RSA POWER ANALYSIS OBFUSCATION: A DYNAMIC FPGA ARCHITECTURE

THESIS

John W. Barron, Captain, USAF

AFIT/GE/ENG/12-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/12-02

RSA POWER ANALYSIS OBFUSCATION: A DYNAMIC FPGA ARCHITECTURE

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

John W. Barron, B.S.E.E.

Captain, USAF

March 2012

AFIT/GE/ENG/12-02

RSA POWER ANALYSIS OBFUSCATION: A DYNAMIC FPGA ARCHITECTURE

John W. Barron, B.S.E.E.

Captain, USAF

Approved:

/signed/	22 Feb 2012
Maj Todd R. Andel, PhD (Chairman)	date
/signed/	22 Feb 2012
Lt Col Jeffrey W. Humphries, PhD (Member)	date
/signed/	22 Feb 2012
Maj Mark D. Silvius, PhD (Member)	date

AFIT/GE/ENG/12-02

Abstract

The modular exponentiation operation used in popular public key encryption schemes, such as RSA, has been the focus of many side channel analysis (SCA) attacks in recent years. Current SCA attack countermeasures are largely static. Given sufficient signal-to-noise ratio and a number of power traces, static countermeasures can be defeated, as they merely attempt to hide the power consumption of the system under attack. This research develops a dynamic countermeasure which constantly varies the timing and power consumption of each operation, making correlation between traces more difficult than for static countermeasures. By randomizing the radix of encoding for Booth multiplication and randomizing the window size in exponentiation, this research produces a SCA countermeasure capable of increasing RSA SCA attack protection.

Acknowledgements

First, I would like to thank my wife and son for their support during long hours and sometimes frustrating research. I would like to thank my thesis advisor and committee for guidance at the many crossroads in my research. I would also like to thank my fellow researchers in the lab for uncountable ways to waste time watching YouTube videos, researching tech specs, and “Sketch Lunch Fridays.” Lastly, I would like to thank the lab SNACO for keeping an almost endless supply of cold Diet Mountain Dew. Massive amounts of caffeine kept me awake in unbearable 8AM algorithms class.

John W. Barron

Table of Contents

Page

Page

Page

List of Figures

Figure Page

List of Tables

RSA POWER ANALYSIS OBFUSCATION: A DYNAMIC FPGA ARCHITECTURE

I. Introduction

In the groundbreaking 1999 paper [4], Kocher et al. presented a method to recover a secret key from cryptographic hardware by monitoring the hardware's power consumption. In recent years, these so called side channel analysis (SCA) attacks have become a focus of the cryptographic community. These attacks are conducted by collecting power consumption data of the hardware, referred to as power traces, over many cryptographic cycles, and statistically correlating this data to the likely cryptographic key. These attacks allow an attacker to recover keys much faster than traditional cryptanalysis.

The RSA public key encryption algorithm [5] has been the target of many of these attacks. Current methods of SCA protection fall into two categories: masking and hiding [6]. Masking introduces randomness into the input text such that there is no correlation with power consumption. Hiding countermeasures obscure any correlation by introducing electrical noise or designing the hardware in such a way that minimizes the signal an attacker wishes to capture, thus making successful attacks more difficult. The goal of both methods is to lower the signal to noise ratio (SNR) to a level that makes the attack infeasible.

1.1 Motivation

In today's ultra-connected society, virtually every facet of life involves the flow of information over computer networks. From financial transactions through online banking and Internet purchases, to personal information backed-up on the "cloud," the need for a secure flow of information is as important as ever. RSA encryption, as a part of SSL and TLS [7], is a major factor in that security. If a major bank or stock market exchange were to succumb to attack, untold worldwide financial havoc would result. Additionally, the United States Government authorizes the RSA algorithm for electronic transmission of classified data [8]. Thus, protecting RSA encryption from attack is key to national security.

In recent years side channel attacks have become an increasingly important design consideration for cryptographic implementations. Observing the power consumption of an RSA implementation can lend information about the operations, timing, operands, and ultimately the secret keys. With a modest investment in a digital oscilloscope and third party software, adversaries have the capability to conduct such attacks with minimal knowledge of the system under attack. This motivation drives research to develop a implementation resistant to side channel attacks.

1.2 Problem Statement

Current SCA attack countermeasures are largely static. Given enough time and a sufficient number of power traces, static countermeasures can be broken as they merely attempt to increase the difficulty of attack to an infeasible level. However, countermeasures are constantly changing, as in a dynamic architecture, timing and power consumption would be a "moving target," making correlation between traces much more difficult than static countermeasures.

1.3 Research Objectives and Contributions

The goal of this research is to develop and synthesize an architecture on an FPGA that is capable of protecting an RSA implementation from SCA attacks. This dynamic architecture contains randomizations at each level of cryptographic operations, randomized multiplier used by randomized exponentiator. The hypothesis of this research is that a dynamic circuit is capable of randomizing timing and power consumption in such a manner that SCA attacks are no more successful than brute force attacks.

This research provides a VHDL coded dynamic architecture for synthesization on a Xilinx Virtex-5 FPGA. This architecture provides two-way communication with Riscure Inspector's side channel attack software for power trace acquisition and analysis, as well as custom written MATLAB scripts implementing common side channel attacks.

1.4 Thesis Organization

This thesis is organized into five sections. Chapter 2 covers the background of RSA encryption operations, side channel attacks against RSA, and known countermeasures against side channel attacks. Chapter 3 covers the methodology used to develop the design of experiments in this research. Chapter 4 covers the design of the dynamic architecture and results from experimental SCA attacks. Lastly, Chapter 5 summarizes the work accomplished, major contributions, and recommendations for future work in the area.

II. Background

The following chapter covers the background of RSA encryption operations, side channel attacks against RSA, and known countermeasures against side channel attacks.

2.1 RSA Encryption Algorithm

In contrast to traditional symmetric key ciphers, public key encryption assigns each member a private (secret) key and public key. The RSA public key encryption scheme [5] shown in Figure 1 is based on the assumption that factorization of large integers composed solely of two prime factors (i.e., modulus n) is computationally infeasible in polynomial bounded time [9].

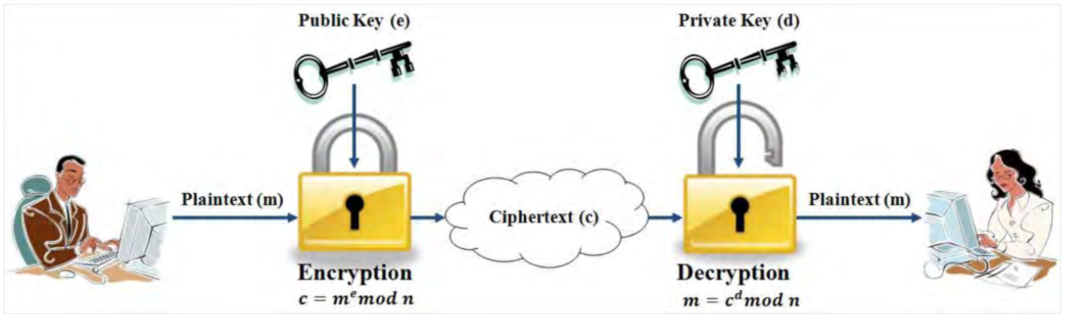

Figure 1: RSA Public Key Encryption [1]

This research focuses on the modular exponentiation operations of the cryptosystem. Algorithm 1 shows the operations for encryption and decryption. In this provable security cryptosystem, it is computationally infeasible, to a polynomial-time bound attacker, to factor n to determine the private key.

2.1.1 Exponentiation Methods.

Modern cryptographic algorithms use very large integers for keys (e.g., 4096-bit RSA). Many public key cryptographic algorithms, including RSA, rely on the modular exponentiation operation for encryption. Naively computing m^e via $e - 1$ multiplications of m is impractical because of

Algorithm 1 RSA Algorithm [10]

```
*Summary*: B encrypts a message $m$ for A, which A decrypts.

*Encryption*: B should do the following:
  Obtain A's authentic public key (n,e).
  Represent the message as an integer $m$ in the interval $[0, n-1]$
  Compute $c = m^e \pmod{n}$
  Send the cipertect $c$ to A.
*Decryption*: To recover the plaintext $m$ from $c$, A should do the following:
  Use the private key $d$ to recover $m = c^d \pmod{n}$
```

memory and timing constraints [10]. Therefore, algorithms are required to decrease storage requirements and the number of steps required for the modular exponentiation operation. These algorithms are known as square-and-multiply algorithms. The algorithms represent the exponent (i.e., key) in binary and repeatedly compute modular multiplication operations. These algorithms keep the memory space limited as they perform modular reduction during exponentiation and complete after $\log_2 d$ iterations. Additionally, exponentiation methods can operate on a single bit of the exponent (binary) or multiple bits of the exponent (k-ary) for a given iteration.

2.1.1.1 Binary Exponentiation Methods. Binary exponentiation is the most straightforward method for exponentiation; operating on a single bit of the binary representation of the exponent (key) each iteration. This process can be performed either from the left-to-right (MSB to LSB) or right-to-left (LSB to MSB). A square-and-multiply algorithm for left-to-right can be seen in Algorithm 2. Similarly, the square-and-multiply Algorithm 3 is shown for right-to-left binary exponentiation. Note that the order of squaring and multiplying is reversed since the exponent is now traversed LSB to MSB. Also note that left-to-right binary exponentiation only requires a single variable. Since many cryptographic systems are under tight memory and area constraints (e.g., smart cards), the left-to-right variant is more popular [11].

Algorithm 2 Left-to-Right Binary Exponentiation [10]

```
INPUT: base $m$ and exponent $e = (e_t, e_{t-1} ... e_1, e_0)_2$
OUTPUT: $m^e$
$R := 1;$
for $i = k - 1$ downto 0 do
  $R := R^2 \pmod N$; —Square
  if $e_i = 1$ then then
    $R := R \cdot m \pmod N$; —Multiply
  end if
end for
return $R$
```

Algorithm 3 Right-to-Left Binary Exponentiation [10]

```
INPUT: base $m$ and exponent $e = (e_t, e_{t-1} ... e_1, e_0)_2$
OUTPUT: $m^e$
$R_0 := 1; R_1 := m;$
for $i = 0$ to $k - 1$ do
  if $e_i = 1$ then then
    $R_0 := R_0 \cdot R_1 \pmod N$; —Multiply
  end if
  $R_1 := R_1^2 \pmod N$; —Square
end for
return $R_0$
```

2.1.1.2 K-ary (Windowing) Exponentiation Methods. It is also possible to operate on multiple bits of the exponent at once. For maximum efficiency, the goal is to compute m^e using the fewest number of operations, given that it is only possible to multiply two already computed powers of m [12]. Since multiple bits are being operated on at once, there are fewer total operations needed. This approach, referred to as windowing, provides some speedup. This speedup comes as trade off of higher memory requirements. As seen in Algorithm 4, precomputation and storage of additional powers of m are needed. The higher the window size k, the less multiplication operations needed and more memory space required.

Algorithm 4 Constant Window Method [10]

```
INPUT: base $m$, window size $k$, and exponent $e = (e_t, e_{t-1} ... e_1, e_0)_2$
OUTPUT: $m^e$
PRECOMPUTE:
$m_0 := 1$;
**for** $i = 1$ to $2^k - 1$ **do**
    $m_i := m_{i-1} \cdot m$; — Compute Powers For Window
**end for**
$R := 1$;
**for** $i = k - 1$ downto $0$ **do**
    $R := R^{2^k} \pmod{N}$;
    $R := R \cdot m_{e_i} \pmod{N}$; —Use Precomputed Powers
**end for**
**return** $R$
```

By maximizing the number of 0 exponent strings, it is possible to further minimize the number of operations, since 0 exponent bits do not require the additional multiplication step. The Sliding Window method can be used to further lower the number of required multiplications. The Sliding Window method is a modification of Algorithm 4. Instead of choosing a constant window size, choose a maximum window size k. Select up to k bits to operate on, with the goal being to select the longest bitstring within the maximum window size with the last bit of the string as a one. This reduces the number of multiplications and precomputations needed. These methods have been extended in [13] and [14] with randomized windows to increase obfuscation. Authors in [15], [16], explore optimal choice of high-radix Montgomery multipliers for speedup and their side channel leakages. Several classical popular methods for modular exponentiation have been shown. The next section discusses attacks that exploit side channel leakages specific to each algorithm.

2.2 Side Channel Attacks

Prior to Kocher et al's paper [4], cryptographers' primary concern was traditional cryptanalysis against the primary plaintext to ciphertext information stream. Nowadays, the security of public key encryption algorithms, including RSA, no longer rely on only the provable security of the plaintext to ciphertext conversion. Implementors of cryptographic systems must consider a collection of design and coding practices such that their implementations are not left open to side channel attacks. Side channel attacks begin by collecting side channel information such as power consumption or electromagnetic (EM) emissions [17] of the system under attack during the encryption operation, known as collecting traces. Trace collection usually requires physical access to the device via direct power consumption monitoring or via EM probes. Side channel attacks can be classified as passive or active. Passive attacks only require power traces and known messages, while active attacks require chosen plaintext or fault interjection.

2.2.1 Simple Power Analysis (SPA). SPA involves directly observing power traces to gather information about the secret key [4]. SPA leaks are caused by programming conditional branch decisions based on secret keys or intermediate values. For example, in binary exponentiation as described in Algorithm 2, if there is a distinguishable difference between a squaring operation and a multiplying operation (time or power), the implementation is vulnerable to SPA. Figure 2 shows a power trace that an attacker can easily discern the squaring and multiplying operations of RSA encryption and visually read the secret key from the power trace. To easily prevent SPA attacks, operations should be kept independent from the secret keys and intermediates [4].

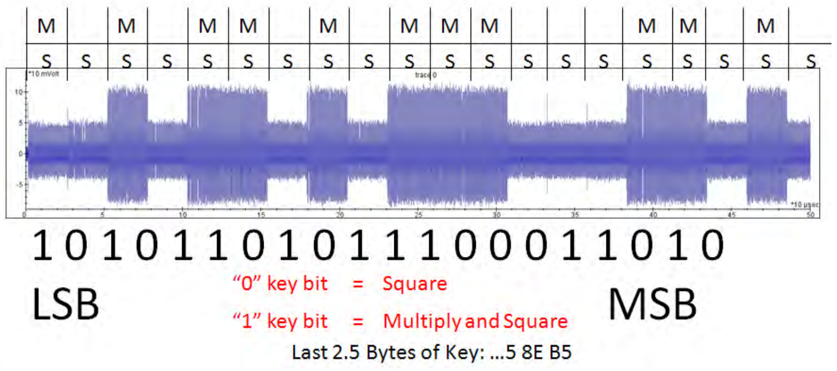

Figure 2: SPA Against Unprotected Binary Exponentiation [1]

2.2.2 Differential Power Analysis (DPA). Differential power analysis, presented by Kocher [4], is a very common and very powerful side channel attack. In contrast to SPA, where few traces are needed to directly infer secret material, DPA may require hundreds to thousands of traces to statistically correlate operands to key guesses. Since DPA attacks require little knowledge about the underlying encryption implementation, they are the most popular type of power attack [6]. Based on key guesses, an attacker calculates an expected intermediate bit, and checks whether the difference between the mean traces partitioned according to this bit differ. DPA is based on the fact that unique operands create different switching activity and power consumption.

2.2.2.1 Difference of Means. As outlined in [4], an attacker may create a key hypothesis and selection function to separate the traces into two groups. These two groups are averaged to eliminate random noise, hoping that there exists some statistical similarities in both group. Next, subtract the two groups from each other. This result leaves a differential trace, as seen in Figure 3.

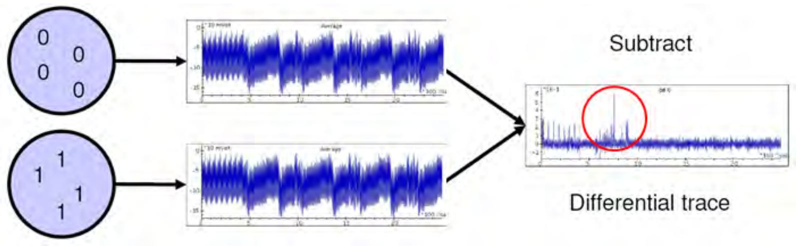

Figure 3: Difference of Means [2]

As seen in Figure 4, if the key guess was incorrect the resultant trace will be flat randomness, but if the key guess was correct the differential trace will contain spikes at points of correlation verifying the correct key guess.

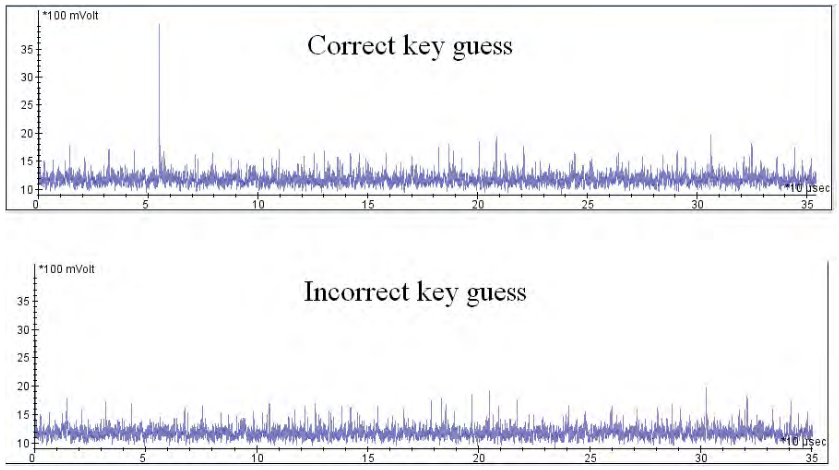

Figure 4: Differential Traces Showing Correct vs. Incorrect Key Guess

Successful DPA attack depends entirely on being able to distinguish a correlation spike from the noise floor of the differential trace. Given Equation 1 [4], it is clear that the differential trace will become less noisy as more traces are collected. This effect is because uncorrelated events diminish by a factor of $\frac{1}{\sqrt{n}}$.

$$\lim_{n \to \infty} \Delta D[j] \approx 0 \tag{1}$$

Where n is the number of traces collected and $\Delta D[j]$ is the differential trace.

2.2.2.2 Correlation Power Analysis. Correlation power analysis requires attackers to calculate an expected intermediate value and check for statistical correlation within the captured power traces. There are many variants of this attack using either the Hamming Weight (HW) of the intermediate value, Hamming Distance (HD), zero value model, etc [6]. These attacks depend heavily upon being able to calculate the correct intermediate values, seen in Equation 2 [6].

$$r_{i,j} = \frac{\sum_{d=1}^{D}(h_{d,i} - \bar{h}_i) \cdot (t_{d,j} - \bar{t}_j)}{\sqrt{\sum_{d=1}^{D}(h_{d,i} - \bar{h}_i)^2 \cdot \sum_{d=1}^{D}(t_{d,j} - \bar{t}_j)^2}} \tag{2}$$

Where r is the coefficient of correlation,

D is the number of traces,

h is the HW, $\bar{h}$ is the average HW,

t is a power trace sample, $\bar{t}$ is the average power,

i is the number of samples in j power traces.

Using this equation, the correlation coefficient is computed across all i samples within j power traces for all possible key guesses looking for a spike in correlation. Using the HW power model usually yields better results than difference of means.

2.2.2.3 Higher Order Differential Power Analysis. Data masking, as described in section 2.3.3, is an effective way to defeat DPA. However, higher order differential power analysis is able to defeat masking countermeasures. Second-order differential power analysis (SODPA) as shown in [18], [19] exploits the power consumption between two intermediate values sharing the same mask. There is no direct correlation between input message and masked intermediate value, but there is correlation between two intermediate values with the same mask. Higher order differential power analysis's effectiveness is directly related to the number of masks used and frequency of generating new masks. Since masking is computationally expensive, SOPDA is usually the highest order attack needed [6].

2.2.3 Timing Attacks. In [20] Kocher shows how observing the time required for private key operations may allow an attacker to find the secret keys. Much like power consumption in SPA, if the timing is dependent on the secret key, the time variance can be calculated and can help in identifying the correct key guess. This attack can be defeated by making all operations consume an equal amount of time, but this process is difficult due to cache misses, instruction timing, etc. Kocher suggests exponent blinding as discussed in Section 2.3.3.

2.2.4 Electromagnetic (EM) Attacks. Electromagnetic Attacks (EM) [17] exploit electromagnetic radiation that is emitted from the cryptographic system. As transistors switch and consume power, electrical current traveling along wires emit EM radiation. EM probes are capable of collecting localized power consumption. EM attacks are also able to scan and find desired frequency hot spots to avoid other system noise. Once an attacker has collected EM power traces, they complete SPA and DPA as previously outlined. These attacks are often referred to as Simple EM analysis (SEMA) and differential EM analysis (DEMA).

2.2.5 Comparative Power Analysis (CPA). The authors of [21] propose the new title of "Comparative Power Analysis" for a growing class of power attacks. These attacks are active chosen message attacks that generate intermediate collisions during cryptographic operations. By observing the power consumption over several traces during the collisions, an attacker is able to match trace patterns and infer which operation was accomplished and thus the secret key.

2.2.5.1 Doubling Attack. This attack [11] is based on the fact that encrypting messages $X \pmod N$ and $X^2 \pmod N$ has the potential to generate intermediate collisions between encryption cycles. When a collision occurs, the result is the same intermediate value i_2 loaded into registers as some other encryption operation i_1. If the power consumption and operation performed on i_1 is known, an attacker is able to tell if the operation performed on i_2 was the same or different (i.e., square or multiply). In Table 1, the grey values show collisions.

Table 1: Doubling Attack Example

i	Key	Encrypt X	Encrypt X^2
1	1	$(1)^2$	$(1)^2$
		$1 * X$	$1 * X^2$
2	0	$(X)^2$	$(X^2)^2$
3	0	$(X^2)^2$	$(X^4)^2$
4	1	$(X^4)^2$	$(X^8)^2$
		$X^8 * X$	$X^{16} * X^2$

Recall that left-to-right binary exponentiation, Algorithm 2 has an additional multiply for exponent bits of 1. Table 1 shows that if the $i - 1$ key bit is 0, a collisions between X_{i-1}^2 and X_i exists. This process can be repeated to reveal all key bits. Note that because of the difference in which order the squaring and multiplying is accomplished in binary exponentiation, this attack only works for left-to-right implementation. This attack is the first known to distinguish between left-to-right and right-to-left algorithms.

2.2.5.2 Yen's Attack. Similarly to the doubling attack, Yen's attack [22] also relies on a chosen message pair to create intermediate collisions. This attack takes the form of choosing a message pair $X \pmod{N}$ and $-X \pmod{N}$.

Table 2 shows that when the key bit i is 0 a collision can be observed at $i + 1$ for both X and $-X$ operations.

2.2.5.3 Homma et al. Attack. This attack [21] requires a chosen message pair Y and Z such that $Y^\alpha \equiv Z^\beta \pmod{p}$ to create intermediate collisions

Table 2: Yen's Attack Example

i	Key	Encrypt X	Encrypt $-X$
1	1	$(1)^2$	$(1)^2$
		$1 * X$	$1 * -X$
2	0	$(X)^2$	$(-X)^2$
3	0	$(X^2)^2$	$(X^2)^2$
4	1	$(X^4)^2$	$(X^4)^2$
		$X^8 * X$	$X^8 * -X$

at arbitrary points in time. Using modular math described in detail in [21], Homma et al. generate an example message pair $Y^{24} \equiv Z^3 \pmod{p}$

Table 3: Homma's Attack Example

i	Key	Encrypt Y	Encrypt Z
1	1	$(1)^2$	$(1)^2$
		$1 * Y$	$1 * Z$
2	1	$(Y)^2$	$(Z)^2$
		$Y^2 * Y$	$Z^2 * Z$
3	0	$(Y^3)^2$	$(Z^3)^2$
4	0	$(Y^6)^2$	$(Z^6)^2$
5	0?	$(Y^{12})^2$	$(Z^{12})^2$
6	$SorM$	$(Y^{24})^2$ or $Y^{24} * Y$	

Table 3 assumes the operation and power consumption for squaring Z at $i = 3$ is known, and an attacker seeks to compare against Y at $i = 6$. Since a collision was generated at $Y^{24} \equiv Z^3 \pmod{p}$ at $i = 3$ and 6, an attacker can discern whether the power signature at $i = 3$ matches $i = 6$ (same operation, key bit is 0) or power is different (different operation, key bit is 1). Homma et al. present example of successful attacks on binary (up and down), windowing (constant and sliding), square-and-multiply-always, Montgomery powering ladder, and other highly regular algorithms.

2.2.6 Fault Attacks. Conducting a fault attack consists of interrupting a cryptographic device (e.g., clock glitch, induced bit flips, etc.) at a specific point in time and monitoring its reaction [23]. RSA was the first cryptosystem to be targeted by fault attacks [24]. One very well known attack is the safe error [25] attack. This attack is used on countermeasures containing dummy operations, such as the square-and-multiply always algorithm. By locally inserting a fault before the multiplication operation, an attacker can check the output to determine if the fault propagated. If the fault does not propagate, the multiplication operation was a dummy operation and the key bit is 0. If the fault propagates, the result from multiplication was taken and the key bit is 1. This attack is viable for all countermeasures that include dummy operations.

2.2.7 Big Mac Attack. In [26], Walter describes an attack on sliding window exponentiation. The Big Mac Attack compares the power consumption of each multiplication operation within a trace against the power consumption of the pre-computations at the beginning of the algorithm, looking for a match. This attack is conducted within a single trace, which has inherent advantages. Exponent blinding countermeasures, as described in Section 2.3.3, are rendered useless since they create randomness between traces not within them. Also, since more cross checking is possible as the key size increases, this attack becomes more viable with larger keys.

2.3 Countermeasures

In general, side channel analysis countermeasures fall into two groups: hiding and masking [6]. Hiding methods include injecting noise, inserting dummy operations, inserting delay, shuffling order of operations, dual rail logic, etc. Masking countermeasure integrate some randomness into the data such that they are “masked” to reduce correlation with the input text. It is the goal of designers to hide the power by making it completely random or by flattening. In [27], the authors present a power sensing amplifier able to sense instantaneous power consumed and dissipate power through a shunt to maintain a constant power consumption. There have also been many modifications to the Montgomery modular multiplication for increased protection, including [28] and [29]. The remainder of this section discusses six common countermeasure approaches.

2.3.1 Square-And-Multiply-Always. As described in Section 2.2.1, when secret material determines branching operations, information is vulnerable to SPA and timing attacks. Also mentioned previously, an easy way to avoid this leakage is to ensure encryption programs are written to have a constant execution path. The simplest method to modify binary exponentiation to include a constant path is with a Square-And-Multiply-Always algorithm, such as the one seen in Algorithm 5. Notice that the operations are now independent of the secret material. While this does help prevent SPA type attacks, the dummy multiplication now make this countermeasure vulnerable to fault attacks such as safe error. These countermeasures are considered hiding countermeasures.

Algorithm 5 Square-and-Multiply Always [30]

```
INPUT: base $n$ and exponent $e = (e_t, e_{t-1} ... e_1, e_0)_2$
OUTPUT: $n^e$
$R := 1$;
for $i = k - 1$ downto 0 do
    $R_0 := R \cdot R \pmod{N}$;
    $R_1 := R_0 \cdot X \pmod{N}$;
    $R := R_{e_i}$
end for
return $R_0$
```

2.3.2 Montgomery Powering Ladder. Peter Montgomery's "Power Ladder" [31] can be used as an efficient way to speed up exponentiation multiplication. As seen in Algorithm 6, the powering ladder has the added advantages that it is regular and contains meaningful multiplication and squaring each cycle (i.e, no dummy operations). This algorithm is also a great candidate for parallel computing, as discussed and analyzed in [32].

Algorithm 6 Montgomery powering Ladder

```
INPUT: base $n$ and exponent $e = (e_t, e_{t-1}...e_1, e_0)_2$
OUTPUT: $n^e$
$R_0 := 1; R_1 := X;$
**for** $i = 0$ to $k - 1$ **do**
  **if** $e_i = 1$ then **then**
    $R_0 := R_0 \cdot R_1 \pmod N$;
    $R_1 := R_1 \cdot R_1 \pmod N$;
  **else**
    $R_1 := R_1 \cdot R_0 \pmod N$;
    $R_0 := R_0 \cdot R_0 \pmod N$;
  **end if**
**end for**
**return** $R_0$
```

Since the powering ladder is "regular" like square-and-multiply-always, it is protected from SPA, and since there are no dummy operations, the powering ladder is also immune to safe error attacks.

Similar to the Montgomery Powering Ladder, Joye presents several highly regular algorithms in [33]. These algorithms include the same security features of the Montgomery Powering Ladder, but also being right-to-left algorithms they protect against the Doubling Attack [11] and Big Mac Attack [26].

2.3.3 Masking. Masking countermeasures seek to create independence between the power consumption and true operations. To defeat simple power analysis, one's goal is to make the execution path constant and independent of the secret keys. To defeat DPA, a system must additionally create independence between the power consumption and the message being operated on. This goal can be accomplished by masking, which combines the input data or secret exponent with some randomness such that the HW of the intermediate operands will not correlate between traces. A common method of masking for modular exponentiation is to blind the exponent [6]. This step is accomplished by adding some random multiple of $\Phi(n)$ to the secret decryption exponent d. Adding a multiple of $\Phi(n)$ will change the operands but the modular operation will ensure the plaintext is correct.

$$d_{masked} = d + v \times \Phi(n) \tag{3}$$

Where v is a random value.

In [34], the authors combine blinding with exponent segmenting for increased protection. It is also possible to mask the data itself prior to operations with the secret key. For example, multiply the ciphertext c by some random value v before decrypting via exponentiation with the private key.

$$c_{masked} = v \times c \tag{4}$$

Since ciphertext c is simply $m^e \pmod{n}$, this yields:

$c_{masked} \equiv v \times m^{e \cdot d} \pmod{n} \overset{decryption}{\Rightarrow} (c_{masked})^d \equiv v^d \times m^{e \cdot d} \pmod{n} \Rightarrow v^d \times m \pmod{n}$.

An attacker can easily pull off v^d at the end of the decryption operation, which leaves the original plaintext m. These masks can also be combined; [35] presents triple masking where the exponent, input data, and operations are all masked. Masking does not protect against SPA, and the effectiveness of masking depends on frequency of mask change and the mask length. Additionally, as you can see in Algorithms 3 and 4, there are precomputations, additional memory, and post computations required. These requirements makes masking a computationally expensive countermeasure.

2.3.4 Noise Injection. As seen in Equation 5 [6], adding noise lowers the SNR thus requiring more traces to successfully mount a power analysis attack.

$$SNR = \frac{Var(Exploitable\ Signal)}{Var(Switching\ Noise\ +\ Electrical\ Noise)} \tag{5}$$

In [36], Messerges et al. take an in depth look at noise effects. As previously seen in Equation 1, as the number of traces collected increases the variance of random noise decreases. Noise can be generated via random number generators, ring oscillators, garbage functions, etc. One must ensure that the noise is close to the cryptographic operation's clock frequency and spread throughout the circuit. Naive noise injection attempts can be frequency filtered or bypassed by using EM probes to pinpoint the true power signal of the cryptographic circuitry.

2.3.5 Dual-Rail Logic (DRL) and Wave Dynamic Differential Logic (WDDL). DRL and WDDL are logic level countermeasures to balance power consumption. As seen in Figure 5, this logic takes the inputs a, b and their inverses $\bar{a}$, $\bar{b}$, then computes the output q and its inverse $\bar{q}$.

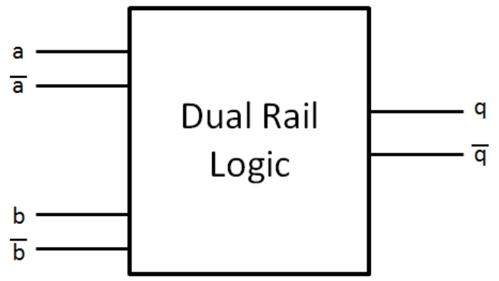

Figure 5: Example of Dual Rail Logic Block

This logic helps mask by balancing the Hamming Weights of computations by concurrently computing the complement, thus maximizing Hamming Weight switching for every operand. Work in [37] and [38] describes how to integrate dual rail logic into design flows for synthesis. While dual rail logic implementations provide a level of DPA security, increases in power consumption and area are significant [39].

2.3.6 Time Delay. Since DPA relies on correlation between power samples at a fixed point in time across all power traces, alignment of traces is very important.

A cryptographic designer can use misalignment as a countermeasure. The use of random delay insertion (RDI), can interrupt a process and vary at what point in time intermediates are operated on. RDI creates misalignments, lowers correlation, and drastically increases the number of traces needed for successful attack [40]. Since inserting delays hurts performance, the maximum delay allowed is usually kept quite short, [41] discuses methods to choose the optimum delay. When adding delays to an operation, the total power consumed is spread out over different time periods. By integrating over these periods instead of correlating on single clock cycles, an attacker can correlate on the total power consumed each operation, independent of delays.

2.4 Conclusion

We have shown a variety of attacks and countermeasures from within current literature. Although we have focused on RSA, these attacks and countermeasures also apply to other public key encryption algorithms using multiply-and-square (add-and-double) algorithms, such as elliptic curve cryptography, as surveyed in [42]. Given all known attacks, we have seen that implementing a specific countermeasure may protect from one attack while creating vulnerabilities to another attack. Undoubtedly many more attacks and countermeasures will be developed in the future. Until an implementation is able to create complete independence between operands, intermediates, and secret keys, attackers will continue to develop successful attacks. For further reading, [43–45] provide additional in depth analysis of power analysis attacks.

III. Methodology

The following chapter outlines the methodology used for design of experiments, as well as developing a dynamic architecture capable of resisting SCA attacks.

3.1 Problem Definition

Current SCA attack countermeasures are largely static. Given enough time and a sufficient number of power traces, static countermeasures can be broken as they merely attempt to increase the difficulty of attack to an infeasible level. However, if countermeasures are constantly changing, as in a dynamic algorithm, timing and power consumption would be a "moving target", making correlation between traces much more difficult than static countermeasures.

3.2 Goals and Hypothesis

The goal of this research is to develop and synthesize an architecture on an FPGA that is capable of protecting an RSA implementation from SCA attacks. The hypothesis of this research is that a dynamic circuit is capable of randomizing timing and power consumption in such a manner that SCA attacks are no more successful than brute force attacks.

3.3 Approach

To accomplish the goals of this research, dynamic architecture is developed to introduce timing and power randomization into the circuit. This goal is accomplished via two approaches: randomizing the radix of Booth multiplication and randomizing the window size in exponentiation.

3.3.1 Randomizing Multiplication. Basic binary multiplication is accomplished by repeated iterations of addition and shift operations. In contrast, the

Booth Multiplier [46] codes operands in such a way to skip unnecessary addition steps. By increasing the radix of the Booth Coding, it is possible to shift twice per cycle, thus halving the required cycles to complete multiplication. This research exploits this concept by developing a variable radix encoding Booth Multiplier. Driven by a pseudorandom number generator, this multiplier shifts a random number of bits each cycle. This random shifting induces randomness in the timing of operand use, randomizes power consumption, and multiplication completes in a nondeterministic number of clock cycles. This process also causes alignment of traces to be lost, making correlation much more difficult.

3.3.2 Randomizing Exponentiation.

As binary multiplication can be seen as repeated addition operations, binary exponentiation can be seen as repeated multiplication. Using multiple bit exponentiation in a windowing fashion [10] randomizes how many key bits are used for exponentiation at each step. This approach adds another layer of randomness in timing and power consumption.

3.4 System Boundaries

The dynamic RSA encryption hardware is the System Under Test (SUT), shown in Figure 6. The SUT is composed of four major components: a Field Programmable Gate Array (FPGA), the RSA encryption algorithm, a processor used for peripheral communication, and the dynamic RSA implementation. The Component Under Test (CUT) is the dynamic RSA implementation. This dynamic hardware is synthesized from VHDL onto a Xilinx Virtex-5 FPGA. The built in hardcore PowerPC 440 processor is used for communication with the third party Riscure Inspector software for plaintext/ciphertext exchange.

3.4.1 FPGA.

The hardware platform used for this research is a the Xilinx Virtex-5 FX FPGA. VHDL code is synthesized using the Xilinx design suite and downloaded to the board for hardware configuration. The characteristic of an

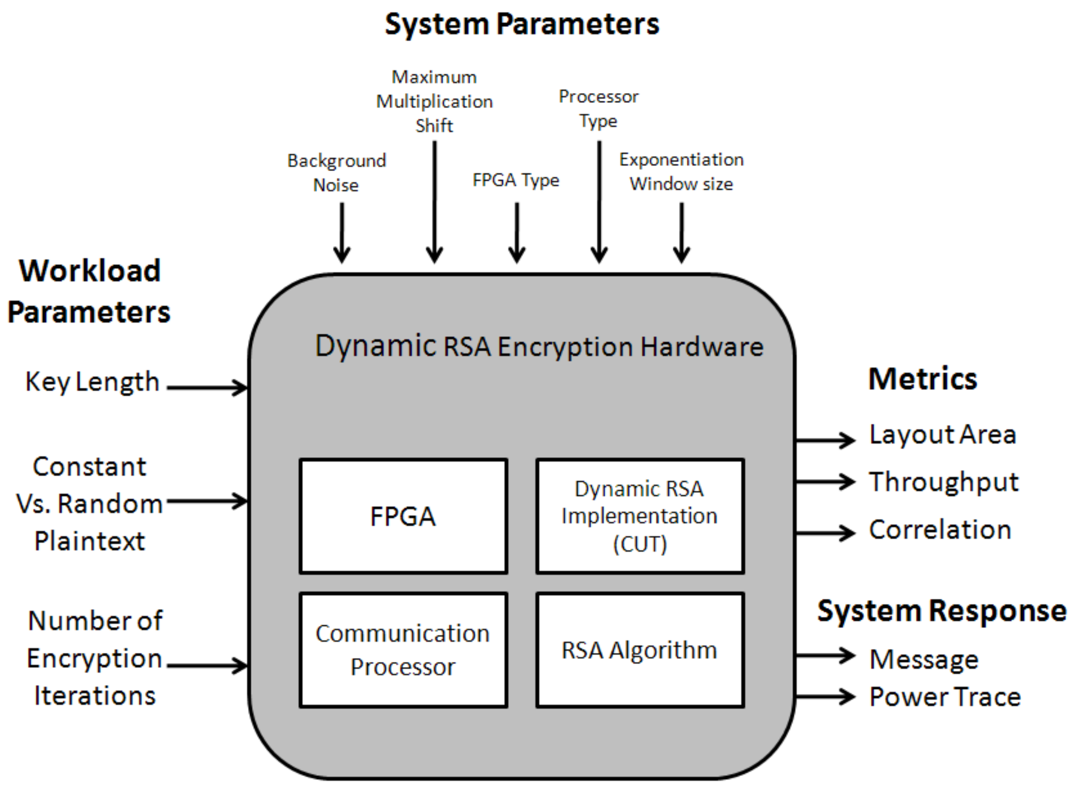

Figure 6: System Block Diagram

FPGA make it an ideal research platform. FPGAs are easily reconfigured with new iterations of hardware configurations and allow for rapid development.

3.4.2 PowerPC Processor.

The Xilinx Virtex-5 FX series used in this research includes an on chip hardcore PowerPC440 processor. This processor is used to run C code for communication between Riscure Inspector software to pass the modulus, keys, and messages to the cryptographic system. This processor receives the data and commands from Inspector and loads/reads the registers of the dynamic RSA implementation.

3.4.3 Dynamic RSA Implementation.

The dynamic RSA implementation is the CUT. This implementation consists of a variable shift multiplier and variable window size modular exponentiator, as introduced in Section 3.3. These subcomponents randomize the timing and power consumption of cryptographic operations, presumably making SCA attacks much more difficult. This research implements the RSA encryption algorithm with various hardware changes in attempts to better secure the implementation from SCA attacks.

3.5 System Services

The system in this research supplies two services: RSA encryption and SCA obfuscation.

3.5.1 RSA encryption.

This system completes RSA encryption on a plaintext message given a modulus and key. This service is easily measured as pass/fail via verifying correct ciphertext/plaintext pairs.

3.5.2 SCA obfuscation.

The obfuscation service is a randomization of timing and power consumption. This service is successful if the obfuscated power trace results in less correlation than the baseline implementation. Any increase in protection is defined as successful service. The level of increased protection is later evaluated and compared against other experimental runs.

3.6 Workload Parameters

The workload parameters of this system are: key length, constant versus random plaintext, and the chosen number of encryption iterations.

3.6.1 Key Length. Key length drives the number of operations in modular exponentiation. Keys of longer length cause RSA encryption to compute more multiplication steps and decrease throughput. SCA attacks bypass the functional security of RSA. Which means an increase in key length results in only a linear increase in SCA difficulty, as opposed to the exponential increase in brute force attack difficulty when using a longer key. For ease of trace collection and processing, this research implements 512-bit RSA with a 32-bit key. Since each key bit is attacked individually, attacking this configuration is identical to attacking the first 32-bits of larger key implementations.

3.6.2 Constant Versus Random Plaintext. Continuous identical messages being encrypted is not realistic. However, in a research context, constant plaintext messages allows for additional analysis. Given static SCA countermeasures, the power signature and timing of two identical messages may be extremely similar. This would allow an adversary to average out background noise. However, a dynamic SCA countermeasure is capable of much different power signatures and timing when encrypting two identical messages. Constant plaintext encryption also allows for the standard deviation of various countermeasures to be compared, revealing which has the most variability for identical messages, and allowing for experimental repeatability.

3.6.3 Number of Encryption Iterations. The probability of success for many SCA attacks is a function of the number of message/power trace pairs captured. Increasing the pool of message/power trace pairs lowers noise and increases correlation, both of which are critical for successful attack. Based on pilot studies, the number of encryption iterations are chosen such that attacks are possible, but without making successful attacks trivial. Setting a benchmark for number of encryption iterations is critical for useful analysis when comparing different experiments and multiple hardware setups.

3.7 Performance Metrics

This research uses four performance metrics: layout area, encryption throughput, and SCA correlation.

3.7.1 Layout Area. Many public key encryption systems are implemented on smart cards and other constrained systems. These constrained systems drive the need for small implementations. While this research is focused on protecting an encryption system from SCA attacks, it is useless to develop a system that is unrealizable. Therefore, the baseline implementation of RSA is compared against the dynamic protection system to determine the area increase.

3.7.2 Throughput. Much like the area overhead metric, a protection method that renders an encryption system too slow is also not practical. Therefore, the throughput of encryption is also measured and compared to the baseline implementation.

3.7.3 Correlation. The primary focus of this research is SCA protection of an encryption system. Since obfuscation and protection are qualitative measurements, a quantitative measurement of correlation is used to characterizes protection. Correlation is a metric of how well an attempted attack is able to map key guesses to power traces. This metric is also used to compare against the baseline implementation.

3.8 System Parameters

System parameters are characteristics within the system boundary that affect the performance metrics. The parameters of the SUT are background noise, maximum shift in multiplication, exponentiation window size, FPGA type, and processor type.

3.8.1 Background Noise.

With any experiment there is measurement noise. Background noise is particularly important when conducting SCA attacks. This research uses an inductive probe to non-invasively measure the power consumption of the encryption system. These measurements are particularly sensitive to input power supply noise, clock generating circuits, and ambient electromagnetic white noise at the measured frequencies.

3.8.2 Randomized Radix Encoding Booth Multiplication.

A primary element of the CUT is a randomized radix encoding Booth multiplication. When the radix is not randomized the multiplier operates as a normal Booth Multiplier. However, when radix of multiplication is randomized, it randomizes power and timing, thus decreasing the correlation and increasing the obfuscation.

3.8.3 Exponentiation Method.

The baseline implementation of the encryption hardware uses simple binary exponentiation. To increase obfuscation, multiple bit windowing exponentiation methods are implemented.

3.8.4 FPGA Model.

This research is conducted on a Xilinx Virtex-5 FPGA. Although this parameter is within the system boundary and remains constant throughout this research, different FPGA families or manufacturers can affect system performance metrics. Changes in underlying FPGA hardware may increase (or decrease) side channel leakage. This change in leakage is particularly true on the Xilinx Virtex-5. As seen in Figure 3.8.4, several power capacitors on the bottom the ML507 evaluation board produce a particularly clear power signal. Smaller power signature changes could also be caused by using other FPGA manufacturers since their proprietary tools will implement different place and route algorithms during design synthesis.

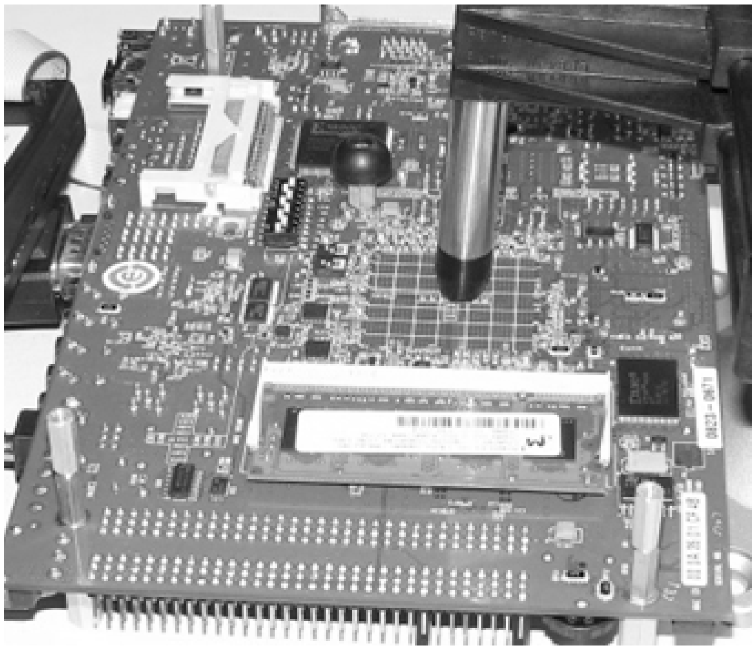

Figure 7: Power Capacitors On Bottom of Virtex-5 ML507 Development Board

3.8.5 Processor Type. For communication, this research uses the hardcore PowerPC440 processor on the physical Virtex-5 FX. Although this also remains a constant throughout this research, different processors have the ability to affect power signatures. Simply choosing another processor with a clock frequency significantly different from the encryption CUT's clock frequency allows an attacker to filter out the processor noise, thus leaving a cleaner power signature of the CUT.

3.9 Factors

This experiment uses two factors each with two levels. These factors and levels are summarized in Table 4. The combination of static radix Booth multiplication and binary exponentiation represent the baseline configuration for this research.

Table 4: Factors and Levels

Factors	Levels
Multiplication Radix	Static & Randomized
Exponentiation Window Size	Binary & Randomized

As the levels of each factor are varied from static to randomized, the expected result is an increase in randomization in power and timing. However, this is a trade off since randomizing multiplication radix and increasing exponentiation window size require more precomputations and increasingly complex hardware, decreasing throughput and increasing power consumption.

3.9.1 Booth Multiplication Radix. To maximize randomization in timing and power consumption, the radix of Booth encoding is changed within each multiplication operation. This factor is tested at two levels: static 2-radix (standard Booth multiplication) and randomized radix.

3.9.2 Exponentiation Window Size. To build upon the randomized multiplier, a variable modular exponentiator introduces further randomness in timing and power. The levels of windowing size of this exponentiator are standard binary exponentiation and randomized window.

3.10 Evaluation Technique

The evaluation method of this research is direct measurement. The SUT is coded in VHDL and synthesized onto a Xilinx Virtex-5 FPGA for each configuration. Using Riscure's Inspector SCA software as a driver, modulus, keys, and messages are sent via RS-232 to the SUT hardware on FPGA. At the beginning of the encryption cycle, the SUT triggers a Lecroy WavePro 725zi oscilloscope to begin sampling a Willtek 1207 inductive probe that is mounted directly above the FPGA. These power traces are transfered from the oscilloscope to the Inspector software via Ethernet. This setup is shown in Figure 8.

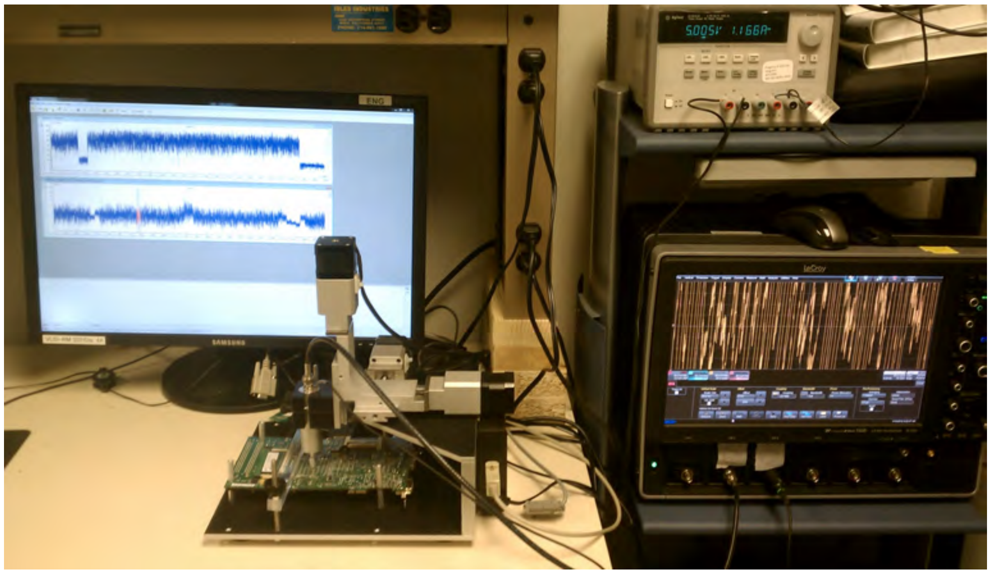

Figure 8: Hardware Test Setup for Trace Collection

These power traces are imported into MATLAB for statistical analysis. Custom software is run in MATLAB to compute correlation values and attempt a Hamming Weight model Differential Power Analysis (DPA) attack. This method implements a very common and powerful DPA attack to evaluate each experimental run. The evaluation technique is validated on the baseline implementation. Results from the DPA attack on the baseline implementation are compared against the known secret key to validate successful attack. Subsequent experimental runs compare attack results with their known secret keys to determine success of attack. Correlation values are also compared against the baseline implementation to measure success toward the protection goals and measures of obfuscation.

3.11 Experimental Design

This research includes a full factorial design of experiments. Defining two factors with two levels each yields four total experiments. These experiments are replicated three times resulting in 12 total experiments. A 95% confidence interval is used to evaluate this research. This confidence interval is chosen because it allows for sufficient differentiation between experimental configurations.

3.12 Methodology Summary

The goal of this research is to determine whether a dynamic architecture on an FPGA is capable of protecting an RSA implementation from SCA attacks. This goal is accomplished by randomizing timing and power consumption through a variable radix multiplier and variable window size exponentiator. By feeding the system representative workload parameters, verifying system services fulfillment, and monitoring system metrics, the results are indicative of the CUT's performance. Using two factors varied by two levels, four experiments are run in a full factorial experiment. Each experiment is evaluated by a hardware test using real time encryption and physical side channel collections. These results are validated against the baseline implementation to determine the effectiveness of the dynamic architecture as a defense against SCA attacks.

IV. Countermeasure Design and Results

The following chapter outlines the design, simulation results, and results from real world attacks on the dynamic architectural countermeasure developed in this research.

4.1 Basic Hardware Design

In order to achieve the goals of of this research, to develop and synthesize an architecture capable of protecting an RSA implementation against power analysis attacks, the modular exponentiation operation is obfuscated. This obfuscation is achieved by dynamic algorithms that randomize the power consumption and timing of each calculation. Randomization is introduced at two levels within the architecture, multiplication and exponentiation window size.

4.1.1 Booth Multiplication. Basic binary multiplication is accomplished by repeated addition and shift operations. In contrast, Booth encoded multiplication [46] recodes operands in such a way to reduce the number of costly addition steps. Booth multiplication was chosen as the base multiplication architecture because it allows for multiple bits to be encoded and operated on in a single iteration. As shown in the flowchart in Figure 9, Booth multiplication tests the LSBs of the multiplier to determine whether adding or subtracting the multiplicand is needed in each iteration. Booth's approach has a speedup benefit since arithmetic logic unit (ALU) operation is only required on 0-to-1 and 1-to-0 transitions. For each iteration, the LSBs of the multiplier are tested, ALU operation is completed (if needed), and multiplier and product registers are shifted.

4.1.1.1 Modular Reduction. To complete modular multiplication, the product of Booth multiplication must be reduced via a modulo operation. There are two main classes of reduction techniques for modular multiplication: multiply-then-reduce and reduce-as-you-go [47]. This research implements the reduce-as-you-go approach. Using the simple reduction method outlined in Algorithm 7 keeps the product within the bounds of the modulus. This reduction algorithm is left-to-right (MSB to LSB), which forces Booth encoding to also be preformed left-to-right. Although it is not implemented in this research, this architecture does not prevent implementing the popular Montgomery reduction [48].

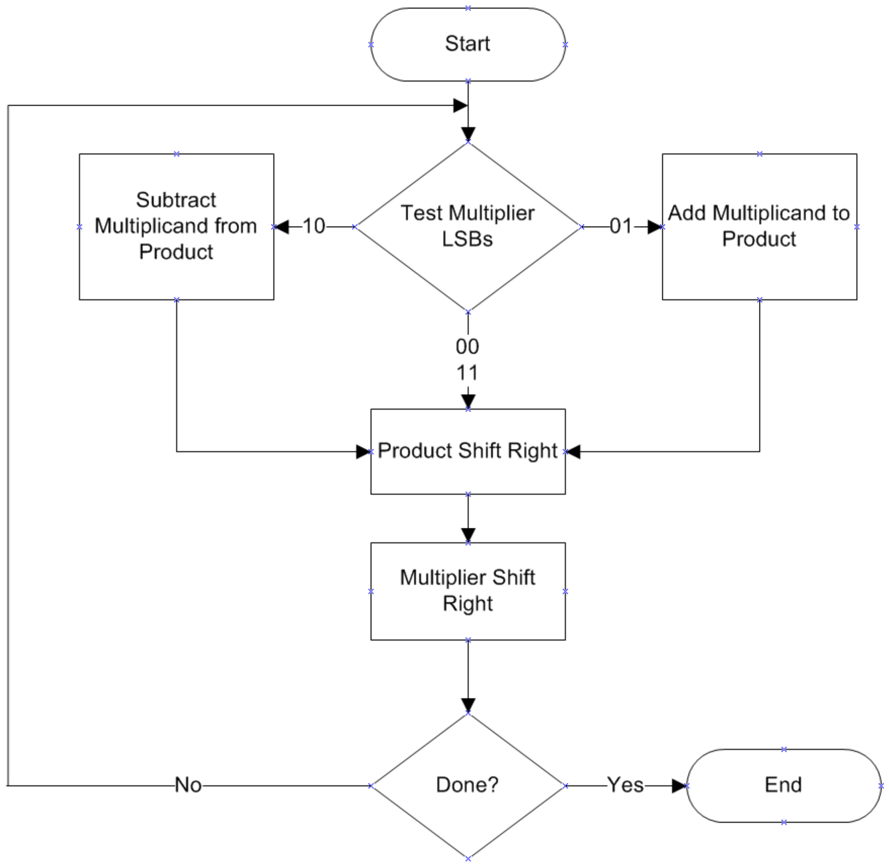

Figure 9: Flowchart of Booth Multiplication [3]

The resultant modular Booth multiplier design is shown in Figure 10. The Booth multiplier hardware concurrently computes all possible encoding results and uses a multiplexer to choose the correct result based on the Booth encoding of the MSBs of the multiplier. This result is fed into the modular reducer section of the hardware. Here, all possible reductions are computed concurrently and the MSB of

Algorithm 7 Modular Reduction

```
INPUT: Multiplier $A$, Multiplicand $B$, Modulus $N$
OUTPUT: $A \cdot B \pmod{N}$
Double the product register
If product register is $> |N|$ modulus, Add/subtract to adjust
Add/Subtract via Booth operation
If product register is $> |N|$ modulus, Add/subtract to adjust
```

the results determine the correct reduction. Using 2's complement arithmetic, a MSB of 1 represents negative numbers while MSB of 0 represents positive numbers. The modular reducer selects the smallest nonnegative result. Therefore, this architecture implements combinational logic that computes all possible Booth results, selects the correct Booth result, computes all possible modular reductions, and selects the correct modular reduction all in a single clock cycle.

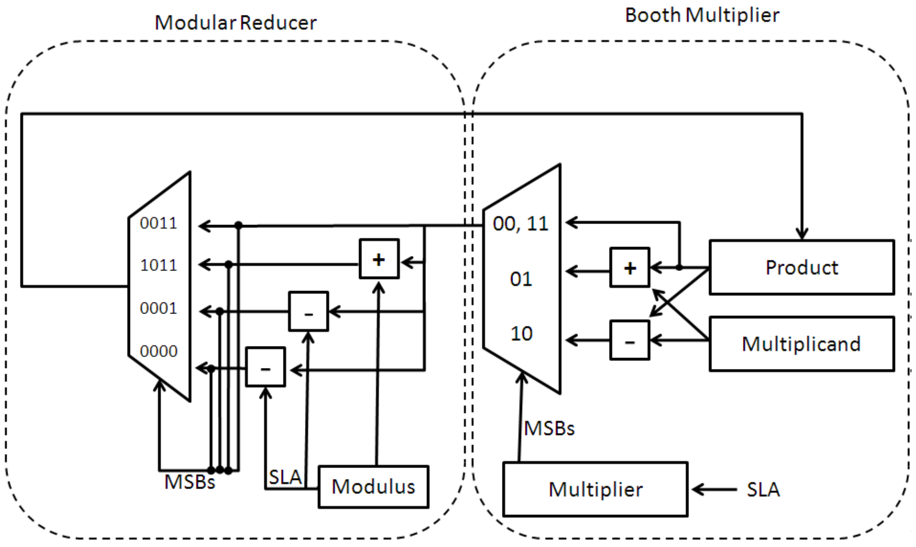

Figure 10: Modular Booth Implementation

4.1.2 Modular Exponentiation. As reviewed in the multiply-and-square algorithms presented in Section 2.1.1, exponentiation can be performed via multiple modular multiplication steps. In order to complete the modular multiplications $R^2 \pmod{N}$ and $R \cdot m \pmod{N}$, repeated operations using the modular Booth multiplier hardware is used. This paper focuses on a modification of the Booth concept to dynamically randomize the calculation, which provides a level of SCA protection to RSA.

4.2 Dynamic Architectural Countermeasures Design

The following sections outline the design of the dynamic architectural countermeasure.

4.2.1 Randomized Radix Encoding Booth Multiplier. This research proposes a custom design for a Booth multiplier with randomized radix encoding. Increasing the radix of Booth encoding causes more bits to be encoded each cycle, thus further decreases the number of required cycles to complete multiplication. The Booth encoding for radix 2, 4, and 8 are shown in Table 5. Note that all multiples of the multiplier M are trivial (i.e., shifts and 2's complements) except the multiple of three. However, the multiple of three is easily precomputed and stored for later use.

Table 5: Table of Booth Encoding For Given Radix

Radix-2	M	Radix-4	M	Radix-8	M
00	0	000, 111	0	0000, 1111	0
01	M	001, 010	M	0001, 0010	M
10	$-M$	011	$2M$	0011, 0100	$2M$
11	0	100	$-2M$	0101, 0110	$3M$
		101, 110	$-M$	0111	$4M$
				1000	$-4M$
				1001, 1010	$-3M$
				1011, 1100	$-2M$
				1101, 1110	$-M$

The radix of Booth encoding is randomly varied to provide protection against SCA. Driven by a pseudorandom number generator, henceforth referred to as random, the multiplier hardware randomly selects a radix at each iteration. This dynamic architecture induces randomness into the timing of operand use, randomizes intermediate operands, randomizes power consumption, and multiplication completes in a nondeterministic number of clock cycles. This randomization causes alignment of traces to be lost, making correlation much more difficult.

4.2.2 Variable Window Exponentiator. Building upon the randomness introduced via the randomized radix encoding Booth multiplier, the window size of the exponentiator is also randomized. Starting with the general windowing Algorithm 4 from Section 2.1.1.2, the precomputations needed for a maximum window size of 3 are computed. Continuing into the second loop of the algorithm, the window size k is randomly varied between 1, 2, or 3 each iteration. If window size 1 is chosen, the algorithm simplifies to simple binary exponentiation. If a window size of 2 (or 3) is chosen, $R = R^4 (R = R^8)$ is computed and $R = R \cdot c_{d_i}$ is calculated using the precomputed powers of c. Similar to randomizing the radix in Booth multiplication, randomizing the window size in exponentiation further introduces timing randomness, operations randomness, and randomness in calculated intermediate values. Although this research presents an architecture with window sizes of 1, 2, and 3, it is possible to increase the window size further for added randomization. However, increasing window size is a trade off since increasing the window size exponentially increases the number of precomputations and storage needed for very large integers.

4.3 Simulation Results

The discussed design is developed in VHDL and simulated using Mentor Graphics ModelSim. The following sections discuss simulated execution of the randomized radix encoding Booth multiplier, variable window exponentiator, and the combined countermeasure.

4.3.1 Baseline. The baseline configuration consists of traditional Radix-2 Booth multiplication and simple binary exponentiation. Figure 11 shows a calculation of $146^{187} \pmod{207} = 47$. Note that timing is constant and all modular multiplication operations take 11 cycles. This consistency is a best case scenario for an attacker. Calculating intermediate values based on a correct key guesses will easily generate a correlation spike as all traces are aligned.

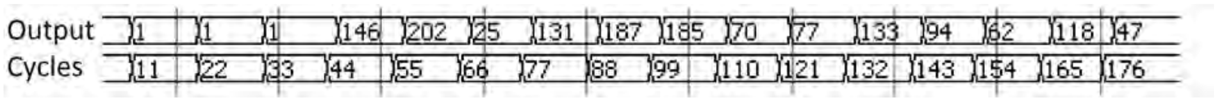

Figure 11: Baseline Configuration Simulation of $146^{187} \pmod{207} = 47$

4.3.2 Randomized Multiplication Simulation. Three computations of $146 \cdot 85 \pmod{207}$ were simulated to show the induced randomness in timing and operations. The simulation results are presented in Figure 12. It is clearly seen that all three modular multiplication computations' *output* arrive at the correct answer of $146 \cdot 85 \pmod{207} = 197$. However, the simulations complete in 9, 5, and 6 iterations respectively.

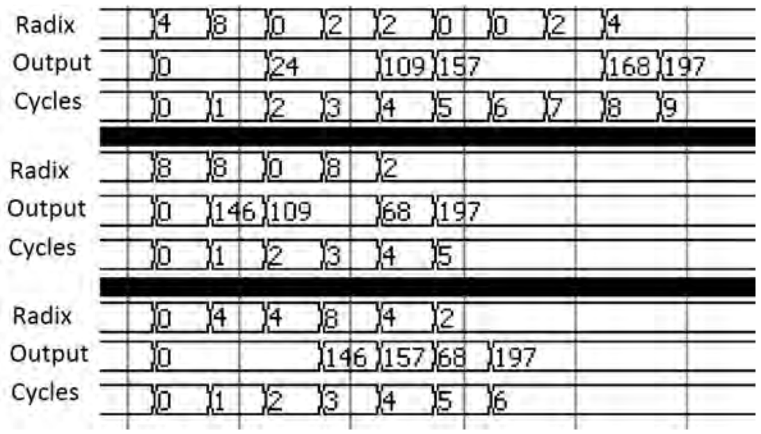

Figure 12: Three Simulations of $146 \cdot 85 \pmod{207} = 197$

Table 6 presents the computations in tabular form in order to show the randomness in the calculations, where the multiplier 85 is represented in binary. Notice as each computation traverses from MSB to LSB of the multiplier, horizontal separating lines show which bit(s) each cell operates on. Notice where cells end on the same bit (horizontal lines line up), the calculated intermediate value is the same, but where iterations end on different bits of the multiplier (horizontal lines do not

Table 6: Three Computations of $146 \cdot 85 \pmod{207}$ [Entire Table $\pmod{207}$]

	Multiplier	1st Run	Radix	2nd Run	Radix	3rd Run	Radix
MSB	0	$P = 0$ $2^2 \cdot P = 2^2 \cdot 0 = 0$ $P + 0 = 0$	$4 = 00\underline{0}$	$P = 0$ $2^3 \cdot P = 2^3 \cdot 0 = 0$ $P + M = 146$	$8 = 000\underline{1}$	$P = 0$ $2^2 \cdot P = 2^2 \cdot 0 = 0$ $P + 0 = 0$	$4 = 00\underline{0}$
	0						
	0	$2^3 \cdot P = 2^3 \cdot 0 = 0$ $P + 3M = 0 + 3 \cdot 146 = 24$	$8 = 010\underline{1}$			$2^2 \cdot P = 2^2 \cdot 0 = 0$ $P + M = 0 + 146 = 146$	$4 = 01\underline{0}$
	1			$2^3 \cdot P = 2^3 \cdot 146 = 133$ $P - 3M = 133 - 3 \cdot 146 = 109$	$8 = 101\underline{0}$		
	0					$2^3 \cdot P = 2^3 \cdot 146 = 133$ $P + 3M = 133 + 3 \cdot 146 = 157$	$8 = 010\underline{1}$
	1	$2 \cdot P = 2 \cdot 24 = 48$ $P - M = 48 - 146 = 109$	$2 = 1\underline{0}$				
	0	$2 \cdot P = 2 \cdot 109 = 11$ $P + M = 11 + 146 = 157$	$2 = 0\underline{1}$	$2^3 \cdot P = 2^3 \cdot 109 = 44$ $P + 3M = 44 + 3 \cdot 146 = 68$	$8 = 010\underline{1}$		
	1	$2 \cdot P = 2 \cdot 157 = 107$ $P - M = 107 - 146 = 168$	$2 = 1\underline{0}$			$2 \cdot P = 2^2 \cdot 157 = 7$ $P - M = 7 - 146 = 68$	$4 = 10\underline{1}$
	0	$2^2 \cdot P = 2 \cdot 168 = 51$ $P + M = 51 + 146 = 197$	$4 = 01\underline{0}$				
LSB	1			$2 \cdot P = 2 \cdot 68 = 136$ $P - M = 136 - 146 = 197$	$2 = 1\underline{0}$	$2 \cdot P = 2 \cdot 68 = 136$ $P - M = 136 - 146 = 197$	$2 = 1\underline{0}$

line up) the addition operation and results are different. This effect is significant because although the algorithm is computing the same answer, the power signature will be vastly different not only because of the induced timing variance, but also the switching activity from different operands in each iteration.

4.3.3 Randomized Window Exponentiator. A simulation of three calculations of $146^{187} \pmod{207}$ are presented in Figure 13. First, all three iterations are verified to arrive at the correct answer of $146^{187} \pmod{207} = 47$. Multiplication is held constant, 11 cycles to complete, and the only variability is from the randomized window. The timing variance of the randomized exponentiation leads to completion in 231, 220, and 242 cycles respectively.

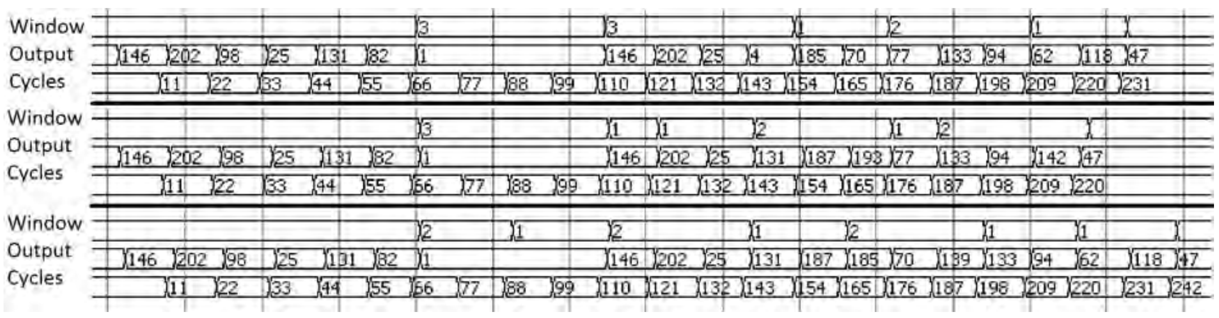

Figure 13: Random window Configuration Simulation of $146^{187} \pmod{207} = 47$

Table 7: Three Computations of $146^{187} \pmod{207}$ [Entire Table $\pmod{207}$]

	Exponent	1st Run	Window	2nd Run	Window	3rd Run	Window
MSB	0	$R = 1$		$R = 1$		$R = 1$ $R^2 = 1^2 = 1$	
	0	$R^2 = 1^2 = 1$ $R^2 = 1^2 = 1$	3	$R^2 = 1^2 = 1$ $R^2 = 1^2 = 1$	3	$R^2 = 1^2 = 1$	2
	1	$R^2 = 1^2 = 1$ $R \cdot M = 1 \cdot 146 = 146$		$R^2 = 1^2 = 1$ $R \cdot M = 1 \cdot 146 = 146$		$R^2 = 1^2 = 1$ $R \cdot M = 1 \cdot 146 = 146$	1
	0	$R^2 = 146^2 = 202$		$R^2 = 146^2 = 202$	1	$R^2 = 146^2 = 202$	
	1	$R^2 = 202^2 = 25$ $R^2 = 25^2 = 4$	3	$R^2 = 202^2 = 25$ $R \cdot M = 25 \cdot 146 = 131$	1	$R^2 = 202^2 = 25$ $R \cdot M = 25 \cdot 146 = 131$	2
	1	$R \cdot M^3 = 4 \cdot 98 = 185$		$R^2 = 131^2 = 187$ $R^2 = 187^2 = 193$		$R^2 = 131^2 = 187$ $R \cdot M = 187 \cdot 146 = 185$	1
	1	$R^2 = 185^2 = 70$ $R \cdot M = 70 \cdot 146 = 77$	1	$R \cdot M^3 = 193 \cdot 98 = 77$	2	$R^2 = 185^2 = 70$ $R^2 = 70^2 = 139$	
	0	$R^2 = 77^2 = 135$		$R^2 = 77^2 = 133$	1	$R \cdot M^2 = 139 \cdot 202 = 133$	2
	1	$R^2 = 135^2 = 94$ $R \cdot M^1 = 94 \cdot 146 = 62$	2	$R^2 = 133^2 = 94$		$R^2 = 133^2 = 94$ $R \cdot M = 94 \cdot 146 = 62$	1
LSB	1	$R^2 = 62^2 = 118$ $R \cdot M = 118 \cdot 146 = 47$	1	$R^2 = 94^2 = 142$ $R \cdot M^3 = 142 \cdot 98 = 47$	2	$R^2 = 62^2 = 118$ $R \cdot M = 118 \cdot 146 = 47$	1

More interestingly, the tabular calculation shown in Table 7 is inspected to see the countermeasure's effect on intermediate calculations. Because intermediate calculations, power consumption, and timing are of such importance to DPA attacks, this randomized modular exponentiation architecture is a countermeasure to defeat SCA attacks. Again take notice where cells end on the same bit (horizontal lines line up), the calculated intermediate value is the same, but where iterations end on different bits of the exponent (horizontal lines do not line up) results are different.

4.3.4 Combined Countermeasure Simulation. To maximize the power and timing randomization of the countermeasure, the randomized Booth multiplier and the randomized exponentiator are combined. Three calculations of $146^{187} \pmod{207}$ are once again simulated and presented in Figure 14. First, the simulations are again verified to have computed the correct answer of $146^{187} \pmod{207} = 47$. This configuration combines the timing variance from both the previous countermeasures. The combined timing variance of the randomized multiplication and exponentiation lead to completion in 117, 106, and 119 cycles respectively. These results validate via simulation that this proposed dynamic architectural countermeasure successfully randomizes timing, randomizes intermediate values, and causes large misalignment in traces. These randomizations all increase SCA attack difficulty.

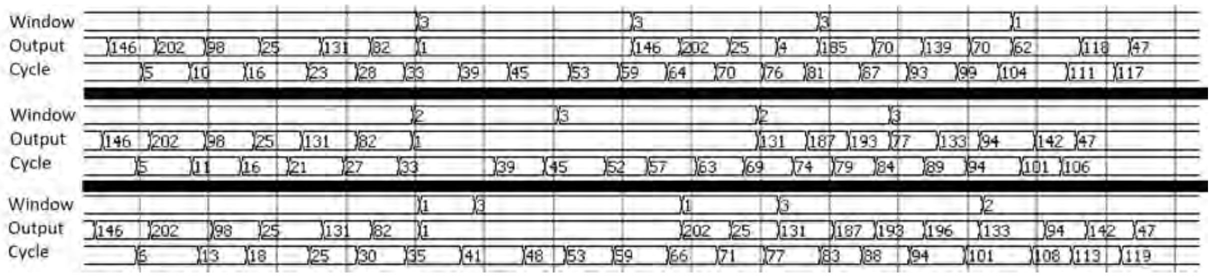

Figure 14: Three Simulations of 146^{187} (mod 207) = 47

4.4 Hardware Trace Results

The simulation results from the previous sections are the result of VHDL code. This code was subsequently synthesized onto a Xilinx Virtix-5 FPGA using Xilinx's XPS Design Suite. As previously described in depth in Section 3.10, the cryptographic hardware on the FPGA interfaces with the Riscure Inspector's 3rd party software and a Lecroy Oscilloscope to collect power traces. All configurations are synthesized as 512-bit RSA cryptosystems. To aid in the speed of trace collection and processing, only a 32-bit key is used. The smaller key does not effect the method binary multiplication. It only reduces the number of modular multiplications and shortens the overall encryption time for testing purposes.

4.4.1 Baseline Configuration. The first configuration to be tested is the baseline configuration. Figure 15 shows the raw power trace. Because the traces are collected via EM probe, they are very noisy. Figure 16 shows the frequency spectrum of the raw trace. Clearly, the traces are riddled with higher harmonics of the 12.5 mHz clock frequency. Using DSP, the noisy traces are filtered with a bandpass filter centered on the 12.5 mHz clock frequency (10.5-14.5 mHz bandpass). Figure 17 presents the post processed frequency spectrum, and Figure 18 shows the resultant trace with much less noise. Notice how the beginning and end of the filtered trace show very little power consumption, while the unfiltered trace shows unrelated noise at those times.

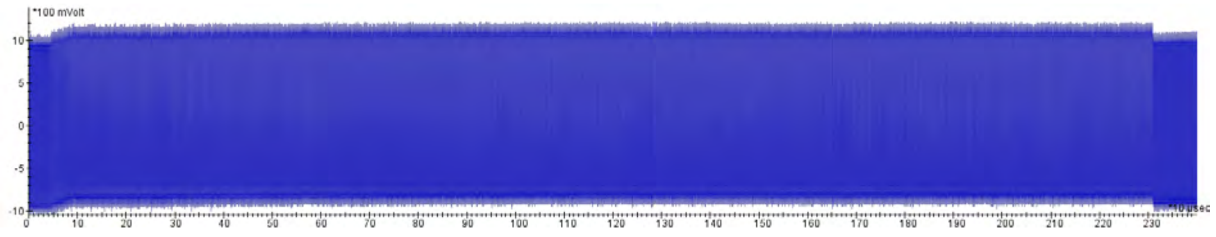

Figure 15: Baseline Configuration - Raw Trace

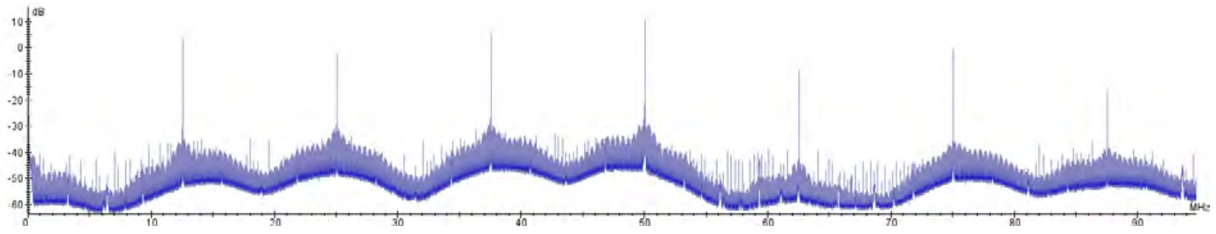

Figure 16: Baseline Configuration - Spectrum Analysis

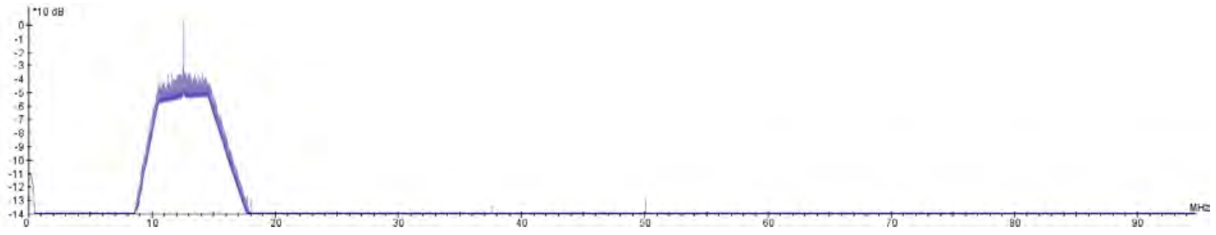

Figure 17: Baseline Configuration - Bandpass Filtered Spectrum

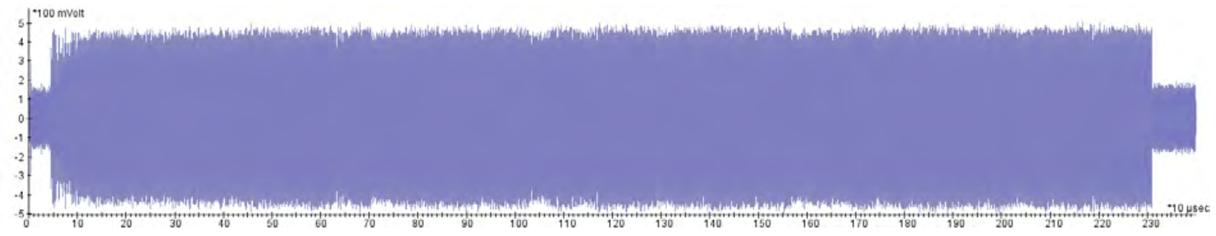

Figure 18: Baseline Configuration - Post Filtering

Zooming in to the processing of the first few bits, it is possible to clearly define each block of execution in the filtered trace, Figure 19. Figure 20 shows several bits from two encryption operations with the same plaintext but different keys. Notice there is no discernible timing or power difference in the squaring or multiplying operations. This validates that the implementation is resistant to Simple Power Analysis (SPA). However the baseline configuration is still very susceptible to Differential Power Analysis (DPA).

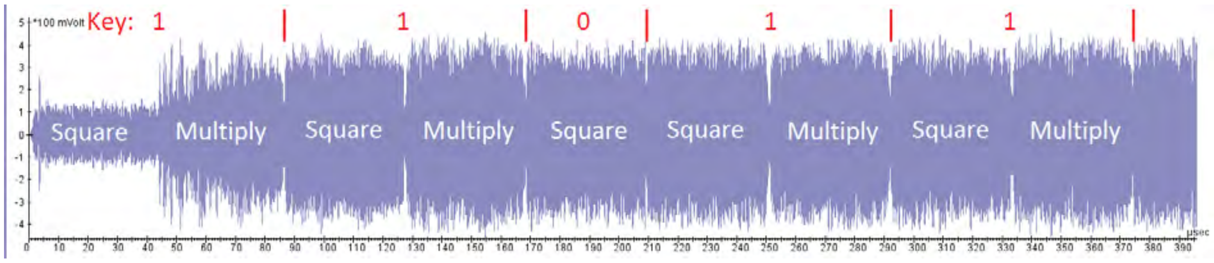

Figure 19: Baseline Configuration - Zoomed, Filtered

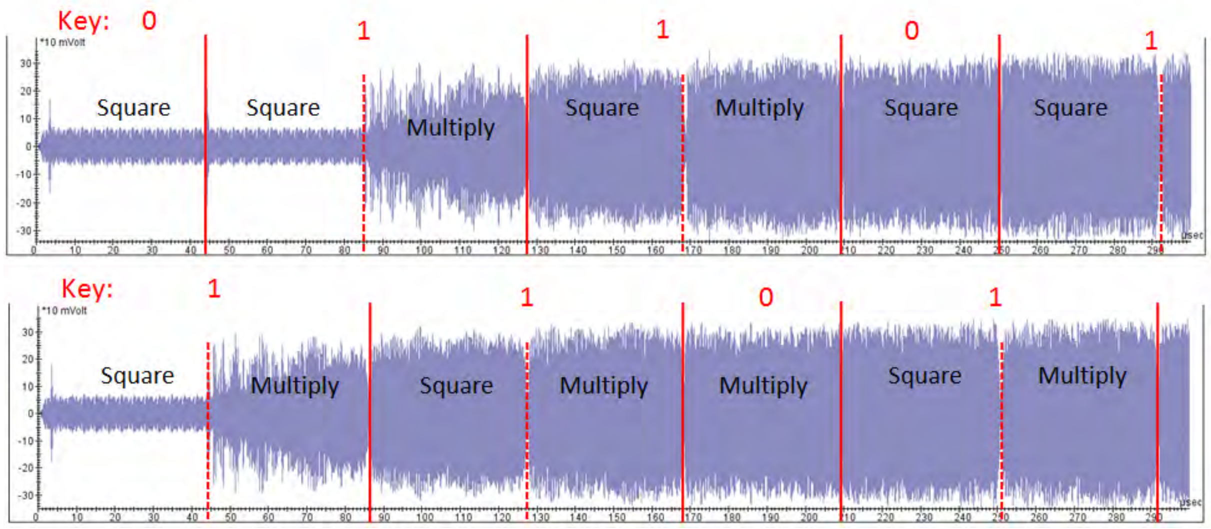

Figure 20: Baseline Configuration - Immune to SPA

Lastly, Figure 21 shows the ending of five power traces. Notice that the timing is perfectly aligned. The aligned power spikes create a very favorable condition for SCA attacks. Figure 21 is used as a benchmark to gauge other configurations' timing.

4.4.2 Randomized Radix Encoding Booth Multiplier. This section presents hardware test results from the randomized radix encoding Booth multiplier. The

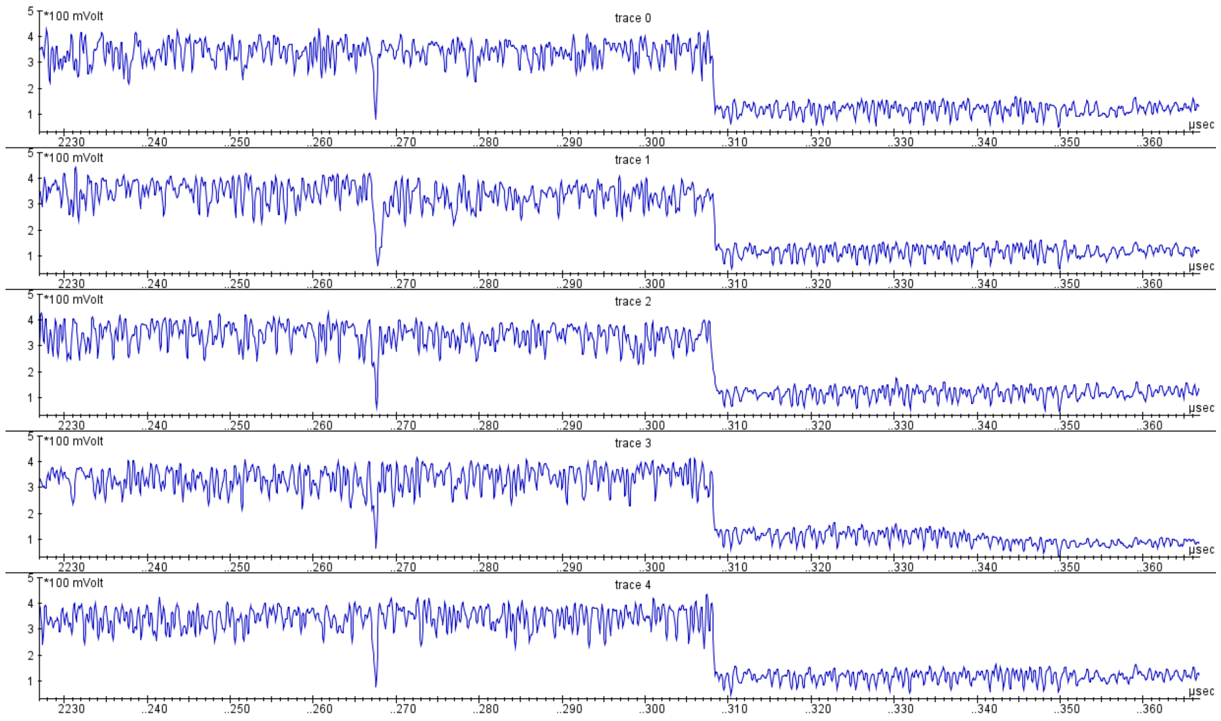

Figure 21: Timing Consistencies of the Baseline Configuration

randomized radix encoding Booth multiplier configuration uses static binary exponentiation built upon a Booth multiplier that randomly selects between radix of 2, 4, or 8. The raw trace shown in Figure 22 has higher power spikes than the baseline configuration. This higher power consumption is due to multiple copies of the ALU hardware needed to compute the additional operands seen in the higher radix columns of Table 5. These power spikes can be seen more easily in Figure 23 after removing noise from the trace via a bandpass filter centered on the 12.5 mHz clock frequency. Zooming in to examine the first few bits of encryption, shown in Figure 24, the power spikes clearly show the hardware is front loaded and high switching activity draws more power as soon as new operands are shifted into the registers.

In order to show the misalignment of traces, Figure 25 presents the ending times of five power traces of the randomized multiplier configuration. In this sample of traces, the misalignment is very apparent when compared to the baseline ending times previously shown in Figure 21. Taking the standard deviation of 10,000 col-

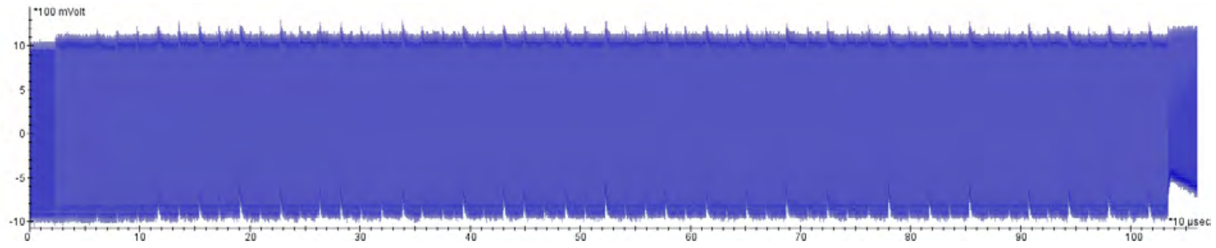

Figure 22: Randomized Multiplier Configuration - Raw EM Power Trace

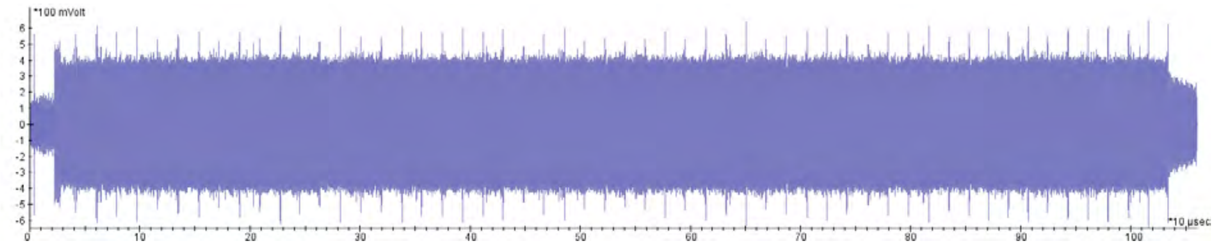

Figure 23: Randomized Multiplier Configuration - Filtered Trace

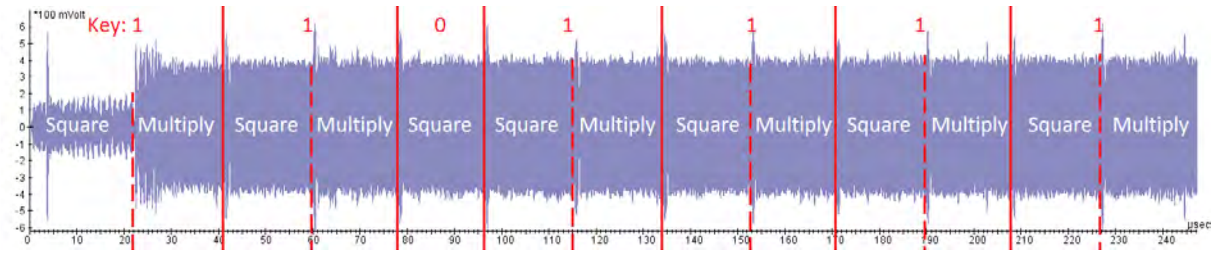

Figure 24: Randomized Multiplier Configuration - Zoomed, Filtered

lected traces, Figure 26 shows a normal distribution of ending times from $1,027\mu s$ to $1,043\mu s$. This distribution is as expected and reflects the system randomly selecting a radix each iteration. The majority of traces have a mixture of high radix and low radix iterations, while few traces randomly select mostly high radix (i.e., shorter overall runtime) or mostly low radix (i.e., longer overall runtime). Thus, a normal distribution of end times is expected.

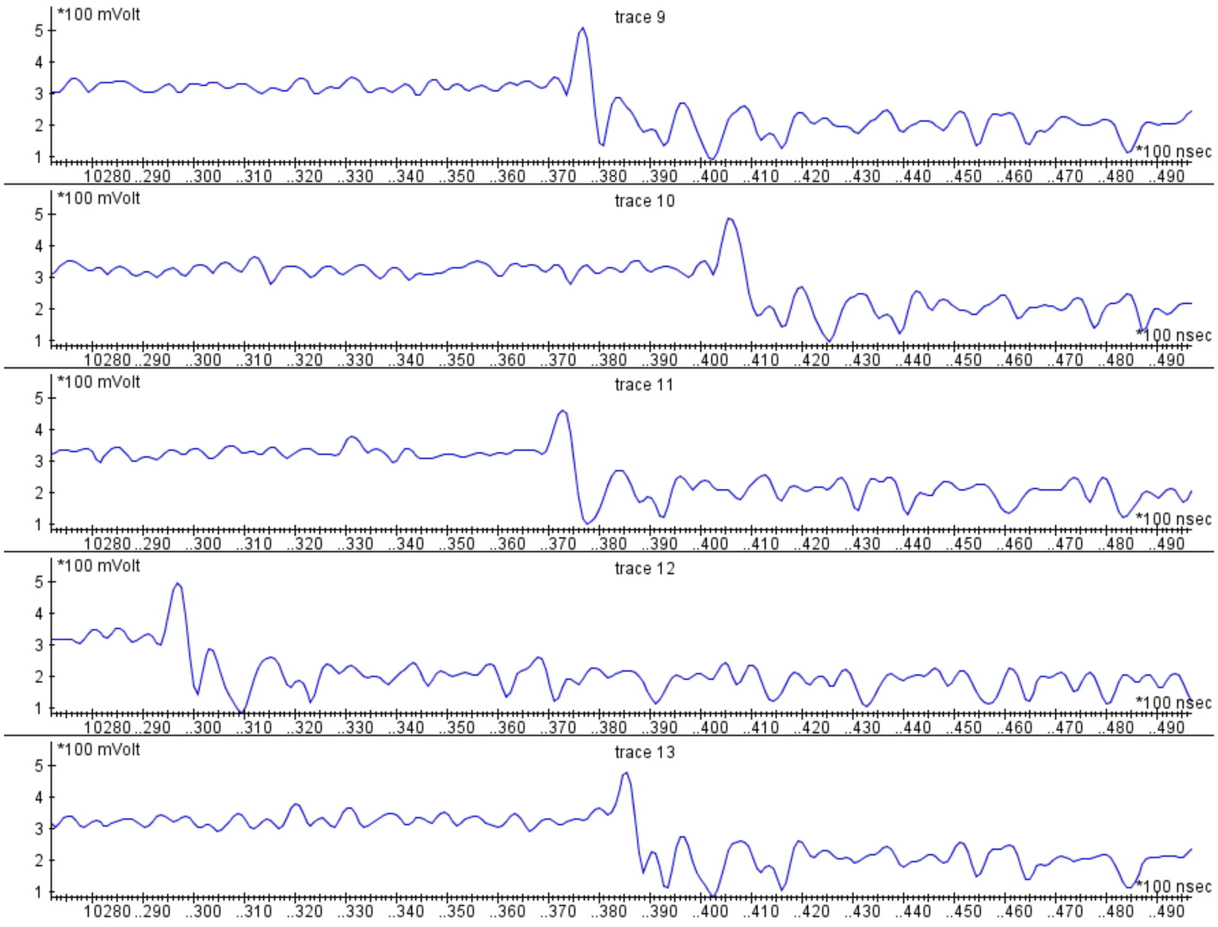

Figure 25: Timing Misalignment of the Randomized Multiplier Configuration

4.4.3 Random Exponentation Window Configuration. The next section details the hardware test for the variable windowing configuration. This configuration uses static 2-radix Booth multiplication, but varies the exponentiation window size between 1, 2, or 3. The raw power trace, shown in Figure 27, appears to be more sim-

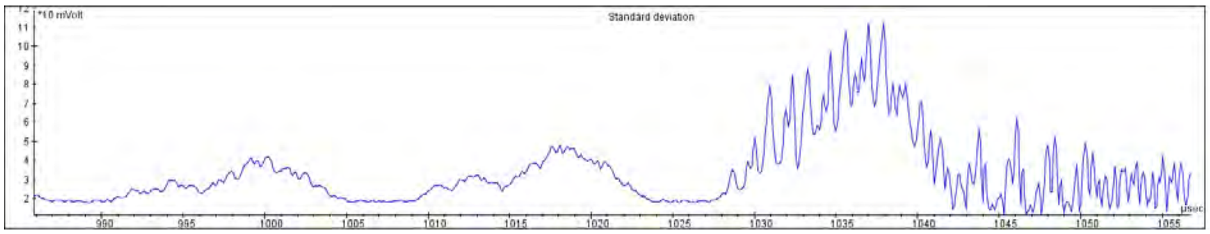

Figure 26: Randomized Multiplier Configuration - 10,000 Trace Standard Deviation

ilar to the baseline configuration than the randomized multiplier configuration. This similarity is due to the underlying hardware. The random window configuration, like the baseline configuration, contains only radix-2 Booth multiplier hardware. It is expected that the random window power trace is most similar to the baseline configuration since the random window countermeasure implements randomization at the finite state machine (FSM) level, not at the low level multiplication hardware. After applying the same bandpass filter to reduce noise, Figure 28 shows the resultant power trace. Notice the dip in power consumption starting at $250\mu s$. The operations before this dip are the precalculations (i.e., the first loop of Algorithm 4). The dip in power is due to the first few operations following initializing the variable $R = 1$. Squaring R at this point drives very little switching activity, thus draws very little power consumption (e.g., $R^2 = R * R = 1 * 1 = 1$). This effect can more clearly be seen in Figure 29. Note that window size of 1 requires R^2 before multiplying, where window sizes of 2 and 3 require calculating R^4 and R^8 respectively. Figure 29 also displays the randomness in window size selection.

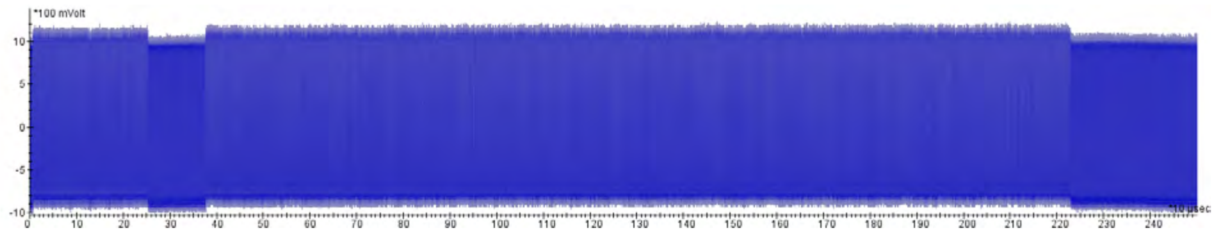

Figure 27: Randomized Window Configuration - Raw Power Trace

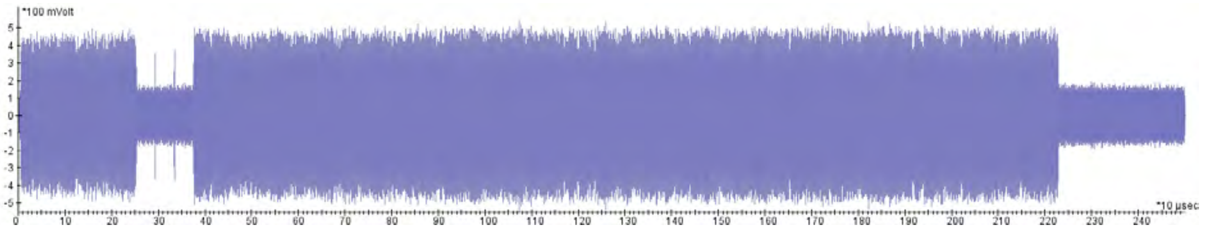

Figure 28: Randomized Window Configuration - Filtered Trace

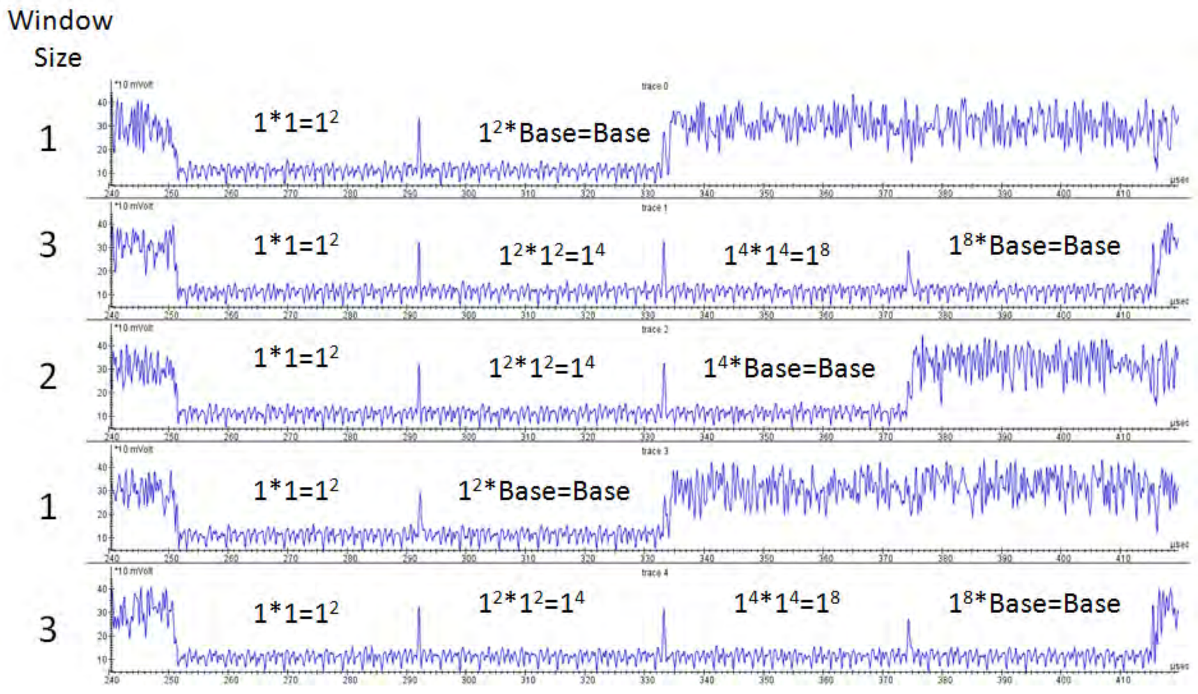

Figure 29: Comparison of Initial Calculations in Random Window Configuration

Once again, it is interesting to examine the ending times of this configuration. Figure 30 shows the much increased timing variance of this configuration. Notice that while the ending times are different, there is no misalignment; the deep downward power spikes are all aligned. However, the number of execution "blocks" the traces takes to complete is randomized. Looking at the standard deviation of 10,000 collected traces, Figure 31 shows a normal distribution of ending times from $2,060\mu s$ to $2,350\mu s$. Notice the randomized window configuration has a timing variance of $300\mu s$ where as the randomized multiplier only generates $15\mu s$ of variance. This increased timing variance will make alignment and correlation more difficult in SCA attacks.

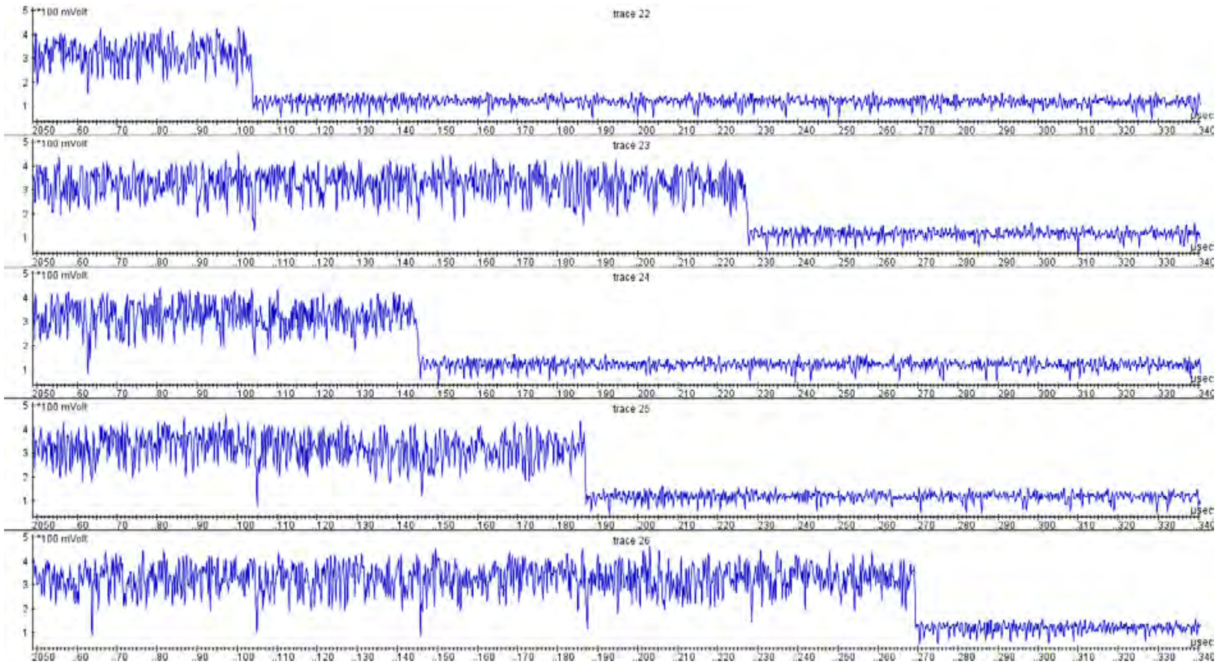

Figure 30: Timing Misalignment of the Randomized Window Configuration

4.4.4 Combined Countermeasure Configuration. This dynamic countermeasure combines the randomized radix encoding Booth multiplier hardware with the randomized exponentiation window architecture. The raw power trace shown in Figure 32 looks similar to both the independent countermeasures. It exhibits

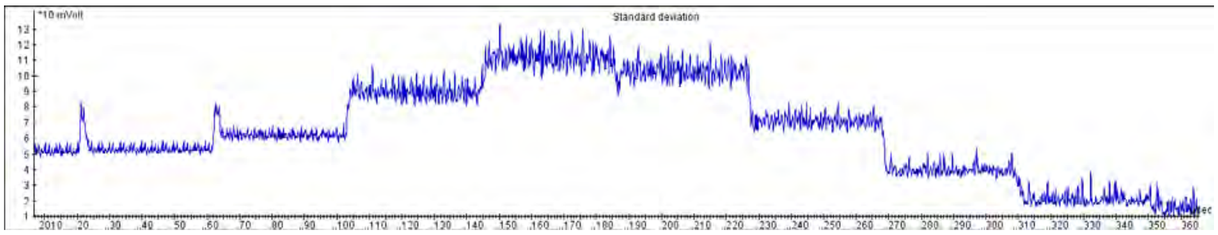

Figure 31: Randomized Window Configuration - 10,000 Trace Standard Deviation

higher power spikes from the increased randomized Booth hardware, as well as a dip in power immediately following the precomputations. These attributes can be seen more clearly in the filtered power trace in Figure 33.

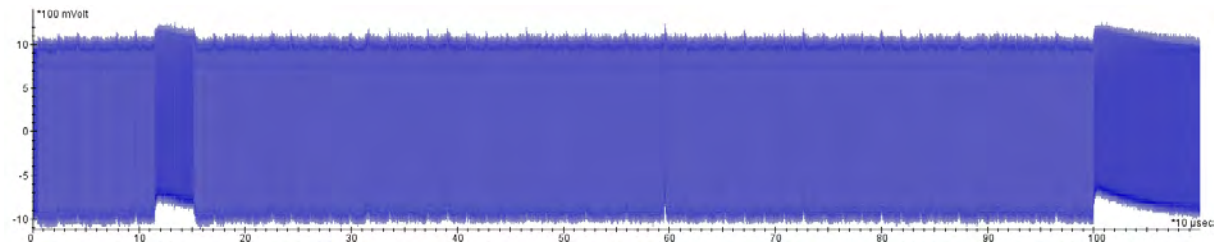

Figure 32: Combined Countermeasure Configuration - Raw Power Trace

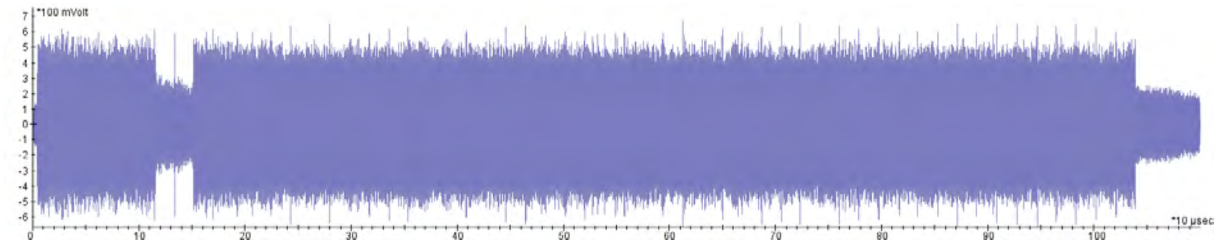

Figure 33: Combined Countermeasure Configuration - Filtered Trace

Once again, the ending times of several power traces are examined to comment on trace misalignment and timing variance. Figure 34 presents the ending times of the combined countermeasure. While ending times appear very similar to the variable window configuration, the addition of the randomized multiplier has added low level trace misalignment. Notice that the large power spikes are no longer aligned as they were with only the random window configuration. Similarly to previously shown, the standard deviation of 10,000 traces yields a normal distribution of ending times and can be seen in Figure 35. While the combined countermeasure produces a large variance in ending times like the random window configuration, the addition of the randomized multiplier “smooths” the edges such that bell curve is more evenly distributed.

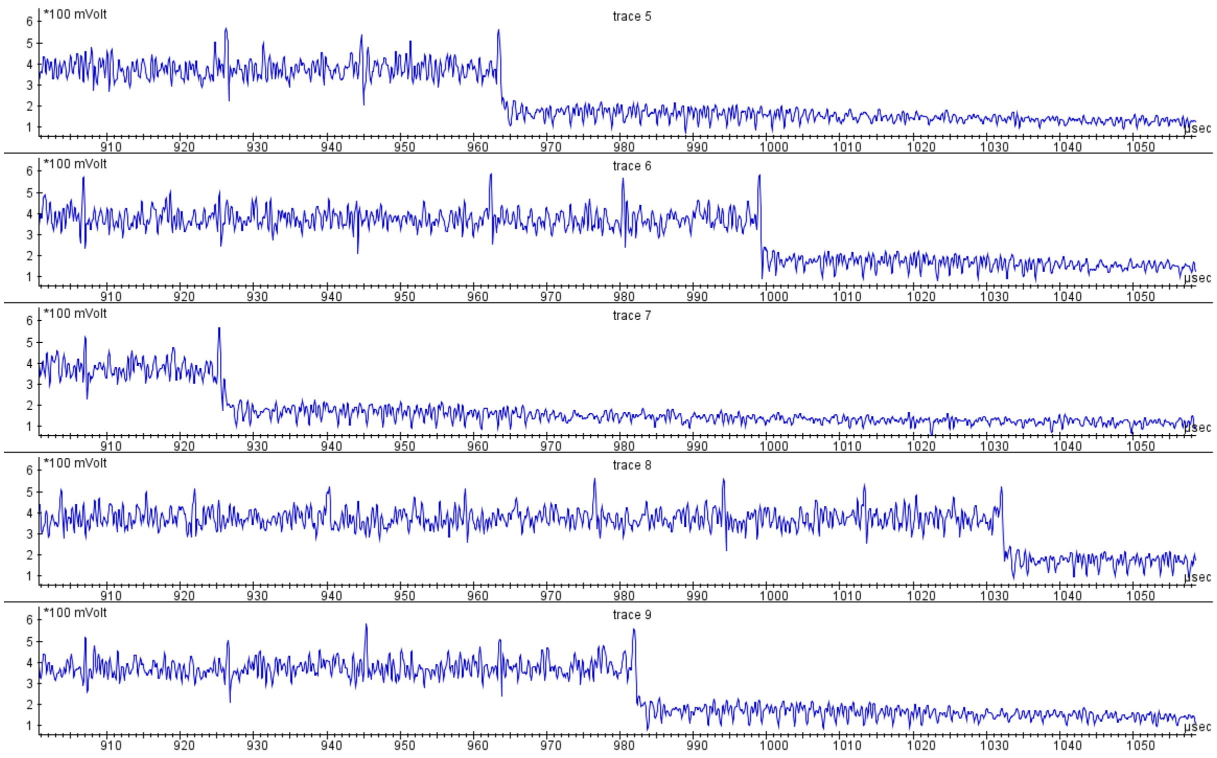

Figure 34: Timing Misalignment of the Combined Countermeasure Configuration

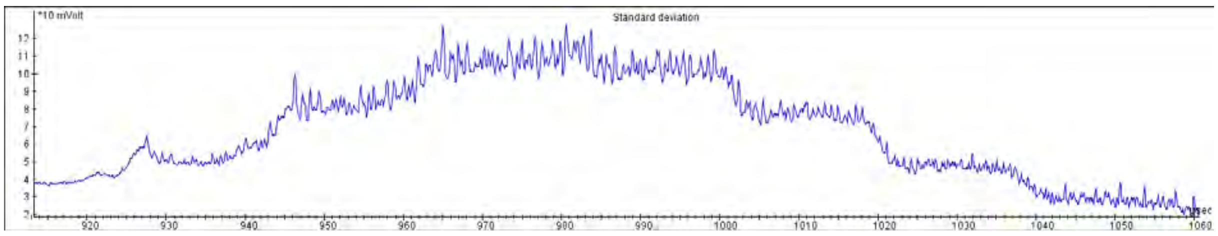

Figure 35: Combined Countermeasure Configuration - 10,000 Trace Standard Deviation

4.5 Hardware Attack Results

The following section details the development and results from conducting a real world side channel power analysis attack. This attack is a correlation power analysis attack, as previously described in Section 2.2.2.2. First, a collection of 10,000 power traces with random ciphertexts are decrypted. During decryption, EM probes are used to collect emissions related to power consumption. These emissions and ciphertexts are saved for processing the attack. This attack setup is identical for each configuration under attack. Next, a key bit guess is made, and an intermediate value is calculated based on this key guess. The Hamming Weight (HW) of 8-bits of the resultant intermediate guess is calculated. These 10,000 8-bit HW guesses are statistically correlated to the collected EM power traces using Equation 2 from Section 2.2.2.2. If the resultant correlation trace contains a correlation spike, the power consumption matches the HW values and the key guess is verified as correct. If no correlation spike is seen, the key bit guess is incorrect.

4.5.1 Attack On Previous AFIT Research. This research is a continuation of previous work [1]. To begin working with SCA, the previous work was attacked and broken to prove attack validity. A power trace from the previous research's implementation is shown in Figure 36. Note that this implementation is also vulnerable to simple power analysis. The result of the correlation power analysis attack on the previous research's implementation can be seen in Figure 37. This attack generates a very large correlation spike many times larger than the noise floor. This very successful attack gives an attacker high confidence that the correct bit has been guessed.

4.5.2 Attack on Baseline Configuration. The first implementation from this research to be attacked is the baseline configuration. As expected, correlation power analysis is able to easily recover the key. Without any randomizations of intermediate values, randomizations in power, or trace misalignments, the attack is

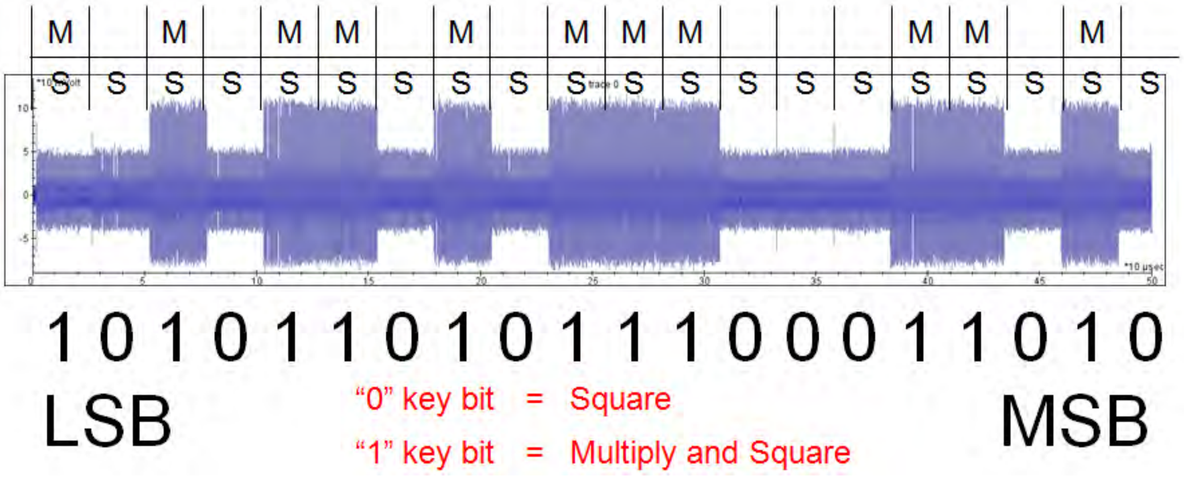

Figure 36: Previous Work's SPA Susceptible

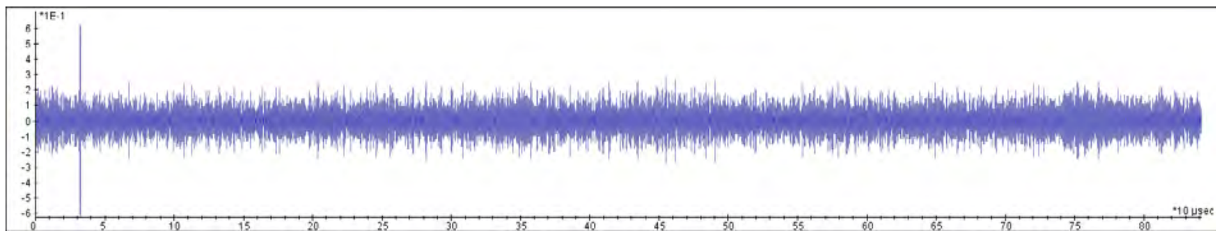

Figure 37: Previous Work Broken Using Correlation Power Analysis

successful as seen in Figure 38. To provide the reader an idea of an unsuccessful attack, Figure 39 presents the correlation trace with an incorrect key guess. Notice the noise floor remains the same but a correlation spike is not present. Therefore, the baseline configuration is not protected against correlation power analysis.

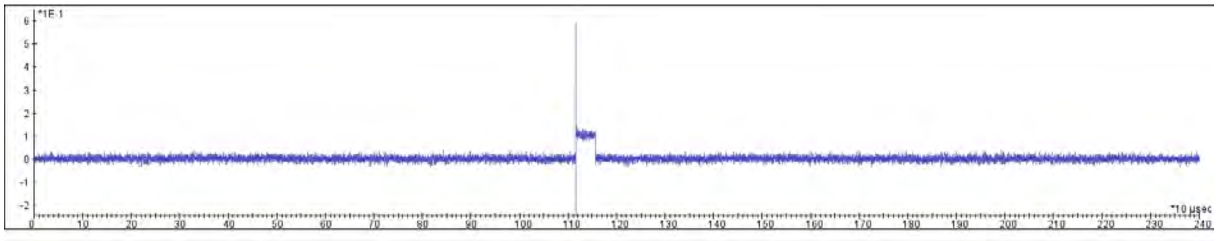

Figure 38: Correlation Attack on Baseline Configuration - Correct Guess

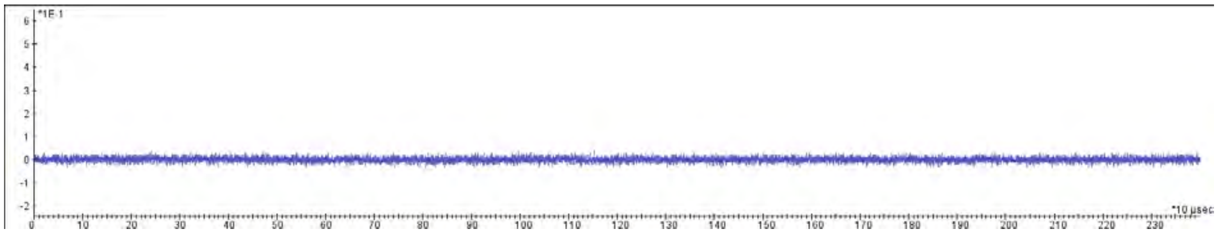

Figure 39: Correlation Attack on Baseline Configuration - Incorrect Guess

4.5.3 Attack On Randomized Radix Encoding Booth Multiplier Configuration. Since the randomized multiplier configuration has elements of low level misalignment and randomization, the effectiveness of the SCA attack is expected to be degraded. Looking at Figure 40, the attack still generates a correlation spike. However, since there is timing randomness the correlation is spread out among many clock cycles. This causes the correlation spike to be shorter and wider instead of the narrow tall spike seen in the attack against the baseline configuration. For an "apples to apples" comparison, the correlation trace is set to same y-axis scale as the baseline attack, Figure 41. It is now much more apparent that the spike is smaller than the baseline attack. Although the randomized multiplier countermeasure significantly reduced the correlation spike, the spike is still much larger than the noise floor. Thus, the attack is successful and this countermeasure is not protected against correlation power analysis attacks.

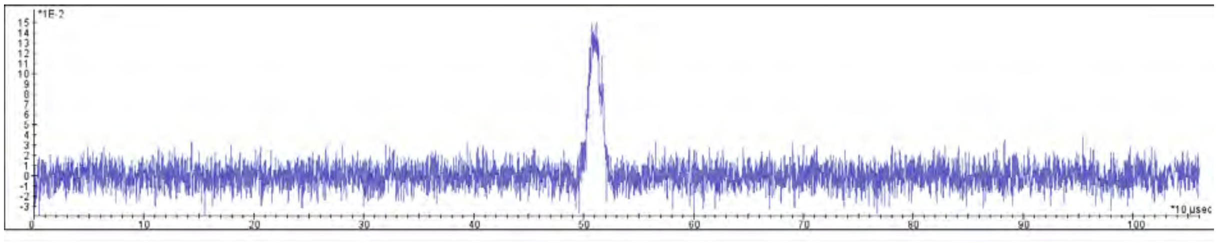

Figure 40: Randomized Multiplier Configuration - Correct Guess

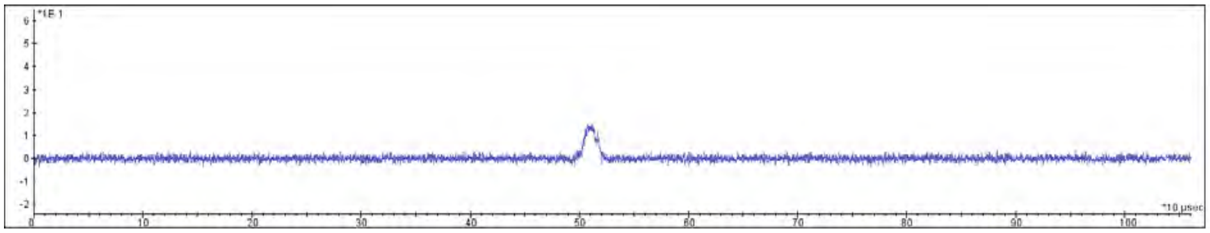

Figure 41: Correlation Spike on Baseline Y-Axis Scale

4.5.4 Attack On Random Exponentiation Window Configuration. Based on the ending times variance examined previously, the random window configuration is expected to provide increased protection. Figure 42 shows a small area of increased correlation near the $1,200\mu s$ mark. This correlation area is much more disperse than the previous two configurations, indicating that the random window configuration provides the best protection so far. Comparing the attack results on the same y-axis scale of the baseline attack, Figure 43 shows it is extremely difficult to make out the correlation near the $1,200\mu s$ mark. While this configuration is still considered vulnerable in this 10,000 trace correlation power analysis attack, the configuration would be considered protected if it were possible to limit the amount of power traces an attacker could collect to a smaller number.

4.5.5 Attack On Combined Countermeasure Configuration. Combining the low level randomness of the random multiplier with the architectural randomness of

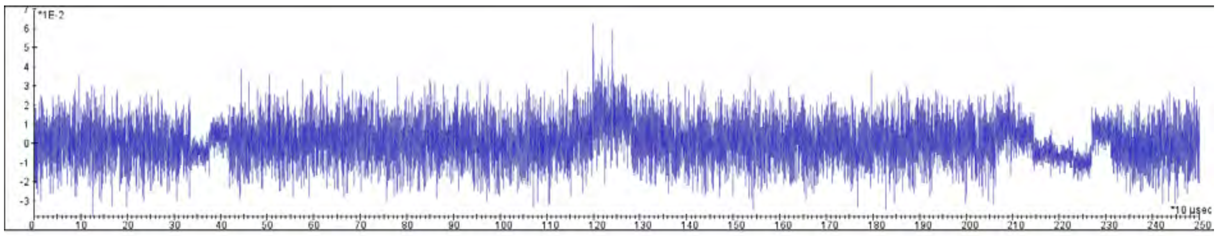

Figure 42: Random Exponentiation Window Configuration - Correct Guess

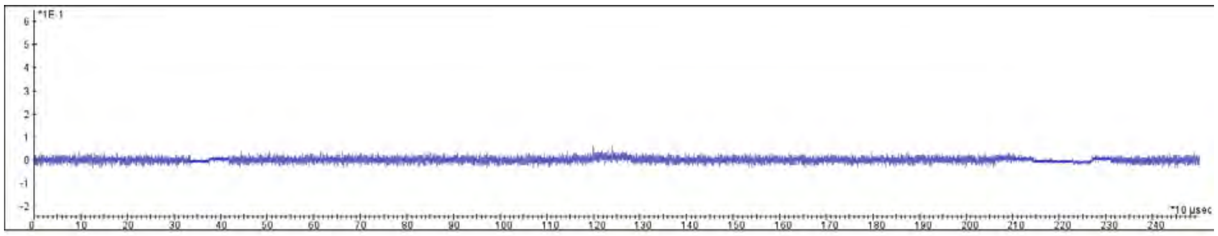

Figure 43: Correlation Spike on Baseline Y-Axis Scale

the random window exponentiatior proves to be the best chance at protection. Figure 44 presents the results from attacking the combined countermeasure configuration. There is no correlation discernible from the noise floor as expected near the $500\mu s$ mark. Because the correlation cannot be distinguished from the noise floor in the correlation trace, this configuration is considered protected in this attack. Furthermore, when compared on the same y-axis scale of the baseline attack (Figure 45), this result looks nearly identical to the incorrect key guess trace previously shown in Figure 39, indicating the implementation is protected. This configuration is declared protected against the 10,000 trace correlation power analysis attack. In an effort to uncover the breaking point of this countermeasure the number of traces collected was increased by an order of magnitude. Requiring about 48 hours of constant CPU processor time, 100,000 traces were collected, filtered, and attacked. The results from this 100,000 trace correlation power analysis attack are shown in Figure 46. There is no discernible difference in the correct vs. incorrect key guess in this attack. Therefore, the combined countermeasure is also considered protected against a 100,000 trace correlation power analysis attack. However, an attacker able to collect many orders of magnitude more traces may be able to reduce the noise floor such that the correlation is discernible, and the combined countermeasure configuration is no longer protected. Although, collection of traces can be limited to reasonable amounts with sufficiently often key changes.

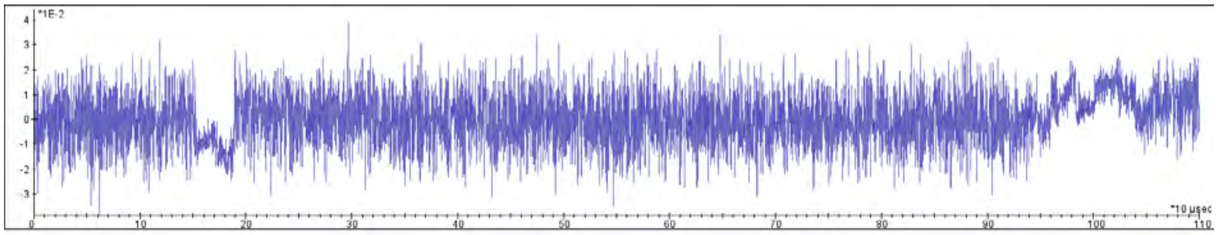

Figure 44: Combined Countermeasure Configuration - Correct Guess

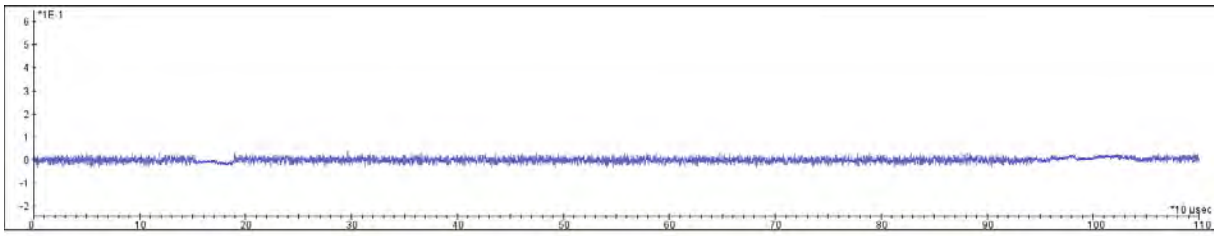

Figure 45: Correlation Spike on Baseline Y-Axis Scale

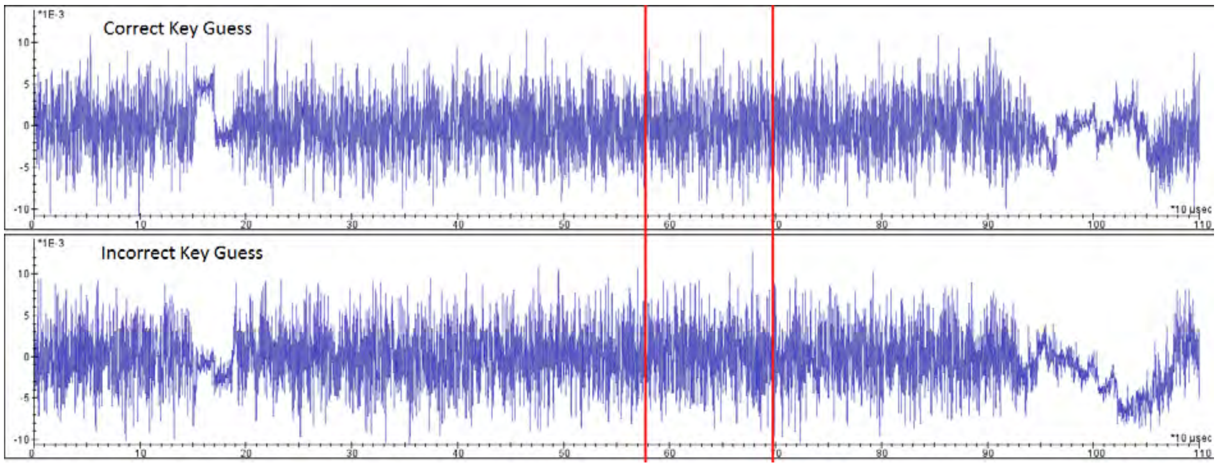

Figure 46: Combined Countermeasure Configuration - 100,000 Trace Attack

4.5.6 Comparisons. In order to clearly see each configuration's effect on the attack, Figures 47 and 48 compare the results of attacks on independent and identical y-axis scales, respectively. While the correlation is not discernible in these countermeasures, it is not hidden; the correlation is simply misaligned. This research seeks to force attackers to attempt a brute force type correlation attack. If an attacker knew each random window size and radix, they could easily align/filter traces to reveal the correlation. Because these windows and radixes are randomized, an attacker would be forced to attempt correlation on all possible combinations of window sizes and radixes (i.e., brute force) for each key bit. Future advances in the field could produce a countermeasure with side channel brute force security better than the brute force security of the underlying encryption scheme.

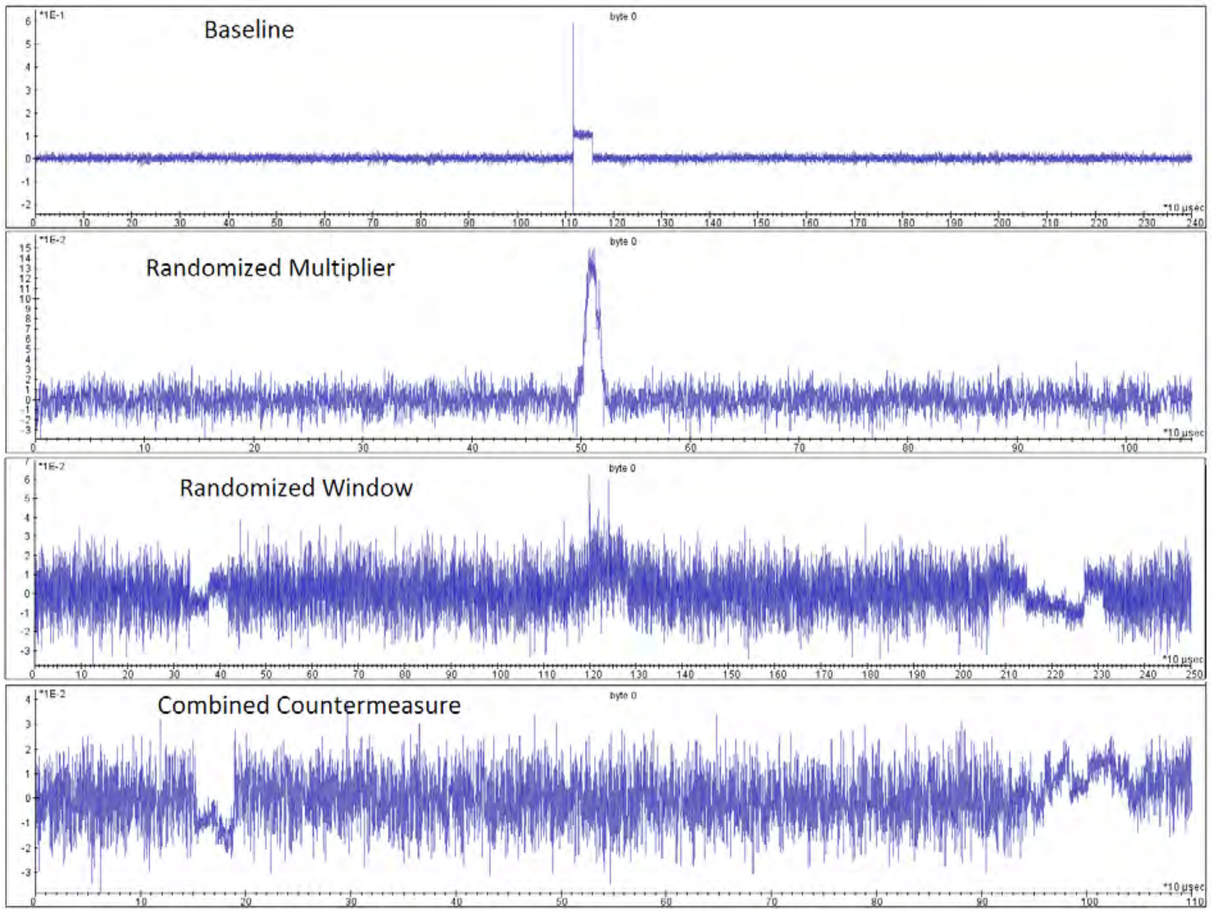

Figure 47: 10,000 Trace Correlation Power Analysis Attack Compared

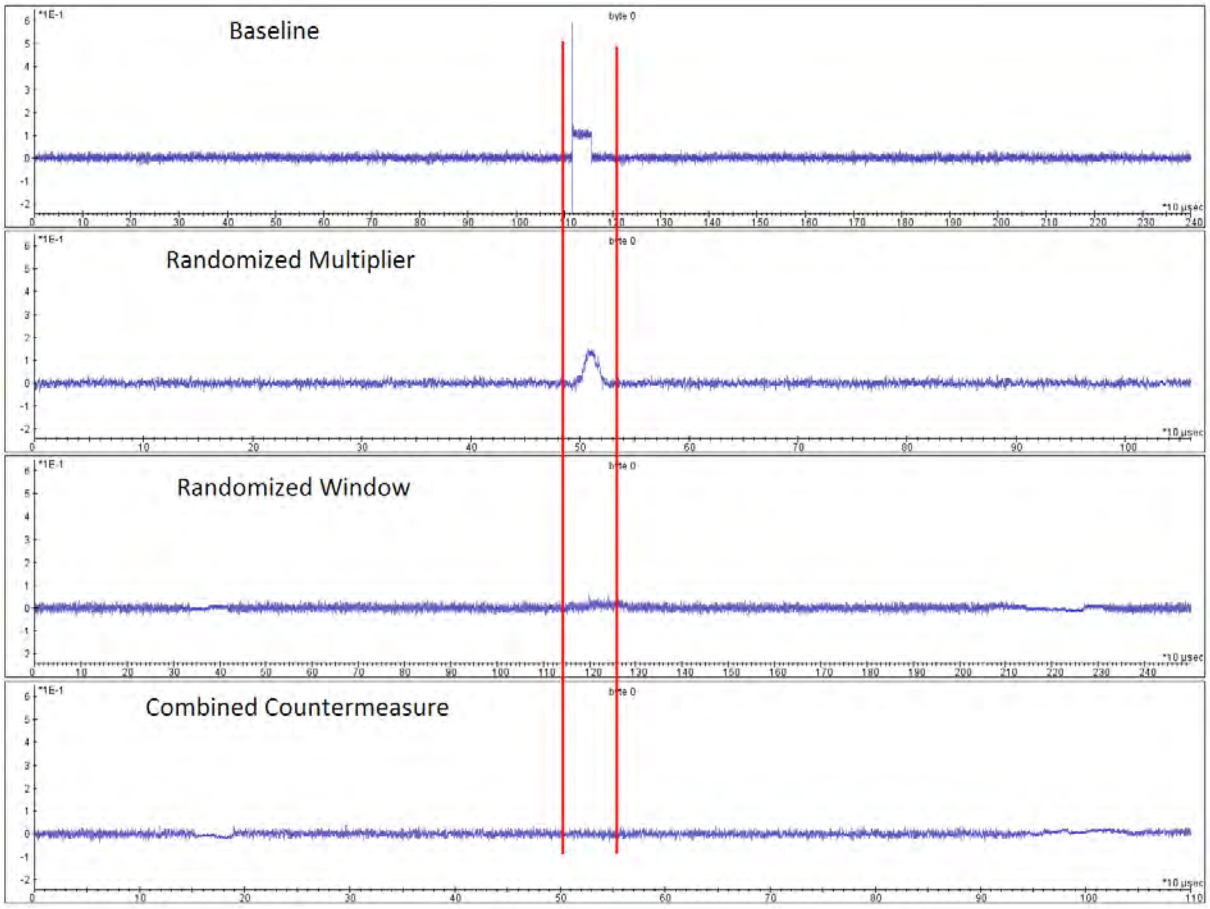

Figure 48: 10,000 Trace Correlation Power Analysis Attack Compared - Same Scale

4.6 Metrics

Regardless of a countermeasure's security, no implementation is realizable if it does not meet the users needs. A perfectly secure system is useless if it is too large to be manufactured or cannot provide the deciphered information in a timely manner. This section presents the results of the following metrics: execution time, FPGA area size, and level of increased protection.

4.6.1 Execution Time (ET).

The first metric examined is execution time. Any countermeasure that increases encryption/decryption execution time decreases throughput. All configurations in this research are synthesized and running at 12.5 mHz. Although Table 8 shows that the randomized multiplier and combined countermeasure contain more path delays and must be run at the slower clock rate, there is actually very little performance penalties involved for these countermeasures. Taking into account the execution times shown in Figure 49, the 55% speedup achieved by the randomized multiplier and combined countermeasure offset the 50% decrease in clock frequency.

Table 8: Table of Maximum Frequency of Configurations

Configuration	Maxiumum Frequency
Baseline	25.2 mHz
Random Multiplier	12.9 mHz
Random Window	26.4 mHz
Combined Countermeasure	13.7 mHz

4.6.2 Required FPGA Area.

The next metric discussed is FPGA area required. Area and execution time are common inverse tradeoffs. A system may be designed to operate quicker with parallel hardware, but this comes at an increased layout area size for the additional hardware. As detailed in Table 9, the randomized multiplier and randomized window countermeasure affect different aspects of area. Because the randomized multiplier has many additional ALUs, the multiplier requires many more look up tables (LUTs) to make these calculations. However, the

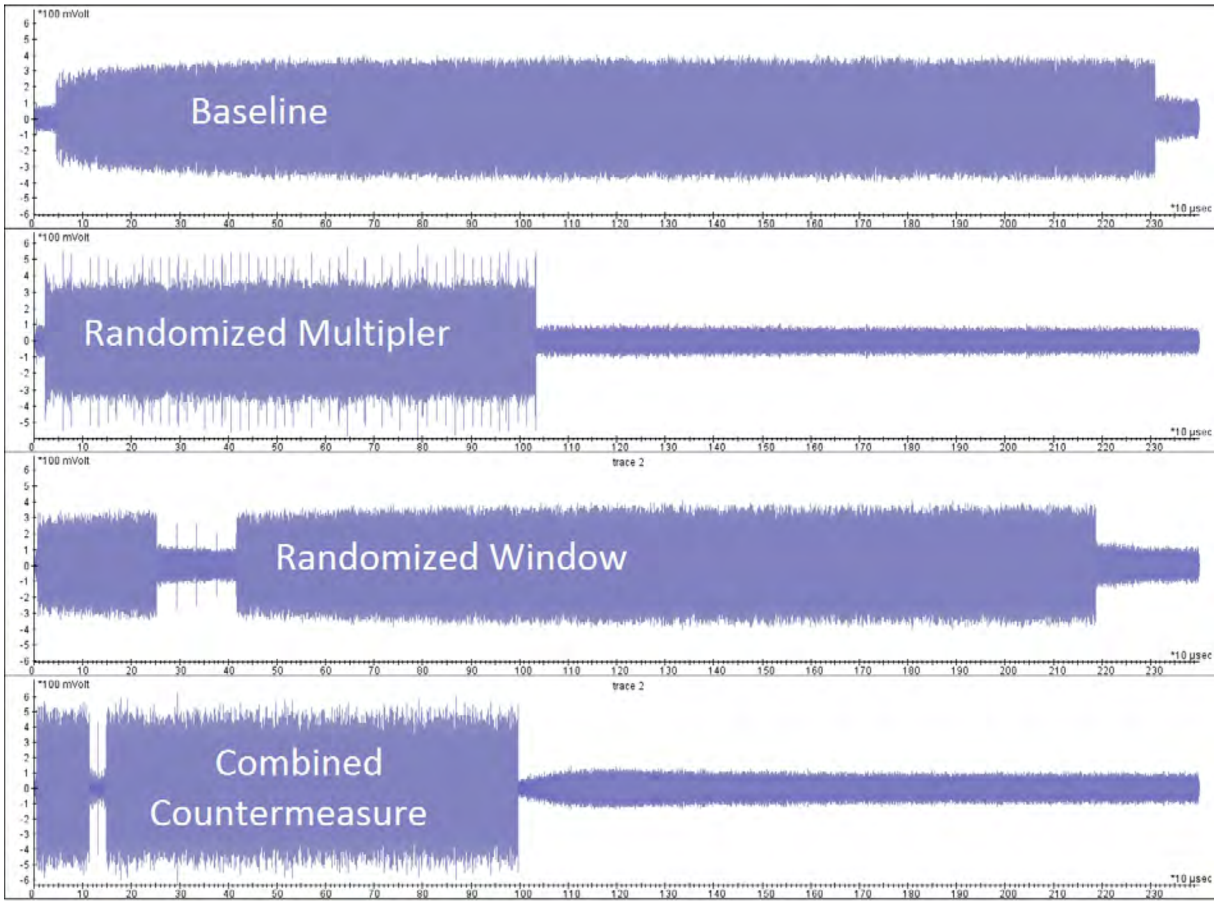

Figure 49: Execution Time of Configurations Compared (Running At 12.5 mHz)

randomized window countermeasure requires additional precomputations and storage. This increased storage drives an increase in flip flops (FFs). As expected, the combined countermeasure contains the combined increases in both FFs and LUTs.

Table 9: Table of FPGA Area Required For Each Configuration

Configuration	Flip-Flops	LUTs	Increased FF	Increased LUTs
Baseline	10425	14791	N/A	N/A
Random Multiplier	10462	29827	0.4%	101%
Random Window	13576	18075	30%	22%
Combined Countermeasure	13613	34156	31%	131%

4.6.3 Protection. Protection, as defined for this research, is measured as the number of power traces required to recover secret key bits. Based on the 10,000 trace correlation power analysis attack presented in the last section, the following graphs detail how the correlation spike and noise floor change as more traces are collected. Recalling the relationship presented in Equation 1 from Section 2.2.2.1, increasing the number of traces collected decreases noise and increases SNR of the desired leakages. Examining the baseline configuration, Figure 50 reveals that beyond 30 traces the noise floor is reduced to a level that reveals the correlation. Attacks attempted with less than 30 traces will be unsuccessful as it will be impossible to discern the correlation spike from the noise floor. However, attacks with greater than 30 traces will be successful because an attacker can distinguish the correlation spike in correct key bit guesses. Also notice that the correlation spike neither increases nor decreases as excess power traces are collected, the noise floor simply shrinks further. This reduction is due to the static nature of the implementation. There is a constant amount of leakage. An attacker only needs to collect enough traces to lower the noise floor to a level that allows distinguishment of the correlation spike. This demonstrated inherent weakness plagues all static countermeasures, such as dual-rail logic.

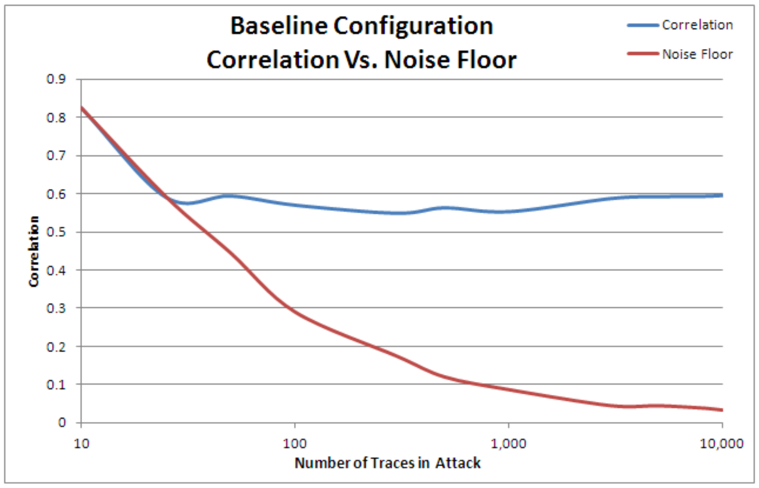

Figure 50: Baseline - Correlation Vs. Noise Floor

The next configuration inspected is the randomized multiplier. Figure 51 shows some increased protection. This configuration requires 80 power traces for successful attack. Notice how the correlation no longer remains at a constant level as more traces are collected. The changing power consumption and misalignments introduced by the random multiplier continually decrease correlation. Although the noise floor was able to be reduced at a steeper slope, revealing the correlation, this tenet of dynamic countermeasures highlights the promise of their use for SCA obfuscations.

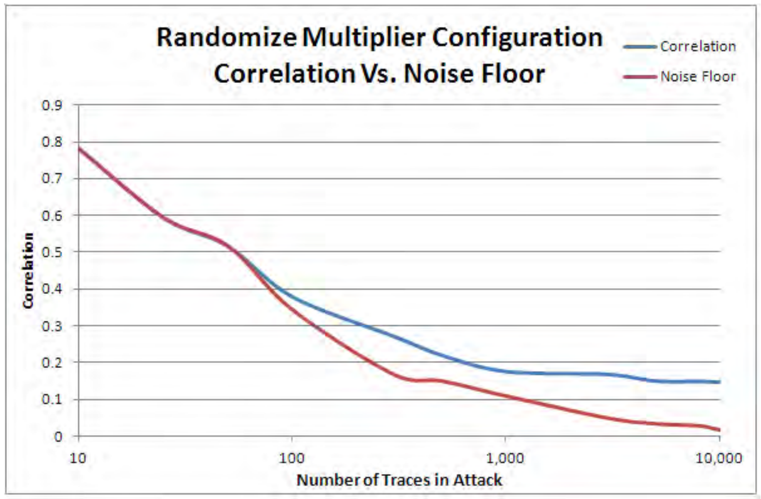

Figure 51: Randomized Multiplier - Correlation Vs. Noise Floor

Moving on to the randomized window configuration, Figure 52 presents the results of the attack. It is immediately apparent that the randomized window configuration is able to maintain correlation below the noise floor up until 2,000 traces. The increased timing variance previously shown in Figure 31 allows this configuration to spread the correlation over a larger time frame, which decreases its impact.

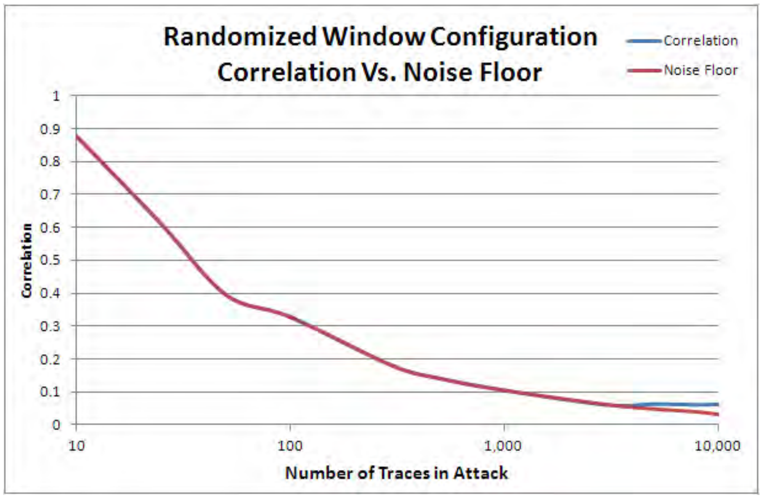

Figure 52: Randomized Window - Correlation Vs. Noise Floor

Next, the results of an attack on the combined countermeasure are examined, Figure 53. Within the original 10,000 trace attack, it is not possible to distinguish the correlation. Even when the attack was increased by an order of magnitude to 100,000 traces, no correlation was revealed. For the attack setups presented in this research, the combined countermeasure is considered protected. However, if the amount of traces used in the attack was greatly increased, it would likely reveal the correlation at some point.

Finally, Table 10 is presented to summarize the increased levels of protection of each configuration. This research concludes that the only major trade off penalty for this countermeasure's increased protection is over doubling the required LUTs. Notice that the increase in FFs and LUTs for the combined countermeasure is only a summation of the two individual countermeasure, but the increase in protection is many times greater than their summation. Combining the two forms of random-

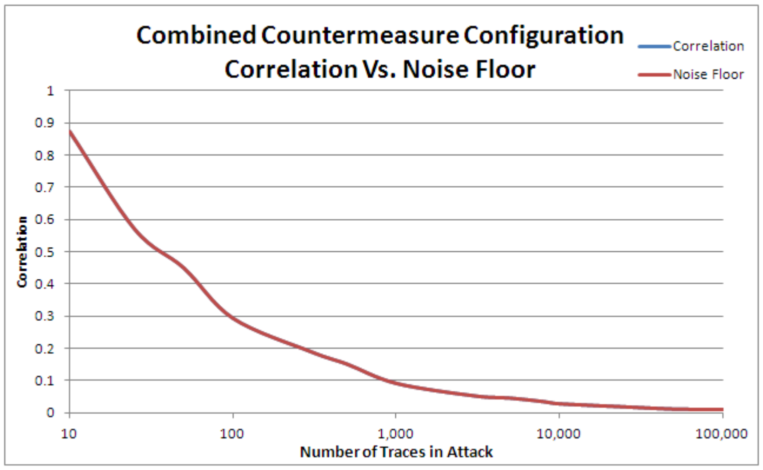

Figure 53: Combined Countermeasure - Correlation Vs. Noise Floor

ness (low level hardware randomness and FSM randomness) creates an synergistic increase in protection.

Table 10: Table of Number of Traces Required for Successful Attack

Configuration	Number of Traces	Increased Protection
Baseline	30	N/A
Random Multiplier	80	166%
Random Window	2,000	6,500%
Combined Countermeasure	+100,000	+300,000%

4.7 Conclusions

Table 11 is presented to summarize the increased protection and associated trade-offs of each configuration.

Table 11: Table of Tradeoffs

Configuration	Increased FFs	Increased LUTS	Increased ET	Increased Protection
Baseline	N/A	N/A	N/A	N/A
Random Multiplier	0.4%	101%	Negligible	166%
Random Window	30%	22%	Negligible	6,500%
Combined Countermeasure	31%	131%	Negligible	+300,000%

As stated previously, this research is driven by the fact that current static SCA countermeasures rely on hiding the signals, not protecting them. These countermeasures' success are based upon the quality of equipment and quality of signal an attacker is able to capture. In contrast to the existing countermeasures, the dynamic architectural countermeasure presented in this paper does not rely on hiding signals among noise. This approach is much more in line with Kerckhoffs's principle [10], which states that a cryptosystem should be secure even if everything about the system, except the key, is public knowledge. The new countermeasure presented in this paper makes no attempt to hide power consumption, only to randomize it. Because the signal is not hidden beneath noise, the probability of successful attack is not determined by the sophistication of an attacker's equipment or quality of signal they capture. This randomized approach forces an attacker into a brute force side channel attack in which the attacker must calculate every possible combination of intermediate values for every trace. Therefore, this research lays a foundation for exponential difficulty side channel attack protection. Assuming the correct trade off choices are made with respect to performance and hardware area, it is conceivable that large enough pools of radixes and window sizes could lead to a future system with brute force side channel attack difficulty no worse than the underlying encryption algorithm's brute force security, thus rendering side channel attacks impractical.

V. *Conclusion*

The following chapter summarizes this research's completed objectives, contributions to the field of study, and options for future work.

5.1 Completed Objectives

The objectives laid out for this research are obtained as follows:

5.1.1 Dynamic RSA Design.

The primary objective of this research is to further the side channel attack countermeasure field of study. This objective is achieved through design of a dynamic architectural countermeasure to protect RSA encryption against side channel attacks. Unlike many popular static countermeasures that merely hide the leakage beneath noise, this dynamic countermeasure randomizes timing and power consumption such that correlation is much more difficult. A randomized radix encoding Booth multiplier induces lower level hardware timing misalignments and randomizes power. Building upon the randomized multiplication, a randomized window exponentiatior further misaligns the traces and randomizes intermediate calculated values. The combined countermeasures produced by this research lay a foundation for future SCA countermeasures that may have side channel attack difficulty of at least the brute force security of the underlying encryption scheme, thus rendering SCA attacks useless.

5.1.2 Implementation on FPGA.

The newly designed dynamic countermeasure was implemented to prove viability and testing. This implementation was completed on a Xilinx Virtix-5 FPGA. VHDL code was synthesized using Xilinx's XPS design suite and downloaded to the FPGA. The 3rd party Riscure Inspector software was used to facilitate key passing and random plaintext generation. Traces were compared against the key values and resulting ciphertext to verify correct implementation and encryption results.

5.1.3 Real World Attacks. The last objective of this research is to conduct real world attacks on the countermeasures. These attacks are the ultimate verification and provide quantifiable levels of increased protection. Correlation power analysis attacks were conducted and the results verify the increases in protection for each countermeasure designed.

5.2 Contributions

The completed research objectives provide numerous contributions to the field of side channel attack protection.

5.2.1 Developed Theoretical Countermeasure Verified by Attacks. This research lays the groundwork for new dynamic algorithm class of SCA countermeasures. Not only developing a theoretical model for the countermeasure, but also implementing them and conducting real world attacks to verify their validity.

5.2.2 Developed Custom MATLAB Script for Correlation Power Analysis (CPA) Attacks. This research developed custom MATLAB scripts capable of extracting secret keys. This contribution provides a method of SCA attack without relying on licensing of the 3rd party Riscure Inspector software. These MATLAB scripts not only mirror Inspector's capabilities, but also exceed them in attack automation.

5.2.3 Developed a Method of Comparison Power Analysis Attacks. A process of trace collection and post processing was developed to conduct CPA attacks. This contribution allows for another class of real world side channel attacks to be executed on in house implementations for testing.

5.3 Future Work

5.3.1 Develop Additional Attacks. Developing the capability to execute additional real world attacks in house would allow for more thorough validation of new countermeasures. Procuring the needed testbed to induce hardware faults would also allow future in house research to characterize fault attack tolerance of new countermeasures.

5.3.2 Speeding MATLAB Processing. Although the MATLAB scripts developed by this research are capable of completing attacks, the scripts run painfully slow. Riscure Inspector software uses Java classes optimized for very large integer (i.e. 512-bit) manipulation and uses CUDA cores on video cards to increase throughput. Mirroring the calculation speed of Inspector in MATLAB would allow for faster processing enabling more attacks to be attempted.

5.3.3 Brute Force Work. A major argument in favor of this dynamic countermeasure is the need for brute force search to realign the power traces for successful attack. A modified implementation of the countermeasure that could record all randomized selections would allow for a simulated brute force attack. Knowing each random choice at the time of attack would enable research into the exponential security of these countermeasures.

5.4 Summary

This research has designed, implemented, and validated a dynamic architectural countermeasures to increase protection of RSA encryption from side channel power analysis attacks. The combined countermeasure increases the number of traces required for successful attack (i.e., increases protection) by over 300,000% without suffering any throughput penalties. The cost for this tradeoff is only increased FPGA area. This research provides a foundation for future SCA countermeasures based on dyanimcally randomizing power at all levels of an implementation's architecture that may one day provide side channel attack protection greater than the underlying encryption algorithm's brute force security, thus rendering side channel attacks impractical.

Bibliography

1. Falkinburg, J., *Dynamic Polymorphic Reconfiguration To Effectively Cloak A Circuit's Function*, Master's thesis, Air Force Institute of Technology, 2010.

2. Riscure, *Inspector Training Slides*.

3. Patterson, D. A. and Hennessy, J. L., *Computer organization & design: the hardware/software interface*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

4. Kocher, P., Jaffe, J., and Jun, B., "Differential Power Analysis," *Advances in Cryptology CRYPTO 99*, edited by M. Wiener, Vol. 1666 of *Lecture Notes in Computer Science*, Springer-Verlag, 1999, pp. 388–397.

5. Rivest, R. L., Shamir, A., and Adleman, L., "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, Vol. 21, February 1978, pp. 120–126.

6. Mangard, S., Oswald, E., and Popp, T., *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

7. Dierks, T., "Request for Comments: 5246," The Transport Layer Security (TLS) Protocol Version 1.2.

8. NSA, "NSA Suite B Cryptography," http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml.

9. Trappe, W. and Washington, L. C., *Introduction to Cryptography: With Coding Theory*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd ed., 2006.

10. Menezes, A. J., Oorschot, P. C. V., Vanstone, S. A., and Rivest, R. L., *Handbook of Applied Cryptography*, 1997.

11. Fouque, P. A. and Valette, F., "The Doubling Attack Why Upwards is Better Than Downwards," *Workshop on Cryptographic Hardware and Embedded Systems 2003 (CHES 2003), LNCS 2779*, Springer-Verlag, 2003, pp. 269–280.

12. Nedjah, N. and Mourelle, L., "Efficient Hardware for Modular Exponentiation using the Sliding-Window Method with Variable-Length Partitioning," *Proc. 9th Int. Conf. for Young Computer Scientists ICYCS 2008*, 2008, pp. 1980–1985.

13. Itoh, K., Yajima, J., Takenaka, M., and Torii, N., "DPA Countermeasures by Improving the Window Method," *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '02, Springer-Verlag, London, UK, UK, 2003, pp. 303–317.

14. Walter, C., "MIST : An Efficient, Randomized Exponentiation Algorithm for Resisting Power Analysis," *Topics in Cryptology CT-RSA 2002*, edited by B. Preneel, Vol. 2271 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2002, pp. 142–174, 10.1007/3-540-45760-7_5.

15. Miyamoto, A., Homma, N., Aoki, T., and Satoh, A., "Systematic Design of RSA Processors Based on High-Radix Montgomery Multipliers." *IEEE Trans. VLSI Syst.*, Vol. 19, No. 7, 2011, pp. 1136–1146.

16. Miyamoto, A., Homma, N., Aoki, T., and Satoh, A., "SPA against an FPGA-Based RSA Implementation with a High-Radix Montgomery Multiplier," *Proc. IEEE Int. Symp. Circuits and Systems ISCAS 2007*, 2007, pp. 1847–1850.

17. Homma, N., Aoki, T., and Satoh, A., "Electromagnetic information leakage for side-channel analysis of cryptographic modules," *Proc. IEEE Int Electromagnetic Compatibility (EMC) Symp*, 2010, pp. 97–102.

18. Messerges, T., "Using Second-Order Power Analysis to Attack DPA Resistant Software," *Cryptographic Hardware and Embedded Systems CHES 2000*, edited by e. Ko and C. Paar, Vol. 1965 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2000, pp. 27–78, 10.1007/3-540-44499-8_19.

19. Joye, M., Paillier, P., and Schoenmakers, B., "On Second-Order Differential Power Analysis," *Cryptographic Hardware and Embedded Systems CHES 2005*, edited by J. Rao and B. Sunar, Vol. 3659 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2005, pp. 293–308, 10.1007/11545262_22.

20. Kocher, P. C., "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," Springer-Verlag, 1996, pp. 104–113.

21. Homma, N., Miyamoto, A., Aoki, T., Satoh, A., and Samir, A., "Comparative Power Analysis of Modular Exponentiation Algorithms," *IEEE Trans. Comput.*, Vol. 59, June 2010, pp. 795–807.

22. Yen, S.-M., Lien, W.-C., Moon, S., and Ha, J., "Power Analysis by Exploiting Chosen Message and Internal Collisions Vulnerability of Checking Mechanism for RSA-Decryption," *Progress in Cryptology Mycrypt 2005*, edited by E. Dawson and S. Vaudenay, Vol. 3715 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2005, pp. 183–195, 10.1007/11554868_13.

23. Giraud, C., "An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis," *IEEE Trans. Comput.*, Vol. 55, September 2006, pp. 1116–1120.

24. Boneh, D., DeMillo, R. A., and Lipton, R. J., "On the importance of checking cryptographic protocols for faults," *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, Springer-Verlag, Berlin, Heidelberg, 1997, pp. 37–51.

25. Yen, S.-M. and Joye, M., “Checking before output may not be enough against fault-based cryptanalysis,” *Computers, IEEE Transactions on*, Vol. 49, No. 9, sep 2000, pp. 967 –970.

26. Walter, C., “Sliding Windows Succumbs to Big Mac Attack,” *Cryptographic Hardware and Embedded Systems CHES 2001*, edited by e. Ko, D. Naccache, and C. Paar, Vol. 2162 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2001, pp. 286–299, 10.1007/3-540-44709-1_24.

27. Ratanpal, G., Williams, R., and Blalock, T., “An on-chip signal suppression countermeasure to power analysis attacks,” *Dependable and Secure Computing, IEEE Transactions on*, Vol. 1, No. 3, july-sept. 2004, pp. 179 – 189.

28. Bayam, K. A. and Ors, B., “Differential Power Analysis resistant hardware implementation of the RSA cryptosystem,” *Proc. IEEE Int. Symp. Circuits and Systems ISCAS 2008*, 2008, pp. 3314–3317.

29. Fournaris, A. P., “Fault and simple power attack resistant RSA using Montgomery modular multiplication,” *Proc. IEEE Int Circuits and Systems (ISCAS) Symp*, 2010, pp. 1875–1878.

30. Coron, J.-S., “Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems,” *Cryptographic Hardware and Embedded Systems*, edited by e. Ko and C. Paar, Vol. 1717 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 1999, pp. 725–725, 10.1007/3-540-48059-5_25.

31. Montgomery, P. L., “Speeding the Pollard and Elliptic Curve Methods of Factorization,” *Math. Comp.*, Vol. 48, No. 177, 1987, pp. 243–264.

32. Joye, M., Koc, C. K., Paar, C., and Yen, S.-M., “The Montgomery Powering Ladder,” 2002.

33. Joye, M., “Highly Regular Right-to-Left Algorithms for Scalar Multiplication,” *Cryptographic Hardware and Embedded Systems - CHES 2007*, edited by P. Paillier and I. Verbauwhede, Vol. 4727 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2007, pp. 135–147, 10.1007/978-3-540-74735-2_10.

34. Zhang, Y., Zheng, X., and Peng, B., “A side-channel attack countermeasure based on segmented modular exponent randomizing in RSA cryptosystem,” *Proc. 11th IEEE Singapore Int. Conf. Communication Systems ICCS 2008*, 2008, pp. 148–151.

35. fang Jin, J., hong Lu, E., and wei Gao, X., “Resistance DPA of RSA on Smartcard,” *Proc. Fifth Int. Conf. Information Assurance and Security IAS ’09*, Vol. 2, 2009, pp. 406–409.

36. Messerges, T. S., Dabbish, E. A., and Sloan, R. H., “Investigations Of Power Analysis Attacks On Smartcards,” *Proceedings of the USENIX Work-*

shop on Smartcard Technology on USENIX Workshop on Smartcard Technology, USENIX Association, Berkeley, CA, USA, 1999, pp. 17–17.

37. Tiri, K. and Verbauwhede, I., "A digital design flow for secure integrated circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, Vol. 25, No. 7, july 2006, pp. 1197 –1208.

38. Mokari, A., Ghavami, B., and Pedram, H., "SCAR-FPGA : A Novel Side-Channel Attack Resistant FPGA," *Proc. SPL Programmable Logic 5th Southern Conf*, 2009, pp. 177–182.

39. Popp, T. and Mangard, S., "Implementation aspects of the DPA-resistant logic style MDPL," *Proc. IEEE Int. Symp. Circuits and Systems ISCAS 2006*, 2006.

40. Clavier, C., Coron, J.-S., and Dabbous, N., "Differential Power Analysis in the Presence of Hardware Countermeasures," *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '00, Springer-Verlag, London, UK, 2000, pp. 252–263.

41. Lu, Y., O'Neill, M. P., and McCanny, J. V., "FPGA implementation and analysis of random delay insertion countermeasure against DPA," *Proc. Int. Conf. ICECE Technology FPT 2008*, 2008, pp. 201–208.

42. Fan, J., Guo, X., De Mulder, E., Schaumont, P., Preneel, B., and Verbauwhede, I., "State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures," *Proc. IEEE Int Hardware-Oriented Security and Trust (HOST) Symp*, 2010, pp. 76–87.

43. Messerges, T. S., Dabbish, E. A., and Sloan, R. H., "Examining Smart-Card Security under the Threat of Power Analysis Attacks," *IEEE Trans. Comput.*, Vol. 51, May 2002, pp. 541–552.

44. Popp, T., Oswald, E., and Mangard, S., "Power Analysis Attacks and Countermeasures," *IEEE Design & Test of Computers*, Vol. 24, No. 6, 2007, pp. 535–543.

45. Standaert, O.-X., Peeters, E., Rouvroy, G., and Quisquater, J.-J., "An Overview of Power Analysis Attacks Against Field Programmable Gate Arrays," *Proceedings of the IEEE*, Vol. 94, No. 2, feb. 2006, pp. 383 –394.

46. Booth, A. D., "A Signed Binary Multiplication Technique," *The Quarterly Journal of Mechanics and Applied Mathematics*, Vol. 4, No. 2, 1951, pp. 236–240.

47. Daly, A. and Marnane, W., "Efficient architectures for implementing montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic," *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, FPGA '02, ACM, New York, NY, USA, 2002, pp. 40–49.

48. Montgomery, P. L., "Modular Multiplication Without Trial Division," *Math. Computation*, Vol. 44, 1985, pp. 519–521.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 22–03–2012	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Aug 2010 — Mar 2012

4. TITLE AND SUBTITLE RSA Power Analysis Obfuscation: A Dynamic FPGA Architecture	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Barron, John W., Capt, USAF	**5d. PROJECT NUMBER** 12G292O
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GE/ENG/12-02
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Robert L. Herklotz Program Manager - Information Operations and Security Air Force Office of Scientific Research (AFOSR/RSL) 875 N. Randolph Street, Suite 325, Room 3113 Arlington, VA 22203-1768 (703) 696-6565 robert.herklotz@afosr.af.mil	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFOSR/RSL
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

The modular exponentiation operation used in popular public key encryption schemes, such as RSA, has been the focus of many side channel analysis (SCA) attacks in recent years. Current SCA attack countermeasures are largely static. Given sufficient signal-to-noise ratio and a number of power traces, static countermeasures can be defeated, as they merely attempt to hide the power consumption of the system under attack. This research develops a dynamic countermeasure which constantly varies the timing and power consumption of each operation, making correlation between traces more difficult than for static countermeasures. By randomizing the radix of encoding for Booth multiplication and randomizing the window size in exponentiation, this research produces a SCA countermeasure capable of increasing RSA SCA attack protection.

15. SUBJECT TERMS

RSA Encryption, Countermeasures, Differential Power Analysis, Encryption, FPGA, Side-Channel Analysis

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Maj Todd Andel, USAF (ENG)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U	UU	88	(937) 255–3636, ext 4901; todd.andel afit.edu