Compound Effects of Clock and Voltage Based Power Side-Channel Countermeasures

Jacqueline Lagasse

COMPOUND EFFECTS OF CLOCK AND VOLTAGE BASED POWER SIDE-CHANNEL COUNTERMEASURES

A Thesis Presented

by

JACQUELINE R. LAGASSE

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

May 2020

Electrical and Computer Engineering

COMPOUND EFFECTS OF CLOCK AND VOLTAGE BASED POWER SIDE-CHANNEL COUNTERMEASURES

A Thesis Presented

by

JACQUELINE R. LAGASSE

ACKNOWLEDGMENTS

I would like to first thank my advisor, Professor Wayne Burleson, for providing me with the opportunity to do research in this field, as well as for providing me with support and guidance throughout this process. This thesis would not have been possible without him. In addition, I have also received guidance and encouragement from Professors Daniel Holcomb, Russell Tessier, and Israel Koren. Their comments and perspectives on my work have been very helpful.

I am also indebted to my colleagues. Christopher Bartoli was a close collaborator in the early stages of this work, and his contributions helped me to get started on the right path with this research. In addition, his work was crucial in writing the publication that was the foundation for this work. I also want to thank Shayan Moini and George Provelengios for their advice and helpfulness in lab.

I would like to acknowledge that this research was partially funded by The Raytheon Company. Principal Engineering Fellow Jay Lala of The Raytheon Company and his colleagues also gave me feedback on this work from an industrial perspective. Last but not least, I also want to thank my parents, brothers, friends, and especially my partner Michael for their support and encouragement throughout this process.

ABSTRACT

COMPOUND EFFECTS OF CLOCK AND VOLTAGE BASED POWER SIDE-CHANNEL COUNTERMEASURES

MAY 2020

JACQUELINE R. LAGASSE

B.S., UNIVERSITY OF MASSACHUSETTS AMHERST

M.S.E.C.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Wayne Burleson

The *power side-channel attack*, which allows an attacker to derive secret information from power traces, continues to be a major vulnerability in many critical systems. Numerous countermeasures have been proposed since its discovery as a serious vulnerability, including both hardware and software implementations. Each countermeasure has its own drawback, with some of the highly effective countermeasures incurring large overhead in area and power. In addition, many countermeasures are quite invasive to the design process, requiring modification of the design and therefore additional validation and testing to ensure its accuracy. Less invasive countermeasures that do not require directly modifying the system do exist but often offer less protection.

This thesis analyzes two non-invasive countermeasures and examines ways to maximize the protection offered by them while incurring the least amount of overhead. These two countermeasures are called *clock phase noise* (CPN) and *voltage noise*

(VN), and are placed on the same FPGA as an AES encryption module that we are trying to protect. We test these designs against a highly effective algorithm called *correlation power analysis* (CPA) and a preprocessing technique called the *sliding window attack* (SW).

We found that the combined effects of the two countermeasures was greater than the impact of either countermeasure when used independently, and published a paper in the 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP) on our findings. We found that our best combined countermeasure protected about 76% of the maximum amount of traces that a well-known but invasive competitor, *wave dynamic differential logic* (WDDL), could with only about 41% of the area and 78% of the power. However, the sliding window attack significantly reduced the amount of protection our combined countermeasure could offer to only 11% of that offered by WDDL. Since then, we updated our methodology and made some adjustments to VN and CPN. Our CPN countermeasure greatly improved, and therefore so did our combined countermeasure, which on average protected up to about 90% of the maximum amount of traces that WDDL could with only about 43% of the area and about 60% of the power. This is remarkable because these results are after the sliding window attack, meaning that our post-proposal countermeasures protect almost as well as WDDL while requiring only about half of the resources.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

A *side-channel attack* is an attack that is based on information leaked during the normal operation of a computer system, in comparison to exploiting a vulnerability. The leaked information can be any physical phenomena on the system that has some relationship to the calculations being performed on the device. Some of the most popular side-channel attacks are based off measuring computation time, power consumption, or electromagnetic radiation [2]. In a timing side-channel attack, small differences in computation time for different calculations are exploited [3]. Electromagnetic side-channel attacks exploit small changes in the electromagnetic field near the chip during computations. And in power side-channel attacks, changes in power are captured and exploited to determine secret information [4]. The goal of conducting any side-channel attack is to correlate the leaked measurements to some computation, where the computation has some correlation to the secret information.

In this work, we are focusing on power-side channel attacks conducted on *encryption modules*. Encryption modules are used to convert *plaintext* inputs into unintelligible *ciphertext* outputs using a secret key, allowing sensitive plaintext information to only be known to those who can decrypt the ciphertext. This work uses the 128-bit key version of the *Advanced Encryption Standard* (AES-128) for our encryption module, which has a symmetric key, meaning that the secret key used to encrypt plaintext data can also be used to decrypt ciphertext data [5]. AES-128 encrypts a 128-bit plaintext using a 128-bit key over the course of ten rounds. In each round, four computations are performed on the plaintext to compute the ciphertext; *substi-*

tute bytes, *shift rows*, *mix columns*, and *add round key*. The only exception is the last round, which does not include the *mix columns* step. For each round, a key schedule also computes a new round key to be used during the *add round key* step. Depending on the plaintext and the value of the original key, each step computes different immediate values, which each consume slightly different amounts of power.

We are concerned with an attacker with recovering the secret encryption key, which if obtained could be used to decrypt all data on the device. To prevent power side-channel attacks, the power consumption of the device must be manipulated to obscure the correlation between the power consumption and the encryption module's calculations. However, each of these countermeasures impose some sort of penalty. Some penalties include increased area usage, power consumption, and computation time. Our countermeasures incur additional penalties as well, but offer more protection per penalty than other existing countermeasures.

There are many existing countermeasures against power side-channel attacks. Some of these countermeasures are invasive, meaning that they alter the way that an encryption module is constructed and operates. Performing hardware modifications on an encryption module has the consequence of requiring a unique design, such that chip designers are unable to use standard *intellectual property* (IP) modules that have already been verified and are straightforward to add to a design. To modify an encryption module using software, the instructions used by the operating system are altered to modify the behavior of the encryption module, while the hardware is untouched. Invasive countermeasures can be very effective, but remove the ability for a designer to use standard IP blocks, and can introduce the potential for flaws in the encryption process if unverified. All of our designs are *non-invasive*, meaning that they do not require any internal modification of the AES module, which allows designers to implement AES as standard IP. These non-invasive countermeasures can be scaled to be suitable for different designs. For example, a resource-constrained design could use a countermeasure that uses less resources, while a more security-conscious design could use a countermeasure that may offer more protection with the caveat of requiring more resources on the chip.

Previous work has shown that these non-invasive countermeasures are not as substantial as other invasive countermeasures. To overcome this, we experimented with different ways of designing these countermeasures individually, and then combined these non-invasive countermeasures in a novel way to maximize their effectiveness. Our two main countermeasures are *clock phase noise* (CPN) and *voltage noise* (VN). CPN creates a randomly phase-shifted clock that can be used as the input clock to an external module in order to make the execution of the module unpredictable. VN simply generates a random amount of power that is used to obfuscate the amount of power consumed by the encryption module. We published our findings, and a course was developed to teach undergraduate students the basic concepts of power side-channel attacks and other computer engineering security related concepts. We then improved our experimental methodologies, made further modifications to VN and CPN, and conducted a new set of experiments with these countermeasures separately and combined.

The rest of this document is organized as follows: Chapter 2 describes important background information on different power side-channel attacks, including the one used in this work, as well as the countermeasures we used and other existing countermeasures. Chapter 3 covers our experimental methodology, including what hardware and software were used in this work to our threat model and how our results were measured. Chapter 4 details each experiment and its results. Chapter 5 descibes the development of the course and our educational results. And finally, Chapter 6 states our conclusions and lists areas for future work.

CHAPTER 2

BACKGROUND AND RELATED WORKS

2.1 Attacks

2.1.1 Simple Power Analysis

There are several well-known attacks that can be performed on leaked power traces. The most basic is *simple power analysis* (SPA), which is described as “a technique that involves directly interpreting power consumption measurements collected during cryptographic operations” [4]. The changes in power during different computations must vary significantly and there must be very little noise for SPA to work, as SPA is ineffective on noisy data. When these conditions are met, SPA can be used to solve for an encryption key.

In practice, only one or two traces are needed to perform SPA. Features are recognized and analyzed visually, either from one trace or by comparing a pair of traces [6]. With some existing knowledge of how different components of certain systems, such as different types of encryption modules operate, one can examine a power trace to recognize certain features and deduce where different parts of an encryption are being performed. With just one trace, it is possible to deduce whether or not a 0 or 1 was computed in a particular spot by examining the magnitude of the power consumed. With two traces, the power consumption of the two traces can be subtracted to yield the difference between them, which can better expose the amplitude and timing differences between the two traces. With the appropriate data in hand, an actual SPA attack can be implemented via specific algebraic or collision-based attack to compute the encryption key or other desired information.

2.1.2 Differential Power Analysis

The more advanced *differential power analysis* (DPA) technique uses many power traces from encrypting different input data and derives the encryption key though more advanced signal processing techniques. Unlike SPA, DPA is very effective on noisy signals and therefore can be used on a wider array of devices than SPA. However, DPA can be more time consuming than SPA in the sense that many different encryptions need to be performed for there to be enough data to conduct a DPA attack.

The success of DPA relies on what model is used to compare against the power that was actually measured from the device. One assumption that is often made is that amount of power consumed is proportional to the *Hamming weight*, or the number of 1's, in a given byte of data. There are many possible models to chose from, depending on where in the encryption algorithm we would like to attack, which could be the output of any step in any round of the encryption algorithm. Let's say for example that the attacker is observing the output of the last round after adding the round key. With the attacker knowing what plaintext is used, they would know what the state or *intermediate value* of the algorithm would be after the last round add round key step for different key guesses. Each intermediate value has a different Hamming weight. In DPA, one could use the Hamming weight of this intermediary value to partition traces into different groups. Then, all of the traces in each group are averaged together, and then the two groups are subtracted from each other. If there is some sort of correlation between the Hamming weight of this intermediary value for different traces, then there will be a significant difference between these two groups. Otherwise, there will be no statistical difference between these two groups. By repeating this for all possible key guesses, one may eventually be able to deduce the key.

More specifically, DPA uses a Boolean selection function based on a model as described previously. This selection function which determines if a captured ciphertext C and observed bit b from the power trace indeed correlate to a specific possible key value K [4]. This selection function partitions traces into groups, averages the groups, and then takes the difference of these groups to determine if there is any correlation. DPA is a very powerful technique, however it is susceptible to false positive results due to the fact that the S-box implementation's power consumption is also observed by the selection function, called *ghost peaks*.

2.1.3 Correlation Power Analysis

Correlation power analysis (CPA) is the most powerful attack among the three discussed in this work and is therefore what we use for all of our experiments. CPA is a variation of DPA meant to prevent ghost peaks or false positives [7]. Just as with DPA, in CPA we assume that the attacker has access to the plaintexts and ciphertexts used during encryption in addition to being able to capture the power consumed during encryption. CPA also requires a similar model based on where in the encryption we are attacking, such as the Hamming weight of the intermediary state after adding the round key after the last round. The amount of power consumed in this model is correlated to the Hamming weight of the intermediate value. CPA considers every possible value for each byte of the key, computing the correlation between each possible byte and each captured power trace using this model over many traces. These byte-guesses are ranked by this correlation, and the highest-ranked guess is considered to be the most likely key-byte [8]. In our work, we do in fact correlate power traces with the Hamming weight of the potential last-round state resulting from each byte-guess.

The algorithm used to compute the correlation coefficient for CPA used by the ChipWhisperer software is based on a formula that can be found in "Power Analysis

Attacks” by Mangard et al. on page 124, formula 6.2 [9]. This equation is shown below as Equation 2.1. In ChipWhisperer, the algorithm has been optimized to reduce the amount of calculations required. Essentially, there is a matrix R of estimated correlation coefficients, where each individual coefficient is r. The correlation coefficient is used to determine the linear relationship between columns for h_i and $\mathrm{t_j}$, where h is the hypothetical power consumption for i=1,...,K total possible key byte guesses and t is the measured power trace for j=1,...,T parts of the trace. There are D elements in columns h_i and $\mathrm{t_j}$. ChipWhisperer performs this computation for each possible key guess for each key byte of the key for each power trace collected.

$$r_{i,j} = \frac{\sum_{d=1}^{D}(h_{d,i} - \overline{h}_i) \times (t_{d,j} - \overline{t}_d)}{\sqrt{\sum_{d=1}^{D}(h_{d,i} - \overline{h}_i)^2 \times (t_{d,j} - \overline{t}_d)^2}} \tag{2.1}$$

2.1.4 Sliding Window Attack

Preprocessing techniques can be performed on the raw power traces in order to make the power analysis attacks more powerful. In a recent paper, a ”charge and decay modeling” technique was used against a CR countermeasure. It reduced the number of power traces needed for a successful CPA attack from 3 million to less than fourteen thousand [10]. The full 3 million traces still needed to be captured, therefore the time to capture traces was not reduced, but the time to analyze traces was significantly reduced. This modeling technique mainly relies putting the collected power trace through filters and then detecting peaks in power consumption, in essence realigning the power trace with the clock and making the data easier to process using power side-channel attacks.

Another group used a *sliding window preprocessing attack* (SW). This attack is simply a function that sweeps over x sample points of a power trace at a time, magnifying any correlations within the fixed window x [11]. Inside this window, all samples are aggregated, which amplifies the correlation between the data and the power consumption. However, amplification only occurs under certain circumstances, particularly if the appropriate sized window is used such that multiple leakage points fit inside the window, but not too much information is collected as to blur out the result. Choosing the correct window size is crucial to the success of this attack. Using the sliding window preprocessing technique before performing CPA was shown to be significantly more effective than CPA alone against the clock randomization countermeasure a time. These findings suggest that clock randomization alone is not a sufficient countermeasure against power side-channel attacks. Our implementation of the sliding window attack in Python is shown below.

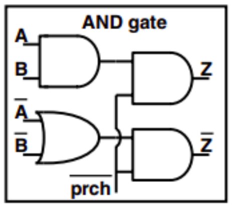
```python
# sliding window attack with window size 'size' on trace array 'traces'
# returns new trace of averaged values

def window_traces(traces, size):

    wndw_traces = []                        # traces to output
    window_size = int(size)                 # store size in int
    half_window = int(window_size/2)        # calculate half of window size

    for trace in traces:
        temp = []
        for i in range(len(trace)):
            cnt = 0
            total = 0
            # sum points within window
            for a in range(-1*half_window, window_size-half_window):
                if(i+a>=0) & (i+a<len(trace)):
                    total += trace[i+a]
                    cnt += 1
            temp.append(total/cnt)
        wndw_traces.append(temp)

    return wndw_traces
```

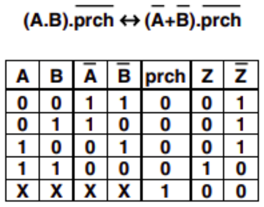
2.2 Hardware Countermeasures

2.2.1 Wave Dynamic Differential Logic

Numerous countermeasures against power side-channel attacks exist and have been proposed. One technique, aimed at removing the correlation between power consumption and data at its source, is to use a completely different logic style that uses the same amount of power for every transition instead of using standard logic gates. One example of this is using *Simple Dynamic Differential Logic* (SDDL) gates, which is inspired by *Sense Amplifier Based Logic* (SABL). SABL is an existing technology that has constant power consumption as it has "exactly one switching event per cycle ... independently of the input value and sequence" but is unable to be implemented using existing standard cell logic gates [1]. SSDL logic gates use the inputs and a precharge signal to determine their output. Their output always consists of two signals, one being logic high and the other being logic low, unless precharge is high n which cause the outputs are both logic zero regardless out the inputs as shown in Figure 2.1. Wave Dynamic Differential Logic (WDDL) is simply SDDL that has been optimized to omit some of the precharge operations in order to reduce the number of standard cells needed for each WDDL logic gate. WDDL logic gates are guaranteed to only have one switching event per cycle. WDDL was implemented and tested in both ASIC and FPGA devices [1].

More thorough testing was performed by implementing an AES-128 module using WDDL on an ASIC device [12]. This device was fabricated using 0.18 µCMOS technology. The 128-bit secret key was not fully disclosed, meaning that some of the bytes of the key were unsolved, after 1.5 million traces using DPA, whereas they found that a standard module would divulge its secret key after only 8,000 traces using the same attack. However, this WDDL device required 3.1 times more area than standard AES to implement, had a maximum operating frequency of only about 25% of standard AES, and used about 3.7 times more power as standard AES. The




$(A.B).\overline{prch} \leftrightarrow (\overline{A}+\overline{B}).\overline{prch}$

A	B	$\overline{A}$	$\overline{B}$	prch	Z	$\overline{Z}$
0	0	1	1	0	0	1
0	1	1	0	0	0	1
1	0	0	1	0	0	1
1	1	0	0	0	1	0
X	X	X	X	1	0	0

Figure 2.1. Example of an AND gate using SDDL logic [1].

metric used to determine whether or not a key has been divulged is *measurements to disclosure* (MTD), which they define as "the crossover point between the correlation coefficient of the correct key and the maximum correlation coefficient of all the wrong key guesses" in a DPA attack. This essentially means that they consider a key, or byte of a key, to be known when the correct key has a greater than or equal to correlation coefficient compared to the most likely wrong key.

While WDDL offers significant protection compared to unprotected AES, it is highly invasive in the sense that it requires the entire encryption module to be redesigned using a different logic style. This creates the potential for design flaws and removes the possibility of a designer being able to use standard IP blocks, which can be a significant drawback.

2.2.2 Voltage Noise

The *voltage noise* (VN) countermeasure is simply a noisy circuit. By generating noise, one attempts to mask the correlation between power consumption and encrypted data by adding extraneous power consumption to the circuit. One approach to creating an efficient VN circuit in an FPGA, in the sense that it generates as much power as possible using the least amount of hardware, involves using *shift register LUTs* (SRL) [13]. Shift registers are simply circuits that input one bit at a time, pushing all of its contents over by one bit, creating a cascading effect. A LUT is a lookup table, which is a fundamental circuit of FPGAs that can be reprogrammed to represent many different functions. When the SRL module is initialized to a value consisting of an alternating pattern of 1's and 0's, and is fed inputs that continue this pattern, maximum power is generated as the circuit is constantly changing the value of all of its registers.

The overall topology of our noisy circuits is what we refer to as VN, which is shown in Figure 2.2. In this topology, there are r rows of shift register modules. An input clock is used to control when a pseudo-random number generator produces new outputs, as well as for clocking the shift registers in each row. The output of the pseudo-number generator is r bits wide, such that there is one bit for each row. These bits are used as enable signals for all of the shift registers in a certain row. In this way, r controls the variance of the noise, as it controls which rows are operating at a given time. In each row, there are s individual shift registers. By increasing or decreasing s, one is able to change the amplitude of noise generated when a row is enabled.

We have tested two different designs which were used as the contents of the rows in Figure 2.2. The first design is the simplest, and is denoted as SRL after the shift register LUTs themselves. For the remainder of this work, SRL will refer to this design shown in Figure 2.3 instead of an individual shift register. In this design, s 32-bit shift registers are chained together such that their 1-bit inputs D and 1-bit outputs Q are connected. The same clock and enable signals are used for all shift registers, and come from the clock and pseudo-random number generator output shown in Figure 2.2. On every clock cycle, a new input is inserted into the shift register while the remaining values are shifted down, where the last value is shifted out entirely and becomes the output.

The other design is referred to as SRL-LFSR, in that it is nearly identical to SRL except for the fact that each row is shaped as a type of pseudo-random number generator called an *Linear-Feedback Shift Register* (LFSR). LFSR modules are essentially

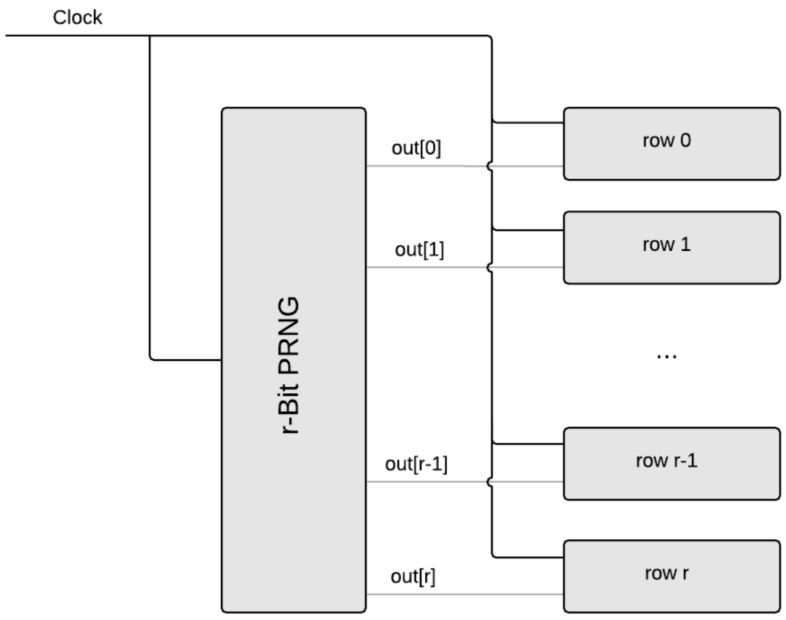


Figure 2.2. Diagram of the overall topology for VN, where a PRNG selects which rows of noisy circuits are activated at a time.

chains of registers that are connected together, where some of the inputs or outputs of the registers are XOR-ed together and used as the input for the first register in the chain. In doing so, the chain of registers receives a pseudo-random feedback, as the output looks random but is repeatable. The sequence of a LFSR depends on what the registers were initially storing and on what inputs or outputs of these registers were chosen to be XOR-ed, which can be represented as a polynomial. For this work, we chose to XOR the outputs of the shift registers and use an appropriate length irreducible polynomial to maximize the amount of pseudo-randomness we can generate.

Using an SRL-LFSR structure will obviously generate less protection in terms of noise amplitude than simply using SRL, because the internal values inside each shift register are no longer always alternating between 0's and 1's. However, the amount of power generated by SRL-LFSR is more variable than SRL in the sense that it is not

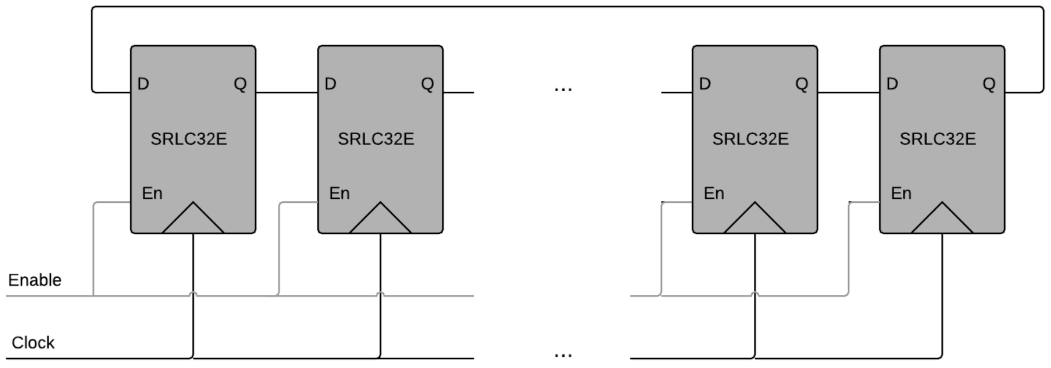


Figure 2.3. Diagram of the SRL design, which can be used as the contents for the rows of VN. Shift registers are chained together, flipping values whenever the clock and enable are active.

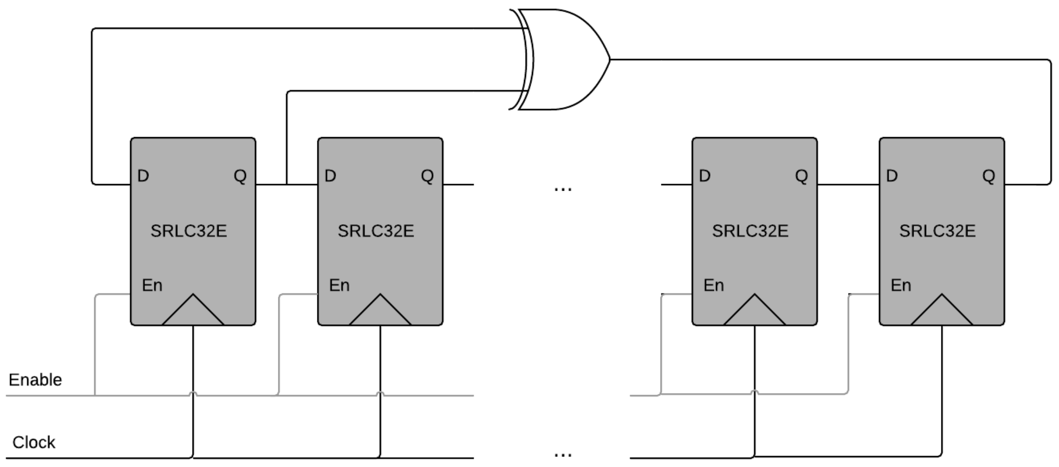


Figure 2.4. Diagram of the SRL-LFSR design, which can be used as the contents for the rows of VN. SRL-LFSR generates less power than SRL, but is more irregular and theoretically more difficult for an attacker to model.

as dependent on VN's pseudo-random number generator and that each shift register is not generating a fixed amount of power. This might make SRL-LFSR more useful

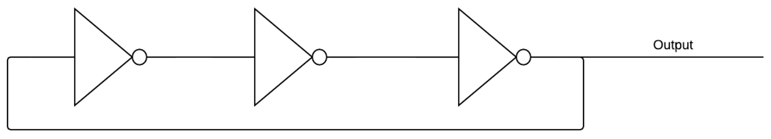


Figure 2.5. Example of a three-stage ring oscillator.

than SRL in the event that an attacker was able to predict the amount of power generated by VN and was somehow able to subtract it from a captured power trace.

By itself, VN is a very weak countermeasure because power side-channel attacks like DPA and CPA are designed to be resistant to relatively small and statistically insignificant changes in power due to noise. In order to provide significant protection by itself, a very large VN module would need to be used, which would most likely not be able to offer enough protection to justify the amount of resources it would occupy, which we discuss in further detail with our results, and is the reason why we combine VN with other countermeasures.

Shift registers are not the only circuit that can be used to generate noise. We considered using *ring oscillators*, another compact and noisy circuit, instead of using shift registers. Figure 2.5 is an example of a ring oscillator, which are simply chains of inverters. However, we felt that shift registers would be more effective than ring oscillators because shift registers change value only at the clock edge, meaning that we have more control over where noise occurs if we use shift registers instead of ring oscillators. For example, if AES and the shift registers use the same input clock, then the shift registers will certainly be able to obfuscate the power consumed by AES directly on the clock edge, ensuring that a relevant part of the power trace is obfuscated. For a ring oscillator, we would assume it to be a free-running circuit which may be more difficult to align with AES.

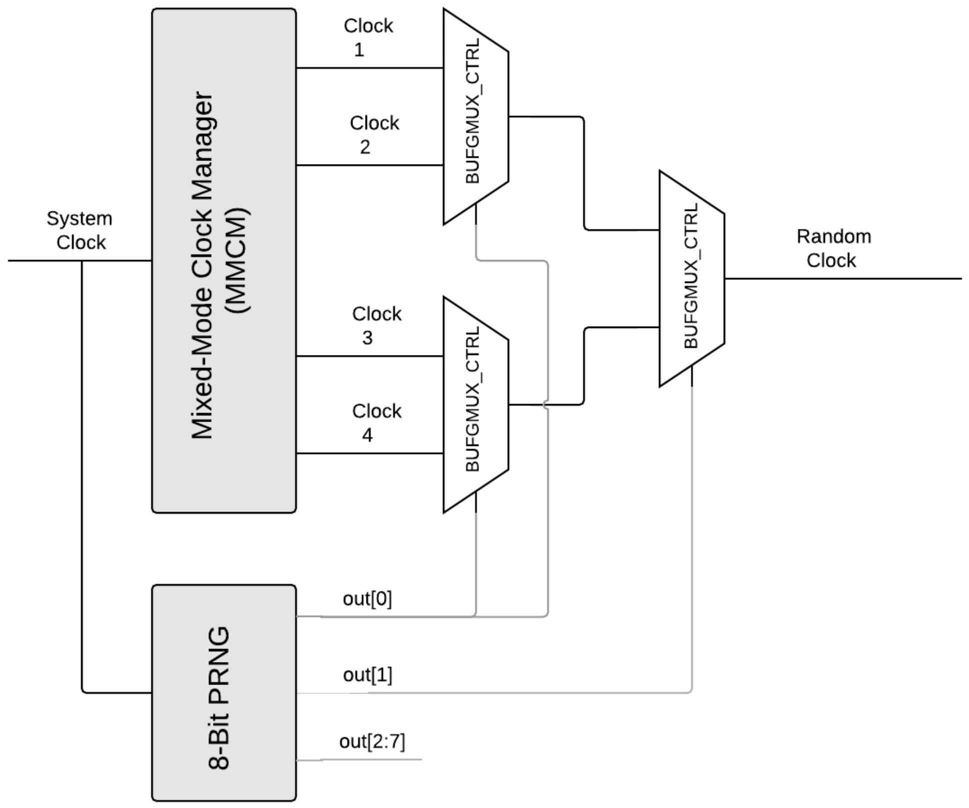


Figure 2.6. Diagram of a generic 4 clock phase CPN implementation. A MMCM module generates phase-shifted clocks. The clock multiplexers are enabled by the ouput of PRNG, which determines which clock is used as the random clock output.

2.2.3 Clock Phase Noise

The *clock phase noise* countermeasure, previously referred to as the *clock randomization* (CR) countermeasure, is module that generates a pseudo-random clock signal. This random clock output is created by randomly sampling multiple clock signals with different phases [13]. This ultimately generates a clock signal that has random delays in between pulses. When used as the input to an encryption module, this clock is used to add delays in between operations performed by the encryption module. An example of a generic implementation is shown in Figure 2.6.

In order to generate multiple clocks on an FPGA, one can use a *mixed-mode clock manager* (MMCM) or *phase-locked loop* (PLL) to generate different clock signals from a reference clock. Typically, all of the clocks generated by the MMCM or PLL have the same frequency, but have different phases. To select which clock should be the output of the CR module at any given time, special clock multiplexers are used. These clock multiplexers are connected to the clocking fabric of the FPGA such that the resulting clock is still able to drive other logic, which would not be possible if a standard multiplexer was used instead. These clock multiplexers also ensure that the output is not cut off prematurely, allowing all clock pulses to completely finish before switching to another clock source, which might introduce a delay. A pseudo-random number generator is used to select between the different clock pulses, where the output of the PRNG is used as the enable bits for the clock multiplexers.

CPN makes it more difficult for an attacker to find where the encryption is occurring in the trace when used as the input clock of an encryption module by introducing random delays to the clock being used by the encryption module. CPN has proven to be significantly more effective than VN in previous works, however it is not an acceptable countermeasure by itself. This is because CPN has been found to be highly vulnerable to preprocessing techniques such as the previously mentioned Sliding Window Attack, which work to "correct" the jitter and delay introduced into the clock before an attack is run on the collected traces. Of course, in order for an encryption to work properly, whatever clock used by the encryption module should also be used to feed inputs and sample outputs from the encryption module as well, or else the encryption module will not work properly.

2.3 Software Countermeasures

We came across two interesting examples of software-based power side channel countermeasures. One was called MUTE-AES, which used a dual processor architecture where two processors encrypt with AES in parallel but used complementary data in order to balance out the total power used [14]. This design required a second processor, effectively doubling the amount of hardware needed. When no encryption was being performed, the two processors executed tasks independently. When encryption was needed, both processors were used in conjunction, where one processor was computing the desired encryption and the other processor was being fed instructions using an algorithmic balancing technique, which balanced out the power consumption. This work showed that DPA was ineffective on this countermeasure.

A similar work made this countermeasure possible on a single core, instead of requiring two separate processors as in MUTE-AES. This was achieved by doubling the width of a typical AES module in order to run the original and complementary AES data at the same time on the same core [15]. This work also used an algorithmic balancing technique to control the complementary AES module. This countermeasure required only a 16.6% increase in instruction memory overhead and a 15.8% increase in data memory overhead. It was also able to render DPA ineffective.

Both of these software-based implementations are invasive but in a different manner than WDDL, as they require the encryption module's operations to be restructured, creating an opportunity to produce a flawed encryption module. We are also more interested in preventing these leaks from occurring at the hardware level, rather than using software to patch these issues. For these reasons, we didd not use software countermeasures in this work.

CHAPTER 3
METHOD

3.1 Hardware

To capture live power traces, we used two boards specially designed for testing power side-channel attacks from NewAE Technology [16]. The ChipWhisperer CW305 board, as shown in Figure 3.1, is the target board. The target board contains an Artix-7 *field-programmable gate array* (FPGA) and is what we used to test AES and our countermeasures. FPGAs can be reprogrammed to represent different hardware designs, allowing us to test many designs on real physical devices very quickly. The target board has a shunt resistor placed in between the wire connecting the power supply to the FPGA's voltage pin. A shunt resistor is simply a resistor with a very low resistance. Knowing the resistance value of the shunt in Ohms and the voltage drop across the resistor, we can measure the power consumption. The target board also includes an operational amplifier with a gain of 20 dB to increase the value of the power consumption so that it is readable by the capture board.

The ChipWhisperer-Lite CW1173, shown in Figure 3.2, is the capture board. The capture board is used to process the power traces from the target board. For our work, we connected probes from across the 20 dB operational amplifier on the target board to the capture board. The capture board has a 55 dB low-noise amplifier as well as an *Analog to Digital* (ADC) converter, which is used to convert the analog power trace into discrete values. The discrete output of the capture board is then stored in files on the connected computer, which are later used by the ChipWhisperer software.

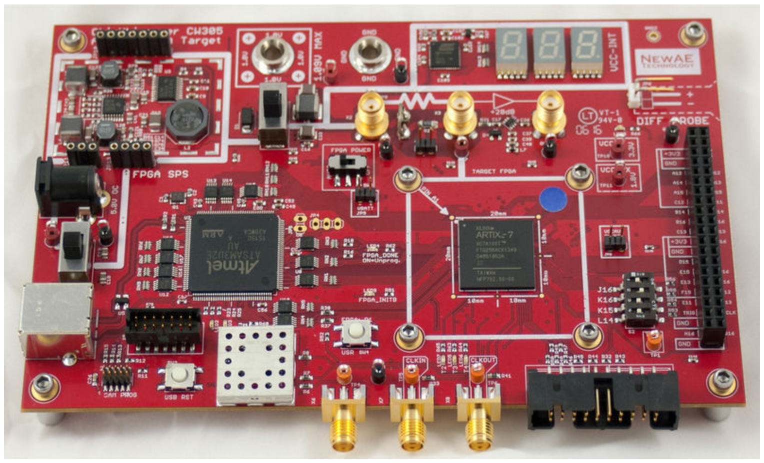


Figure 3.1. ChipWhisperer CW305 Target Board with Artix-7 FPGA.

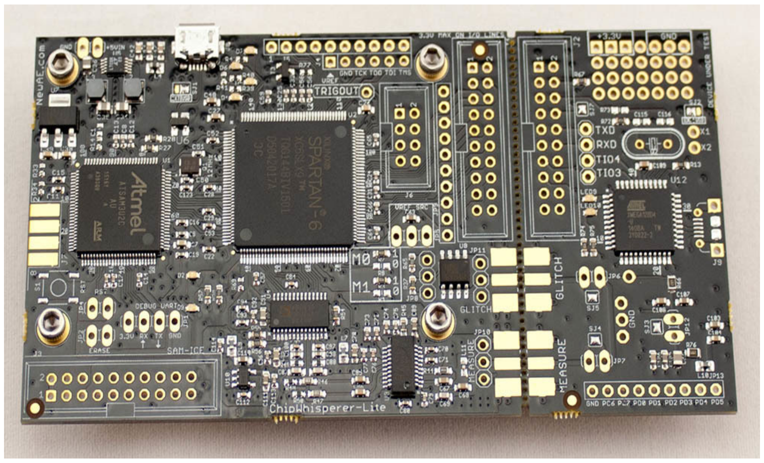


Figure 3.2. ChipWhisperer-Lite CW1173 Capture Board.

3.2 Software

We used Xilinx Vivado Design Suite for synthesizing and testing our designs, as well as for generating the final bitstreams used to program the FPGA device. All of our circuit designs were described in Verilog, which is a *hardware description language* (HDL). Vivado provides report files describing board area utilization, estimates on timing and power consumption, and more. NewAE Technology Inc., the company that sells the target and capture boards used in this work, also have software available to assist in conducting the attacks on their boards. More detail on how we used this software to configure our experiments is provided below.

3.2.1 Pre-Proposal Setup using ChipWhisperer 4

Our experiments were originally conducted using version 4.0.2 of ChipWhisperer's software. Version 4 and all previous software versions were developed using Python version 2. ChipWhisperer 4 consisted of two different programs, Capture and Analyze, each requiring the user to interact with a different graphical user interface (GUI). In Capture, we were able to create a new ChipWhisperer project and set the clock source and desired frequency, sample rate, and upload the bitstream to the target board. The captured power traces were saved in files within the project. In Analyzer, we were able to customize the attack scripts and plot results. Despite the ability to run Python scripts inside of each GUI, there was limited functionality exposed to the user, so many tasks could not be automated. In our experiments using ChipWhisperer 4, we set the clock frequency of all of our countermeasures to 100 Hz, sampling at a rate of 400 Hz.

3.2.2 Post-Proposal Setup using ChipWhisperer 5

We upgraded to version 5 of the ChipWhisperer software upon its release. ChipWhisperer 5 is built off of Python version 3 and is integrated with Jupyter Notebooks. Instead of having separate GUI environments for both Capture and Analyzer, the user is able to write their own custom functionality for both into a single Jupyter Notebook project. Because of this, version 5 is easier to customize, allowing the user the ability to more freely write scripts for modifying settings, algorithms, plots, and more. For example, we were able to write scripts to generate and save plots and result files automatically, run bash scripts to execute different experiments consecutively and automatically, and write custom code to re-use plaintext files for multiple experiments. The first two changes were implemented for convenience and saving time, as these experiments can take hours to days to execute and test, while the last change was implemented to reduce variability in our results. All of these changes were much more difficult to implement on ChipWhisperer 4 to the point that we were not able to implement them at all, since every change required modifying ChipWhisperer 4 source code. This is because ChipWhisperer 5 has an API available to expose more functionality of the user, and the removal of the GUI makes directly manipulating scripts more straightforward.

In addition to these changes, we experimented with different clock settings and found that we were able to collect better power traces when running the boards at 19.2 MHz and sampling at 76.8 MHz. The improvements gained by reducing the frequency included cleaner data, as in that it is easier to observe all of the individual rounds of an encryption across all countermeasures, and reducing the number of failures when capturing power traces. The ease of observing the encryption rounds is mostly relevant to attacks using CPN. Failure to capture power traces happens periodically in countermeasures containing CPN, as the changing clock makes it more likely that the two boards will not sync up the sending and receiving of the target signal, causing a power trace to not be captured properly.

3.3 Threat Model

We assume a very ideal situation for the attacker, as our experiments are performed in an ideal setup where it is easy to collect data from our device. We assume that the attacker has unlimited physical access to the target board in order to capture power traces from it, as we capture however many traces necessary to obtain the encryption key, which can take hours or days to obtain. In order to perform our experiments, we use a shunt resistor, probes, amplifiers, and an ADC converter. An attacker would need to have the knowledge and resources to obtain these components and set them up properly to create a comparable setup. However, none of the equipment required to capture or analyze power traces is very sophisticated or expensive, so an attacker could truly be anyone ranging from an individual actor as well as a more capable group or organization.

In a very critical system, it would be unlikely that an attacker would have this level of access to the device. For a standard home appliance or computer, however, this scenario may be perfectly feasible. In addition, we assume that the attacker either has knowledge on how to write a program to conduct a CPA attack on the power traces, or has access to already made scripts and algorithms. Such software can be executed on any standard modern computer, but may require many hours to execute. The attacker would also need to be able to interpret results of such software properly.

3.4 Measuring Protection

The effectiveness of each countermeasure is measured in terms of the resultant *partial guessing entropy* (PGE) for each byte of the secret key. The PGE is the number of incorrect key byte guesses which are ranked above the correct key byte in terms of correlation. The correlation is calculated using the equation for the correlation coefficient between the measured power consumption and a model of power consumption for key byte guess, as described in the explanation of Correlation Power Analysis in the last chapter. AES-128 has a 128-bit key, and therefore has sixteen different key bytes. Each byte can represent 256 different values. A PGE value between 0 and 255 is given to each possible value of each key byte, where 255 represents when the correct key byte is ranked last, and a PGE of 0 represents that the correct key byte is ranked as the key byte that is most likely to be correct. As more power traces are analyzed, more information is available for the algorithm to repeatedly recalculate the correlation between the key byte guesses and the observed power consumption. For a weak countermeasure, not many power traces are needed until the PGE for all key bytes reduces to zero. For a strong countermeasure, millions of power traces may need to be collected before the correct key guess is divulged. An example of the PGE values changing over time for each of the sixteen key bytes in an unprotected AES-128 module is shown in Figure 3.3, where each line on the plot represents the PGE of a different key byte. At around 800 traces, the PGE for all key bytes has reduced to zero, meaning that the secret key has been determined. As one can see, unprotected AES is solved easily with less than a thousand power traces, which means that this attack only takes a few minutes to capture and analyze using our board and software.

As previously stated, a PGE value of 0 occurs when the correct key guess has a higher amount of correlation than all of the incorrect key guesses. Another commonly used metric to measure the success of a power side-channel attack is called *measurements to disclosure* (MTD), and was briefly discussed in the previous chapter. MTD is the amount of measurements or power traces collected when the correlation coefficient of the correct key is first equal to or greater than the correlation coefficient of the highest ranked incorrect key. This means that MTD is equivalent to a PGE of zero for all key bytes. However, the difference in correlation between different key byte guesses can be very slim and can fluctuate, leading to situations where the correct key guess is ranked within the top few guesses but has not surpassed all other keys.

In this work we assume that an attacker is more sophisticated and does not need to wait until the entire key has been disclosed. Instead, we consider a successful attack to occur when all of the key bytes have a PGE of less than five, implying that the correct key byte is among the top five guesses for each key byte. This amount of remaining guesses is considered to be *brute forcible*, or susceptible to being solved using a brute force attack. The reasoning for this is that if an attacker can narrow the byte guesses to five choices for each byte of the key, the amount of possible combinations remaining is small enough such that it can be computed in a reasonable

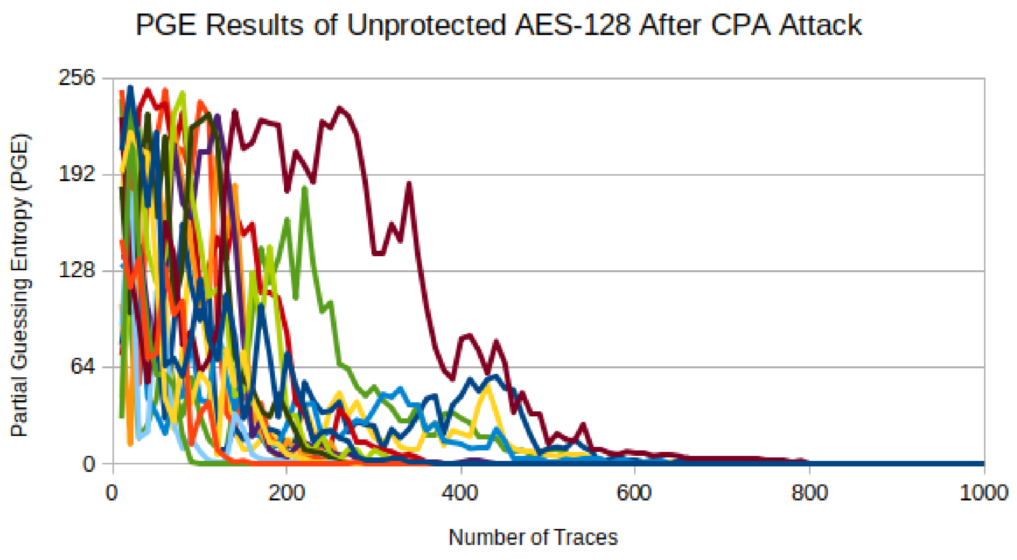


Figure 3.3. The PGE of unprotected AES-128, where each line is the PGE value for each byte of the key over the total number of power traces analyzed.

amount of time, and as such a brute force approach could be used by an attacker to determine the correct key from the remaining values [11].

3.5 Measuring Cost

Cost is measured in terms of area, power consumption, and delay. For each countermeasure, we measured these three variables and compared them to an unprotected AES-128 module. Area in terms of the number of components used on the FPGA board is provided in reports generated by Xilinx Vivado, which gives a full breakdown of how many of each type of component has been used. It also shows the exact placement and configuration of each component. Xilinx Vivado also gives an estimate of power consumption and timing. The power consumption estimate given is in terms of how many Watts of power the board will consume at any given time, which is useful for an overall estimate of power consumption. Actual values can be measured using an oscilloscope. Additionally, plots of the power traces generated from ChipWhisperer software can show changes in power over each power trace collected, giving a more granular look at the power consumed during encryption. A timing report generated by Xilinx Vivado states if timing constraints, including setup and hold time, are met. Timing also can be inferred from observing the power traces from the ChipWhisperer software, as it is possible to view each of the rounds in the power trace.

CHAPTER 4
EXPERIMENTS & RESULTS

Two different setups were used to test these experiments, as explained in the previous chapter. One was used for experiments conducted before the proposal for this project (*pre-proposal*), while the other was used for all experiments conducted afterwards (*post-proposal*). Our pre-proposal experiments involved testing Voltage Noise (VN) and Clock Phase Noise (CPN) separately and combined, and were used to obtain the results that were published in the 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP) [17]. Our post-proposal work expands on our pre-proposal work by performing different experiments with VN, and testing CPN and a combined countermeasure, but by using slightly different software and settings. In the following sections, we describe our pre- and post-proposal work for all countermeasures.

4.1 Voltage Noise

4.1.1 Pre-Proposal

Our pre-proposal results first looked at Voltage Noise as an individual countermeasure to protect AES, with a topology as shown in Figure 4.1. The overall design of VN is shown in Figure 2.2, which shows a PRNG generating outputs to control rows of noisy circuits. Two different row types of noisy circuits were discussed, SRL and SRL-LFSR. SRL simply uses chains of shift register LUTs as a noise generator, while SRL-LFSR is identical to SRL except that it mixes up the internal values of

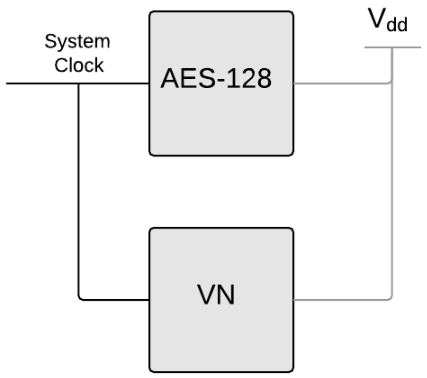


Figure 4.1. Overall topology of VN as an individual countermeasure to protect AES from power side-channel attacks, where the same system clock is used as input to both VN and AES.

the shift registers in order to make the power output less predictable. These are both depicted in Figures 2.3 and 2.4 respectively.

For all of our experiments, we used a 32-bit LFSR as our PRNG module. The number of shift register in each row was fixed at 32, meaning that s was always set to 32 and therefore the amplitude of the noise for all experiments was fixed. We changed the variance of the noise by using four different amounts of r rows. Specifically, we tested using 8, 16, 24, and 32 rows, which is equivalent to 256, 512, 768, and 1024 individual shift register LUTs. Each shift register is 32-bits wide and is initialized with alternating bits of 1's and 0's.

We then used the CPA attack against the SRL and SRL-LFSR implementations of VN five separate times, each using a different plaintext but the same key each time. Because each plaintext was completely random, each trial produced a slightly different PGE result. Our results are the average of these five trials, and are shown in Table 4.1 along with the power consumption and amount of LUTs and registers used. Notice that adding VN does increase protection by a few hundred traces compared to unprotected AES, as shown in Figures 4.2 and 4.3. Additionally, the amount of protection does increase slightly as more shift registers are used. However, this

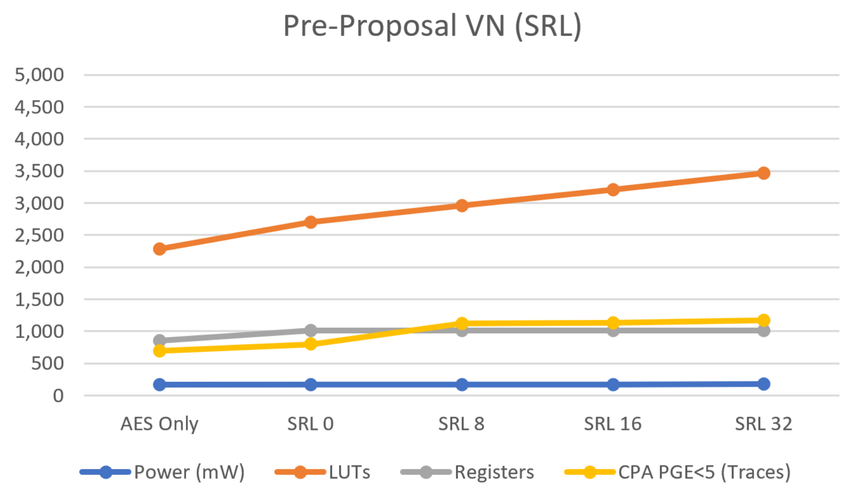


Figure 4.2. Results of our pre-proposal experiments on VN, specifically SRL.

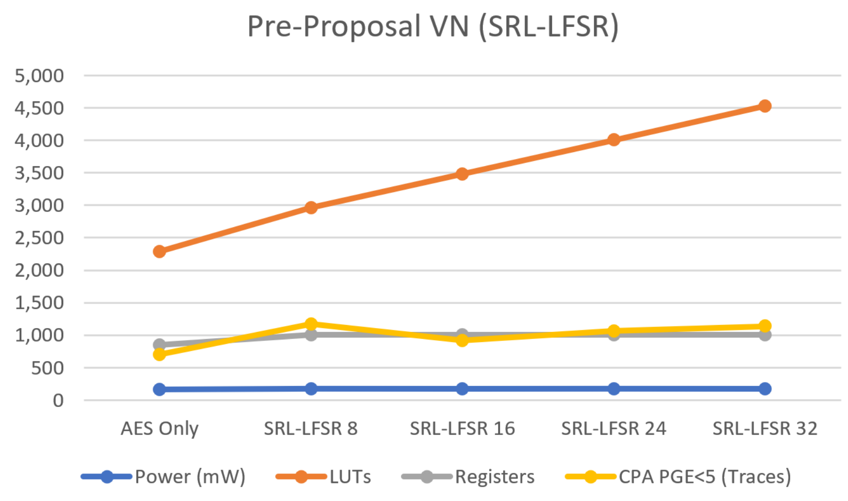


Figure 4.3. Results of our pre-proposal experiments on VN, specifically SRL-LFSR.

increase in the amount of traces that an attacker would need to collect could be obtained in a matter of minutes. For example, using our setup, we can capture and analyze 1,000 traces in about 2 minutes. It is clear from these results that neither SRL or SRL-LFSR are effective countermeasures against CPA on their own.

Experiment	Rows	Power (mW)	LUTs	Registers	PGE<5
AES Only	0	165	2,287	851	780
SRL	8	172	2,702	1,011	980
	16	172	2,958	1,011	1,124
	24	173	3,213	1,011	1,130
	32	176	3,469	1,011	1,168
SRL-LFSR	8	173	2,966	1,011	1,174
	16	176	3,484	1,011	920
	24	175	4,005	1,011	1,070
	32	176	4,525	1,011	1,144

Table 4.1. Power, area, timing, and protection measurements for pre-proposal experiments using unprotected AES, SRL, and SRL-LFSR.

4.1.2 Post-Proposal

When collecting our pre-proposal results for CPN, it was observed that the *Mixed-Mode Clock Manager* (MMCM) component used to generate clocks uses a very large amount of power. MMCM and PLL modules are known to consume large amounts of power relative to other components [18]. As described in detail later on in this chapter, the CPN module offers much more protection than the VN module. We hypothesized that the success of CPN could be due to the power-hungry MMCM module instead of the countermeasure itself. So in order to make a more fair comparison between CPN and VN, and to ensure that the success of CPN was not largely due to the MMCM module, we implemented an MMCM module in all of our post-proposal experiments using VN. The MMCM module was simply used to generate a fixed 19.2 MHz clock as shown in Figure 4.4, or multiples of this clock frequency in other experiments as shown in Figure 4.5. No phase shifted clocks were generated by the MMCM module for experiments with VN. All pre-proposal experiments used completely random plaintexts, but the same plaintext sequence was used for all of the post-proposal experiments in an effort to increase consistency among the results. For all of our post-proposal results, we only used the SRL implementation of VN.

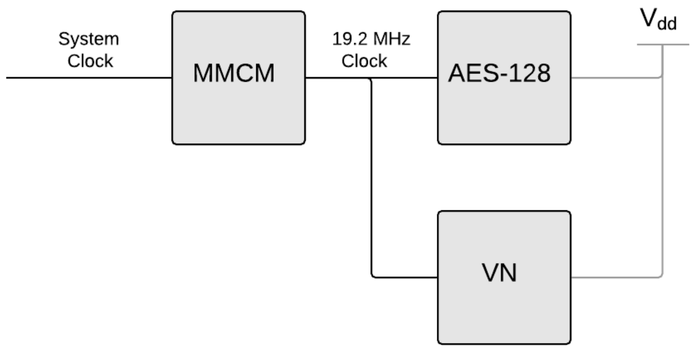


Figure 4.4. Overall topology of VN as an individual countermeasure, including a MMCM module into the design to make for a more fair comparison between VN and CPN.

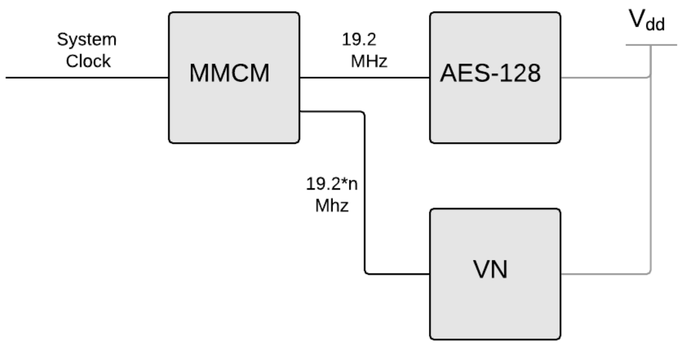


Figure 4.5. Overall topology of VN as an individual countermeasure, where the input clock for VN is a multiple of the clock used for AES.

One can compare the results between the pre- and post-proposal experiments shown in Tables 4.1 and 4.2 and see that there is a slight increase in protection for unprotected AES in the post-proposal results. In the pre-proposal results, unprotected AES reached a PGE <5 for all key bytes, or became brute forcible, at around 780 traces. The post-proposal results using the MMCM module became brute forcible at around 1300 traces. This shows that adding the MMCM module to VN does add about 1.5 times more protection. However, at such small values this increase does not offer meaningful protection, as it would take an attacker only a minute or so to collect

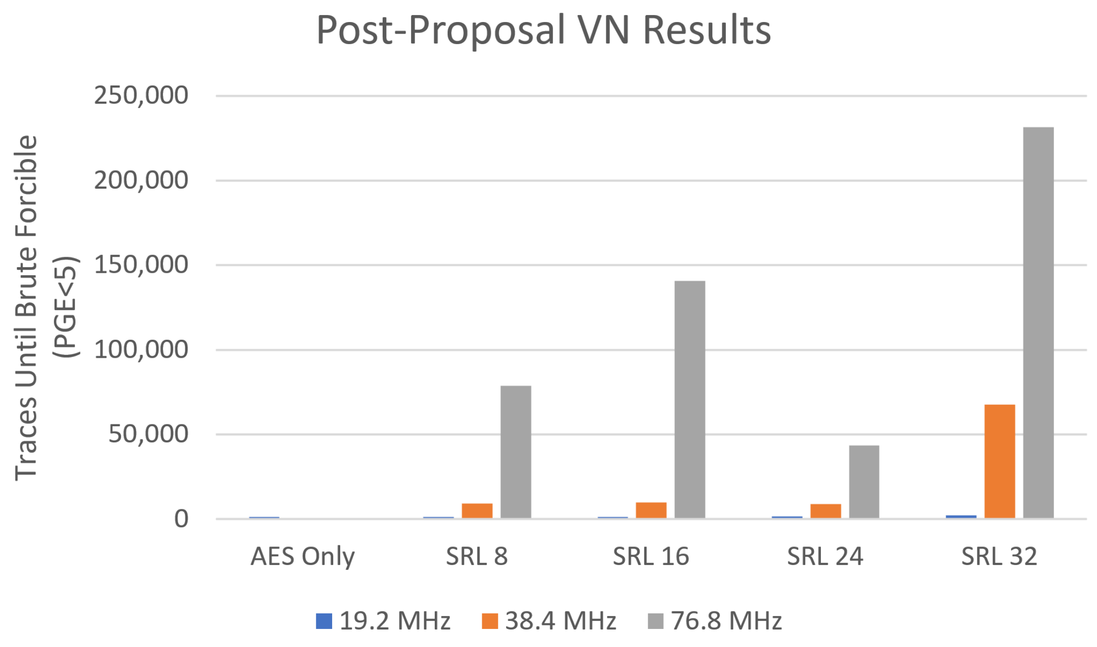


Figure 4.6. Results of our post-proposal experiments using VN, which shows that increasing the frequency used by VN significantly increases the number of traces required until it can be solved by brute force.

an additional few hundred traces. Therefore, we do not believe that the presence of the MMCM module has a significant effect on the protection offered by VN as an individual countermeasure.

The second component of out post-proposal testing of VN as a standalone countermeasure consisted of experimenting with increasing the clock frequency used by our VN module. A diagram of the topology used for these experiments is shown in Figure 4.5. Note that the clock frequency used by the AES module remains at 19.2 MHz for all experiments. In addition, only SRL was used and all designs used an MMCM module. We experimented with two different frequencies for VN, 38.4 MHz which is double the frequency used by AES, and 76.8 MHz which is quadruple the frequency used by AES.

With each increase in the frequency, a significant increase in protection was obtained, as shown in Figure 4.6. We can see that when SRL is added to AES and they are both operating at the same clock frequency, there is no significant increase in the amount of protection, which mirrors our pre-proposal results in Table 4.1. When the frequency for VN is increased from matching AES to double the frequency of AES, the protection increases anywhere from about 5.5 to 30 times compared to when the frequency for AES and VN are the same. This is substantial, but still not necessarily enough for it to be used as a standalone countermeasure. When the frequency for VN is quadrupled, it offers 27 to 103 times the amount of protection possible when AES and VN use the same frequency clock. This is much more substantial, and at the higher end offers enough protection where it may be suitable as a standalone countermeasure for systems which only need to protect up to a few hours worth of data collection. We can see from Table 4.2 that the protection offered is significant without dramatically increasing area, power consumption, or execution time in a way that would be prohibitive to implement.

Experiment	Rows	Power (mW)	LUTs	Registers	PGE<5	Time (µs)
AES Only	0	238	2,473	979	1,300	5.85
SRL @ 19.2 MHz	8	239	2,730	1,011	1,275	6.38
	16	239	2,985	1,011	1,325	6.38
	24	239	3,240	1,011	1,600	6.38
	32	240	3,498	1,011	2,250	6.38
SRL @ 38.4 MHz	8	219	2,729	1,011	9,225	6.38
	16	220	2,985	1,011	9,750	6.38
	24	220	3,241	1,011	8,900	6.38
	32	221	3,497	1,011	67,575	6.38
SRL @ 76.8 MHz	8	246	2,729	1,011	78,700	6.38
	16	247	2,986	1,011	140,725	6.38
	24	248	3,241	1,011	43,425	6.38
	32	249	3,497	1,011	231,600	6.38

Table 4.2. Power, area, timing, and protection measurements for post-proposal experiments of unprotected AES and increased clock frequency SRL.

Another way to look at the impact of increasing the clock frequency of VN as an individual countermeasure is to directly observe power traces. Figure 4.7 shows four different power trace plots. The top left plot is of unprotected AES, while the

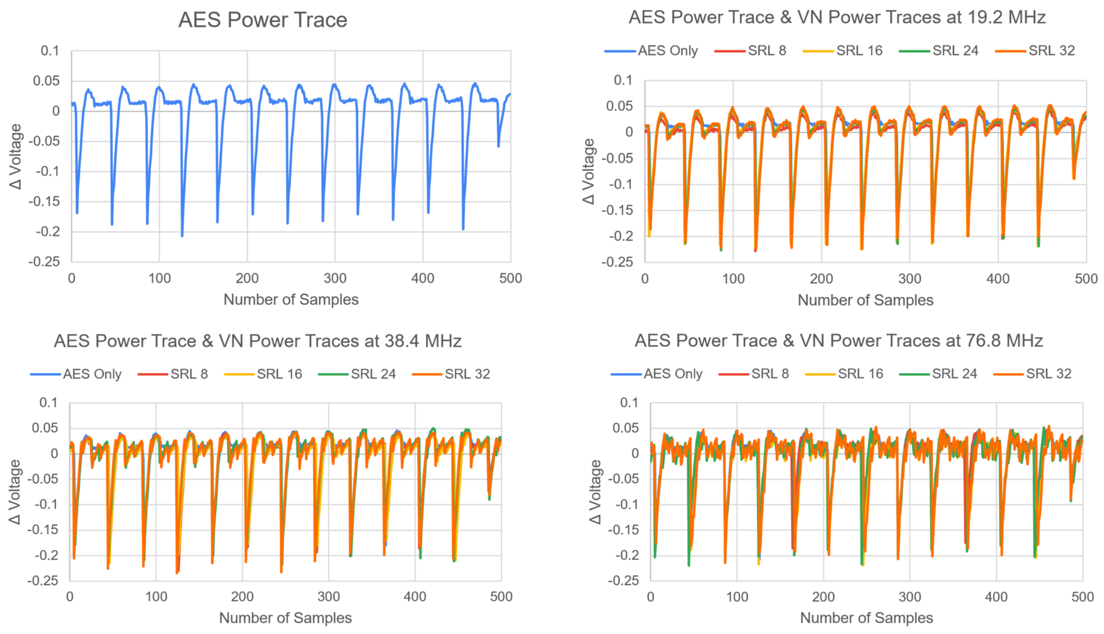


Figure 4.7. Diagram of power traces for unprotected AES and the VN countermeasure, where SRL is used and VN uses different frequency clocks. Higher frequency clocks disfigure the power trace more significantly.

other three are of unprotected AES and different versions of the increased frequency countermeasure. One can see that as the frequency used by VN is increased, it becomes much more difficult to identify the original unprotected AES trace, or even characteristics of the AES trace. This would plausibly make it more difficult for CPA to be performed successfully.

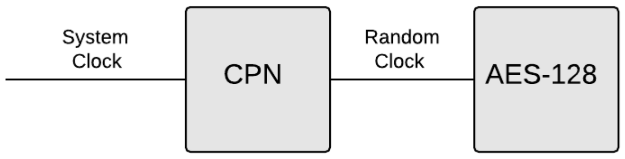


Figure 4.8. Overall topology of CPN as an individual countermeasure, where CPN generates a random clock based on the system clock to be used as the clock for AES.

4.2 Clock Phase Noise

4.2.1 Pre-Proposal

The overall topology for CPN as an individual countermeasure is shown in Figure 4.8, which shows CPN generating a random clock from the system clock, which is then used as the input clock for AES. As part of our pre-proposal experiments, three CPN modules were developed and tested, each implementing a different number of clock phases. The design for the four-phase implementation of these experiments is shown in Figure 4.9. For CPN designs with fewer than four phases, fewer clocks are generated from the MMCM module and fewer clock buffers are required. In this design, the system clock is the first clock that is being used as part of the random clock output, and is running at 100 MHz. We experimented with using one, two, and three additional clocks generated from the MMCM module, corresponding to 2, 3, and 4 total clock phases. Each of these clocks ran at 100 MHz as well. The PRNG used for all CPN experiments is an 8-bit LFSR.

The two-phase implementation selected between two clocks, the first being the system clock and the other being a phase-shifted clock from the MMCM module, which was shifted 90°from the system clock. Only one clock multiplexer and therefore one bit from the PRNG was required to generate the random clock output. The three-phase implementation used the system clock and two phase-shifted clocks generated from the MMCM module at 90°and 180°each. Only two clock multiplexers and two

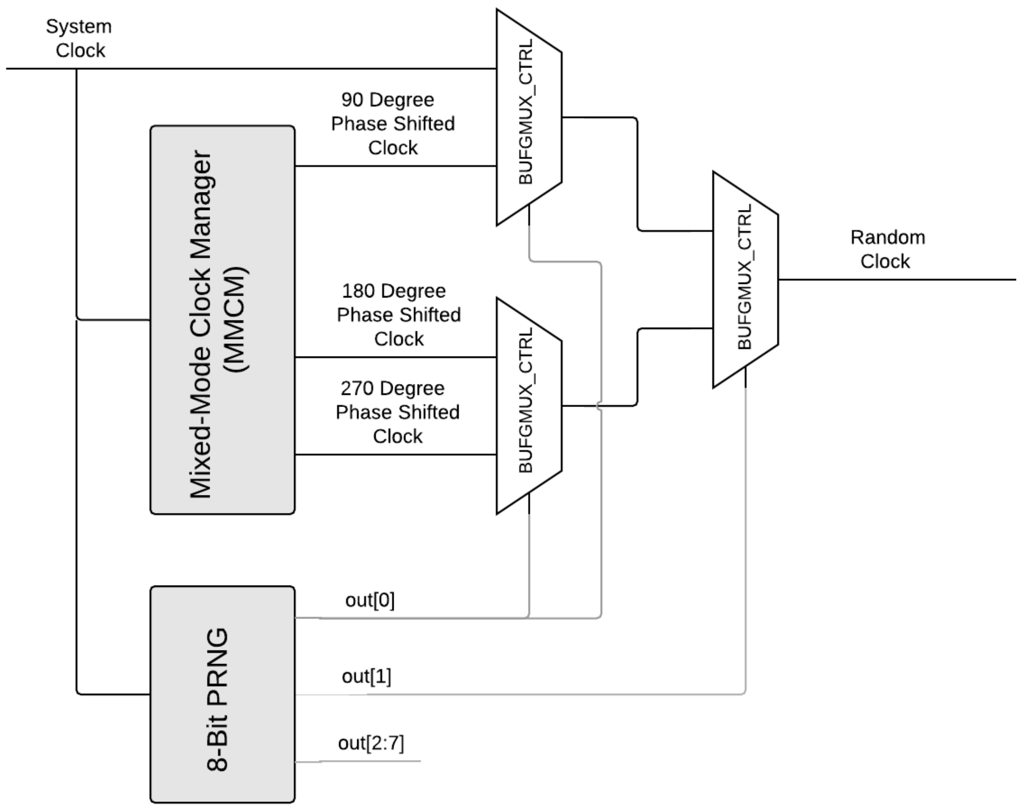

System Clock
90 Degree Phase Shifted Clock
Mixed-Mode Clock Manager (MMCM)
180 Degree Phase Shifted Clock
270 Degree Phase Shifted Clock
BUFGMUX_CTRL
BUFGMUX_CTRL
BUFGMUX_CTRL
Random Clock
8-Bit PRNG
out[0]
out[1]
out[2:7]

Figure 4.9. Diagram showing the pre-proposal implementation of CPN, where the system clock is used along with phase-shifted clocks generated from an MMCM module to produce a random clock output.

bits from the PRNG were required. The four-phase implementation used an additional 270°clock and an additional clock buffer, as shown in Figure 4.9.

Each of these CPN implementations generated a random clock, which was used as the main clock input for AES-128 and tested against CPA. The results are described in Figure 4.10 and Table 4.3. One trial was performed for each clock phase, with all encryptions being performed using using the same 128-bit key and different random plaintexts. Our results show that with each additional clock phase, CPN offers substantially greater protection. This is due to the increased variation in possible clock

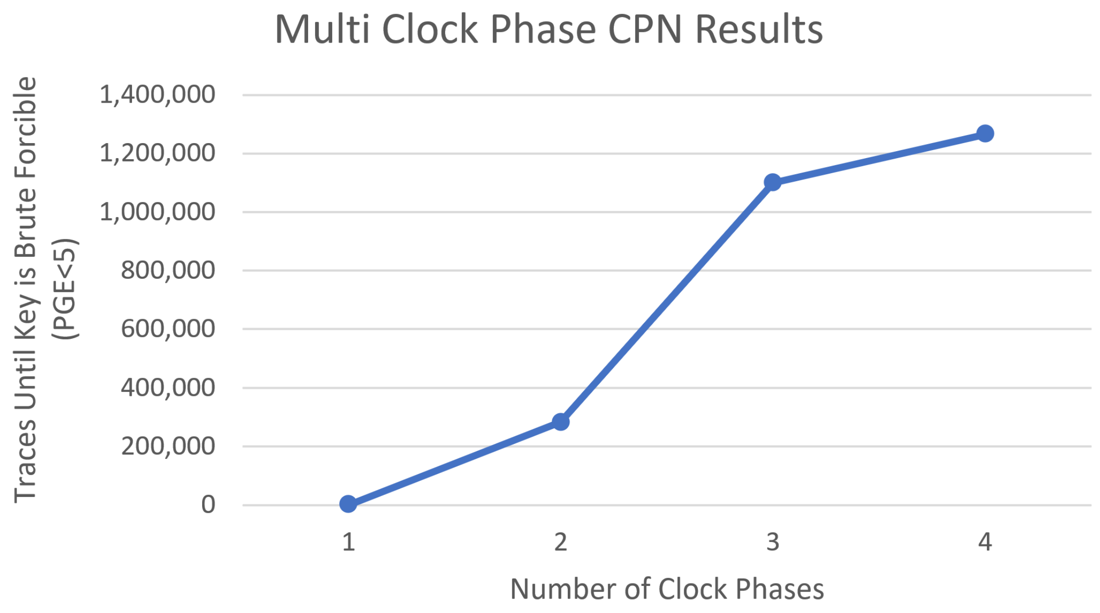


Figure 4.10. Results of CPN countermeasure with AES, which shows that increasing the number of clock phases in CPN increases the number of traces needed until the key is brute forcible.

Number of Phases	CPA PGE<5
1-Phase (AES Only)	786
2-Phases	283,280
3-Phases	1,098,840
4-Phases	1,265,400

Table 4.3. Pre-proposal results for different CPN implementations.

delay lengths and combinations. Note that the implementation corresponding to only one clock phase is simply plain AES-128.

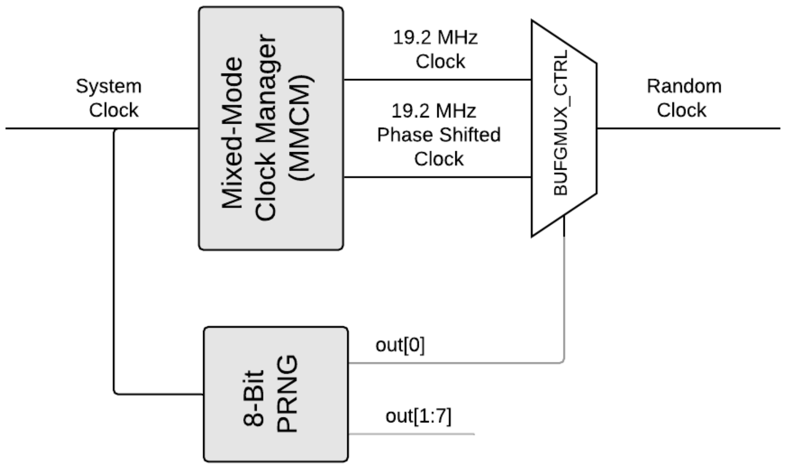


Figure 4.11. Diagram showing the post-proposal implementation of CPN, where two 19.2 MHz clocks are generated from the MMCM module to produce a random clock output.

4.2.2 Post-Proposal

In our post-proposal experiments, we tested CPN as shown in Figure 4.11, where two 19.2 MHz clocks are generated from the MMCM module. One of these clocks is phase shifted 90°from the other, mimicking the two-phase example from the first phase of experiments. We limited our post-proposal experiments to only the two-phase implementation due to time restrictions, and because our pre-proposal experiments already proved that protection drastically increases as more clock phases are included to create the random clock output. We also only performed the SW-CPA attack instead of both the CPA and SW-CPA attacks.

We performed three different trials of our experiment and averaged the results, as shown in Table 4.4. Each trial produced a significantly different result, with Trial 1 producing the maximum of about 1.3 million traces, Trial 2 producing the minimum of 330 thousand traces, and with an average of about 766 thousand traces until brute forcible. All results are significantly higher than our pre-proposal results, in which

Trial	SW-CPA PGE<5
1	1,309,600
2	330,825
3	659,025
Average	766,483

Table 4.4. Post-proposal results for CPN over three trials with an average.

	Power (mW)	LUTs	Registers	SW-CPA	Time (µs)
Pre CPN 2-Phase	402	2,453	986	50,960	2.8
Post CPN 2-Phase	218	2,476	986	766,483	11.4

Table 4.5. Comparing pre- and post-proposal results for the CPN countermeasure. SW-CPA is measured in traces until PGE<5, or it is brute forcible.

the 2-phase results after CPA were about 283 thousand traces and the results after SW-CPA were about 50 thousand traces until brute forcible, as shown in Tables 4.5 and 4.6. SW-CPA denotes when the sliding window attack was performed on the collected power traces before executing CPA. This difference can only be attributed to the slight design change of only using clocks generated from the MMCM module instead of incorporating the system clock. It could be that using the system clock in our pre-proposal design did not offer much of a benefit, making our pre-proposal design less effective than our post-proposal design. A less likely cause could be some discrepancy between the different software or configuration of such software used for the pre- and post-proposal results.

We can observe the effect of the CPN countermeasure by directly observing power traces. Figure 4.12 shows a power trace of unprotected AES and an example of CPN. One can clearly see in unprotected AES that each round completes in the same amount of time. However, in the power trace of CPN, some rounds take almost twice as long to execute compared to unprotected AES due to the clock randomization. One can see how this delays AES in time, as unprotected AES finishes around 500 samples while AES with CPN finishes at around 550 traces. One can also see that

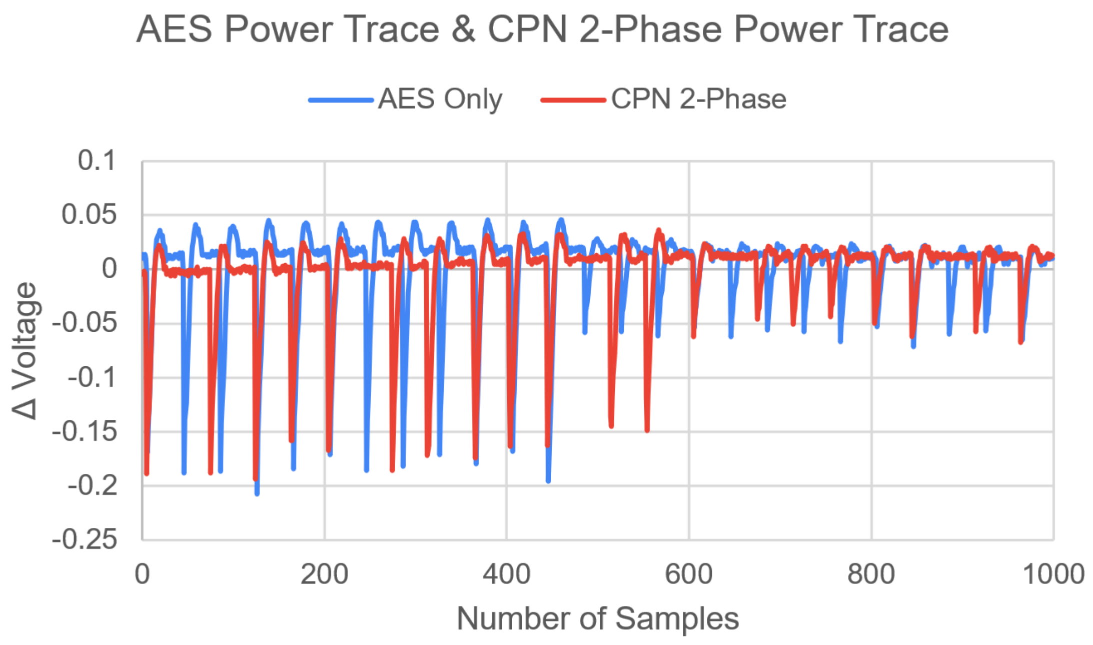


Figure 4.12. Diagram of power traces for unprotected AES and the CPN countermeasure, where SRL is used and VN uses different frequency clocks. Higher frequency clocks disfigure the power trace more significantly.

after the delay, the following clock pulses are indeed phase shifted and no longer aligned with unprotected AES. In this example, only two clock cycles are extended. But the worst case timing scenario, every clock cycle would be delayed, causing AES with CPN to take about twice as long to execute as unprotected AES.

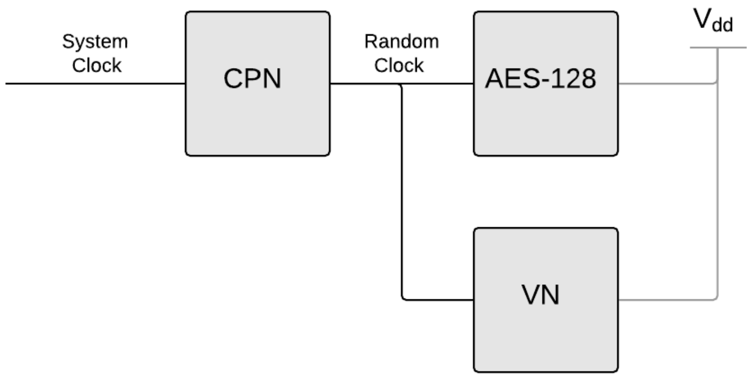


Figure 4.13. Overall topology of the combined countermeasure, where CPN generates a random clock based on the system clock to be used as the clock for AES and VN.

4.3 Combined Experiments

4.3.1 Pre-Proposal

In our pre-proposal experiments, we tested one combined experiment as shown in Figure 4.13, where CPN generates a random clock based on the system clock. This random clock is used as the input for both AES and VN. Note that in all of the combined experiments, we only tested VN modules with 16 rows and CPN with two clock phases as to limit the total number of experiments performed. Figure 4.14 shows the amount of protection offered by CPN alone and CPN combined with the two different VN countermeasures against CPA and SW-CPA. Again, SW-CPA denotes when the sliding window attack was performed on the collected power traces before executing CPA. The first pair of bars show that CPN by itself can withstand about 283 thousand traces until the key is brute forcible when attacked using CPA, but this protection is reduced by about 83% to about 50 thousand traces if the sliding window attack is performed before CPA. This confirms that the SW preprocessing attack is very effective against CPN.

The next two pairs of bars show the combined results, first using SRL as the VN implementation and the second with the SRL-LFSR implementation. There is

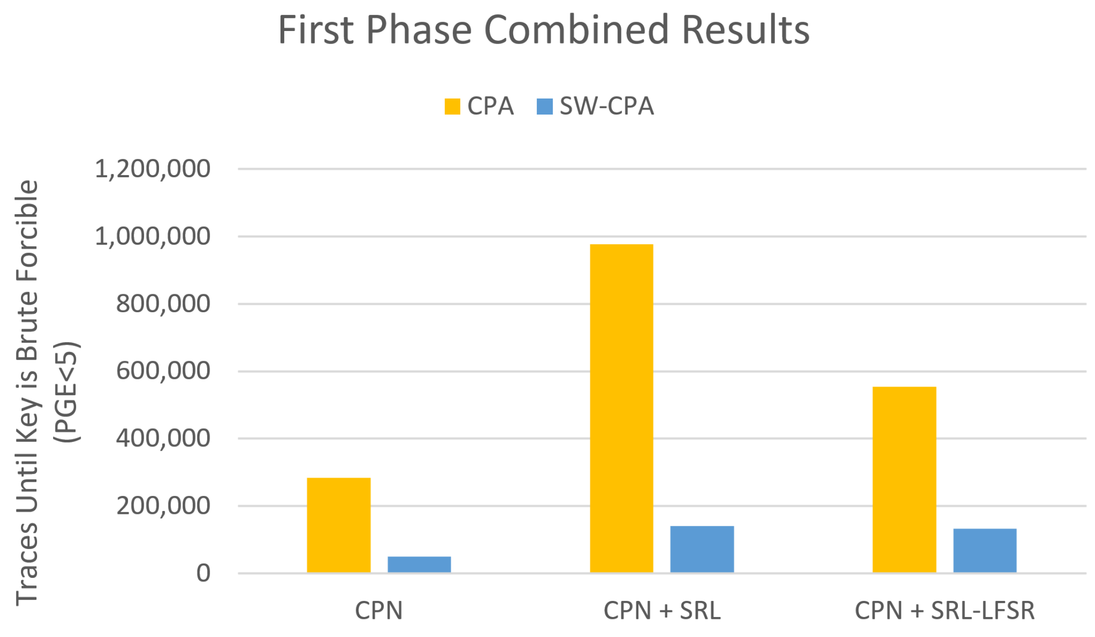


Figure 4.14. Pre-proposal results of CPN and our combined countermeasure, showing that the combined countermeasure offers a substantial increase in protection

a dramatic increase in protection against CPA for both combinations. The overall protection from CPA increases by about 245% to 975 thousand traces until the key is brute forcible for SRL, and 95% to 550,000 traces for SRL-LFSR. This is surprising because both SRL and SRL-LFSR offer almost negligible protection by themselves, as shown in Figures 4.2 and 4.3. The combined experiment's protection is also reduced significantly by SW, as the protection is reduced by about 85% for SRL and by about 76% for SRL-LFSR. The SW-CPA results for CPN with SRL are quite similar to those of CPN alone, implying that Sw is equally effective against both countermeasures. SW is about 10% less vulnerable to SW based on these results, which is a slight improvement but is clearly not enough to remove the vulnerability.

Table 4.6 shows a summary of all of our results from our pre-proposal experiments in terms of protection and resources required, as well as a comparison to the WDDL countermeasure. This graph includes protection, area, and power results for unpro-

	Power (mW)	LUTs	DPA	CPA	SW-CPA
AES Only	165	2,287		780	
SRL	172	2,958		1,124	
SRL-LFSR	176	3,484		920	
CPN	402	2,453		283,280	50,960
CPN + SRL	383	2,811		977,230	141,360
CPN + SRL-LFSR	377	3,484		553,800	133,130
WDDL	486	6,861	1,276,186		

Table 4.6. Pre-proposal summary of all results, where each VN countermeasure is the 16 row version and CPN uses only 2-phases. CPA and SW-CPA are in terms of PGE<5, or traces until brute forcible.

tected AES-128, the individual countermeasures, the combined countermeasures, and WDDL [12]. WDDL was implemented on an ASIC and quantified area in terms of millimeters squared, where all of our results are from an FPGA and are in terms of the amount of LUTs and Registers utilized. In order to make a fair comparison, we looked at the total area required for WDDL versus AES in their results, and noticed that WDDL used about 3 times as much area as the original unmodified AES module. Therefore, we took the number of LUTs and Registers used by our implementation of AES and multiplied it by 3 to get a rough estimation of the area we would expect WDDL to require compared to our experiments. The total power consumption for WDDL was already listed in Watts.

WDDL also did not use CPA as their attack algorithm, but rather used DPA and used measurements to disclosure (MTD) as their way of measuring protection. WDDL performed their experiments multiple times, obtaining minimum, mean, and maximum values for MTD. The minimum MTD for WDDL is 21,185 traces, the mean MTD is 255,391 traces, and the maximum MTD is 1,276,186 traces. We did not repeat our experiments multiple times for our pre-proposal combined countermeasures, and therefore do not have such a range available. Therefore, we consider our results to be the maximum amount of protection, and therefore use the maximum amount of traces until disclosure that WDDL listed for the protection offered by WDDL. Compared to our metric of PGE<5 for all key bytes, measurements to disclosure is roughly equivalent to PGE=0 for all key byes. Obviously, PGE<5 is easier to obtain than PGE=0, and therefore the maximum amount of traces offered by WDDL would be at least somewhat less than its current value if the PGE<5 metric was used.

Table 4.6 shows that WDDL offers more protection than our most effective countermeasure, but also requires significantly more area and power. WDDL protects up to 1.3 times more traces than our best combined countermeasure, but as previously discussed, this value is likely higher than it would be if the same metric was used for measuring protection. WDDL uses 3 times more area and 2.9 times more power than AES, compared to our best combined countermeasure which only uses 1.4 times the area and about 2.3 times more power than AES. It could be argued that the increase in protection offered from WDDL is not great enough, considering the increase in area and power, to be used instead of our best countermeasure. However, WDDL has not been tested against the sliding window attack or CPA. Since WDDL is not a clock based countermeasure, we assume that it would be significantly more resilient to the sliding window attack than our countermeasures.

4.3.2 Post-Proposal

In our post-proposal results, we essentially repeated the same experiment as in our pre-proposal work. However, we only tested against SW-CPA instead of both CPA and SW-CPA, as SW-CPA the more effective. We repeated our experiment three times and averaged the results, as shown in Table 4.7. Similarly to out post-proposal CPN results, each trial produced a significantly different result. Trial 3 produced the maximum result of about 1.7 million traces, Trial 2 produced the minimum result of 530 thousand traces, and the average was about 1.1 million traces until brute forcible. These results were about 17% larger than our pre-proposal result of about 977 thousand traces until brute forcible, which is significant, but close enough where

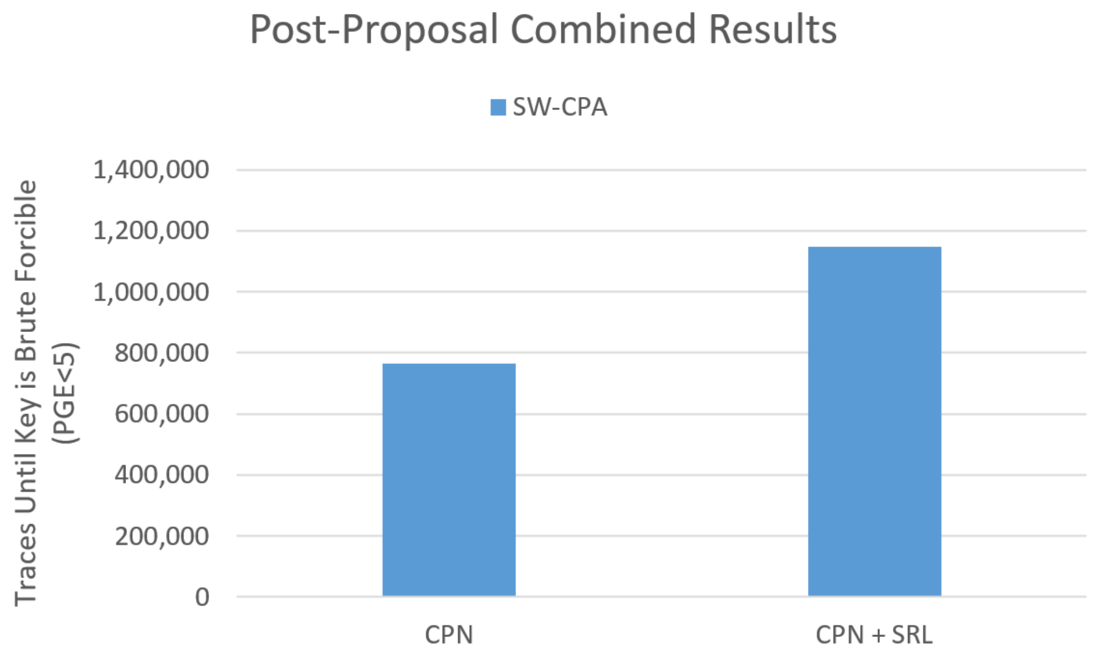


Figure 4.15. Post-proposal averaged results of CPN and our combined countermeasure, again showing that the combined countermeasure offers an increase in protection.

Trial	SW-CPA PGE<5
1	1,184,425
2	530,875
3	1,730,350
Average	1,148,550

Table 4.7. Post-proposal results for the combined countermeasure over three trials with an average.

we could expect this difference to decrease if more trials were conducted on both our pre- and post-proposal experiments. The average of our CPN and combined results are shown in Figure 4.15 in order to visually compare them to our pre-proposal results in Figure 4.14.

Table 4.8 is a summary of all of our post-proposal results. We have included timing measurements in our post-proposal results, which were obtained by visually measuring the worst case amount of samples required to capture AES and multiplying by the

	Power (mW)	LUTs	DPA	CPA	SW-CPA	Time (µs)
AES Only	238	2,473		1,300		5.85
SRL	239	2,985		1,325		6.38
CPN	218	2,476			766,483	11.45
CPN + SRL	291	2,986			1,148,550	11.45
WDDL	486	6,861	1,276,186			

Table 4.8. Post-proposal summary of all results, where each VN countermeasure is the 16 row version and CPN uses only 2-phases. CPA and SW-CPA are in terms of traces until brute forcible.

period for each sample. For WDDL, only the operating frequency was provided, so we were unable to calculate worst case completion time. These results show that our combined countermeasure is even more compelling than in our pre-proposal results. Our post-proposal results for SW-CPA mirror our pre-proposal results under just CPA. Our combined countermeasure uses about 43% of the area in LUTs and 59% of the power used by MMCM, but on average protects almost 90% of the maximum traces that MMCM can protect against. We attribute this increase to the increase in protection offered by our post-proposal implementation of CPN.

4.4 Key Dependency

Another area of interest in this research was to determine whether or not our results were significantly dependent on certain key values. In other words, we wanted to make sure that certain keys did not have characteristics that always led to more or less successful attacks. For all of our experiments discussed previously, we use the same key for consistency, so it would be good to know if certain keys could be used to produce different results. To test this, we automated the data capture and analysis of unprotected AES. Each trial used a different key throughout, but the same sequence of plaintexts as all other trials.

For each trial, we recorded the key used and the number of traces until brute forcible with CPA. We then calculated the Hamming weight of each byte of the key, since our attack relies on Hamming weight values of potential key guesses. If there was a dependency between the results and certain keys, we would expect keys from results that required few traces to brute force to have a different distribution of bytes with certain Hamming weights than keys that required more traces to brute force. For example, all results that were easiest to brute force would have significantly more bytes with low Hamming weights than results that were more difficult to brute force. However, this is just an example as there is no specific pattern we are searching for, just the presence of any pattern.

First, we tested keys that were random numbers and edge cases, such as all 0's. We tested 799 keys, with about 50 of them being edge cases. Our results are shown in Figure 4.16, which on the left has a histogram of the number of traces that became brute forcible at different numbers of traces. On the right, it has a color coded plot of the Hamming weight of the key used for different trials, sorted by trials that required the least to most traces to brute force. The average number of traces needed until brute forcible is 669, with a minimum of 350, maximum of 1400, and standard deviation of 146.

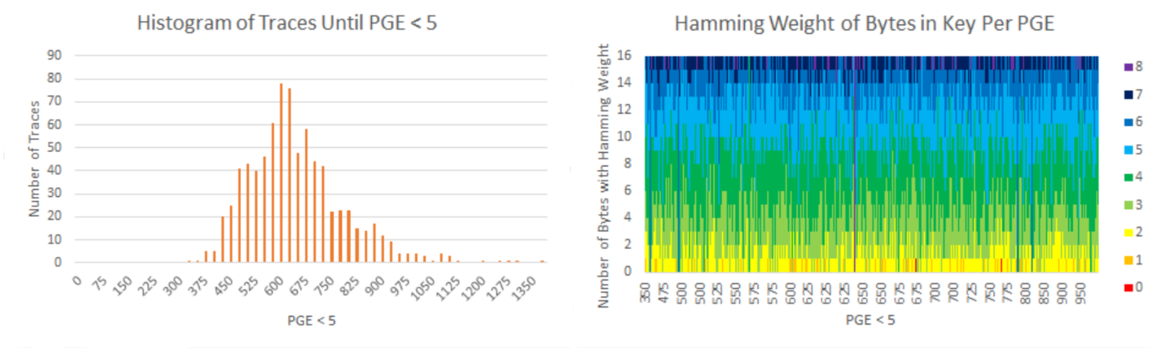


Figure 4.16. Histogram and plot of Hamming weight of keys for 799 random and edge case key values.

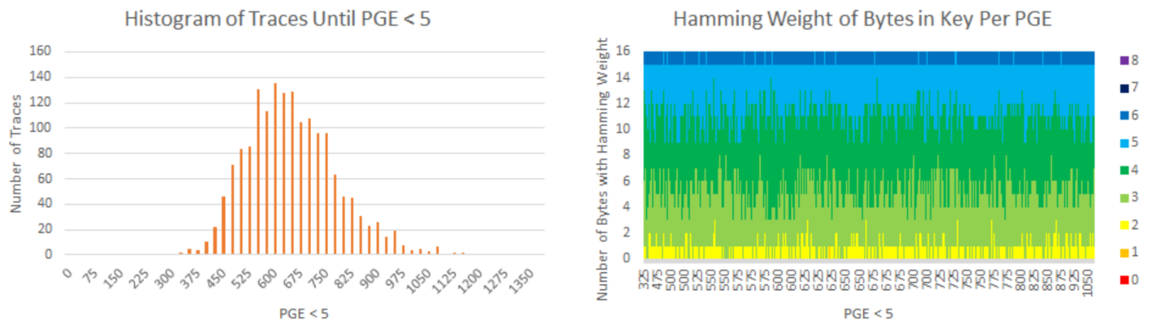


Figure 4.17. Histogram and plot of Hamming weight of keys for 1672 key values.

In the right plot, each vertical slice represents the Hamming weight for the key used to produce a given set of results, with easier to brute force keys on the left and more difficult to brute force keys on the right. Each key is 128 bits or 16 bytes. The y-axis of this plot simply breaks up the key into 16 bytes, where the color between y=0 and y=1 represents the hamming weight of one byte, the color between y=1 and y=2 represents the hamming weight of another byte, and so on. The bytes are not sorted by index, but by Hamming weight; byte 0 of the key can be placed anywhere in the plot, but the byte of the key with the lowest Hamming weight is always placed closer to the bottom of the graph. Vertical lines of one solid color are specific edge cases that tested keys with the same Hamming weight throughout, such as a key of all 0's or f's in hex.

We can see from examining this plot that there is no obvious dependence between certain keys and the amount of traces until their trial became brute forcible. If there were, we would see some sort of pattern, whereas the keys seem to be somewhat evenly distributed. For example, if having many bytes with low Hamming weights always led to fewer traces required until brute forcible, all of the leftmost keys would have many bytes with low Hamming weight (ie many yellow, orange, and red lines) which would decrease almost linearly as we progress more towards the right side. However, this is just not the case, nor does there seem to be any trend (linear or otherwise) with any of the colors.

Lastly, we tested 1672 keys produced by an AES key generator. Unlike the random numbers and edge cases used in the previous test, all of these keys have a generally uniform distribution because they are meant to be keys. For example, none of these keys are all 0's or 1's or any other value. In fact, none of the bytes have Hamming weights below 2 or above 6 at all. In Figure 4.17, we can see that the left plot of the histogram is very similar to the one from Figure 4.16. The average number of traces needed until brute forcible is 675, with a minimum of 325, maximum of 1225, and standard deviation of 134. All of these values are slightly less extreme than our previous results. We again do not observe any pattern in the rightmost graph, linear or otherwise. With a total population of 2^{128} keys and a sample size of 1672 keys, we know our results are accurate with 95% confidence plus or minus about 2.4%. It seems that there is no relationship between having more key bytes with a certain Hamming weight and offering more or less protection from CPA.

CHAPTER 5

CURRICULUM DEVELOPMENT & EDUCATIONAL RESULTS

While working towards the completion of this thesis, a new course for undergraduate students was developed titled *ECE 317: Introduction to Security Engineering.* It is a junior level class that introduces security concepts, FPGA usage, and hardware description languages and reinforces skills in programming in Python and C. We developed the majority of the laboratory experiments that the students performed, all of which focused on applying different security topics to real situations. Teams were broken up into groups of two or three students, with about 70 students in the class in total. We also held office hours in the laboratory for students to receive help with the experiments, and held sessions for the students to demonstrate their final projects. Typical teaching assistant work such as grading and responding to students via email and an online forum was also done. The other teaching assistant for this course was Shayan Moini, who also helped with many of the tasks required to create and maintain this course.

There were five laboratory experiments in this course. The first course involved performing a buffer overflow in a C program. The second composed of writing C code to capture an image from a camera attached to the board, and then encrypting and decrypting the image in two different modes of AES, one being cipher-block chaining and the other being electronic code book. The third experiment involved modifying Python code to encrypt and decrypt a static image using RSA. The fourth experiment had the student use Verilog for their first time to implement a linear feedback shift register, which was then used to generate an initialization vector for AES in cipher block chain mode used in experiment two. The last project explored power side-channel attacks, where the students were provided with traces captured from a board running AES and performed and analyzed different attacks on the trace data. Out of these projects, we designed all but the third project.

The last laboratory project was heavily related to this research, where the power trace samples the students worked with were directly captured from the boards that we use. They specifically were given 5,000 power traces worth of encryptions from an unprotected AES-128 module. The students downloaded a copy of the ChipWhisperer 4 software and used the default attacks, an attack for DES and another attack for CPA on AES, to first gain an idea of what these attacks did. The attack for a DES encryption module was unsuccessful, as the students were using traces from AES, and therefore the attack was not relevant to the actual power trace. The other default attack for an AES encryption module using CPA was also unsuccessful, as that specific attack looked at the SBox output, which is more difficult to break using CPA on hardware and would require more than 5,000 traces to break.

They were then asked to modify the default CPA attack script to attack a more fruitful location, specifically the last round state which we use for this research, and which should have been successful at less than 1,000 traces. The students had to examine the provided scripts, source code, and online examples to figure out what needed to be changed in the attack script. They then had to figure out how to add noise and clock jitter to the traces, which can be done in preprocessing using ChipWhisperer. Each team of students had to add a different amount of noise or jitter to their experiments depending on the value of their student ID's. This led to some groups having both attacks be successful, as a high amount of noise was added, while other groups did not. The students were required to write a lab report explaining what they did, what their final Python scripts were, and plots of their results. They also demonstrated their results to us and answered simple questions about the lab. For example, some of these questions were *"What do noise and jitter do to the attacks?"* and *"What would you need to do to solve an attack that was unsuccessful?"*. Most students received full credit for the demonstration portion of the lab, and the overall average grade for this assignment was an A.

Formal feedback from students reported all 4's and 5's out of 5 total for all questions asked about the quality of the course. When asked about the instructor's teaching of the course, students responded quite positively, but also made some comments about some small bugs in the assignments that we did not find until the students started working on them. As this was the first version of this course, we ran into a few hiccups along the way. We particularly found that students needed more clarification on the instructions than expected, that there were small typos in some of the materials, and that we ran into new bugs that we did not expect when testing the experiments ourselves. Overall, we were able to resolve this issues in a timely manner. For every laboratory assignment, students received a small amount of credit for explaining what they did and did not like about the experiment, issues that they encountered, how work was divided with their group, and how many hours were spent on the project. This information will be used to update the course materials for next year.

CHAPTER 6

CONCLUSIONS & FUTURE WORK

The following sections discuss thoughts on each experiment tested and their results, as well as suggestions for future experiments and other areas of improvement.

6.1 Experimental Methodology

We were able to successfully use Xilinx Vivado, ChipWhisperer 4, and ChipWhisperer 5 for our research. We developed several Python scripts for automating tests, reusing plaintexts, processing our results, and more. However, there is room for expansion. One improvement that we considered implementing into our experiments was to better ”lock in” the location of certain components of our designs into certain regions of our FPGA, to keep our experiments more consistent. This would prevent some components from being optimized out of the design or placed in different locations, which could potentially skew the results of our experiments. For example, if one design had a component optimized away unknowingly, that design may generate slightly less power and offer slightly less protection than expected. Changes in placement and routing may lead to similar effects.

We experimented with using *incremental compile* in Vivado as a way to ensure that designs that were similar would keep some similar placement and routing. For example, we used the same Xilinx project for different VN experiments, but changed which type or number of noisy circuits that was used each time we generated a new bitstream. However, the changes were too dramatic for the incremental compile tool to work properly, as the increase in size from the different sizes of VN modules were probably more than what the tool was intended for, and therefore generated designs with many timing violations. A second method we tried was to export the base AES design as an IP block and import it into all of the designs such that the AES configuration would be identical in all designs. However, we were unable to get Vivado to properly import our IP blocks, and so this was abandoned.

One aspect of conducting this research is that capturing and analyzing interesting data can take a very long time. When possible, we used multiple ChipWhisperer boards on multiple computers in order to increase throughput and capture as much data at a time as possible. We would recommend having multiple boards for anyone conducting similar research. The main bottleneck for capturing power traces is the board itself, while the main bottleneck for analyzing traces is processing power. For future experiments, it may be helpful to use the newly released LASCAR integration with ChipWhisperer to speed up analysis, or to offload the analysis onto the cloud or a dedicated server.

6.2 Voltage Noise

We were able to thoroughly examine Voltage Noise as an independent countermeasure between the different pre- and post-proposal experiments that we conducted. We could have continued to test implementations of VN using different numbers of rows, shift registers per row, or sizes of individual shift registers. However, it was apparent that these differences were very insignificant and only showed more useful insights when combined with other countermeasures. Our pre- and post-proposal results both showed that VN is an ineffective countermeasure by itself, only increasing protection by a negligible few hundred traces. The only potential exception was found in the post-proposal experiments where the clock frequency of VN was increased, in which we found that as much as 231 thousand traces were required until the secret key was brute forcible. This is still not satisfactory for a critical application, but is a dramatic improvement over our previous results. We would anticipate that running VN at even higher frequencies than we tested could potentially provide enough protection for VN to be used as a standalone countermeasure in some situations. In addition, it would be useful to formally compare VN to ring oscillators to test if our intuition that shift registers offer more protection than ring oscillators is true.

6.3 Clock Phase Noise

Our pre-proposal results showed that CPN can offer substantial protection on its own, protecting more than a million traces if three or more clock phases were used. However, this protection was reduced by about 83% in the face of SW-CPA. In our post-proposal work, we slightly changed out design such that the system clock was no longer used as an output of the CPN module, but instead all clocks were generated from an MMCM module. We found that protection dramatically increased, requiring about 766 thousand traces on average until brute forcible after SW-CPA with only two clock phases. We attribute this dramatic increase to our design change, but it could have also been impacted by our changes to our experimental methodology.

We would have liked to test more than four clock phases in total for both our pre- and post-proposal results to see if the relationship between protection and clock phases constantly increases, or if it converges to some maximum value. For our post-proposal results, we only tested the two-phase countermeasure. We were able to repeat our post-proposal experiments three times to gain an understanding of how the amount of protection offered by CPN can vary. We obtained minimum and maximum estimates of protection offered by CPN. However, these results could have been further improved by conducting more trials. Other modifications, such as changing value of the phase offset to values other than 90°increments may also have been interesting, especially if it was found that certain phase offset combinations would be resistant to any window size of the sliding window attack.

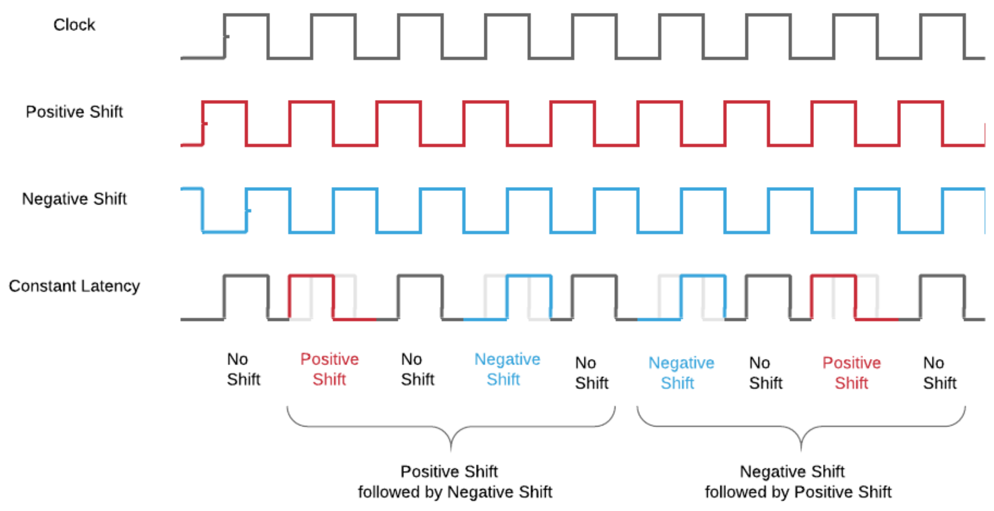


Figure 6.1. Diagram of Constant Latency design for CPN, where the resultant clock ensures that one AES encryption will always complete in the same amount of time regardless of how many times the clock is switched.

Another idea to be explored is a version of CPN that always takes the same amount of time to encrypt a plaintext regardless of what clock sequence was used to generate the random clock. This would have been referred to as Constant Latency, as shown in Figure 6.1 and would be implemented by enforcing that a certain clock was used immediately after one that was picked by the pseudo random number generator. For example, if we used three clocks of the same frequency but with one clock phase shifted -90°and another phase shifted +90°, we could theoretically always enforce that the positive shifted clock would be used for the next clock cycle if the negative clock was selected and vice versa. However, preventing timing violations is a significant concern for this implementation, as this design inherently would reduce the amount of time in between rising clock edges in some cases. Our current CPN countermeasure only increases time between rising clock edges because the clock multiplexers simply wait to select a new clock until the next rising clock edge if a period would be cut short, preventing timing issues.

6.4 Combined Experiments

In both our pre- and post-proposal combined experiments, we made a compelling case for our combined countermeasures. In our pre-proposal work, our best combined countermeasure protected about 76% of the maximum amount of traces that WDDL could with only about 41% of the area and 78% of the power required for WDDL. However, the sliding window attack significantly reduced the amount of protection our combined countermeasure could offer to only 11% of that offered by WDDL. In our post-proposal results, our CPN countermeasure performed remarkably better, and as follows so did our combined countermeasure. Our combined countermeasure on average protected up to about 90% of the maximum amount of traces that WDDL could with only about 43% of the area and about 60% of the power required for WDDL. This is remarkable because these results are after the sliding window attack, meaning that our post-proposal countermeasures protect almost as well as WDDL while requiring only about half of the resources.

Other combined countermeasures to be examined include using different clocks for AES and VN, where only VN or AES uses the clock generated from CPN and not both. It would be interesting to see if there would indeed be less protection, as we would expect, if only one of these modules used the random clock while the other used a fixed clock. Other potential combined countermeasures include using the increased frequency for VN and the output of CPN for AES. Since the increased frequency version of VN was so successful, we would expect this combination to add a very significant increase in protection compared to our existing countermeasure with very little overhead.

BIBLIOGRAPHY

[1] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," in *Automation and Test in Europe Conference and Exhibition Proceedings Design*, vol. 1, Feb. 2004, pp. 246–251 Vol.1.

[2] F. Koeune and F.-X. Standaert, "A tutorial on physical security and side-channel attacks," in *Foundations of Security Analysis and Design III: FOSAD 2004/2005 Tutorial Lectures.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 78–108. [Online]. Available: https://doi.org/10.1007/11554578

[3] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology — CRYPTO '96*, N. Koblitz, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 104–113.

[4] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO' 99*, ser. Lecture Notes in Computer Science, M. Wiener, Ed. Springer Berlin Heidelberg, 1999, pp. 388–397.

[5] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, and J. F. Dray Jr, "Announcing the Advanced Encryption Standard (AES)," Nov. 2001. [Online]. Available: https://www.nist.gov/publications/advanced-encryption-standard-aes

[6] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, Apr 2011. [Online]. Available: https://doi.org/10.1007/s13389-011-0006-y

[7] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *Cryptographic Hardware and Embedded Systems - CHES 2004*, ser. Lecture Notes in Computer Science, M. Joye and J.-J. Quisquater, Eds. Springer Berlin Heidelberg, 2004, pp. 16–29.

[8] T.-H. Le, J. Clédière, C. Canovas, B. Robisson, C. Servière, and J.-L. Lacoume, "A Proposition for Correlation Power Analysis Enhancement," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, ser. Lecture Notes in Computer Science, L. Goubin and M. Matsui, Eds. Springer Berlin Heidelberg, 2006, pp. 174–186.

[9] S. Mangard, E. Oswald, and T. Popp, in *Power Analysis Attacks: Revealing the Secrets of Smart Cards.* Springer US, 2016, p. 124.

[10] P. Hodgers, N. Hanley, and M. O'Neill, "Pre-processing power traces to defeat random clocking countermeasures," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 85–88.

[11] D. Fledel and A. Wool, "Sliding-Window Correlation Attacks Against Encryption Devices with an Unstable Clock," Tech. Rep. 317, 2018. [Online]. Available: http://eprint.iacr.org/2018/317

[12] D. Hwang, K. Tiri, A. Hodjat, B.-C. Lai, S. Yang, P. Schaumont, and I. Verbauwhede, "AES-based security coprocessor IC in 0.18-µcmos with resistance to differential power analysis side-channel attacks," vol. 41, no. 4, pp. 781–792, 4 2006.

[13] T. Guneysu and A. Moradi, "Generic Side-Channel Countermeasures for Reconfigurable Devices," in *Cryptographic Hardware and Embedded Systems – CHES 2011*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 6917, pp. 33–48. [Online]. Available: http://link.springer.com/10.1007/978-3-642-23951-9_3

[14] J. Ambrose, S. Parameswaran, and A. Ignjatović, "MUTE-AES: A multiprocessor architecture to prevent power analysis based side channel attack of the AES algorithm," Nov. 2008, pp. 678–684.

[15] A. Arora, J. A. Ambrose, J. Peddersen, and S. Parameswaran, "A double-width algorithmic balancing to prevent power analysis Side Channel Attacks in AES," in *2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Aug. 2013, pp. 76–83.

[16] NewAE Technology Inc. (2019) Chipwhisperer. [Online]. Available: https://newae.com/tools/chipwhisperer/. [Online]. Available: https://newae.com/tools/chipwhisperer/

[17] J. Lagasse, C. Bartoli, and W. Burleson, "Combining clock and voltage noise countermeasures against power side-channel analysis," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160-052X, 2019, pp. 214–217.

[18] N. Dumpala, S. B. Patil, D. Holcomb, and R. Tessier, "Energy Efficient Loop Unrolling for Low-Cost FPGAs," Apr. 2017, pp. 117–120.