MUTE-AES: A Multiprocessor Architecture to prevent Power Analysis based Side Channel Attack of the AES Algorithm

Jude Angelo Ambrose, Sri Parameswaran and Aleksandar Ignjatovic
School of Computer Science and Engineering,
University of New South Wales, Sydney, Australia
Email: {ajangelo, sridevan, ignjat}@cse.unsw.edu.au

***Abstract*— Side channel attack based upon the analysis of power traces is an effective way of obtaining the encryption key from secure processors. Power traces can be used to detect bitflips which betray the secure key. Balancing the bitflips with opposite bitflips have been proposed, by the use of opposite logic. This is an expensive solution, where the balancing processor continues to balance even when encryption is not carried out in the processor.**

We propose, for the first time, a multiprocessor *algorithmic* balancing technique to prevent power analysis of a processor executing an AES cryptographic program, a popular encryption standard for embedded systems. Our technique uses a dual processor architecture where two processors execute the same program in parallel, but with complementary intermediate data, thus balancing the bitflips. The second processor works in conjunction with the first processor for balancing *only* when the AES encryption is performed, and both processors carry out independent tasks when no encryption is being performed.

Accessing the encryption key or the input data by the first processor begins the obfuscation by the second processor. To stop the encryption by the second processor, we use a novel signature detection technique, which detects the end of the encryption automatically. The multiprocessor balancing approach (*MUTE-AES*) proposed here reduces performance by 0.42% and increases the size of the hardware by 2X (though reduces to 0.1% when no encryption is being performed). We show that Differential Power Analysis (DPA) fails when our technique is applied to AES. We further illustrate, that by the use of this balancing strategy, the adversary is left with noise from the power profile with little useful information.

I. Introduction

The rapid increase in the use of embedded systems for performing secure transactions, has proportionally increased the security threats which are faced by such devices. Side channel attack, a sophisticated security threat to embedded devices like smartcards, mobile phones and PDAs, exploits the external manifestations like processing time [8], power consumption [17] and electromagnetic emission [25] to identify the internal computations. Power analysis attack, introduced by Kocher et al. [16] in 1998, is still used by adversaries to eavesdrop on confidential data while the device is executing a secure transaction. The adversary observes the power trace dissipated/consumed by the chip during the encryption/decryption of a cryptographic program (such as DES, AES, RSA, ECC, etc.) and predicts the secret key used for encryption by extracting necessary information from the power trace.

There are two key power analysis methods: (1), Simple Power Analysis (SPA), which reveals direct information about the intermediate data from the power magnitude; and (2), Differential Power Analysis (DPA), which requires multiple power traces to perform a statistical analysis to predict the data used in computations [16]. SPA identifies the Hamming weights (i.e., number of 1's set in the output [7]) of the intermediate data using the power magnitude of certain instruction executions, based on the hypothesis that the higher the Hamming weight, the higher the power magnitude [19]. This hypothesis allows an attacker to guess the correct key more quickly, than if he/she were to use brute force. DPA is a more powerful technique than SPA, which is based on the hypothesis that there is a significant difference in the power consumption in processing 1's and 0's [16].

The bitflips (or the Hamming weight) caused by the secret key are significantly visible in the power trace, causing enough variations to reveal values. Balancing such bitflips during the encryption is one of the most appropriate solutions for these power analysis attacks. Recent upsurge in the use of multiprocessors in embedded systems and their vast deployment in devices performing secure transactions (such as mobile phones) [2, 22, 36] motivated the use of a solution involving a multiprocessor architecture to balance the bitflips. The balancing can be performed using multiprocessors by executing the algorithm using instructions with proper data in one processor, while executing the inverted data in parallel in the second processor.

This paper proposes a multiprocessor balancing technique using two processors (called *MUTE-AES*), where a second processor is utilized to execute instructions of the AES program in parallel with the first processor but with inverted intermediate data. The balancing is performed only when necessary, by combining the second processor during the time when the encryption program is executed in the first processor.

An encryption program starts by accessing the key and the input data from the data memory. In this paper, the balancing is triggered when the key or the input data is accessed from the data memory and the balancing is stopped when the signature[1] of the AES encryption expires. Both processors will execute independently and the second processor will start balancing when the first processor identifies an encryption routine (by accessing the input data or key which are in specific locations). The second processor is allowed to continue its original execution after the balancing is ceased.

Despite there being sufficient noise in a multiprocessor environment, where multiple processors might be doing tasks in parallel, masking the actual behavior in the power profile from a single processor is difficult. Millions of samples taken will statistically average out the noise, which then will reveal the encryption key. Hence, the balancing technique described here guarantees the power analysis resistance all of the time and does not allow any leakage of secure information to an adversary.

The rest of the paper is organized as follows. Section II discusses previous countermeasures proposed to combat power analysis based attacks. The algorithmic balancing methodology for AES is presented in Section III. Section IV defines the system architecture and details the signature detection. The experimental setup is explained in Section V. Experimental results are presented in Section VI. A discussion is provided in Section VII. The paper is concluded in Section VIII.

[1]Signatures can be used to detect the encryption routines in a program based on certain unique patterns of instruction executions. For example, in AES, there are closely clustered XOR instructions, signifying that encryption is taking place. We use such a signature in AES encryption, computed based on concomitance analysis [14]

978-1-4244-2820-5/08/$25.00 ©2008 IEEE

II. Related Work

Power analysis attacks have been researched extensively in the recent past, and various countermeasures have been proposed to combat such attacks. The key countermeasures are masking, current flattening, non-deterministic processing and balancing.

Masking techniques [9, 12, 20] involve the use of random values in the actual computation to obfuscate the power profile from revealing a correlated magnitude with the actual data being processed. To protect the AES algorithm, the result after each round is additionally computed with random values to produce random traces at critical points in the power profile [34].

Current flattening technique proposed by Muresan and Gebotys [21] uses *nops* to provide sufficient discharge to maintain the dissipated current at a fairly constant value. The secure coprocessor [31], which is designed for AES-based biometric applications, uses a constant power dissipating logic for any bit transitions. This coprocessor increases area cost by 3X and power cost by 4X. A signal suppression technique is proposed by Ratanpal et al. [26] to reduce the variations in the power profile for different inputs using a separate hardware component.

A non-deterministic processor [18] can be used to execute the program out-of-order, where the adversary will not be able to determine the instructions and their corresponding points in the power trace. This will not only scramble the power patterns for every execution of the program but also fails in Differential Power Analysis (DPA), breaching the correlation between the simulation and the power measurement.

We have observed several logic/circuitry level balancing techniques [13, 30, 32, 33] proposed in the recent past. These techniques use a complementary logic or a modified secure logic to balance bitflips. For example, if the original logic flips from $0 \rightarrow 1$, the complementary logic is designed to flip from $1 \rightarrow 0$ at the same time as the original circuitry. Dual-Rail logic [30] (also known as Dual-Rail pre-charge (DRP)) contains double the logic, one the original logic circuit and the other a similar logic but complementing the discharge from bitflips of the original [28]. This DRP design dissipates the same amount of power regardless of the data. Sense Amplifier Based Logic (SABL) [13] is designed to dissipate an unvarying amount of dynamic power for all bit transitions. The four possible bit transitions are $1 \rightarrow 0$, $0 \rightarrow 1$, $1 \rightarrow 1$ and $0 \rightarrow 0$. The Wave Dynamic Differential Logic (WDDL) is proposed by Tiri and Verbauwhede [32, 33] to dissipate power which is input independent. WDDL utilizes a complementary gate in parallel to the original, which receives the inverted inputs of the original, thus producing inverted outputs of the original gate. This results in an unvarying balanced power for different inputs.

Most masking techniques are algorithmic specific approaches requiring a higher degree of manual intervention and are proved to be vulnerable to second-order DPA attacks [15, 23, 35]. A highly dependent code segment cannot be executed out-of-order, and thus not protected by non-deterministic processors [18]. The current flattening technique [21] flattens the dissipated current based on basic blocks and consumes 75% additional runtime. Even though the circuitry level balancing techniques [13, 30, 32, 33] provide strong resistance to power analysis attacks, they also increase the chip area by 2X (some techniques require 4X), to accommodate the complementary logic. These additional logic circuits, which are permanently built inside the chip, are futile when no encryption is performed. WDDL techniques [32, 33] also require the routing of wires to be balanced, and the DRP logic [30] needs an additional compilation of special libraries.

Our multiprocessor balancing technique, *MUTE-AES*, also requires manual intervention and it is algorithm specific at this stage, similar to masking techniques [9, 12, 20]. However, *MUTE-AES* is comparatively easy to generalize by examining the algorithm and is not vulnerable to second-order DPA. *MUTE-AES* does not need a complete software modification compared to current flattening [21] and it does not cause much runtime overhead. Compared to the hardware balancing methods [13, 28, 30, 32, 33], *MUTE-AES* consumes twice the hardware only when balancing is required, by utilizing the already available processor. A miniscule amount of additional hardware is associated for the synchronization and signature detection circuit in *MUTE-AES*. The second processor is borrowed in *MUTE-AES* only when an encryption/decryption part in a cryptographic program is executed by the first processor, and otherwise the second processor is left for normal processing of other tasks. *MUTE-AES* does not need any libraries to be modified or compiled as has to be done in for DRP logics [30]. Hence, *MUTE-AES* is an easily implementable system with reduced area overhead usage for switching and synchronizing when no balancing is required.

A. Contributions

- For the first time, an algorithmic level multiprocessor balancing technique is proposed for AES.
- A Signature Detection technique is presented to switch off the balancing processor.

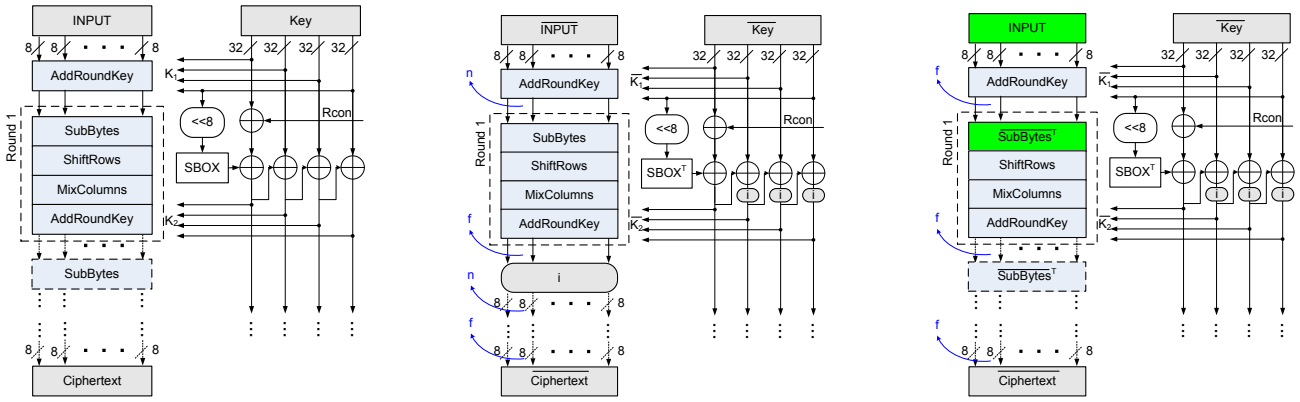
B. Limitations and Assumptions

- Our technique addresses only multiprocessor embedded systems with at least two identical processors.
- We assume that our system is self contained with separate memories for each of the processors.
- Caching is disabled during balancing.
- The secret key and the input data are stored in fixed data memory locations.
- Our balancing system has minimal or no operating system support.
- Both processors are clocked by a single source.
- The system only balances the AES algorithm, and not other encryption algorithms. Other algorithmic methods are needed to balance other encryption programs.

III. Algorithmic Balancing

We present here the algorithmic balancing as applied to AES to protect AES from power analysis. As shown in Figure 1(a), the AES encryption has several main functions: a key scheduling process which will generate subkeys (K_1,K_2,...) for each round from the original *Key*, the *AddRoundKey* function to XOR the *INPUT* with the *Key*, the *SubBytes* function for the SBOX lookups, *ShiftRows* and *MixColumns* to scramble the intermediate bytes. There are four SBOXes used in the *SubBytes* function.

Figure 1(b) and Figure 1(c) depict two different inversion approaches (partial and complete) in AES algorithm. The partial inversion approach is presented here only to emphasize the significance of the complete inversion. Both inversion approaches have the same key scheduling function as shown in Figure 1. The inverted key $\overline{Key}$ of the original AES is divided into subkeys and the inversion is performed when and where necessary to create inverted subkeys (denoted as i in a round box in Figure 1 on the right side segment of both figures). The $SBOX^T$ used in key scheduling is a transposed version (i.e., indices swapped) of the original SBOX, so called due to the inverse value used for the index. As the partial inversion in Figure 1(b) reveals, the inverted input $\overline{INPUT}$ is bitwise XORed (the encryption in 128-bit AES is performed in a 4×4 byte matrix) with the inverted first subkey $\overline{K_1}$. Since this will produce the normal output as the original AES (normal output denoted as n) the normal SBOX accesses will be performed. This will be followed by the normal *ShiftRows* and *MixColumn* operations. The final function in the first round (*Round 1*) is the *AddRoundKey* function which will



(a) Original AES Algorithm (b) Partial Inversion (c) Complete Inversion

Fig. 1. AES Algorithmic Balancing (images influenced by [29])

XOR the intermediate data with the second inverted subkey $\overline{K_2}$. Hence, an inverted output will be produced (denoted as f) after *Round 1*. This inverted output is again inverted (the inversion is denoted as a round box labeled i in Figure 1(b)) and the normal value of the original AES is sent to the next round. Similar process continues till the end of the AES encryption.

The complete inversion as shown in Figure 1(c) has the same key scheduling process of the partial inversion, but has two main different components in the encryption process (the changed components are shaded). Instead of the inverted input $\overline{INPUT}$, the original input *INPUT* is used. And all the SBOXes (four SBOXes) in $\overline{SubBytes}^T$ are inverted and transposed of the original. The *AddRoundKey* operation for the original input *INPUT* and the inverted subkey $\overline{K_1}$ will produce the inverted output of the original (denoted as f to specify *flipped*). Since $\overline{SubBytes}^T$ is inverted and transposed the inverted indices coming into the SBOXes will produce the inverted outputs compared to the SBOX outputs in the original AES. There will be four inverted outputs (each from an SBOX) and the *AddRoundKey* operation with the inverted subkey $\overline{K_2}$ will produce the inverted outputs of the original. This will continue till the end of the program. These modifications have produced a complete inversion in terms of data bits throughout the encryption.

The partial inversion in Figure 1(b) does not process the inverted data at the SBOX operations, but generates inverted outputs after the *AddRoundkey* operation of each round. This has a considerable effect in balancing (even though it is not completely balanced) especially when we look at the implementation of the AES encryption for each round, which does the XOR with the subkey first and then the SBOX accesses. For example, the first 32 bits intermediate result Y_0 in an encryption round is produced (in the C code) as $Y_0 = \overline{K_1}\hat{}FT0\hat{}FT1\hat{}FT2\hat{}FT3$; where $FT0$, $FT1$, $FT2$ and $FT3$ are the four SBOX lookups. According to this implementation it is visible that there is balancing in the process, since the inverted subkey produces inverted intermediate data after each XOR. However, the SBOXes (which are the main attack points) are receiving the normal input as the original and producing normal output. Since there is pipelining in the processor, there exists a chance that the balancing in the pipelines obfuscates the unbalanced SBOX access pipeline stages.

Since the inverted approaches shown in Figure 1 use certain extra flipping operations (denoted as i in round blocks), the original AES program should also have similar operations with the same set of instructions to synchronize both programs (i.e., original and inverted). Note that balancing is performed by executing same instructions in parallel but with complemented data values, which shows that the synchronization between processors is important. Hence, we created variables for such flipping operations, assigning all 0's in the original program and all 1's in the inverted program. XORing at both instances with that variable will perform the required task.

The attack point (the place where DPA is performed) in AES is the SBOX access, where an 8-bit intermediate data is loaded and stored into the memory. The following analysis proves that complete algorithmic balancing will provide an effective countermeasure against power analysis side channel attack for AES. To do this we consider a power model based on Hamming distance [6], as shown in the following Equation

$$P = kH + n, \tag{1}$$

where P is the power consumed, H is the Hamming weight function, k is the scalar gain and n is a noise term. H is given by $Y \oplus X$, where X is the previous value in the register and Y is the new value after the operation. As in the paper by Brier et al. [6] we assume that the initial value X = 0 (such an assumption is valid for any pre-charged logic [33]).

The 8-bit intermediate data in the Original AES (shown in Figure 1) is referred as x and the 8-bit intermediate data in the complete inversion (shown in Figure 1(c)) is referred as $\overline{x}$. The values of x and $\overline{x}$ are complementary, and as such the Hamming weights between x and $\overline{x}$ will be the number of bits in x. Since the Hamming weights for $x \oplus \overline{x}$ is always 8, the attack point is no longer vulnerable; k and n are constants, and since H is constant, P is a constant value. If the attack also considers the power consumption caused by the bitflips in the bus during load and store, the power model is added with an additional component rH_b as explained in [4]. The modified power model is presented in Equation 2, where r is the scalar gain and H_b is the Hamming weight in the bus during load or store.

$$P = kH + rH_b + n \tag{2}$$

Since the complete balancing uses complementary index and retrieving complementary outputs from the SBOX (as shown in Figure 1), the resulting Hamming weight H_b is also constant. Hence, the power consumption P is still maintained at a constant value.

IV. System Architecture

In this section we present the *MUTE-AES* architecture, which includes an in-built module (called *FUNIT*) for signature detection to stop balancing by identifying the encryption routine in AES. *It is assumed that the key and input data are stored and retrieved from well known fixed memory locations, thus the start time is well understood.*

A. Signature Detection

The instruction(s) sequences in an execution trace can be monitored and analyzed to accurately identify the encryption routines in a processor [5]. Concomitance analysis [14] is used in this paper to capture the encryption routine in AES, by looking at instruction executions to realize the patterns of temporal correlation. According to this analysis the signature for the encryption routine of the AES is identified as the consecutive XOR instructions occurring within a 15 instruction window as shown in Figure 2.

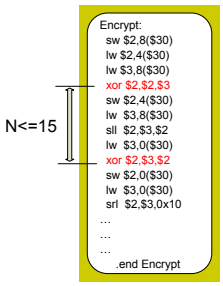


Fig. 2. Signature to Capture AES encryption

A comprehensive analysis of signature detection within encryption programs, and non-cryptographic programs are reported in [5], where it is shown that there is almost no signature hits which occur in non-cryptographic programs (with very few false positives). The signature detection unit, *FUNIT*, is shown in Figure 3 which exploits two flag registers: (1), *xorreg*, to indicate an XOR instruction execution (i.e., turns on when an XOR instruction is executed); and (2), *sel*, to indicate whether two consecutive XORs are seen within an instruction window of 15 (i.e., turns on when only two consecutive XORs are seen). A counter is used to count each instruction execution for the window computation. When an XOR instruction is executed while the counter is under 15 and above 0, the *sel* flag is either turned on or left on, the *xorreg* is turned off, and the counter is reset. If the counter is above 15, the *sel* is turned off and the *xorreg* is turned off, and the counter is reset.

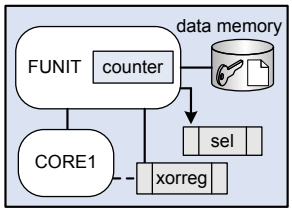


Fig. 3. Signature Detection Circuit

AES encryption uses key scheduling to create subkeys before encryption. Such an operation also requires balancing where the subkeys need to be created on-the-fly using the inverted secret key. However our signature detection captures only the encryption routine but not the key scheduling part. Therefore in our approach we propose to set the *sel* flag when the secret key or input data is initially accessed from the data memory, by using the fixed addresses the key and data are stored. Since the encryption continues after the last access of the memory location where the encryption key or input data is stored, it is not possible to stop the balancing based on memory locations. Note that the only way to stop the second processor from balancing is when the signature expires. Thus, the signature detection unit is a necessity to stop balancing. *i.e., the start of the balancing is triggered by first the access to the memory location of the encryption key or the input data, and the balancing is ended when the signature is no longer detected.*

If there is only a fixed encryption program, then it is possible to use the start and the end of the instruction memory addresses to start and stop balancing. However, our system allows for relocatable AES code to be implemented, by only having to know the data memory locations (which are usually fixed). The operating system can be also used to indicate the encryption program for balancing instead of a signature detection for the encryption routine in AES.

B. Processor Design

The multiprocessor balancing architecture, *MUTE-AES*, is presented in Figure 4. Here we present the setup for only the complete balancing, which provides balancing throughout the algorithm as shown in Figure 1. Two processors, CORE1 and CORE2, are designed to execute the same program executing in parallel (CORE1 and CORE2 executing the programs designed in Figure 1(a), Figure 1(c) respectively), and perform independently when no encryption is performed. Separate instruction memories (1 and 2) and data memories (1 and 2) are used for CORE1 and CORE2 as shown in Figure 4. The Data Memory 1 of CORE1 is initialized with the proper key (K), input data (D) and proper SBOXes, whereas Data Memory 2 of CORE2 is initialized with the inverted key (K'), input data (D) and modified SBOXes (the key scheduling SBOX is transposed and the SubBytes SBOXes are inverted and transposed as explained in Section III). There are three main flag registers used in the architecture for balancing: (1), *sel* register, which is set and reset using the Signature Detection unit as explained in Section IV-A; (2), *start* register, which is used to start both processors (CORE1 and CORE2) for balancing; and (3), *hold* register, which is used to pause balancing when an interrupt has to be serviced by any one of the processors in the middle of balancing.

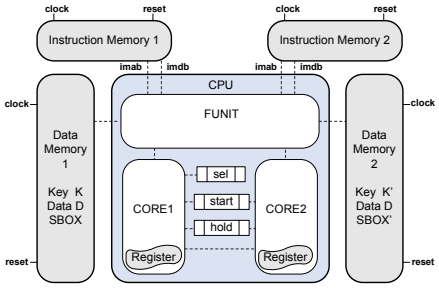


Fig. 4. Processor Architecture

The CPU includes two processors (CORE1 and CORE2), which fetch instructions from their own instruction memories, and *FUNIT* for signature detection as shown in Figure 4. The AES program binary used for balancing is stored in a part of Instruction Memory 2. When the *sel* flag is set, CORE1 sends an interrupt to CORE2. CORE2 saves its state in the stack (i.e., all the registers, PC, etc.). Until CORE2 is ready to balance, CORE1 is stalled. CORE2 sets the *start* flag register after saving its own state. As soon as the *start* flag is set both cores start to execute

their program for balancing (the balancing program for CORE2 is pointed to by a vector - much like an interrupt vector). When *sel* flag is cleared, CORE1 clears the *start* flag. When both flags (*sel* and *start*) turn off, CORE2 restores its stack and continues executing its original process. Note that both programs are in lock step mode.

Any one of the processors (CORE1 or CORE2) can receive an external interrupt from the operating system while balancing is in progress. If such an interrupt occurs the processor which receives the interrupt will set the *hold* flag (or else it can be an inter-processor interrupt) and both processors will pause. The processor which receives the interrupt saves its state and will service the interrupt. After the interrupt is served, the processor restores its state and clears the *hold* flag. Then both processors will resume encryption and balancing. The interrupts from one processor to another are typically handled by operating systems using software interrupts [11, 27]. In our experiments we did not have an operating system executing, and as such no interrupts were simulated for power measurements.

Note that the caches are disabled during balancing, since having a cache will cause problems in synchronizing processors. However, a scratchpad memory can be used for the encryption program if disabling the caches causes excessive performance penalty.

V. Experimental Setup

Our framework is implemented in a processor with the PISA (Portable Instruction Set Architecture) instruction set (as implemented in SimpleScalar tool set with a six stage pipeline [24]) without cache. The experimental setup for the power analysis implementations of a Single Processor (used as a base processor) and the *MUTE-AES* processor is outlined in Figure 5. The AES program in C is compiled using the GNU/GCC cross compiler for the PISA instruction set, and binaries are produced. ASIPMeister [1], an automatic processor design tool, is used to generate synthesizable VHDL description of the processor.

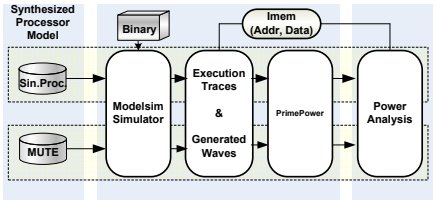


Fig. 5. Experimental Setup

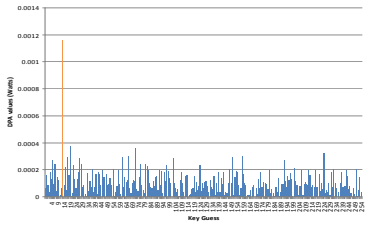
The synthesizable VHDL versions of the Single Processor and the *MUTE-AES* processor are synthesized using the Synopsys Design Compiler. ModelSim hardware simulator is used to simulate the program binary with the synthesized processor to generate the stimulus wave with switching information. The execution trace is extracted for future use from ModelSim after simulation. Power measurements are performed using PrimePower in watts (W). As shown in Figure 5 the address ($Addr$) and instruction opcode ($Data$) of instruction memory ($Imem$) are extracted from the execution trace. Perl scripts are used to reannotate the power values to the execution trace. DPA is implemented in a separate C program, and the execution extracts the necessary instruction power values from the trace.

VI. Results

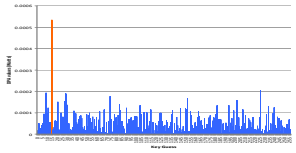
In this section we present the experimental results starting with the differential power analysis plots and then with hardware and runtime analysis.

A. Differential Power Analysis (DPA)

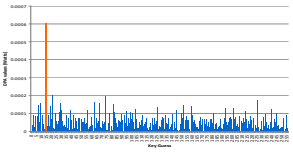
We performed the Differential Power Analysis (DPA) on AES to predict the correct 8 bits of the secret key based on the definitions from [12, 16], where the first output bit from the forth SBOX in first round is used for partitioning. The attack point for power measurement is the load instruction from the SBOX. All the DPA plots here are drawn for the DPA bias values (Y axis in *watts*) versus the possible 256 key values (i.e., 0 to 255 for 8 bits). A single processor (without any countermeasure) is attacked to determine the scenario of the attack and also as a base case. Figure 6 depicts the DPA plots for a single processor, where the top plot is attacked at the load (LW) instruction, the bottom left at the XOR instruction and the bottom right plot using the average of the power consumption during the SBOX access (i.e., average of the power magnitudes starting from load till the store after the SBOX lookup). In all three cases shown in Figure 6 the correct key (value of 14) is clearly identified by a significant peak, thus successfully passing the attack hypothesis.



(a) @ LW



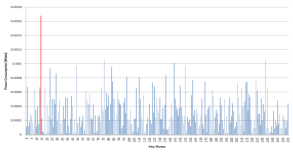
(b) @ XOR



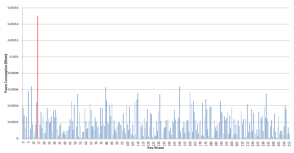
(c) Using Mean

Fig. 6. DPA plots on a single processor: No Balancing

To justify the necessity of the complete inversion algorithm for the countermeasure, the partial inversion explained in Section III is attacked using DPA. As Figure 7 depicts, the correct key is still predicted using the load (LW) instruction and the XOR instruction, both of which reveal a significant peak. This experiment shows that the balancing effect caused by operations other than the SBOX accesses cannot mask the key. Hence, the SBOX accesses play an important role in revealing the key, and has to be balanced completely.



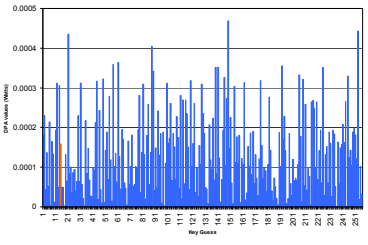
(a) @ LW



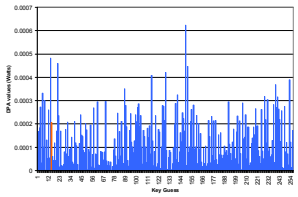
(b) @ XOR

Fig. 7. DPA plots for Partial Balancing

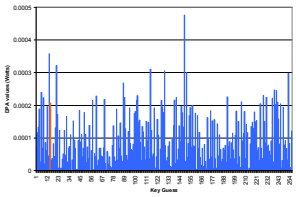
Figure 8 presents the DPA plots for the completely balanced processor architecture which is explained in Section III. As the plots reveal, the DPA signals at the correct key guess (value 14) failed to produce significant peaks for all the three cases (i.e., load instruction, XOR instruction and average during SBOX access). The DPA bias values are much smaller and have a smaller variation when compared to the values observed for the single processor, especially at the load (LW) instruction (which is the main attack point exploited by previous researchers [12]).



(a) @ LW



(b) @ XOR



(c) Using Mean

Fig. 8. DPA plots for Complete Balancing

To demonstrate the security of our method we compare runs of the encryption algorithm with two different inputs (these are randomly chosen) and subtract the two "clock cycle accurate" power traces. The only information available to the attacker comes from the differences between the power traces for different data inputs chosen. Thus, for an attacker to be able to extract any usable information about the key from the power traces, the power traces for different inputs must be distinguishable.

We used Fast Fourier Transform (FFT) analysis to examine the spectrum available. Our experiments show that the difference when balancing is used (shown in Figure 9(c)), is much lower with more zeroes than the difference for a single processor (shown in Figure 9(a)), and that the frequency spectrum of the difference looks much like white noise (shown in Figure 9(d)), unlike the spectrum of the difference for a single processor (shown in Figure 9(b)) that exhibits many well defined peaks.

Similarly, when the same input data but two different keys are used, and the two clock cycle accurate power traces are subtracted, and the FFT of the resulting signal is obtained, the spectrum largely resembles that of white noise, suggesting that there is little information available. The resulting FFT spectrum value is also much smaller in magnitude when compared to that of a single processor. However, if an attacker is able to extract any usable information from the power trace (about the key), then the power trace and the key value must have non negligible correlation. Thus, the FFT of the difference of power traces for different keys must be above the noise threshold. Thus, the magnitude of the difference and its spectrum indicate that no information above the noise level is present in the power trace, regardless of what data or key is used. This also proves that

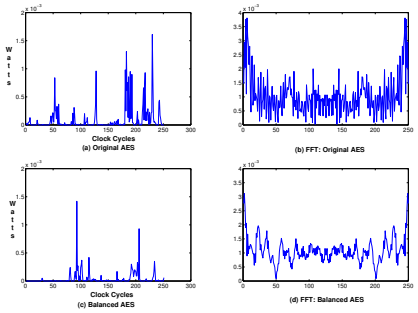


Fig. 9. FFT Analysis

our balancing technique (*MUTE-AES*) prevents the system from Simple Power Analysis (SPA). Since *MUTE-AES* balances the intermediate data throughout the AES algorithm (i.e., Hamming weight of the processed data is always balanced) there won't be any correlation between Hamming weight and power magnitude.

B. Hardware Summary

Table I shows the hardware details of the balanced processor (*MUTE-AES*) architecture, generated by Synopsys Design Compiler. The first column of Table I categorizes the main hardware properties (which are area, clock, dynamic power and leakage power). The second column presents the property values for Single Processor. The properties for the *MUTE-AES* architecture without the signature detection unit is presented in the third column, and column four details the properties for *MUTE-AES* with the signature detection unit implemented.

	Single	**No Sig.**	**With Sig.**
Area (*cell*)	110921.67	213457.61	213855.72
Clock (ns)	41.63	49.61	50.50
Power: Dyn. (mW)	38.44	76.78	80.19
Leak. (μW)	1.49	2.87	2.85

TABLE I
HARDWARE SUMMARY

When compared to the single processor the area, dynamic power (Dyn.), leakage power (Leak.) are doubled for the *MUTE-AES* architecture, as shown in Table I, since *MUTE-AES* uses two processors. The clock period slightly increased for *MUTE-AES* compared to the single processor, due to signature detection. The hardware is increased approximately by 0.1% when the signature detection unit is added to *MUTE-AES*.

C. Performance Overhead

The performance overhead caused, when the second processor is switched for balancing, is tabulated in Table II. Normal AES program costs 175,600 clock cycles including memory accesses. There will not be any delays in finishing the currently executing pipelines before switching, since the signature is detected at the memory stage.

As shown in Table II, every time balancing is performed, there is a delay of 722 clock cycles, which includes saving and restoring necessary registers, setting and clearing the flag for switching and memory accesses. This delay costs only 0.42% percentage in runtime. Note that this overhead does not include any delay in software interrupts which might occur while the system is encrypting.

	Clock Cycles
Basic AES	175600
Delay	
Save Registerfile	320
Save Registers,PC	40
Set start flag	1
Clear start flag	1
Restore registerfile	320
Restore Registers,PC	40
Total Delay	722
Performance Overhead	0.42 %

TABLE II
PERFORMANCE OVERHEAD

VII. DISCUSSION

We used a specific processor to act as the master processor to identify the signature using the attached flag registers and affix the second processor for balancing when needed. If there is an operating system, it can force an application to run on one processor [3], thus scheduling the AES program to always execute on the primary processor. In any other dynamic scheduling case each processor in the multiprocessor system should be attached with necessary flag registers, so that the processor which is acting as master at a particular instance can perform signature detection by updating its associated flags. This would also help the operating system schedule the cryptographic program on an available processor or a processor executing a lower priority task. Note that, operating systems are not preferred for smart cards but are heavily utilized in mobile phones and PDAs.

Similar algorithmic level balancing can be done using a VLIW processor, where a normal instruction and a complementary instruction can be included in a word [10]. But in such a case, VLIW will not be able to execute any other program when encryption is not performed. *MUTE-AES* is more flexible and requires less manual software modification than a VLIW.

Since balancing is done by a specific processor all of the time, a powerful magnetic probe can be placed on top of the chip to observe the electro magnetic (EM) dissipation only on that processor, which is executing the correct program. There is a chance the correct key can be exposed from these measurements. To prevent this scenario, the place and route of the chip should be performed in such a way, so that it is impossible for the adversary to extract the EM profile of a specific core.

VIII. CONCLUSIONS

This paper presented a multiprocessor balancing technique to prevent power analysis attacks in the AES cryptographic program. A dual processor chip is used where the second processor is affixed for balancing only when the encryption program in AES is detected using a signature by the first processor.

Since balancing is performed only when necessary, the performance of our system is significantly improved in comparison to other balancing methods. The same methodology applies with minimal changes to any encryption program that operates in a "bit-wise" manner, by either permuting or flipping bits in an essentially independent way (such as TripleDES). However, similar methods can be developed for non-bit-wise methods such as RSA, but are harder to implement and, while significantly safer than non-balanced single processor methods, do not result in perfect masking as for AES.

Our technique successfully prevents Power Analysis attacks and with careful place and route prevent Electro Magnetic Analysis attacks. The performance penalty is only 0.42% each time balancing is performed with around 2X in hardware cost. Note that there is only 0.1% hardware cost when no balancing is performed.

REFERENCES

[1] The PEAS Team. ASIP Meister, 2002. Available at: eda-meister.org/asipmeister.

[2] Chip Multi Processor Watch, 2007. Available at: http://view.eecs.berkeley.edu/wiki/Chip_Multi_Processor_Watch.

[3] Technology@Intel Magazine, 2007. Available at: http://www.intel.com/technology/magazine/computing/Core-programming-0606.htm.

[4] M.-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart. Power analysis, what is now possible... In *Asiacrypt '00*, pages 489–502, London, UK, 2000. Springer-Verlag.

[5] J. A. Ambrose, R. G. Ragel, and S. Parameswaran. A Smart Random Code Injection to Mask Power Analysis Based Side Channel Attacks. In *CODES+ISSS '07: Proceedings of the 5th international conference on Hardware/software codesign and system synthesis*, pages 51–56, New York, NY, USA, 2007. ACM Press.

[6] E. Brier, C. Clavier, and F. Olivier. Optimal Statistical Power Analysis, 2003. Cryptology ePrint Archive, Report 2003/152.

[7] E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In *CHES*, pages 16–29, 2004.

[8] D. Brumley and D. Boneh. Remote timing attacks are practical. In *USENIX*, August 2003.

[9] J.-S. Coron and L. Goubin. On boolean and arithmetic masking against differential power analysis. In *Ches '00*, pages 231–237, London, UK, 2000.

[10] J. Daemen and V. Rijmen. Resistance against implementation attacks: a comparative study of the AES proposals, 1999.

[11] M. T. DiBrino. Apparatus and method for managing interrupts in a multiprocessor system. *U.S. Patent 5265215*, 1993.

[12] C. Gebotys. A Table Masking Countermeasure for Low-Energy Secure Embedded Systems. *IEEE Trans. on VLSI*, 14(7):740–753, 2006.

[13] D. D. Hwang, P. Schaumont, K. Tiri, and I. Verbauwhede. Securing embedded systems. *IEEE Security and Privacy*, 4(2):40–49, 2006.

[14] A. Janapsatya, A. Ignjatovic, and S. Parameswaran. Exploiting statistical information for implementation of instruction scratch memory in embedded system. *IEEE Trans. on VLSI*, 14(8):816–829, 2006.

[15] M. Joye, P. Paillier, and B. Schoenmakers. On second-order differential power analysis. In *CHES*, pages 293–308, 2005.

[16] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. 1998. DPA First Article.

[17] S. Mangard. A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion. In *icisc 2002*, pages 343–358, 2003.

[18] D. May, H. L. Muller, and N. P. Smart. Non-deterministic processors. In *Acisp '01*, pages 115–129, London, UK, 2001. Springer-Verlag.

[19] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Investigations of power analysis attacks on smartcards. In *WOST'99*, pages 17–17, Berkeley, CA, USA, 1999. USENIX Association.

[20] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Examining smart-card security under the threat of power analysis attacks. *IEEE Trans. Computers*, pages 541–552, 2002.

[21] R. Muresan and C. H. Gebotys. Current flattening in software and hardware for security applications. In *CODES+ISSS*, pages 218–223, 2004.

[22] M. Nikitovic and M. Brorsson. An adaptive chip-multiprocessor architecture for future mobile terminals. In *CASES '02*, pages 43–49, 2002.

[23] E. Oswald, S. Mangard, C. Herbst, and S. Tillich. Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. In *ct-rsa 2006*, pages 192–207. Springer, 2006.

[24] J. Peddersen, S. L. Shee, A. Janapsatya, and S. Parameswaran. Rapid embedded hardware/software system generation. In *VLSID '05*, pages 111–116, 2005.

[25] J.-J. Quisquater and D. Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, pages 200–210, 2001.

[26] G. B. Ratanpal, R. D. Williams, and T. N. Blalock. An on-chip signal suppression countermeasure to power analysis attacks. *IEEE Transactions on Dependable and Secure Computing*, 01(3):179–189, 2004.

[27] T. Samuelsson, M. Akerholm, P. Nygren, J. Stärner, and L. Lindh. A comparison of multiprocessor real-time operating systems implemented in hardware and software. In *ARTOSS*, Porto, Portugal, July 2003.

[28] H. Saputra, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, R. Brooks, S. Kim, and W. Zhang. Masking the energy behavior of des encryption. *date*, 2003.

[29] S. Shimizu, H. Ishikawa, A. Satoh, and T. Aihara. On-demand design service innovations. *IBM J. Res. Dev.*, 48(5/6):751–765, 2004.

[30] D. Sokolov, J. Murphy, A. Bystro, and A. Yakovlev. Design and analysis of dual-rail circuits for security applications. *IEEE Trans. Comput.*, 54(4):449–460, 2005.

[31] K. Tiri, D. Hwang, A. Hodjat, B. Lai, S. Yang, P. Schaumont, and I. Verbauwhede. A side-channel leakage free coprocessor ic in 0.18um cmos for embedded aes-based cryptographic and biometric processing. In *Dac '05*, pages 222–227, New York, NY, USA, 2005. ACM Press.

[32] K. Tiri and I. Verbauwhede. A logic level design methodology for a secure dpa resistant asic or fpga implementation. In *DATE*, pages 246–251, 2004.

[33] K. Tiri and I. Verbauwhede. A digital design flow for secure integrated circuits. In *IEEE Trans. on CAD of Integrated Circuits and Systems*, volume 25, pages 1197–1208, 2006.

[34] E. Trichina, D. D. Seta, and L. Germani. Simplified adaptive multiplicative masking for aes. In *Ches '02*, pages 187–197, London, UK, 2003. Springer-Verlag.

[35] J. Waddle and D. Wagner. Towards efficient second-order power analysis. In *CHES*, pages 1–15, 2004.

[36] W. Wolf. Multimedia applications of multiprocessor systems-on-chips. In *Date '05*, pages 86–89, Washington, DC, USA, 2005. IEEE Computer Society.