Multiprocessor Information Concealment Architecture to prevent Power Analysis based Side Channel Attacks

Jude A. Ambrose[†] Roshan G. Ragel[‡] Sri Parameswaran[†] Aleksandar Ignjatovic [†]

[†]School of Computer Science and Engineering,

University of New South Wales, Sydney, Australia

{ajangelo, sridevan, ignjat}@cse.unsw.edu.au

[‡]Department of Computer Engineering,

University of Peradeniya, Sri Lanka

roshanr@ce.pdn.ac.lk

Abstract

Side channel attackers observe external manifestations of internal computations in an embedded system to predict the encryption key employed. The ability to examine such external manifestations (power dissipation or electromagnetic emissions), is a major threat to secure embedded systems.

This paper proposes a secure multiprocessor architecture to prevent side channel attacks, based on a dual-core algorithmic balancing technique, where two identical cores are used. Both cores use a single clock and encrypt simultaneously, with one core executing the original encryption, while the second executes the complementary encryption. This effectively balances the crucial information from the power profile (note that it is the information and not the power profile itself), hiding the actual key from the adversary attempting an attack based on Differential Power Analysis (DPA).

The two cores normally execute different tasks, but will encrypt together to foil a side channel attack. We show that, when our technique is applied, DPA fails on the most common block ciphers, DES and AES, leaving the attacker with little useful information with which to perpetrate an attack.

Index Terms

Side Channel Attack, Multiprocessor Balancing, Power Analysis

I. Introduction

The explosion of embedded systems and their resulting ubiquity is a cause for their frequent use in secure transactions. Thus researchers have begun to investigate secure techniques to prevent confidential data transmissions being accessed by an adversary. Side Channel Attacks (SCAs) are a security threat to embedded systems, where adversaries eavesdrop on external manifestations like power usage [1, 2], processing time [2, 3] and electro-magnetic (EM) emission [4] from the device and correlate these properties with internal computations, obtaining critical information to predict the key. These techniques have been shown to reveal the secret keys of cryptographic programs like DES, AES, RSA and SEAL [2]. Power analysis has been the most extensively used Side Channel Attack technique to extract secret keys during the execution of cryptographic algorithms [1, 5, 6].

There are two key power analysis methods: (1), Simple Power Analysis (SPA), which reveals direct information about the data being executed; and (2), Differential Power Analysis (DPA), which requires multiple power traces to perform statistical analysis to predict the data used in computations [2]. There are several SPA models used in the literature to attack the key from a single (or few) traces. One such model [7] predicts the Hamming weights (i.e., number of 1's set in the output) of a computed data using the power magnitude, based on the hypothesis that the higher the Hamming weight, the higher the power magnitude [8]. Another model exploits the conditional branches to identify the data used by the conditional statements in the code. For example, the attacker would simply distinguish the square and multiply patterns of the RSA from the power profile [9]. These SPA models allow 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 [2]. Fig. 1 depicts a brief overview of a DPA, where the adversary feeds several inputs to the chip and records the power traces. Several guesses of the encryption key are made. For each of the key guesses the output power traces (of several separate inputs) are separated into two, depending upon whether we expect the output to be a "1" or a "0" as shown in Fig. 1. The difference between the averages of these sets is known as the DPA signal (also known as DPA bias signal). A peak in this DPA signal trace, which is plotted against possible key guesses, will reveal the correct key. In Advanced Encryption Standard (AES [10]) encryption, the most commonly exploited intermediate bit for the DPA attack is one of the output bits from an SBOX [11]. Further references about DPA can be found in [2, 5].

Such successful DPA attacks [2, 5, 6, 12, 13] prove that bit-flips caused by secret keys enforce significant variation in the dissipated power sequence. Executing a program with the complementary algorithm will produce complementary bit-flips, with the original algorithm (e.g., bit-flip 1 $\rightarrow$ 0 will correspond to bit-flip 0 $\rightarrow$ 1). Therefore, if we execute both the original and complementary encryptions simultaneously, the information on the power profile due to bit-flips of the secret key will cancel out. Note that the power profile itself will be the addition of both cores. Specifically, it is the correlation between the critical information and the power profile that we mask. Thus it is not our intention to have a flat power profile, but a power profile which is absent of information containing the encryption key.

The plethora of embedded systems containing multiprocessors such as cell phones, PDAs, gaming consoles, audio players, video recorders and video cameras [14–16] motivated the use of one in-built core of a multi-core processor to function as a balancing unit, countering the real effects caused by the secret key in the power profile.

In this work, we have implemented the proposed balancing technique on a dual-core multiprocessor. The two cores of the processor generally execute independent tasks (or threads), but when one core starts to execute the encryption algorithm, the second core automatically starts the complementary encryption program. Note that the second core is used for balancing only when there is an encryption algorithm running on the first core.

*Part of this work was presented at ICCAD (2008) [**?**], where the balancing mechanism for AES was explained. This journal generalizes the balancing mechanism by enforcing it to DES and AES. The balancing control and switching mechanisms are presented in detail.*

Motivation

Significant variation in the dissipated power sequence, due to the bit-flips occurring while processing encryption keys, enable power analysis attacks. Balancing these bit-flips would mask the correlation produced by the actual key. However, previous balancing techniques [17–22] used complementary logic within the chip that performs balancing even when there is no encryption, making them area and power hungry. Real time embedded systems have to not only be secure, but must remain small, fast and cost-effective. As such, it is imperative that components be used for actual processing as much as possible, and only used for obfuscation when absolutely necessary. Using an already available core for algorithmic balancing inside a multicore chip will prevent adding extra hardware components, which would have been necessary otherwise.

Paper Organization

The rest of the paper is organized as follows. Section II summarizes previously proposed countermeasures. Our balancing methodology is explained in Sections III and the framework is presented in Section IV. Section V explains the experimental setup used for measurement and analysis. Results are presented in Section VI and a discussion is provided in Section VII. Finally, the paper is concluded in Section VIII.

II. Related Work

In recent years, several countermeasures have been proposed to prevent side channel attacks. Effective countermeasures include masking, current/power flattening, non-deterministic processing, instruction injection and balancing.

Masking techniques [5, 11, 23] use random values during the actual computation to prevent the processed data being exploited by the adversary. For example, Trichina et al. [24] proposed a masking technique to protect AES, where random values are used for additional computation of the result after each round.

Muresan and Gebotys [25] proposed a current flattening technique, where the dissipated current is flattened by adding *nop*s in the code to provide sufficient discharge. A signal suppression circuit in [26] is used to suppress the dissipated current.

Non-Deterministic Processors [27] execute the instructions out-of-order, issuing independent code segments randomly during runtime. The adversary cannot identify the places where specific instructions are executed by looking at the power profile, since the instruction execution is non-deterministic.

A number of researchers have stated that the insertion of dummy instructions (NOPs) could be a solution to protect systems from side channel attacks [2, 28]. Several dummy operation insertion techniques are proposed for ECC cryptosystems to create a constant execution path [29, 30]. Random instruction injection techniques are presented in Ambrose et al. [31, 32]

Recently, several researchers [**?**, 17–22, 33] have proposed logic/circuitry level balancing techniques to prevent side channel attacks, using complementary logic or modified secure logic to balance bit-flips. Dual-Rail circuits [20] are designed to consume the same power regardless of data processed. In Dual-Rail circuits, each logic circuit is attached to complementary logic, complementing the discharge occurring in the original logic circuit due to bit-flips [19]. Sense Amplifier Based Logic (SABL) [18], is a circuit which dissipates the same dynamic power regardless of the bit transition ($1 \rightarrow 0$, $0 \rightarrow 1$, $1 \rightarrow 1$ or $0 \rightarrow 0$). Tiri and Verbauwhede [21, 22] proposed Wave Dynamic Differential Logic (WDDL), which uses parallel combination of two complementary gates to dissipate power independent of input. The second gate in WDDL uses the inverted inputs of the original, producing the inverted output of the original (first) gate, thus dissipating balanced power. Some improved versions of the mentioned circuitry techniques are proposed in [17, 33].

In general, masking techniques have been vulnerable to second order DPA attacks [12, 34, 35], which are algorithm specific approaches requiring higher degree of manual intervention. Non-Deterministic Processors [27] are infeasible in highly dependent software code, which cannot be executed *out-of-order*. The Non-Deterministic Processor [27] has complex circuitry (though no overhead has been reported). The current flattening technique [25] increases execution time by up to 75%, and flattens locally, based upon basic blocks. Dummy or Random instruction injections [29–32] can be eliminated using time shifting [36] and by using a large number of samples. The circuitry level balancing solutions [17, 18, 20–22, 33], considered the most appropriate solutions to prevent DPA, double the original chip size due to the complementary balancing logic circuits. These balancing logic circuits (which are permanently implemented inside the chip) are unnecessary when a non-cryptographic program is executing on the chip. In addition to the significant area overhead, WDDL techniques [21, 22] require balanced routing of wires and [33] additional effort to produce the compilation of special libraries. Furthermore, certain circuitry level balancing techniques (such as Dual-rail Logics [20]) are proved to be still vulnerable for side channel attacks when enforcing glitches [**?**] and EM attacks without any backend-level countermeasures [**?**].

Our multiprocessor balancing technique also requires manual intervention and it is algorithm specific at this stage, similar to masking techniques [5, 11, 23]. However, our solution is comparatively easy to generalize by examining the algorithm and is not vulnerable to second-order DPA. Our multiprocessor balancing does not need a complete software modification compared to current flattening [25] and it does not cause significant runtime overhead. Compared to the hardware balancing methods [17–22, 33], ours does not require additional hardware, and utilizes one of the already available cores. A miniscule amount of additional hardware is associated for the synchronization. The second core is utilized only when an encryption/decryption part in a cryptographic program is executed by the first core, and otherwise the second core is left for regular processing of other tasks. Our approach does not need any libraries to be modified or compiled as has to be done in [33]. Hence, the multiprocessor balancing is an easily implementable system with reduced area overhead usage for switching and synchronizing when no balancing is required.

A. Contributions

- An algorithmic level balancing architecture to conceal information is proposed for a dual-core multiprocessor
- A synthesized hardware implementation is presented and the security is justified by exploiting the attack using power measurements
- The balancing is demonstrated on both DES and AES, which are well known block ciphers

B. Limitations and Assumptions

- Our technique addresses only multiprocessor embedded systems with at least two identical cores.
- We assume that our system is self contained with separate memories for each of the cores.
- Cache is disabled during balancing.
- Both cores are clocked by a single source.

III. Algorithmic Balancing

This section presents the balancing approaches proposed for DES and AES cryptographic algorithms.

A. Algorithmic Balancing in DES

Data Encryption Standard (DES [37]) is one of the popular and an aged block cipher [**?**]. We use DES for balancing to demonstrate the proof of our concept. As depicted in Fig. 2(a), the DES algorithm contains 16 rounds. In the first round, as per the algorithm the input is passed through an initial permutation and then split into two parts, L_0 and R_0. XOR (denoted as $\oplus$) is applied to R_0 and the original sub key K_1 (which is 48 bits wide and is one of 16 subkeys). R_0 and K_1 are XORed and the output is used for SBOX look-up. An XOR operation is applied to the output from the SBOX and L_0 to produce the intermediate value a_1. This value a_1 is placed in R_1 and R_0 is placed in L_1. This completes the first round.

A similar procedure is continued for all rounds (16 rounds), as shown in Fig. 2(a), and finally the output is generated using an inverted permutation. The point at which side channel power analysis occurs is the store operation of the intermediate result a_i in DES. Though differing places for exploitation have been tried by previous researchers, they have, so far, only succeeded when attacking a_i [11].

The balancing effect in DES is accomplished by inverting both data and the key as shown in Fig. 2(b). Similar to the original DES algorithm in Fig. 2(a) the inverted DES algorithm in Fig. 2(b) passes the inverted input through the initial permutation and then splits it into two parts, $\overline{L_0}$ and $\overline{R_0}$. XOR is again applied to the fragment of inverted data present in $\overline{R_0}$ and the inverted sub key $\bar{K_1}$. This results in the same output as applying XOR to non-inverted data and non-inverted key as shown in Fig. 2(a). The output, which is the XOR of $\overline{R_0}$ and $\bar{K_1}$, is used for SBOX look-up, and the SBOX value produced will be the same as the one obtained with original data and original key. However, XOR is now applied to the output from the SBOX, and data $\overline{L_0}$, thus resulting in inverted output $\bar{a_1}$ being placed in $\overline{R_1}$. $\overline{R_0}$ is passed to $\overline{L_1}$; thus, $\overline{L_1}$ is complementary of L_1. The R_i and L_i in the original algorithm shown in Fig. 2(a) are completely complementary to $\overline{R_i}$ and $\overline{L_i}$ of the inverted algorithm shown in Fig. 2(b). This property of complementary execution will be preserved throughout all rounds of the DES algorithm. In this way a total complement of essential information is achieved between the core performing the encryption with the real key and data with the core acting on the inverted key and inverted data.

The attacking point (the place where DPA is performed) in DES is the store of a_i. Therefore having the same index for the SBOX lookup in both programs will not cause any vulnerability, since complementary outputs are stored. The following analysis proves that algorithmic balancing will provide an effective countermeasure against power analysis side channel attack for DES. To do this we consider a power model based on Hamming distance [38], as shown in Equation 1, where P is the power consumed, H is the Hamming weight function, k is the scalar gain and n is the 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.

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

As can be seen from Fig. 2(a) and Fig. 2(b), a_1 and $\bar{a_1}$ are complementary, and as such the Hamming weight of a_1 and $\bar{a_1}$ will be the number of bits in a_1. Likewise, a_2, a_3 etc will be balanced by their corresponding complemented values as shown in Fig. 2(b). Since the Hamming weights for $a_i \oplus \bar{a_i}$ is always 48, the attack point is no longer vulnerable, since k and n are uncorrelated from the sensitive transitions, and now H is constant, thus making P a constant value. Similar to Brier et al. [38], we assume that the initial value X = 0 (such an assumption is valid for any pre-charged logic [22]).

B. Algorithmic Balancing in AES

Advanced Encryption Standard (AES) is a symmetric-key block cipher encryption algorithm [10] and is used in a wide range of embedded applications [1]. In our experiments we use the 128-bit AES (AES with 192 bits and 256 bits are also currently used). Fig. 3 depicts the AES algorithm, specifying only the necessary parts to analyze the attack. A detailed explanation of AES can be found in [39, 40]. The

128-bit AES is considered for our experiments; others (192-bit and 256-bit) can be also treated in a similar way.

As shown in Fig. 3 the 128 bits input data (which is shown as separate 8 bits blocks — thus input is divided into blocks numbered from 0 to 15) is xor'ed with the 128 bits round key (this initial round key is the actual secret key, and the remaining round keys are generated using a key scheduling algorithm [1]). The result of the xor between the input and key (which are Y0, Y1, Y2 and Y3) will be used as indices for the SBOX (FT0, FT1, FT2 and FT3) lookups. Different lines are used to show which bytes are combined together for different table lookups. For example, the lines are fed into blocks FT0 from Y1[0], FT1 from Y2[1], FT2 from Y3[2] and FT3 from Y0[3] at once. The output from the SBOXes are xor'ed together. Separate xor'ed values are then fed into Y0, Y1, Y2 and Y3. The 128 bits result is then xor'ed with the next round key. This process will continue for several iterations/rounds.

The main part for power analysis is the SBOX lookups. All key bytes have their one and only distinctive place, which they contribute to one of the SBOX lookups. For example, Key byte KEY[3] only contributes to the FT3 lookup in a round as shown in Fig. 3. Therefore if an adversary wants to predict KEY[3], the only place for analysis would be the FT3 lookup.

We present here the algorithmic balancing as applied to AES to protect AES from power analysis. As shown in Fig. 4(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.

Fig. 4(b) and Fig. 4(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 Fig. 4. 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 Fig. 4 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 Fig. 4(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 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 Fig. 4(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 Fig. 4(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 Fig. 4(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 = $\overline{K_1} \oplus FT0 \oplus FT1 \oplus FT2 \oplus FT3$; 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 Fig. 4 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 cores 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 as shown in Fig. 3. The same power model used for DES (illustrated in Equation 1), is considered to prove that complete algorithmic balancing will provide an effective countermeasure against power analysis side channel attack for AES. The 8-bit intermediate data in the Original AES (shown in Fig. 4) is referred to as x and the 8-bit intermediate data in the complete inversion (shown in Fig. 4(c)) is referred to as $\overline{x}$. For example, the intermediate data a and b during FT3 lookup as shown in Fig. 3 will be complementary for the complete inversion algorithm (i.e., the transpose and inverted values in FT3 will make b complementary). 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 weight 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 [28]. 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 Fig. 4), the resulting Hamming weight H_b is also constant. Hence, the power consumption P is still maintained at a constant value.

IV. Multiprocessor Balancing Architectures

The multiprocessor balancing architectures for both DES and AES are presented in this section.

A. Base Architecture

Fig. 5 depicts the schematic diagram of the base dual core processor used in our design. As depicted, the processor has two identical cores with separate instruction and data memories for each core.

B. Balancing Control

In our implementation we use a flag register to indicate the encryption program execution and to start the balancing. This flag register can be set in one of three ways: (i), using the operating system when scheduling the encryption program to the core; (ii), detecting the memory location accesses where the data and key for the encryption are stored; or (iii), using a special instruction to set the flag register. Termination of balancing, by clearing the flag register, can be performed in a similar fashion by using the operating system which can clear the flag register when the scheduled encryption program is completed; or by inserting a special instruction in the source code to clear the flag.

For our experiments, balancing is triggered and terminated by two special instructions which are instrumented in the source code (i.e., the first, *startBal*, and the last, *stopBal*, instructions of the encryption program A as shown in Fig. 6).

The execution of the *startBal* instruction indicates to the CONTROLLER that an encryption program is scheduled in the core. An External Interrupt is raised in CORE2. This is a maskable interrupt which will trigger after all the pipelines are flushed. Necessary registers (Registerfile, LO, HI and Program Counter) of CORE2 are saved in the stack as shown in Fig. 6. After the registers are saved, the CONTROLLER sends the program counter (PC) values to CORE1 and CORE2 on the same clock cycle (i.e., PC values of Encrypt in program A and $\overline{A}$). Both the original and complementary programs are executed in parallel by CORE1 and CORE2 respectively. When encryption is finished, the *endBal* instruction in program A will be executed by CORE1, which will send a signal to the CONTROLLER indicating the completion of encryption. The CONTROLLER restores the saved registers from the stack. CORE2 will resume its execution and the next available program will be scheduled on CORE1.

C. *Switching and Interrupts*

An External Interrupt is supplied by the CONTROLLER to one of the cores to service an interrupt while the balancing is in progress. This interrupt is serviced after the pipelines are flushed, thus the registers are updated with the correct values. Each interrupt routine (e.g., program B) will have three code segments: one, *backupReg*, to save the registers into the stack; two, *restoreReg*, to restore the registers from the stack; and three, *endIntr*, to end the interrupt. The *endIntr* instruction sends a Non-maskable Interrupt (NMI) to the CONTROLLER as shown in Fig. 6. An NMI request will force the controller to change the PC of both cores to their original locations to resume balancing. This is done at the same clock cycle inorder to preserve synchronization. When a CORE receives an interrupt during balancing, the other CORE is put on hold till the interrupt is serviced. However, the other CORE can be also allowed to execute the next program in the queue, but with careful modification in the controller to maintain synchronization for balancing. Table I lists the additional resources used for balancing. The Program Counter (PC) is saved into *PC_backup* at the fetch stage of the *external* interrupt and saved in the stack at the memory stage. The *switch* register is set and cleared by the *startBal* and *endBal* instructions.

Resources	Names
Instructions	startBal, endBal, endIntr
Interrupts	external (masked), NMI
Registers	PC_backup, switch

TABLE I
ADDITIONAL RESOURCES FOR BALANCING

The interrupts from one core to another can be also handled by operating systems using software interrupts [41, 42].

D. The Multiprocessor Balancing Frameworks

Fig. 7 presents the multiprocessor frameworks for DES (called *MUTE-DES*), and for AES (called *MUTE-AES*). In both *MUTE-DES* and *MUTE-AES*, the second core (CORE2) executes the same program as the first (CORE1), in parallel, but with the complementary algorithms as explained in Section III (when not encrypting, both cores run independently and execute different programs). Two separate instruction memories and data memories are used for CORE1 and CORE2. The *MUTE-DES* uses the inverted key (K') and inverted data (D') in its Data Memory 2 compared to its Data Memory 1, which uses the original key (K) and original data (D) as shown in Fig. 7(a). The *MUTE-AES* uses the inverted key, original data and the modified SBOXes (explained in Section III-B) in Data Memory 2 compared to its Data Memory 1 as shown in Fig. 7(b).

Each core fetches instructions from its corresponding instruction memory. The balanced DES and AES programs are stored in a part of CORE2's Instruction Memory. The CONTROLLER handles switching and interrupts for balancing as explained above. Note that the caches are disabled during balancing, since having a cache will not allow the synchronizing of cores. But if this results in excessive performance penalties, a scratchpad memory could be used for the encryption program.

V. EXPERIMENTAL SETUP

The hardware design that was used for the multiprocessor balancing architectures (*MUTE-DES* and *MUTE-AES*) is shown in Fig. 8. The Instruction Set Architecture (ISA) is fed into an automatic processor design tool called (*ASIPMeister* [43]) and two identical cores (CORE1 and CORE2) are generated to create both *MUTE-DES* and *MUTE-AES* separately.

As Fig. 8 depicts, *MUTE-DES* and *MUTE-AES* architectures are designed by combining CORE1 and CORE2 with the requisite flags. The balancing multiprocessor frameworks are implemented in a processor with the PISA (Portable Instruction Set Architecture) instruction set (as implemented in SimpleScalar tool set with a six stage pipeline) processor without cache. Fig. 9 depicts the process of performing power analysis on two different processors; Normal Dual Processor and MUTE processor (i.e., *MUTE-DES* or *MUTE-AES*). The Normal Dual Processor is used as a base processor for comparison. The DES program in C is compiled using GNU/GCC cross compiler for the PISA instruction set, which produces the binaries.

Synopsys Design Compiler is used to synthesize both processor models, which are simulated together with the program binary in the ModelSim hardware simulator to generate the stimulus wave with switching information. The execution trace is verified using Modelsim and extracted for future use. Primepower is used to measure the power values in watts (W). As shown in Fig. 9, 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 of that C program extracts the necessary instruction power values from the trace.

VI. Results

This section illustrates the results for *MUTE-DES* and *MUTE-AES* multiprocessor balancing frameworks, separately. Each architecture is presented with its DPA, hardware summary and performance analysis.

A. Results for MUTE-DES

1) DPA in MUTE-DES: We performed DPA experiments (based on [2] and [11]) on the normal dual-core architecture and *MUTE-DES* processor, by executing the DES cryptographic program and analyzing each SBOX lookup in each round. Two commonly used selection functions are either based on SBOX output bits, or based on the selection bits to the SBOX. Since the DPA, based on the SBOX output bit [11] did not reveal the key in our experiments, we present results of DPA based on selection bits to the SBOX [2]. This DPA follows the method proposed by Messerges et al. [5], which predicts the secret key by inspecting all possible SBOX lookups in each round.

The X axis of the displayed DPA plots give all possible key values from 0 to 255 (the first eight bits of a DES encryption key), which is plotted against the DPA values (*Watts*) in the Y axis for each key guess. The attacking point is the store instruction just after the SBOX lookup, where the SBOX output is XORed with the inverted data and stored (as explained in Section III). Fig. 10 shows a DPA trace performed on a normal dual-core processor, where one core is executing the DES program while the other remains idle.

Predicting the correct key corresponds to the significant peak in the DPA trace. In this example (shown in Fig. 10) the significant peak happens at a key value of 10; DPA was performed on the third selection bit, for the third sbox lookup, in the last round of encryption. Note that another significant peak also appears for the key 233, which is slightly lower than the peak at 10. Reasons for such ghost peaks are given in [7].

Concealment of information in our proposed architecture can be achieved when both data and key are inverted. Fig. 11 is produced when the balancing core uses the inverted secret key and the inverted data of the first core (DPA on the third selection bit, for the third sbox lookup, in the last round of encryption). As shown in Fig. 11, DPA cannot reveal the secret key in the *MUTE-DES* architecture, where the correct key (value of 10) does not produce a significant peak. The DPA values are much less for *MUTE-DES* as shown in Fig. 11, compared to the single core DPA values shown in Fig. 10. This is due to the balancing of the information in processing ‘1’s and ‘0’s.

To demonstrate the security of our method we compare executions of the encryption algorithm with two different inputs (these were 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 Fig. 12(b)), is an order of magnitude lower than the difference when the program is running on a single core (shown in Fig. 12(a)), and that the frequency spectrum of the difference has far less information (shown in Fig. 12(d)), unlike the spectrum of the difference for a single core (shown in Fig. 12(c)) which exhibits many well defined peaks.

Similarly, using the same input data but with two different keys, two clock cycle accurate power traces are obtained. One power trace is subtracted from the other and the obtained result is much smaller for the *MUTE-DES* architecture when compared to the result from the single core. Once again we see a much lower amount of variation in the FFT output. For an attacker to extract any usable information from the power trace about the key, the power trace and the key value must have a non negligible correlation. The difference of power traces for different keys must be above the noise threshold for a successful attack. Thus, the magnitude of the difference and its spectrum indicate that little information above noise level is present in the power trace, regardless of what data or key is used.

To get a better appreciation for the hiding ability of our method, we also ran an experiment which was not perfectly balanced, by complementing only the key, and not the data. The DPA graph is presented on Fig. 13(a), and it clearly shows that the peak that corresponds to the right guess is much lower on *MUTE-DES*. However, when we find the differences of the power traces for two different keys as shown in Fig. 13(b)[1], and examine its spectrum (shown in Fig. 13(b)[2]), we still see several well defined peaks, leaving the possibility that there might be some usable information. This shows that both the key and the data must be inverted to preclude the presence of any attacker usable information in the power trace.

Fig. 14 shows a general picture of the variation of DPA plots for a single core in the standard dual-core processor and for a processing system based on the *MUTE-DES* architecture upon the selection bits, for the third sbox lookup in the last round of DES encryption (which was shown to be vulnerable previously). The six plots in each category (each row) is based on the six selection bits (left to right plots start from the most significant bit to least significant bit). First row of Fig. 14 displays the DPA variation in a single core, while the second row depicts the DPA variation when using the inverted key and inverted data (invKeyinvData) in *MUTE-DES*.

From the variations shown in Fig. 14(a), the DPA when DES is scheduled on one core, produces higher values when compared to *MUTE-DES*. *MUTE-DES* displays a flattened variation (values close to zero) shown in Fig. 14(b). This flattening is due to the balanced bit-flips.

Note that the DPA based on the SBOX output bit [11] showed similar variations, where the DPA values are flattened after balancing.

2) Hardware Summary for MUTE-DES: Table II shows the hardware details of the *MUTE* architecture, generated by Synopsys Design Compiler. The *MUTE-DES* architecture consumes little additional hardware (around 0.1%) compared to the standard dual processor as shown in Table II. The clock period is slightly increased for *MUTE-DES*, because of the switching and synchronizing of the cores.

	Normal Dual Processor	**MUTE-DES**
Area ($cell$)	221842	222242
Clock (ns)	41.63	49.61

TABLE II
HARDWARE SUMMARY

3) Performance Overhead in MUTE-DES: The performance overhead caused when the second core is switched for balancing is tabulated in Table III. The normal DES program costs 76,350 clock cycles including memory accesses.

	Clock Cycles
Basic DES	76,350
Delay	
Save Registerfile	320
Save Registers,PC	40
Flush Pipelines	6
Interrupt to switch	1
Exit the interrupt	1
Restore registerfile	320
Restore Registers,PC	40
Total Delay	728
Performance Overhead	0.94%

TABLE III
PERFORMANCE OVERHEAD

As shown in Table III, every time balancing is performed, there is a delay of 728 clock cycles, which includes saving and restoring necessary registers, setting and clearing the flag for switching, and memory accesses. This delay comprises of only 0.94% percent of the runtime. Note that this overhead does not include any delay in software interrupts, which might occur while the system is encrypting. While the balancing is being performed, the second core will not be doing its usual tasks. Hence, the whole system will have a further delay of 76,350 cycles in the worst case scenario.

B. Results for MUTE-AES

1) 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 [2, 11], 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 standard dual-core processor, executing AES on one core and keeping the other core idle (without any countermeasure), is attacked to determine the scenario of the attack and also as a base case. Fig. 15 depicts the DPA plots for a single core (in a dual-core 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 Fig. 15 the correct key (value of 14) is clearly identified by a significant peak, thus successfully passing the attack hypothesis.

To justify the necessity of the complete inversion algorithm for the countermeasure, the partial inversion explained in Section III-B is attacked using DPA. As Fig. 16 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.

Fig. 17 presents the DPA plots for the completely balanced processor architecture which is explained in Section III-B. 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 core, especially at the load (LW) instruction (which is the main attack point exploited by previous researchers [11]).

Comparisons similar to that of DES (explained in Section VI-A.1) are performed to demonstrate the security of our method in *MUTE-AES* by subtracting two “clock cycle accurate” power traces for two different inputs. We used Fast Fourier Transform (FFT) analysis to examine the spectrum available. Our experiments show that the difference when balancing is used (shown in Fig. 18(c)), is much lower with more zeroes than the difference for an AES execution scheduled on one core of the processor (shown in Fig. 18(a)), and that the frequency spectrum of the difference looks much like white noise (shown in Fig. 18(d)), unlike the spectrum of the difference for the original AES (shown in Fig. 18(b)) that exhibits many well defined peaks.

The magnitude of the difference and its spectrum indicate that no information above noise level is present in the power trace, regardless of what the used data or key. This also proves that 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.

2) Hardware Summary for MUTE-AES: *MUTE-AES* also has the same hardware details of *MUTE-DES* as shown in Table II, since the same flag registers and cores are used. Note that the only difference is the memory setup between *MUTE-DES* and *MUTE-AES* as shown in Fig. 7.

3) Performance Overhead in MUTE-AES: The performance overhead caused, when the second core is switched for balancing, is tabulated in Table IV. Normal AES program costs 175,600 clock cycles including memory accesses.

	Clock Cycles
Basic AES	175600
Delay	
Save Registerfile	320
Save Registers,PC	40
Flush Pipelines	6
Set start flag	1
Clear start flag	1
Restore registerfile	320
Restore Registers,PC	40
Total Delay	728
Performance Overhead	0.42 %

TABLE IV
PERFORMANCE OVERHEAD

As shown in Table IV, every time balancing is performed, there is a delay of 728 clock cycles, which includes saving and restoring necessary registers, setting and clearing the flag for switching and memory accesses. This delay costs 0.42% percentage in additional runtime. This overhead does not include any delay in software interrupts which might occur while the system is encrypting. A further latency of 175,600 clock cycles will be added to the whole processing time in the worst case. This is because that the second core has to pause its normal processing when balancing.

VII. DISCUSSION

In our approach, the primary core executes the cryptographic program (the second one runs the complementary version), which is enhanced with additional flag registers. But in practice, the operating system can decide upon the task scheduling of processors. The operating system can force an application to run on a particular core [44], thus always scheduling the cryptographic program to the primary core. If we allow the OS to schedule the cryptographic program to either core, then additional flag registers have to be attached to all the cores, to enable such a universal execution.

Similar algorithmic level balancing can be performed using a VLIW processor, where a normal instruction and a complementary instruction can be included in a word [45]. But in such cases a VLIW processor will not be able to execute any other program when encryption is not being performed.

Since balancing is performed by a specific core all the time, a powerful magnetic probe can be placed on top of the chip to observe the electromagnetic (EM) dissipation of only one of the cores, which is executing the correct program. There is a high probability that the correct key can be exposed from these measurements. To prevent this scenario, the place and route of both cores can be performed together with both cores overlaid on top of each other, without a clear partition between the two.

Such on overlaid place and route also prevents one of the cores having a greater power footprint (due to physical variations in the wafer) than the other. If one of the cores did have a greater power footprint, then the encryption key can be extracted by side-channel analysis. However, with an overlaid place and route, individual bits in registers can exhibit greater power profile, but such things will be random amongst the two cores and will not remove the balancing ability of the MUTE architecture.

Another consideration is if one core encrypts while the other performs normal computation, then there might be sufficient hiding from the noise of the second core. Hiding the actual behavior in the power profile by the second core is difficult. Millions of samples taken will statistically average out the noise, which once subtracted will then reveal the encryption key. Hence, MUTE guarantees hiding all of the time and does not allow any leakage of secure information to an adversary.

We assume that the instructions are stored in an non-writable (i.e., only readable) ROM. Hence, an attacker can not modify the sensitive contol registers and the signals. However, we should point out that physical attack on the memory is still possible but several solutions exist to prevent physical attacks (which is not our scope). Storing such critical information into a ROM costs in hardware area. All the cryptographic techniques have this memory overhead (or a dedicated ROM to store the key and the code) and hence we did not consider it as an extra overhead.

In this work, we managed to attack the secret key only at the SBOX output in AES and at the SBOX input in DES. None of the other places in DES or AES produced a significant peak in the DPA signal. Hence, we demonstrated the balancing technique, considering only these attacking places. However, it is worth to note that the balancing algorithm would be different if the attack place is different. Our aim is to prove that the algorithmic multiprocessor balancing can be used to prevent power analysis attacks and its up to the designer to deploy this technique based on the attacking point observed.

VIII. Conclusions

This paper proposes a Multiprocessor Balancing Technique to prevent side channel attacks for the most common block ciphers, DES and AES. The second core in a dual-core processor is used to mask the effects caused by the secret key from the first core, by running the complementary program in parallel to the first core.

The balancing is only performed when necessary. The same methodology can be applied with minimal changes to any encryption programs which operate in a "bit-wise" manner, by either permuting or flipping bits independently. Similar methods can also be developed to 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 balancing. Our future work will investigate on such balancing approaches.

REFERENCES

[1] S. Mangard, "A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion," in *Information Security and Cryptology - ICISC 2002, 5th International Conference Seoul, Korea, November 28-29, 2002, Revised Papers*, ser. Lecture Notes in Computer Science, P. J. Lee and C. H. Lim, Eds., vol. 2587. Springer, 2003, pp. 343–358.

[2] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Lecture Notes in Computer Science*, vol. 1666, pp. 388–397, 1999.

[3] D. Brumley and D. Boneh, "Remote timing attacks are practical," in *Proceedings of the 12th USENIX Security Symposium*, August 2003.

[4] J. Quisquater and D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards." in *E-smart*, 2001, pp. 200–210.

[5] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Examining smart-card security under the threat of power analysis attacks." *IEEE Trans. Computers*, vol. 51, no. 5, pp. 541–552, 2002.

[6] S. B. Ors, F. Gurkaynak, E. Oswald, and B. Preneel, "Power-analysis attack on an asic aes implementation," *itcc*, vol. 02, p. 546, 2004.

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

[8] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Investigations of power analysis attacks on smartcards," in *WOST'99: Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*. Berkeley, CA, USA: USENIX Association, 1999, pp. 17–17.

[9] G. Hollestelle, W. Burgers, and J. I. den Hartog, "Power analysis on smartcard algorithms using simulation," http://eprints.eemcs.utwente.nl/798/, Eindhoven University of Technology, Eindhoven, Technical report CSR 04-22, 2004.

[10] National Institute of Standards and Technology, *Advanced Encryption Standard (AES)*, 2001, supersedes FIPS PUB 197–2001 November.

[11] C. Gebotys, "A Table Masking Countermeasure for Low-Energy Secure Embedded Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 7, pp. 740–753, 2006.

[12] E. Oswald, S. Mangard, C. Herbst, and S. Tillich, "Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers," in *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, ser. Lecture Notes in Computer Science, D. Pointcheval, Ed., vol. 3860. Springer, 2006, pp. 192–207.

[13] E. Peeters, F.-X. Standaert, N. Donckers, and J.-J. Quisquater, "Improved Higher-Order Side-Channel Attacks with FPGA Experiments." in *CHES*, 2005, pp. 309–323.

[14] "Chip multi processor watch," 2007, available at: http://view.eecs.berkeley.edu/wiki/Chip_Multi_Processor_Watch.

[15] M. Nikitovic and M. Brorsson, "An adaptive chip-multiprocessor architecture for future mobile terminals," in *CASES '02: Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*. New York, NY, USA: ACM Press, 2002, pp. 43–49.

[16] W. Wolf, "Multimedia applications of multiprocessor systems-on-chips," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 86–89.

[17] D. Hwang, K. Tiri, A. Hodjat, B.-C. Lai, S. Yang, P. Schaumont, and I. Verbauwhede, "Aes-Based Security Coprocessor IC in 0.18um CMOS With Resistance to Differential Power Analysis Side-Channel Attacks," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 4, pp. 781– 792, 2006.

[18] D. D. Hwang, P. Schaumont, K. Tiri, and I. Verbauwhede, "Securing embedded systems," *IEEE Security and Privacy*, vol. 4, no. 2, pp. 40–49, 2006.

[19] H. Saputra, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, R. Brooks, S. Kim, and W. Zhang, "Masking the energy behavior of des encryption," *date*, vol. 01, p. 10084, 2003.

[20] S. Danil, M. Julian, B. Alexander, and Y. Alex, "Design and analysis of dual-rail circuits for security applications," *IEEE Trans. Comput.*, vol. 54, no. 4, pp. 449–460, 2005.

[21] K. Tiri and I. Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation," in *DATE '04: Proceedings of the conference on Design, automation and test in Europe*. Washington, DC, USA: IEEE Computer Society, 2004, p. 10246.

[22] ——, "A Digital Design Flow for Secure Integrated Circuits." *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1197–1208, 2006.

[23] J.-S. Coron and L. Goubin, "On Boolean and Arithmetic Masking against Differential Power Analysis," in *Ches '00*, London, UK, 2000, pp. 231–237.

[24] E. Trichina, D. D. Seta, and L. Germani, "Simplified Adaptive Multiplicative Masking for AES," in *CHES '02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*. London, UK: Springer-Verlag, 2003, pp. 187–197.

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

[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*, vol. 01, no. 3, pp. 179–189, 2004.

[27] D. May, H. L. Muller, and N. P. Smart, "Non-deterministic Processors," in *ACISP '01: Proceedings of the 6th Australasian Conference on Information Security and Privacy*. London, UK: Springer-Verlag, 2001, pp. 115–129.

[28] M.-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart, "Power analysis, what is now possible..." in *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*. London, UK: Springer-Verlag, 2000, pp. 489–502.

[29] M. Barbosa and D. Page, "On the Automatic Construction of Indistinguishable Operations," in *Cryptography And Coding*. Springer-Verlag LNCS 3796, November 2005, pp. 233–247.

[30] C. H. Gebotys and R. J. Gebotys, "Secure Elliptic Curve Implementations: An Analysis of Resistance to Power-Attacks in a DSP Processor," in *CHES '02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*. London, UK: Springer-Verlag, 2003, pp. 114–128.

[31] J. A. Ambrose, R. G. Ragel, and S. Parameswaran, "RIJID: Random Code Injection to Mask Power Analysis based Side Channel Attacks," in *DAC*, 2007, pp. 489–492.

[32] ——, “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*. New York, NY, USA: ACM Press, 2007, pp. 51–56.

[33] T. Popp and S. Mangard, “Masked Dual-Rail Pre-Charge Logic: DPA-Resistance without Routing Constraints,” in *Cryptographic Hardware and Embedded Systems – CHES 2005, 7th International Workshop, Edinburgh, Scotland, August 29 - September 1, 2005, Proceedings*, ser. Lecture Notes in Computer Science, J. R. Rao and B. Sunar, Eds., vol. 3659. Springer, 2005, pp. 172–186.

[34] M. Joye, P. Paillier, and B. Schoenmakers, “On Second-Order Differential Power Analysis.” in *CHES*, 2005, pp. 293–308.

[35] J. Waddle and D. Wagner, “Towards efficient second-order power analysis.” in *CHES*, 2004, pp. 1–15.

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

[37] Computer Systems Laboratory (U.S.), *Data Encryption Standard (DES)*, 1994, category: computer security, subcategory: cryptography. Supersedes FIPS PUB 46-1–1988 January 22. Reaffirmed December 30, 1993. Shipping list no.: 94-0171-P.

[38] E. Brier, C. Clavier, and F. Olivier, “Optimal Statistical Power Analysis,” 2003, cryptology ePrint Archive, Report 2003/152.

[39] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag, 2002.

[40] W. Stallings, “The advanced encryption standard,” *Cryptologia*, vol. XXVI, no. 3, pp. 165–188, 2002.

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

[42] 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 *International Workshop on Advanced Real-Time Operating System Services (ARTOSS)*, 2003.

[43] “The PEAS Team. ASIP Meister,” 2002, available at: http://www.asip-solutions.com/english/.

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

[45] J. Daemen and V. Rijmen, “Resistance against implementation attacks: a comparative study of the AES proposals,” 1999, uRL: http://csrc.nist.gov/CryptoToolkit/aes/round1/pubcmnts.htm.

[46] S. Shimizu, H. Ishikawa, A. Satoh, and T. Aihara, “On-demand design service innovations,” *IBM J. Res. Dev.*, vol. 48, no. 5/6, pp. 751–765, 2004.

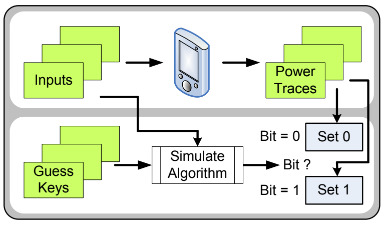
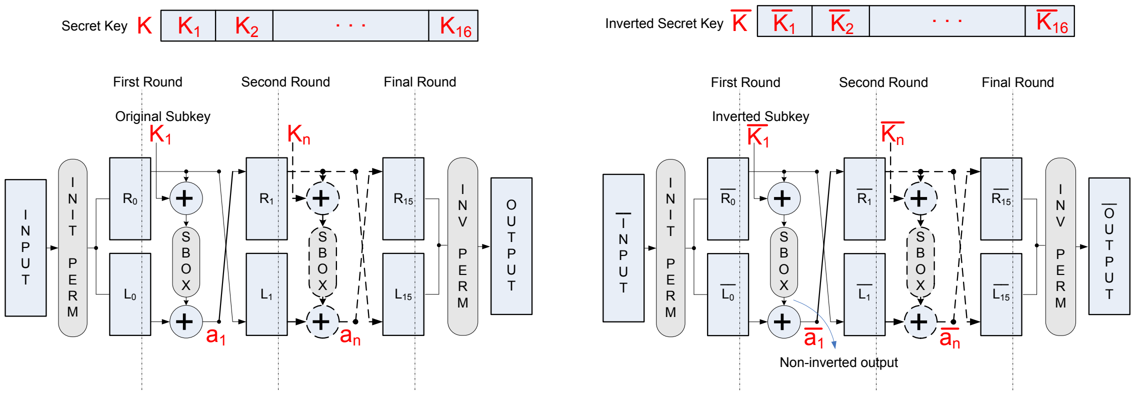


Fig. 1. An overview of DPA



(a) Original DES Algorithm

(b) Complementary DES Algorithm

Fig. 2. Algorithmic Balancing in DES

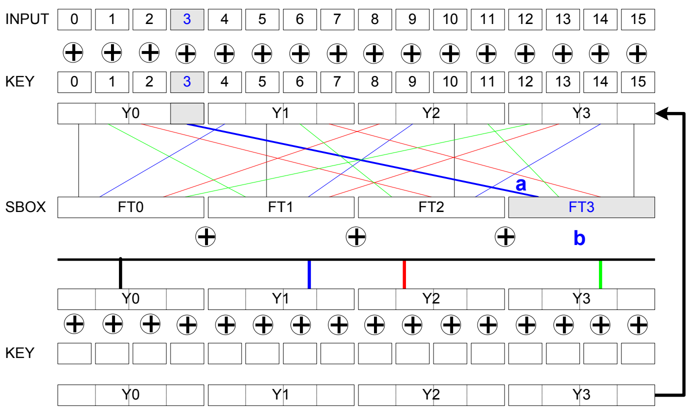
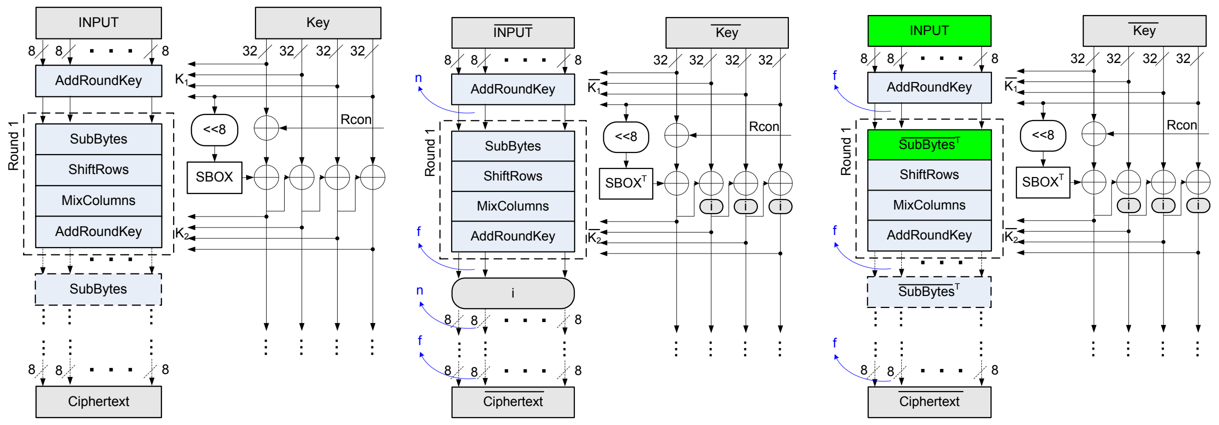


Fig. 3. AES Algorithm



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

Fig. 4. AES Algorithmic Balancing (images influenced by [46])

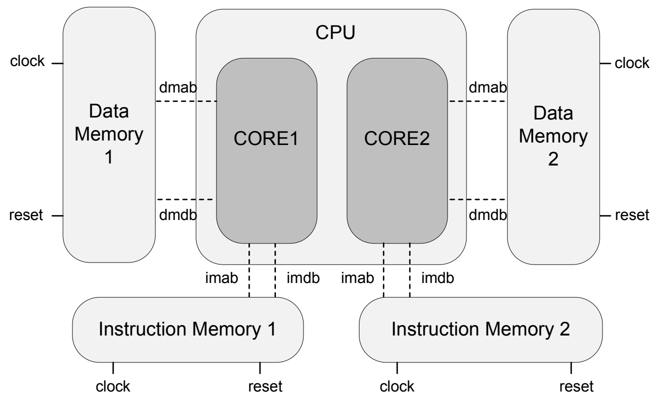


Fig. 5. The Base Dual Core Processor with Memory Modules

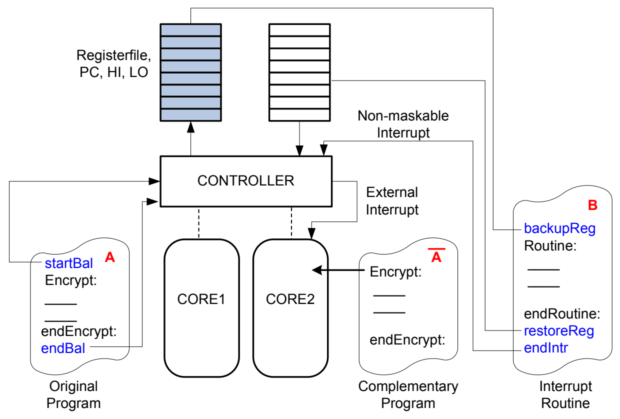


Fig. 6. Switching and Synchronizing

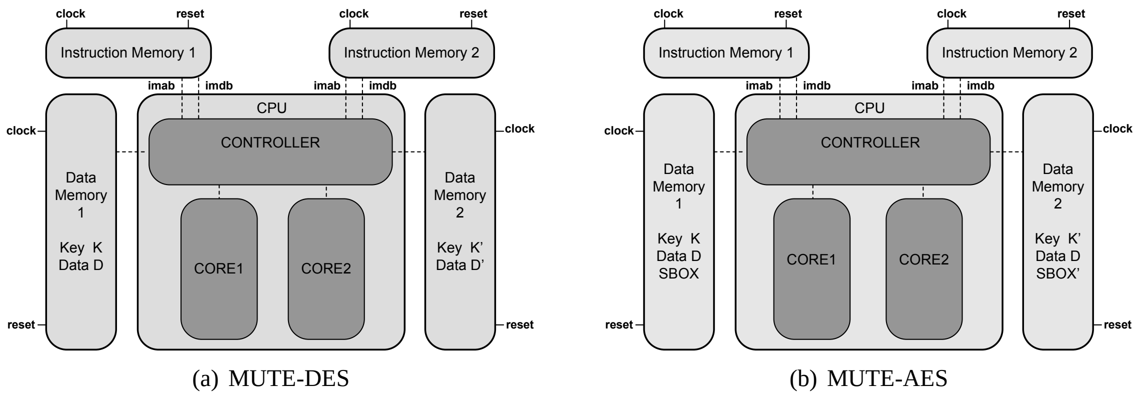


Fig. 7. Multiprocessor Balancing Architectures

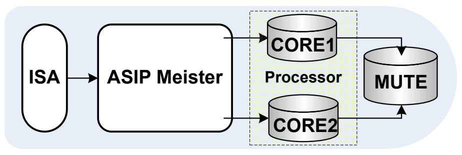


Fig. 8. Hardware Design

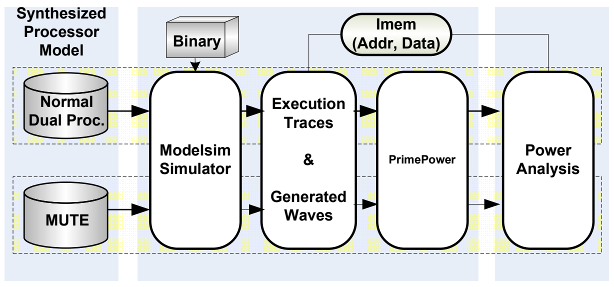


Fig. 9. Experimental Setup

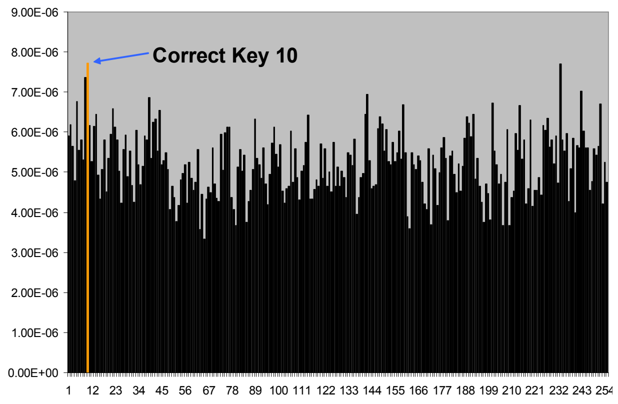


Fig. 10. DPA on a normal dual-core processor

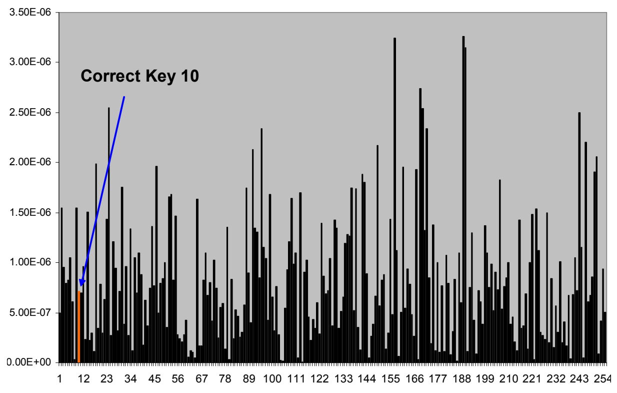


Fig. 11. DPA on MUTE-DES with inverted Key and inverted Data

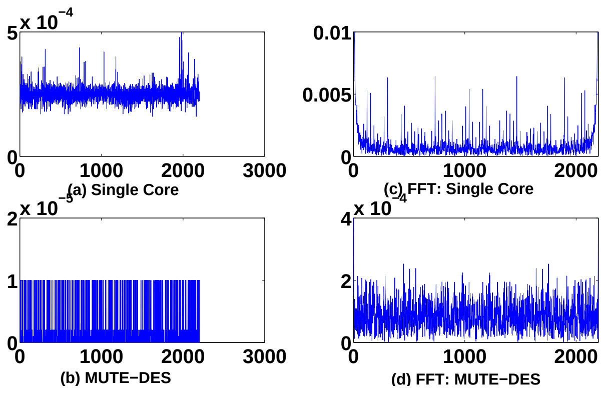
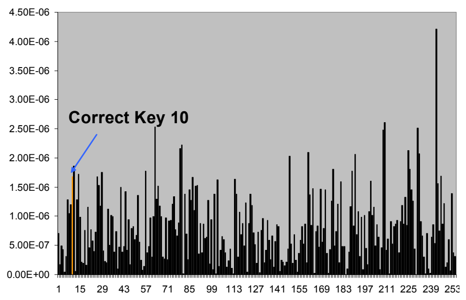
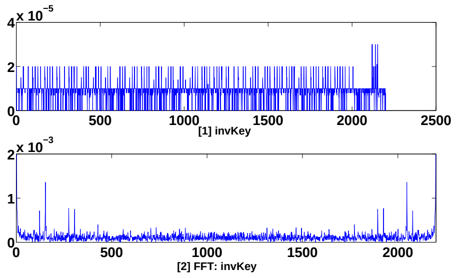


Fig. 12. Difference between power samples: FFT



(a) DPA on MUTE-DES



(b) Power Difference

Fig. 13. Inverted Key and Original Data

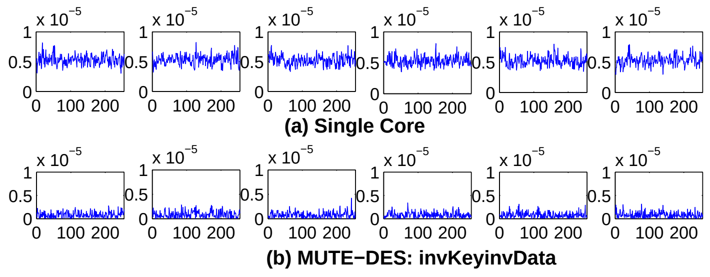
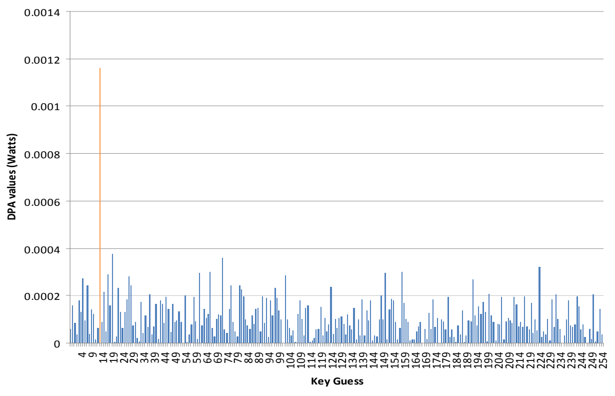
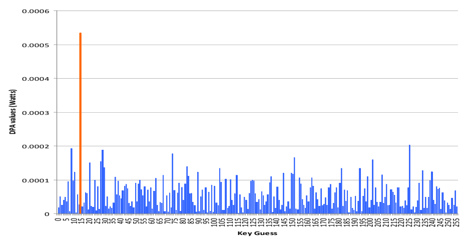


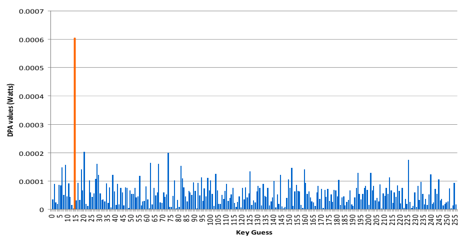
Fig. 14. DPA plots & Selection Bits



(a) @ LW

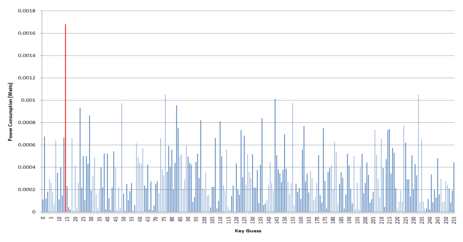


(b) @ XOR

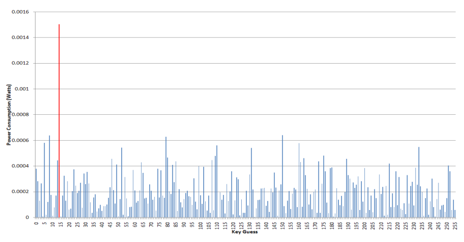


(c) Using Mean

Fig. 15. DPA plots on a single core in a standard dual-core processor: No Balancing

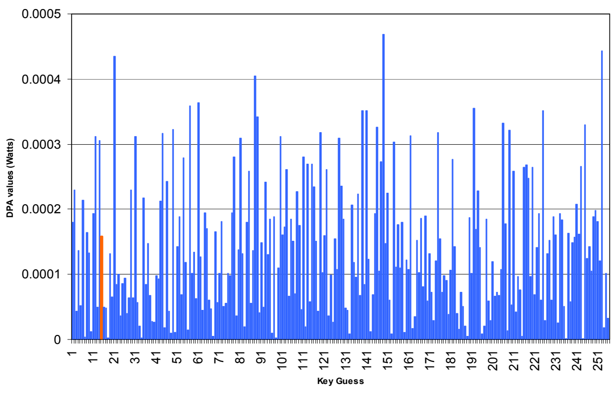


(a) @ LW

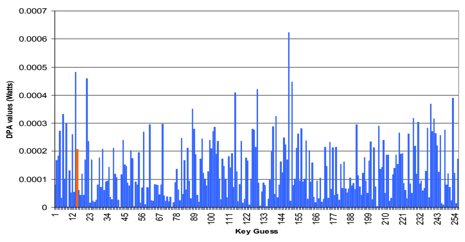


(b) @ XOR

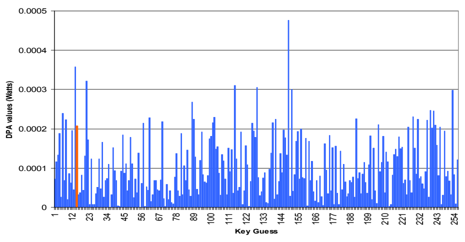
Fig. 16. DPA plots for Partial Balancing



(a) @ LW



(b) @ XOR



(c) Using Mean

Fig. 17. DPA plots for Complete Balancing: MUTE-AES

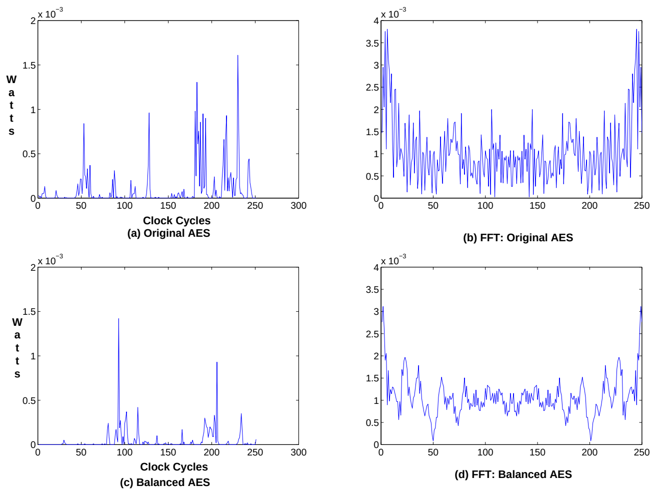


Fig. 18. FFT Analysis