

MITRE Proprietary

SYSTEM SECURITY METRICS VIA POWER SIMULATION FOR VLSI DESIGNS

A Major Qualifying Project Report

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

by

J. Victor Haley

Daniel Hullihen

Date:

Approved:

Professor Thomas Eisenbarth, Advisor

MITRE Proprietary

Abstract

Power side-channel attacks are a growing concern as they allow attackers to extract sensitive information from digital systems with low-cost equipment and minimal knowledge about a device's inner functions. Though countermeasures are available to ASIC designers, these do not completely guarantee side-channel security, and therefore must be validated in the lab post-fabrication. The goal of this project is to verify the efficacy of simulation tools PSCARE & GLIFT to perform simulated power side-channel attacks upon such designs. Verification will be done via comparison of simulations of Advanced Encryption Standard to corresponding measurements of physical implementations on a SASEBO. Successful verification will allow for simulation of power side-channel information leakage at design-time.

Table of Contents

Abstract	i
List of Figures	1
List of Tables	1
1 Introduction	2
2 Background	4
2.1 Digital Hardware Design	4
2.2 Hamming Distance and Hamming Weight	5
2.3 Power Side-Channel Attacks	6
2.4 Advanced Encryption Standard (AES)	8
2.5 Implementations of S-Box in AES	10
2.6 Side-channel Attack Standard Evaluation BOard (SASEBO)	13
2.7 ASIC Designers’ Simulation Flow	14
2.8 PSCARE Tool Flow	17
2.9 Gate-Level Information-Flow Tracking (GLIFT)	18
2.10 Correlation Power Analysis Tool	20
3 Objectives	22
4 Methodology	23
5 Results	27
5.1 Initial Measured and Simulated Trace Results	27
5.1.1 Measured Traces and Normalization	27
5.1.2 Simulated Traces and Introduction of Gaussian Noise	32
5.2 DPA Results: Simulated vs. Measured	35
5.2.1 Composite Field Implementation of AES	35
5.2.1 Three-Stage Positive-Polarity Reed Muller Implementation of AES.....	38
5.2.1 Lookup Table Implementation of AES.....	40
5.3 Impact of GLIFT upon Simulated Designs	41
5.3.1 Lookup Table Implementation of AES.....	41

- 5.3.2 Three-Stage Positive-Polarity Reed-Muller Implementation of AES..... 42
- 5.3.3 Composite Field Implementation of AES 44
- 5.4 Back-Annotation of VCD Files to Pinpoint Information Leakage Points..... 51
- 6 Conclusions..... 52
 - 6.1 PSCARE Simulates Power Analysis Attacks, Produce Realistic Results 52
 - 6.2 GLIFT Successfully Selects Cells of Concern for PSCARE 52
- 7 Further Development..... 54
 - 7.1 Additional Verification and Improvements Upon PSCARE 54
 - 7.2 Additional Evaluation and Possibilities for GLIFT..... 54
- 8 References 56
- 9 Appendices 58
 - 9.1 Appendix A: PSCARE Readme and Documentation..... 58
 - 9.2 Appendix B: GLIFT Readme and Documentation 63
 - 9.3 Appendix C: Full DPA Results for AES Comp 71
 - 9.4 Appendix D: Full DPA Results for AES PPRM3..... 83
 - 9.5 Appendix E: Full DPA Results for AES TBL..... 95
 - 9.6 Appendix D: DPA Results: Full AES Comp vs. GLIFT Results..... 107
 - 9.7 Appendix F: DPA Results: AES Comp GLIFT-Selected and Excluded Cells with Noise 119

List of Figures

Figure 1: CMOS Logic Pair [7]	4
Figure 2: DPA Attack Flow	8
Figure 3: States in AES [17]	9
Figure 4: AES Algorithm [18]	10
Figure 5: Multi-Stage PPRM [20]	12
Figure 6: SASEBO-R [23]	13
Figure 7: General ASIC Design Flow Utilizing Synopsys Design Tools.....	15
Figure 8: PSCARE Flow Diagram.....	18
Figure 9: Gate-Level Information Flow Tracking (GLIFT) Tool Flow	19
Figure 10: Flow Diagram of the Power Analysis Attack Tool	21
Figure 11: Trace Capture Setup for SASEBO-R	24
Figure 12: Sample Voltage Trace Captured from SASEBO-R	28
Figure 13: Voltage Waveforms from SASEBO-R Overlaid for Varying Plaintexts	28
Figure 14: Histogram of Mean Voltages for Traces Captured from SASEBO-R	29
Figure 15: Time-Stamped Scatter Plot of Trace Means for Captured Traces.....	30
Figure 16: Normalized Voltages Traces for SASEBO-R Captures	31
Figure 17: Histogram of Trace Mean for Normalized SASEBO-R Captures	31
Figure 18: Time-Stamped Scatter Plot of Trace Means for SASEBO-R Captures	32
Figure 19: Simulated Power Traces Overlaid for Varying Plaintext Values.....	32
Figure 20: Signal-to-Noise Ratio for Correct Key Guess for Noiseless Simulation Data.....	33
Figure 21: Simulated Power Traces with 19% Gaussian Noise Added.....	34
Figure 22: Signal-to-Noise Ratio for Correct Key Guess for Noisy Simulation Data.....	34
Figure 23: DPA Results for AES Comp Measured (Left) vs. Simulated (Right).....	35
Figure 24: Correlation Values for all 256 Key Candidates vs. Number of Traces Collected.....	36
Figure 25: Signal-to-Noise Ratio For Correct Key Guess vs. Number of Traces Collected	38
Figure 26: DPA Results for AES PPRM3 Measured (Left) vs. Simulated (Right).....	38
Figure 27: Correlation Values for all 256 Key Candidates vs. Number of Traces Collected.....	39
Figure 28: Signal-to-Noise Ratio For Correct Key Guess vs. Number of Traces Collected	39
Figure 29: DPA Results for AES TBL Measured (Left) vs. Simulated (Right).....	40

Figure 30: Correlation Values for all 256 Key Candidates vs. Number of Traces Collected..... 40

Figure 31: Signal-to-Noise Ratio For Correct Key Guess vs. Number of Traces Collected..... 41

Figure 32: Unaltered Simulated Power Trace (Left) and Simulated Power Trace Using Cells Selected by GLIFT when Analyzing the Key (Right) of the Composite Field AES Design..... 45

Figure 33: Simulated Power Trace Using Cells Excluded by GLIFT when Analyzing the Key of the Composite Field AES Design 46

Figure 34: Overlap of All Simulations of Cells Selected by GLIFT for the Composite Field Design of AES 48

Figure 35: Comparison of the Original Design (Left) and the Cells Excluded by GLIFT (Right)for the Composite Field Design of AES..... 49

Figure 36: Signal-to-Noise Ratio for Correct Key Guess of the First Byte of the Original Design (Left) vs. the Cells Excluded by GLIFT (Right) for the Composite Field Design of AES 49

Figure 37: Comparison of the Original Design (Left) and the Cells Excluded by GLIFT (Right) with Gaussian Noise Added for the Composite Field Design of AES 50

Figure 38: Signal-to-Noise Ratio for Correct Key Guess of the First Byte of the Original Design (Left) vs. the Cells Excluded by GLIFT (Right) with Gaussian Noise Added for the Composite Field Design of AES 51

Figure 39: Timing Diagram of AES Comp Design with DPA Hits Included 51

List of Tables

Table 1: Correlation of Measured to Simulated DPA Results for AES Comp	37
Table 2: Correlation of Measured to Simulated DPA Results for AES PPRM3	39
Table 3: Correlation of Measured to Simulated DPA Results for AES TBL	40
Table 4: Breakdown of Cell Count by GLIFT when Analyzing the Key for the Lookup Table Design	42
Table 5: Breakdown of Cell Count by GLIFT when Analyzing the Plaintext for the Lookup Table Design	42
Table 6: Breakdown of Cell Count by GLIFT for the PPRM3 Design	43
Table 7: Breakdown of Cell Count by GLIFT when Analyzing the Plaintext for the PPRM3 Design	43
Table 8: Breakdown of Cell Count by GLIFT when Analyzing the Key for the Composite Field Design	44
Table 9: Breakdown of Cell Count by GLIFT when Analyzing the Plaintext for the Composite Field Design	47
Table 10: A Truth Table for 2-Input AND Gate with Taint Tracking	66

1 Introduction

Encryption standards are used ubiquitously for secure computing and communication in today's world, and consequently are the target of an increasing variety of threats and attacks. These attacks take advantage of the multi-faceted nature of modern electronics systems: implementation vulnerabilities may allow sensitive information to be revealed despite the security of underlying algorithms.

There are different forms of implementation attacks: invasive, semi-invasive and non-invasive. Each of these relate to how much the attacker physically alters the device. Invasive attacks involve probing or inserting something into the device to monitor values the device may be processing internally, violating the physical integrity of the device [1]. Attackers may also perform a fault analysis on the device by influencing how the device behaves in order to “gain erroneous computation results” [1]. Semi-invasive attacks consist of physically manipulating the “outer” layers of the device (e.g. removing a cover) in order to monitor the device or conduct less invasive fault analysis [1]. Non-invasive, or passive, attacks do not involve physically altering the device, focusing instead on aspects such as how much power the device consumes, how much electromagnetic radiation the device produces, and the amount of time the device takes to complete a given operation [1]. Thus, this kind of attack targets “side-channels,” or “any observable side effect of computation that an attacker could measure and possibly influence” [2]. Power side-channel attacks are particularly threatening because they are low-cost, require minimal assumptions about the device, and can go undetected due to their passive nature.

To protect against these threats, it becomes necessary to evaluate the devices and apply countermeasures to ensure system integrity. One approach to this process for very-large-scale integration (VLSI) application-specific integrated circuits (ASICs) involves taking the design to the laboratory after fabrication and performing several side-channel attacks and analyses to pinpoint vulnerabilities. Engineers then apply physical countermeasures to the device where they consider it appropriate. The device is then re-evaluated and this iterative process continues until the device is considered sufficiently secure. While this approach does help make the ASIC more secure to power side-channel attacks, the cost of addressing the vulnerabilities after fabrication is significant. Developers run the risk of unknowingly creating inherent vulnerabilities that are so severe that it becomes prohibitively expensive to try to address them after the design has been

fabricated. Evaluating a device after fabrication and then applying a countermeasure is the equivalent of applying a patch: it only addresses a single symptom and does not fix the root problem.

Designs are beginning to involve more third-party intellectual property (3PIP) that can “come from anywhere in the world” [3]. With this comes the risk of dealing with insecure 3PIP that is being incorporated into what otherwise may be a secure design. This risk significantly impacts the security of the final design and where the design can be used, increasing the need for extensive verification of 3PIP security [4], [5], [6]. Thus, evaluating a design after fabrication is becoming less practical as the use of 3PIP becomes increasingly more common, causing a greater part of the device’s security to come into question.

In order to determine how secure an ASIC design is against power side-channel attacks, designers need access to accurate information regarding the power consumption of their device. A growing need for power efficiency in electronic devices has led to the development of electronic design automation (EDA) tools that are developed to accurately simulate the expected power consumption of an ASIC design. Power Side-Channel Attack Risk Evaluator (PSCARE) is a tool developed at MITRE that repurposes the aforementioned EDA tools to simulate the measurements that attackers would observe on a device during a power side-channel attack. This tool enables engineers to evaluate and improve the power side-channel security of an ASIC during the design process, reducing the amount of resources and time necessary to search for and address such vulnerabilities in the lab.

Though the EDA tools used by PSCARE have been developed to accurately model the power characteristics of an ASIC design, it is necessary to confirm that PSCARE sufficiently models what attackers would observe from the fabricated design. This is warranted in part because of the potentially high cost in damages that could be incurred should these new tools miss a vulnerability that could have otherwise been found in the lab.

2 Background

Evaluating the efficacy of using PSCARE to pinpoint power side-channel vulnerabilities in an ASIC design requires familiarity with several topics. Power side-channel attacks are the main focus of this design, as it is a dangerous type of attack employed to reveal sensitive information. Understanding how the device under test (DUT) performs its secure operation and how it can leak information via a power side-channel is also crucial since this is the information being exploited by attackers. Since PSCARE is built on top of the simulation flow used by ASIC designers, it is necessary to understand how these tools produce the information that is subsequently used by PSCARE, paving the road to understanding the PSCARE tool flow. Since PSCARE simulates power side-channel attacks, it is necessary to understand the tools available to conduct a power side-channel attack in the lab for comparison in order to conduct a thorough evaluation.

2.1 Digital Hardware Design

The basic unit of digital design is the Complementary Metal-Oxide Semiconductor (CMOS) transistor, shown in Figure 1. This logic pair is used because of its extremely low power consumption during periods of inactivity.

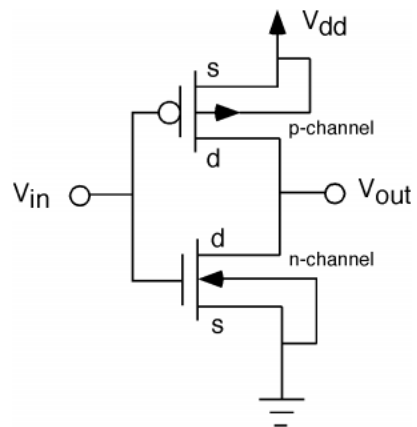


Figure 1: CMOS Logic Pair [7]

However, during the switching period of the CMOS pair, there is a momentary peak of current flowing through the gate. This is caused by both transistors partially conducting for an instant in time, which results in a low-impedance path between the power rail (V_{dd}) and Ground [7], [8]. This glitch is characteristic of CMOS logic, and is a major contributor towards a CMOS design's

power consumption [7], [8]. Additionally, power is consumed due to the capacitive loads on a CMOS design. These capacitive loads require a charging current during the switching cycle, and contribute to overall power consumption of a CMOS design. Due to this property, the power consumption of a chip depends largely upon the number of bit toggles that occur. This dependency leads to information leakage via a device's power consumption.

2.2 Hamming Distance and Hamming Weight

One characteristic of a given bit-signal in a design is its Hamming weight. This is described as the number of non-zero bits in a given signal. For example, the Hamming weight of 0101 is two because there are two bits that are not zero.

While the Hamming weight involves looking at one signal, it is possible to compare two signals by looking at the Hamming distance. This is described as the number of bits that are different between two signals. For example, the Hamming distance between 0101 and 1000 is three because three of the bits between the two signals are different (the first, second and fourth bit).

These characteristics can be used to create a power model of a device. The Hamming weight model is a simple approach that does not require much information about the design, if any at all. The attacker “assumes that the power consumption is proportional to the number of bits that are set in the processed data value,” ignoring previous and subsequent values [9]. This model also does not account for signals that change the same number of bits repeatedly – the Hamming weight will not change, suggesting constant power consumption; but transitions occurred that would result in greater power consumption.

The Hamming-Distance model accounts for this bit-toggle issue by tracking the number of bit transitions within a given period of time. The number of transitions will impact the power consumption since it relates to the transistor toggles mentioned in the previous sub-section. That being said, this model does not account for factors like parasitic capacitances, and it assumes that all bit toggles contribute equally to the power consumption of a design [9]. The Hamming-Distance model is well-suited for modeling register updates and their resulting power consumption. This is because it accounts for the total number of bit toggles that occur, and assumes a constant power consumption for each toggle. However, for a model that accounts for

static leakage, the Hamming-Weight model is more appropriate, as it assumes different power consumption values based upon each individual bit's state.

2.3 Power Side-Channel Attacks

The basic premise of a power side-channel attack is that “the instantaneous power consumption of a cryptographic device depends on the data it processes and on the operation it performs” [9]. An attack can involve running a cryptographic device through a normal cycle of activity while monitoring and recording the power consumption of the device. This monitoring can be done by inserting a small resistance either between the device's ground pin and the actual ground or between the power supply and the device. The voltage across this resistor can then be monitored, resulting in what is known as a “power trace,” or “a set of power consumption measurements taken across a cryptographic operation” [10]. Sometimes multiple power traces can be collected where different data is being processed using the same operations. Depending upon factors such as the encryption algorithm used and its implementation, these power traces can be analyzed using a variety of attack techniques.

One attack is Simple Power Analysis (SPA). This analysis technique is often applied to a small number power traces that reveal information about the sequence of instructions that was executed during the encryption process. For example, different components of a device perform different operations and can also consume different amounts of power. This “characteristic power consumption [can lead to] a characteristic pattern in the power trace” [9]. One example is a move instruction: a different amount of power may be consumed if the data is being moved to or from internal memory versus external memory. Variations like this can enable attackers to exploit the dependency between the data being encrypted and the order of instructions that was called.

A more sophisticated attack is Differential Power Analysis (DPA). Instead of focusing on the sequence of instructions that are executed as in SPA, DPA focuses on the variations of power consumption at a specific point of time between multiple power traces. This is a more popular attack method because no detailed knowledge of the device is required and the power traces can be relatively noisy [9]. With enough power traces, truly random noise approach zero, allowing

the results to be very clear. Mangard et al. provide a general strategy used in DPA attacks, which can be summarized in five steps [9]:

1. **Target an Intermediate Result**

The attacker focuses on an intermediate result of the cryptographic algorithm that they can calculate or derive. This value must be a function of known data (plaintext or ciphertext) provided by the attacker and a part of the secret key being targeted (e.g. a certain byte).

2. **Collect Power Traces**

The attacker collects power traces of the device, providing different plaintexts to be encrypted by the same process for each trace.

3. **Generate Hypothetical Intermediate Values**

The attacker determines hypothetical intermediate values for every possible value of the part of the key that is being attacked. If a byte of a key is being attacked, then there are 2^8 or 256 possible values to guess. There are thus 256 values to calculate with each plaintext in order to generate a set of hypothetical intermediate values.

4. **Generate Hypothetical Power Consumption Values**

These sets of hypothetical intermediate values are then mapped to power consumption values. The mapping process ultimately depends on how much the attacker knows about the device since they have to create a sufficient power model. Two common models are the Hamming-weight and Hamming-distance models mentioned in Section 2.2.

5. **Correlate the Hypothetical Power Consumption with the Measured Power Traces**

These collections of hypothetical power consumption values for a given key byte guess are then compared to the measured power traces. Correlations between each set of intermediate values generated for a given 256 key byte guesses is compared to the measured power traces. A correlation between each of the sets is derived.

Depending upon factors such as the signal-to-noise ratio of the information leakage, the strength of the correlation, and the number of traces collected, the attacker can assume that the key byte guess with the highest correlation value is indeed the one being used by the device. This process can be repeated for the remaining bytes of the key. Figure 2 shows a visual layout of a DPA attack.

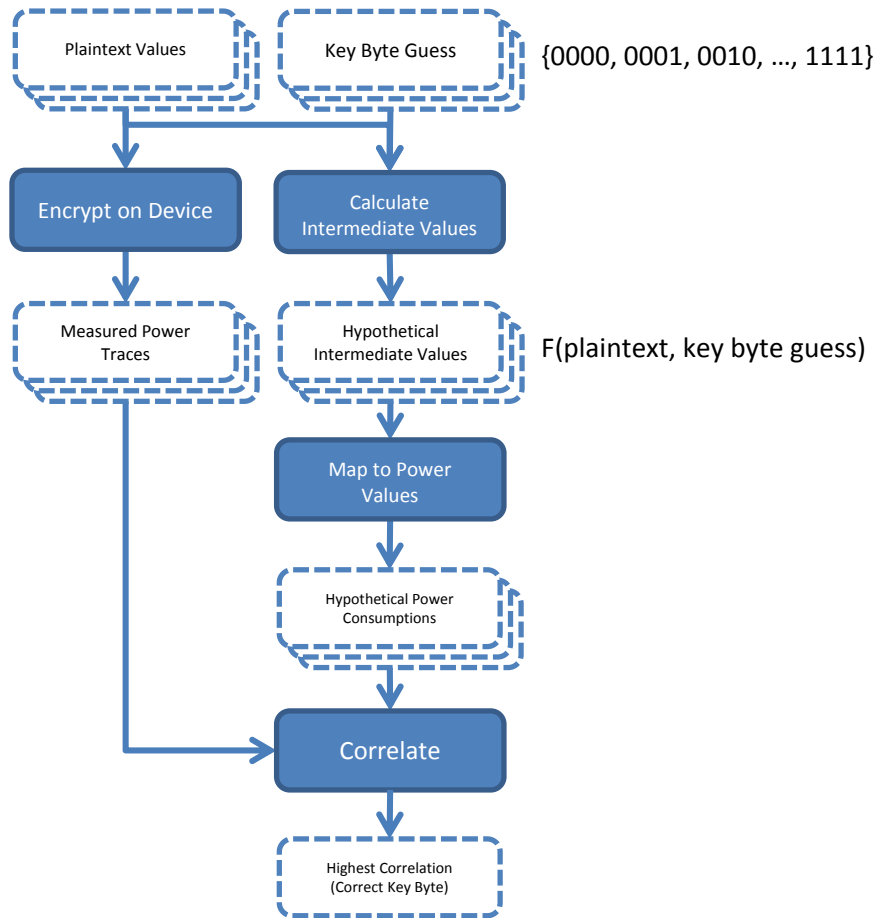


Figure 2: DPA Attack Flow

2.4 Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) is a secure algorithm that is widely used to protect sensitive data. It was announced by the National Institute of Standard and Technology (NIST) in 2001 and is available for public use. AES is a symmetric-key block cipher that uses a block size of 128 bits, and can have a key length of 128, 192, or 256 bits [11]. From an algorithmic point-of-view, it is secure: to use a brute-force attack to guess the secret key would take 2^{128} guesses for the 128-bit AES implementation, and a staggering 2^{256} guesses for the 256-bit implementation [11]. Given the sheer number of possible secret keys, it is extremely unlikely, almost impossible, that a brute-force attack would be successful on an AES.

There are a variety of sources detailing the AES algorithm [12][13][14][15][16]. Data encrypted by the AES algorithm is organized into states, which are 4-column arrays of 4-byte vectors. Figure 3 shows a general visual representation of how states are used in AES:

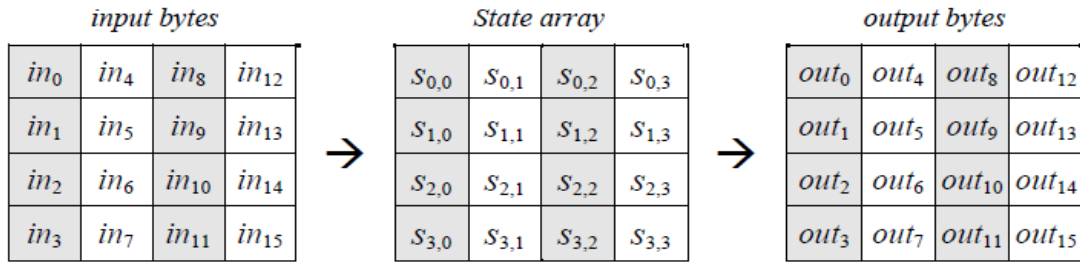


Figure 3: States in AES [17]

While arrays of bytes are passed into the algorithm, they are converted into states for processing. The output state is then converted back into an array and returned to the user.

AES is comprised of four distinct sub-functions:

1. *SubBytes*. A substitution box (or S-Box) takes each byte in the state and replaces it with a corresponding byte specified in the S-Box. Importantly, the S-Box is a direct mapping of one value to another, allowing it to be implemented with a simple lookup table, and also is invertible, meaning the S-Box can be applied in reverse. The SubBytes substitution of a value is found by first taking the multiplicative inverse of this value, and then transforming using an affine transformation.
2. *ShiftRows*. Each row in the state is cyclically shifted over by a varying number of bytes;
3. *MixColumns*. Each column's bytes are rearranged; and
4. *AddRoundKey*. Each byte of the state is added to a corresponding byte of the round subkey, generating the subkey for the next round.

They key length and block size determine the number of rounds (denoted N_r) that must be done in AES. The block size is fixed at 4 words where each word is 32 bytes, so the main difference between each flavor of AES is the key length. AES-128 requires 10 rounds of encryption while AES-192 and AES-256 requires 12 and 14 rounds of encryption, respectively. Initially, the provided key is “expanded” to generate a key for each round of the algorithm and the plaintext is added to the original key to create the initial state. For the next N_r-1 rounds, the *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey* functions are called sequentially on the state. The final round is the same as the previous rounds, except *MixColumns* is skipped. Figure 4 shows a visual representation of the algorithm.

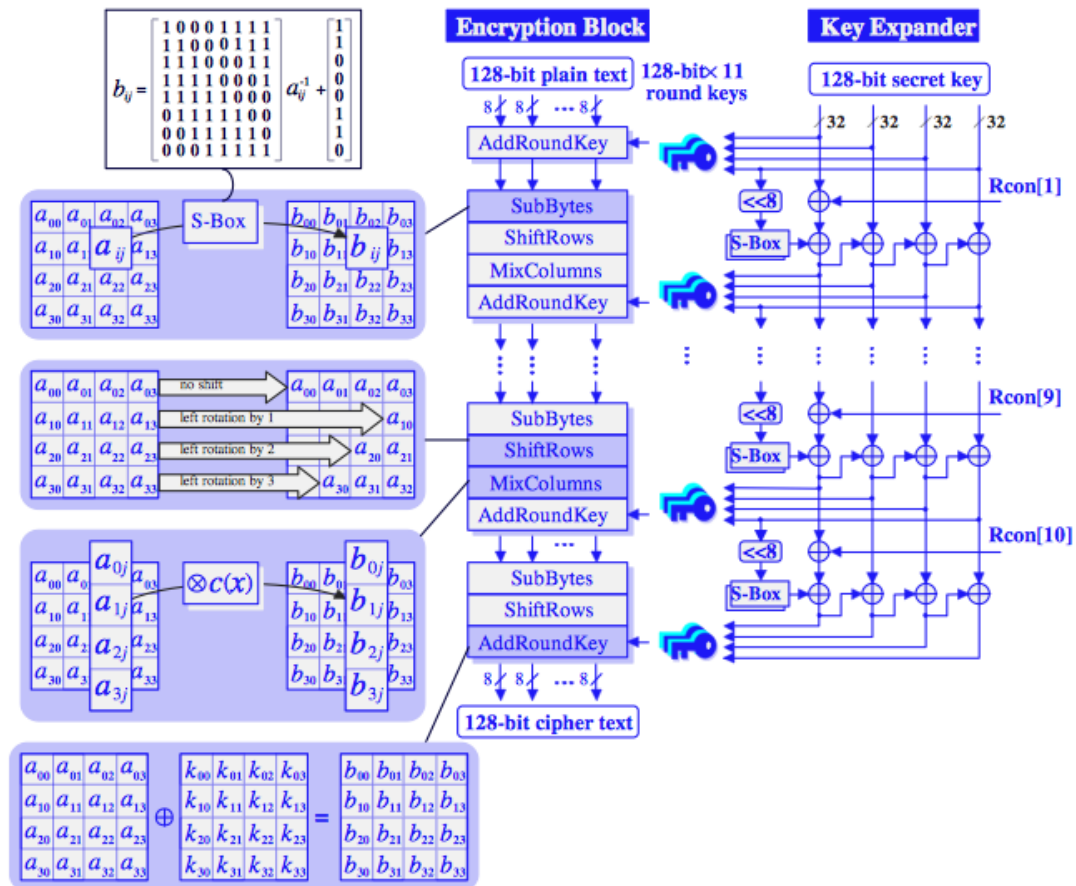


Figure 4: AES Algorithm [18]

2.5 Implementations of S-Box in AES

When AES is implemented in software or hardware, additional information becomes available to an attacker, such as power consumption data. Side-channel leakage becomes very important, as it is an avenue that allows an attacker to guess the secret key one byte at a time, reducing the total number of key guesses for AES-128 and AES-256 to a manageable 2^{12} (4096) guesses and 2^{13} (8192) guesses, respectively [10].

A common target within the AES algorithm is the S-Box circuit. The S-Box is a multiplicative inversion on a Galois Field, $GF(2^8)$, that applies a nonlinear transformation to a given byte of information. [19]. The S-Box circuit has been found to consume a great part of the total power (up to 75%) and size of the design [12][20][21]. Thus, a variety of implementations have been

developed for the S-Box as a means to reduce the power consumption and area of the circuit and/or to increase processing speed. These implementations include:

1) Look-Up Table S-Box

Electronic Design Automation (EDA) tools can be used to implement the S-Box as a look-up table. This was one of the earlier implementations used that is often seen in the form of case statements. One issue with look-up tables is the amount of hardware required, which made it difficult to employ AES in small embedded systems [18]. Because minimal computation is done to look up a value in a table, this implementation has been found to be one of the fastest S-Box implementations while also being relatively resistant to power side-channel attacks [12].

2) Composite Field S-Box

The composite field implementation was designed to “obtain both a small circuit in case of a hardware implementation as well as smaller instruction counts and table sizes in case of software implementations” [19]. A Galois Field, $GF(2^8)$, is created and used to calculate the appropriate output on the fly via composite field arithmetic. In this implementation, the input is mapped to the composite field, its multiplicative inverse is calculated, and the result is mapped back for output [18]. This has been found to be the most resistant S-Box implementation to power-side channel attacks, all the while using the least amount of hardware [12]. However, this design also has the lowest maximum operating frequency.

3) Three-Stage Positive Polarity Reed-Muller (PPRM3) S-Box

PPRM3 was originally developed as a more power efficient alternative to the original composite field implementation. This implementation utilizes Positive Polarity Reed-Muller Expressions (PPRME), which is class of AND-EXOR expressions that is “unique for a given function”, and thus cannot be minimized. Take an arbitrary logic function $f(x_1, x_2, \dots, x_n)$ represented as:

$$f = 1 * f_0 \oplus x_1 * f_2 \tag{1}$$

where $f_0 = f(0, x_2, x_3, \dots, x_n)$, $f_1 = f(1, x_2, x_3, \dots, x_n)$, and $f_2 = f_0 \oplus f_1$. According to Allen, if you apply the expansion shown in Equation 1 to each variable, “an expression consistent of positive literals only” is generated [22]:

$$a_0 \oplus a_1 x_1 \oplus \dots \oplus a_n x_n \oplus a_{13} x_1 x_2 \oplus a_{13} x_1 x_3 \oplus \dots \oplus a_{nn-1} x_n x_{n-1} \oplus \dots \oplus a_{12\dots n} x_1 x_2 \dots x_n \quad (2)$$

Using this form of expression, parts of the S-Box circuit can be converted into two-level logic (one of AND gates, another of XOR gates) [20]. Figure 5 shows a visual representation of how various parts of the S-Box logic can be broken down into two-level logic.

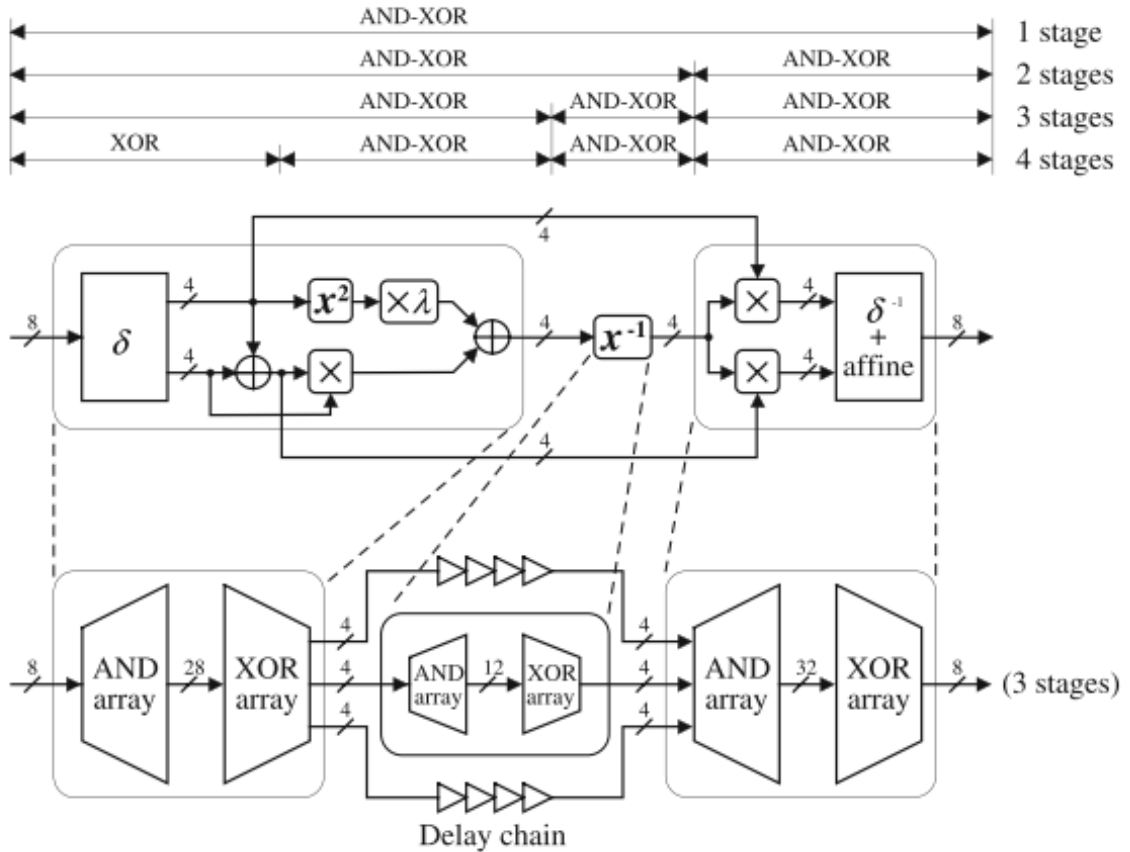


Figure 5: Multi-Stage PPRM [20]

The 3-stage PPRM (PPRM3) was found to achieve the lowest power consumption of the given multi-stage PPRM designs [20]. It was also found to be the second most

DPA-resistant implementation (behind the original composite field implementation) [12]. Thus, designers sacrifice some power side-channel security for an increase in power efficiency when they convert parts of the composite field S-Box to PPRMs.

The procedure detailing how exactly the secret key can be extracted by collecting power traces is detailed in Power Analysis Attacks – Revealing the Secrets of Smart Cards [9].

2.6 Side-channel Attack Standard Evaluation BOard (SASEBO)

In order to reliably test different cryptographic algorithms' vulnerability to power side-channel attacks, the Side-channel Attack Standard Evaluation Board (SASEBO) was developed by Tohoku University and the National Institute of Advanced Industrial Science and Technology (AIST). A variety of evaluation boards exist and continue to be developed by these organizations. Some include reprogrammable FPGAs (e.g. SASEBO-GII) while others have an ASIC comprised of different encryption cores that are controlled by a separate FPGA (e.g. SASEBO-R). These boards are outfitted with various pins and SMA connectors to allow researchers to easily measure the power consumption of the device and perform a power side-channel attack.

One board of interest with respect to evaluating ASIC designs is the SASEBO-R. A picture of the board is shown in Figure 6.

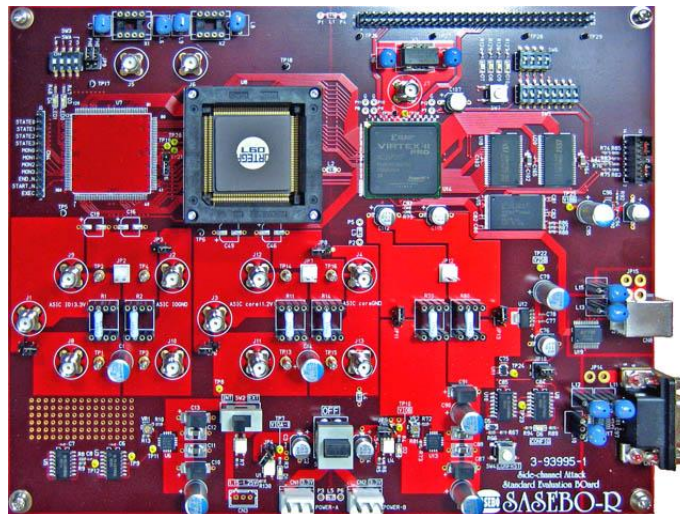


Figure 6: SASEBO-R [23]

The SASEBO-R has a cryptographic LSI that is controlled by a Xilinx Virtex-II Pro FPGA [24]. There are SMA connectors and pins that allow users to measure the power consumption on the VCC or GND side of either the LSI or the I/O between the LSI and the FPGA. The LSI chip has 14 implementations of AES and 8 other cryptographic algorithms including SEED, Triple-DES and RSA. The board also generates a trigger signal that can be used for aligning different power traces [25].

An LSI checker written in C# is provided with the SASEBO-R as a means of testing each core in the LSI on the board. During each encryption, a start signal is raised to indicate the beginning of the encryption process, and an end signal is raised to indicate the end. These signals can be used as triggers on an oscilloscope to align multiple power traces.

Verilog code for some of the cryptographic cores is provided by Aoki Laboratory at Tohoku University [26]. Specifically for AES, there are a number of implementations that are different flavors of S-Box implementations: composite field, look-up table and PPRM3. Thus, it is possible to simulate these designs and compare them (to some degree) to the corresponding cores that were implemented on the LSI. The LSI specification sheet gives more detailed information on how each algorithm was implemented [25].

2.7 ASIC Designers' Simulation Flow

Several EDA tools exist that enable ASIC designers to thoroughly simulate their design to verify that it meets various design constraints, such as timing and power. The diagram in Figure 7 details a general ASIC design flow that incorporate tools developed by Synopsys.

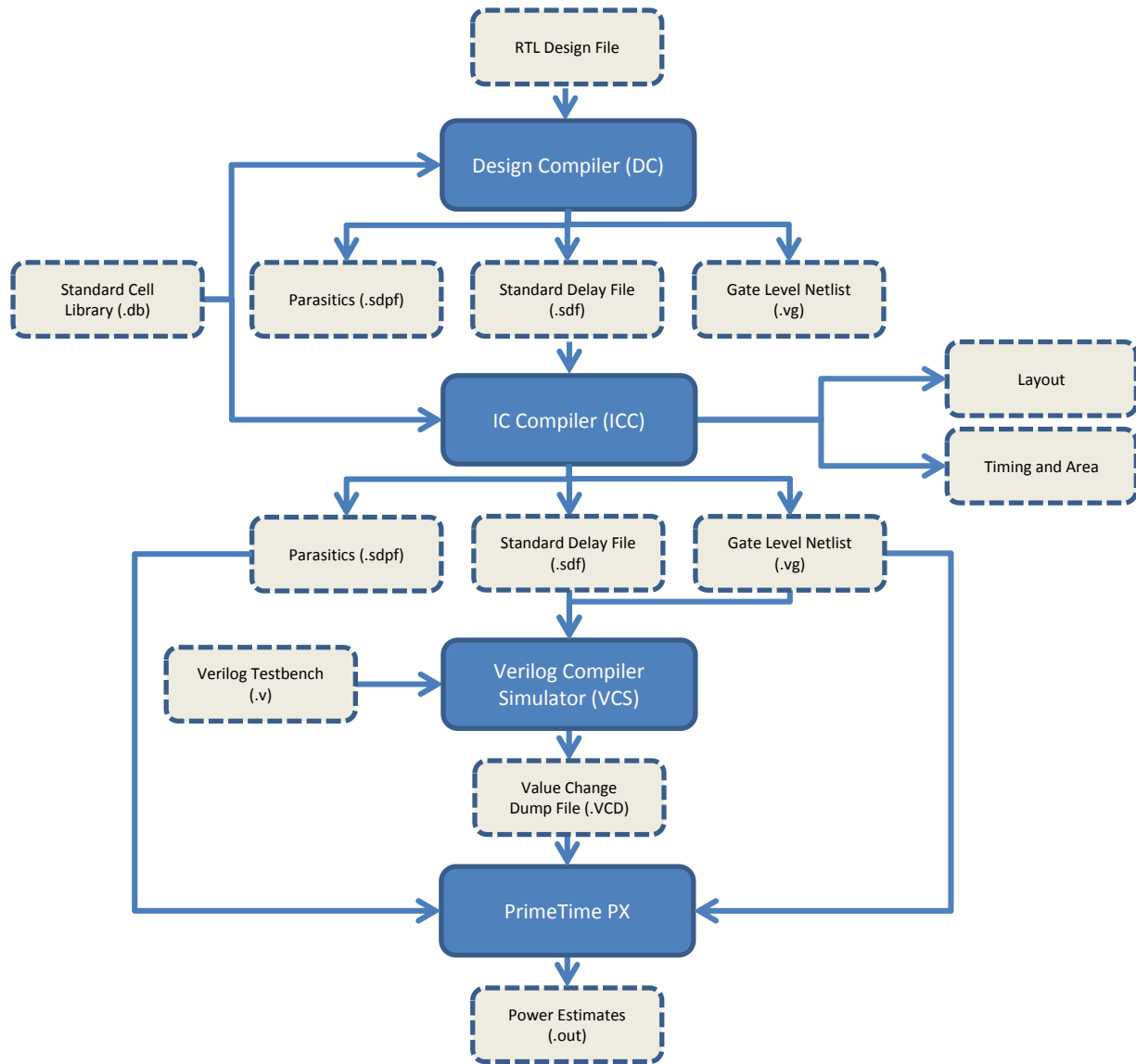


Figure 7: General ASIC Design Flow Utilizing Synopsys Design Tools

Typically, ASIC designers begin with a design file that is written in a Hardware Description Language (HDL) e.g. Verilog and VHDL. The designer then provides the design and a Standard Cell Library to Design Compiler (DC). A Standard Cell Library outlines the power and timing characteristics of each logic cell (e.g. two-input AND gate, four-input OR gate, etc.) that would be physically implemented in the design. This information is often derived from SPICE simulations. The designer then specifies various requirements for the design (e.g. minimum and maximum timing delays for a part of the design) before compiling and synthesizing it into a “gate-level netlist” (referred to simply as a netlist). This represents a theoretical schematic of the design. Files outlining preliminary estimates of the parasitic characteristics (SDPF) and timing

delays (Standard Delay Format or SDF) for each cell instantiated in the design are also generated.

The next step is known as “place and route,” which involves the tool called Integrated-Circuit Compiler (ICC). ICC takes in all of the files generated by DC and the Standard Cell Library. This tool helps specify how each cell in the schematic from DC will be laid out in a given area representing the die. What is generated at this stage of the design flow is sent out for fabrication. Because of this, designers must be sure that every logic cell is correctly placed and properly connected to other cells in the design. In order to ensure the design works as intended, it becomes necessary to simulate the design at this stage. While ICC can write out the same files as DC (e.g. netlist, SDF), designers use ICC to specifically analyze how each cell will ultimately be physically placed and interconnected on a die. Thus, these new files better represent the design that will actually be fabricated.

In order to determine the power characteristics of the design, it is necessary for designers to create a test bench that gives the design a list of instructions to perform. Verilog Compiler Simulator (VCS) takes in this test bench, the netlist, and the standard delay file (SDF) to determine switching activity of each cell in the design. This information is exported into a Value Change Dump (VCD) file, which indicates when each cell in the netlist changes its output value over the course of the simulation outlined in the test bench. Thus, the VCD file describes the switching activity of every cell specified in the netlist while accounting for delays specified in the SDF file. General power characteristics of each cell are described in the Standard Cell Library and also the SDPF file written by ICC. With these files, it is now possible to create a power simulation.

PrimeTime-PX (PT-PX) takes in a VCD file, a netlist, the Standard Cell Library, and the parasitics (SDPF) file. With this, the tool can derive the power consumption of each cell in the design over the course of the simulation that was originally specified by the test bench. Once that is done, the designer has successfully simulated the power consumption of their design. In order to generate a new power simulation, the designer changes the test bench and re-runs VCS to generate the new files that would be used by PT-PX.

2.8 PSCARE Tool Flow

Using the results of the ASIC design-flow, a VCD file detailing the time-stamped bit toggles and an OUT file detailing the power consumption of a specific simulation would be produced for each trial. Specific to an AES core design, the variable in the trial would be the plaintext (the input data to be encrypted). The secret key remains the same, while the plaintext is varied for all the different trials, typically ranging from 500 to 10,000 in number. The PSCARE simulation tool could be used to simulate any number of trials; for the designs tested, 20,000 traces could be produced in approximately 30 hours for a basic AES encryption core with a full power simulation.

The VCD and OUT files are then parsed by PSCARE tools to produce two arrays for each simulation. The result of parsing the VCD file is a power trace created with the toggle count model. This model assumes constant power consumption whenever a net toggles its state, and therefore the power consumption at each point in time is correlated to the number of bit toggles that occurred. The result of parsing the OUT file is a power trace created using the power information generated by PT-PX. These arrays are then resampled to model the bandwidth limitation of an oscilloscope that would be used to measure the physical implementation. The traces are also checked for alignment.

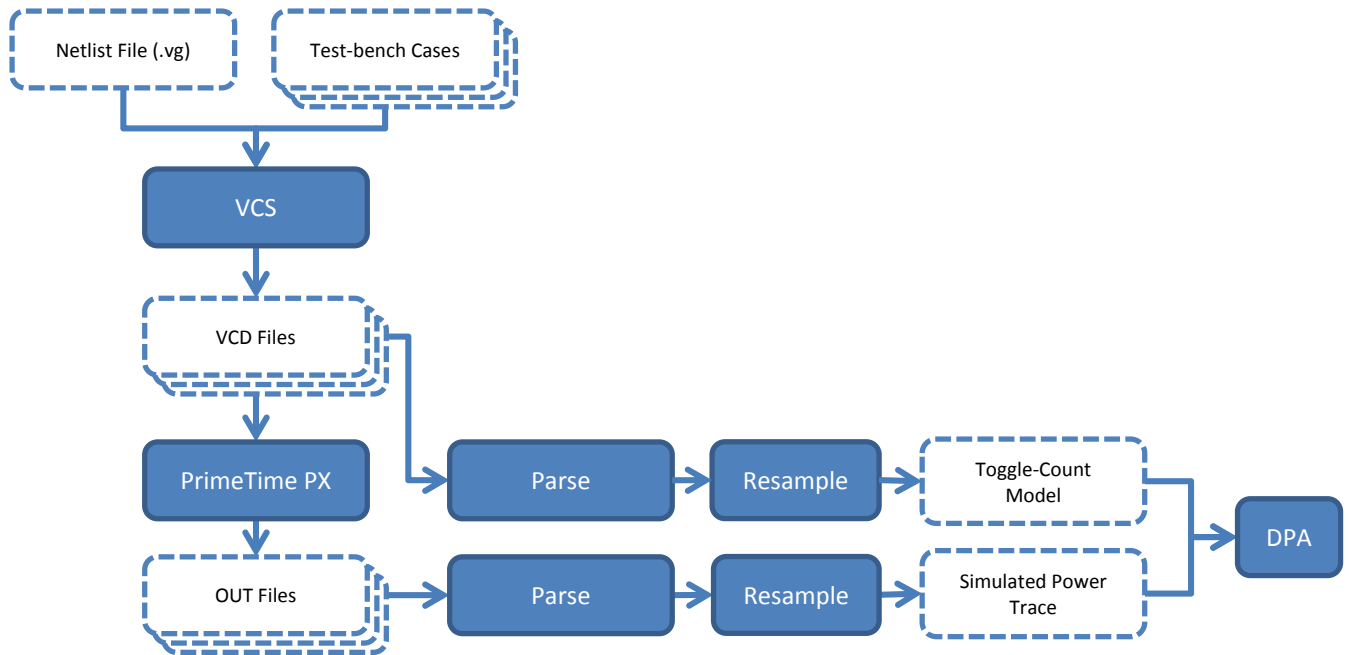


Figure 8: PSCARE Flow Diagram

The final step is to run all of these power traces through a DPA tool which performs a comparison of the traces and targets the secret key in order to gauge how much information leaks from the core through the power side-channel. The final result can be shown graphically as a correlation of power consumption to the secret key versus time, or as a metric, such as the number of traces an attacker would require to perform a successful side-channel attack, the signal-to-noise ratio of the leaking information, or the correlation of a successful DPA hit. Figure 8 shows a visual layout of the PSCARE tool flow.

2.9 Gate-Level Information-Flow Tracking (GLIFT)

GLIFT was originally proposed by researchers at the University of California, Santa Barbara in 2009 as a means of monitoring and controlling the flow of sensitive information at the hardware level of a given electronic design [27]. Before then, most information flow control methods focused on the software level of an electronic device. Tiwari et al. argue that these others techniques “cannot bring the system significantly closer to a formally strong notion of information flow tracking because they do not take into consideration the intricate logic and timing that compose the implementation” [27]. Since all information flow can be broken down to

the gate-level, this bottom-up approach ensures that proper information flow tracking can be obtained.

Since GLIFT focuses on the logic level of a design, it is possible to determine the flow of sensitive information within a gate-level netlist of an ASIC design. If a group of cells leak information, their power consumption will leak information to some degree as well. Thus, GLIFT helps to highlight areas of concern in an ASIC design when looking for power side-channel vulnerabilities.

A tool was developed to implement GLIFT as means to highlight parts of the design that leak information in general. Figure 9 shows the organization of the tool.

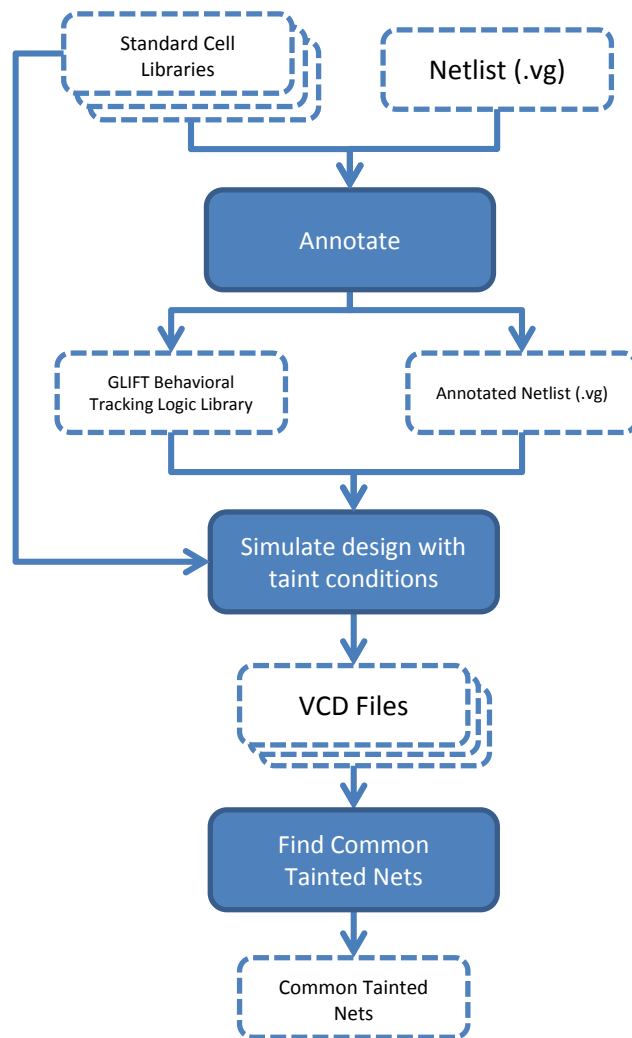


Figure 9: Gate-Level Information Flow Tracking (GLIFT) Tool Flow

This tool can create and maintain a library of tracking cells that are appended to a given standard cell as defined in a standard cell library. The tools take a netlist and the standard cell libraries that were used to generate the netlist and determine the appropriate tracking logic to append to each net in the netlist. A library of Verilog behavioral models describing the tracking cells is also generated and maintained for future reference. This results in a netlist with extra tracking logic that is only used for simulation purposes; no physical characteristics are present in the tracking cell descriptions. This newly annotated design is then simulated with varying tainted inputs, resulting in a collection of VCD files. Users can specify which signals to vary and the kind of variation (e.g. taint one bit at a time, do a byte-wise increment, keep the value constant). The VCD files are then parsed to find the list of nets that became tainted in each simulation. The intersection of each of these lists is then found to determine the list of nets that were consistently tainted across all simulations, resulting in a symbolic dependency between the tainted signals and the nets in the design for a given signal that had various combinations of bits tainted across the simulations. These nets can then be focused on for power analysis since they were the only ones found to leak information of a specified signal.

Designers can use GLIFT to highlight parts of their design that leak information of a given signal (such as a key) that should then be analyzed by PSCARE. This would optimize how much of the design PSCARE needs to analyze as PSCARE would only be applied to parts of the design that are known to leak information in general. This reduces simulation time since less hardware has to be analyzed and it ensures that designers can better focus on the vulnerable parts of their design.

2.10 Correlation Power Analysis Tool

The specific tool that was used to perform Correlation Power Analysis (CPA) was previously developed at The MITRE Corporation. However, this project involved much modification of the existing code to both improve current functionality and add additional functionality to the tool. The general flow of the power analysis tool can be seen in Figure 10.

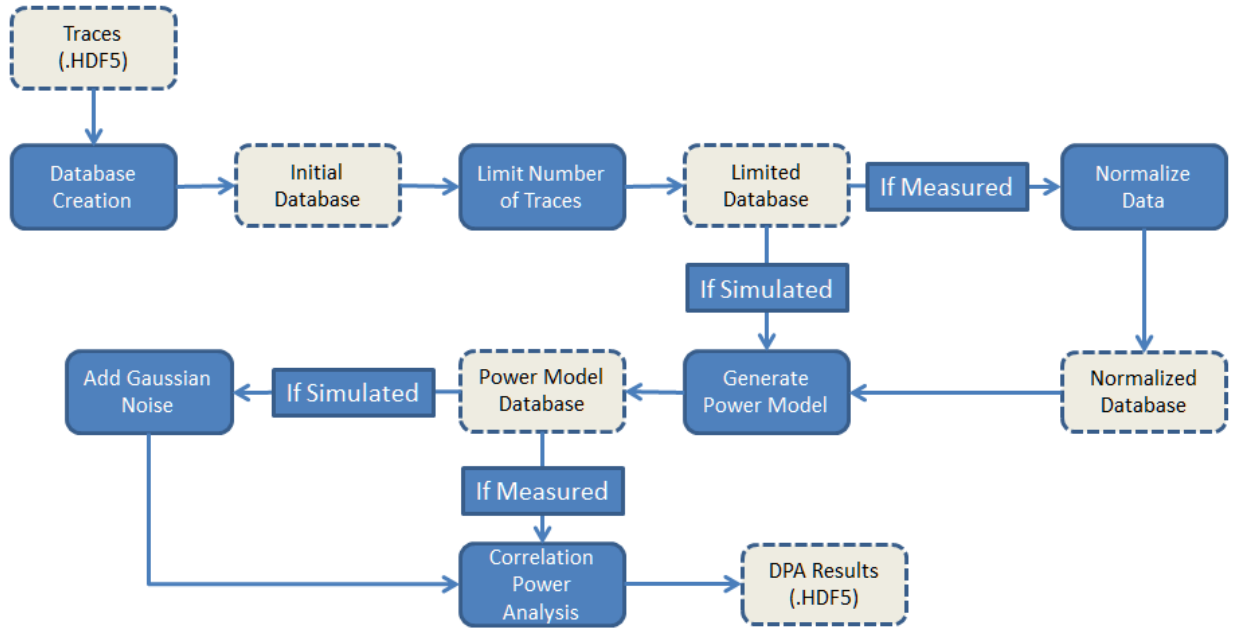


Figure 10: Flow Diagram of the Power Analysis Attack Tool

3 Objectives

The goals of this project are as follows:

1. Evaluate the efficacy of MITRE's Power Side-Channel Attack Risk Evaluator (PSCARE) to produce realistic power analysis results in simulation. The purpose of PSCARE is to simulate power consumption values for varying input values, and then perform a power-analysis attack at design-time upon a design. Evaluation will consist of comparing the simulated attack results to actual attack results conducted using measured data collected from ASIC implementations of the Verilog source code.
 - a. In order for ASIC designers to confidently adopt this tool, the data generated by PSCARE must sufficiently match the data that would otherwise be found in a lab setting.
2. Evaluate how well PSCARE highlights areas of concern in designs of various sizes using MITRE's Gate-Level Information-Flow Tracking (GLIFT) tool.
 - a. Given how large and complicated designs can become, it is necessary that the PSCARE tool flow successfully narrow down the area of concern in an ASIC design for maximum processing efficiency.
3. Create documentation for ASIC designers describing the PSCARE tool flow and how it can be incorporated into their design flow, as well as documenting the GLIFT functionality.

4 Methodology

The main focus of this project will be the testing and verification of the power side-channel leakage evaluation tool known as Power Side-Channel Attack Risk Evaluator (PSCARE). PSCARE will be applied to only the nets of interest in an RTL design, as determined by another developed module that will also be evaluated: Gate-Level Information Flow Tracking (GLIFT). The basic process to be carried out in this project can be categorized into six distinct stages:

1. Simulate the Power Traces of a Design

We applied PSCARE to simulate the composite-field, PPRM3, and look-up table S-Box RTL designs of AES that were implemented on the SASEBO-R. This resulted in a collection of simulated power traces.

2. Simulate the Power Traces of a Design Filtered with GLIFT

We applied GLIFT to the composite-field, PPRM3 and look-up table S-Box RTL designs of AES that were implemented on the SASEBO-R. This resulted in a filtered list of nets that were determined to leak information. Using a one-hot technique, where individual bits of the key were tainted one-at-a-time, 128 distinct VCD files were produced. A set comparison was then performed to determine which cells were consistently tainted throughout all simulations. These cells were the ones targeted by GLIFT. The power consumption of these nets were then be simulated to generate power traces of only the areas of concern of the original design.

3. Conduct a DPA Attack on the Simulated Power Traces

The collections of power traces from PSCARE were then run through a Differential Power Analysis (DPA) tool, producing correlation results that revealed the secret key used for each encryption algorithm that was being targeted. The DPA attack method involved targeting the final round key, and correlating the power consumption of the final round of encryption with a Hamming Distance model of the input-to-output of this final round.

4. Measure Power Traces of the Physical Implementation of the Design

Collected power traces of the physical composite-field, PPRM3 and look-up table S-Box implementations of AES on the SASEBO-R's LSI. These power traces were generated using the same input values as in simulation. Figure 11 shows the capture setup that was used. The oscilloscope used was a Tektronix DPO7254 model. The traces were captured from the oscilloscope at a sampling rate of 1 GS/s. This value was originally chosen to be consistent with the 1 GHz sampling rate that was to be used in simulation. Ideally, the traces from simulation and those actually captured from the SASEBO would be the same length and have the same sampling frequencies. However, the SASEBO documentation was unclear about the clock speed of the actual cryptographic LSI. While the controlling FPGA on the board ran at 24 MHz, the actual cryptographic core was running at 3 MHz. In addition to this, the test-benches that were written to simulate these AES designs were clocked at a rate of 100 MHz. All told, the captured data was taken at a rate of 333.33 samples / clock period. The simulated data was taken at a rate of 10 samples / clock period. This discrepancy is unfortunate, however the results are still conclusive. If a future test was to be done, it would be important to precisely match these sampling rates for a more accurate correlation of simulation to reality.

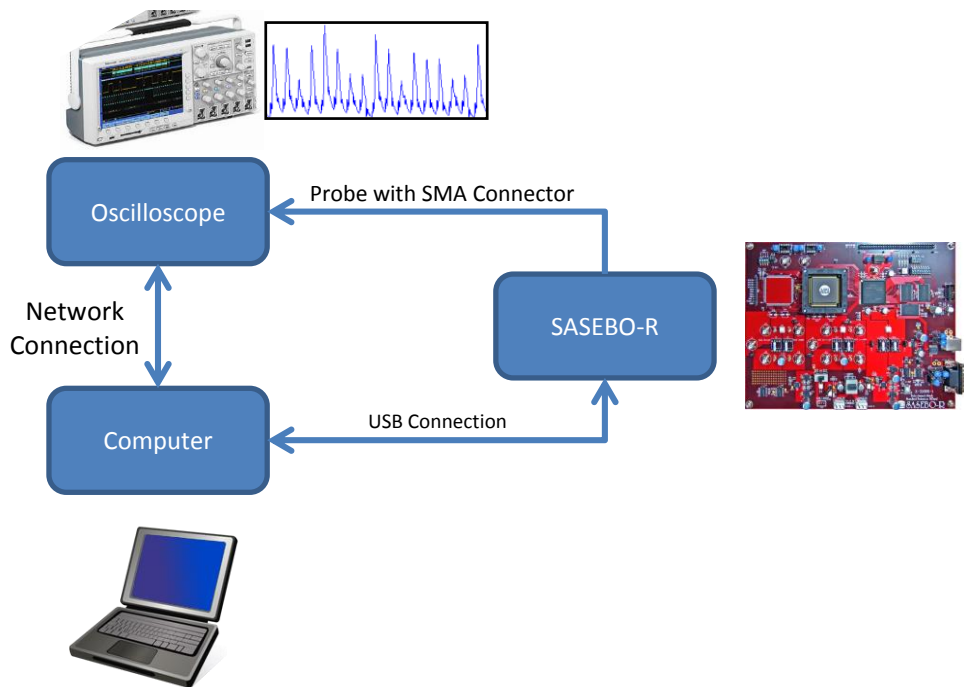


Figure 11: Trace Capture Setup for SASEBO-R

A computer is connected to an oscilloscope via a network connection and to a SASEBO-R via a USB connection. A channel of the oscilloscope is connected to the SASEBO-R with a probe that has an SMA connector. A separate channel of the oscilloscope is used to monitor the start (trigger) signal of the SASEBO-R. The C#-based LSI checker that was provided with the SASEBO-R was repurposed to specify which core will encrypt a given plaintext with a set key at a given time. This repurposed program was incorporated into a script that would perform the following steps from the computer:

1. Arm the trigger on the oscilloscope;
2. Set the plaintext, key and cryptographic core on the SASEBO-R;
3. Tell the SASEBO-R to begin its encryption process; and
4. Store the oscilloscope data on the computer for analysis.

The data collected from the oscilloscope was formatted the same way as the simulated data generated by PSCARE to allow easy and consistent analysis.

5. Conduct a DPA Attack on the Measured Power Traces

The power traces acquired from the SASEBO-R were then processed with the same DPA attack tool that was used on the simulated power traces generated by PSCARE and GLIFT. The same Hamming Distance model used for the simulated power traces was also used for the measured power traces.

6. Compare the Results of the DPA Attacks

The results of the DPA attack conducted on the simulated power traces was compared with that of the power traces measured on the SASEBO-R. Specifically, the DPA of the power traces of the original simulated design (step 1) was compared to the DPA of the measured traces (step 4). Subsequently, the DPA conducted on the power traces of the designs that were run through GLIFT (step 2) was compared to the DPA of the measured power traces (step 4). The correlation values of the DPA hits in the measured and simulated attacks were compared. Additionally, the number of traces necessary to extract the secret key of a given AES implementation via DPA of the measured and simulated

power traces were used to determine the efficacy of PSCARE and GLIFT. These metrics were also used to evaluate the overall power side-channel security of each of the designs.

5 Results

The results of the project are introduced in this section. This reviews the data that was captured from the SASEBO-R and simulated designs as well as the results of the DPA attacks that were conducted.

5.1 Initial Measured and Simulated Trace Results

The Composite Field implementation of AES is used to introduce the concepts in this section, but the same procedures apply for all of the AES designs tested both in simulation and with laboratory measurements.

5.1.1 Measured Traces and Normalization

A sample voltage trace collected from the SASEBO-R via the oscilloscope is shown in Figure 12. This trace is the voltage across a small test resistance inserted between the ground pin of the cryptographic LSI and the actual ground of the board.

$$P = I * V$$

In accordance with the power equation, where electrical power is equivalent to the product of current and voltage, power is directly proportional to voltage for a fixed current, while power is proportional to the square of voltage for a fixed resistance.

$$P = \frac{V^2}{R}$$

Therefore the results of a power-analysis attack will be the same whether a voltage or power waveform are used in the attack. As long as the values used are consistent for a single attack, the correlations will still occur, regardless of the units of the value of interest.

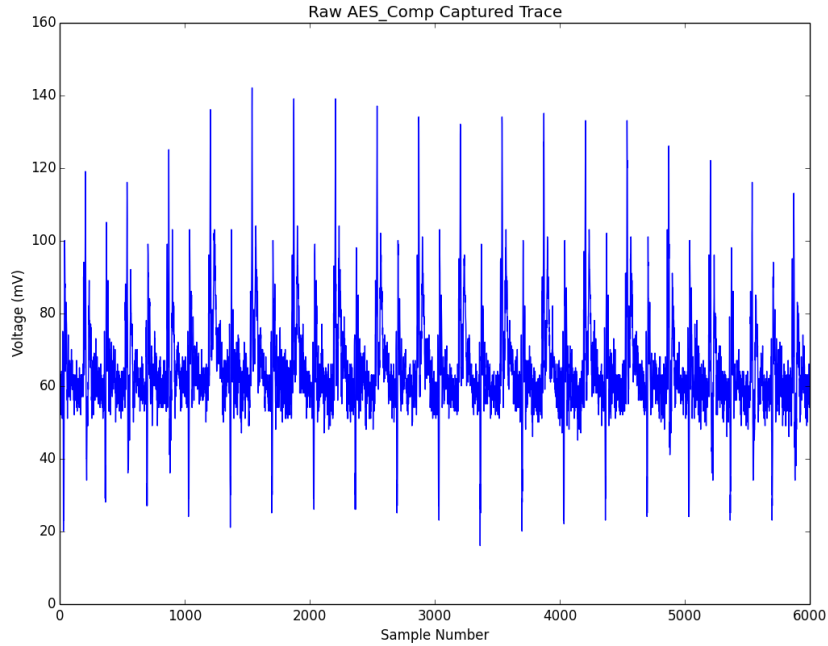


Figure 12: Sample Voltage Trace Captured from SASEBO-R

It can be seen in this figure that there is a sequential operation being performed. Each peak represents one of the 10 rounds of AES. Figure 13 shows multiple voltage traces from the SASEBO-R for varying plaintexts overlaid.

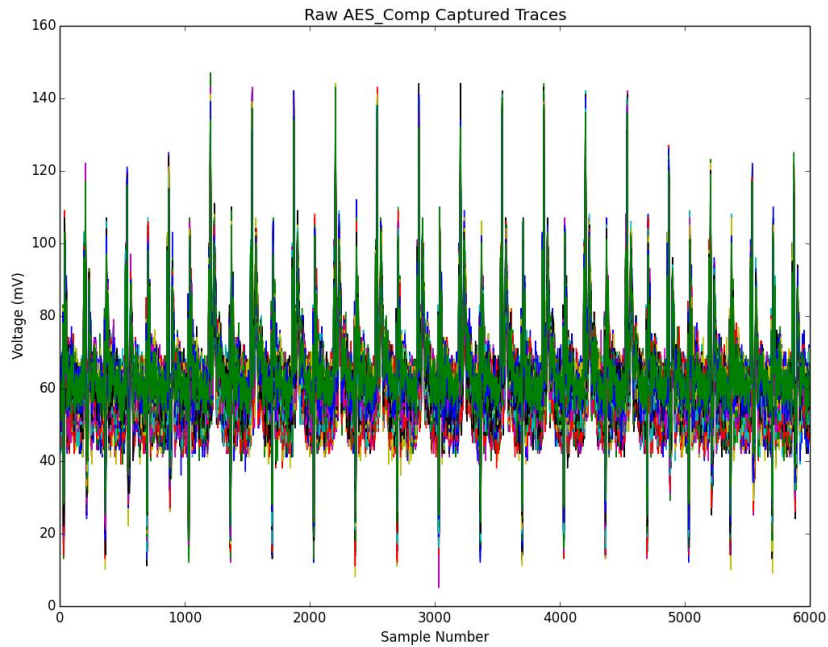


Figure 13: Voltage Waveforms from SASEBO-R Overlaid for Varying Plaintexts

While noise is present in all of the traces, this nevertheless suggests some sort of dependency between the plaintext being encrypted and the power the cryptographic core is consuming.

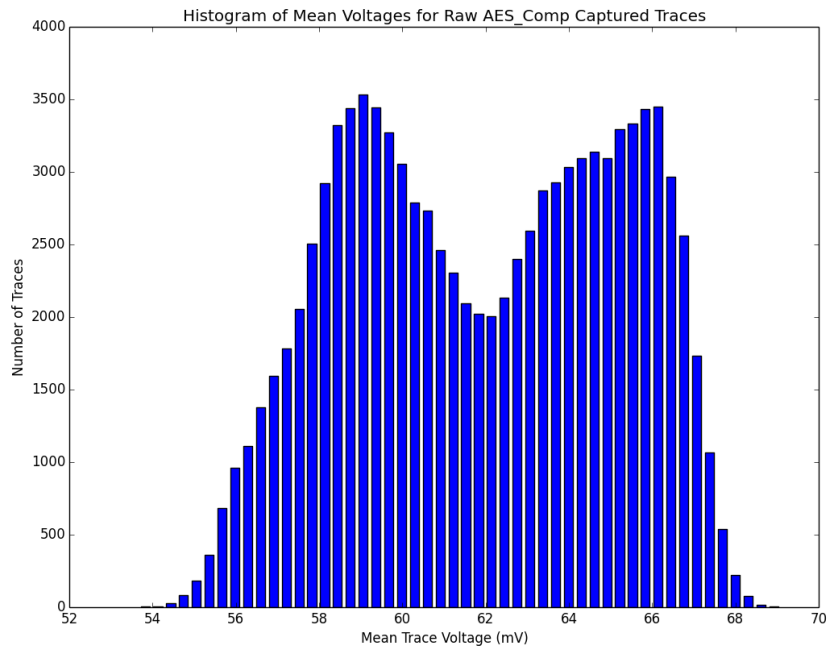


Figure 14: Histogram of Mean Voltages for Traces Captured from SASEBO-R

To conduct a DPA, a large number of measurements is needed for each of the cryptographic cores. The collection process for 100,000 traces took approximately 30 hours, which led to an interesting observation. The air conditioning in the testing environment shuts off every evening, changing the ambient temperature that the cryptographic core was operating in. Figure 14 shows a histogram of the mean trace voltages for the raw collected data. It is obvious that this does not exhibit the ideal Bell curve as expected.

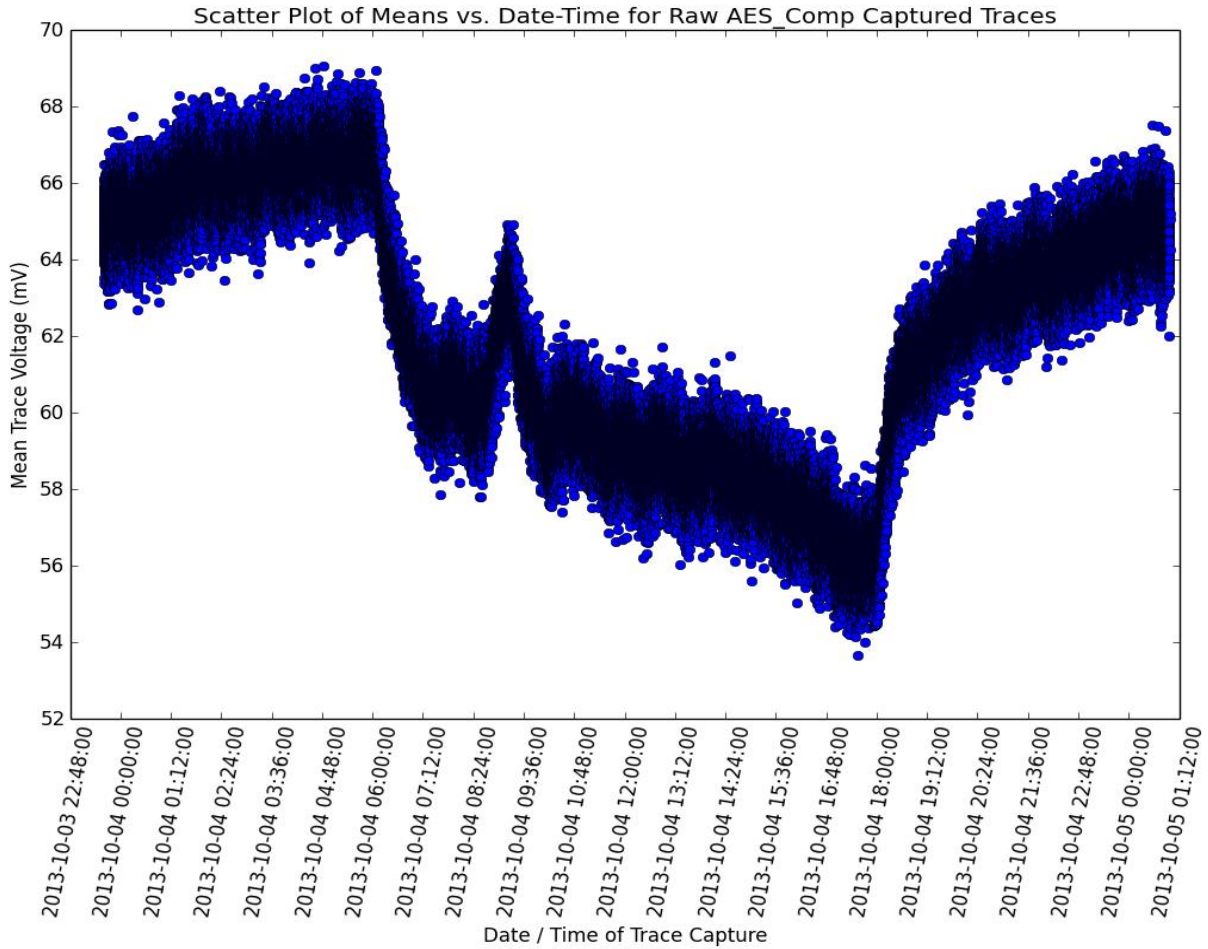


Figure 15: Time-Stamped Scatter Plot of Trace Means for Captured Traces

Also shown in Figure 15 is a time-stamped scatter plot for the mean trace voltages. There are two distinct points where the trend of the mean trace voltages changes, and one stray peak. These were identified to be the two set-points for the air conditioning, at which point the ambient temperature would begin to change.

This shift in the mean trace voltage was compensated for by removing the mean from each of the measured traces. This was done by subtracting the average of each power trace from each of its individual data points. An overlaid plot of the normalized voltage traces is shown in Figure 16.

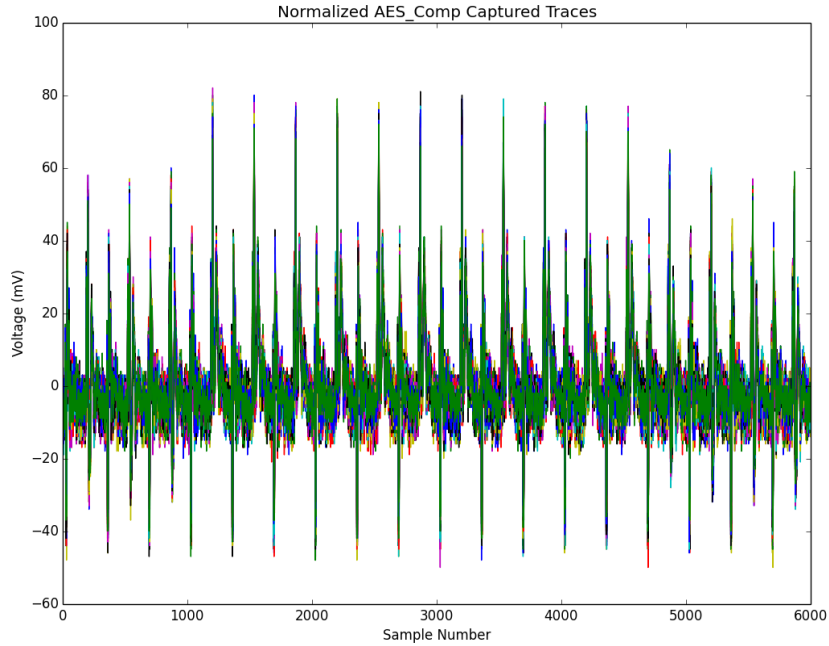


Figure 16: Normalized Voltages Traces for SASEBO-R Captures

The histogram and time-stamped scatter plot for the normalized data are shown in Figure 17 and Figure 18, respectively.

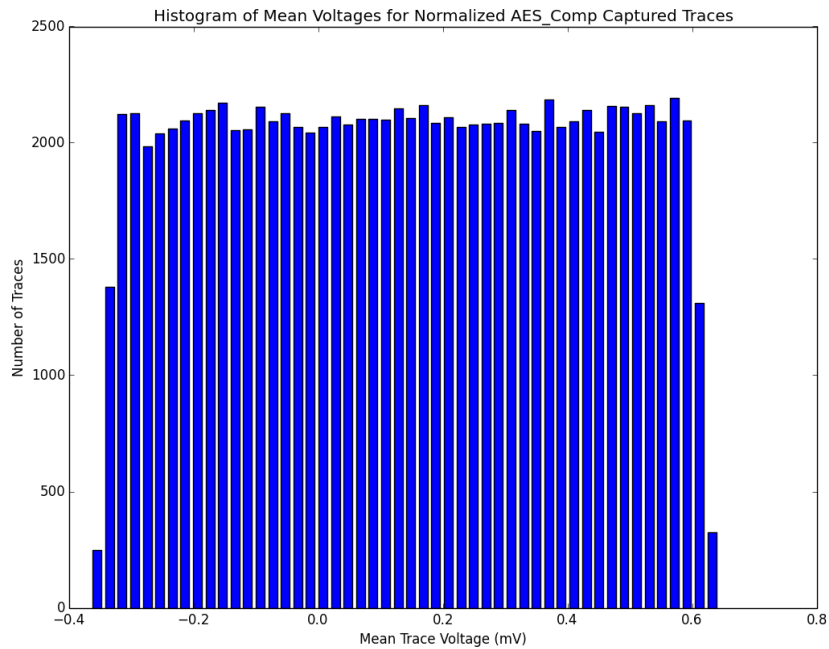


Figure 17: Histogram of Trace Mean for Normalized SASEBO-R Captures

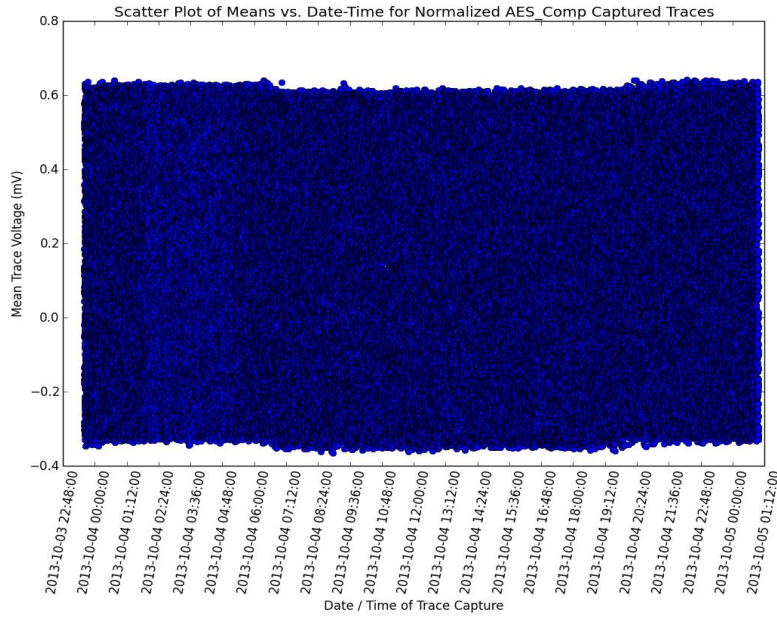


Figure 18: Time-Stamped Scatter Plot of Trace Means for SASEBO-R Captures

5.1.2 Simulated Traces and Introduction of Gaussian Noise

The traces produced from the PSCARE simulation flow represent the same encryption process as the measured traces, but are inherently noiseless because they are all perfectly aligned and have no outside factors affecting the measurements. The simulation does not account for random noise from the environment and other variables. Multiple simulated power traces overlaid are shown in Figure 19.

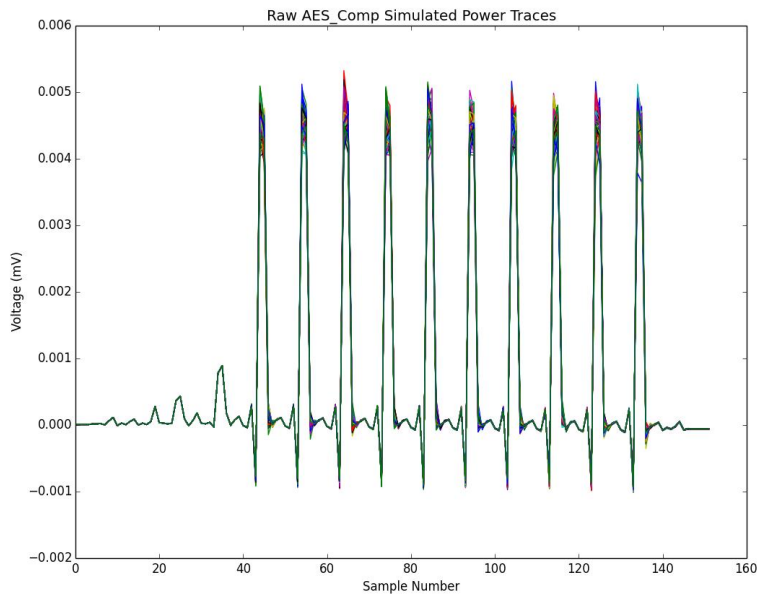


Figure 19: Simulated Power Traces Overlaid for Varying Plaintext Values

It is evident that there is no deviation during certain periods of the simulation, but the power consumption does vary as expected during the actual encryption process. This is due to the data-dependent power consumption, exactly what a Power Analysis focuses on. A side-effect of this noiseless simulation becomes apparent during a CPA attack: the signal-to-noise ratio of the correct-key guess occasionally goes to zero, as shown in Figure 20. This is due to a mathematical error when there is zero variance for a specific point in time, which would occur frequently during a noiseless simulation.

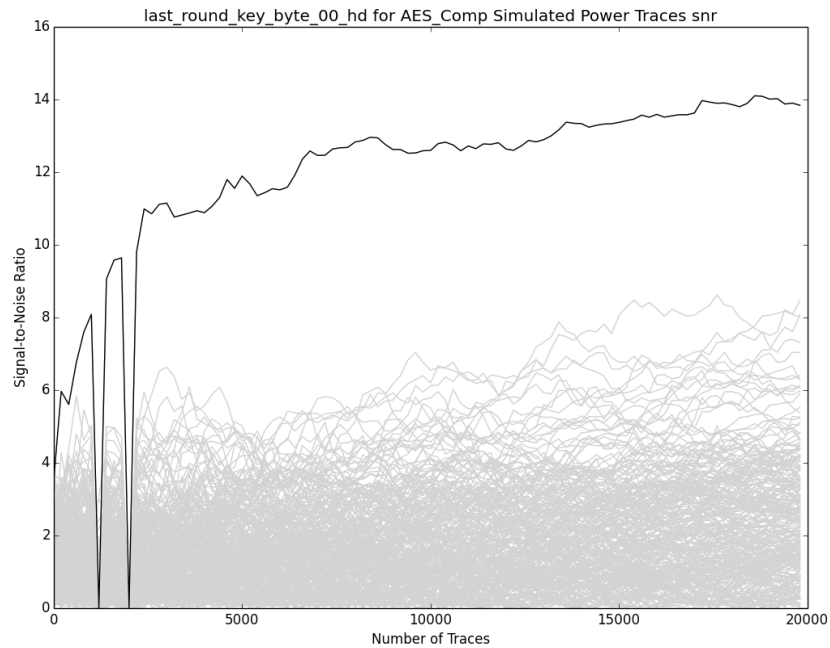


Figure 20: Signal-to-Noise Ratio for Correct Key Guess for Noiseless Simulation Data

This processing error is most likely due to comparison errors stemming from the precision problems associated with using float-type values. A simple remedy to this problem is to add a very small amount of Gaussian noise to each simulated power trace. This was done by taking a percentage of the peak-to-peak value of the initial trace and then adding noise centered around zero with that standard deviation. A value of 19% noise was added to the simulated traces. These traces are still perfectly aligned on the time axis, but now show a more realistic amount of noise, as would be expected with a physical capture setup. A simulated power trace with added Gaussian noise is shown in Figure 21.

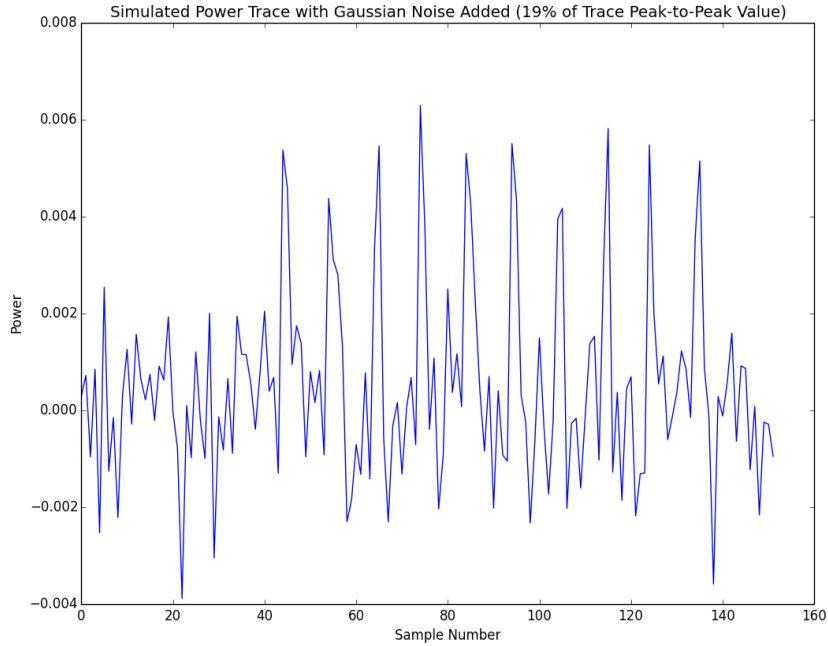


Figure 21: Simulated Power Traces with 19% Gaussian Noise Added

The signal-to-noise ratio for the correct-key guess is shown in Figure 22 for the simulated traces with added Gaussian noise. The errant data points have been corrected while still retaining the correct structure of the results.

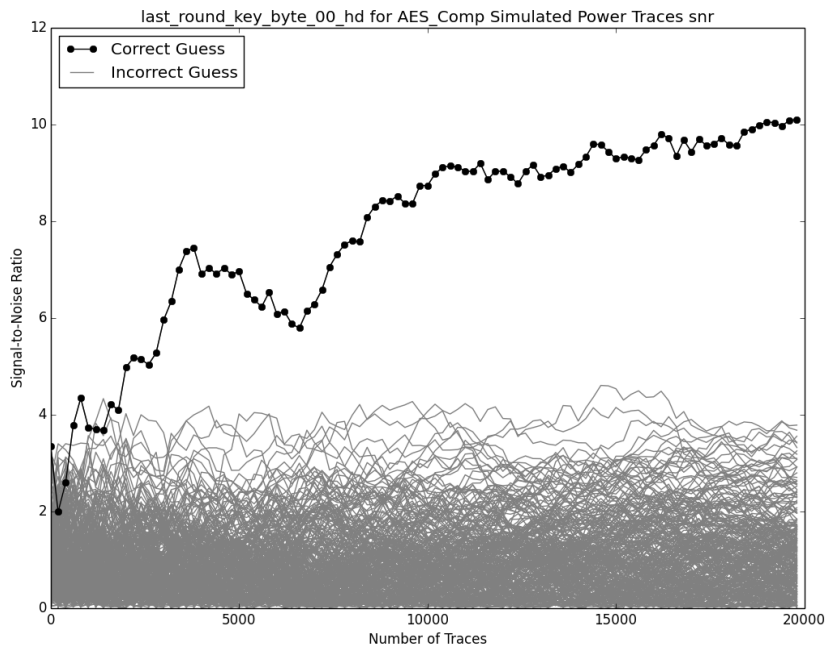


Figure 22: Signal-to-Noise Ratio for Correct Key Guess for Noisy Simulation Data

5.2 DPA Results: Simulated vs. Measured

The DPA attack used for this project was an attack on the 10th round key. This attack takes advantage of the fact that the final round of AES does not include the Mix Columns step, which would then require 4 bytes of the key to be guessed at a time. If an attacker is able to obtain any of the full round keys, it is a direct transformation from any round key back to the original key, thus breaking the encryption. The tenth round consists of a Sub-Bytes (S-Box) step, followed by Shift Rows, and finally an Add Round Key step. To perform this final round attack, an attacker would use a chosen byte of the ciphertext to calculate a byte of the input to the final round. This can be done by performing an exclusive OR operation with the ciphertext byte and the final round key guess byte, and then performing an inverse S-Box operation. The Shift Rows step simply affects which byte of the ciphertext is used to attack a specific byte of the final round key. This attack was successfully performed on all three of the AES designs that were implemented on the SASEBO-R, and the results are explained in the following sections. All simulated DPA attacks were performed with 19% Gaussian noise added to the power traces. The results outlined below are only for key byte 0, but all 16 bytes were successfully attacked, with the plots being shown in Appendix C: Full DPA Results for AES Comp.

5.2.1 Composite Field Implementation of AES

For AES Comp, the correlation vectors for all 256 key candidate guesses for both measured data and simulated data are shown in Figure 23.

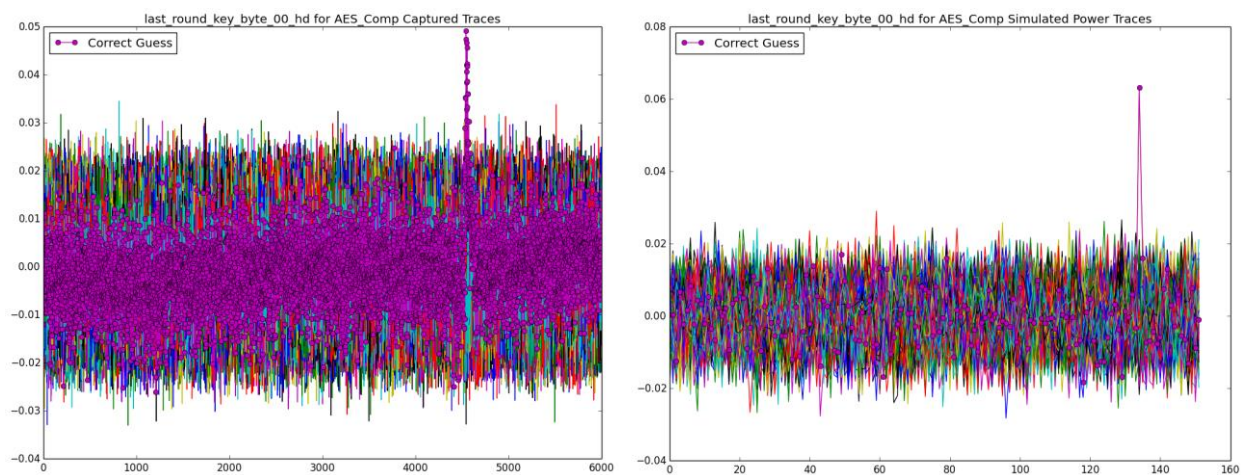


Figure 23: DPA Results for AES Comp Measured (Left) vs. Simulated (Right)

As shown in Figure 24, the correlation values for the correct key candidate are plotted in black. This plot shows the correlation values for a specific key hypothesis versus the number of traces used for the attack. The results from both DPA attacks, using the measured and simulated data, the correct key hypothesis correlation rises well above the incorrect key hypotheses (plotted in gray). The dashed blue line represents the ratio of the correlation of the correct key hypothesis to the maximum incorrect key hypothesis at each point in time. This metric is used to gauge the separation between the correct and incorrect hypotheses.

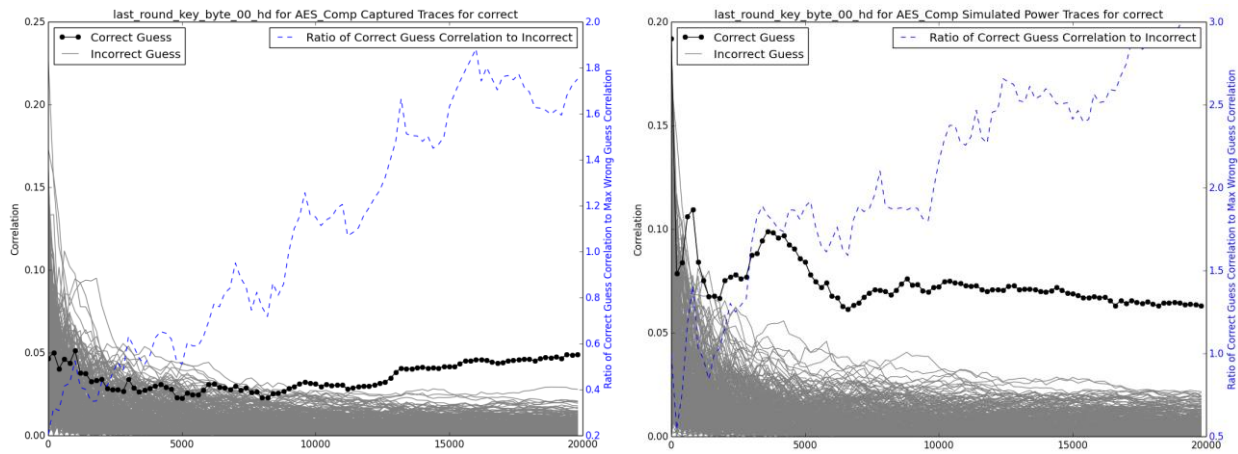


Figure 24: Correlation Values for all 256 Key Candidates vs. Number of Traces Collected

After plotting the ratio of correlations, one further calculation was required to relate the two DPA attacks to each other. As shown in Table 1, the correlation of these ratios was calculated for all 16 key byte attacks. The majority fall within a range of 75 to 95% correlation. However, there are a few outliers, such as key byte 4. This had a correlation of 50% between simulation and reality. This is likely because the random Gaussian noise dominated the power trace and led to less-than-ideal DPA results. Also note that although the Y-Axis scales are different between the two plots, they still represent comparable results. In summary, the following table represents the correlation of the ratios for the power analysis attack results for the measured data versus the simulated data. A ratio of the correlation of the correct key hypothesis to the next-highest incorrect key hypothesis was used to provide a consistent and measurable metric for a power analysis attack result.

Key Byte Attacked	Correlation of Measured DPA Ratio to Simulated DPA Ratio
0	91.2%
1	84.9%
2	87.7%
3	93.6%
4	50%
5	87.9%
6	76.1%
7	94.6%
8	64.3%
9	85.8%
10	88.0%
11	81.8%
12	93.8%
13	74.2%
14	93.7%
15	87.7%

Table 1: Correlation of Measured to Simulated DPA Results for AES Comp

Finally, a metric to approximate the signal-to-noise ratio for each key hypothesis was defined as follows:

$$SNR_t = \frac{abs(x_t)}{\left(\frac{\sum abs(x)}{N}\right)}$$

An approximation of the signal-to-noise ratio for a particular key hypothesis at a given instant in time, t , is given by SNR_t , where x_t is the value of that key hypothesis's correlation at point t . The absolute value of this point is then divided by the mean of that entire key-hypothesis correlation trace. This is calculated by taken the sum of the absolute values of each element, and then dividing by the number of correlation values in the trace. This equation focuses on a single time instant in the attack. For the attack used in this paper, the Round 10 Key attack, the time of interest was during the final round of encryption. The exact instant was calculated by finding the index at which the correct key hypothesis correlation trace with 20,000 traces was at its absolute maximum. Then, to calculate the SNR, for each key hypothesis correlation trace, the absolute value of the point at the time of interest was compared to the mean of the absolute values of that trace. This process was repeated for each key hypothesis at each sequential stage of the DPA attack. This means that the final result was a plot of the signal-to-noise ratios for all 256 key hypotheses after DPA attacks with 200 to 20,000 traces used. These plots are shown in Figure 25, with the results from measured data shown on the left, and simulated data shown on the right.

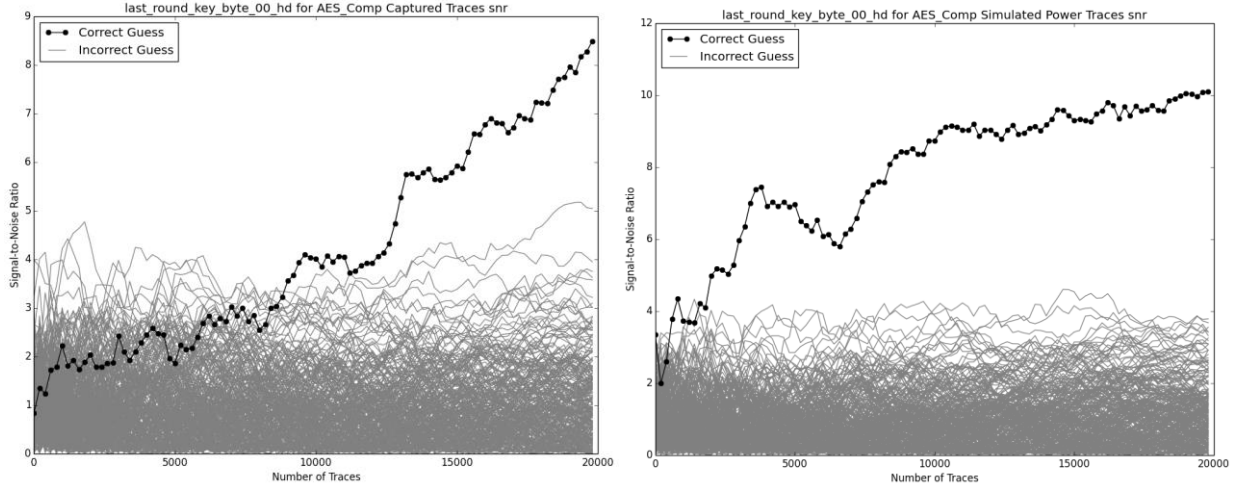


Figure 25: Signal-to-Noise Ratio For Correct Key Guess vs. Number of Traces Collected

The procedures and calculations discussed above were used to analyze the results of the DPA attacks for all three AES implementations that were examined during this project. The following two sections present the results of these calculations for the other two designs, AES PPRM3 and TBL.

5.2.1 Three-Stage Positive-Polarity Reed Muller Implementation of AES

Full attack results for the PPRM3 implementation of AES can be seen in Appendix D: Full DPA Results for AES PPRM3.

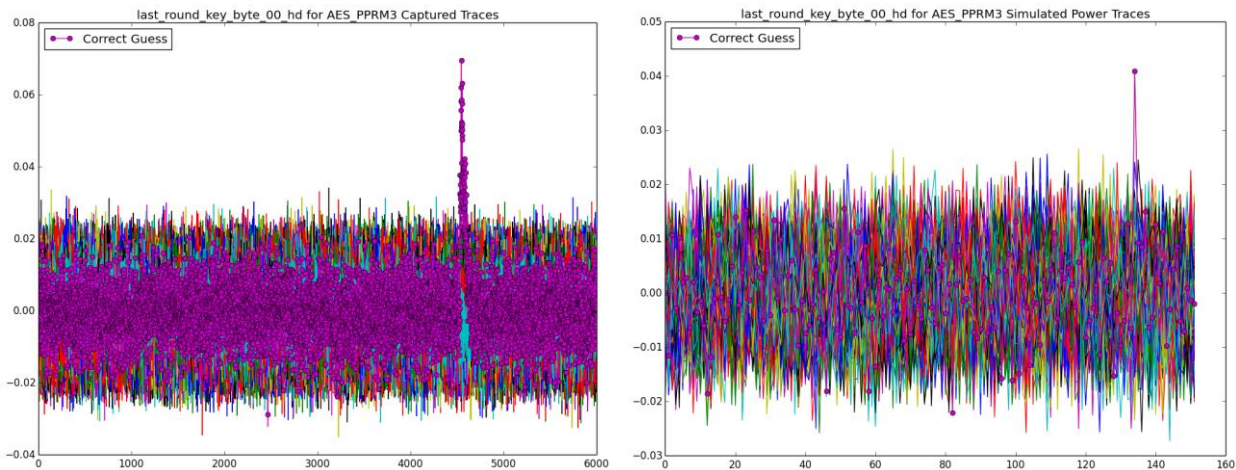


Figure 26: DPA Results for AES PPRM3 Measured (Left) vs. Simulated (Right)

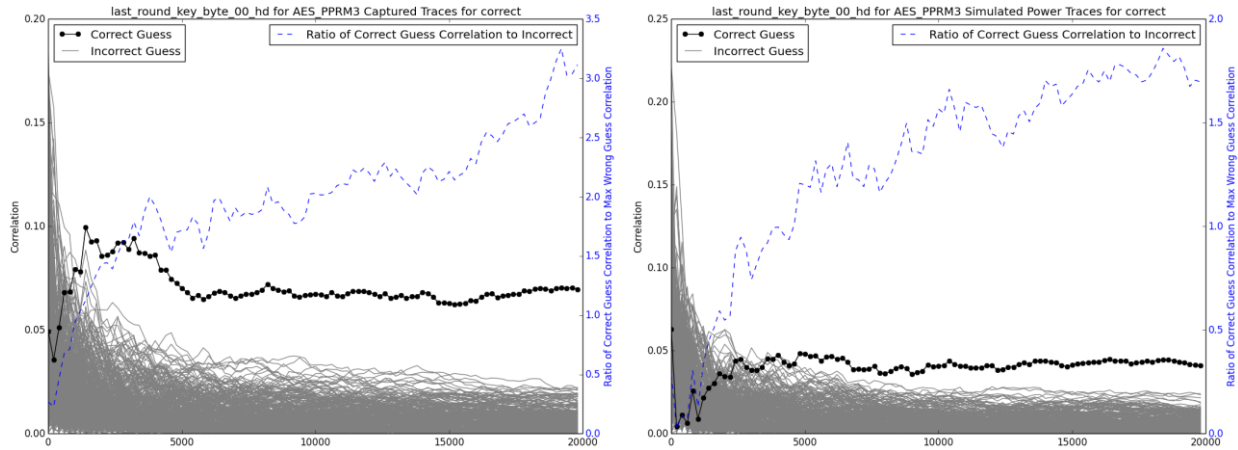


Figure 27: Correlation Values for all 256 Key Candidates vs. Number of Traces Collected

Key Byte Attacked	Correlation of Measured DPA Ratio to Simulated DPA Ratio
0	90.0%
1	59.2%
2	86.0%
3	71.8%
4	87.0%
5	87.3%
6	84.7%
7	93.4%
8	82.9%
9	89.6%
10	78.1%
11	76.0%
12	47.6%
13	67.0%
14	95.6%
15	92.6%

Table 2: Correlation of Measured to Simulated DPA Results for AES PPRM3

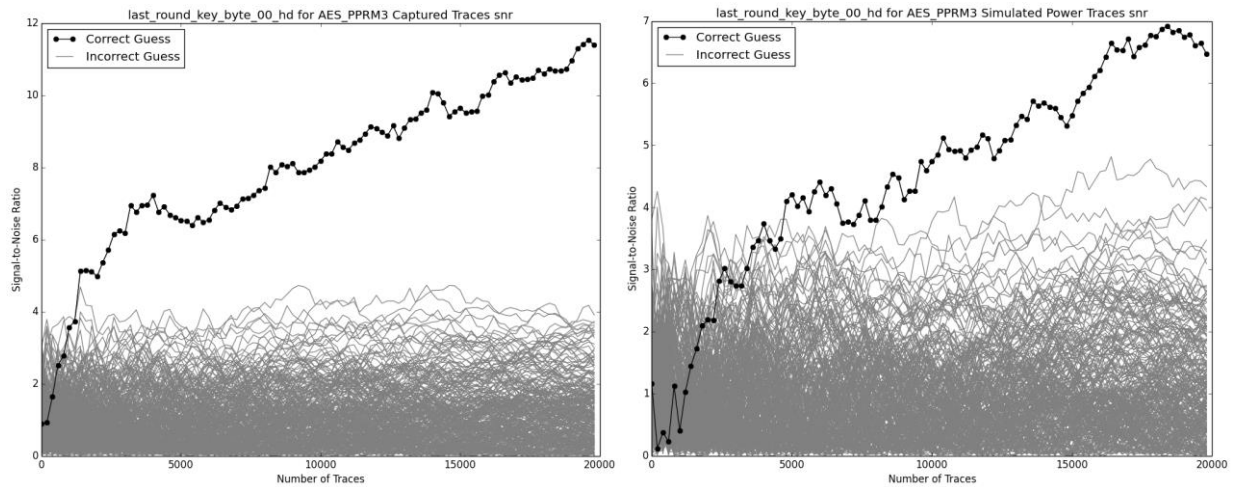


Figure 28: Signal-to-Noise Ratio For Correct Key Guess vs. Number of Traces Collected

5.2.1 Lookup Table Implementation of AES

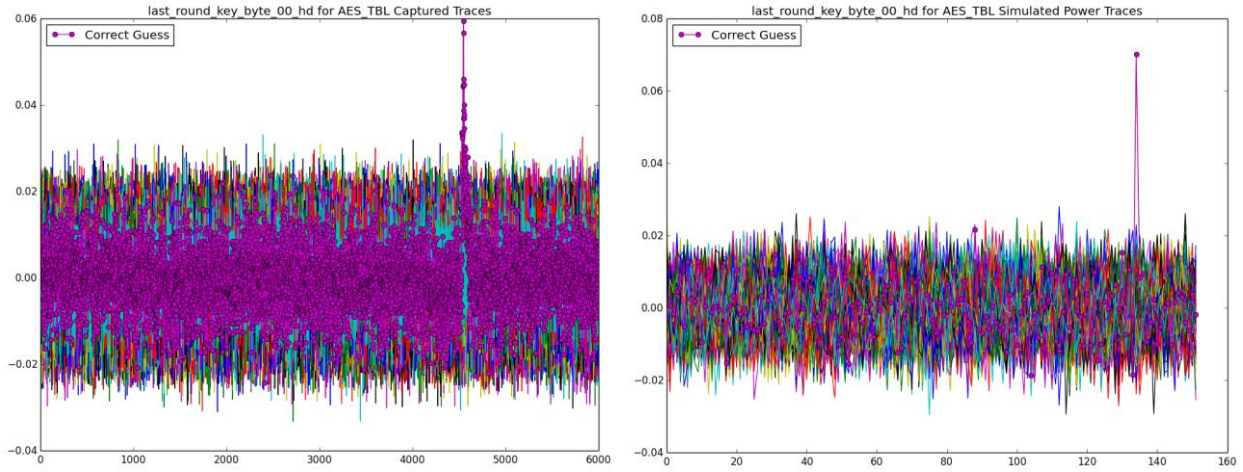


Figure 29: DPA Results for AES TBL Measured (Left) vs. Simulated (Right)

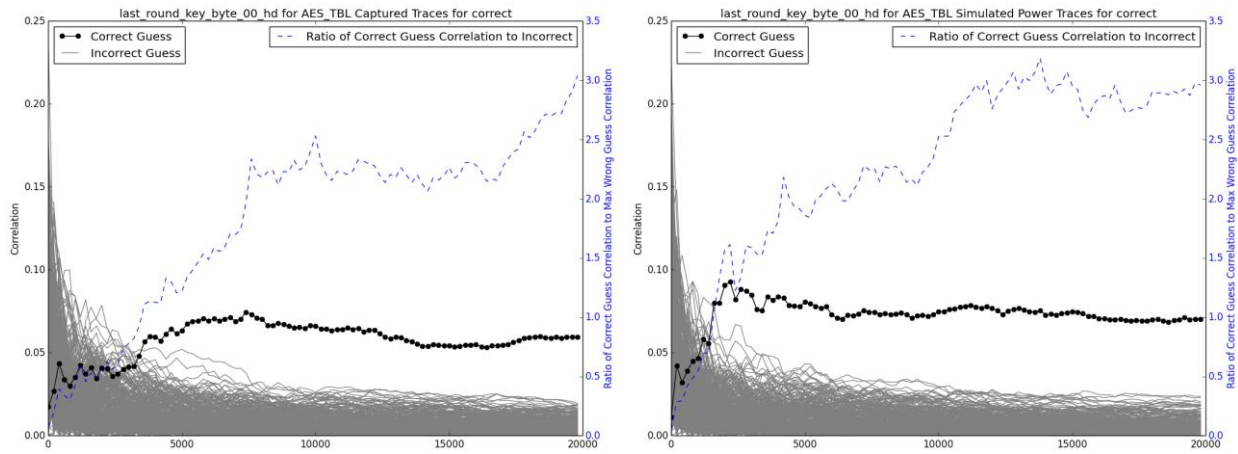


Figure 30: Correlation Values for all 256 Key Candidates vs. Number of Traces Collected

Key Byte Attacked	Correlation of Measured DPA Ratio to Simulated DPA Ratio
0	90.9%
1	81.4%
2	94.9%
3	92.0%
4	95.3%
5	88.7%
6	91.5%
7	57.9%
8	86.3%
9	83.3%
10	93.5%
11	88.2%
12	84.9%
13	95.0%
14	91.0%
15	91.0%

Table 3: Correlation of Measured to Simulated DPA Results for AES TBL

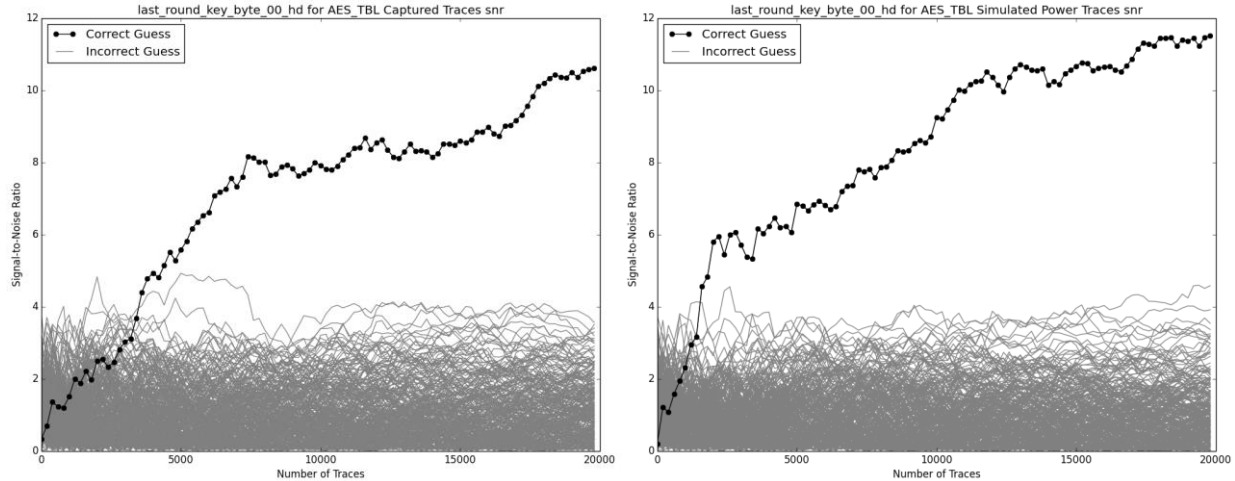


Figure 31: Signal-to-Noise Ratio For Correct Key Guess vs. Number of Traces Collected

5.3 Impact of GLIFT upon Simulated Designs

GLIFT was applied to each of the AES designs in various ways in order to determine which cells leak information about a given signal. This list of cells can then be passed on for analysis with PSCARE. The cells that were excluded from the designs were also examined to verify that GLIFT was indeed providing an accurate list of cells that leak information on a logic level. Between all of the designs, only the Composite Field implementation of AES had both an encryption and decryption block. Thus, this design is the best available design to use to evaluate how well GLIFT filters cells that do not leak information for a given signal. This section will delve into the efficacy of GLIFT on each of the AES designs used throughout this project.

5.3.1 Lookup Table Implementation of AES

The Lookup Table implementation used only had an encryption block, mitigating the impact of GLIFT. The design was simulated 128 times where, during each simulation, a bit of the key was tainted. Both the key and plaintext were set to the hexadecimal value “00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F.” Table 4 shows the cell count breakdown of analyzing the VCD files.

	Total Cells	Consistently Tainted Cells	Consistently Untainted Cells	Intermittently Tainted Cells
Lookup Table	9,864	9,290	61	513

Table 4: Breakdown of Cell Count by GLIFT when Analyzing the Key for the Lookup Table Design

Thus, 95.6% of the cells were found to consistently leak information. The results of the DPA conducted on this new group of cells was compared to the results found for the original design. The correlations were found to be so similar that the differences could not be discerned between the various plots being used. This stems from the fact that so little of the design was filtered out.

In order to further evaluate GLIFT, another 128 simulations were conducted where each bit of the plaintext was tainted. The key and plaintext values themselves were kept the same as before. The VCD files were then analyzed with the same method as before to find a list of cells that were consistently outputting tainted information. Table 5 shows the results of this analysis.

	Total Cells	Consistently Tainted Cells	Consistently Untainted Cells	Intermittently Tainted Cells
Lookup Table	9,864	9,290	445	129

Table 5: Breakdown of Cell Count by GLIFT when Analyzing the Plaintext for the Lookup Table Design

Thus, according to the table, the number of cells found to be consistently tainted did not change, but the number of cells found to be consistently untainted had increased. This ties into the fact that parts of the cryptographic core dealing with the key were not necessarily tainted by the plaintext value itself, be it because of the constant values that were used or the inherent structure of the design.

5.3.2 Three-Stage Positive-Polarity Reed-Muller Implementation of AES

The Three-Stage Positive-Polarity Reed-Muller (PPRM3) implementation that was used only had an encryption block, mitigating the impact of GLIFT. The design was simulated 128 times

where, during each simulation, a bit of the key was tainted. Table 6 shows the cell count breakdown of analyzing the VCD files.

	Total Cells	Consistently Tainted Cells	Consistently Untainted Cells	Intermittently Tainted Cells
Three-Stage Positive-Polarity Reed-Muller	10,760	10,186	61	513

Table 6: Breakdown of Cell Count by GLIFT for the PPRM3 Design

Thus, 95.9% of the cells were found to consistently leak information. The results of the DPA conducted on this new group of cells was compared to the results found for the original design. Like was the case for the Lookup Table design, the correlations were found to be so similar that the differences could not be discerned between the various plots being used.

As was done with the Lookup Table design, 128 simulations were conducted where each bit of the plaintext was tainted. The key and plaintext values themselves were kept the same as before. The VCD files were then analyzed with the same method as before to find a list of cells that were consistently outputting tainted information. Table 7 shows the results of this analysis. The cells that were always tainted across the 128 simulations are shown in the third column, while those that were never tainted in any of the simulations are shown in the fourth column. The last column contains the cells that were tainted is some, but not all, of the simulations.

	Total Cells	Consistently Tainted Cells	Consistently Untainted Cells	Intermittently Tainted Cells
Three-Stage Positive-Polarity Reed-Muller	10,760	10,186	445	129

Table 7: Breakdown of Cell Count by GLIFT when Analyzing the Plaintext for the PPRM3 Design

Similar to the results found with the Lookup Table design, the number of cells found to be consistently tainted did not change, but the number of cells found to be consistently untainted had increased. This is due to the fact that parts of the cryptographic core dealing with the key

were not necessarily tainted by the plaintext value itself, be it because of the constant values that were used or the inherent structure of the design.

5.3.3 Composite Field Implementation of AES

Since the Composite Field design of AES included both an encryption and decryption block, there was a significant number of cells that were found to be consistently untainted. This suggests a significant number of cells were essentially producing noise during the simulations with respect to a given signal. To begin, 128 simulations were run where one bit of the key was tainted at a time. The resulting VCD files were then analyzed to find a set of cells common to all of the VCD files. Table 8 shows a breakdown of the number cell count in the design.

	Total Cells	Consistently Tainted Cells	Consistently Untainted Cells	Intermittently Tainted Cells
Composite Field	11,403	5,217	5,673	513

Table 8: Breakdown of Cell Count by GLIFT when Analyzing the Key for the Composite Field Design

This table shows that less than half (45.8%) of the entire design was consistently tainted. This is the number of cells determined to leak information regarding the key. This largely reflects the fact that the decryption block is unused during the encryption process (save for cells that may be shared between the two processed). Figure 32 shows a side-by-side comparison of an unaltered simulated power trace of this design and the simulated power trace using only the cells selected by GLIFT.

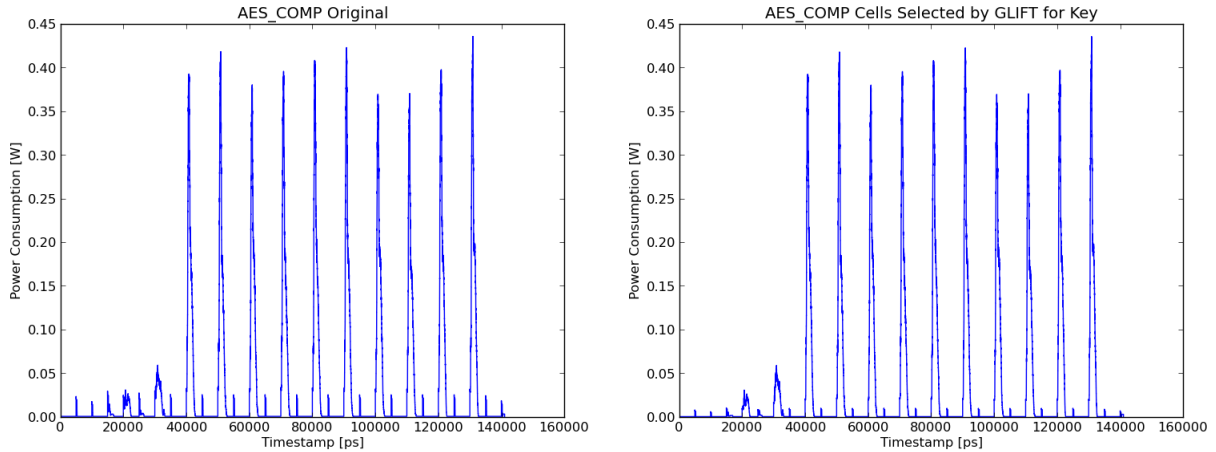


Figure 32: Unaltered Simulated Power Trace (Left) and Simulated Power Trace Using Cells Selected by GLIFT when Analyzing the Key (Right) of the Composite Field AES Design.

One noticeable aspect of the unaltered simulated power traces is the small peaks that are seen between the large peaks of each round of encryption. A list of cells selected by GLIFT was then passed to PrimeTime-PX for simulation, reducing the total amount of cells to be simulated, resulting in the simulated power trace on the right. The small peaks that were originally seen in the unaltered simulation have been reduced in the simulation done with the GLIFT-selected cells. The remainder of the plot has also changed slightly. For comparison, the cells that were excluded by GLIFT were separately simulated, resulting in the power trace shown in Figure 33.

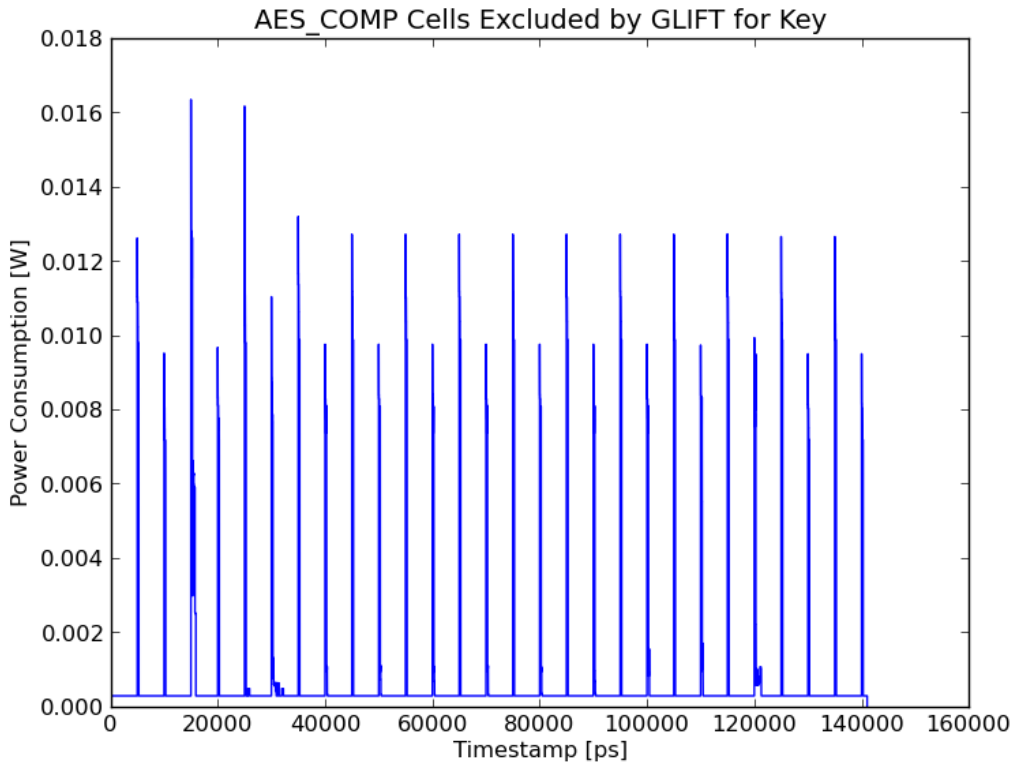


Figure 33: Simulated Power Trace Using Cells Excluded by GLIFT when Analyzing the Key of the Composite Field AES Design

Gone are the regular peaks that were seen in the original design; instead, cells primarily driven by the clock are shown. The scale of power consumption is also significantly lower than that seen in Figure 32. Given that the excluded cells are largely that of the decryption block, this plot mainly shows the impact of the decryption block during the encryption process in terms of power consumption. The scale of this second plot is about 1/10th that of the original full design presented earlier.

After deriving the list of cells dependent on the key, a list of cells dependent on the plaintext was derived. Another 128 simulations were generated where each bit of the plaintext (i.e. state) was tainted. These VCD files were then analyzed to find a set of cells common to each of the files. Table 9 shows the breakdown of the number of the original cells in the design and the number of cells selected and excluded by GLIFT.

	Total	Consistently	Consistently	Intermittently
--	--------------	---------------------	---------------------	-----------------------

	Cells	Tainted Cells	Untainted Cells	Tainted Cells
Composite Field	11,403	5,217	6,185	1

Table 9: Breakdown of Cell Count by GLIFT when Analyzing the Plaintext for the Composite Field Design

Compared to Table 8, the same number of cells were found to be consistently tainted, despite the fact that two different signals were analyzed; however, when the plaintext was analyzed, more cells were found to be consistently untainted. This stems from the fact that focusing on the plaintext itself may remove some cells that are involved in the key expansion process that were originally found when examining the key. This is consistent with the results found with the Lookup Table and PPRM3 design.

In order to further evaluate the efficacy of GLIFT, a DPA was performed on the group of cells that were selected and the group of cells excluded by GLIFT. To get a better idea of the variations seen between each simulation of the cells selected by GLIFT, Figure 34 shows an overlay of all of the simulations conducted with these cells for the Composite Field design.

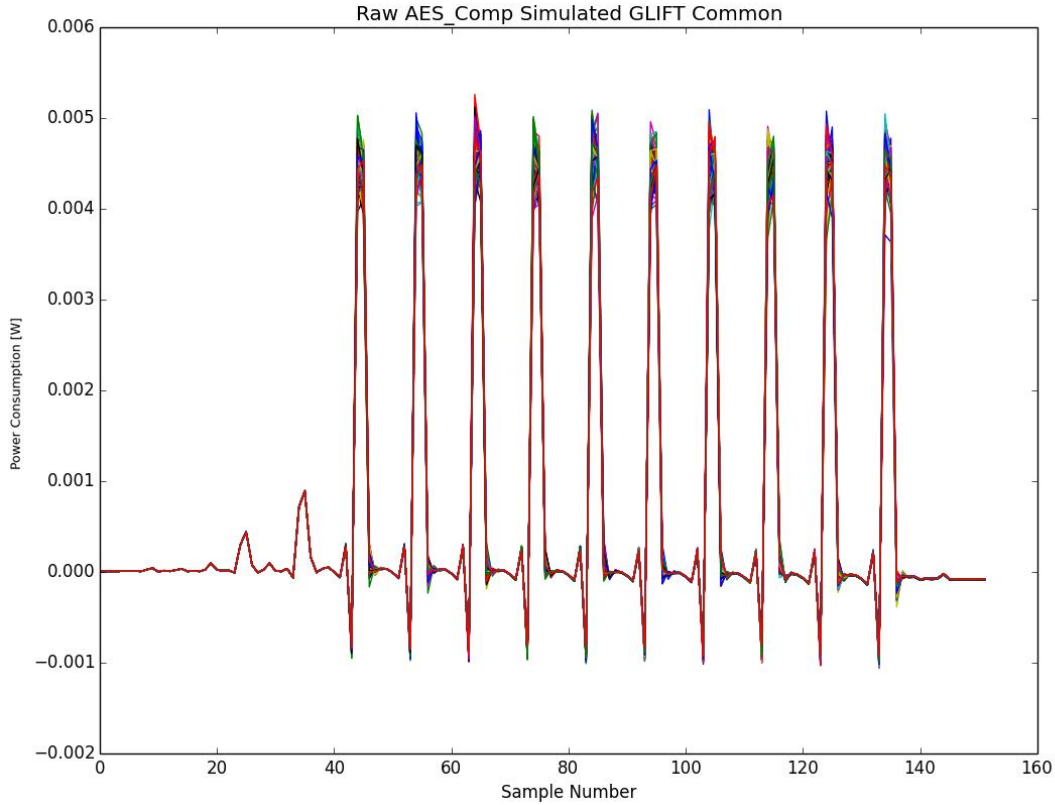


Figure 34: Overlap of All Simulations of Cells Selected by GLIFT for the Composite Field Design of AES

Thus, based on the figure, there is variation in power consumption of the design depending on the value being encrypted. DPA was then applied to the group of cells selected by GLIFT to determine how much the correlations changed compared to when DPA was conducted on the original design. The results between the two designs were too similar to discern from each other. This is a result of the fact that the “noise” produced by the decryption block was consistent and thus removed in the DPA. Full results can be seen in Section 9.6.

For comparison, a DPA was performed on the group of cells that were excluded by GLIFT. Figure 35 shows a comparison of the original design and the cells excluded by GLIFT when attacking the first byte of the key with DPA.

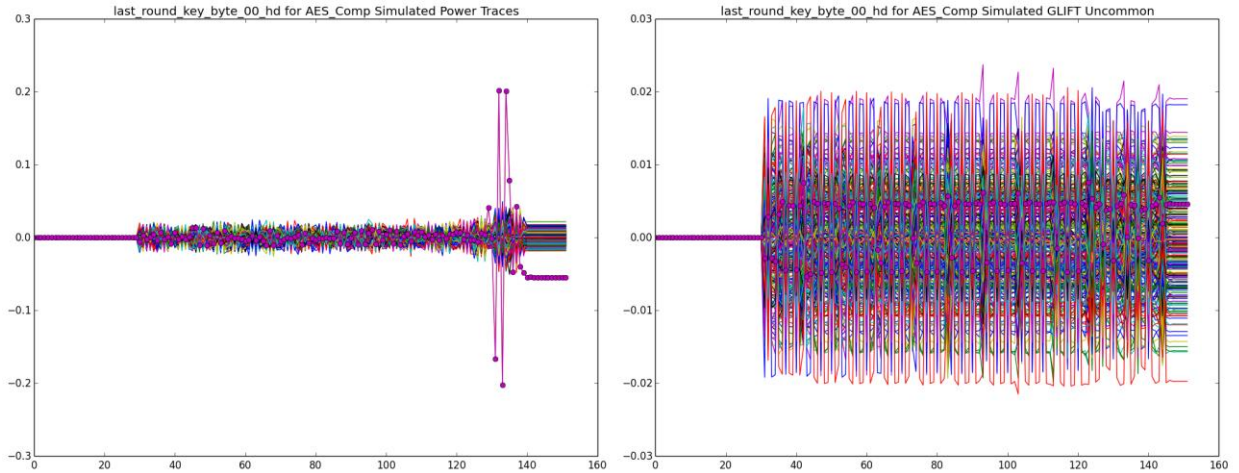


Figure 35: Comparison of the Original Design (Left) and the Cells Excluded by GLIFT (Right) for the Composite Field Design of AES

This figure shows that while a value was found for the first byte of the key for design, no value was found when analyzing the cells that were excluded by GLIFT. It is important to note that the scale of the second correlation plot for the GLIFT-excluded cells is 1/10th that of the full design. This is done to showcase the small variations in correlation, but overall there are no significant hits of the same magnitude as those found for the full design simulation.

Another metric of comparison is the signal-to-noise ratio that was described in Section 5.2. A comparison of the original design and the cells excluded by GLIFT for the first byte of the key can be seen in Figure 36.

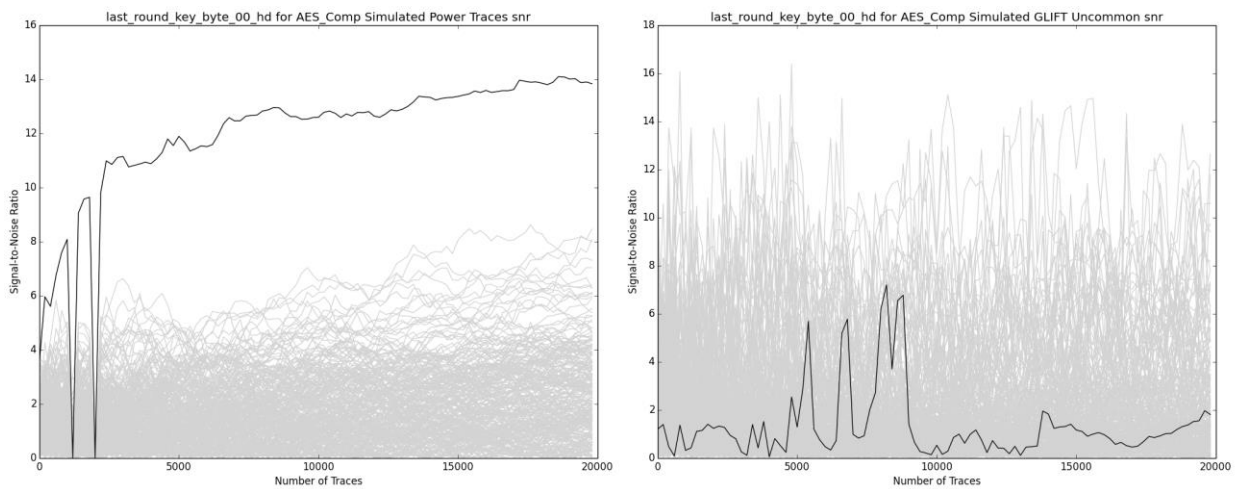


Figure 36: Signal-to-Noise Ratio for Correct Key Guess of the First Byte of the Original Design (Left) vs. the Cells Excluded by GLIFT (Right) for the Composite Field Design of AES

Comparing the two plots in this figure, there is no clear indication of a correct key guess when analyzing the cells excluded by GLIFT. Results for all of the other bytes of the key can be found in Appendix D: DPA Results: Full AES Comp vs. GLIFT Results.

As was mentioned in Section 5.2, some equality errors that arose when comparing to float-type numbers resulted in the dips seen on the plot to the left. To correct this, Gaussian noise was added to the power trace. For thoroughness, the same process was done with the cells excluded by GLIFT. The results are shown in Figure 37.

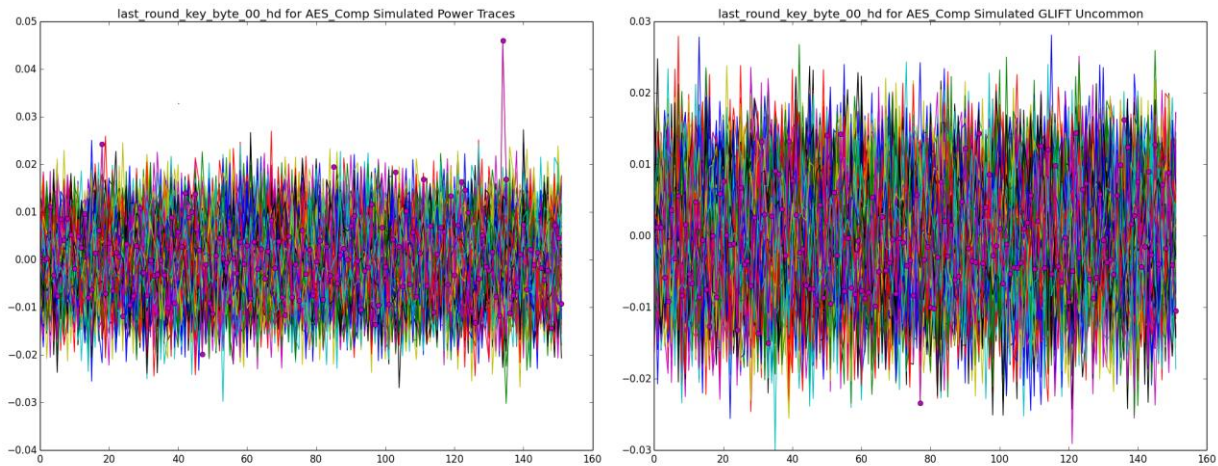


Figure 37: Comparison of the Original Design (Left) and the Cells Excluded by GLIFT (Right) with Gaussian Noise Added for the Composite Field Design of AES

In this figure, a DPA hit was still found for the original design and also better represents what was found with the measurements taken in the lab, but there was still no hit to be found with the cells excluded by GLIFT.

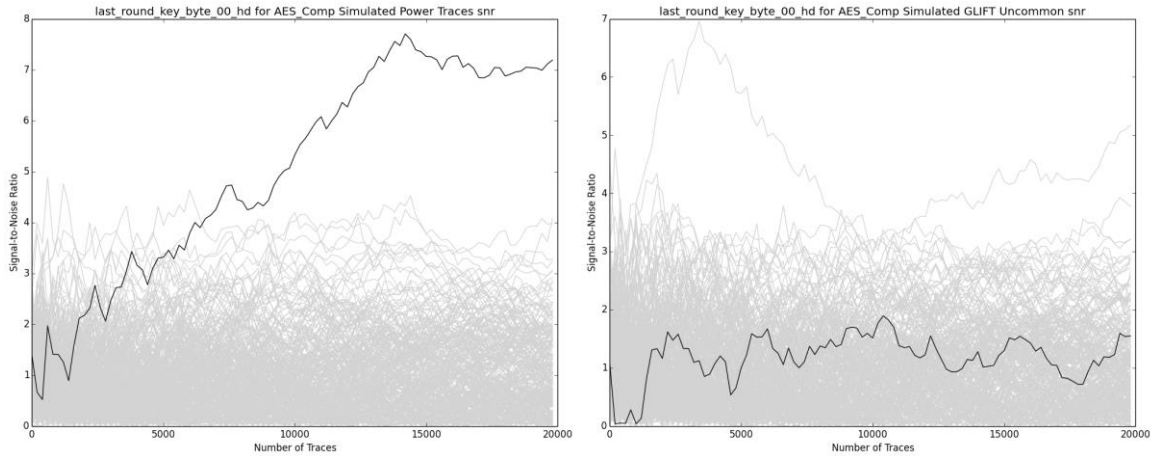


Figure 38: Signal-to-Noise Ratio for Correct Key Guess of the First Byte of the Original Design (Left) vs. the Cells Excluded by GLIFT (Right) with Gaussian Noise Added for the Composite Field Design of AES

As was previously found, introducing Gaussian noise removed the equality errors, but did not quantitatively improve the signal-to-noise ratio seen with the cells excluded by GLIFT. Results for all of the keys can be seen in Appendix F: DPA Results: AES Comp GLIFT-Selected and Excluded Cells with Noise.

5.4 Back-Annotation of VCD Files to Pinpoint Information Leakage Points

The final tool developed was a way to view the results of a power-analysis attack during simulation of a design. This is especially useful for digital hardware designers, as they can see precisely at which instant in time a correlation hit occurs from an attack. This may make it much easier for the designer to then target a specific cell or operation that is leaking information, and modify it.

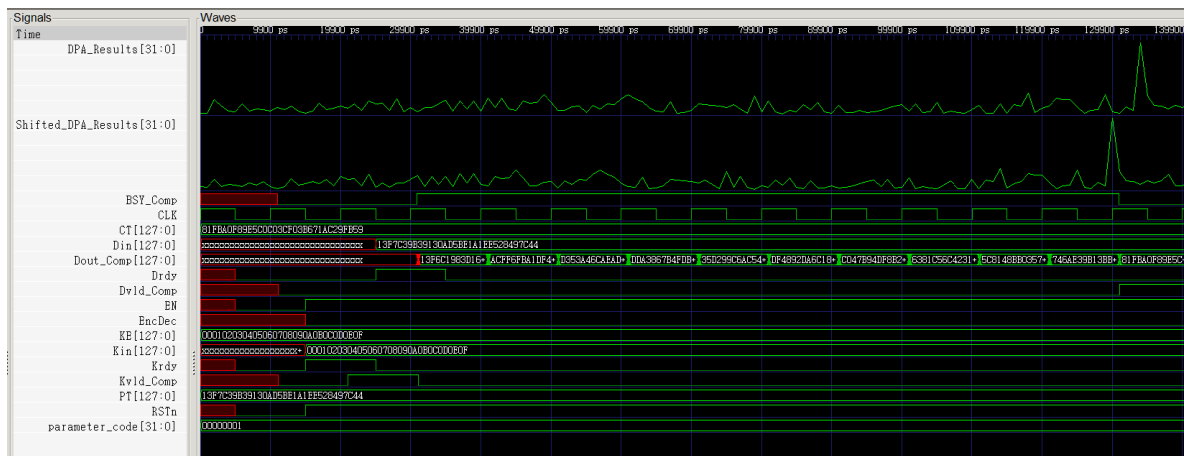


Figure 39: Timing Diagram of AES Comp Design with DPA Hits Included

6 Conclusions

After conducting thorough evaluations of PSCARE and GLIFT, many conclusions can be drawn. These conclusions are outlined in this section with respect to the efficacy of PSCARE and GLIFT.

6.1 PSCARE Simulates Power Analysis Attacks, Produce Realistic Results

As outlined in Section 5.2, PSCARE produces correlation values in simulation that are similar to those produced in reality. This is extremely useful, as a designer is now much closer to being able to protect a digital design from power-analysis attacks and information leakage while the design is still in simulation. This is hugely important, as iterating through designs is relatively inexpensive, both in terms of money and time, while still in the simulation stages.

Another important evaluation procedure that should be performed is to test PSCARE with AES designs that are specifically produced to be resistant to power analysis attacks. While this project did focus on three distinct designs, AES-Comp, AES-PPRM3, and AES-TBL, these were all various implementations the S-Box, which would not have significant effect upon the final-round register-update attack model. However, there are two designs, Wave-Dynamic Differential Logic (WDDL) and Masked Dual-Rail Pre-Charge Logic (MDPL) that are examples of power analysis-resistant implementations of AES. Evaluation of these two additional designs was attempted during this project, but was not successfully completed due to compilation errors of the Verilog source code.

6.2 GLIFT Successfully Selects Cells of Concern for PSCARE

Based on the results outlined in Section 5.3, GLIFT is able to determine which cells in a design leak information on a logic level, and in turn leak information via the power side-channel. While the Lookup Table and PPRM3 designs did not have many cells that did not contribute to the encryption process, the Composite Field's decryption block allowed a more thorough evaluation of GLIFT's ability to remove cells that were not leaking information during the encryption process. The fact that the correlations of the DPA hits between the original design simulations and the GLIFT-selected cells simulation were consistent shows that no information was lost due

to GLIFT. By performing DPA on the cells that were excluded by GLIFT, it could be determined that those cells did not leak any information and could indeed be left out during analysis conducted with PSCARE, speeding up the simulation process. Additionally, comparing the number of cells that were found to be consistently tainted and untainted when different signals were tainted (key vs. plaintext) verifies the expectation that less of the design would be found to leak information in general when the plaintext is tainted. This is because the plaintext may interact with less of the design during the AES encryption process compared to the key itself.

7 Further Development

While conclusions have been drawn in Section 6 using the results outlined in Section 5, there is still further room for development and investigation with regards to PSCARE and GLIFT. These avenues of research are outlined in this section.

7.1 Additional Verification and Improvements Upon PSCARE

One major area of improvement for PSCARE is reproducing exact conditions in both simulation and reality. Due to an oversight during simulation and very confusing documentation during actual data measurement, the sampling rate per clock period was not the same for simulation and capture setups during this project. As outlined previously, the test bench developed used a 100 MHz clock, the SASEBO documentation referenced a 24 MHz clock, and the cryptographic LSI actually used a 3 MHz clock signal. As a result the number of data points per clock period varied from simulation to reality. However, this should not have affected the main results of this project, which were still positive.

7.2 Additional Evaluation and Possibilities for GLIFT

While GLIFT has been evaluated in and determined to successfully highlight cells of concern with regards to power side-channel vulnerabilities, there are nevertheless some other evaluations that could be performed to further validate its efficacy. This subsection will outline those areas of opportunity.

1. Evaluate GLIFT using DPA-Hardened Designs and Other Cryptographic Algorithms

GLIFT has been evaluated using the Composite-Field, Lookup Table and PPRM3 designs of AES. While this has yielded results, GLIFT could still be applied to other designs such as the Wave Dynamic Differential Logic (WDDL) implementation of AES and even other encryption algorithms such as RSA.

2. Evaluate GLIFT in terms of Other Side-Channel Vulnerabilities and Attack Methods

This project has focused on power side-channel vulnerabilities and differential power analysis. This is only one side-channel and attack method used by adversaries that may try to extract sensitive information from a design. Thus, to ensure general applicability, GLIFT should be applied to designs to determine how well it finds vulnerabilities in other side-channels to other attack methods.

3. Investigate Developing GLIFT to Address Vulnerabilities

While GLIFT has been found to highlight areas of concern within a given design, the next step would be to automatically address those areas of concern from the start. This may involve replacing cells found to propagate taint with a combination of cells that prevents taint propagation in a design. This borders upon the purpose of tools like Design Compiler and Integrated Circuit Compiler, but may still serve as a starting point for ASIC designers to work from as they try to secure their designs against power side-channel attacks.

8 References

- [1] IAIK, “Introduction to Implementation Attacks.” .
- [2] D. Naccache, “Side-Channel Attacks on Cryptographic Software,” *IEEE Security Privacy Magazine*, vol. 7, no. December, pp. 65–68, 2009.
- [3] M. Banga, M. S. Hsiao, and V. Tech, “Trusted RTL : Trojan Detection Methodology in Pre-Silicon Designs *,” pp. 56–59, 2010.
- [4] E. Love, Y. Jin, S. Member, Y. Makris, and S. Member, “Proof-Carrying Hardware Intellectual Property : A Pathway to Trusted Module Acquisition,” vol. 7, no. 1, pp. 25–40, 2012.
- [5] J. Oberg, R. Kastner, and T. Sherwood, “Eliminating Timing Information Flows in a Mix-Trusted System-on-Chip,” *IEEE Design & Test*, vol. 30, no. 2, pp. 55–62, 2013.
- [6] S. Drzevitzky, U. Kastens, and M. Platzner, “Proof-Carrying Hardware: Towards Runtime Verification of Reconfigurable Modules,” in *2009 International Conference on Reconfigurable Computing and FPGAs*, 2009, pp. 189–194.
- [7] B. Wilson, “CMOS Logic,” 2008. [Online]. Available: <http://cnx.org/content/m1029/latest/>. [Accessed: 08-Sep-2012].
- [8] A. Sarwar, “Cmos power consumption and cpd calculation,” *Proceeding: Design Considerations for Logic Products*, no. June, 1997.
- [9] S. Mangard, M. E. Oswald, and T. Popp, *Power Analysis Attacks - Revealing the Secrets of Smart Cards*, vol. 54. 2007, pp. I–XXIII, 1–337.
- [10] P. Kocher, J. Jaffe, and B. Jun, “Differential Power Analysis,” *Analysis*, vol. 1666, pp. 388–397, 1999.
- [11] W. E. Burr, “Selecting the Advanced Encryption Standard,” *IEEE Security Privacy Magazine*, vol. 1, pp. 43–52, 2003.
- [12] K. H. B. K. H. Boey, P. Hodgers, Y. L. Y. Lu, M. O’Neill, and R. Woods, “Security of AES Sbox designs to power analysis,” *Electronics Circuits and Systems ICECS 2010 17th IEEE International Conference on*, pp. 1232–1235, 2010.
- [13] J. A. Ambrose, N. Aldon, A. Ignjatovic, and S. Parameswaran, “Anatomy of Differential Power Analysis for AES,” *2008 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 459–466, 2008.
- [14] E. Prouff, “DPA Attacks and S-Boxes,” pp. 424–441, 2005.

- [15] J. Moser, “A Stick Figure Guide to the Advanced Encryption Standard (AES),” *MOSEWARE*, 2009. [Online]. Available: <http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html>. [Accessed: 22-Sep-2013].
- [16] N. I. of S. and T. (NIST), “Announcing the Advanced Encryption Standard (AES),” 2001.
- [17] N. Fips, “Announcing the ADVANCED ENCRYPTION STANDARD (AES),” *Byte*, vol. 2009, pp. 8–12, 2001.
- [18] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, “A Compact Rijndael Hardware Architecture with S-Box Optimization,” pp. 239–254, 2001.
- [19] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi, “Efficient Rijndael Encryption Implementation with Composite Field Arithmetic,” no. April, pp. 171–184, 2001.
- [20] S. Morioka and A. Satoh, “An Optimized S-Box Circuit Architecture for Low,” pp. 172–186, 2003.
- [21] U. S-boxes and C. L. C. Tn, “Ultra-Low Power S-Boxes Architecture for AES,” vol. 15, no. 1, 2008.
- [22] J. Allen, *Logic Synthesis and Optimization*. New York, NY: Springer Science+Business Media, 1993, p. 381.
- [23] “AIST RCIS: SASEBO-R.” [Online]. Available: <http://www.rcis.aist.go.jp/special/SASEBO/SASEBO-R-en.html>.
- [24] SASEBO, “Side-channel Attack Standard Evaluation Board SASEBO-R Specification.”
- [25] SASEBO, “ISO / IEC 18033-3 Standard Cryptographic LSI with Side Channel Attack Countermeasures Specification,” 2009.
- [26] “Cryptographic Hardware Project.” [Online]. Available: <http://www.aoki.ecei.tohoku.ac.jp/crypto/web/cores.html>.
- [27] M. Tiwari, H. M. G. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood, “Complete Information Flow Tracking from the Gates Up,” *ACM SIGPLAN Notices*, vol. 44, no. 3, p. 109, Feb. 2009.

9 Appendices

9.1 Appendix A: PSCARE Readme and Documentation

The following is the readme document that was given to The MITRE Corporation upon completion of this project detailing the use of Power Side-Channel Risk Evaluator. PSCARE was developed by the authors of this paper to simulate a design's power side-channel leakage and to determine whether or not sensitive information would be leaked through a device's power consumption's characteristics. Additional functionality was developed to allow a designer to view the results of a Differential Power Analysis attack directly adjacent to that design's waveforms. This is accomplished using a waveform viewer and an annotated value-change dump file that is produced after DPA. This functionality allows a designer to see at exactly which points in time a device is leaking sensitive information.

Power Side-Channel Attack Risk Evaluator Simulation Package

README

Developed by:

J. Victor Haley (jhaley@mitre.org)

Daniel Hulihan (dhulihan@mitre.org)

Copyright: The MITRE Corporation, 2013

INTRODUCTION

Power Side-Channel Attack Risk Evaluator (PSCARE) is a tool developed to produce simulated power traces for an RTL design. After the design has been synthesized, this tool will simulate the power consumption for varying input conditions.

Specifically, PSCARE is used to simulate encryption core designs by varying the values of plaintext encrypted while using a user-specified secret key.

To run the PSCARE flow, a VCS Runtime Net license is required, as well as PrimeTime and Primetime-PX licenses. These are the main Synopsys tools used for simulation

PSCARE was built and tested with Python 2.7.5 and requires the following modules:

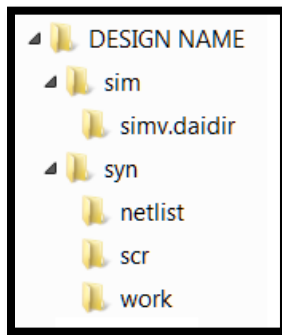
- Crypto
- HSPY

MITRE Proprietary

- Matplotlib
- NumPy
- PyCrypto
- SciPy

To run the simulation, the proper SIMV simulator must be compiled to expect input plaintext, key and ciphertext values. A standard delay file, netlist, and PT-PX script are also required. More details on the specific implementation of these simulation elements can be found in Section X.

To invoke the PSCARE flow, the correct hierarchy must first be constructed containing all the necessary simulation files. This hierarchy is as follows:



Top-Level Simulation Directory:

sim: Directory containing simv

simv.daidir: Directory containing libraries used by simv

syn:

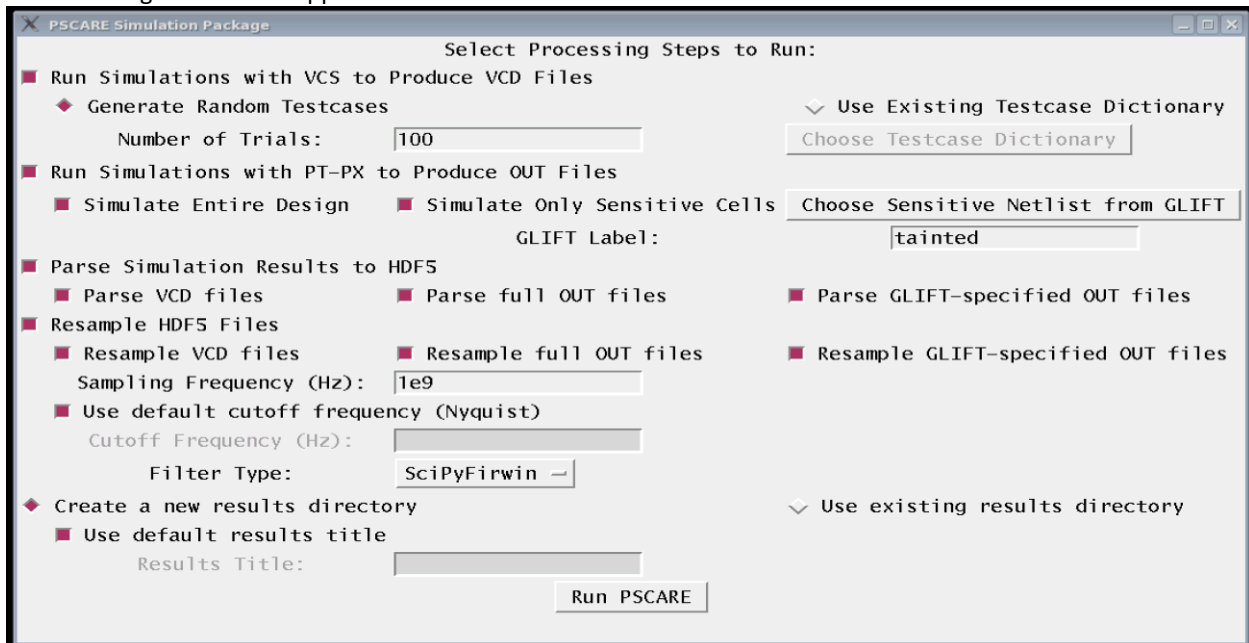
netlist: Directory containing the netlist for the design

scr: Directory containing the script to run PT-PX

work: Directory containing the parasitics and delay files

When all the correct files are in place, PSCARE is invoked by calling `pscare_gui.py`. This program makes use of an interactive GUI, and therefore the `DISPLAY` environmental variable must be properly set before using PSCARE.

The following window will appear:



PSCARE consists of four distinct stages:

- Simulating with VCS to produce value-change dump (VCD) files containing gate-level toggle information
- Simulating with PT-PX to produce OUT files containing power values

MITRE Proprietary

- Parsing these VCD and OUT files to produce numerical arrays representing the bit toggles and power consumption, which are then stored in HDF5 file format.
- Resampling the time-stamped numerical arrays to regularly-sampled arrays in HDF5 format.

While these steps are sequential and do depend upon files produced by the previous tasks, the GUI allows the user to specify any or all steps that they wish to run. Upon running any step, PSCARE looks for existing files needed to process the current task.

For example, if a user generated 20,000 VCD files and parsed them to HDF5 files, but then came back at a later date and wanted to generate 20,000 corresponding OUT files, all the user must do is to specify the pre-existing results directory. PSCARE will then source those existing VCD files and correctly produce the OUT files.


The results of a PSCARE execution are stored in a results directory, which can either be automatically created or specified, in the case of sourcing a previous execution's results for additional processing. In the case of creating a new results directory, by default the directory will be named 'run-YYYY-MM-DD-hhmmss' with the current date and time being inserted into the name. The user can also specify a results title to use in place of this automatically-created name. This new directory will be created in the root simulation directory, as specified later in the execution.

A high-level overview of a general PSCARE execution run is as follows:

- Choose which processing steps to run (Run VCS Simulations, Run PT-PX Simulations, Parse Results, Resample HDF5)
- Specify if a new results directory is to be created, or an existing one to be sourced
- Press 'Run PSCARE'
- Choose the root simulation directory (DESIGN NAME in Figure 1)
- If applicable, choose the existing results directory with the directory browser dialog
- If applicable, choose the existing GLIFT-specified out files directory for parsing
- If applicable, choose the existing GLIFT-specified HDF5 arrays for resampling

PROCESSING STEP DESCRIPTIONS:

Run Simulations with VCS to Produce VCD Files:



The screenshot shows a GUI window titled "Run Simulations with VCS to Produce VCD Files". It contains two main options: "Generate Random Testcases" and "Use Existing Testcase Dictionary". Under "Generate Random Testcases", there is a text input field for "Number of Trials:" with the value "100". Under "Use Existing Testcase Dictionary", there is a button labeled "Choose Testcase Dictionary".

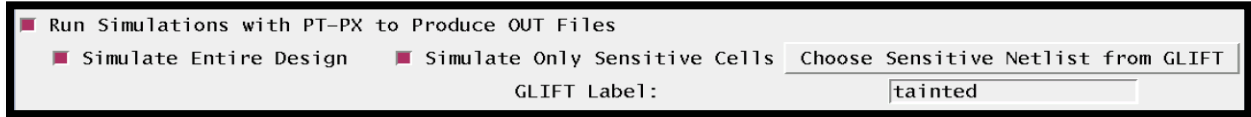
When this processing step is selected, the user has two options for choosing which test-cases are used to run the simulations. A test-case contains a plaintext (128-bit), a key (128-bit) and a corresponding ciphertext (128-bit). For this implementation, these three values are stored as 32-character strings consisting of hex characters 0-f. The first option is to generate a user-specified number of random plaintext values, while using the same default key: ('000102030405060708090a0b0c0d0e0f'). The corresponding ciphertext outputs from running AES-128 with this key and the plaintext values are then calculated and stored. The entire set of test-cases is then stored as a pickled dictionary (Python storage scheme) file. The directory is automatically created and is located in the results directory under a new directory called 'testcases'.

The other option for test-cases is to source an existing test-case dictionary. To do this, the user simply checks 'Use Existing Testcase Dictionary' and then clicks the 'Choose Testcase Dictionary' button. This will bring up a file browser dialog, in which the user should select the pickled (.p) testcase dictionary already produced by a previous execution of PSCARE.

MITRE Proprietary

After the test-cases are either generated or sourced, VCS-Runtime Net is then called to run the test-cases with the SIMV and produce VCD files. These VCD files are stored in the results directory in a new directory called 'vcd_files'.

Run Simulations with PT-PX to Produce OUT Files:



This processing step causes PSCARE to invoke PrimeTime-PX to simulate the design using the previously generated VCD files to produce OUT files that contain time-stamped power consumption values for the design. There are two options that can be selected at this stage: simulate the entire design or only simulate specific cells. If the latter option is chosen, the user must provide a selective netlist generated from GLIFT. This specifies only the cells of interest that are to be simulated by PrimeTime-PX. These OUT files will be written to a new directory within the results directory called 'out_files_GLIFT_LABEL', while the full simulation results will be stored in a directory called 'out_files'.

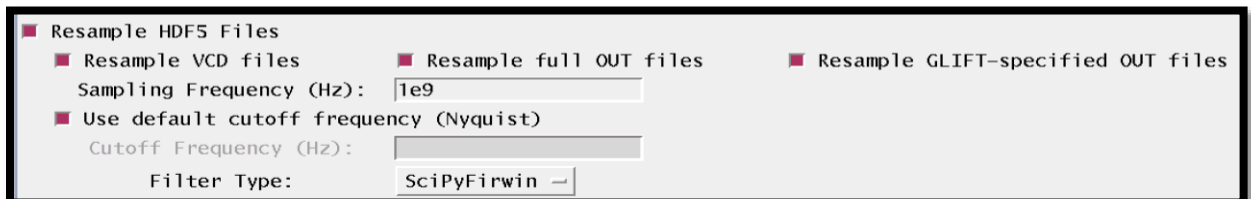
Parse Simulation Results to HDF5

This step parses the simulation output files (VCD and OUT) to time-stamped, numerical arrays stored in HDF5 format.



When VCD files are parsed, the output array contains the number of bits that changed state, or toggled at each specific instant in time of the simulation. These bit-toggle arrays can be represented as a series of impulses. For the OUT files, the output array contains the instantaneous power consumption values for each instant in time in the simulation. These power consumption arrays can be looked at as a step function, as a new value indicates that the power consumption has changed.

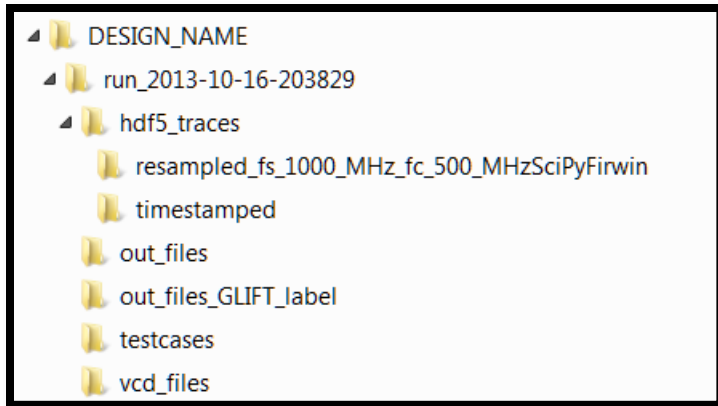
Resample HDF5 Files



This last step is important, as the output arrays from parsing are not uniformly sampled. The timestamps of each point can be any value, as they are only reported when values change in the design. The resampling step uses a filter of specified characteristics to go through at a regular sampling interval and create a new output array. However, the scaling factor of the particular destroys the specific units of each array. This means that if the input array is reported in mW, the output array will still have a base unit of Watts, but the prefix is unknown at this time. For the bit toggles, the input array is reported in bit toggles / timestamp, and the output array will be a scaled version of bit toggles.

The final output directory structure is presented in the figure below:

MITRE Proprietary



The DESIGN_NAME directory should already be in place, and all the other directories will be auto-created based upon which steps the user selects.

9.2 Appendix B: GLIFT Readme and Documentation

GLIFT: Gate-Level Information-Flow Tracking

When developing ASIC designs that deal with sensitive information (such as an encryption process), it is necessary to be able to pinpoint areas that may leak information. A variety of methods exist, including simulating attacks on the design to determine any vulnerabilities. Another method is tracking the flow of information through the design. Information flow tracking (IFT) is a technique that has been applied to several layers of electronic systems that allows users and developers to keep track of how information propagates through a given design. With this information, it is possible to develop a policy that restricts the flow of information and mitigate the amount of information that may be leaked to attackers.

GLIFT takes information-flow tracking down the gate level of an ASIC design. Specifically, a netlist that may be generated by Synopsys Design Compiler (DC) or Integrated-Circuit Compiler (ICC) can be used by GLIFT. Designers can specify any part or all of a signal as “tainted,” meaning that it is possible that the signal has been altered by an attacker. Designers can then see which logic cells in the design consequently become tainted in a simulation. If the value of the signal can be altered by an attacker, then the outputs of a given collection of cells can be altered as well. Some cells will be consistently altered, regardless of a given combination of values and taint statuses for a given signal, reflecting a symbolic dependency between them and the tainted signal.

A collection of Python scripts were written to allow ASIC developers to implement GLIFT in their design. The following modules are necessary for GLIFT:

- H5Py
- Sympy

This is in addition to the modules that are expected to be available in a Python environment (i.e. NumPy, re, glob, and os).

The main script to be called is *glift_top.py*. This script incorporates the main functions developed in all of the other scripts, providing a clean, simple module for ASIC designers to use. Each module and function has been thoroughly documented with in-line comments and doc-strings for

future-reference should designers want to dive deeper and repurpose various functions or modules.

The functions implemented in *glift_top* are as follows:

load_stand_cell_libs

The purpose of this function is to load up all the standard cell libraries that will be used in the design. The main argument, *lib_names*, is a list of file paths to the standard cell libraries. This information is then passed to the *library_manager* module which loads all of the files to be used and passes them to the *parse_lib* module. Each liberty file is walked through, looking for the name of the library, each cell and each pin of each cell. Boolean expressions describing the behavior of the output pin of each cell are logged. If a given cell is determined to be non-combinational (i.e. a register, buffer, etc.), then the input and output pins of concern are logged. This means that things like enable (EN) pins and output pins that are inverses of another output pin (QN vs. Q) are ignored. The reasoning for this will become more apparent during the annotation process.

load_glift_lib

This function allows users to load a series of files that describe the behavior of taint-tracking cells that are added to a given netlist during the annotation process. The user can provide a list of file names or directories, and the function will load each Verilog file. These Verilog files are also generated by GLIFT for future reference.

verify_libs

Once both the standard cell libraries and the GLIFT library are loaded, the user may want to verify that each standard cell available has a corresponding taint-tracking cell described in the GLIFT library. This function takes in both dictionaries and compares them for any inconsistencies. Any standard library cells found to not have a corresponding tracking cell are returned, along with useful information that can be used to generate the tracking cell (i.e. the truth table signature and boolean expression).

create_glift_lib

While the user should ideally never start without a GLIFT library, it may be necessary to create one from scratch. This function was made to be a last-ditch effort if the GLIFT library cannot be found, recovered or otherwise use. The reason for this is the time it will take to generate a new GLIFT library.

When creating a GLIFT library, the script will compare the loaded standard cell library to an empty GLIFT library, which will result in a dictionary of cells that need corresponding taint-tracking logic and the information necessary to create the logic and behavioral files. The way these taint-tracking files are generated is as follows:

1. Take the boolean equation describing the behavior of a given standard cell and convert it to a Sympy-compatible format. This involves parsing the equation to find the symbols being used and converting non-standard operators to standard operators.
2. With a usable boolean equation, a truth table of testcases is created, including the possible taint status for each symbol. For example, a 2-input AND gate would have the following truth table made:

Case #	A	B	A_taint	B_taint	Y	Y_taint
0	0	0	0	0	0	0
1	0	0	0	1	0	0
2	0	0	1	0	0	0
3	0	0	1	1	0	1
4	0	1	0	0	0	0
5	0	1	0	1	0	0
6	0	1	1	0	0	1
7	0	1	1	1	0	1
8	1	0	0	0	0	0
9	1	0	0	1	0	1
10	1	0	1	0	0	0
11	1	0	1	1	0	1
12	1	1	0	0	1	0

13	1	1	0	1	1	1
14	1	1	1	0	1	1
15	1	1	1	1	1	1

Table 10: A Truth Table for 2-Input AND Gate with Taint Tracking

3. The original boolean equation is then evaluated at each possible testcase. At this point, the taint bit combinations are analyzed, resulting in the values seen in the “Y_taint” column. If a taint status is 1, then the corresponding bit is toggled and the output of the corresponding testcase is compared to the output of the original testcase. If they are the same, then that tainted bit did not impact the output of the logic cell, and therefore the output will not be tainted. If the outputs are different, then the tainted bit can have an impact, and the output is thus tainted.

For example, take case number 6. A is equal to 0 while B is equal to 1, but A is tainted. That means the attacker can manipulate/toggle A.. If it is turned into 1, then the output would change from 0 to 1. Thus, the output must be assumed to be tainted.

4. With a list of testcases where taint will propagate for a cell, another list is generated with don't care (X) conditions. This is due to the fact that, at the beginning of some simulations, registers may not have defined input, resulting in an undetermined output (denoted as 'X'). That being said, some cells may leak information regardless of the value of taint status of one of the signals. In order to ensure this actually happens, these “don't care” conditions have to be derived. This is done by looking at each bit of each testcase and toggling all of the other bits of the testcase to see if there is ever a change in the taint status output. If it is found to not change, then that bit can be marked as a “don't care.” This is a time-consuming process given the fact that every case and sub-case must be examined.

5. Once all of the possible testcases have been found that result in taint propagation, this behavior must be written out. The Python script *glift2verilog* takes the list of testcases and generates a Verilog file that has a case statement to determine when the taint-tracking cell will indicate taint propagation. No physical information is included in the cell since this is only intended for simulation purposes.

With everything written out to a collection of Verilog files, it is only a matter of referencing the files when the SIMV file is generated by VCS.

annotate_netlist

In order for taint to propagate in a design, the netlist must be annotated. This netlist is often generated by tools like Design Compiler and Integrated Circuit Design. The main requirement is that it must contain a list of gate-level instigations from the standard cell libraries being used. These standard cell libraries must be the same one that was read in using the *load_standard_cell_libs* function. If the libraries were read in, then the necessary information will have been returned that can then be passed to *annotate_netlist*. The user must also provide the GLIFT library that was generated from the *load_glift_lib* function. All of this information will allow the script to go line-by-line, identify a cell, find the corresponding tracking logic in the GLIFT library and add it appropriately to the netlist. This net cell takes in the taint-tracking signals of the corresponding data cells used in the original cell and outputs a signal indicating the taint status of the output signal of the original cell.

Non-combinational cells are treated a little differently: depending on information found from when the Liberty file was parsed, only certain signals (if any) will actually have a corresponding taint signal. For example, registers will have a clock signal, but the taint-tracking register should not have a taint-tracking signal of the clock itself – the register would only store information when the clock was found to be tainted instead of the actual data that is being passed to the register. Stuff like this is accounted for when tacking on non-combinational logic to track taint propagation for non-combinational cells.

The result is a new netlist that contains all of the annotations such that nearly every cell has a corresponding taint-tracking cell. It is possible that some cells simply do not deal with data that may propagate taint (for example, clock buffers), so it is possible for a design to not be completely annotated.

gen_vcd_files

With an annotated netlist it is now possible to run the design through some simulations and track the taint propagation through the design. These simulations can be generated automatically using the *gen_vcd_files* function. This function allows users to specify a list of signals of concern, their bit-width and specify the combination of values to use for each signal across all simulations. Users specify to have one bit of the signal set to 1 for each simulation, have each byte of a signal increment together, or have the signal fixed at a certain value across all simulations. This script will then create all the simulations necessary to satisfy this requirement using the annotated netlist and the collection of Verilog files that describe the behavior of GLIFT's taint-tracking cells. This library is necessary since the simulation program (e.g. VCS) needs to know how these new cells behave.

For example, a designer of a cryptographic core can set the plaintext to a specific value (i.e. the hexadecimal value 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F), the key to the same value and set a given bit of the taint status of the key to 1 for each simulation (e.g. in binary it would be 1000, 0100, 0010 and 0001). If the key is 128 bits long, then 128 simulations will be generated. Another example would be the user specifies that the plaintext does a byte-wise increment for each simulation, resulting in 255 values of the plaintext (e.g. 00 00, 01 01, 02 02, etc.) and also have each bit of the taint status of the key set to 1 for each simulation (like in the previous example). If the key is 128 bits long, then there would be $128 \times 255 = 32,640$ simulations generated since every possible combination of the two signals would be made.

derive_common_nets

With a collection of VCD files generated that presumably cycle through various possible combinations of signal values and taint statuses, it becomes necessary to look at each of

these files and determine which cells in the design were found to constantly propagate tainted information. This will result in the list of cells that designers should be concerned about.

The function essentially parse search VCD file and generate a set of cells of that had the corresponding taint-tracking cell indicate that taint was propagating from the original cell. Each of these sets are compared to find the “intersection” or collection of cells common to all sets. Users may also get a list of cells that were found to be consistently untainted across all the simulations.

This list of cells is written out to HDF5 and can consequently be uploaded to PSCARE for further analysis.

Further Development

While GLIFT has been evaluated and determined to successfully highlight cells of concern with regards to power side-channel vulnerabilities, there are nevertheless some other evaluations that could be performed to further validate its efficacy. This subsection will outline those areas of opportunity.

1. Evaluate GLIFT using DPA-Hardened Designs and Other Cryptographic Algorithms

GLIFT has been evaluated using the Composite-Field, Lookup Table and PPRM3 designs of AES. While this has yielded conclusive results, GLIFT could still be applied to other designs such as the Wave Dynamic Differential Logic (WDDL) implementation of AES and even other encryption algorithms such as RSA.

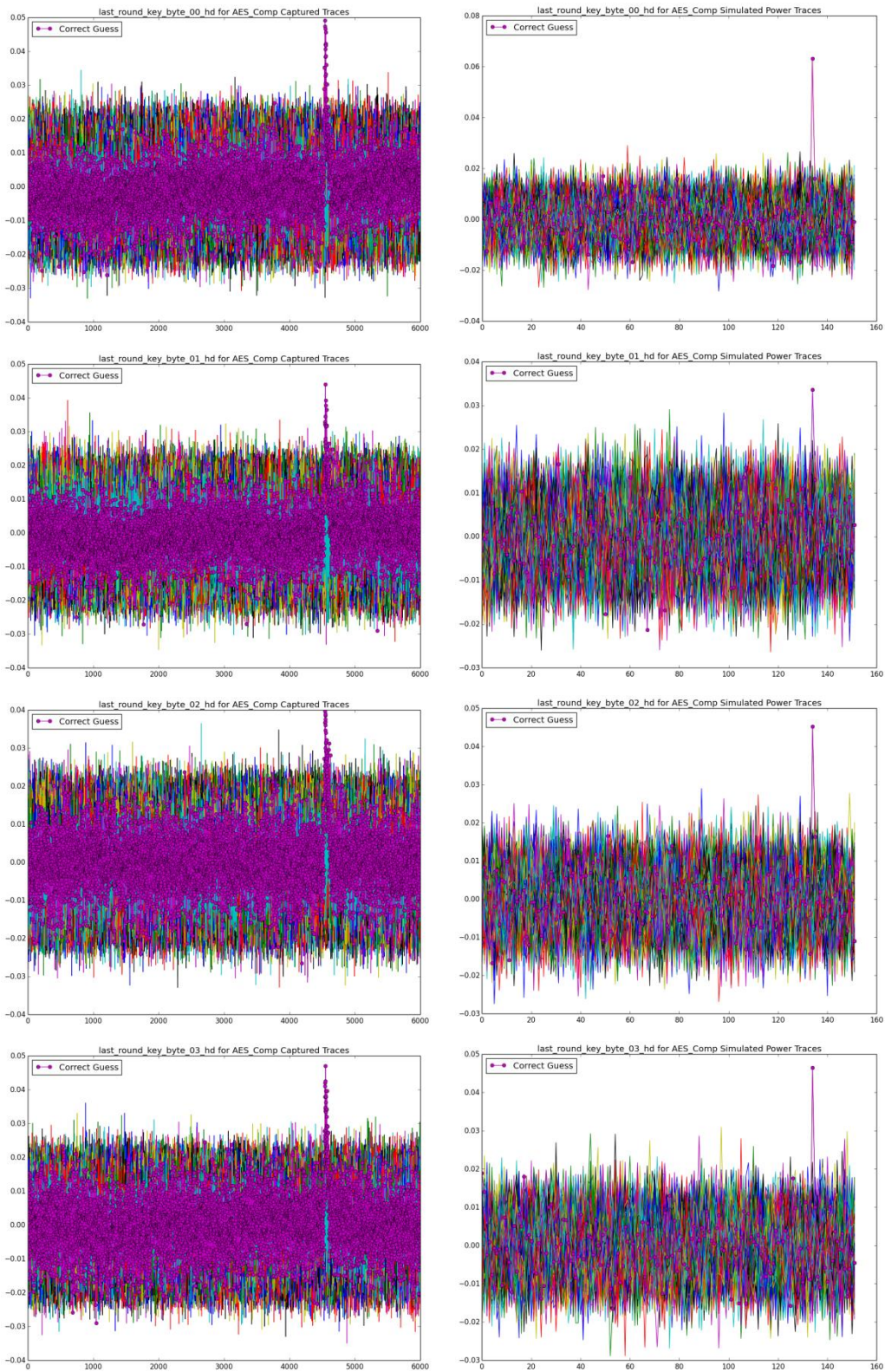
2. Evaluate GLIFT in terms of Other Side-Channel Vulnerabilities and Attack Methods

This project has focused on power side-channel vulnerabilities and differential power analysis. This is only one side-channel and attack method used by adversaries that may try to extract sensitive information from a design. Thus, to ensure general applicability, GLIFT should be applied to designs to find vulnerabilities in other side-channels and attack methods so as to provide a thorough evaluation.

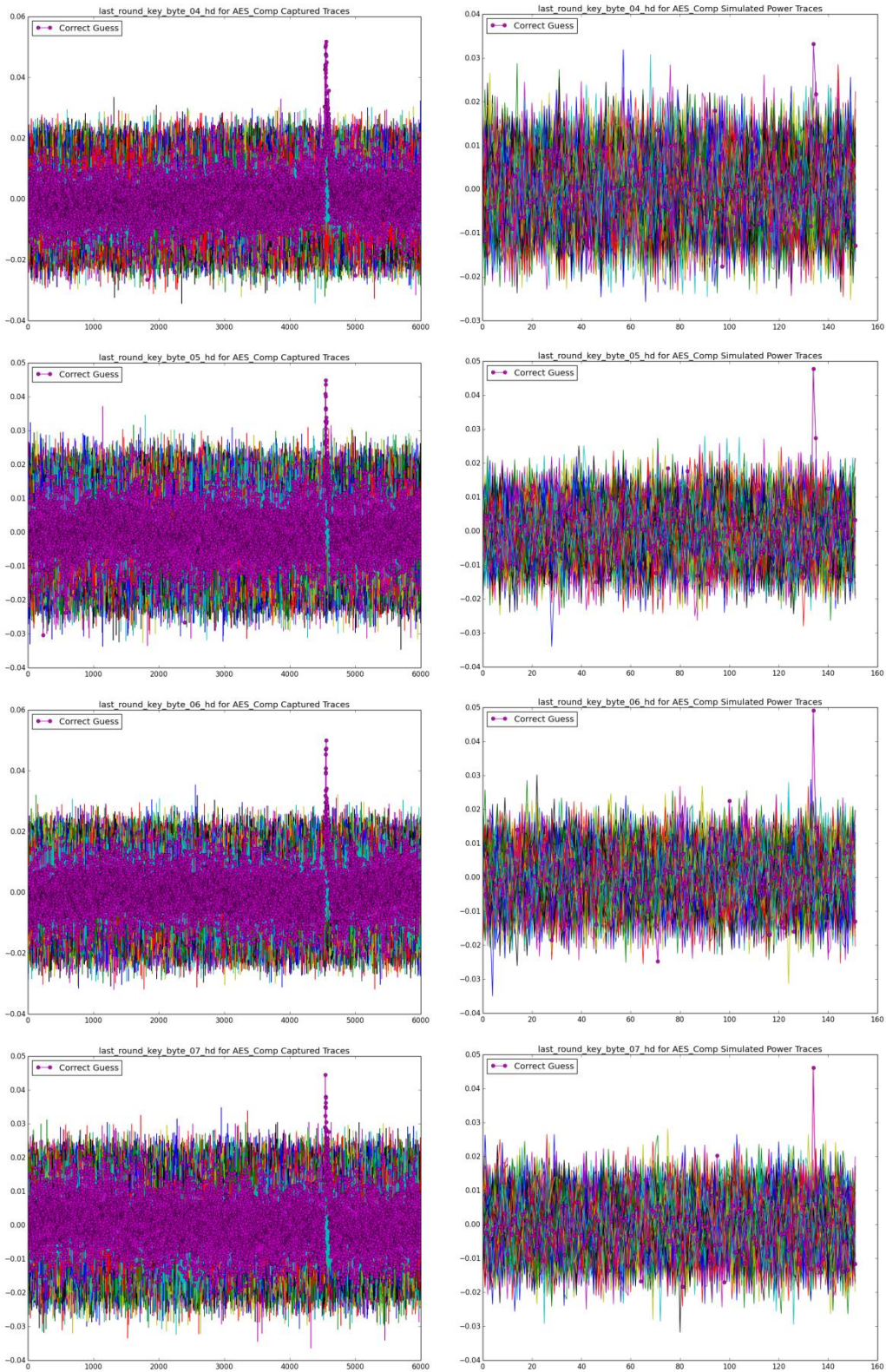
3. Investigate Developing GLIFT to Address Vulnerabilities

While GLIFT has been found to highlight areas of concern within a given design, the next step would be to automatically address those areas of concern from the start. This may involve replacing the cells found to leak tainted information with a combination of cells involve replacing cells found to propagate taint with a combination of cells that prevents taint propagation in a design. This encroaches upon the realm of tools like Design Compiler and Integrated Circuit Compiler, but may still serve as a starting point for ASIC designers to work from as they try to secure their designs against power side-channel attacks.

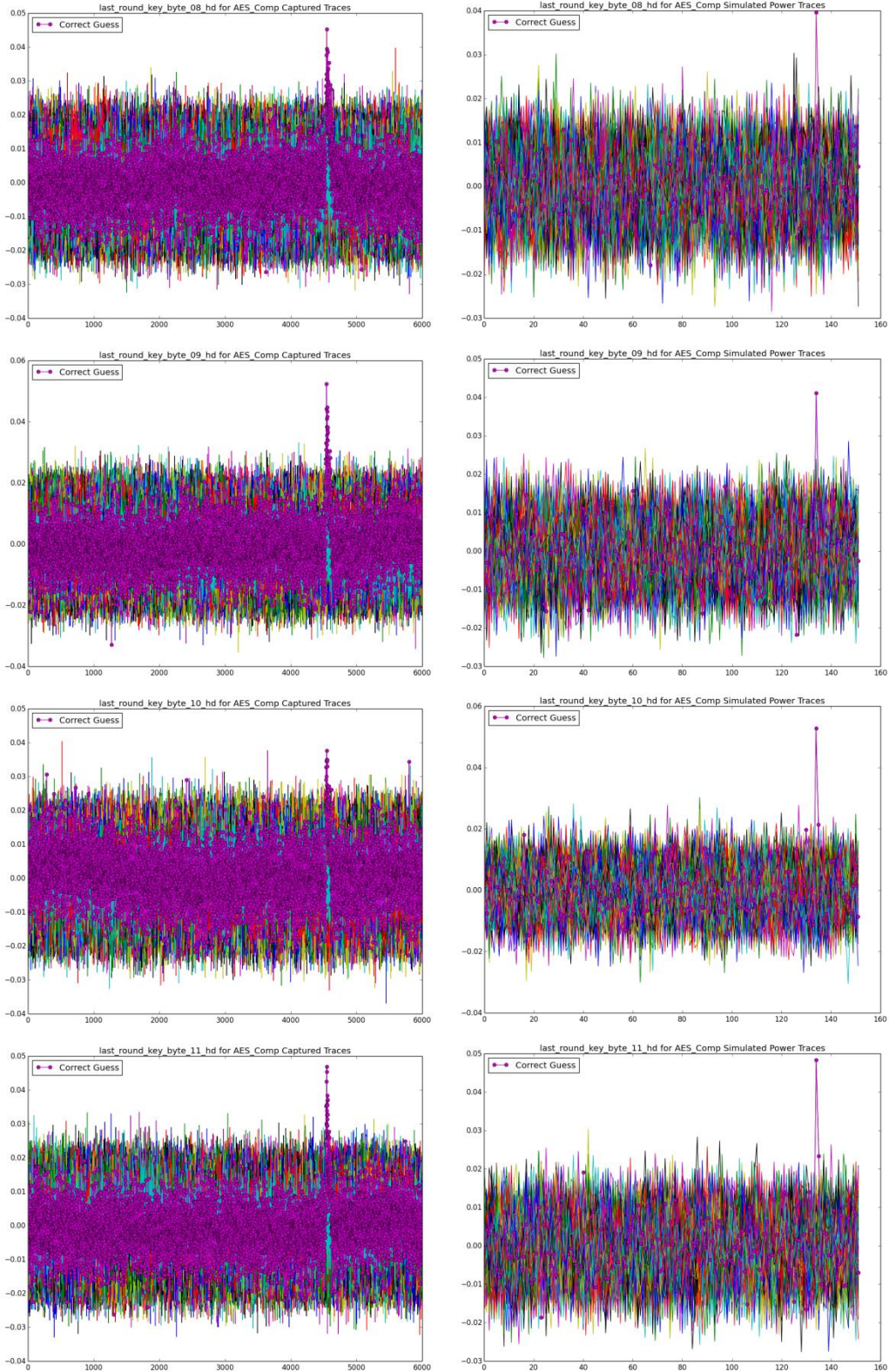
9.3 Appendix C: Full DPA Results for AES Comp



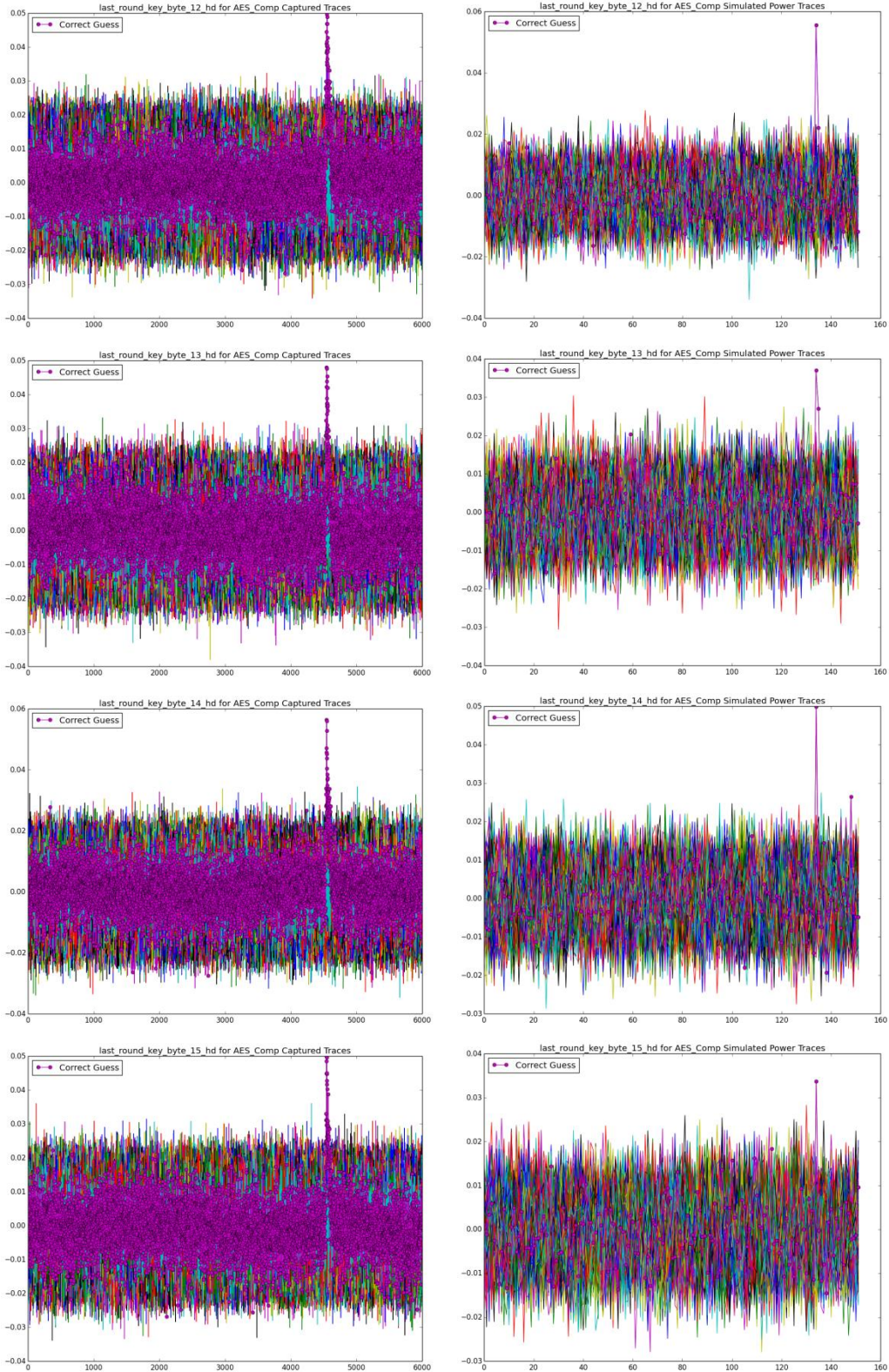
DPA Results for AES Comp Measured (R) and Simulated (L), key bytes 0-3 with 20k traces



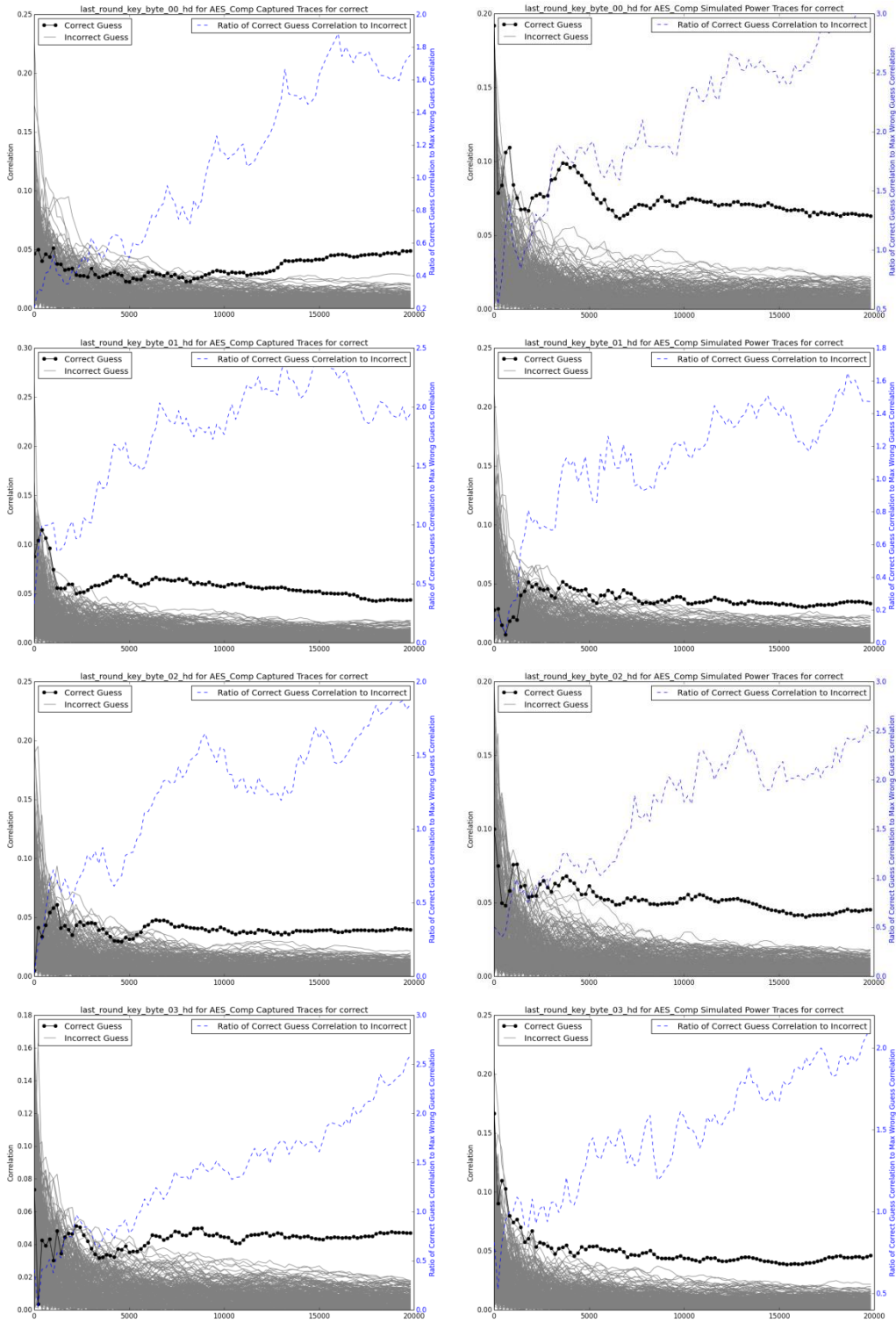
DPA Results for AES Comp Measured (R) and Simulated (L), key bytes 4-7 with 20k traces



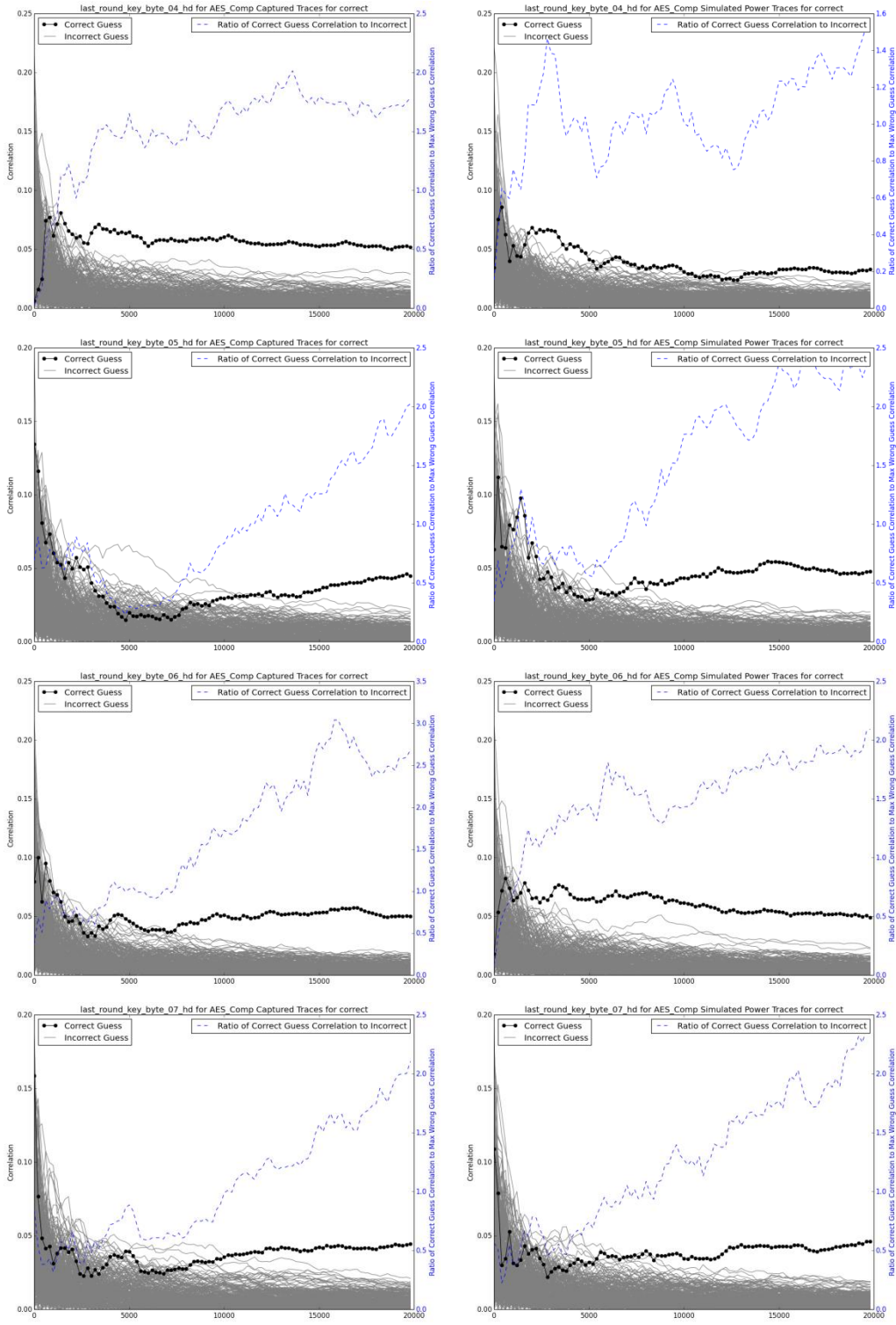
DPA Results for AES Comp Measured (R) and Simulated (L), key bytes 8-11 with 20k traces



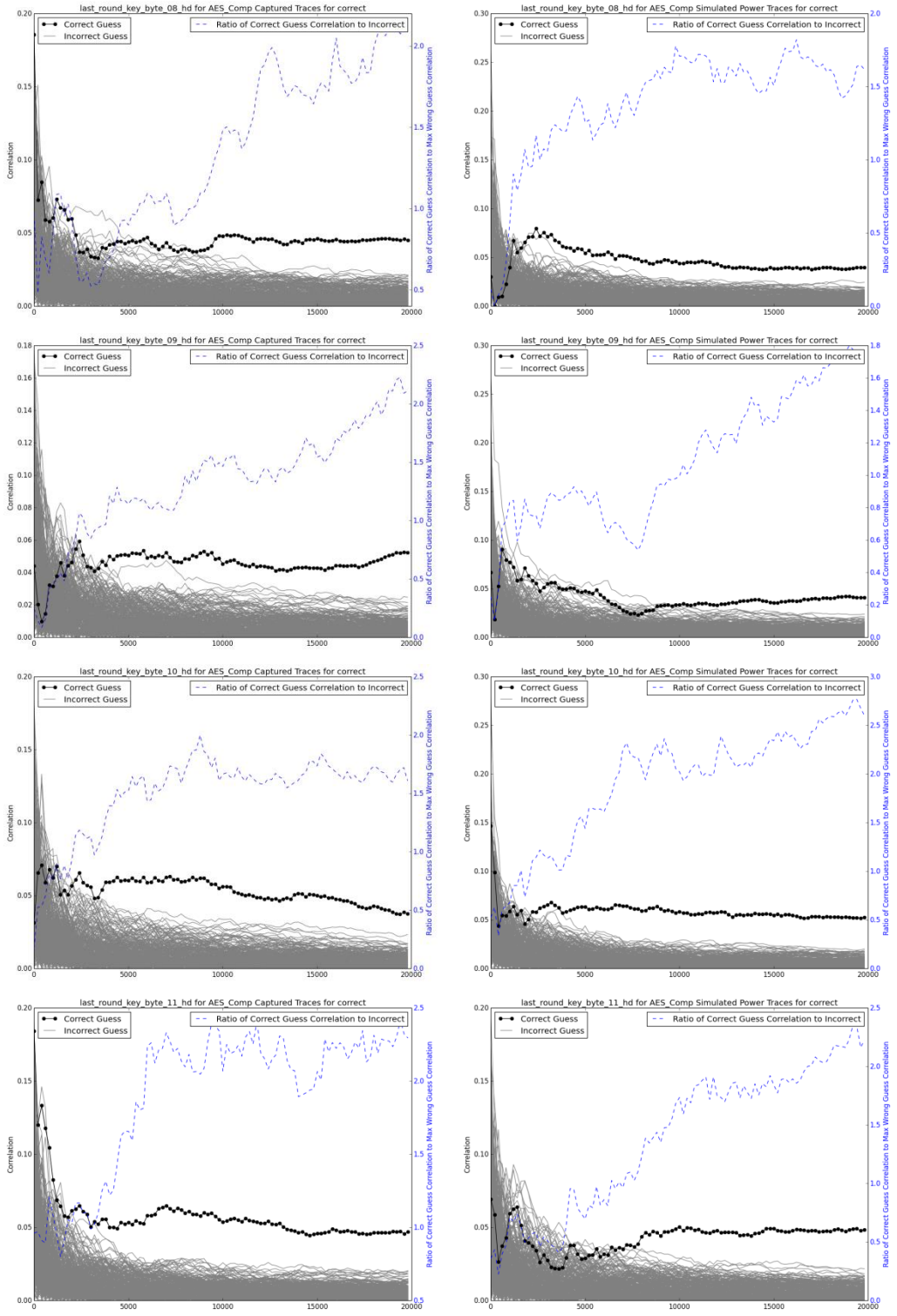
DPA Results for AES Comp Measured (R) and Simulated (L), key bytes 12-15 with 20k traces



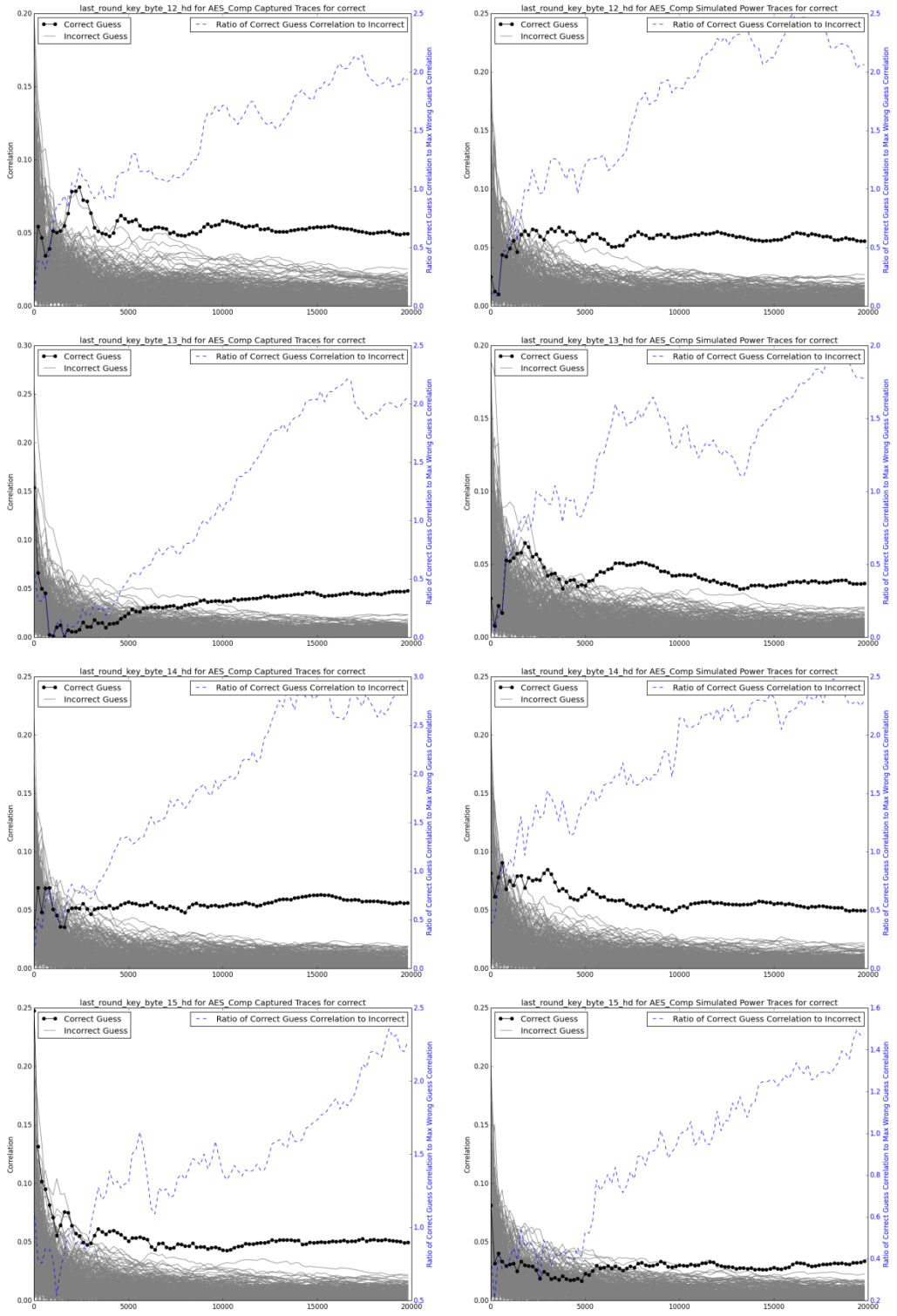
Correlation Ratios for AES Comp Measured (R) and Simulated (L), key bytes 0-3 with 20k traces



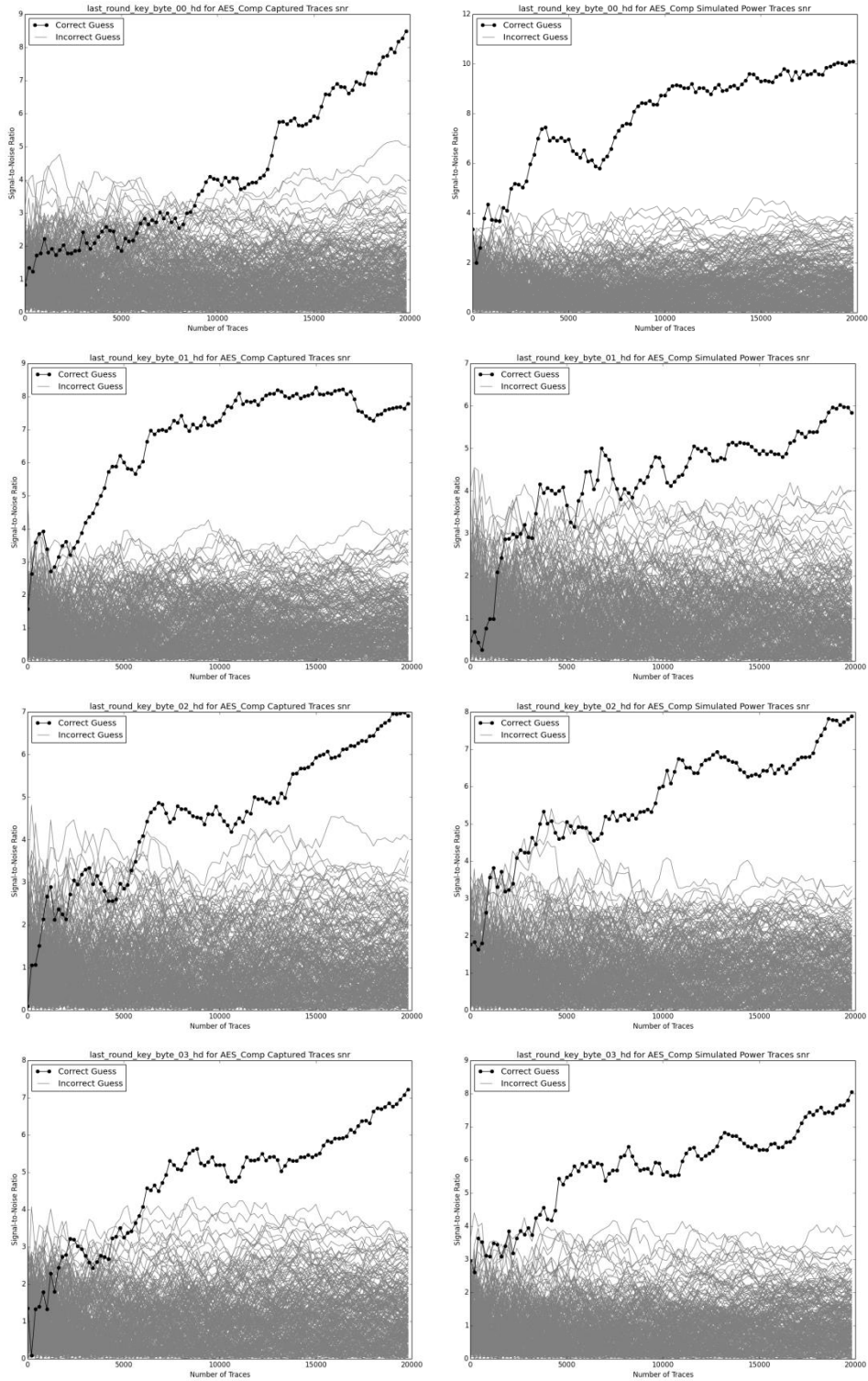
Correlation Ratios for AES Comp Measured (R) and Simulated (L), key bytes 4-7 with 20k traces



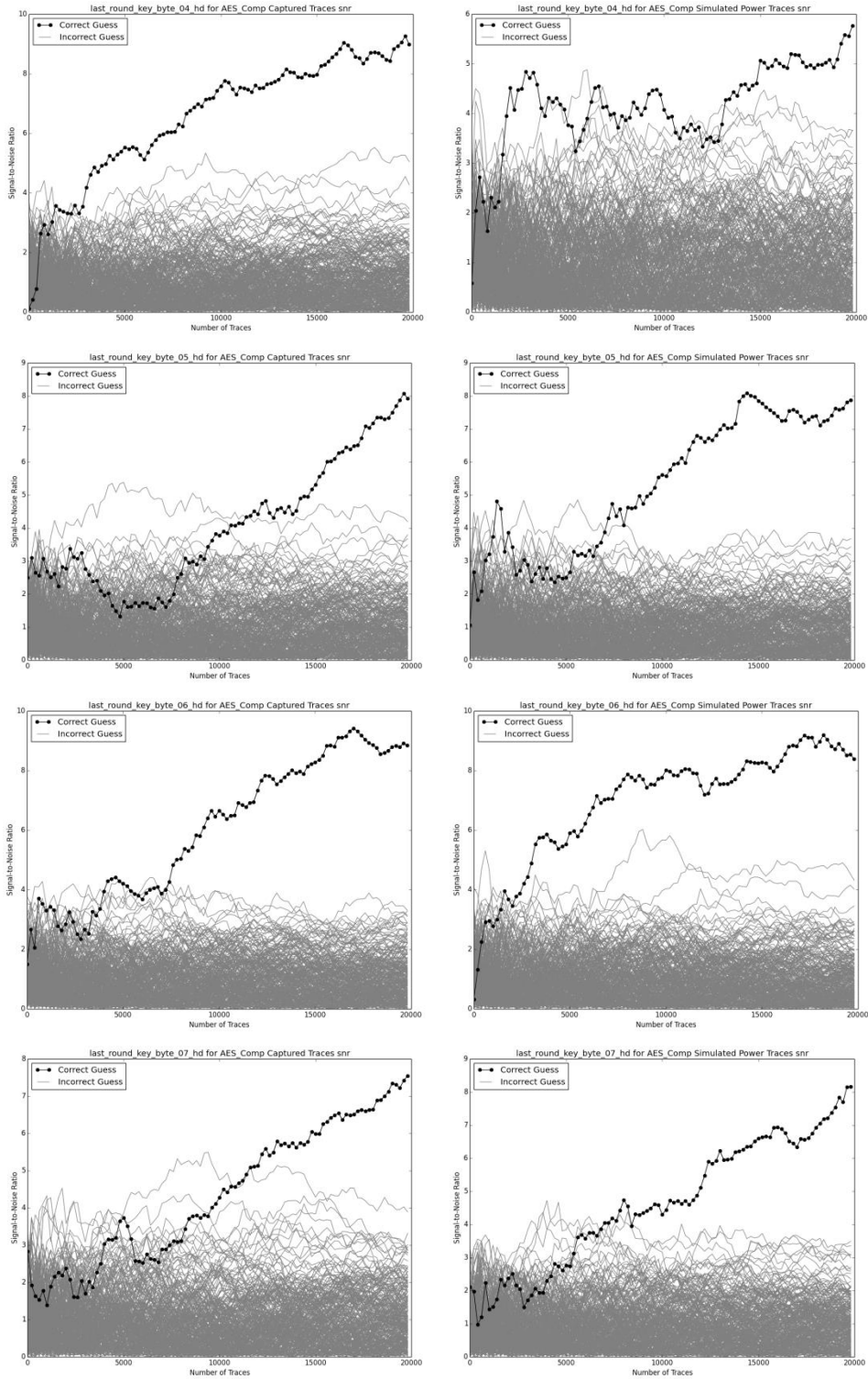
Correlation Ratios for AES Comp Measured (R) and Simulated (L), key bytes 8-11 with 20k traces



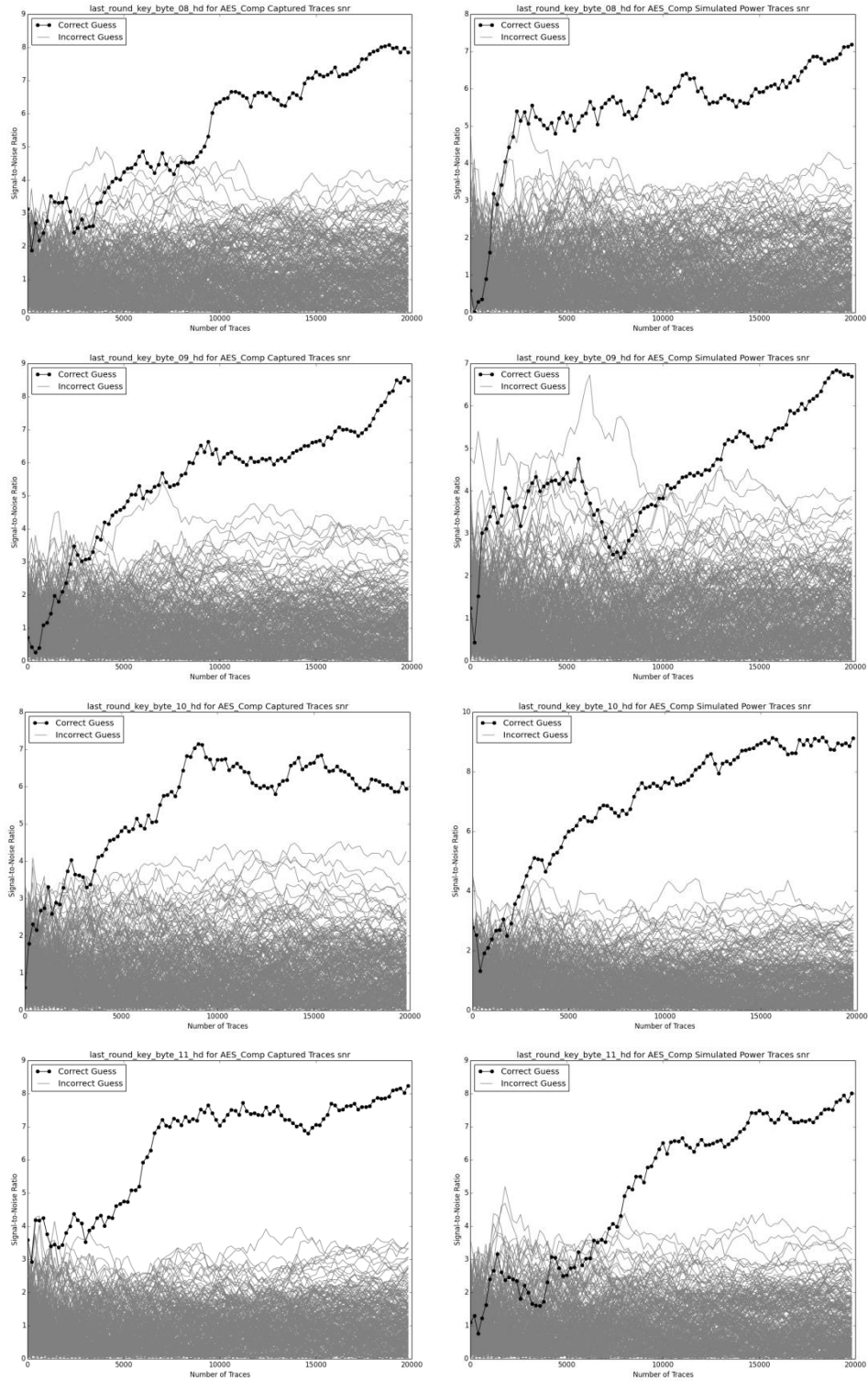
Correlation Ratios for AES Comp Measured (R) and Simulated (L), key bytes 12-15 with 20k traces



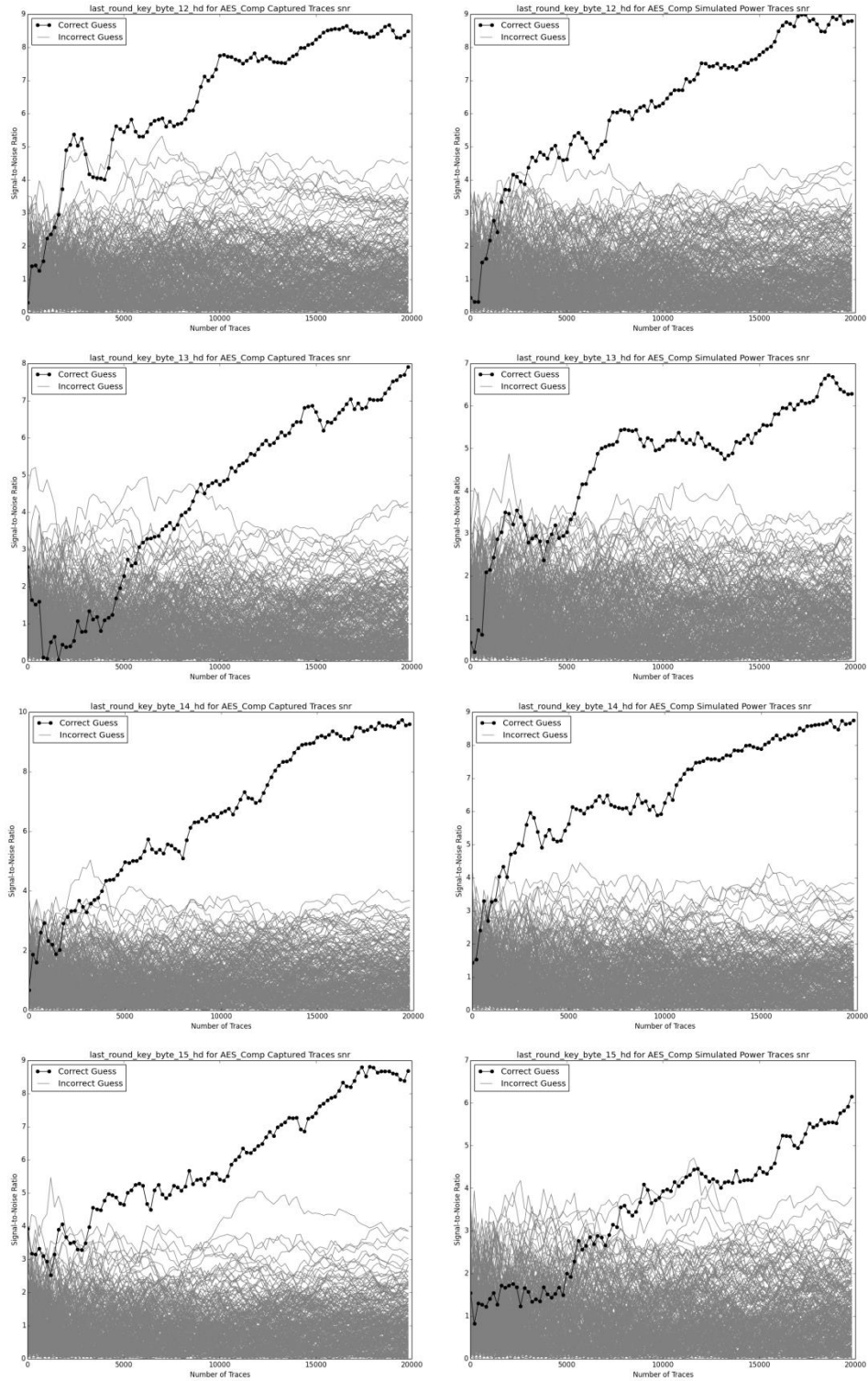
Signal-to-Noise Ratios for AES Comp Measured (R) and Simulated (L), key bytes 0-3, all guesses



Signal-to-Noise Ratios for AES Comp Measured (R) and Simulated (L), key bytes 4-7, all guesses

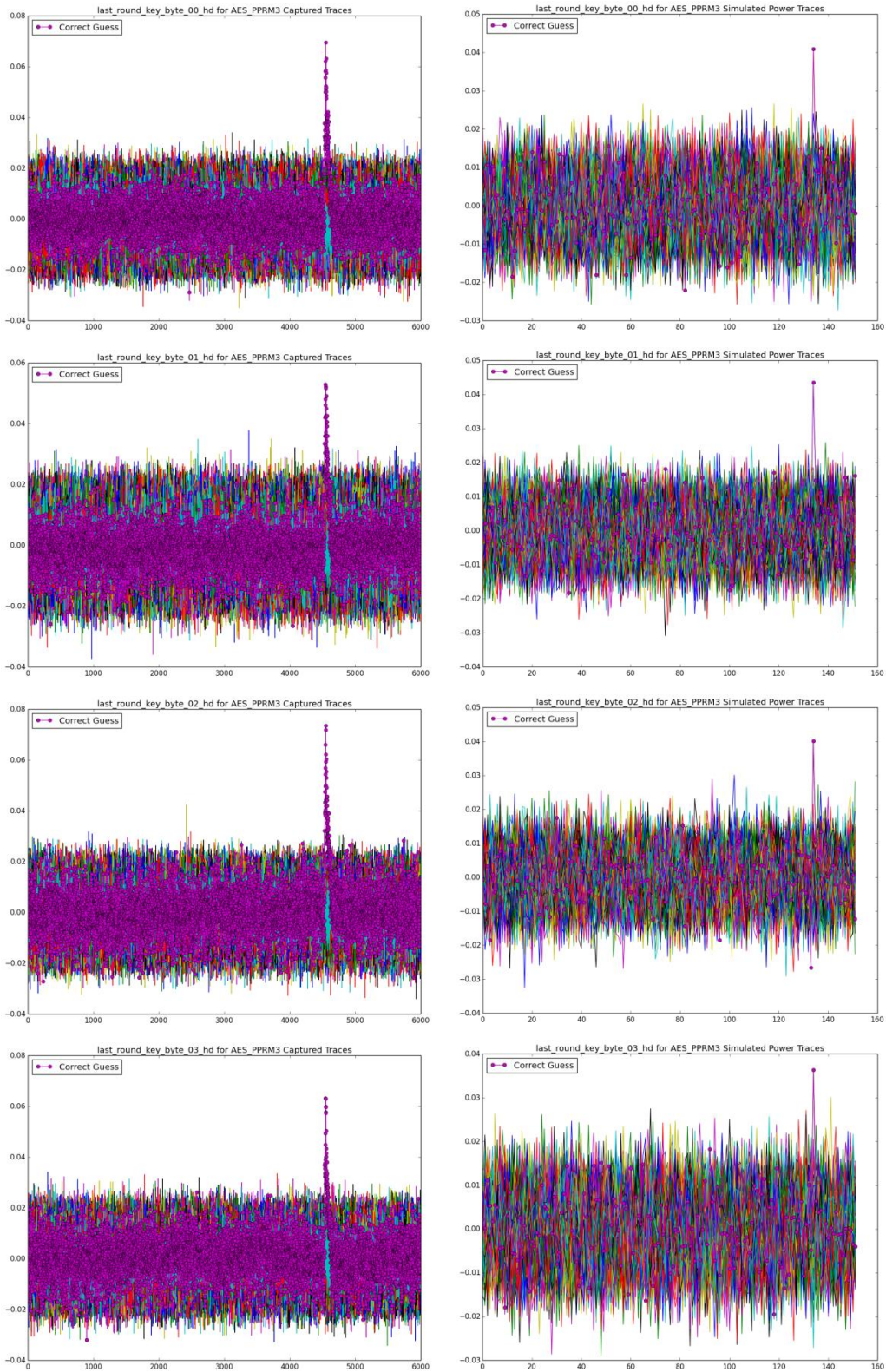


Signal-to-Noise Ratios for AES Comp Measured (R) and Simulated (L), key bytes 8-11, all guesses

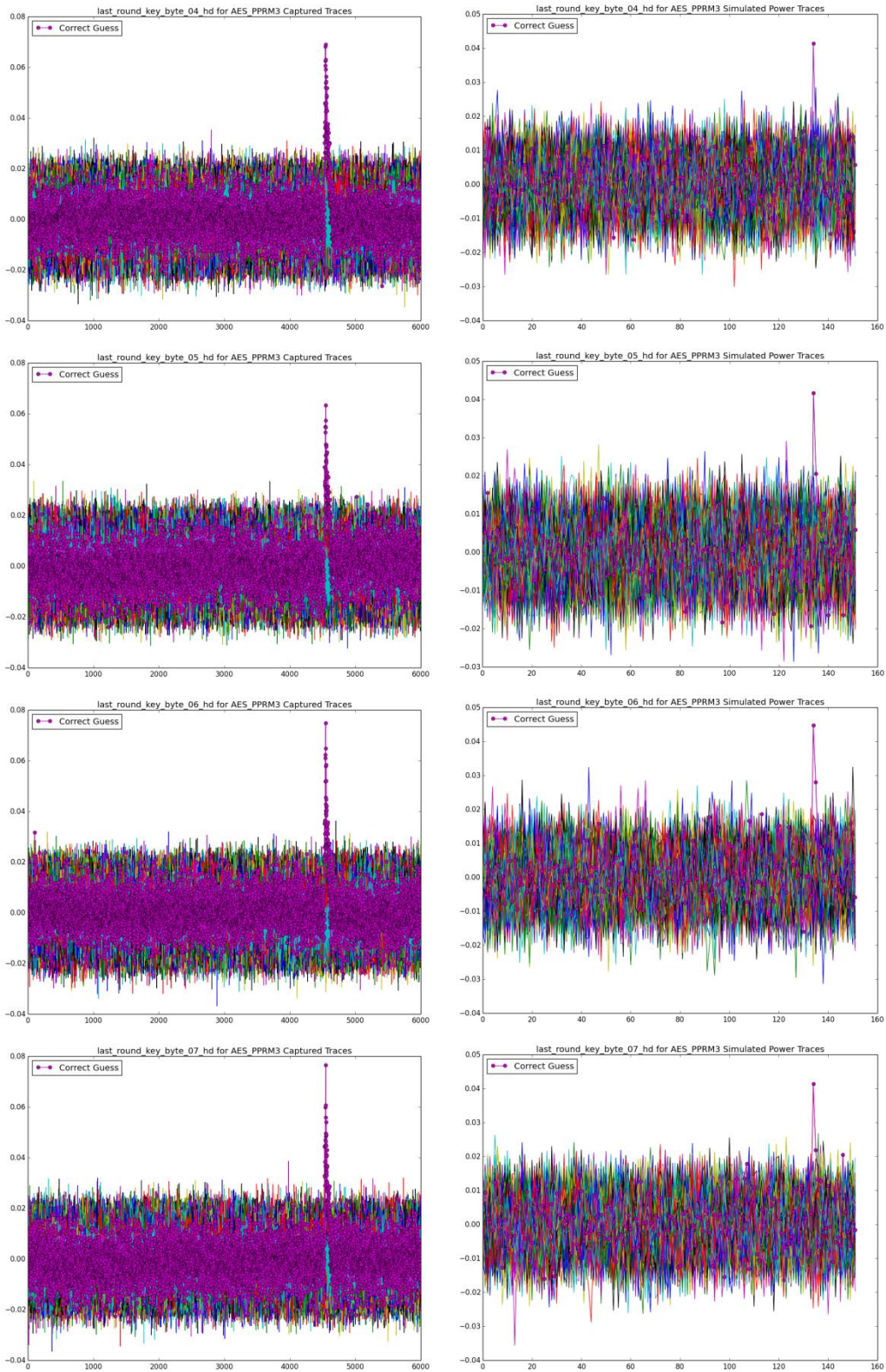


Signal-to-Noise Ratios for AES Comp Measured (R) and Simulated (L), key bytes 12-15, all guesses

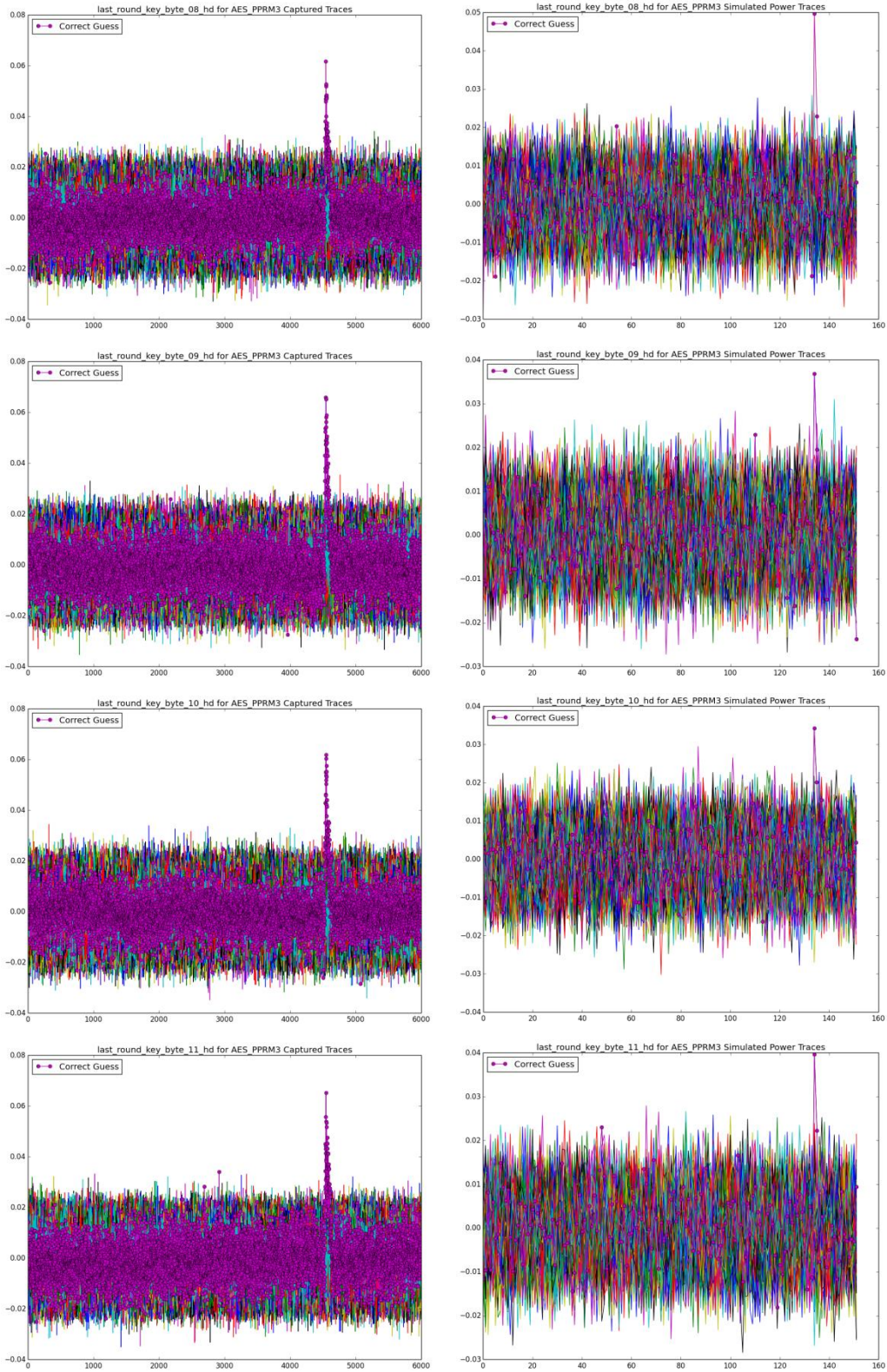
9.4 Appendix D: Full DPA Results for AES PPRM3



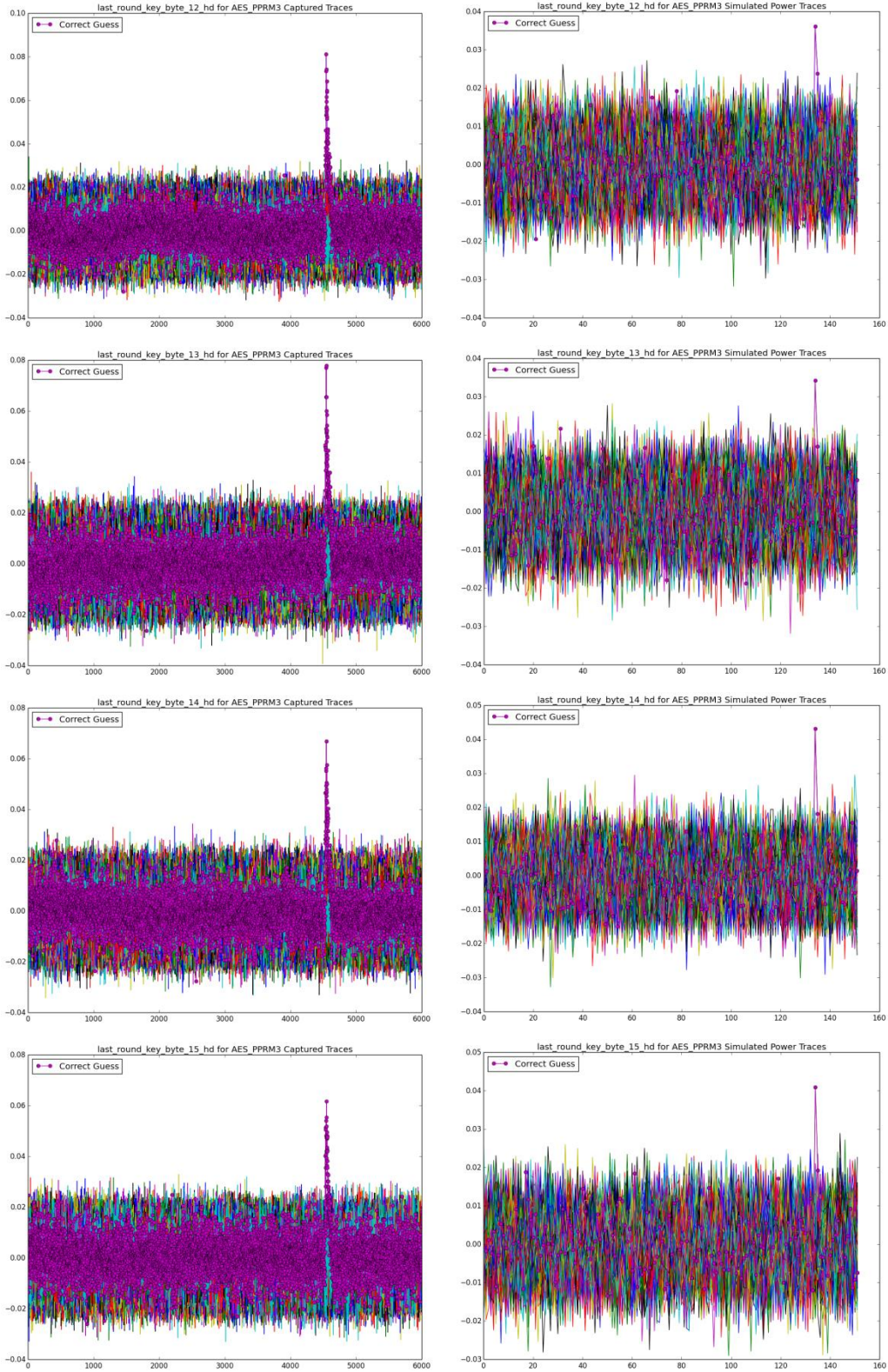
DPA Results for AES PPRM3 Measured (R) and Simulated (L), key bytes 0-3 with 20k traces



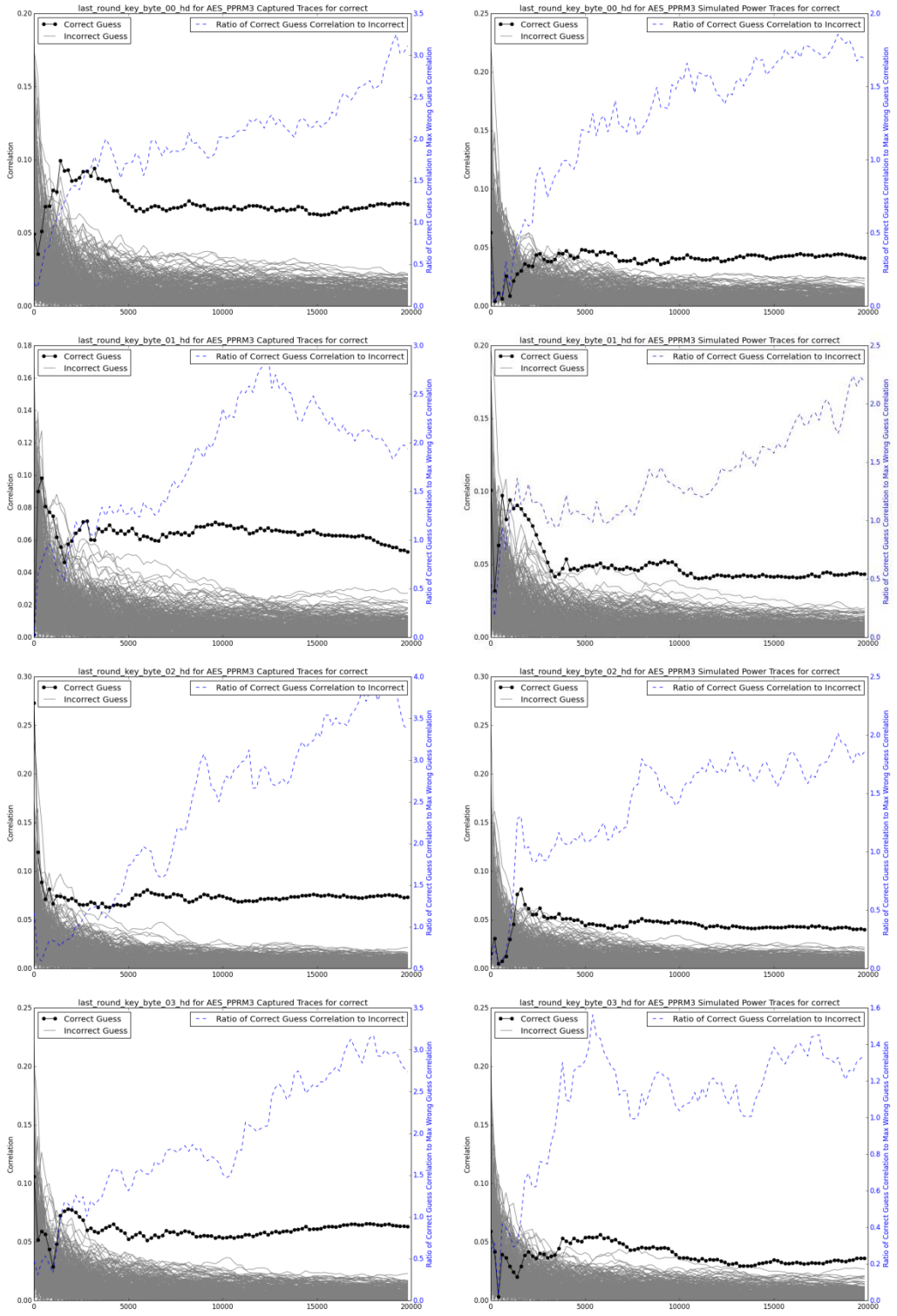
DPA Results for AES PPRM3 Measured (R) and Simulated (L), key bytes 4-7 with 20k traces



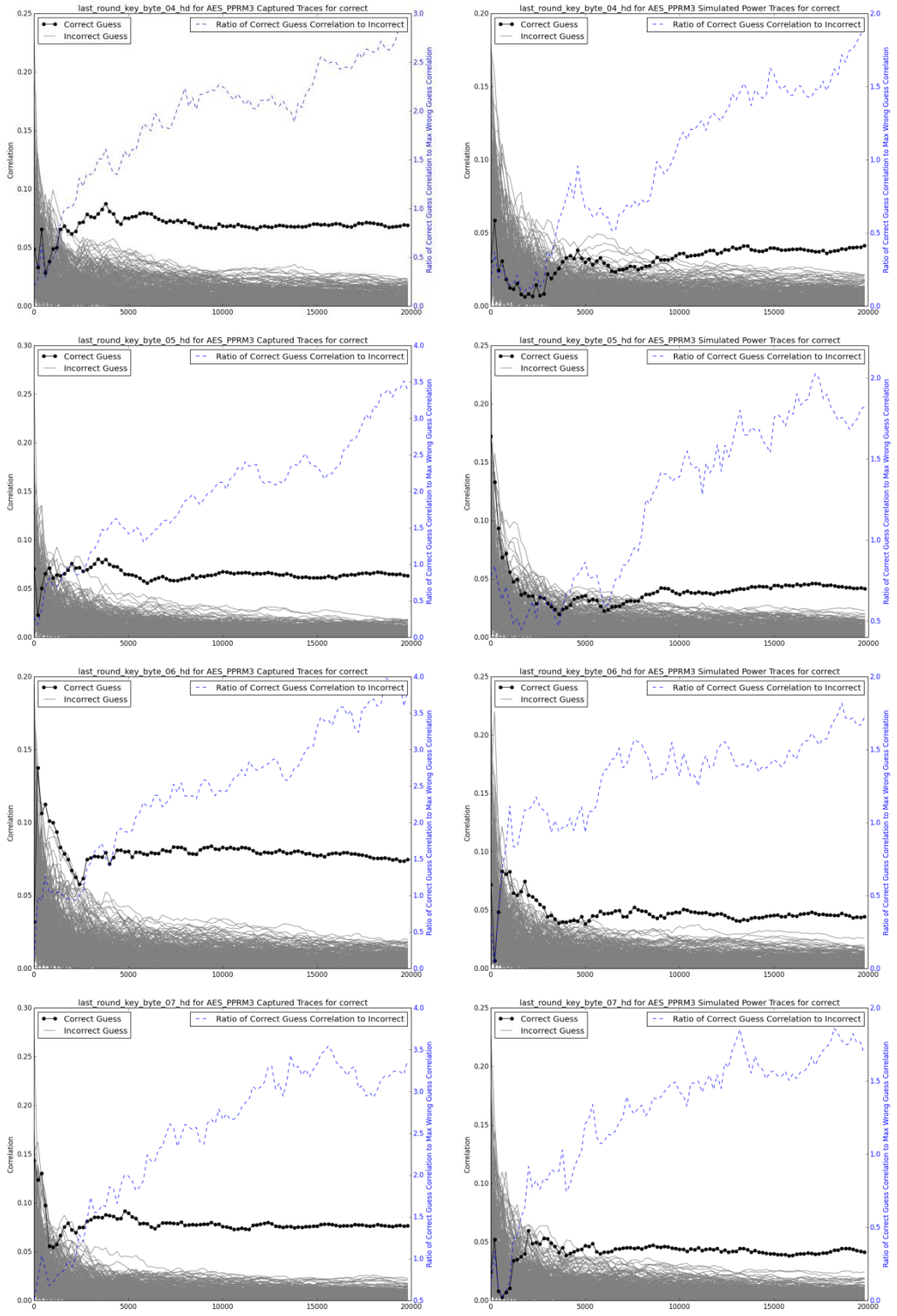
DPA Results for AES PPRM3 Measured (R) and Simulated (L), key bytes 8-11 with 20k traces



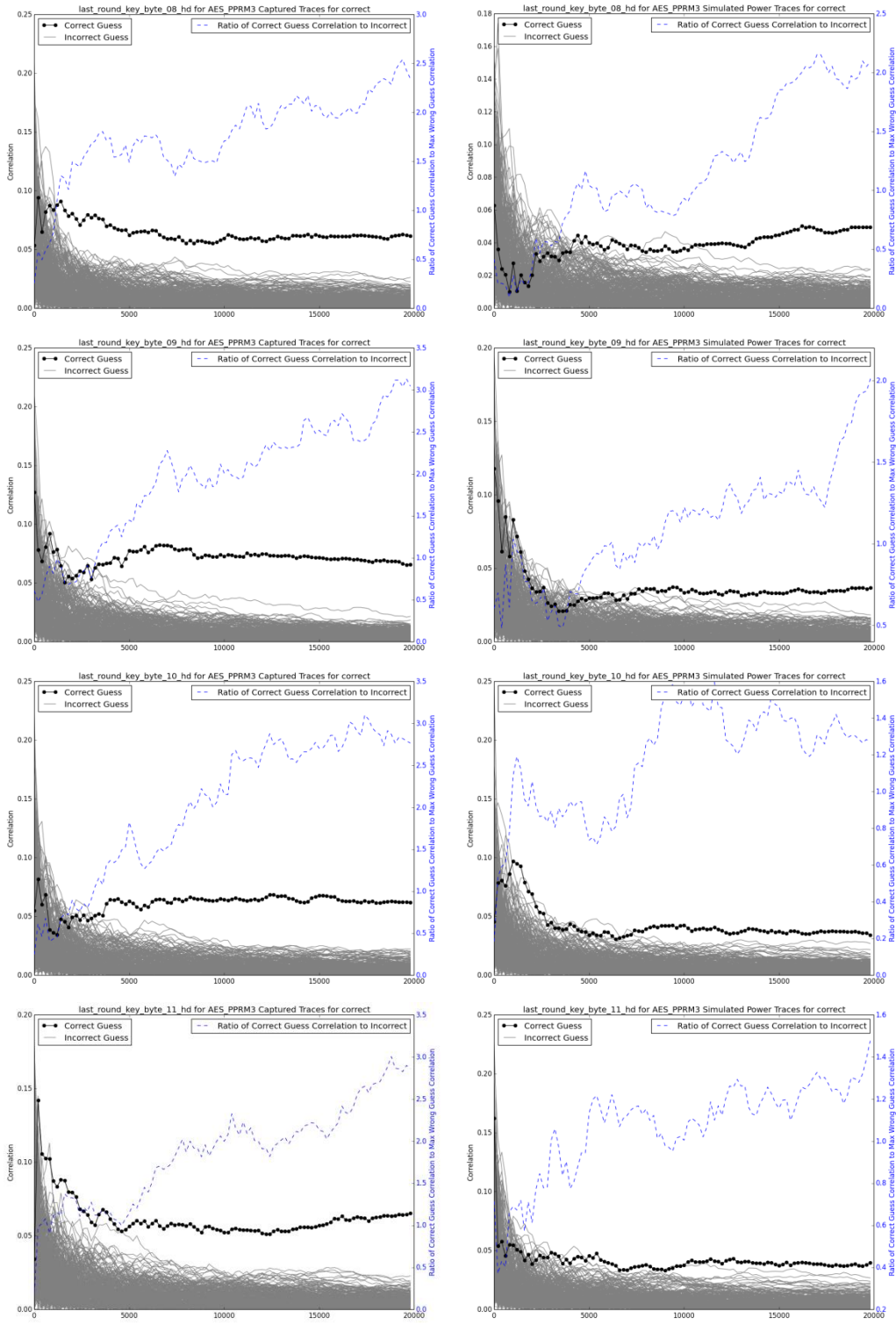
DPA Results for AES PPRM3 Measured (R) and Simulated (L), key bytes 12-15 with 20k traces



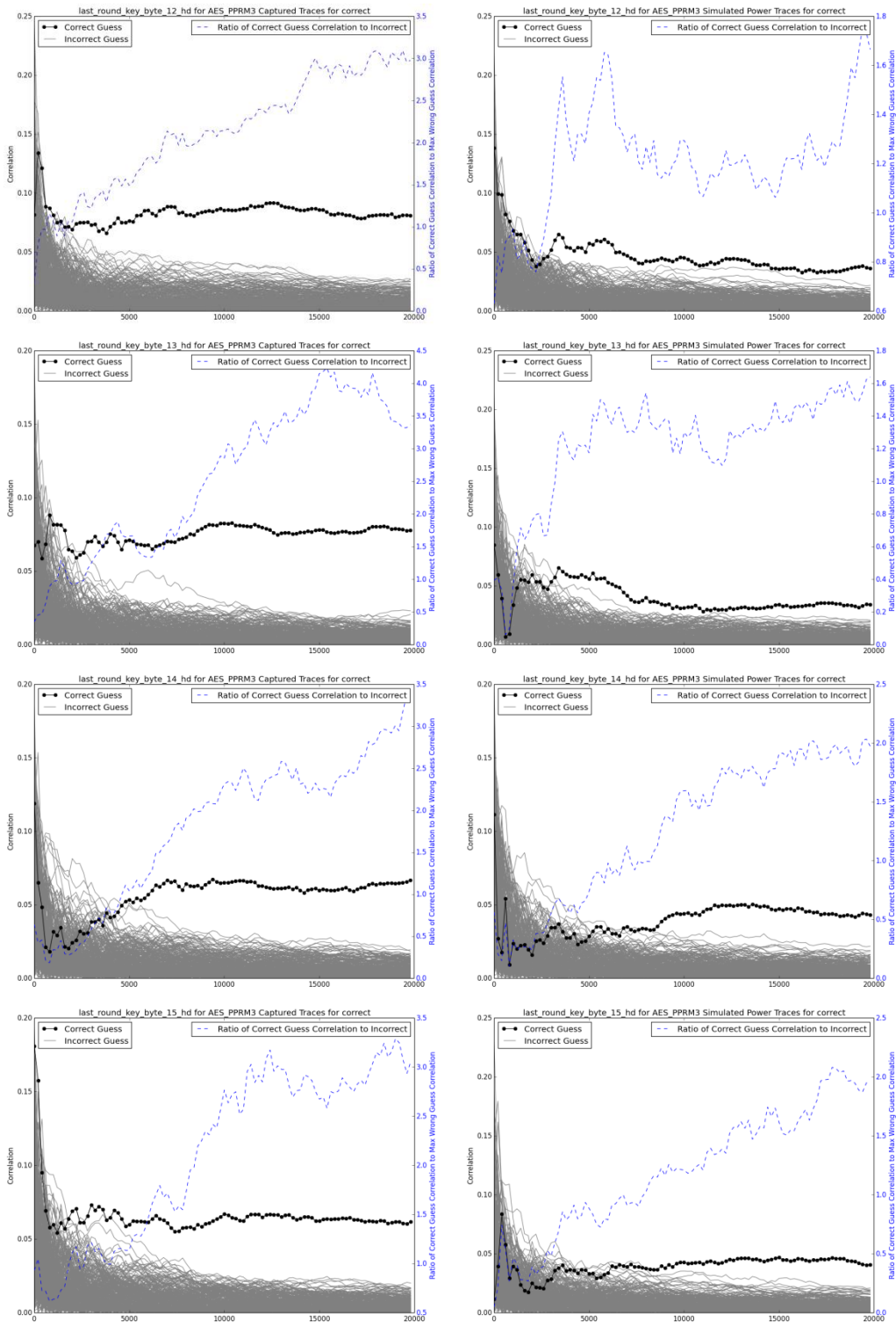
Correlation Ratios for AES PPRM3 Measured (R) and Simulated (L), key bytes 0-3 with 20k traces



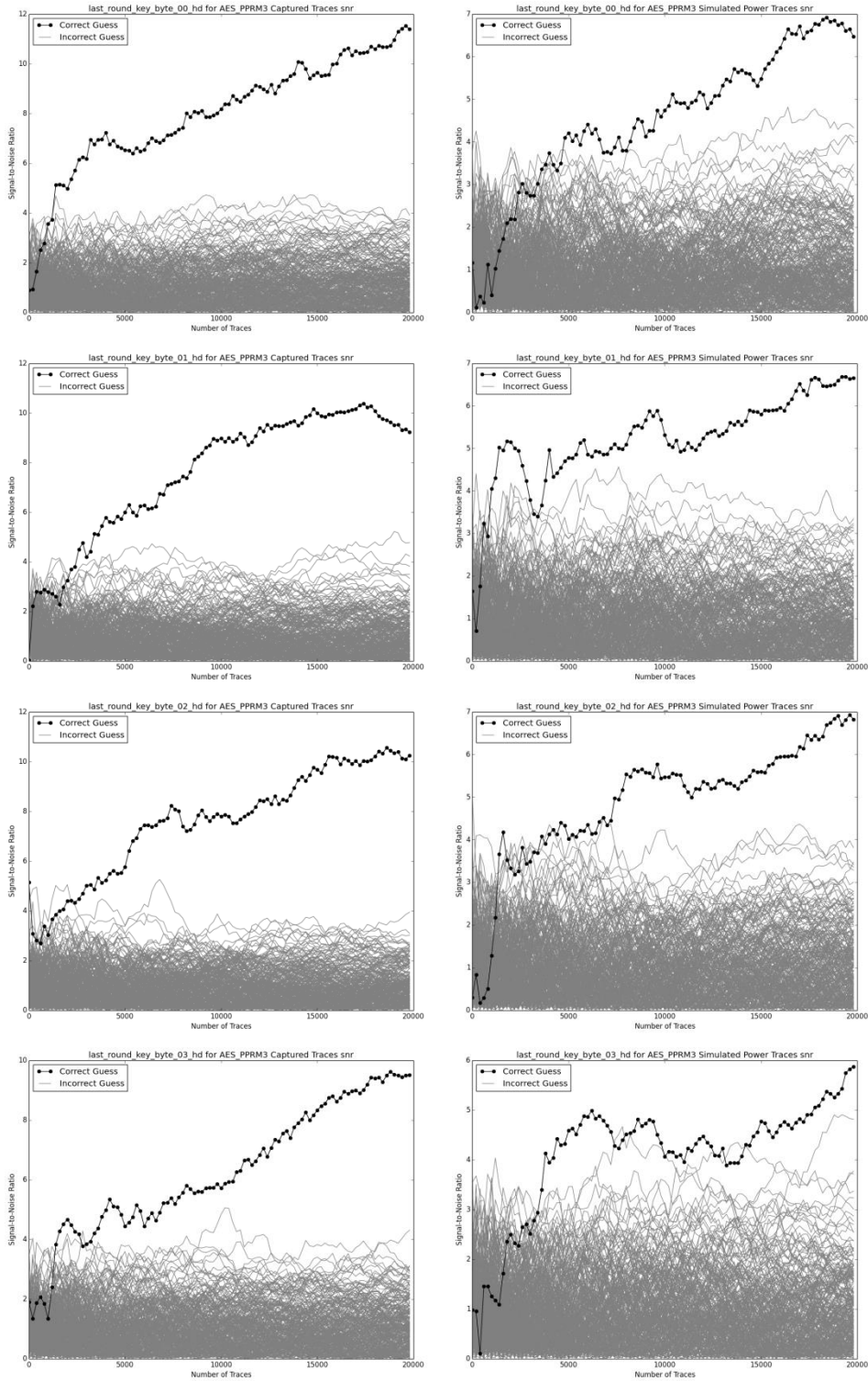
Correlation Ratios for AES PPRM3 Measured (R) and Simulated (L), key bytes 4-7 with 20k traces



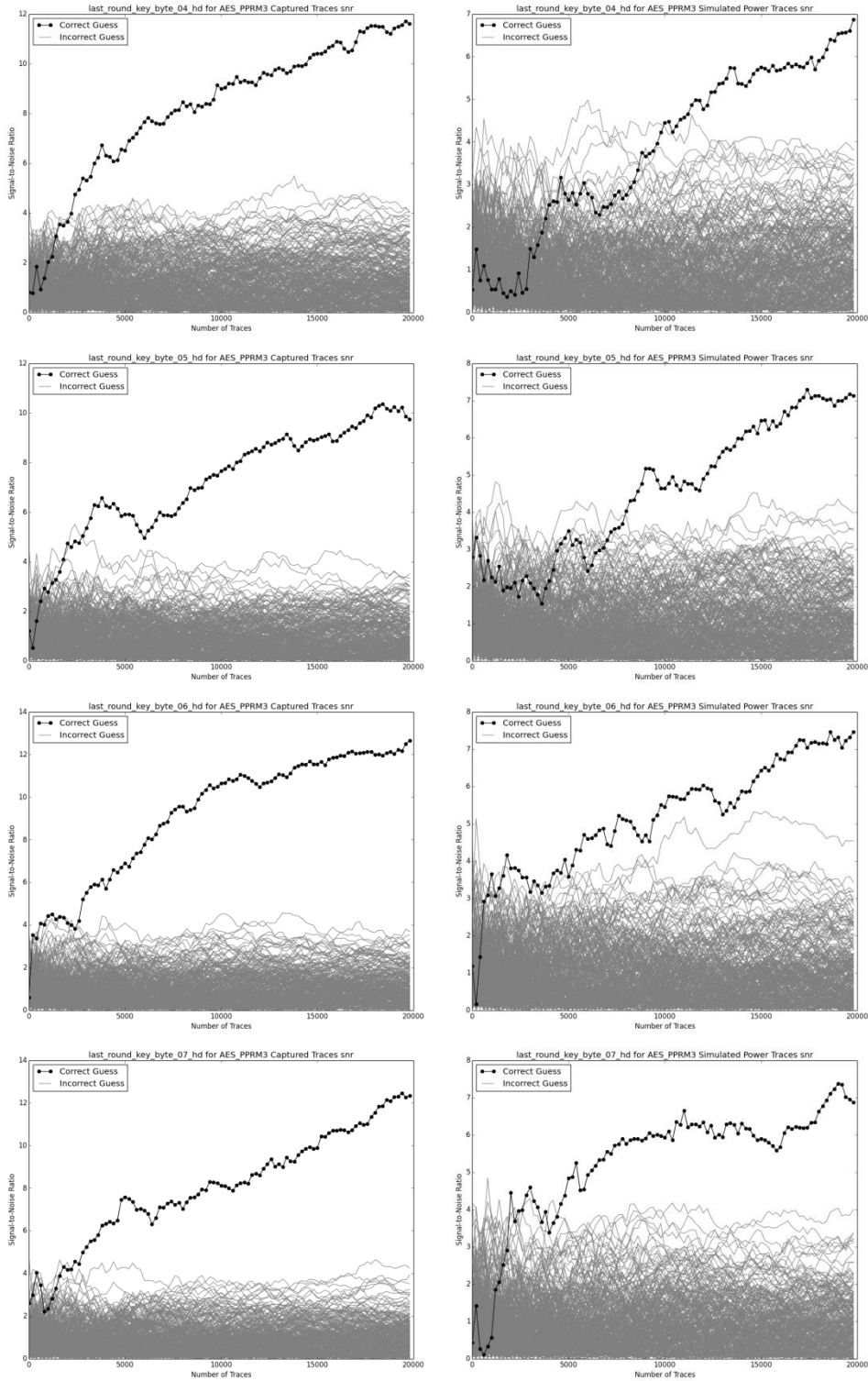
Correlation Ratios for AES PPRM3 Measured (R) and Simulated (L), key bytes 8-11 with 20k traces



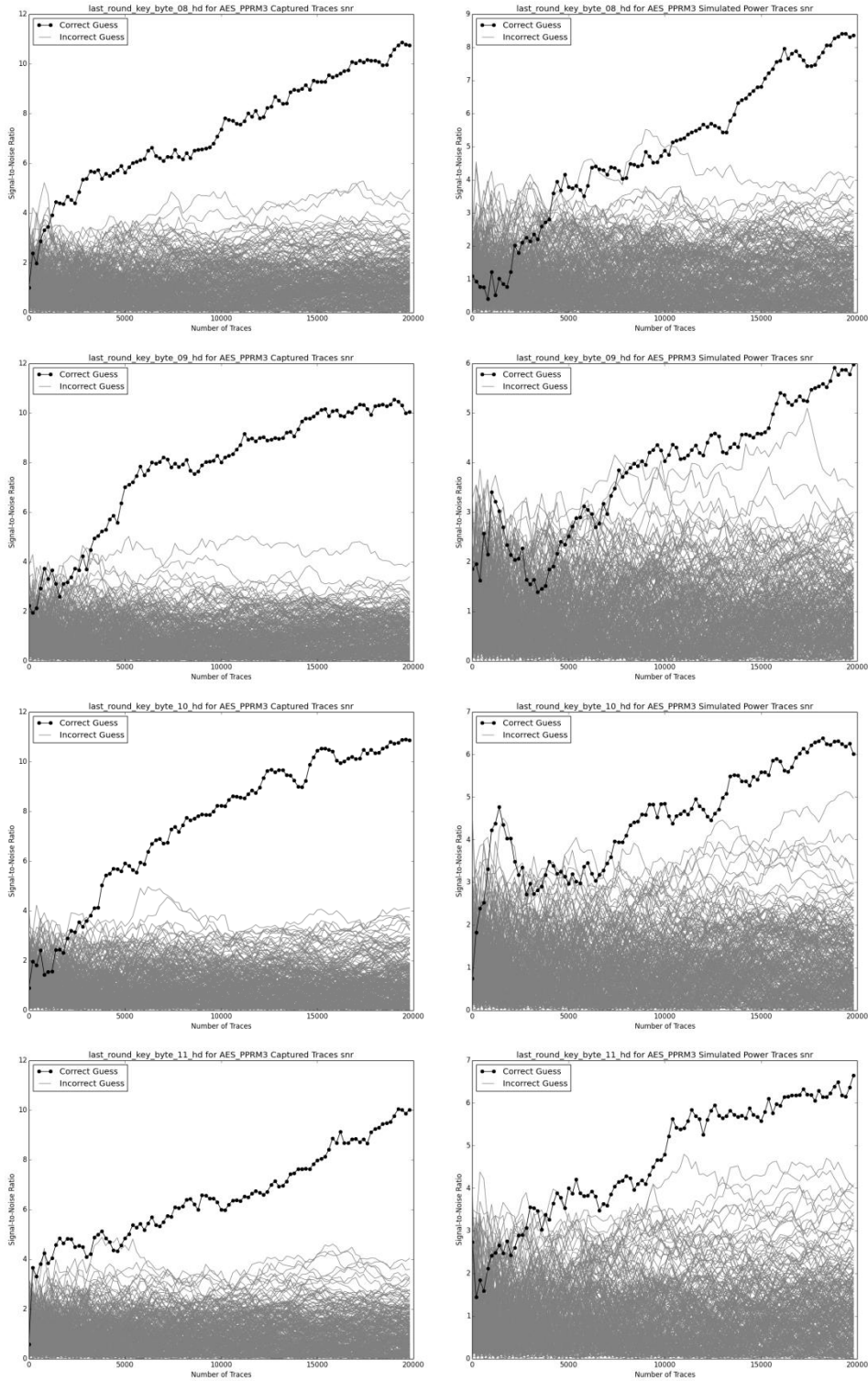
Correlation Ratios for AES PPRM3 Measured (R) and Simulated (L), key bytes 12-15 with 20k traces



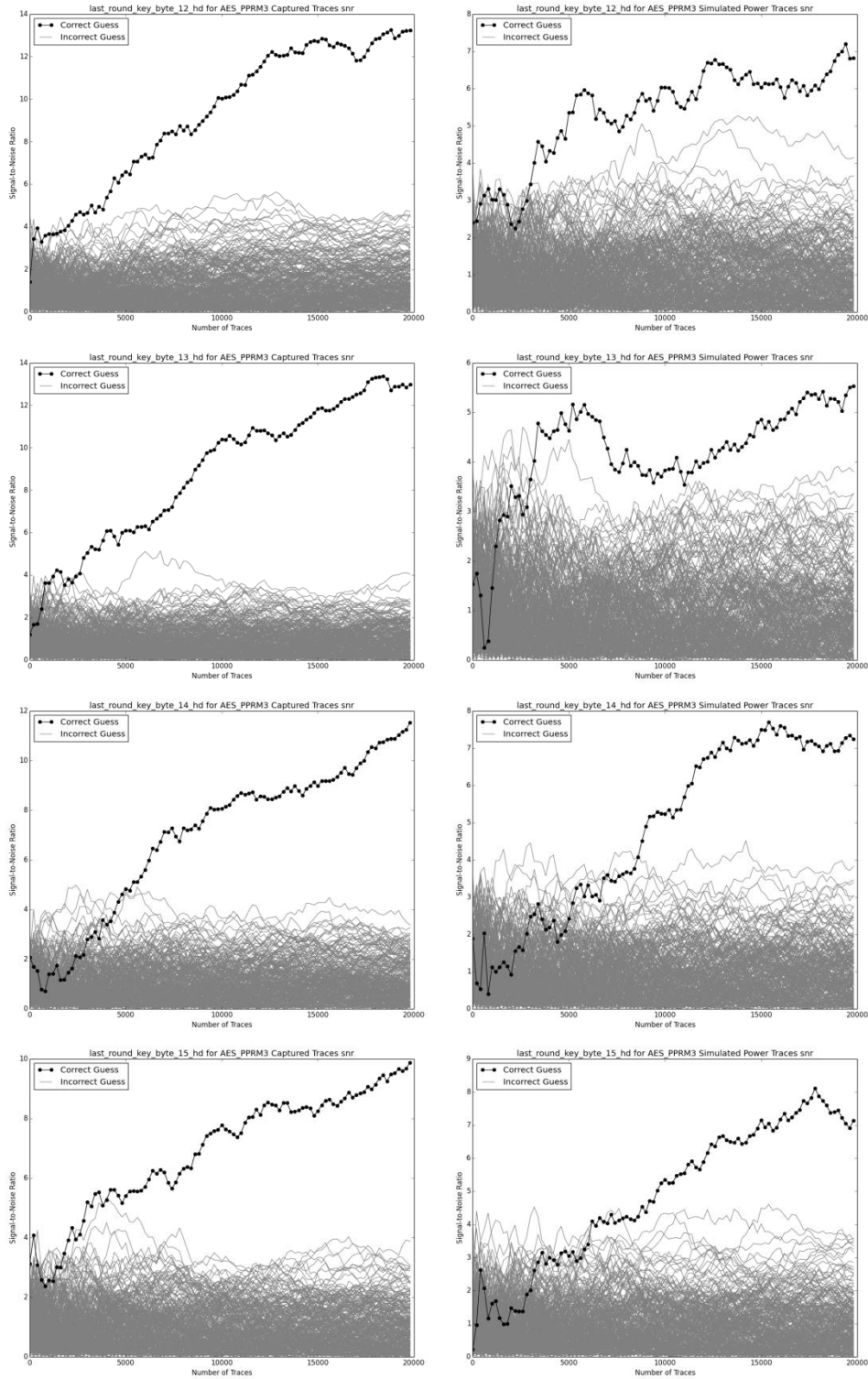
Signal-to-Noise Ratios for AES PPRM3 Measured (R) and Simulated (L), key bytes 0-3, all guesses



Signal-to-Noise Ratios for AES PPRM3 Measured (R) and Simulated (L), key bytes 4-7, all guesses

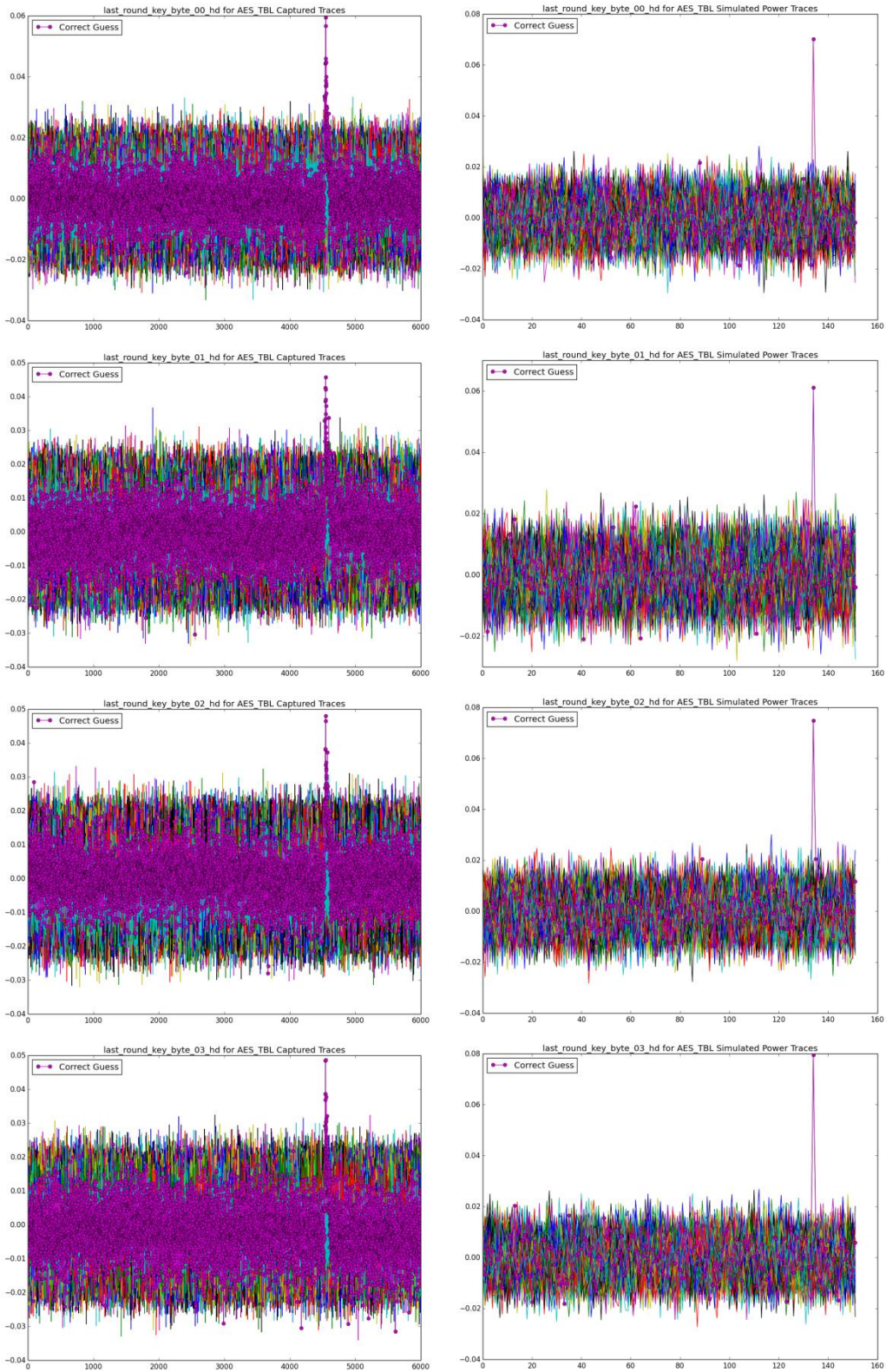


Signal-to-Noise Ratios for AES PPRM3 Measured (R) and Simulated (L), key bytes 8-11, all guesses

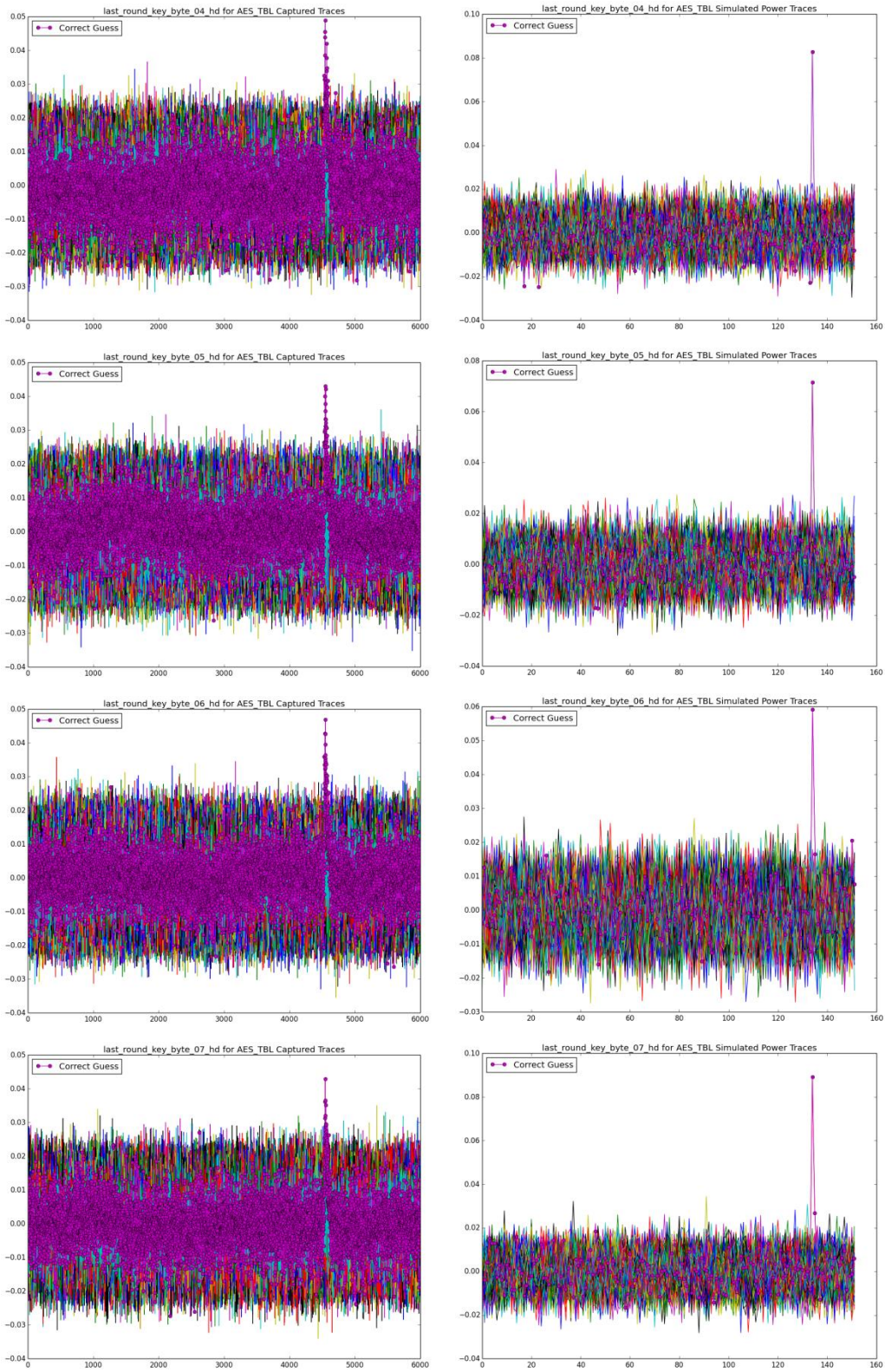


Signal-to-Noise Ratios for AES PPRM3 Measured (R) and Simulated (L), key bytes 12-15, all guesses

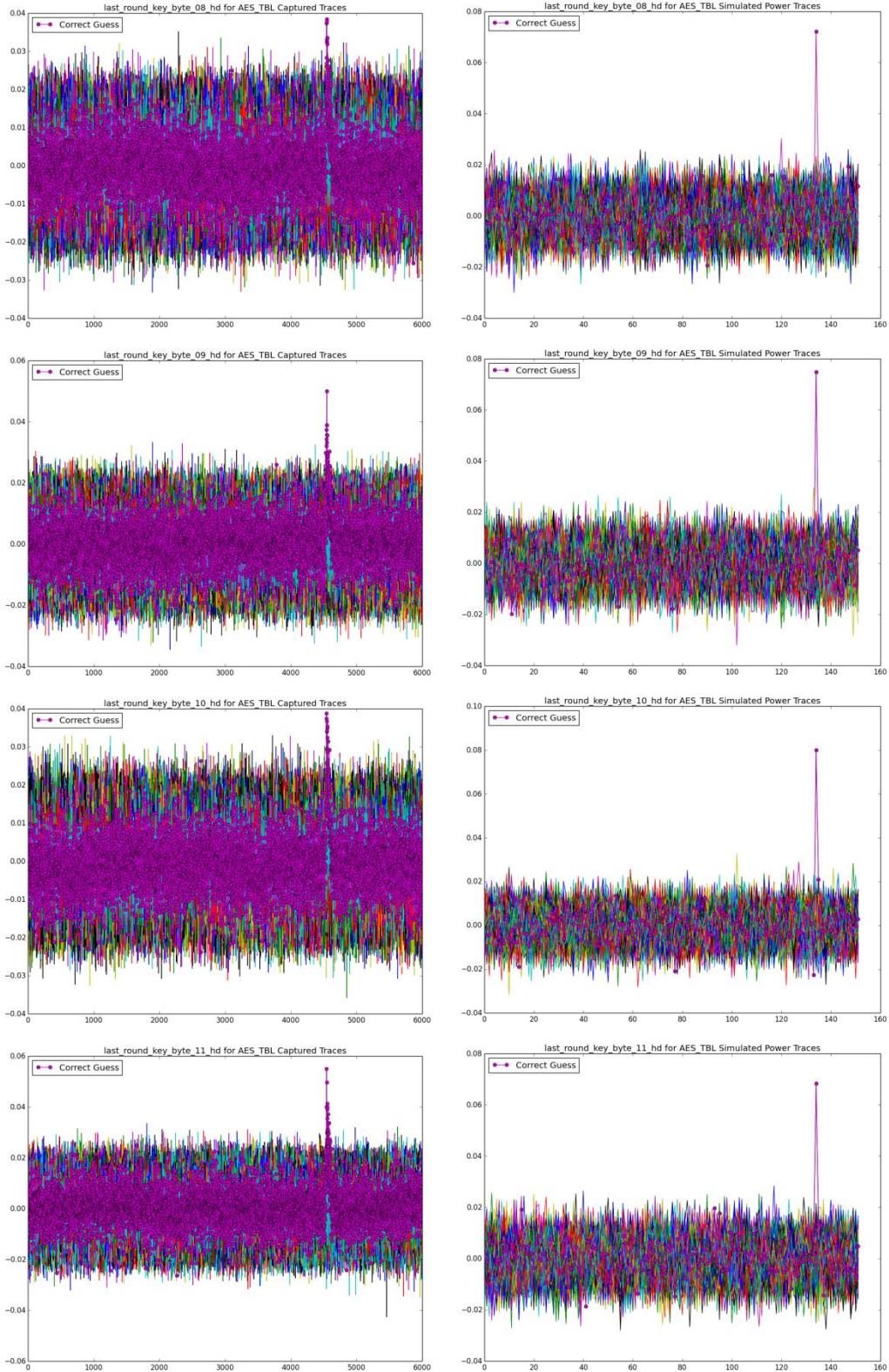
9.5 Appendix E: Full DPA Results for AES TBL



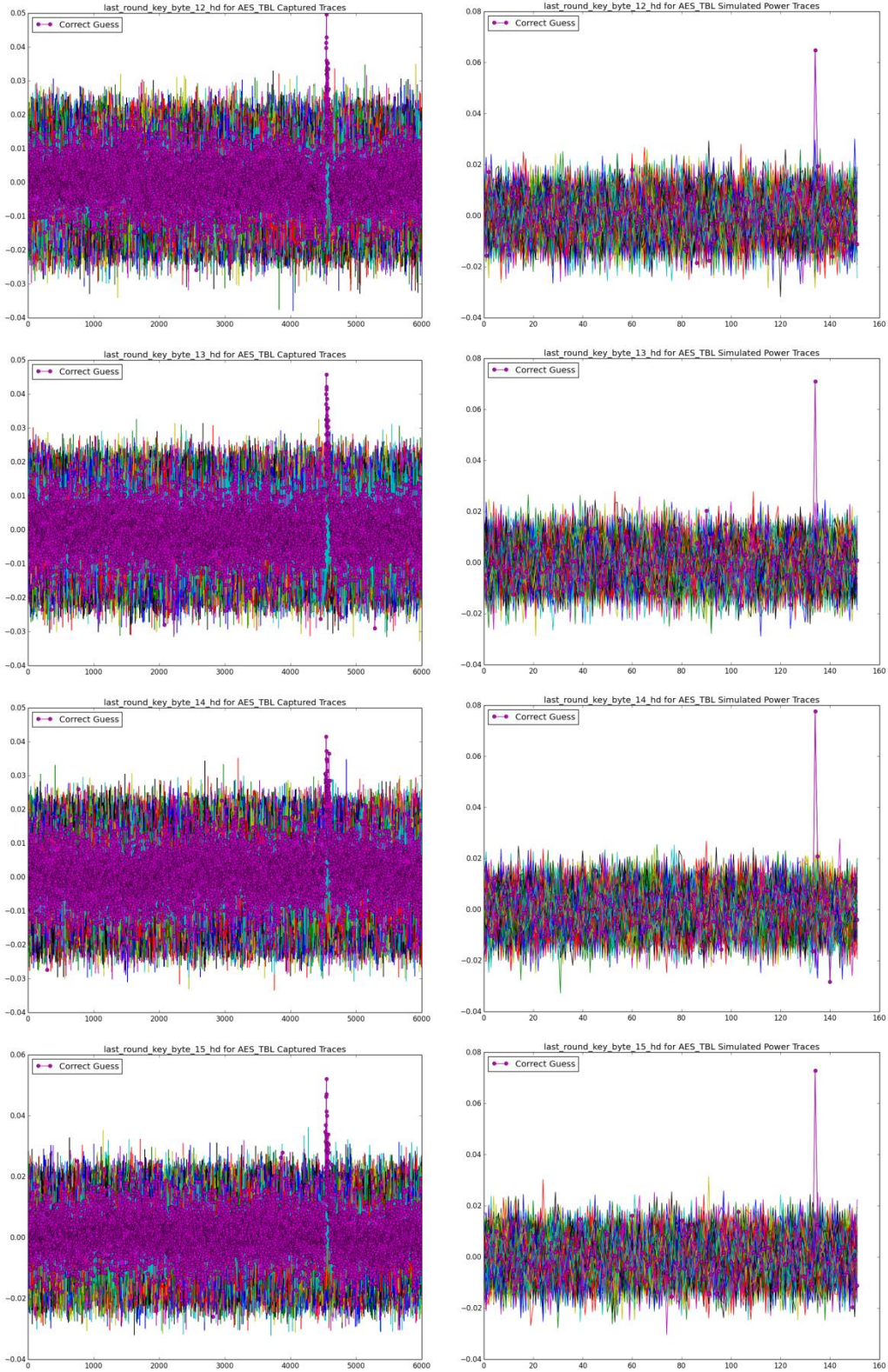
DPA Results for AES TBL Measured (R) and Simulated (L), key bytes 0-3 with 20k traces



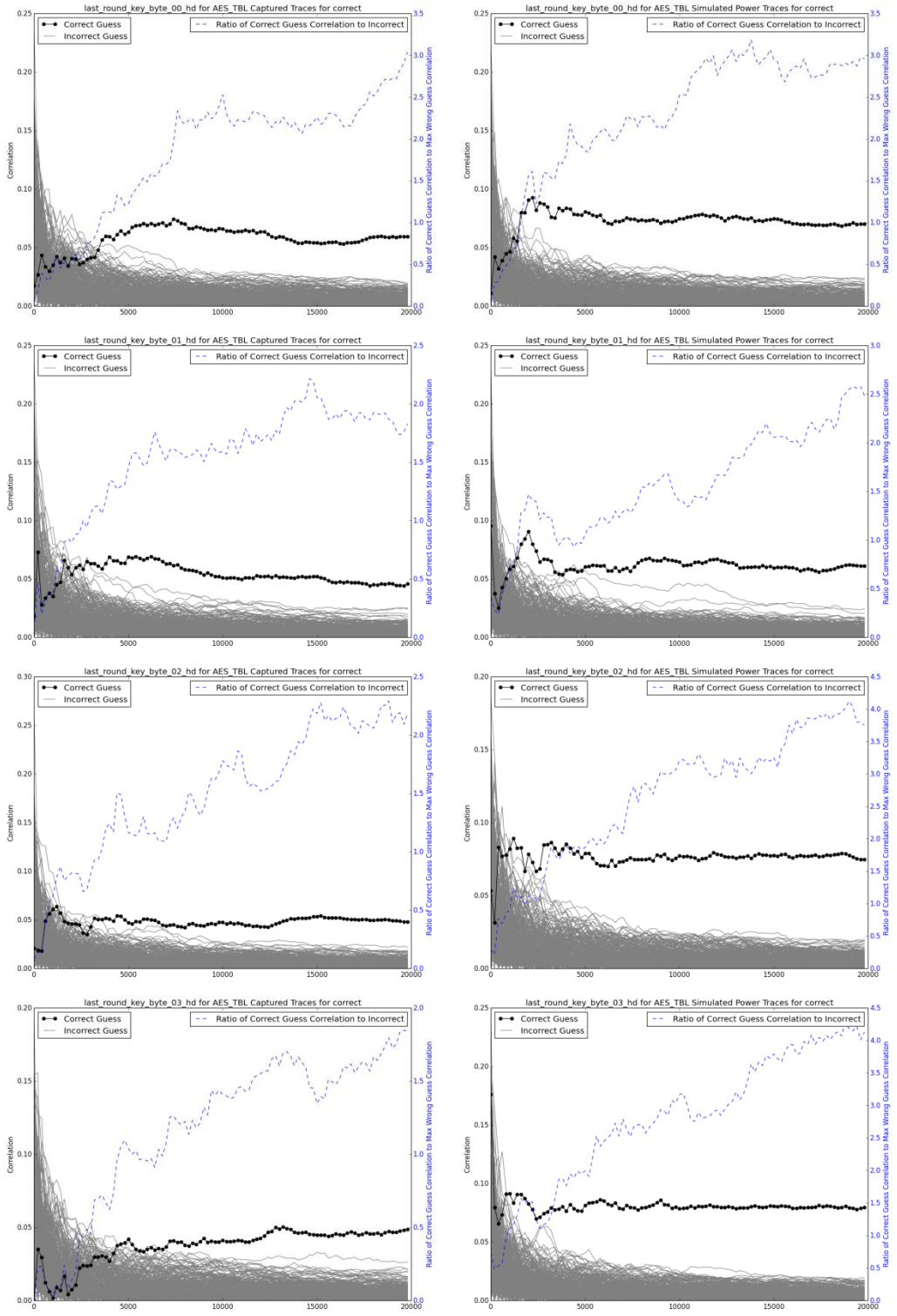
DPA Results for AES TBL Measured (R) and Simulated (L), key bytes 4-7 with 20k traces



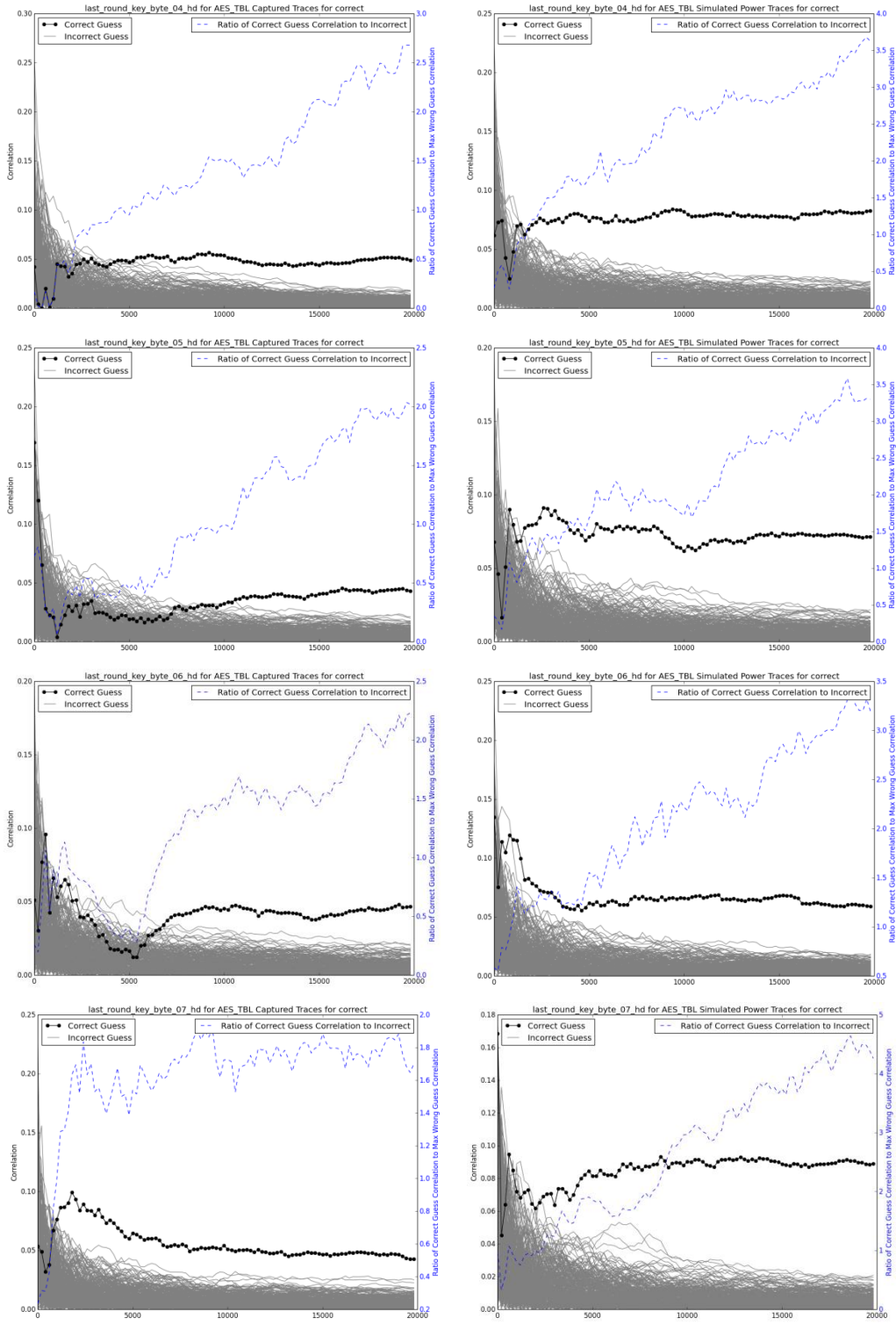
DPA Results for AES TBL Measured (R) and Simulated (L), key bytes 8-11 with 20k traces



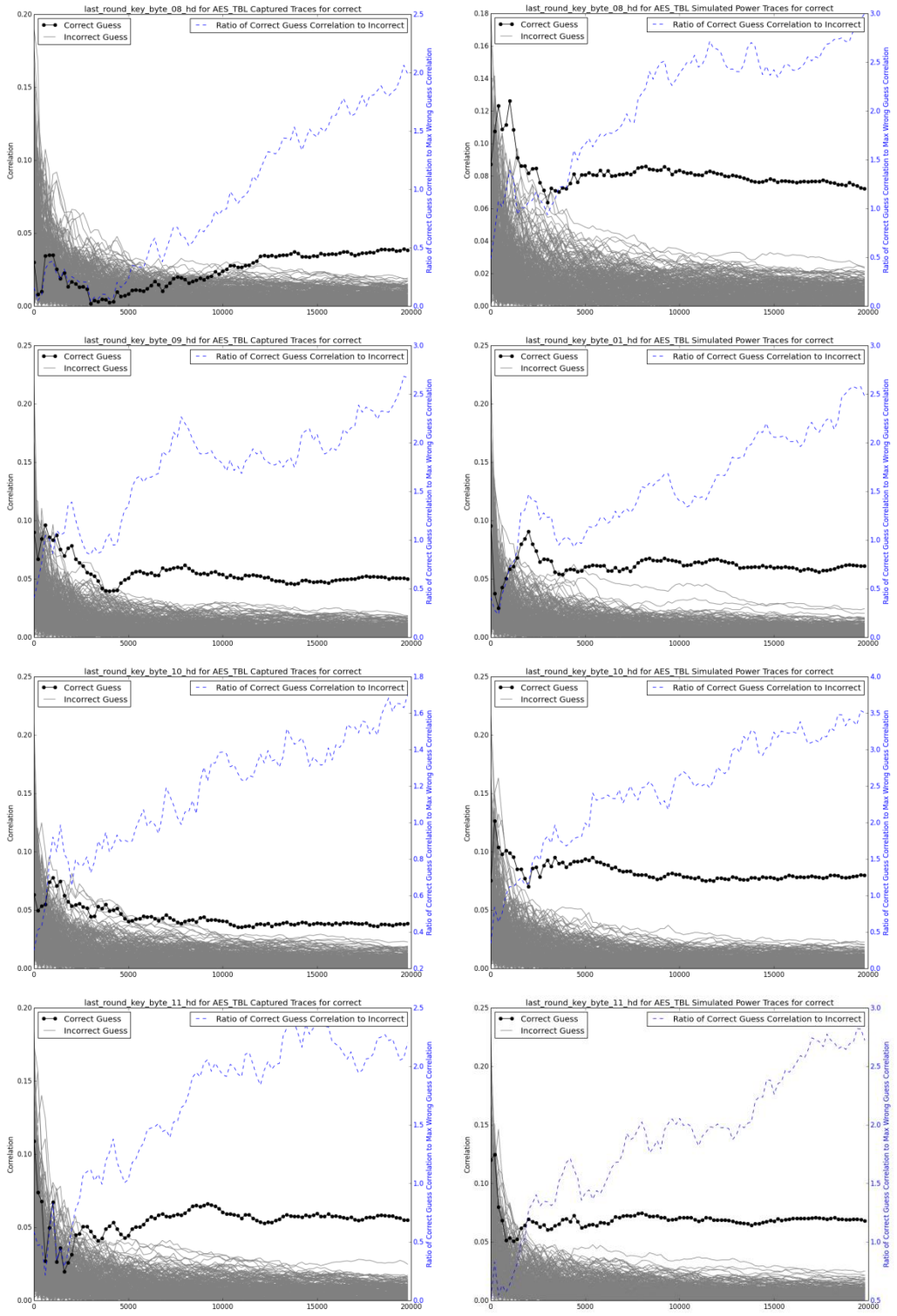
DPA Results for AES TBL Measured (R) and Simulated (L), key bytes 12-15 with 20k traces



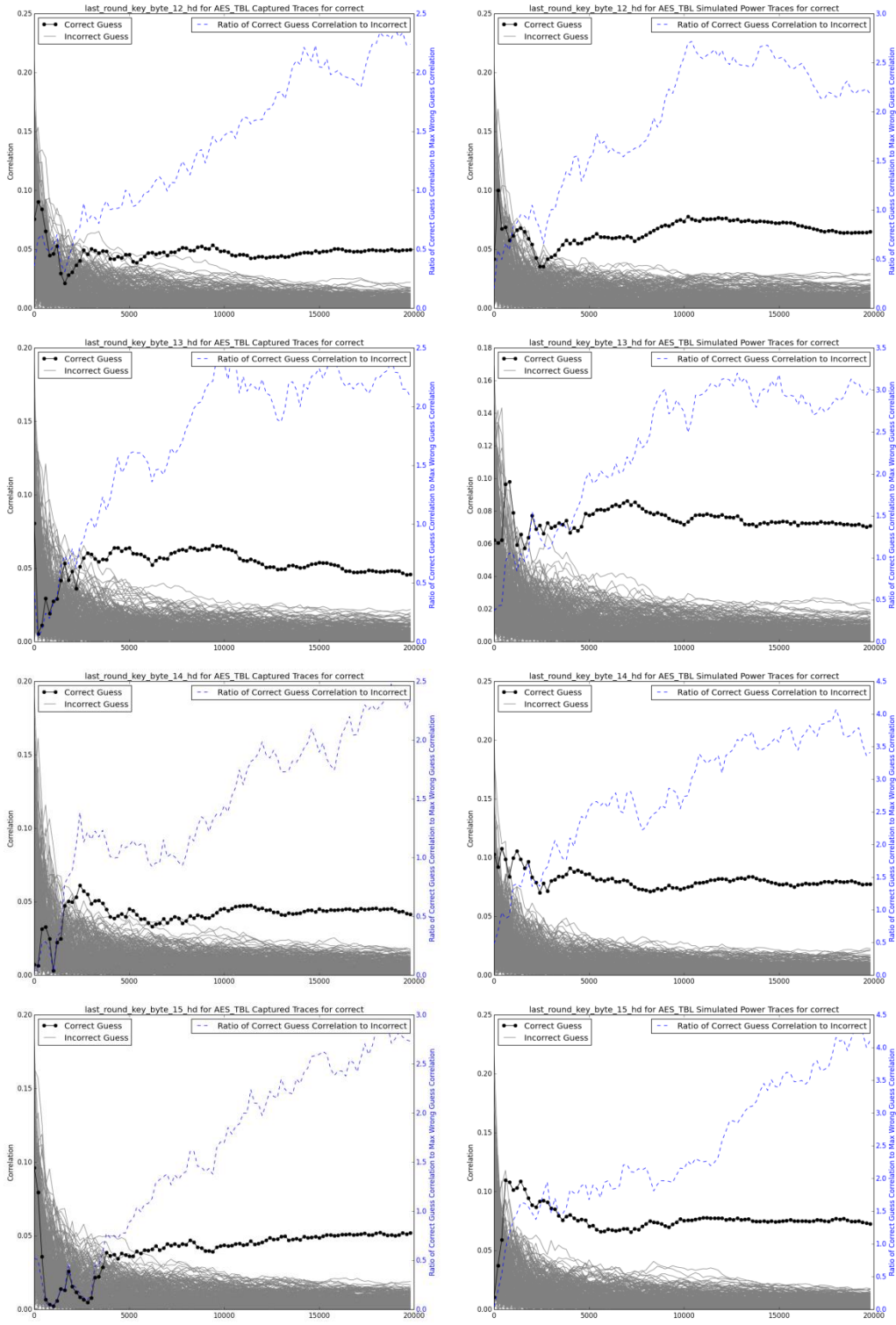
Correlation Ratios for AES TBL Measured (R) and Simulated (L), key bytes 0-3 with 20k traces



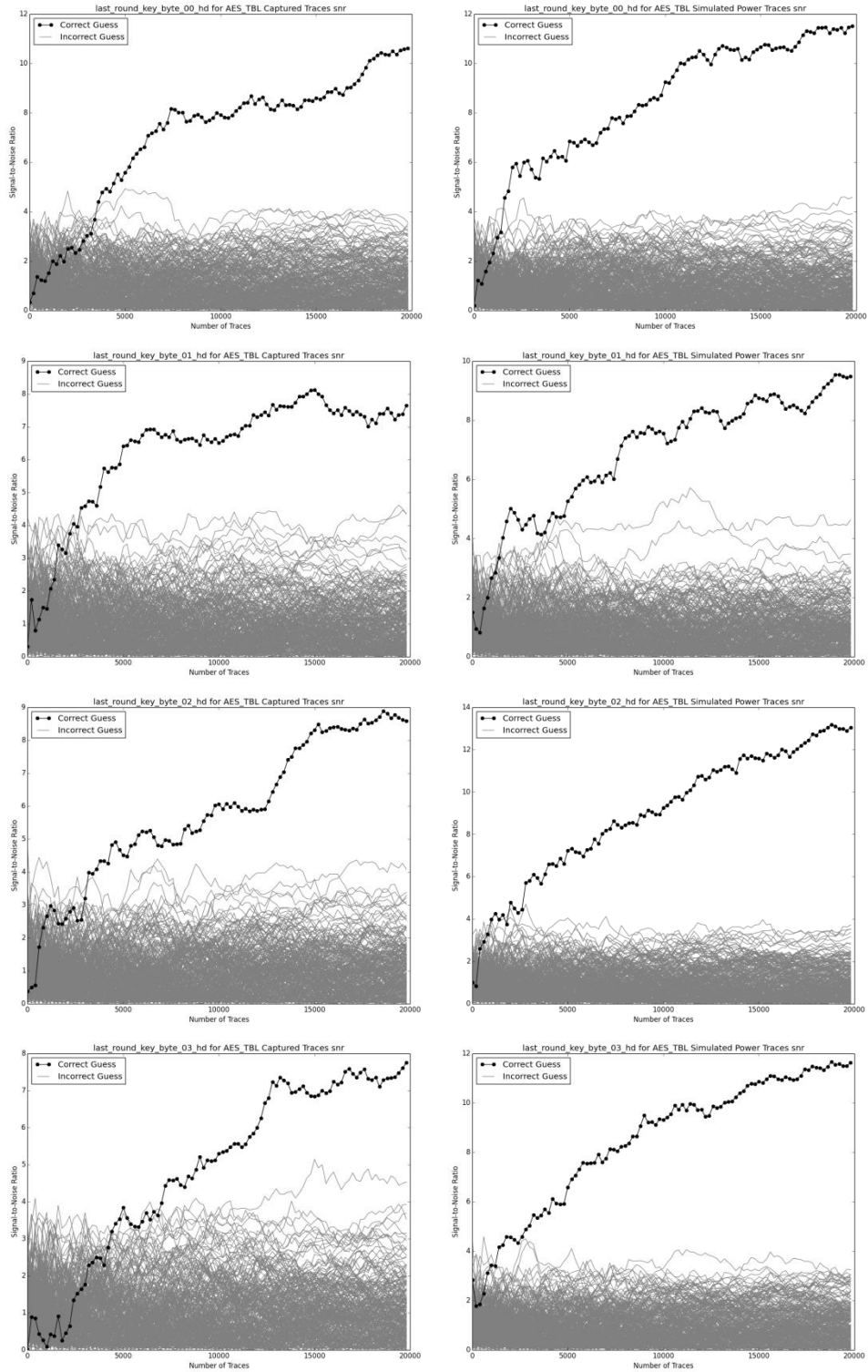
Correlation Ratios for AES TBL Measured (R) and Simulated (L), key bytes 4-7 with 20k traces



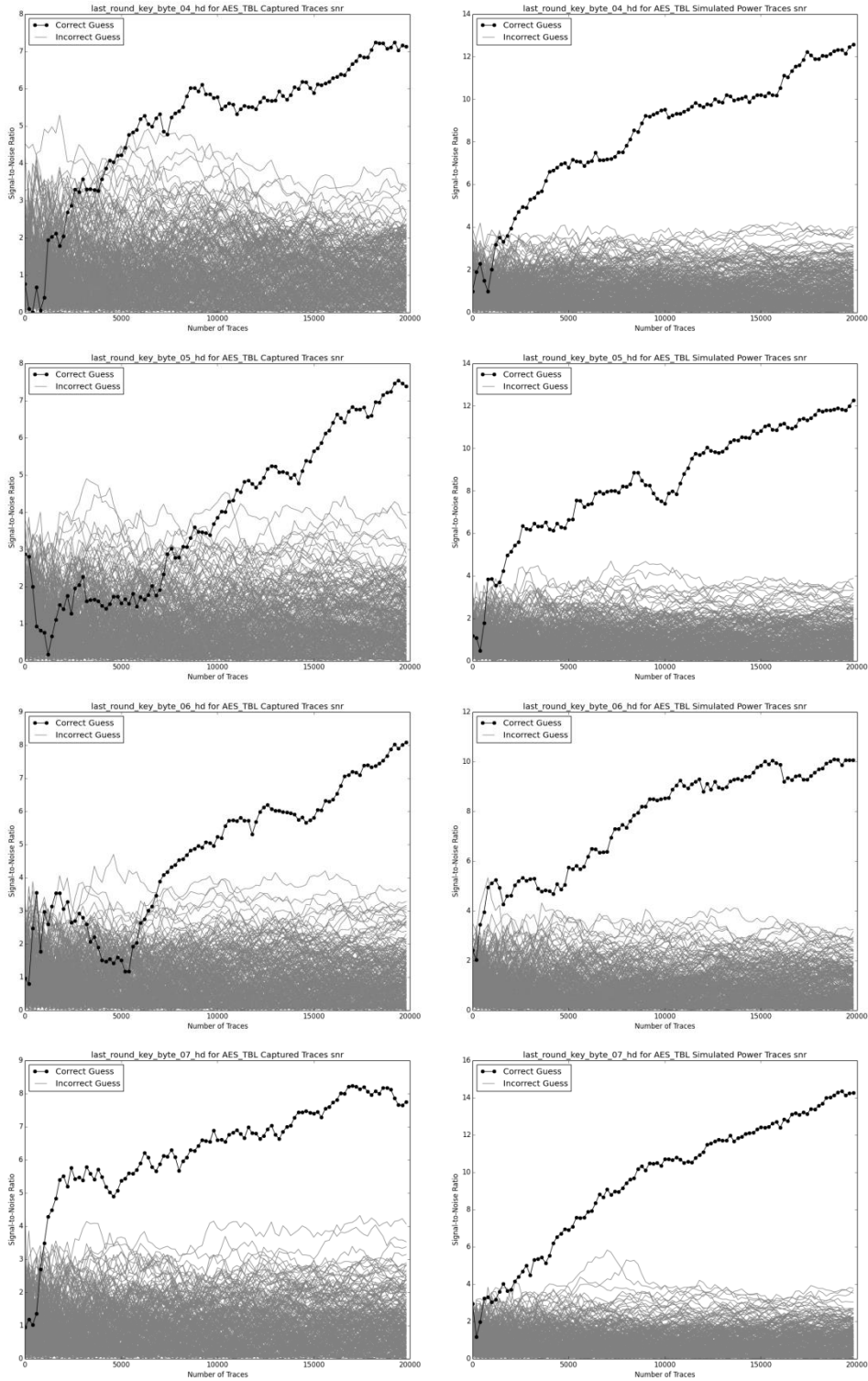
Correlation Ratios for AES TBL Measured (R) and Simulated (L), key bytes 8-11 with 20k traces



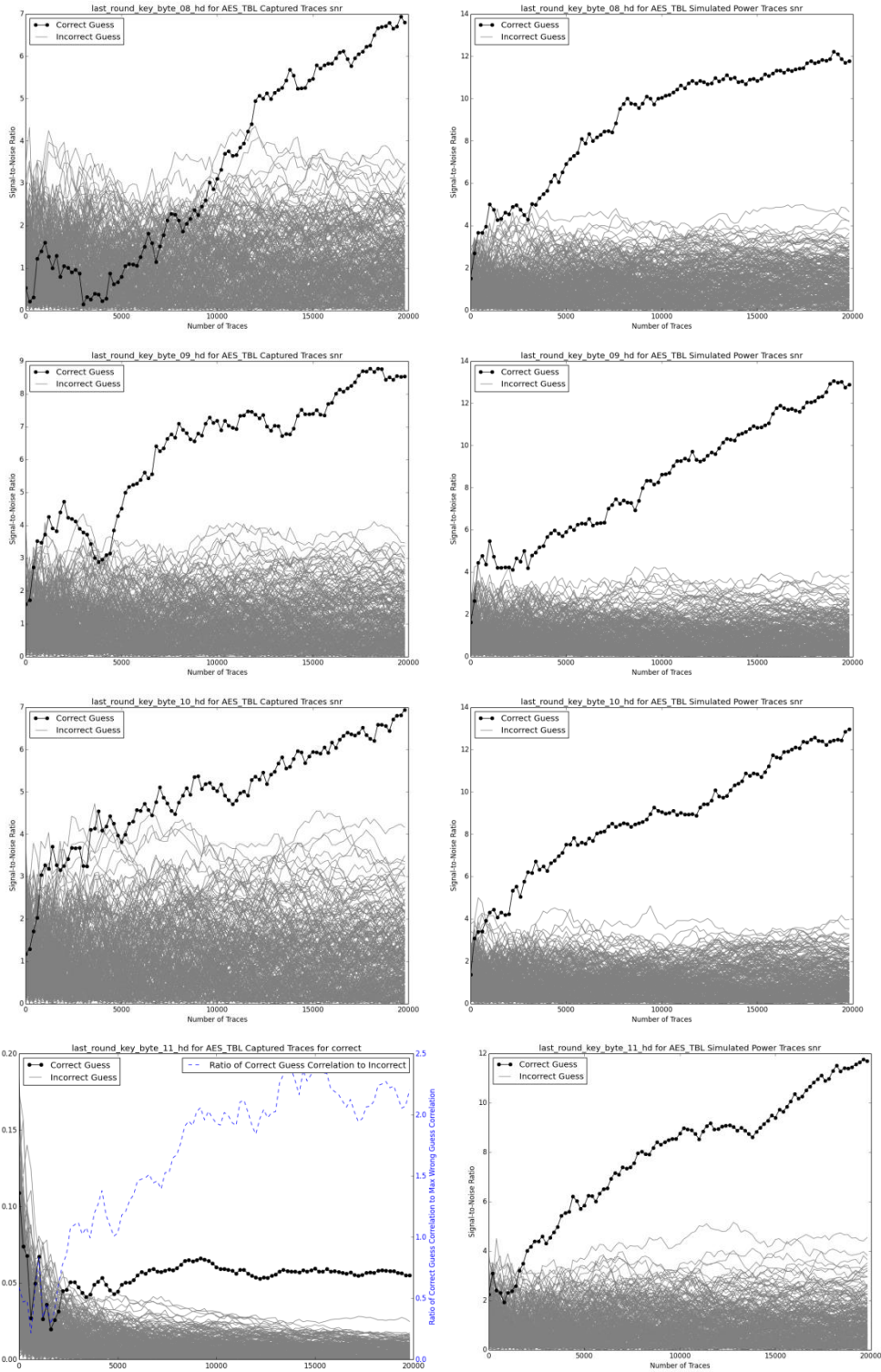
Correlation Ratios for AES TBL Measured (R) and Simulated (L), key bytes 12-15 with 20k traces



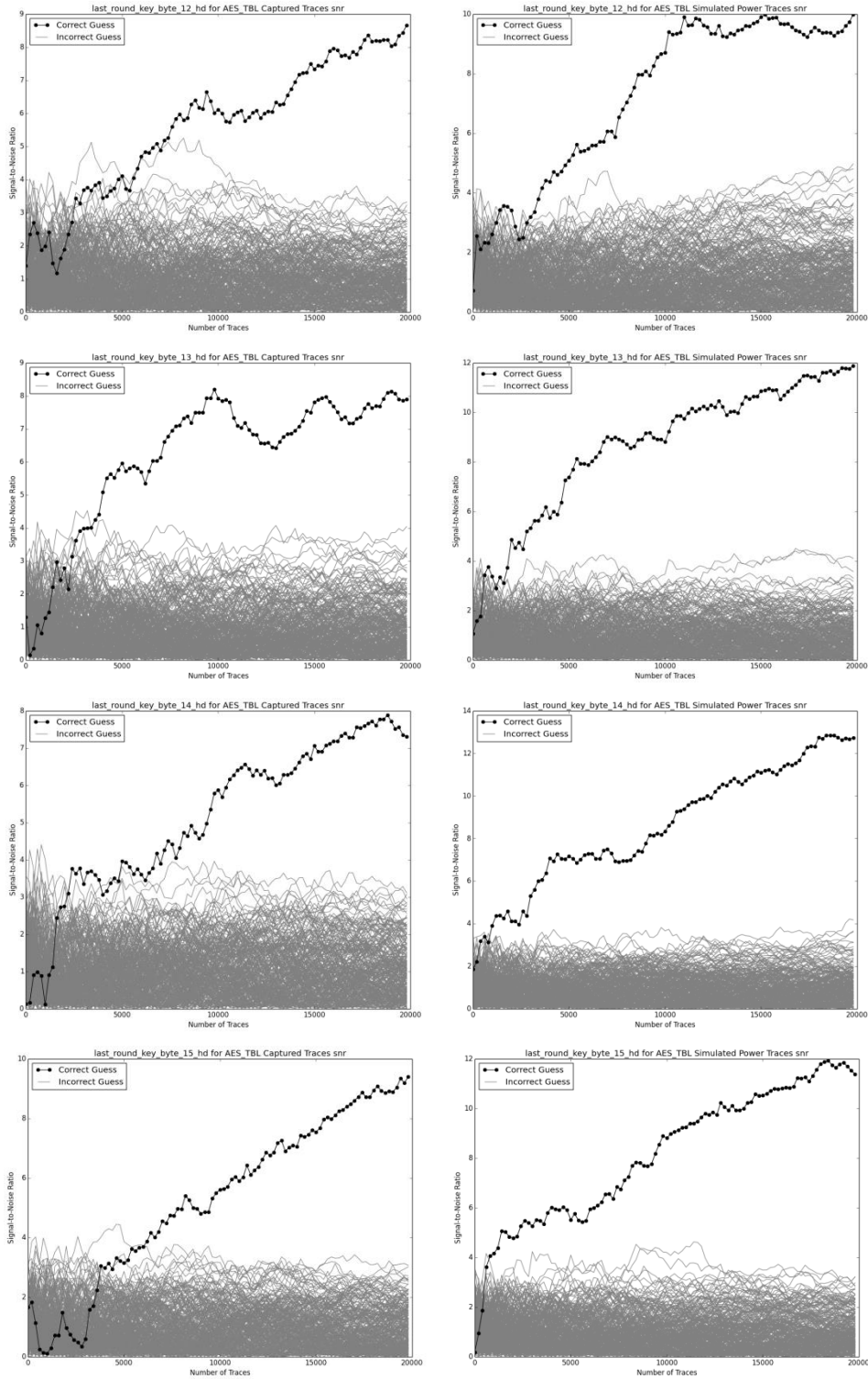
Signal-to-Noise Ratios for AES TBL Measured (R) and Simulated (L), key bytes 0-3, all guesses



Signal-to-Noise Ratios for AES TBL Measured (R) and Simulated (L), key bytes 4-7, all guesses

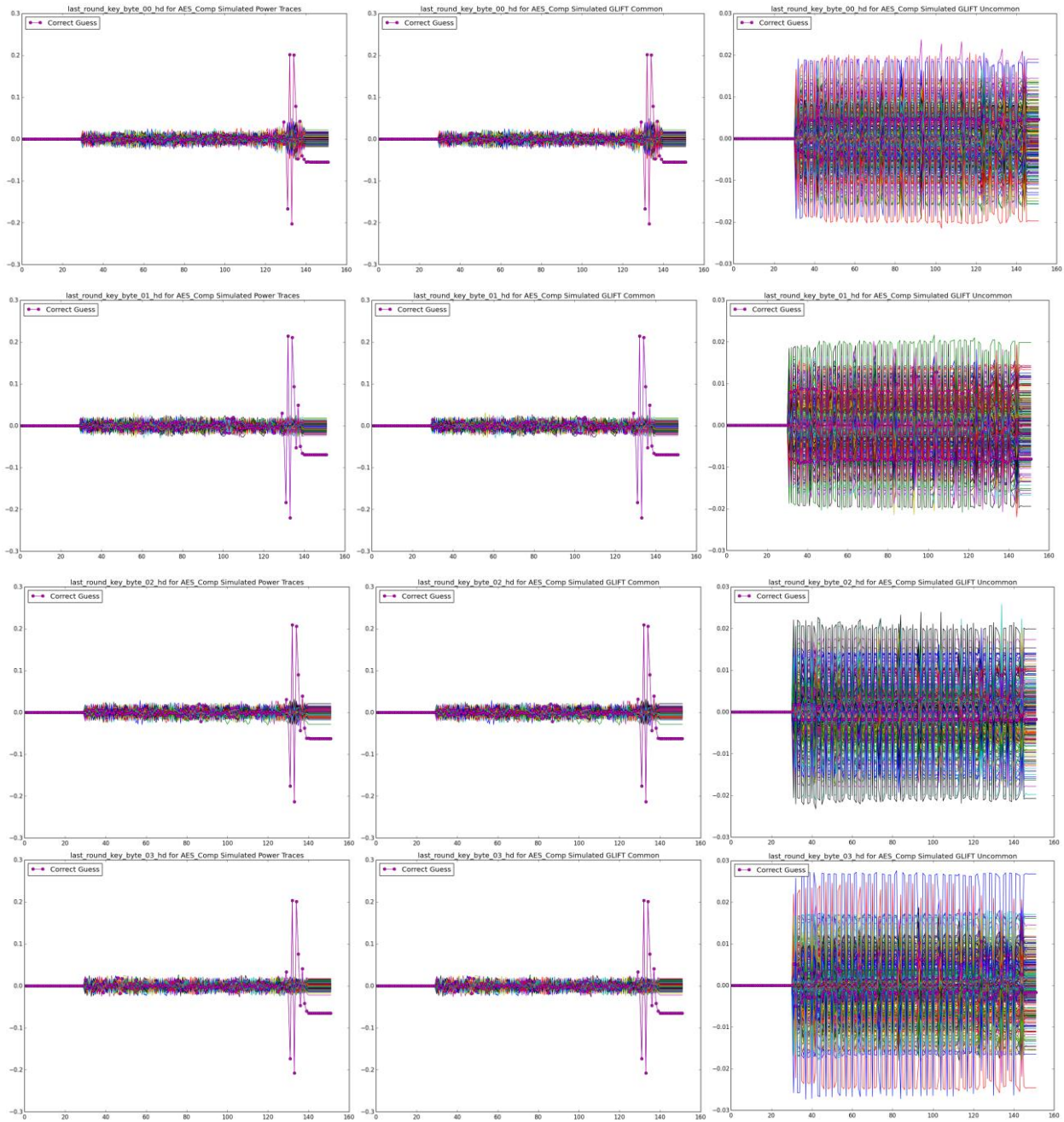


Signal-to-Noise Ratios for AES TBL Measured (R) and Simulated (L), key bytes 8-11, all guesses

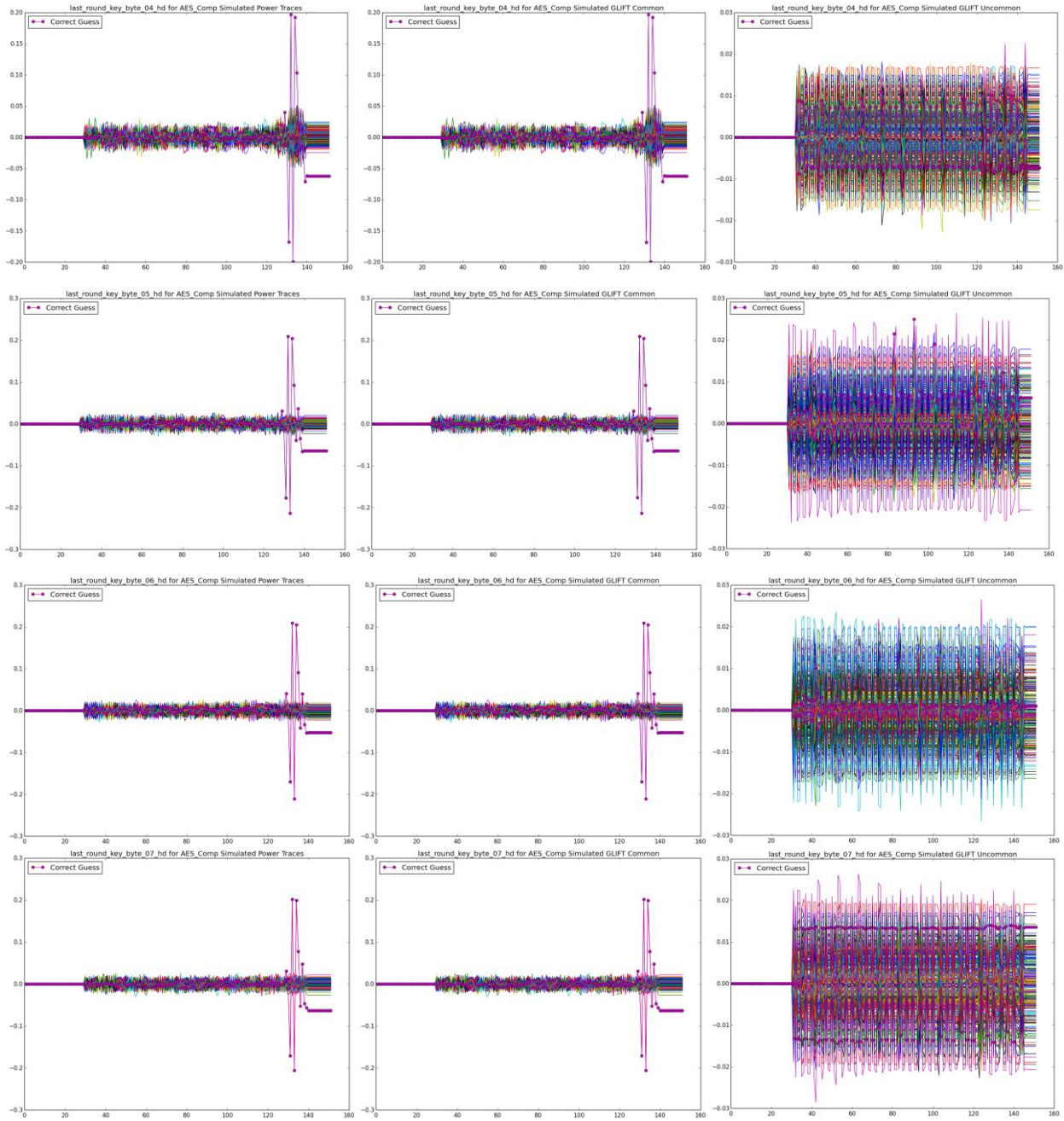


Signal-to-Noise Ratios for AES TBL Measured (R) and Simulated (L), key bytes 12-15, all guesses

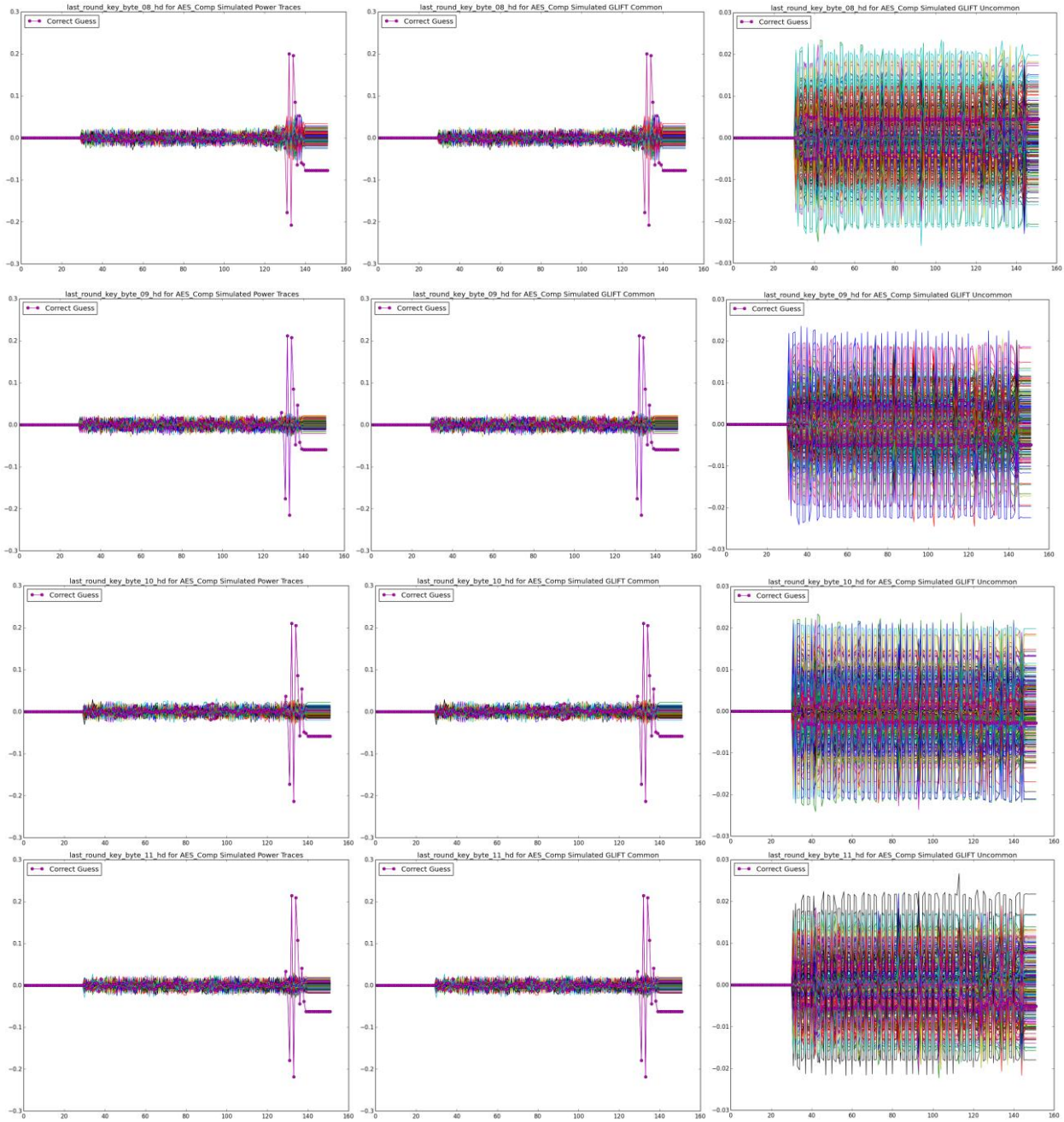
9.6 Appendix D: DPA Results: Full AES Comp vs. GLIFT Results



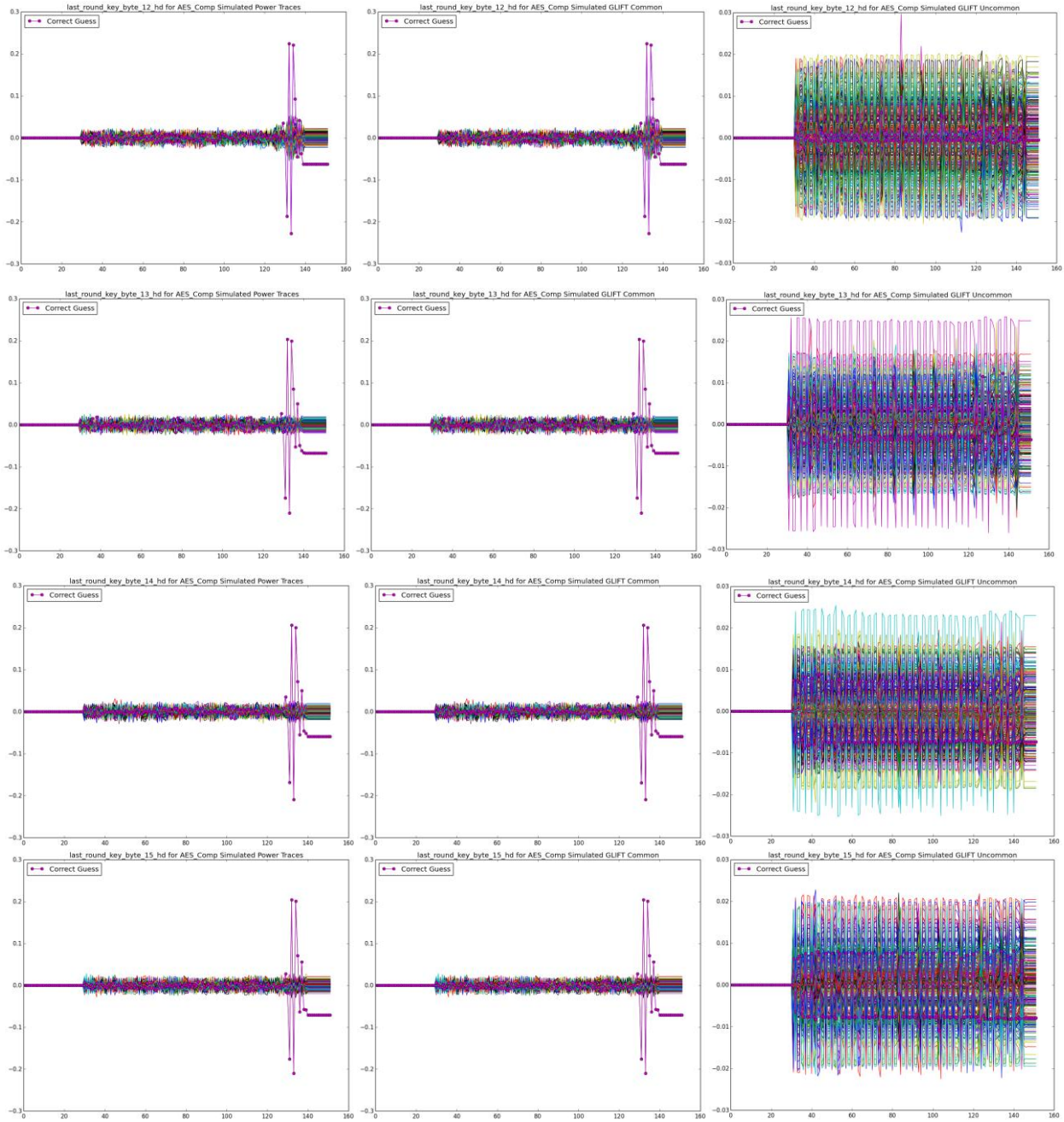
DPA Results for Original Simulated AES Comp (Left), Cells Selected by GLIFT (Center), and Cells Excluded by GLIFT (Right) key bytes 0-3 with 20k Traces, Zero Noise



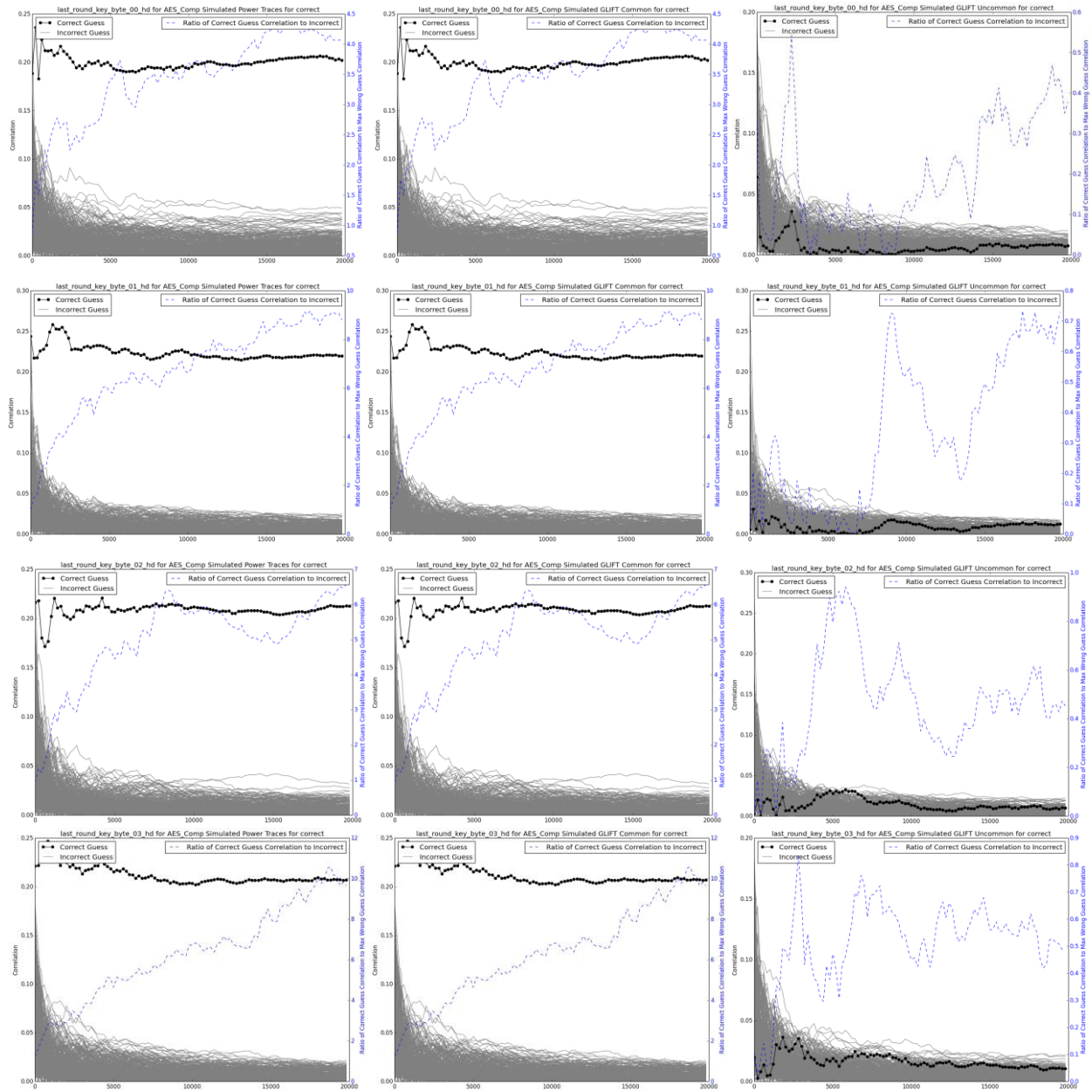
DPA Results for Original Simulated AES Comp (Left), Cells Selected by GLIFT (Center), and Cells Excluded by GLIFT (Right) key bytes 4-7 with 20k Traces, Zero Noise



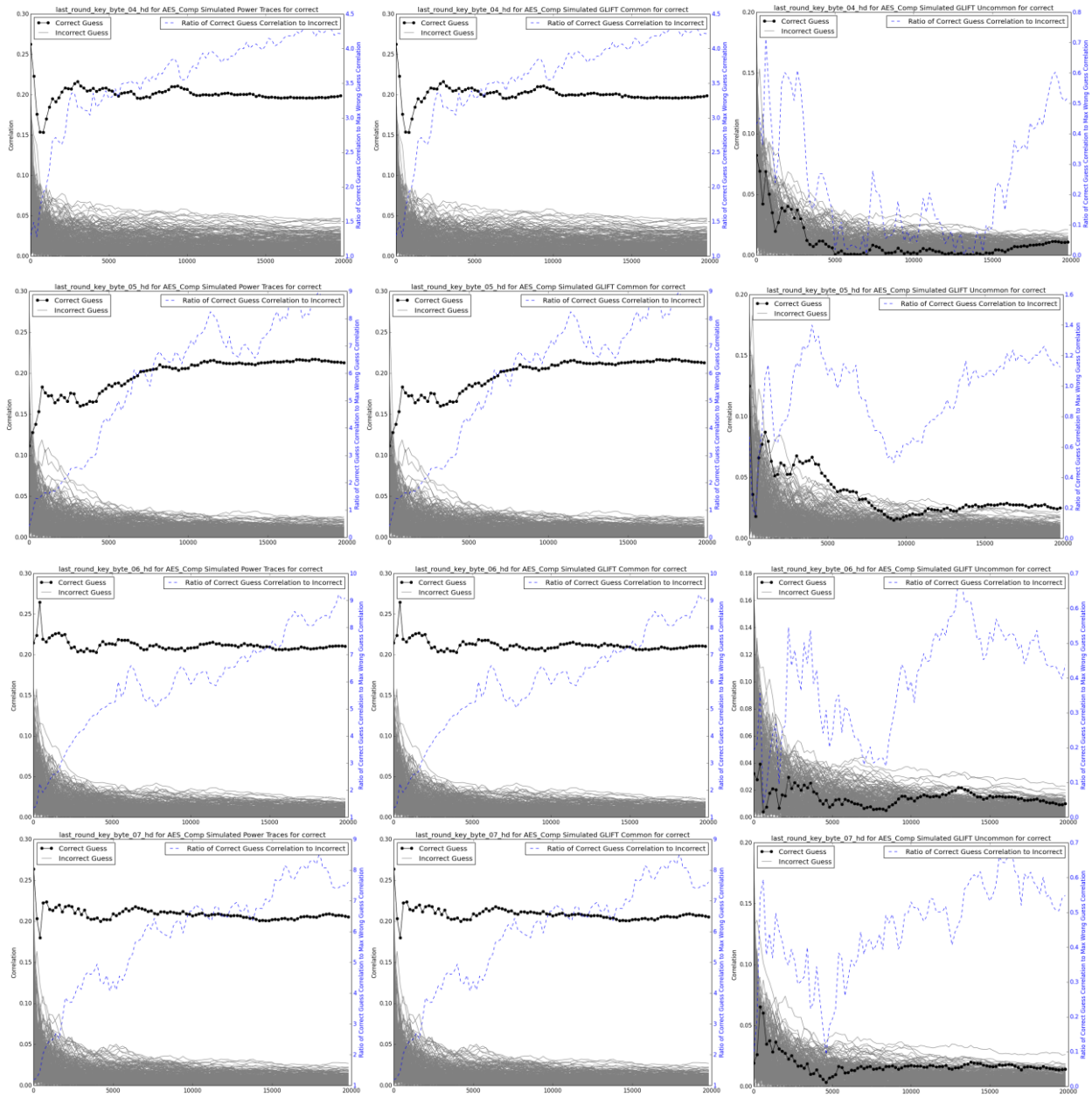
DPA Results for Original Simulated AES Comp (Left), Cells Selected by GLIFT (Center), and Cells Excluded by GLIFT (Right) key bytes 8-11 with 20k Traces, Zero Noise



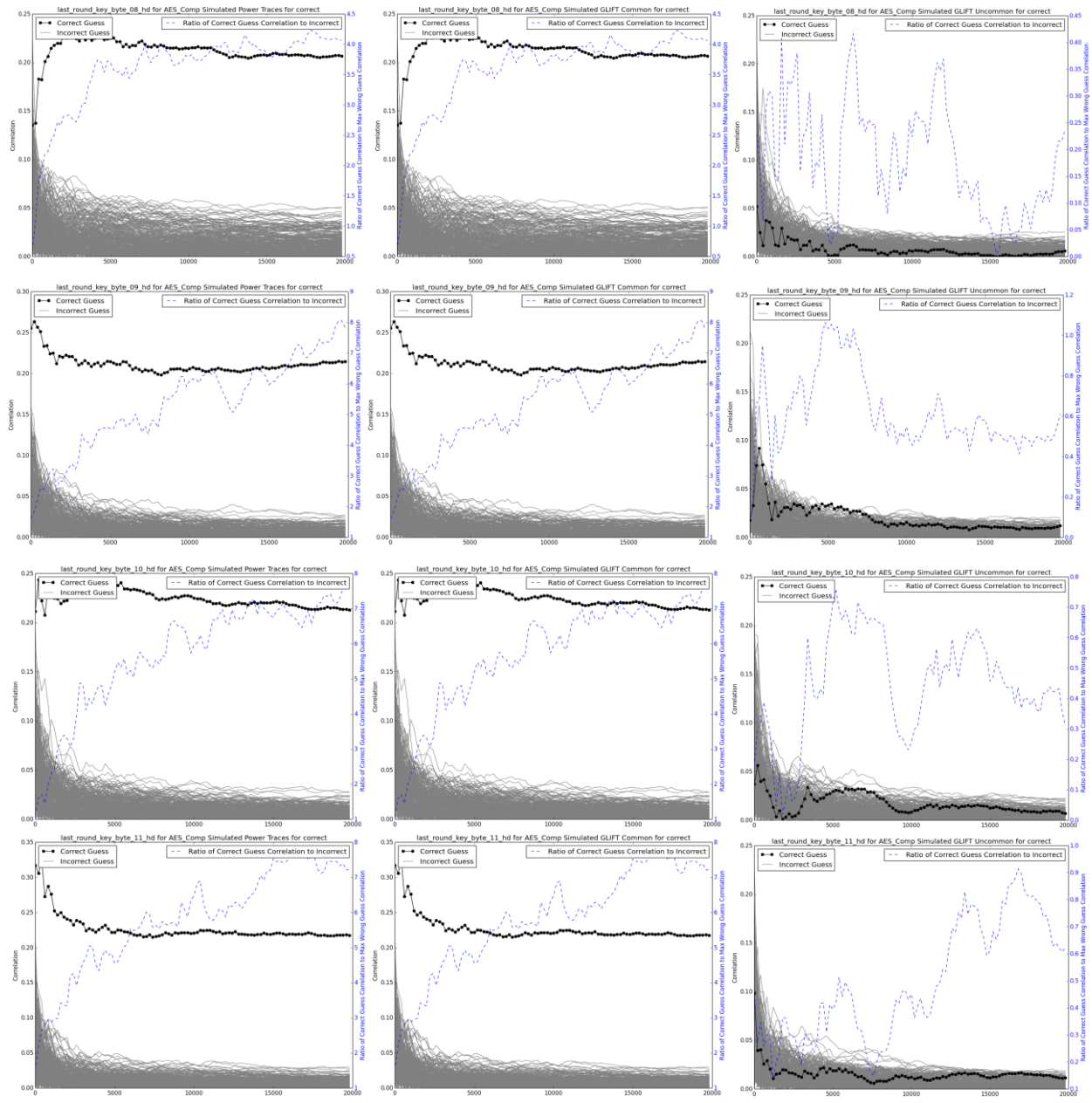
DPA Results for Original Simulated AES Comp (Left), Cells Selected by GLIFT (Center), and Cells Excluded by GLIFT (Right) key bytes 12-15 with 20k Traces, Zero Noise



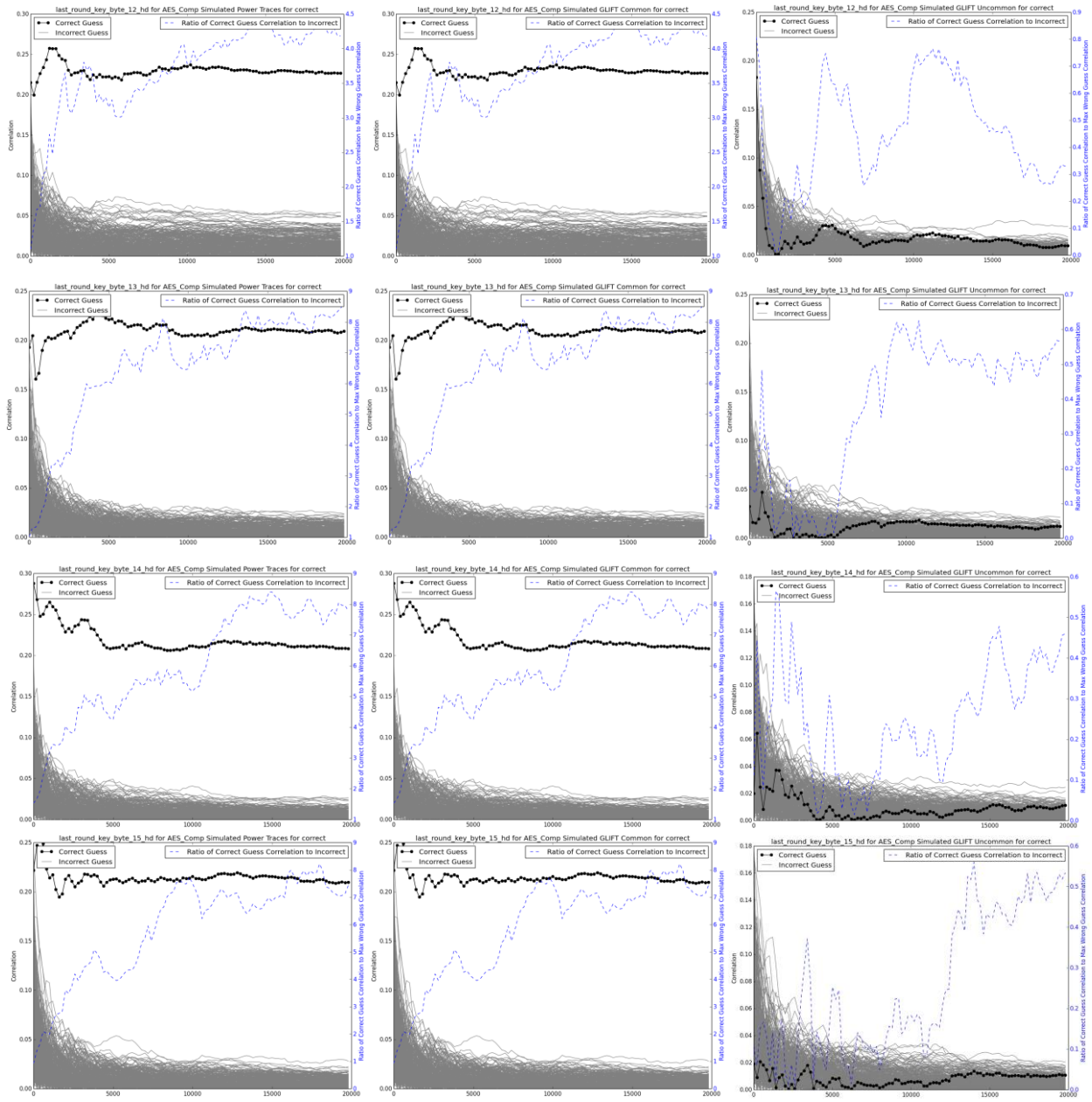
Correlation Ratio for Guesses: Simulated AES Comp (Left), Cells Selected by GLIFT (Center), and Cells Excluded by GLIFT (Right) key bytes 0-3 with 20k Traces, Zero Noise



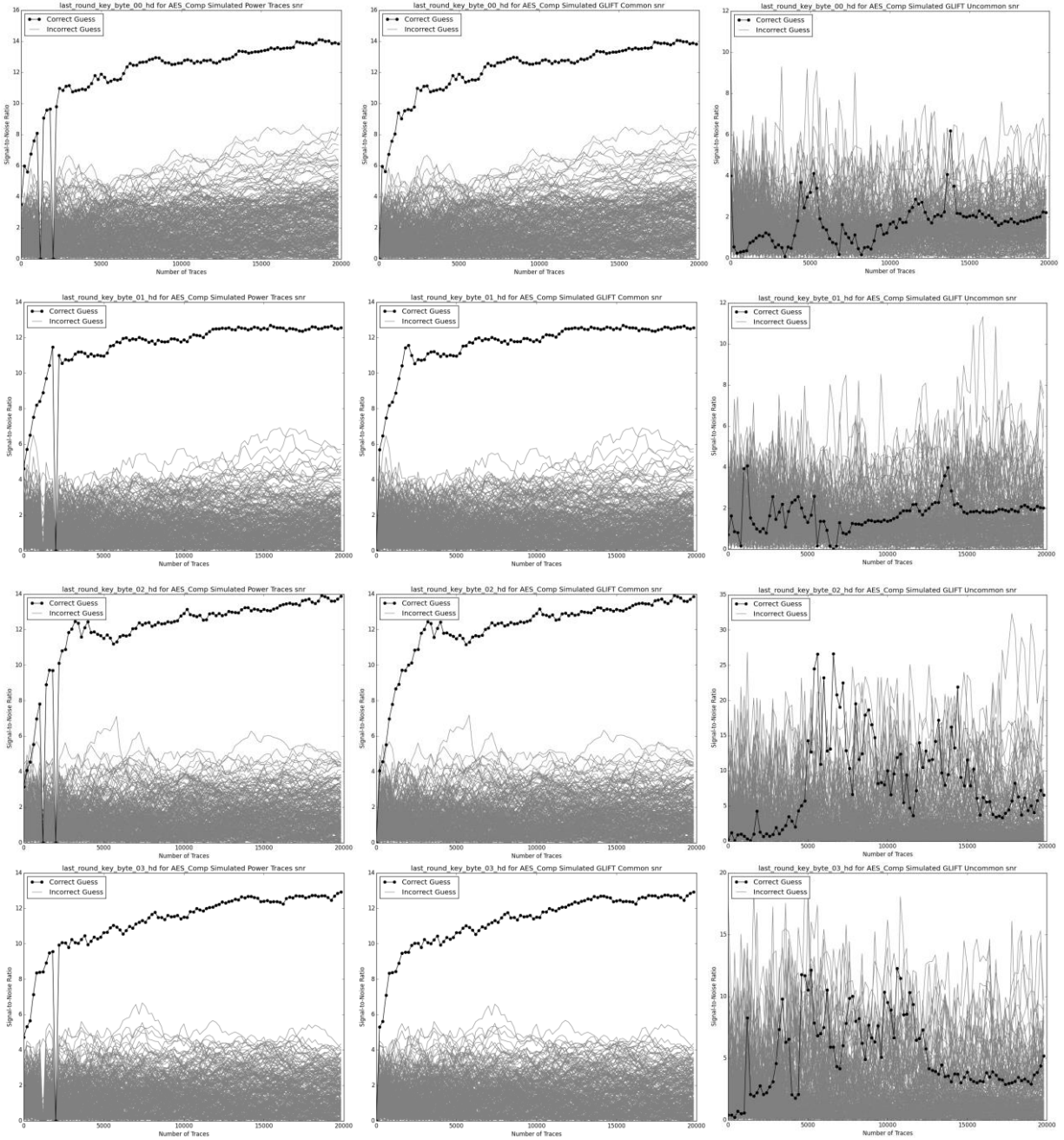
Correlation Ratio for Guesses: Simulated AES Comp (Left), Cells Selected by GLIFT (Center), and Cells Excluded by GLIFT (Right) key bytes 4-7 with 20k Traces, Zero Noise



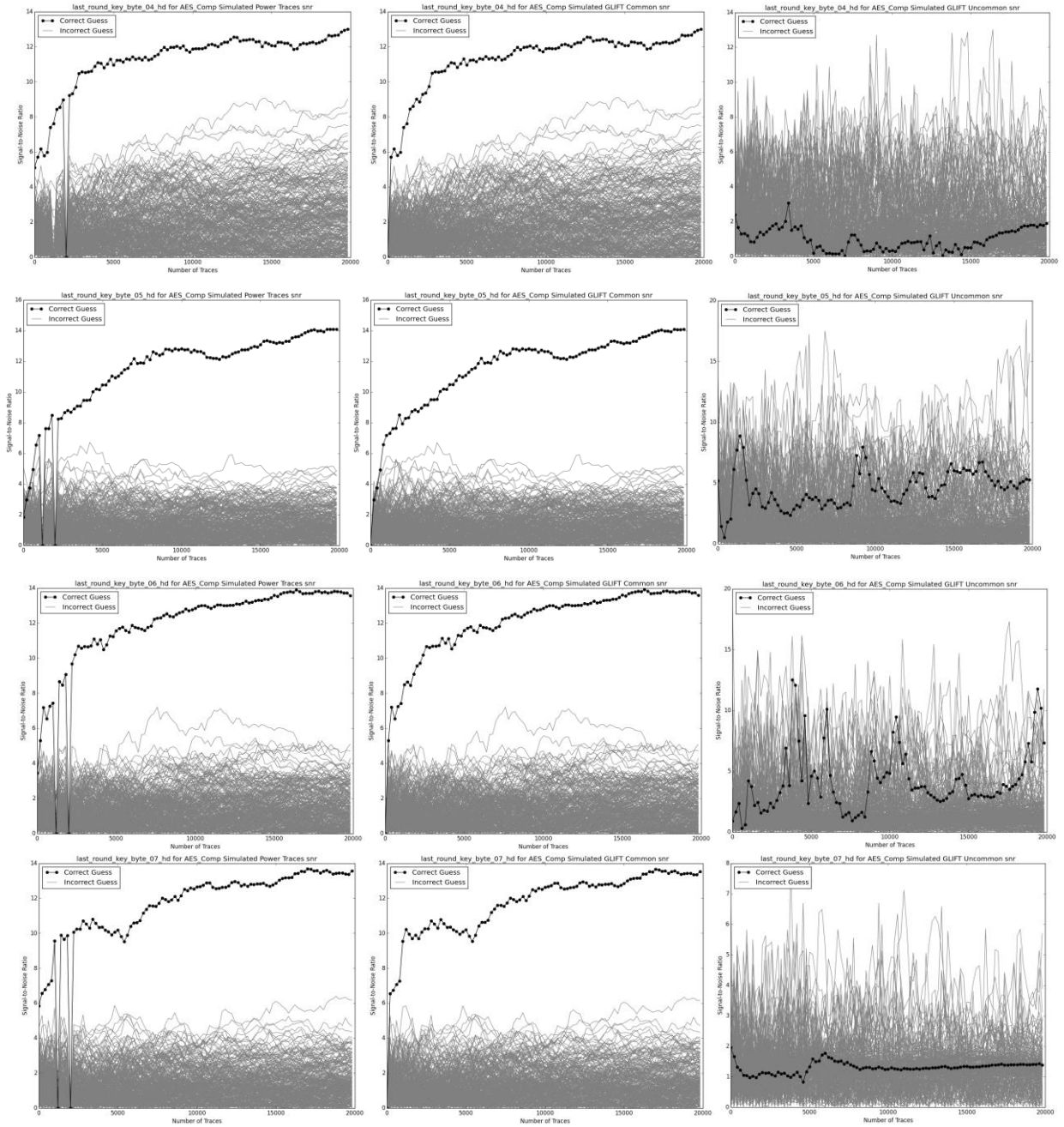
Correlation Ratio for Guesses: Simulated AES Comp (Left), Cells Selected by GLIFT (Center), and Cells Excluded by GLIFT (Right) key bytes 8-11 with 20k Traces, Zero Noise



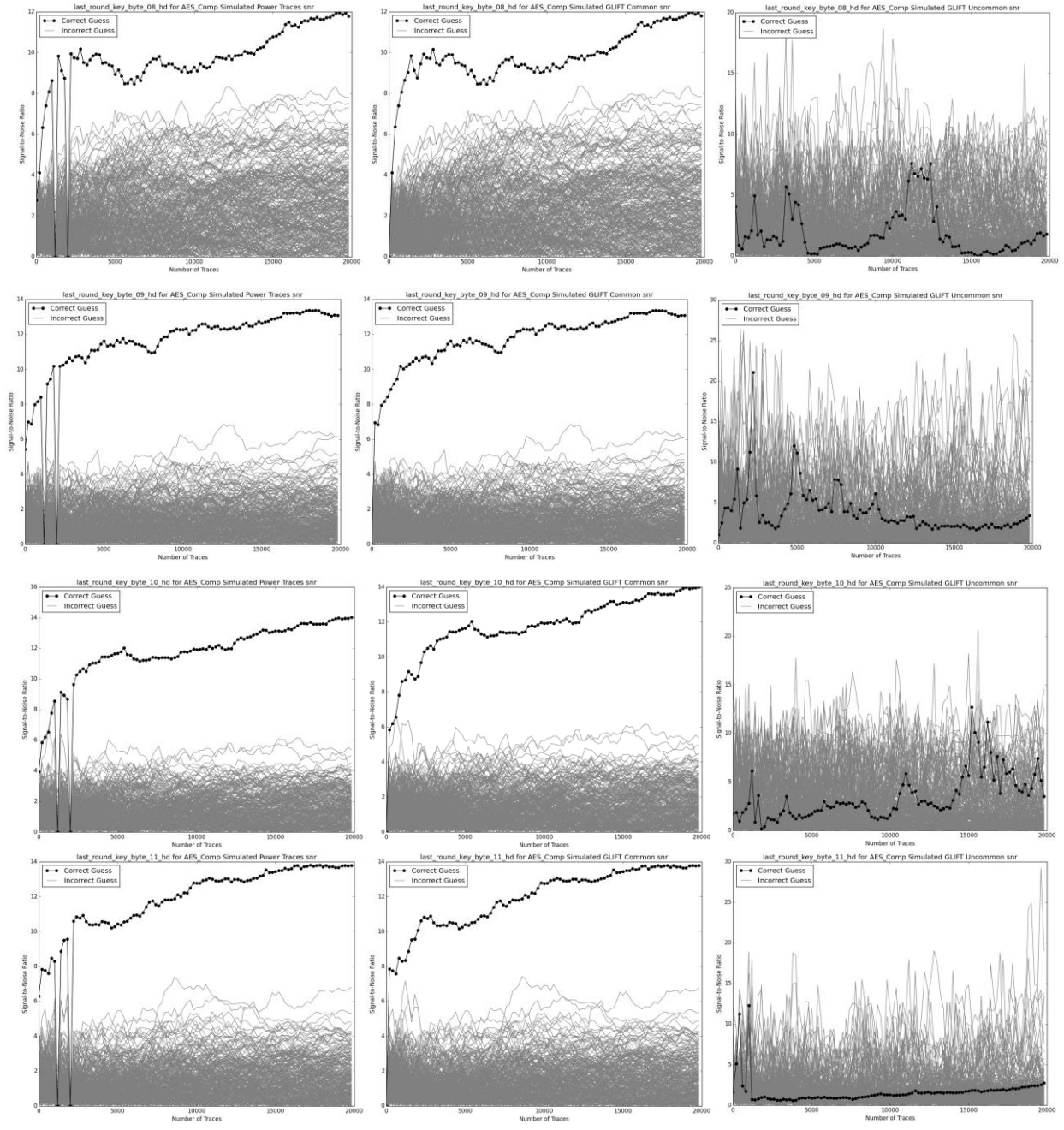
Correlation Ratio for Guesses: Simulated AES Comp (Left), Cells Selected by GLIFT (Center), and Cells Excluded by GLIFT (Right) key bytes 12-15 with 20k Traces, Zero Noise



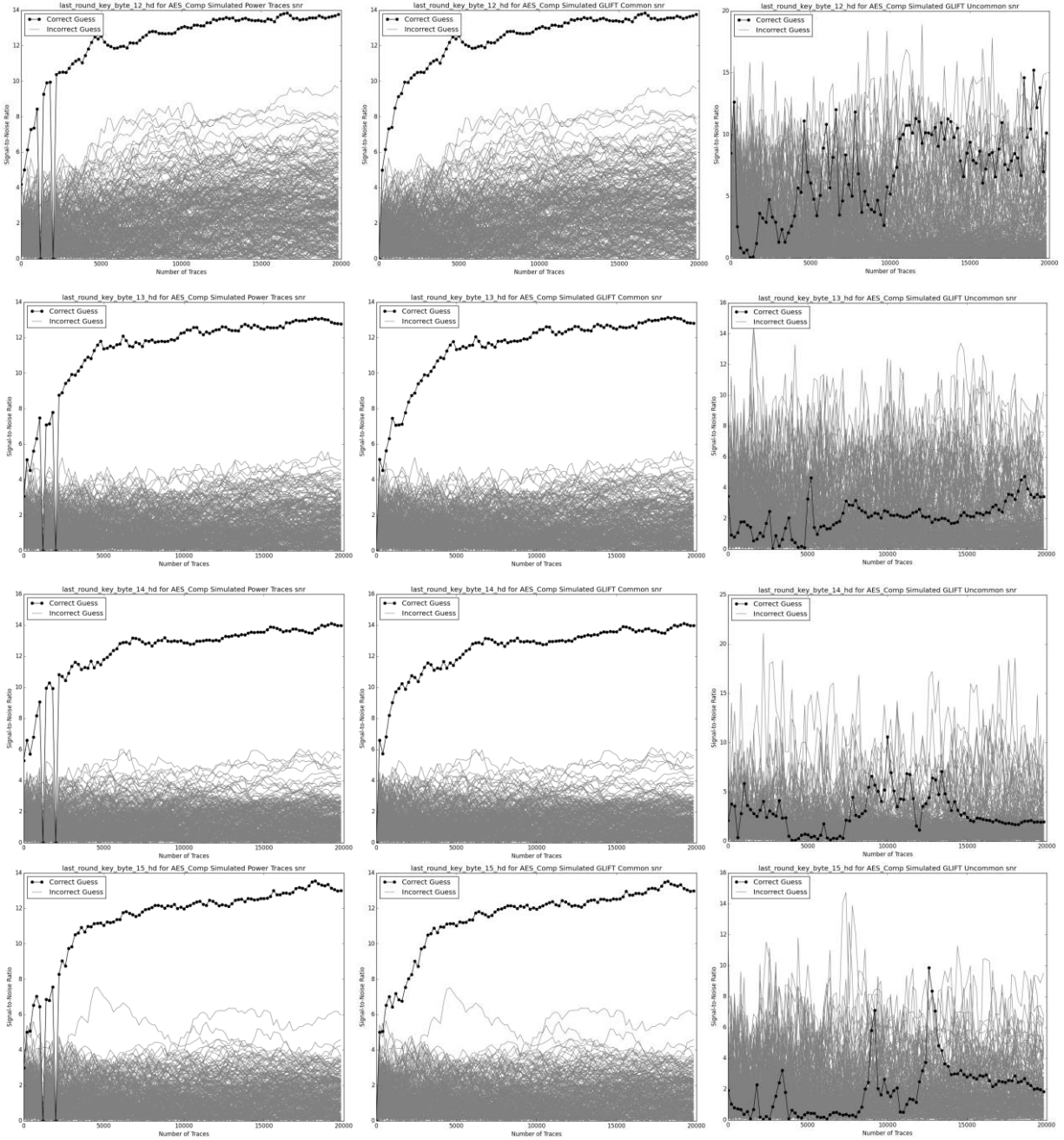
Signal-to-Noise Ratio: Simulated AES Comp (Left), Cells Selected by GLIFT (Center), and Cells Excluded by GLIFT (Right) key bytes 0-3 with 20k Traces, Zero Noise



Signal-to-Noise Ratio: Simulated AES Comp (Left), Cells Selected by GLIFT (Center), and Cells Excluded by GLIFT (Right) key bytes 4-7 with 20k Traces, Zero Noise

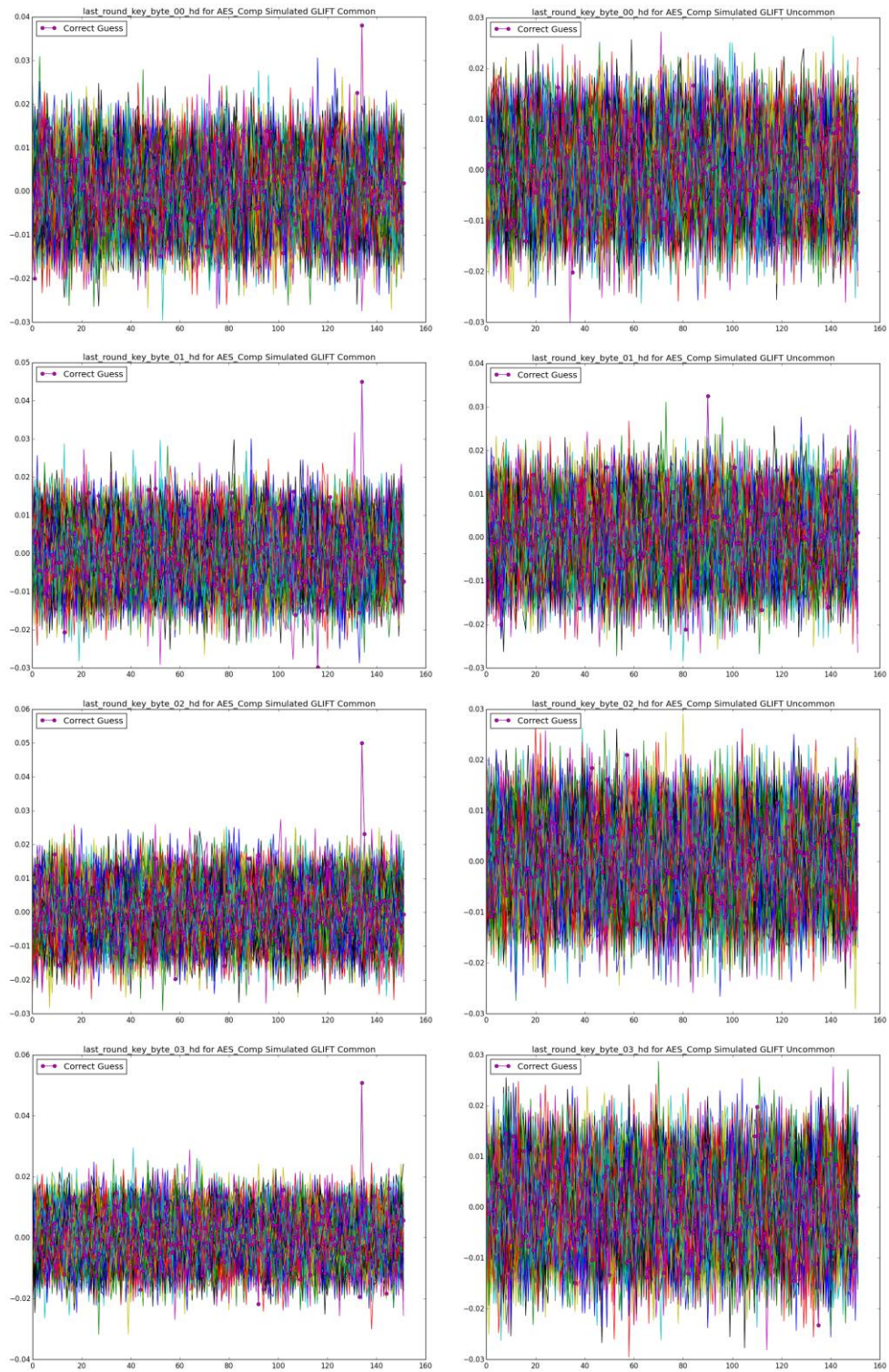


Signal-to-Noise Ratio: Simulated AES Comp (Left), Cells Selected by GLIFT (Center), and Cells Excluded by GLIFT (Right) key bytes 8-11 with 20k Traces, Zero Noise

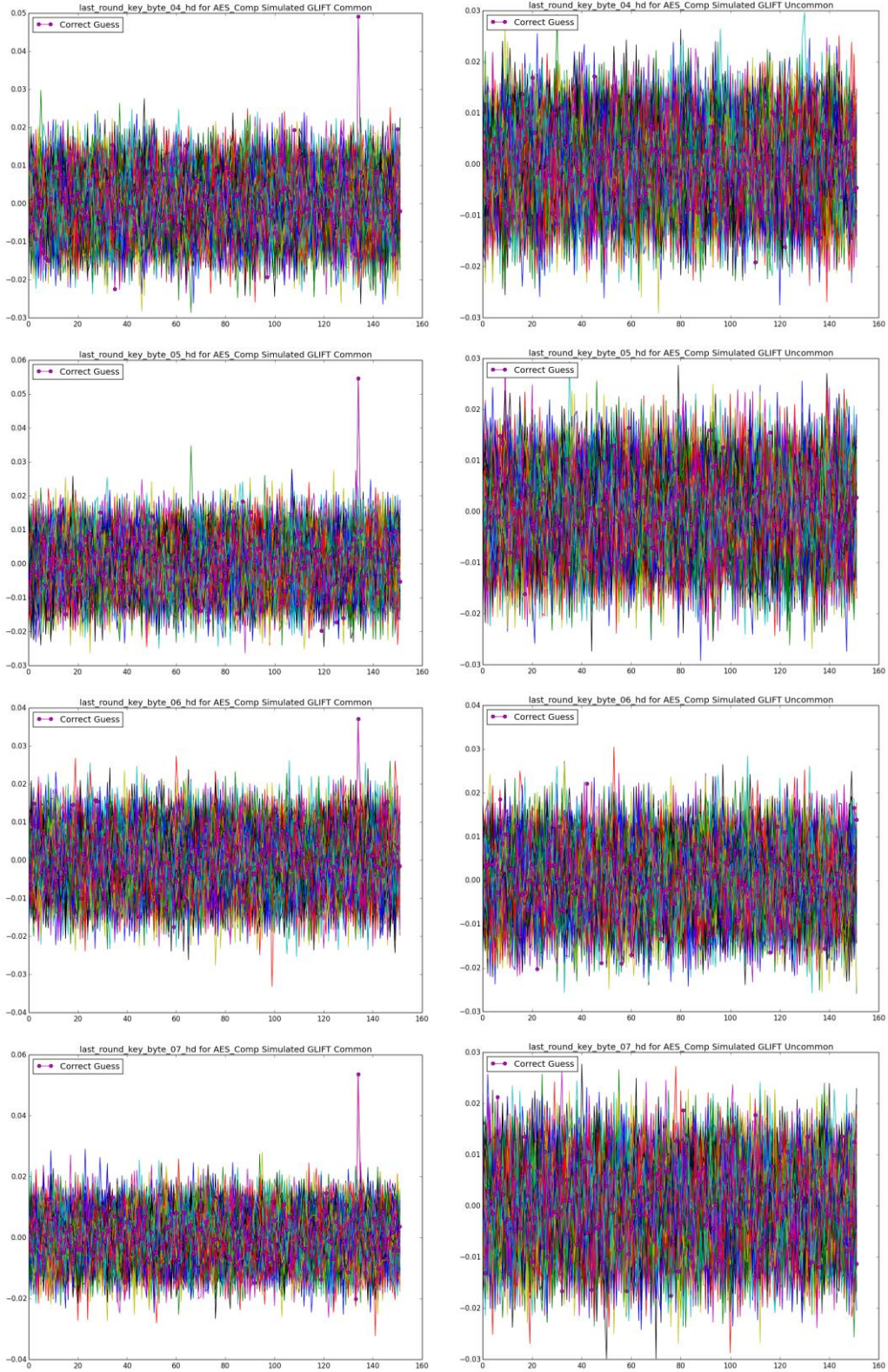


Signal-to-Noise Ratio: Simulated AES Comp (Left), Cells Selected by GLIFT (Center), and Cells Excluded by GLIFT (Right) key bytes 12-15 with 20k Traces, Zero Noise

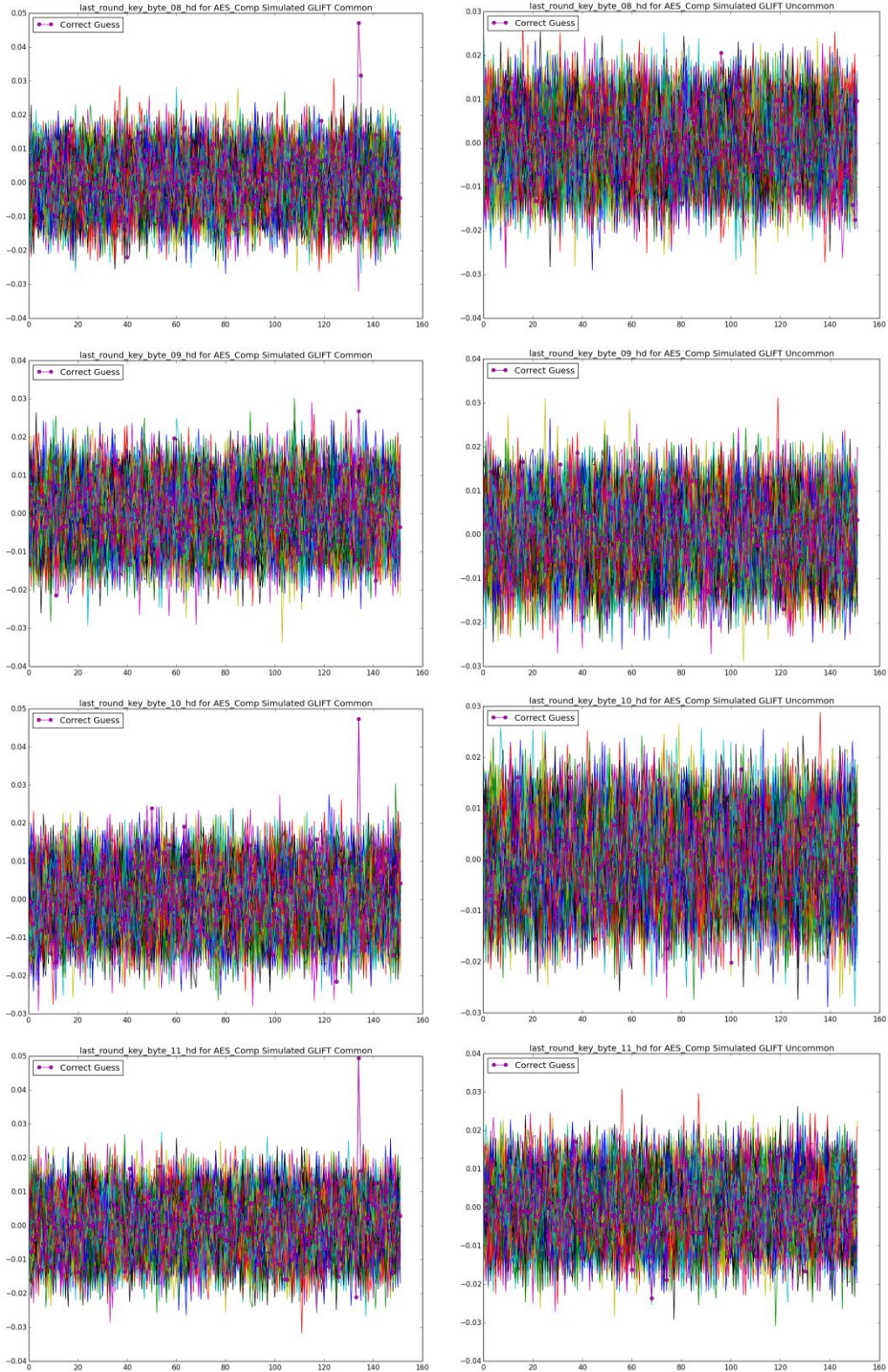
9.7 Appendix F: DPA Results: AES Comp GLIFT-Selected and Excluded Cells with Noise



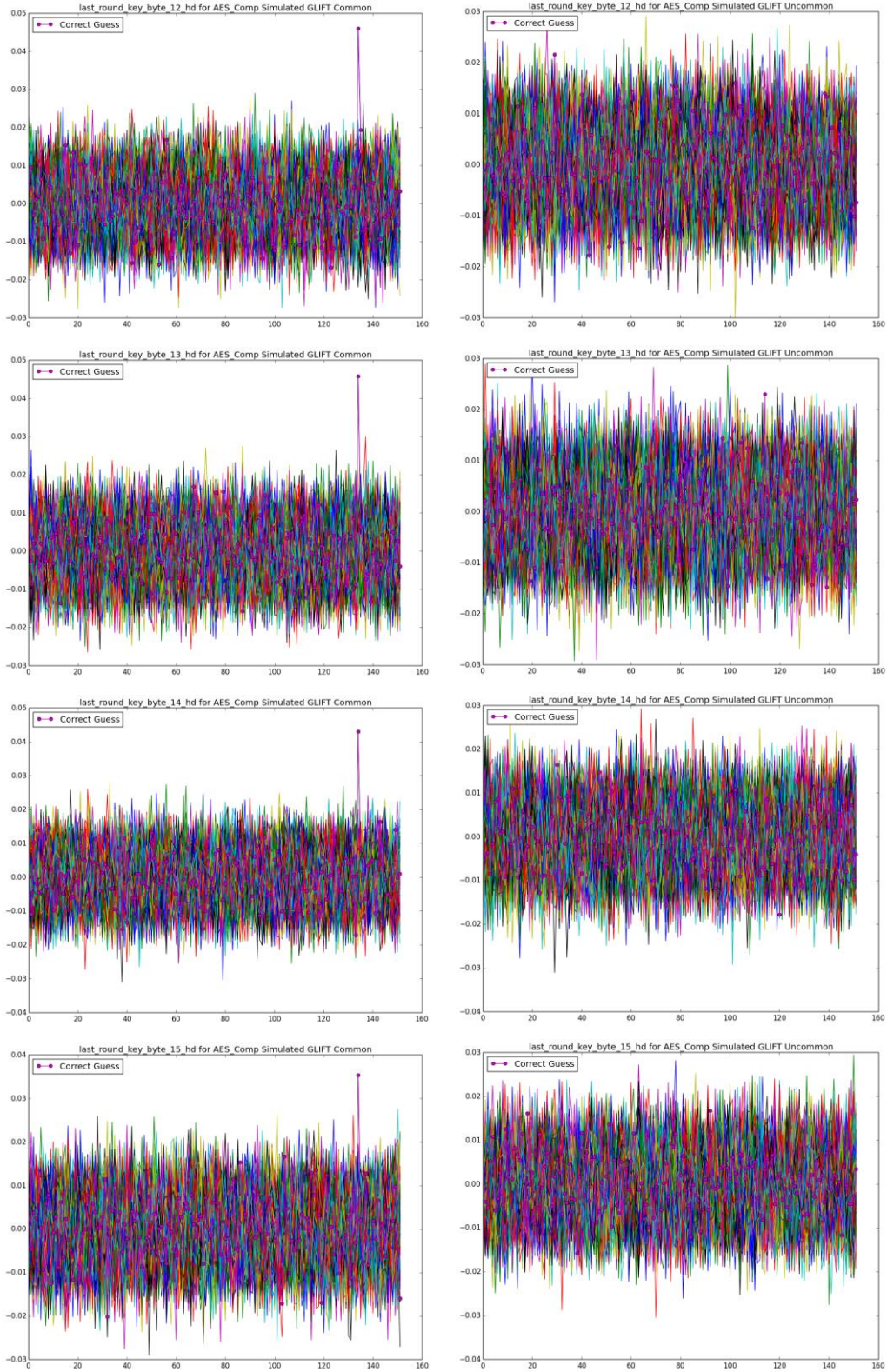
DPA Results for AES Comp Measured (R) and Simulated (L), key bytes 0-3 with 20k traces



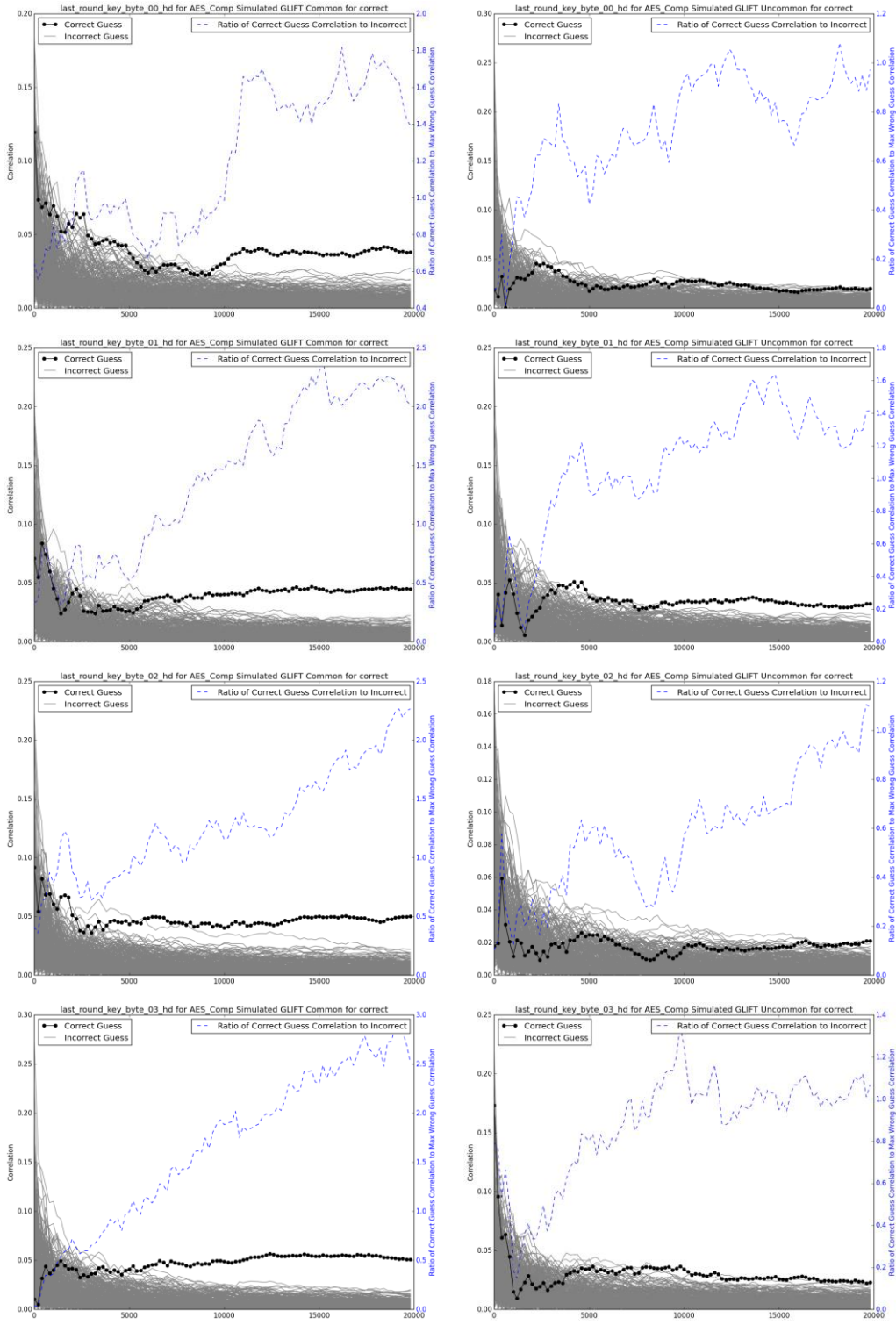
DPA Results for AES Comp Measured (R) and Simulated (L), key bytes 4-7 with 20k traces



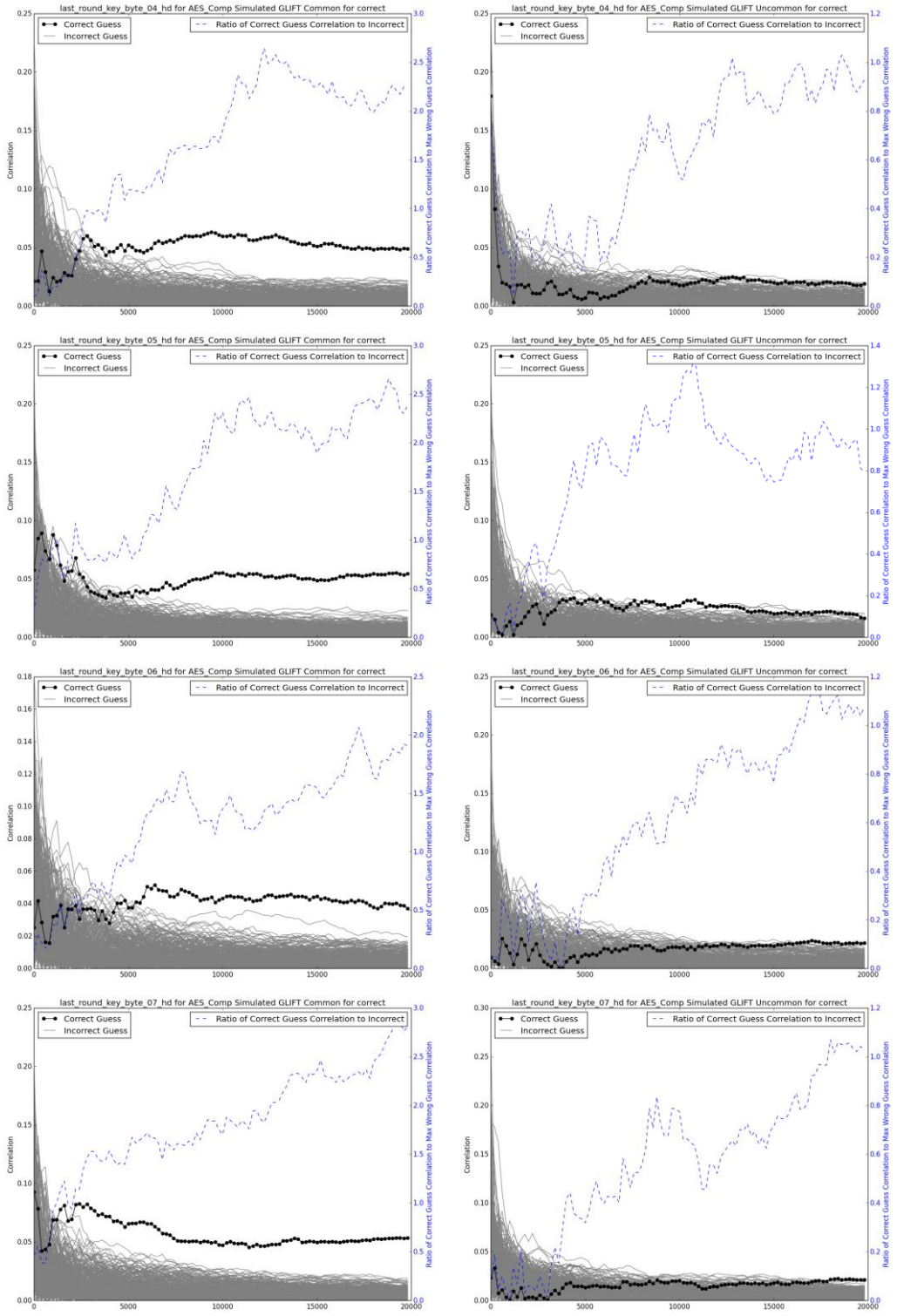
DPA Results for AES Comp Measured (R) and Simulated (L), key bytes 8-11 with 20k traces



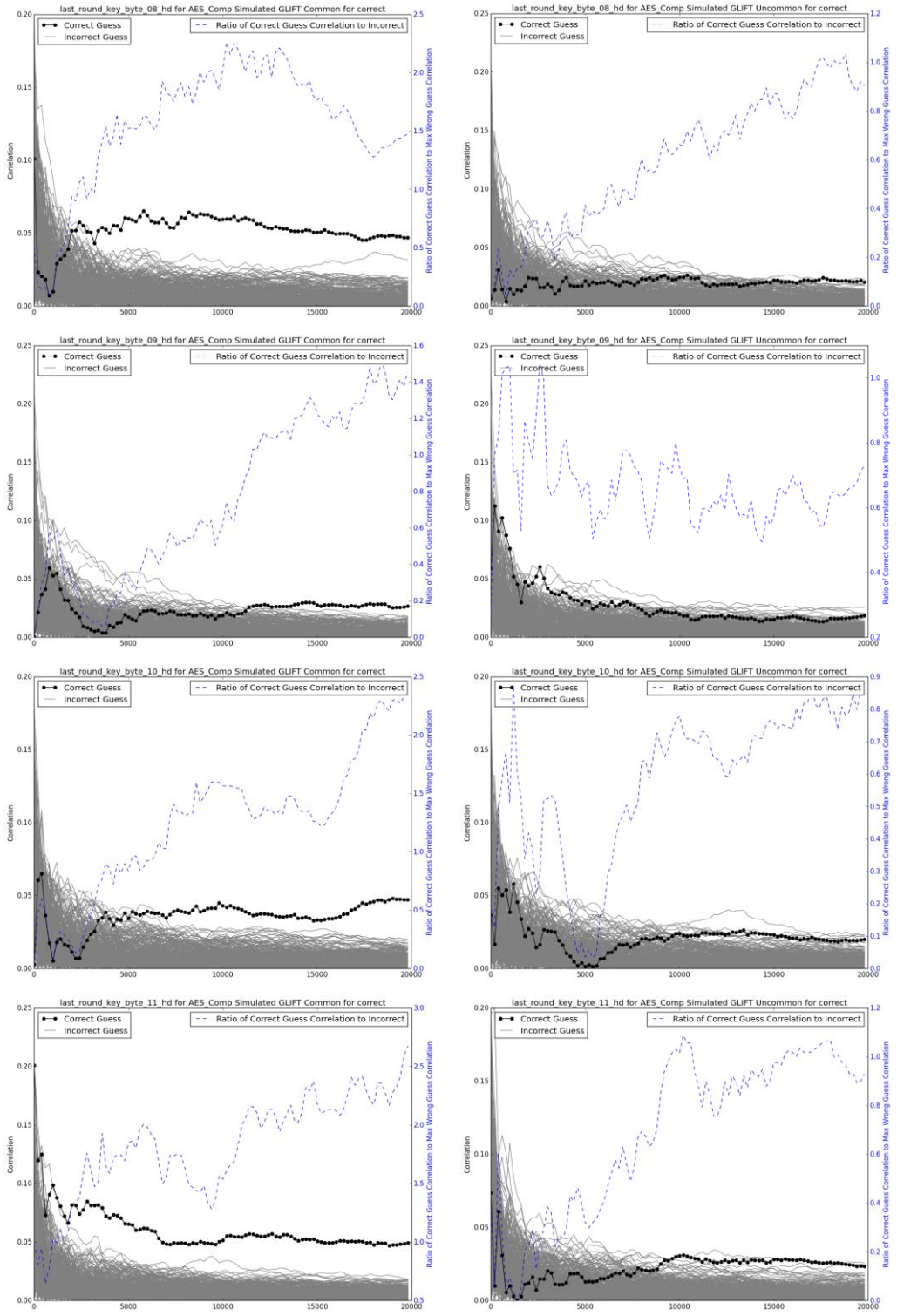
DPA Results for AES Comp Measured (R) and Simulated (L), key bytes 12-15 with 20k traces



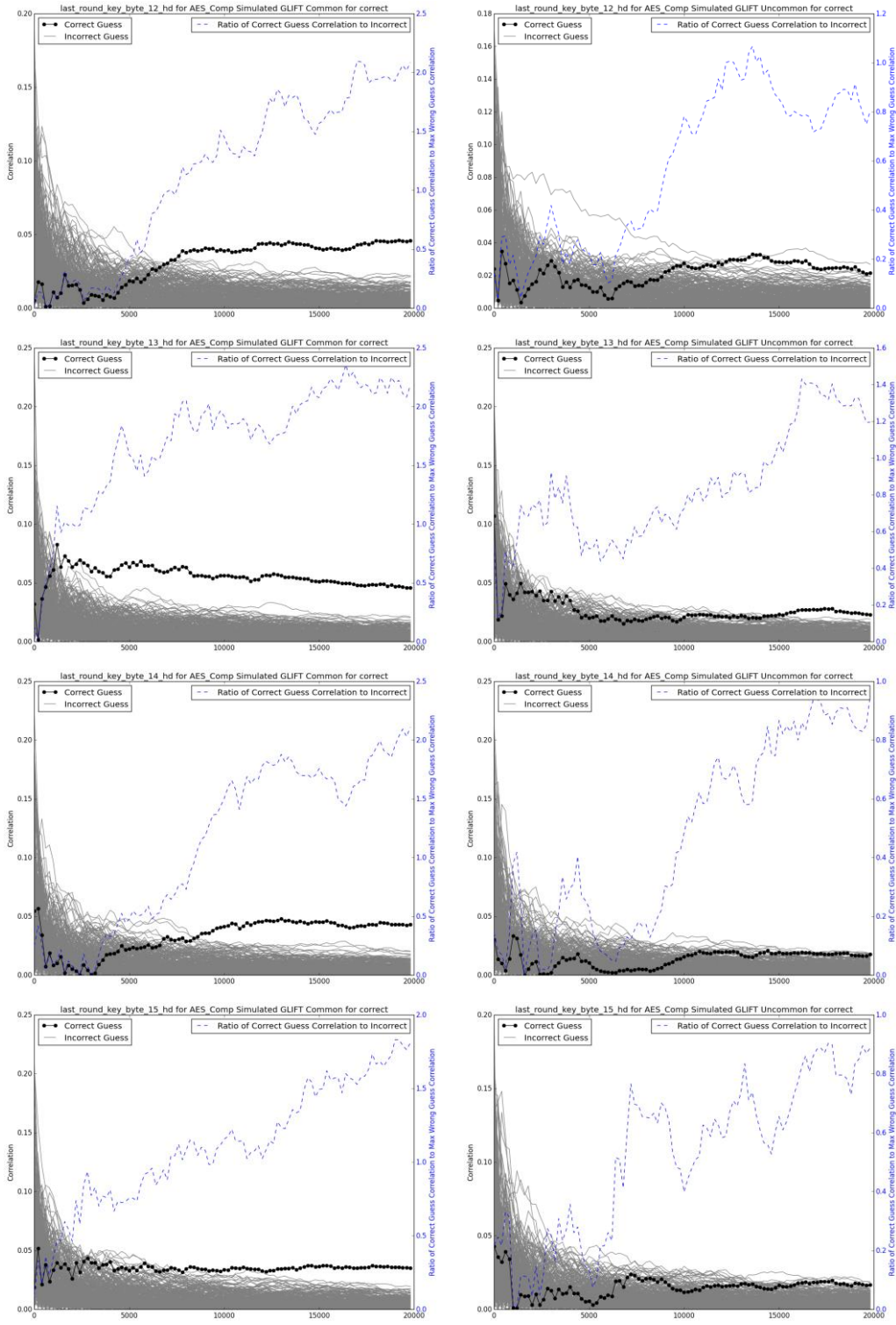
Correlation Ratios for AES Comp Measured (R) and Simulated (L), key bytes 0-3 with 20k traces



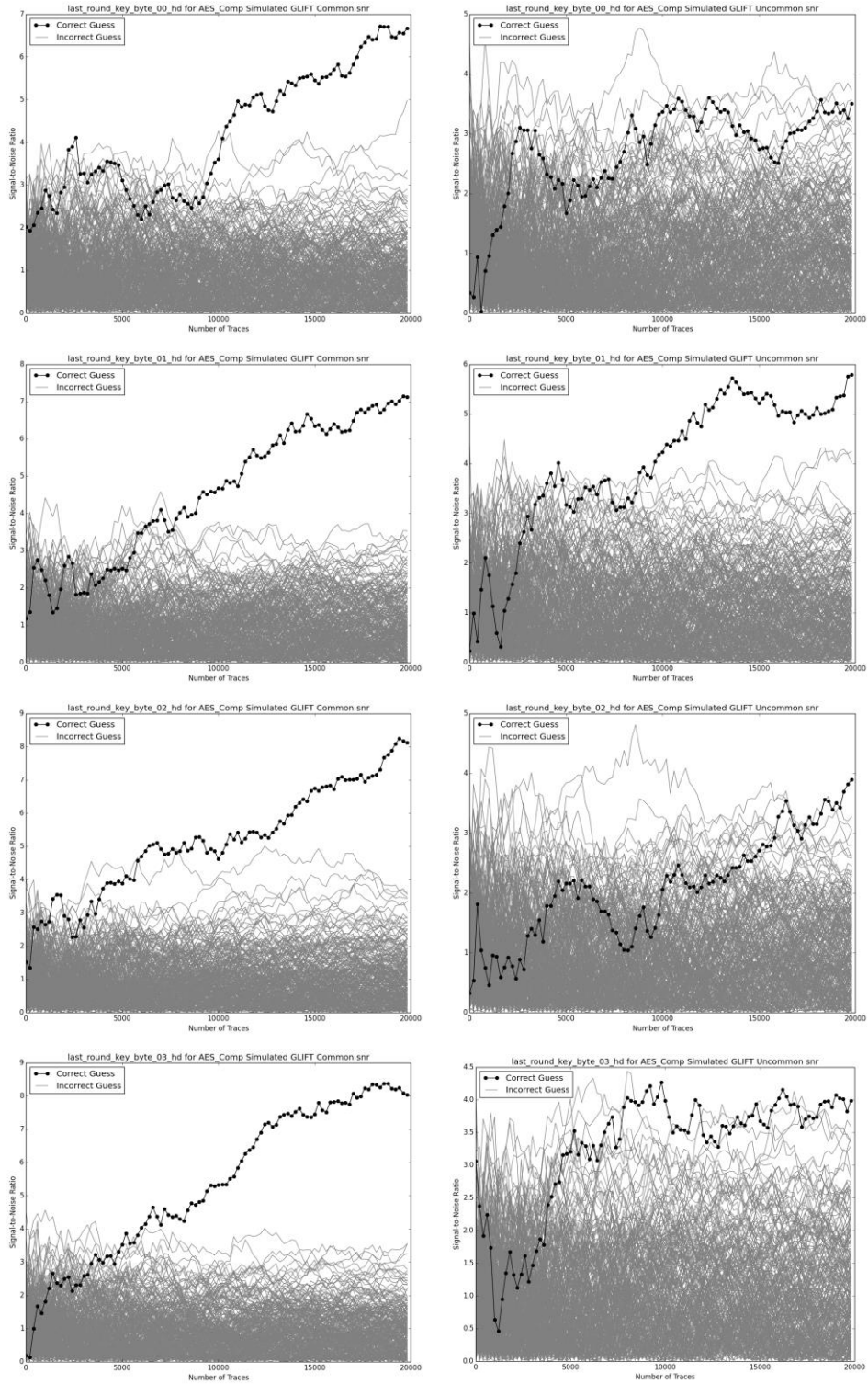
Correlation Ratios for AES Comp Measured (R) and Simulated (L), key bytes 4-7 with 20k traces



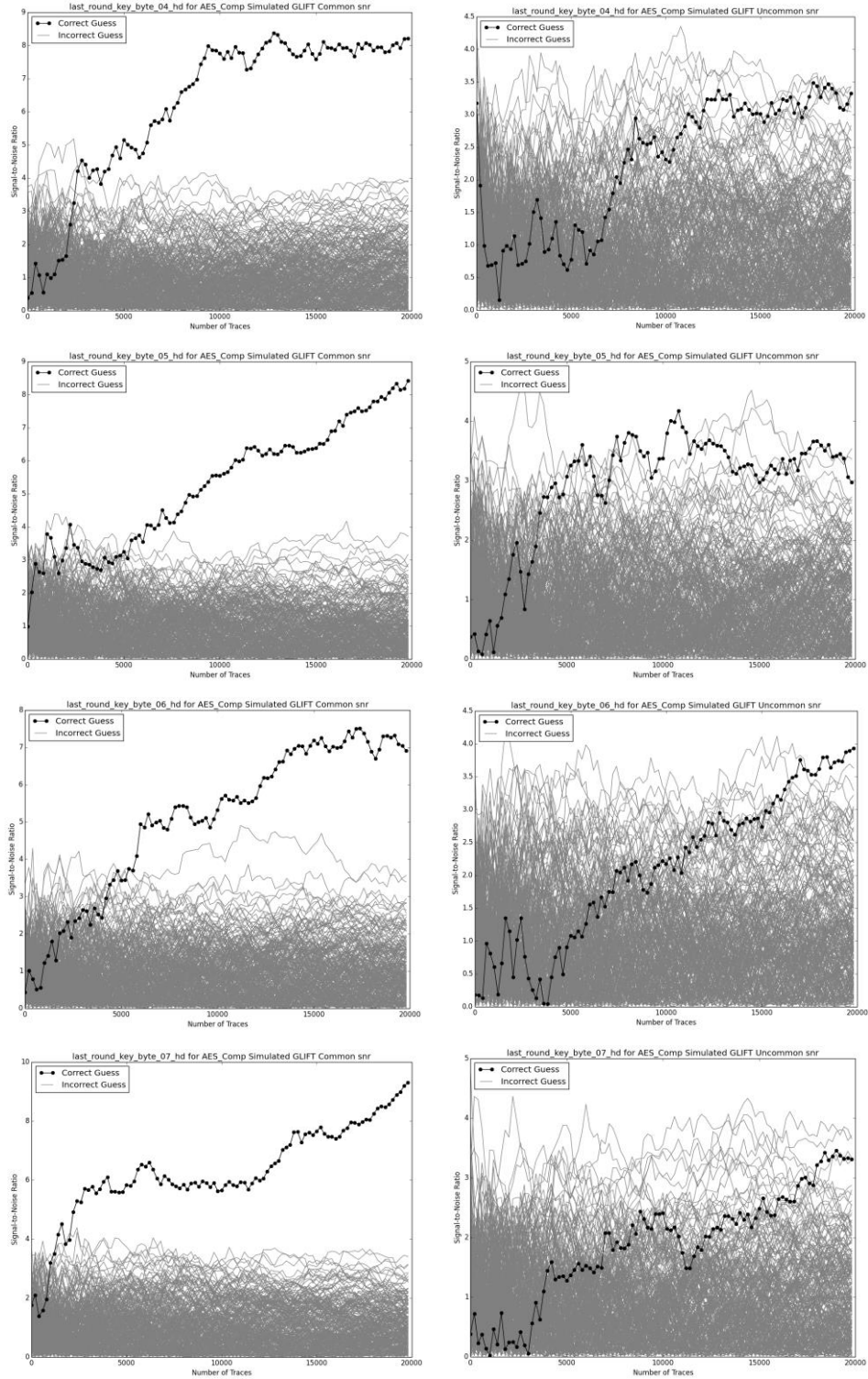
Correlation Ratios for AES Comp Measured (R) and Simulated (L), key bytes 8-11 with 20k traces



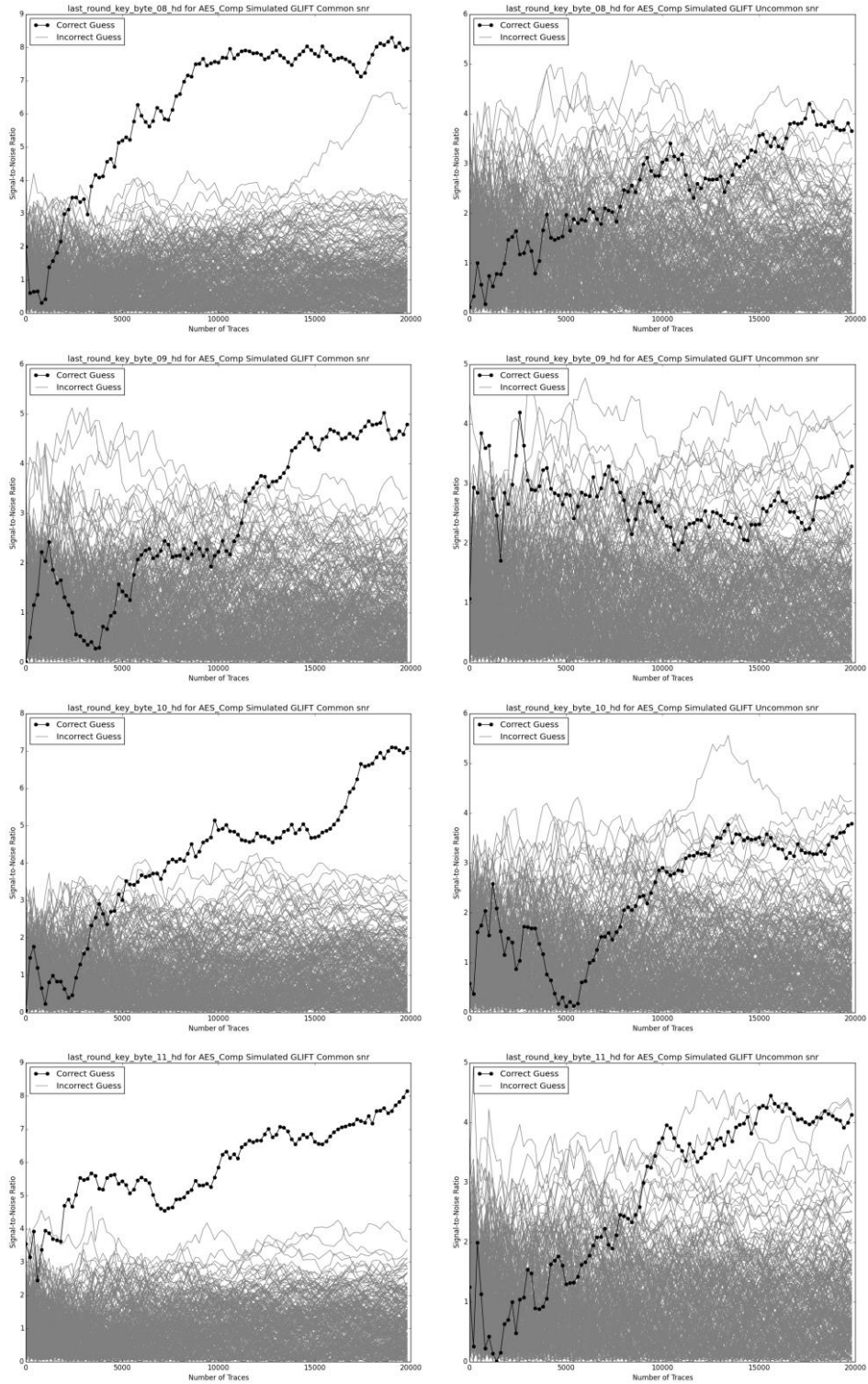
Correlation Ratios for AES Comp Measured (R) and Simulated (L), key bytes 12-15 with 20k traces



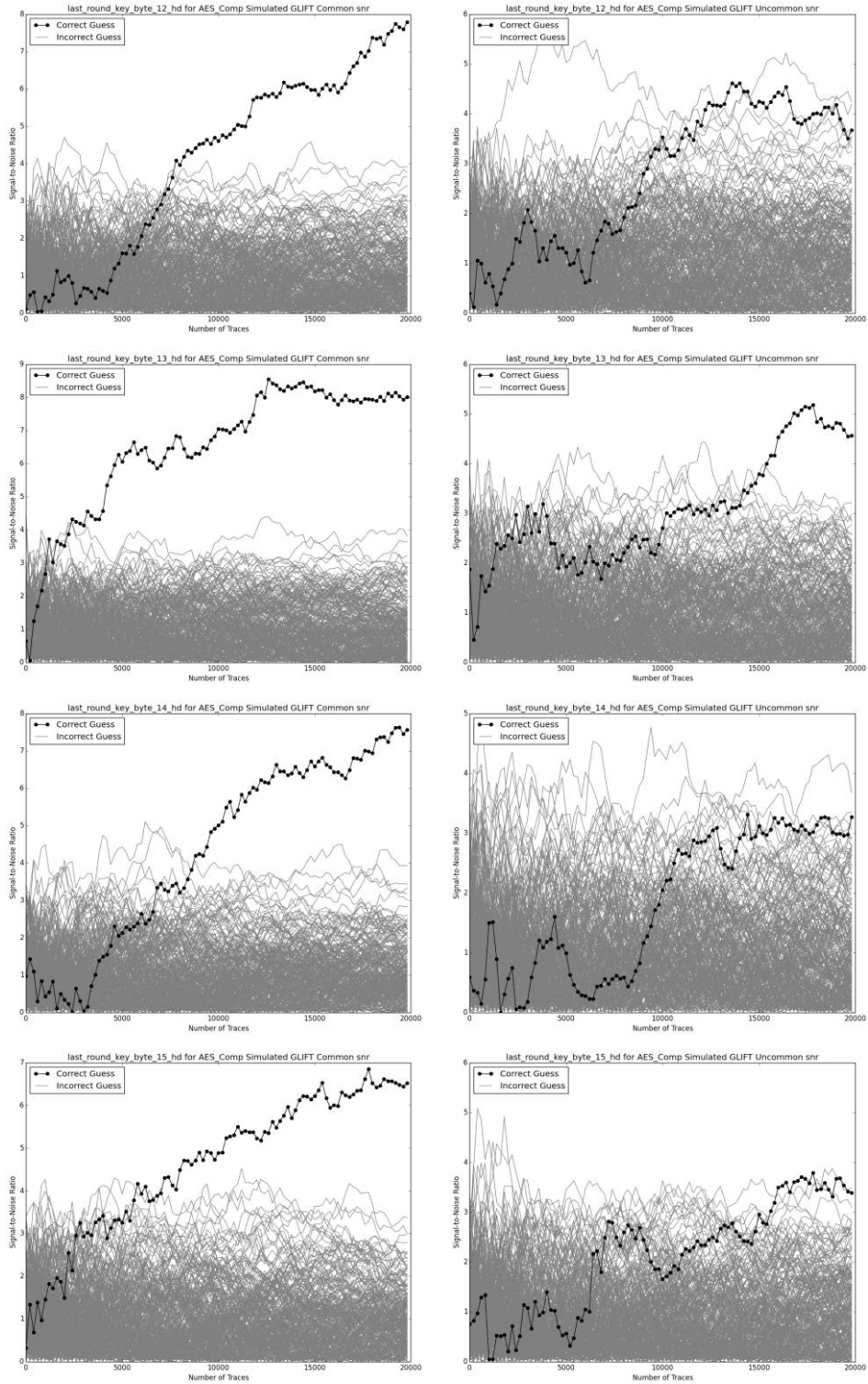
Signal-to-Noise Ratios for AES Comp Measured (R) and Simulated (L), key bytes 0-3, all guesses



Signal-to-Noise Ratios for AES Comp Measured (R) and Simulated (L), key bytes 4-7, all guesses



Signal-to-Noise Ratios for AES Comp Measured (R) and Simulated (L), key bytes 8-11, all guesses



Signal-to-Noise Ratios for AES Comp Measured (R) and Simulated (L), key bytes 12-15, all guesses