International Journal of Computing and Digital Systems

RSA Power Analysis Obfuscation: A Dynamic Algorithmic Hardware Countermeasure

[1]Todd R. Andel, [2]John W. Barron, [1]J. Todd McDonald,[3]Jeffrey W. Humphries

[1] University of South Alabama, Mobile, AL, USA
[2]Air Force Institute of Technology, WPAFB, OH, USA
[3] Covenant College, Lookout Mountain, GA

E-mail address: tandel@southalabama.edu, john.barron@us.af.mil,jtmcdonald@southalabama.edu, jeff.humphries@covenant.edu

Received 21 Feb.2014, Revised 19 Mar. 2014, Accepted 29Mar. 2014, Published 1May. 2014

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 static, referring to the algorithmic elements as implemented in hardware. 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 paper develops a dynamic countermeasure which constantly varies the timing and power consumption of each operation, making correlation between traces more difficult. By randomizing the radix of encoding for Booth multiplication and randomizing the window size for exponentiation, we demonstrate a SCA countermeasure can be constructed which increases RSA SCA attack protection up to at least 100,000 encryption cycles, as well as a reduced execution time due to a reduction in required operations.

Keywords: Countermeasure, Side-Channels, FPGA, DPA, RSA

1. INTRODUCTION

In the seminal 1999 paper [1], Kocher et al. present 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 adversary to recover keys much faster than traditional cryptanalysis. The RSA public key encryption algorithm [2] has been the target of many of these attacks. Current methods of SCA protection fall into two main categories: masking and hiding [3]. Masking introduces randomness into the input text to introduce independence between intermediate calculations and power consumption. Hiding obscures 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. However, these static[1] countermeasures are insufficient because they can be broken by collecting more or higher fidelity power traces, and by improved post processing using better algorithms.

This paper introduces a dynamic countermeasure that incorporates run-time algorithmic randomness into operations, intermediate values, power consumption, and timing. This constantly changing algorithmic signature drives side channel attackers to perform a "brute force" search to achieve the correlation needed for successful attacks. The paper is organized as follows: Section 2 provides a short background on RSA, Booth multiplication, and modular exponentiation methods. Section 3 introduces the dynamic algorithmic countermeasure. Sections 4 and 5 discuss several well-known side channel attacks and current existing countermeasures. Sections 6 and 7 present the results of implementing the countermeasures and metrics respectively. Lastly, Section 8 provides the research conclusions.

2. BACKGROUND

Modern cryptographic algorithms use very large integers for keys, e.g., 4096-bit RSA. Many public key cryptographic algorithms, including RSA, rely on modular exponentiation operations. RSA decryption, as seen in Equation 1, raises the ciphertext*C* to the power of the private key *d* modulo *N*. The result is the plaintext message *M*.

1 We use the term **static** to refer to non-changing algorithmic elements utilized at run-time.

E-mail: tandel@southalabama.edu
This material is based in part upon work supported by the National Science Foundation under Grant No. 1305369

$$M = C^d \pmod{N} \qquad (1)$$

Naively computing C^d via $d-1$ multiplications of C is impractical because of memory and timing constraints [4]. 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.

A. Modular Exponentiation

Binary exponentiation is the most straightforward method for exponentiation; operating on a single bit of the binary representation of the exponent each iteration. This process can be performed either from the left-to-right (MSB to LSB) or right-to-left (LSB to MSB). The square-and-multiply algorithm for left-to-right binary exponentiation can be seen in Algorithm 1.

Algorithm 1 Left-to-Right Binary Exponentiation [4]

INPUT: Base C, Modulus N, and Exponent $d = (d_t, d_{t-1}...d_1, d_0)_2$
OUTPUT: $C^d \pmod{N}$
$R := 1$;
for $i = t$ downto 0 **do**
  $R := R^2 \pmod{N}$; — Square
  if $d_i = 1$ **then**
    $R := R \cdot C \pmod{N}$; — Multiply
  end if
end for
return R

It is also possible to operate on multiple bits of the exponent each iteration. For maximum efficiency, the goal is to compute C^d using the fewest number of operations, given that it is only possible to multiply two already computed powers of C [5]. Since multiple bits are being operated on at once, there are fewer total operations needed. This approach, referred to as windowing, provides some speedup; however, this speedup comes as trade off of higher memory requirements and required precomputations. As seen in Algorithm 2, precomputation and storage of additional powers of C are needed. The larger the window size k, the fewer multiplication operations needed and more memory space required.

Algorithm 2 Window Method [4]

INPUT: Base C, Modulus N, Window Size k, Exponent $d = (d_t, d_{t-1}...d_1, d_0)_{2^k}$
OUTPUT: $C^d \pmod{N}$
PRECOMPUTE:
$c_0 := 1$;
for $i = 1$ to $2^k - 1$ **do**
  $c_i := c_{i-1} \cdot C$; — Compute Powers For Window
end for
EXPONENTIATE:
$R := 1$;
for $i = t$ downto 0 **do**
  $R := R^{2^k} \pmod{N}$;
  $R := R \cdot c_{d_i} \pmod{N}$; — Use Precomputed Powers
end for
return R

In order to complete the modular multiplications we must implement a multiplier. Basic binary multiplication is accomplished by repeated addition and shift operations. In contrast, Booth encoded multiplication [6] recodes operands to reduce the number of costly addition steps. This paper focuses on a modification of the Booth concept to dynamically randomize the calculation, which provides a level of SCA protection to RSA.

B. Booth Multiplication and Modular Reduction

Booth multiplication was chosen as the base multiplication architecture because it allows for multiple bits to be encoded and operated on during a single iteration. 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 operations are only required on 0-to-1 and 1-to-0 transitions. For each iteration, the LSBs of the multiplier are tested, arithmetic operation is completed (if needed), and multiplier and product registers are shifted.

To complete modular multiplication, the product of Booth multiplication must be reduced viaa modulo operation. There are two main classes of reduction techniques for modular multiplication: multiply-then-reduce and reduce-as-you-go [7]. This paper implements the reduce-as-you-goapproach.

The resultant modular Booth multiplier design is shown in Figure 1. 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 multiplier and the shift-left-amount (SLA). This result is fed into the modular reducer section of the hardware. Here, all possible reductions are computed concurrently and the MSB of 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 non-negative result. Therefore, this architecture implements combinational logic that concurrently computes all possible Booth results, selects the correct Booth result, computes all possible modular reductions, and selects the correct modular reduction all within a single clock cycle.

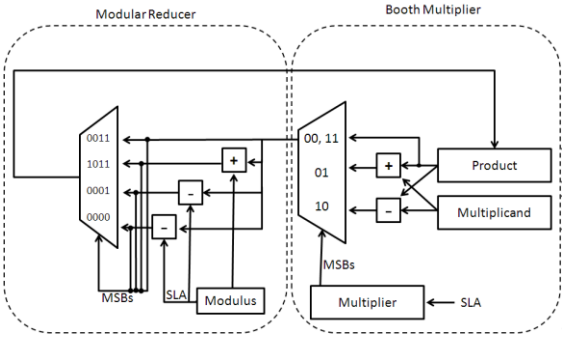

Figure 1. Modular Booth Implementation

3. DYNAMIC ALGORITHMIC COUNTERMEASURES

As previously stated, we use the term static to refer to non-changing algorithmic elements utilized at run-time. Figure 2 shows a complete categorizationof our terms of static and dynamic as used in this paper. The classification of our dynamic algorithmic countermeasures is indicated by the shaded portion.

	Algorithmic	Physical Structure
Static	non-varying algorithm	non-varying structure
Dynamic	random/data-driven algorithm	physical polymorphism

Figure 2. Static vs. Dynamic Countermeasures

A. Randomized Radix Encoding Booth Multiplier

This paper introduces a 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 1. 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 pre-computed and stored for later use.

The radix of Booth encoding is randomly selected as 2, 4, or 8 to provide SCA protection. Driven by a pseudorandom number generator, henceforth referred to as random, the multiplier hardware randomly selects a radix at each iteration. A simple linear feedback shift register is utilized for this research; however, more appropriate sources of randomization and a larger choice of radix bases could be used during implementation. This dynamic approach induces randomness into the timing of operand use, randomizes intermediate operands, randomizes power consumption, and multiplication completes in a non-deterministic number of clock cycles. This randomization causes alignment of traces to be lost and varied power consumption, making correlation much more difficult.

Table 1. 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	$-M$

B. Randomized Window Exponentiator

Building upon the randomness introduced via the variable radix encoding Booth multiplier, the window size of the exponentiator is also randomized by selecting between three options. Starting with the general windowing Algorithm 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, essentially determining the multiplier and multiplicand inputs into the Booth multiplier. 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 $R = R \cdot c_{di}$ is calculated using the precomputed powers of *c*. Similar to randomizing the radix in Booth multiplication, randomizing the window size for exponentiation further introduces timing randomness, operations randomness, and randomness in calculated intermediate values. Although this paper 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 tradeoff, since increasing the window size exponentially increases the number of precomputations and storage needed for very large integers.

4. SIDE CHANNEL ATTACKS

Our goal of adding randomness to timing and operations is to decrease the side channel attack susceptibility of the modular exponentiation operation. There are many known side channel attacks. Here we highlight a few well known attacks and reasons why our architecture is able to thwart them.

A. Simple Power Analysis

Simple Power Analysis (SPA) involves directly observing power traces to gather information about the secret key[1]. 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 1, if there is a distinguishable difference between a squaring operation and a multiplying operation (time or power), the implementation is vulnerable to SPA. The dynamic countermeasure introduced in this paper uses a single set of hardware constructs for multiplication. Squaring and multiplication are both completed as simple modular multiplication operations using the same hardware; there is no distinction between operations. Additionally, any possible timing or power variations will be difficult to discern as the true operation versus the randomness introduced by this dynamic countermeasure.

B. Differential and Correlation Power Analysis

Differential power analysis (DPA), presented by Kocher [1], is a very common and very powerful side channel attack. 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. Similarly in correlation power analysis, attackers 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 of the intermediate value, Hamming distance, zero value model, etc [3]. All of these attacks depend heavily upon first being able to calculate the correct intermediate values. With our randomly changing architecture there are many possible intermediate values for each multiplication and exponentiation bit. Secondly, DPA attacks require these intermediate values to be correlated to the power traces. Even in the instances when our architecture calculates the same intermediate values, there is randomization in the way they are calculated. Different methods of calculation will drive different transistor switching activity which draws different power, also decreasing correlation. Lastly, DPA relies on trace alignment to correlate operations at the same slice in time. While there are advanced algorithms to aid in trace alignment, all the randomness heretofore discussed makes this task more difficult.

5. EXISTING COUNTERMEASURES

As previously mentioned, many SCA countermeasures are static. That is, once these countermeasures are designed into a system, little is done to randomize the countermeasure at run-time. Static countermeasures rely on hiding the signal from an attacker. Several common ways this is attempted isdual-rail precharge logic (DPL) [8], bit balancing [9], noise addition [10], and delay insertion [11]. DPL and bit balancing concurrently calculate anintermediate result and its inverse, with the goal of hiding the true switching activity based on constant Hamming weights. The problems with these

countermeasures are imperfections in device fabrication or variations in the FPGA synthesis process (i.e., place and route algorithms). It is nearly impossible to fabricate a device that perfectly balances switching power consumption [12]. Static noise addition countermeasures can be defeated by simply taking additional traces, isolating the power of only the cryptocircuit under attack, or filtering. While random delay insertion has some dynamic operations, there are multiple techniques to defeat this countermeasure, such as integration [13]. The security of these countermeasures relies only on the signal quality the attackers are able to capture and the sophistication of attack processing. In contrast, the security of the dynamic countermeasure presented in this paper relies on the random algorithmic choices at two layers of operations within the architecture.

6. RESULTS

A. Simulation Results

We develop our customized countermeasure design 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.

1) Randomized Multiplication Simulation

Three identical computations of 146 ·85 (mod 207) were simulated to show the induced randomness in timing and operations. The simulation results are presented in Figure 3. It is clearly seen that all three modular multiplication computations' output arrive at the correct answer of 146 · 85 (mod 207) = 197. However, the simulations complete in 9, 5, and 6 cycles respectively.

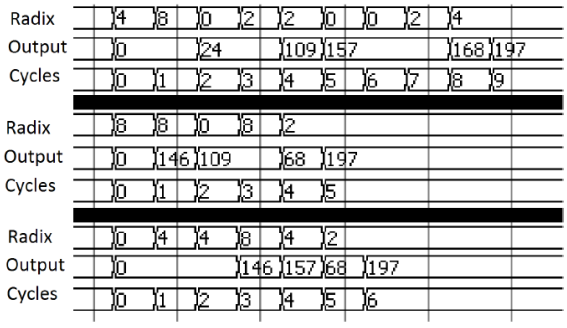

Figure 3. ThreeSimulationsof146·85(mod207)=197

Table 2 presents the three computations in tabular form in order to show the randomness in the calculations, where the multiplier 85 is represented in binary. As each computation traverses from MSB to LSB of the multiplier, horizontal separating lines show which bit(s) each cell operates on. Notice where given radix bases (i.e., radix iterations) end on the same bit (n.b., horizontal lines line up), the calculated intermediate value is the same, but where iterations end on different bits of the multiplier (n.b., horizontal lines do not 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.

Table 2. Three Computations of 146 · 85 (mod 207) [Entire Table (mod 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					$2^2 \cdot P = 2^2 \cdot 0 = 0$ $P + M = 0 + 146 = 146$	$4 = 01\underline{0}$
	1	$2^3 \cdot P = 2^3 \cdot 0 = 0$ $P + 3M = 0 + 3 \cdot 146 = 24$	$8 = 010\underline{1}$	$2^3 \cdot P = 2^3 \cdot 146 = 133$ $P - 3M = 133 - 3 \cdot 146 = 109$	$8 = 101\underline{0}$		
	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			$2 \cdot P = 2 \cdot 68 = 136$ $P - M = 136 - 146 = 197$	$2 = 1\underline{0}$	$2 \cdot P = 2 \cdot 68 = 136$ $P - M = 136 - 146 = 197$	$2 = 1\underline{0}$

2) Randomized Window Exponentiator

A simulation of three identical calculations of 146^{187} (mod 207) is presented in Figure 4. First, all three iterations are verified to arrive at the correct answer of 146^{187} (mod 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.

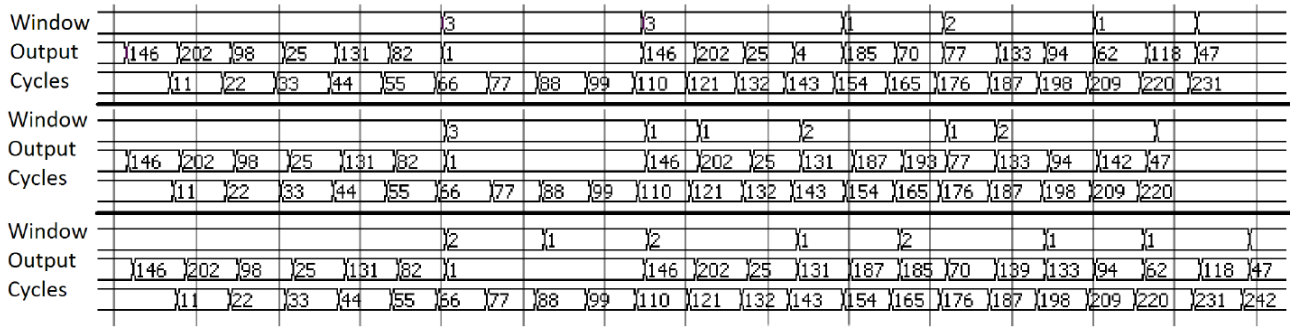

Figure 4. Random Window Configuration Simulation of 146^{187} (mod 207) = 47

More interestingly, the tabular calculation shown in Table 3 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 given exponentiation windows (i.e., window iterations) end on the same bit (n.b., horizontal lines line up), the calculated intermediate value is the same, but where iterations end on different bits of the exponent (n.b., horizontal lines do not line up) results are different.

Table 3. Three Computations of 146^{187} (mod 207) [Entire Table (mod 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	$R^2 = 1^2 = 1$		$R^2 = 1^2 = 1$		$R^2 = 1^2 = 1$	2
	0	$R^2 = 1^2 = 1$	3	$R^2 = 1^2 = 1$	3		
	1	$R \cdot M = 1 \cdot 146 = 146$		$R \cdot M = 1 \cdot 146 = 146$		$R^2 = 1^2 = 1$ $R \cdot M = 1 \cdot 146 = 146$	1
	0	$R^2 = 146^2 = 202$		$R^2 = 146^2 = 202$	1	$R^2 = 146^2 = 202$	
	1	$R^2 = 202^2 = 25$		$R^2 = 202^2 = 25$		$R^2 = 202^2 = 25$	2
	1	$R^2 = 25^2 = 4$ $R \cdot M^3 = 4 \cdot 98 = 185$	3	$R \cdot M = 25 \cdot 146 = 131$	1	$R \cdot M = 25 \cdot 146 = 131$	
	1			$R^2 = 131^2 = 187$ $R^2 = 187^2 = 193$		$R^2 = 131^2 = 187$ $R \cdot M = 187 \cdot 146 = 185$	1
	1	$R^2 = 185^2 = 70$ $R \cdot M = 70 \cdot 146 = 77$	1	$R \cdot M^3 = 193 \cdot 98 = 77$	2		
	0					$R^2 = 185^2 = 70$ $R^2 = 70^2 = 139$	2
	0	$R^2 = 77^2 = 135$		$R^2 = 77^2 = 133$	1	$R \cdot M^2 = 139 \cdot 202 = 133$	
	1	$R^2 = 135^2 = 94$ $R \cdot M^1 = 94 \cdot 146 = 62$	2			$R^2 = 133^2 = 94$ $R \cdot M = 94 \cdot 146 = 62$	1
	1	$R^2 = 62^2 = 118$		$R^2 = 133^2 = 94$ $R^2 = 94^2 = 142$	2	$R^2 = 62^2 = 118$	
LSB	1	$R \cdot M = 118 \cdot 146 = 47$	1	$R \cdot M^3 = 142 \cdot 98 = 47$		$R \cdot M = 118 \cdot 146 = 47$	1

B. Hardware Attack Results

The simulation results from the previous sections are the result of VHDL code. This code was subsequently synthesized onto a Xilinx Virtex-5 FPGA using Xilinx's XPS Design Suite. Four configurations were ultimately synthesized to hardware for attack analysis. The configurations used follow as:

1. **Baseline Configuration.** RSA configuration with no countermeasures.
2. **Randomized Multiplication Configuration**. Configuration uses a dynamic approach to binary exponentiation utilizing Booth multiplier that randomly selects between radix of 2, 4, or 8.
3. **Random Window Exponentiator Configuration**. The random window configuration uses static 2-radix Booth multiplication, but varies the exponentiation window size between 1, 2, or 3.
4. **Combined Countermeasure Configuration.** Dynamic countermeasure that combines the randomized radix encoding Booth multiplier hardware with the randomized exponentiation window architecture.

The synthesized configurations were subjected to a side channel electromagnetic (EM) correlation power analysis attack. First, a collection of 10,000 power traces with random ciphertexts is decrypted. During decryption, EM probes are used to collect emissions related to power consumption. These emissions and ciphertexts are saved for attack processing. 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 guess. The Hamming weight (HW) of 8-bits of the resultant intermediate guess is calculated. These 10,000 8-bit HW values are statistically correlated to the collected EM power traces. If the resultant correlation trace contains a correlation spike, the power consumption matches the HW values and the key-bit guess is verified as correct. If no correlation spike is seen, the key-bit guess is incorrect or the protection countermeasure has been successful. If the current key-bit can be identified, the traces are subsequently analyzed with the next subsequent key-bit guess, repeating until the key-bit sequence is revealed.

1) Attack on Baseline

The first implementation from this paper to be attacked is the baseline configuration. As expected, correlation power analysis is able to easily recover the key-bit. Without any randomizations of intermediate values, randomizations in power, or trace misalignments, the attack is successful as seen in Figure 5. To provide the reader an idea of an unsuccessful attack, the correlation trace with an incorrect key-bit guess is also included. Notice the noisefloor remains the same but a correlation spike is not present. Therefore, the baseline configuration is not protected against correlation power analysis.

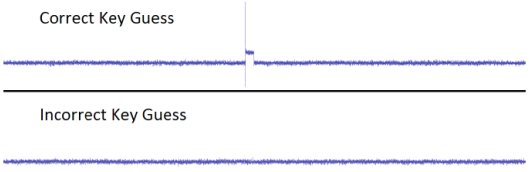

Figure 5. Correlation Attack on Baseline Configuration - Correct Vs. Incorrect Guess

Figure 6 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 constant amount of leakage is due to the static nature of the implementation. 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.

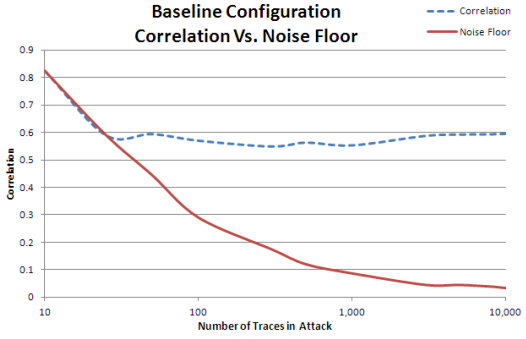

Figure 6. Baseline - Correlation Vs. Noise Floor

2) Attack On Randomized Multiplication

Since the randomized multiplication configuration has elements of low level misalignment and randomization, the effectiveness of the SCA attack is expected to be degraded. 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.

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 does not protect against correlation power analysis attacks. Figure 7 shows some increased protection. This configuration requires ≈ 80 power traces for successful attack. Also, notice how the correlation no longer remains at a constant level as more traces are collected. The changing power consumption and misalignment introduced by the random multiplier continually decrease correlation, although the noise floor was able to be reduced at a steeper slope thus revealing the correlation. This tenet of dynamic countermeasures highlights the promise of their use for SCA obfuscations.

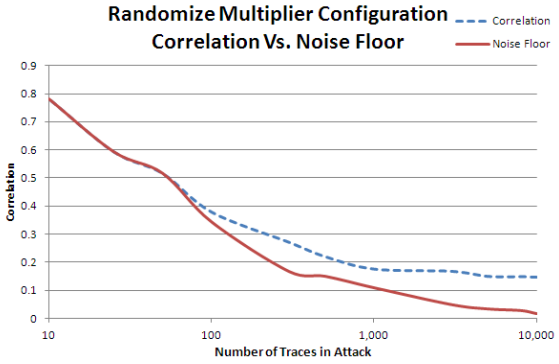

Figure 7. Randomized Multiplier - Correlation Vs. NoiseFloor

3) Attack On Random Window Exponentiator

Based on the ending times variance examined previously, the random window exponentiator configuration is expected to provide increased protection. Attacks against this configuration indicate a small area of increased correlation. This correlation area is much more dispersed than the previous two configurations, indicating that the random window configuration provides the best protection so far. Looking at Figure 8, it is immediately apparent that the randomized window configuration isable to maintain correlation below the noise floor up until≈2, 000 traces. The increased timing variance allows this configuration to spread the correlation over a larger time frame, which decreases its impact.

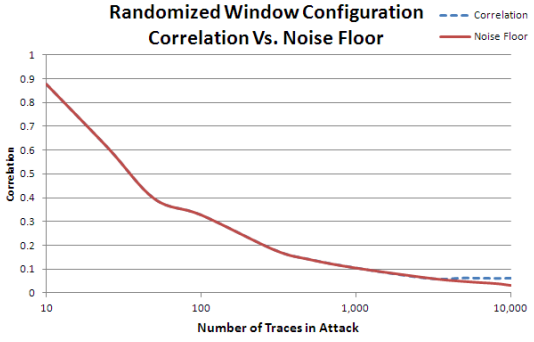

Figure 8. Randomized Window - Correlation Vs. NoiseFloor

4) Attack On Combined Countermeasure

Combining the low level randomness of the random multiplier with the randomness of the random window exponentiator proves to be the best chance at protection. Performing a 10,000 trace attack indicates no correlation discernible from the noise floor. In this scenario, the correlation cannot be distinguished from the noise floor in the correlation trace; therefore this configuration is declared protected against the 10,000 trace correlation power analysis attack.Subsequently, the number of traces collected against the combined countermeasure configuration was increased by an order of magnitude to 100,000 traces. The 100,000 trace attack required about 48 hours of constant CPU processor time for trace collection, filtering, and associated processing. The results from this 100,000 trace correlation power analysis attack are shown in Figure 9. With 100,000 traces there continues to be no discernible difference in the correct vs. incorrect key guess in this attack and the correlation never becomes greater than the noise floor.

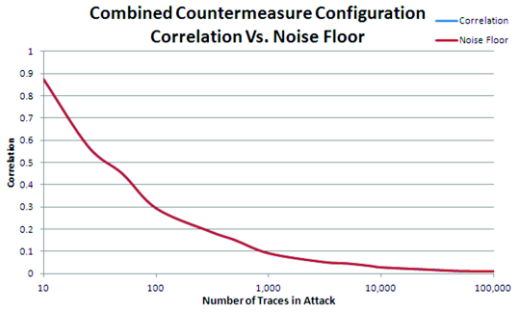

Figure 9. Combined Countermeasure - Correlation Vs. NoiseFloor-100K Traces

It is acknowledged that an attacker may be able to collect many orders of magnitude more traces in order to reduce the noise floor such that the correlation is discernible to break the combined countermeasure configuration. However, the amount of trace collection and processing time, as well as storage requirements, will also significantly increase. This

effect can be limited to reasonable amounts with sufficiently often key changes, with appropriate key lifetimes determined by operational risks assessments.

7. METRICS

Regardless of a countermeasure's security, no implementation is realizable if it does not meet the user's needs. Any security approach is useless if it cannot provide the required functionality in a timely manner.

A. Protection

This research seeks to force attackers into 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.

Recall that the baseline, random multiplier countermeasure, and random window countermeasure were vulnerable to attackers capturing 10,000 traces, with the exact level indicated as when the correlation of the key-bit guess exceeded the noise floor. The combination of both the random multiplier and random window countermeasure did not show any correlation with 10,0000, or even with 100,000, traces and is considered protected. Table 4 summarizes these results. Note that this research did not uncover a successful attack against the combined countermeasure.

Table 4. Number of Traces Required for SuccessfulAttack

Configuration	Number of Traces	Increased Protection
Baseline	30	N/A
Random Multiplier	80	166%
Random Window	2,000	6,500%
Combined Countermeasure	>100,00	>300,000%

B. Execution Time

The other metric examined is execution time. Any countermeasure that increases encryption/decryption execution time decreases throughput. The execution time is shown in Figure 10, with all configurations running at 12.5 MHz[2]. It may seem counter intuitive that the additional logic in the combined countermeasure results in faster run time; however, recall the combined countermeasure is actually using fewer clock cycles due to incorporating the randomized Booth encoding previously described.

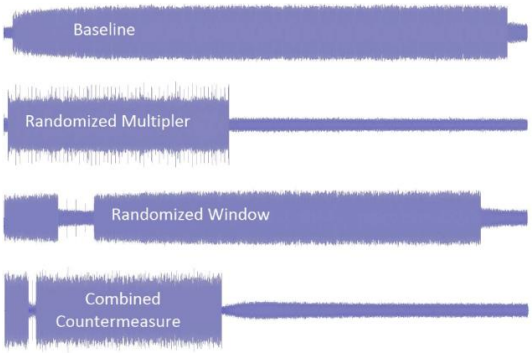

Figure 10. Execution Time of Configurations Compared(Running At 12.5 MHz)

8. CONCLUSIONS

This research is driven by the fact that current static SCA countermeasures rely on hiding the signals, not protecting them. These countermeasures' successes are largely based upon the quality of equipment and quality of signal an attacker is able to capture. The dynamic algorithmic countermeasures presented in this paper do not attempt hide power consumption, only to randomize it. Because the signal is not simply hidden beneath noise, the probability of successful attack is not determined by the sophistication of attacker equipment or quality of signal they capture. Combining the two forms of randomness (e.g., randomized window exponentiator and randomized Booth multiplication) creates a synergistic increase in protection. This randomized approach forces an attacker into a brute force side channel attack in which the attacker must attempt 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 security requirements, it is conceivable that large enough pools of radixes and window sizes could lead to a future system with brute force side

[2] 12.5 MHz is based on the is the maximum frequency for the combined countermeasure on the Xilinx Virtex-5 FPGA.

channel attack difficulty no worse than the underlying encryption algorithm's functional security, thus rendering side channel attacks impractical.

REFERENCES

[1] P.C. Kocher, J. Jaffe, and B. Jun,"Differential power analysis,"in 19th Annual International Cryptology Conference on Advances in Cryptology, M. Wiener, Eds. London, UK: Springer-Verlag, 1999, pp. 388–397.

[2] R.L. Rivest, A. Shamir, and L. Adleman,"A method for obtaining digital signatures and public-key cryptosystems," Commun. ACM, vol. 21, no. 2, pp. 120–126, February 1978.

[3] S. Mangard, E. Oswald, and T. Popp,Power AnalysisAttacks: Revealing the Secrets of Smart Cards. Secaucus, NJ: Springer-Verlag, 2007.

[4] A.J. Menezes, S.A. Vanstone, and P.C. Van Oorschot. Handbook of Applied Cryptography. 1st ed, Boca Raton, FL: CRC Press, 1996.

[5] N. Nedjah and L. Mourelle,"Efficient hardware for modular exponentiation using the sliding-window method with variable-length partitioning,"in 9th Int'l Conf forYoung Computer Scientists, 2008, pp. 1980–1985.

[6] A.D. Booth,"A signed binary multiplication technique," Qrtly Jrnl of Mechanics and Applied Mathematics, vol. 4, no. 2, pp. 236–240, 1951.

[7] A. Daly and W. Marnane,"Efficient architectures for implementing Montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic," in ACM/SIGDA 10thInt'l Symp on Field-Programmable Gate Arrays, New York, NY: ACM, 2002 p.p. 40–49.

[8] T. Popp and S. Mangard,"Implementation aspects of the DPA-resistant logic style MDPL,". in 2006 IEEE Int'l Symp on Circuits and Systems, 2006, pp. 2913–2916.

[9] J.A. Ambrose, S. Parameswaran, and A. Ignjatovic, "MUTE-AES: A multiprocessor architecture to prevent power analysis based side channel attack of the AES algorithm," in IEEE/ACM Int'l Conf on Computer-Aided Design, 2008,pp. 678–684.

[10] P.C. Liu, H.C. Chang, and C.Y. Lee, "A low overhead DPA countermeasure circuit based on ring oscillators,"IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 57, no. 7, pp. 546–550, 2010.

[11] S. Soydan, "Analyzing the DPA leakage of the masked s-box via digital simulation and reducing the leakage by inserting delay cells," in 4th Int'l Conf on Emerging Security Information Systems and Technologies, 2010, pp. 221–227.

[12] L. Sauvage, S. Guilley, J.L. Danger, Y. Mathieu, and M. Nassar,"Successful attack on an FPGA-based WDDL DES cryptoprocessor without place and route constraints,"in Design, Automation Test in Europe Conference, 2009,pp. 640–645.

[13] C. Clavier, J.S. Coron, and N. Dabbous,. "Differential power analysis in the presence of hardware countermeasures," in 2nd Int'l Workshop on Cryptographic Hardware and Embedded Systems, London, UK: Springer-Verlag, 2000, pp. 252–263.

Todd Andelis anAssociate Professor at the Universityof South Alabama's Schoolof Computing. Dr. Andel receivedhis Ph.D. in Computer Science fromFlorida State University in 2007, hisM.S. in Computer Engineering fromthe Air Force Institute of Technologyin 2002, and his B.S. in ComputerEngineering from the University ofCentral Florida in 1998. His researchinterests include computer and information security,side-channel analysis, hardware/software partitioning,network security protocols, and formal methods.

John Barronreceived hisB.S. in Electrical Engineering from theLouisiana Tech Universityin 2007 and a M.S. in Electrical Engineeringfrom the Air Force Instituteof Technology in 2012. His researchinterests include microcontrollers,control systems, computerarchitecture, VLSI, side-channelanalysis, and hardware encryption.

Jeffrey "Todd" McDonaldis an Associate Professorin the School of Computingat the University of SouthAlabama. Dr. McDonald receivedhis Ph.D. in Computer Science fromFlorida State University in 2006, hisM.S. in Computer Engineering fromthe Air Force Institute of Technologyin 2000, and his B.S. in ComputerScience from the U.S. Air ForceAcademy in 1990. His research interests include programprotection and exploitation, secure software engineering,and information assurance.

Jeffrey Humphries is an Associate Professor ofComputer Science in the Department of Computer Science at CovenantCollege. Dr. Humphries receiveda B.S. in Computer Science from the U.S. Air Force Academy, a M.S.degree in Computer Science fromGeorgia Institute of Technology, and a Ph.D. in Computer Science fromTexas A&M University in 2001. Hisresearch interests include cryptography,computer/network security, information assurance, cyberoperations, and software protection.