_N
H24 GND
H25 LA21_P
H26 LA21_N
H27 GND
H28 LA24_P
H29 LA24_N
H30 GND
H31 LA28_P
H32 LA28_N
H33 GND
H34 LA30_P
H35 LA30_N
H36 GND
H37 LA32_P
H38 LA32_N
H39 GND
H40 VADJ
FMC_SD_CD_A
FMC_SD_CD_B
JTAG_HALT
JTAG_NC
JTAG_TDI
JTAG_TDO
JTAG_TCK
JTAG_TMS
GND
MC-LPC-10

Bibliography

[1] Georg T. Becker, Jim Cooper, Elizabeth K. DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, T. Kouzminov, Andrew J. Leiserson, Mark E. Marson, Pankaj Rohatgi, and Sami Saab. Test vector leakage assessment (tvla) methodology in practice. *International Cryptographic Module Conference*, 1001, 2013.

[2] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Lecture Notes in Computer Science, volume 3156*, volume 3156, pages 16–29, 08 2004.

[3] Brendon Chetwynd. *mit-ll/CEP: CEP Release v3.41*, September 2021.

[4] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. *Journal of Cryptographic Engineering*, 1:123–144, 04 2012.

[5] Future Technology Devices International. *FT2232H Used in an FT245 Style Synchronous FIFO Mode*, 11 2015. Version 1.3.

[6] Sylvain Guilley, Laurent Sauvage, and Jean-Luc Danger. Quantifying the quality of side-channel acquisitions. In *COSADE 2011 - Second International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2011.

[7] Yohei Hori, Toshihiro Katashita, Akihiko Sasaki, and Akashi Satoh. Sasebo-giii: A hardware security evaluation board equipped with a 28-nm fpga. In *The 1st IEEE Global Conference on Consumer Electronics 2012*, pages 657–660, 2012.

[8] Xilinx Inc. *Xilinx Virtex-7 FPGA VC707 Evaluation Kit*, May 2012.

[9] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[10] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards.* Springer Publishing Company, Incorporated, 1st edition, 2010.

[11] National Institute of Standards and Technology. Advanced encryption standard. *Federal Information Processing Standards*, 197, November 2001.

[12] Colin O'Flynn and Zhizhang David Chen. Chipwhisperer: An open-source platform for hardware embedded security research. In *COSADE*, 2014.

[13] Tektronix. *5/6 Series MSO MSO54, MSO56, MSO58, MSO58LP, MSO64 Programmer Manual*, July 2018.

[14] Yusuke Yano, Kengo Iokibe, Yoshitaka Toyota, and Toshiaki Teshima. Signal-to-noise ratio measurements of side-channel traces for establishing low-cost countermeasure design. In *2017 Asia-Pacific International Symposium on Electromagnetic Compatibility (APEMC)*, pages 93–95, 2017.