Masking the Energy Behavior of DES Encryption

H. Saputra, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, R. Brooks[†], S. Kim and W. Zhang
Computer Science and Engineering, Applied Research Lab[†]
The Pennsylvania State University
Emails: {saputra, vijay, kandemir, mji, sookim, wzhang}@cse.psu.edu, rrb5@only.arl.psu.edu[†]

Abstract

Smart cards are vulnerable to both invasive and non-invasive attacks. Specifically, non-invasive attacks using power and timing measurements to extract the cryptographic key has drawn a lot of negative publicity for smart card usage. The power measurement techniques rely on the data-dependent energy behavior of the underlying system. Further, power analysis can be used to identify the specific portions of the program being executed to induce timing glitches that may in turn help to bypass key checking. Thus, it is important to mask the energy consumption when executing the encryption algorithms.

In this work, we augment the instruction set architecture of a simple five-stage pipelined smart card processor with secure instructions to mask the energy differences due to key-related data-dependent computations in DES encryption. The secure versions operate on the normal and complementary versions of the operands simultaneously to mask the energy variations due to value dependent operations. However, this incurs the penalty of increased overall energy consumption in the data-path components. Consequently, we employ secure versions of instructions only for critical operations; that is we use secure instructions selectively, as directed by an optimizing compiler. Using a cycle-accurate energy simulator, we demonstrate the effectiveness of this enhancement. Our approach achieves the energy masking of critical operations consuming 83% less energy as compared to existing approaches employing dual rail circuits.

1. INTRODUCTION

With increased usage of smart cards, the financial incentive for security attacks becomes attractive. For example, the smart card usage in North America surged by 37% in 2000, particularly in the financial segment where security is a prime issue. There are various classes of security attacks that can be broadly classified as: microprobing, software attacks, eavesdropping and fault generation [15]. Microprobing is an invasive technique that involves physical manipulation of the smart card circuit after opening the package. Software attacks focus on protocol or algorithm weaknesses, while eavesdropping techniques hack the secret keys and information by monitoring the power consumption, electromagnetic radiation and execution time. The fault generation techniques induce an intentional malfunction of the circuit by using techniques such as varying the supply voltage, inducing clock glitches, exposing the circuit to ionizing radiation, etc. Our focus in this work is on addressing the information leak due to eavesdropping power profile.

Power analysis attack is based on analyzing the power consumption of an operation. The main rationale behind this kind of an attack is that the power consumption of an operation depends on the inputs (in the case of cryptography, the inputs are plaintext and secret key). The differences in values of the operands being operated on result in different switching activities in the memory, buses, datapath units (adders, multipliers, logical units) and pipeline registers of the smart card processor. Among these components, the processor datapath and buses exhibit more data-dependent energy variation as compared to memory components [16].

There are different degrees of sophistication involved in such power analysis based attacks. Simple Power Analysis (SPA) [7] uses only a single power consumption trace for an operation. From this power trace, an attacker can identify the operations being performed (such as whether a branch at point p is taken or not or whether an exponentiation operation is performed). Using this power information and by knowing the underlying algorithm being implemented, such information can reveal the secret key. For example, when a branch is taken based on a particular bit of a secret key being zero, the attacker can identify this bit by monitoring the power consumption difference between a taken and not taken branch. Protecting against this type of simple attack can be achieved fairly easily by restructuring the code. For example, a restructured algorithm is provided in [3] to eliminate branch conditions that were initially revealing the secret key information. Also, techniques that randomly introduce noise into the power measurement can mislead simple power analysis. An example of such technique involves adding dummy modules and activating them at random time intervals. These modules will consume additional power skewing the original power profile. However, such techniques only provide protection from straightforward hacking techniques. Higher-order power analysis techniques can be used to circumvent these protection mechanisms.

Differential power analysis (DPA) is currently the most popular higher-order power analysis. This scheme utilizes power profiles gathered from several runs and relies on the data-dependent power consumption variation to break the key [7]. In [5], Goubin et al. show how the secret key is guessed by using 1000 sample inputs and their corresponding 1000 power consumption traces. Then, a mean of all these power consumption traces represented as *M* is obtained. Next, the hacker guesses a particular key and based on the input determines a theoretical value for one of the intermediate bits (*b*) generated by the program. Then, the outcome of this bit is used to separate the 1000 inputs into two groups (*G1* and *G2*) based on whether *b*=0 or *b*=1. If the mean of the power profiles in Group *G1* is significantly different from that of *M*, this indicates that the guess was correct. This difference is a manifestation of the consequent downstream computational power differences that used the bit *b*. As evident from the above discussion, random noises in power measurements can be filtered through the averaging process using a

* This material is based on work supported in part by the Office of Naval Research under Award No. N00014-01-1-0859, MARCO 98-DF-600 GSRC Award, NSF Awards No. 0093082, 0093085, 0082064, 0103583. Any opinions, findings, and conclusions or recommendations expressed in this presentation are those of the authors and do not necessarily reflect the views of the Office of Naval Research.

1530-1591/03 $17.00 © 2003 IEEE

large number of samples. However, the use of random noises can increase the number of samples to an infeasible number to be of practical concern.

In this work, we propose to counter DPA by modifying the underlying circuit to make its power consumption independent of the operand data using dual rail logic. However, the naïve use of such logic to mask the data dependent energy behavior to protect the secret keys has a significant energy penalty. Our goal is to limit the use of the additional components required for energy masking for selective operations. The key to identify the operations that require energy masking is an optimizing compiler. This compiler allows users to annotate certain critical variables. Whenever these critical variables or values derived from them are operated on, it is important to mask any data dependent energy behavior. This is because the energy profile can be used to reveal sensitive data (e.g., key values in encryption algorithms) without any energy masking. The compiler uses the secure instructions for all operations that directly operate on the annotated variables. Further, it uses a technique called forward slicing to identify other critical operations that depend on the annotated ones. Forward slicing is required to prevent indirect information leak from operations that do not operate on the sensitive variables directly.

We evaluate the proposed scheme focusing on the DES algorithm using a detailed cycle-accurate power analysis with the help of an energy estimation framework. Note that our approach is general and can be extended to other algorithms that need protection against current measurements based breaks. We measure the energy consumed in every cycle in picoJoules throughout the paper. The use of the simulator provides a far greater control of the granularity of information than would be practically possible for a hacker using physical power measurement tools. Thus, our approach is conservative and factors in possible improvements in measurement fidelity. Our experiments show that the proposed approach based on selective use of secure instructions achieves the energy masking of critical operations consuming 83% less energy as compared to existing approaches that apply masking to all operations due to the lack of any compiler analysis.

The remainder of this paper is organized as follows. In Section 2, we discuss related work. In Section 3, we revise the DES algorithm. In Section 4, we present our masking method. This section gives explanation about our secure instruction and our modified architecture. Finally, conclusions and a brief outline of future work are given in Section 5.

2. RELATED WORK

Various researchers have investigated the potential for both invasive and non-invasive attacks against smart cards. An overview of these techniques is presented in [4, 15]. In particular, the retrieval of secret information from a smart card through its leakage called side-channel analysis poses an important class of attack. Analyzing the power profile of an encryption process is one of the popular side-channel attacks. Kocher et al. [7] provide a detailed description of the simple power analysis (SPA) and differential power analysis (DPA) techniques. The difference between these two attacks is that DPA is more sophisticated and involves statistical analysis using a larger sample set.

There have been prior attempts to address the SPA and DPA attacks [2,3,5,6,7,12]. These counter measures can be classified into three types as performed in [5]. First, random timing shifts and noises can be added such that computed means for power consumption do not correspond to the same instruction. However, the difficulty in the protection process is to ensure such random techniques are not vulnerable to statistical treatment using large samples. While complete protection is difficult to achieve using such a counter measure, it could make it infeasible for an attacker to break the key. The second counter measure technique is to modify the underlying software implementation or algorithm [3, 4, 5]. For instance, the use of non-linear transformations of S-box operations is proposed in [5] to avoid some DPA attacks. However, it is observed in [4] that software counter measures may be difficult to design.

The third form of countermeasure involves replacing some of the critical instructions such that their energy consumption behavior does not leak any information. Our approach falls into this category. Dual-rail logic has been identified as one of the most promising ways to provide protection to SPA and DPA [4]. The use of dual-rail encoding and self-timed circuits is also proposed in [12]. The use of self-timed circuits can alleviate problems arising from clock glitches induced in synchronous circuits while the dual-rail logic masks the power consumption differences due to bit variations. Our contribution is the actual modification of an embedded processor to implement this technique and the addition of secure instructions to its instruction set. However, the use of dual-rail logic can increase overall power consumption by almost two times as observed in our experiments. In fact, power is an important constraint for smart card markets such as the GSM industry and specific constraints on maximum power are imposed by the GSM specification for different supply voltages [4].

3. DATA ENCRTYPTION STANDARD

Data Encryption Standard (DES) is one of many symmetric cryptography algorithms that are widely used. DES [1] uses 64 binary bits for the key, of which 56 bits are used in the algorithm while the other 8 bits are used for error detection. Similar to other cryptography algorithms, the security of the data depends on the security provided by the key used to encrypt or decrypt the data. Although the DES algorithm is public, one will not be able to decrypt the cipher unless they know the secret key used to encrypt that cipher.

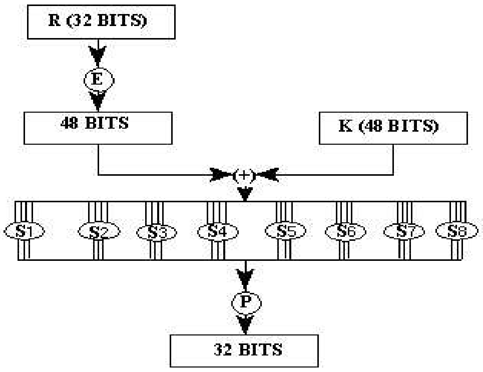

Figure 1. S-Box Operations (f(R,K)) [1]

The plaintext is permuted first before it goes to the encryption process. The core of the DES encryption process consists of 16 identical rounds, each of which has its own sub secret key, called Kn (n=1,2, ..16). Each Kn is 48 bits produced from the original secret key using key-permutation and shift operations. Each round consists of 8 S-Box operations which use the 48-bit input derived from the permutated input and sub secret key Kn (see Figure 1). Each S-box operation takes 6 bits from the 48-bit input that is divided into 8 blocks. These 6-bits are used to index a table. This table look up operation consequently generates 4 bits (for each S-Box). These 32 output bits (4 x 8 S-Boxes) are then used as an input combination for the next round.

The complete encryption process of DES is shown in Figure 2 (left side). Here, (+) denotes bit-by-bit addition modulo 2. From this algorithm, we can identify the operations that can reveal the secret key. These operations are key permutation, left-side and right-side assignment operations and key generation. Next, we need to identify the underlying instructions used to implement operations and consider their secure versions.

Data Initial Permutation (L0,R0) = PermuteIP(Data) **Key Permutation** (C0,D0) = PermuteK1(Key) *= denotes insecure assignment*	**Data Initial Permutation** (L0,R0) = PermuteIP(Data) **Key Permutation** (C0,D0) ← PermuteK1(Key) *← denotes secure assignment*

Mth Rounds
Left Side Operation
Lm = Rm-1
Mth Key Generation
Cm = Rotate(Cm-1,n)
Dm = Rotate(Dm-1,n)
Km =PermuteK2(Cm,Dm)
Right Side Operation
E(R)= PermuteE(Rm-1)
f(Rm-1,K) = S(E(R)(+) Km)
Rm = Lm-1 (+) f(Rm-1,K)
	Mth Rounds
	Left Side Operation
	Lm ← Rm-1
	Mth Key Generation
	Cm ← Rotate(Cm-1, n)
	Dm ← Rotate(Dm-1,n)
	Km ← PermuteK2(Cm,Dm)
	Right Side Operation
	E(R)← PermuteE(Rm-1)
	f(Rm-1,K) ← S(E(R) <+> Km)
	Rm ← Lm-1 <+> f(Rm-1,K)

Output Inverse Permutation Output=PermuteIP^{-1}(R16,L16)	**Output Inverse Permutation** Output=PermuteIP^{-1}(R16,L16)

Figure 2. Modified DES Algorithm

4. ENERGY MASKING FOR DES

4.1 Overview

Our energy masking approach is based on eliminating the input dependencies of an operation. Our approach is focused on four types of operations that are critical in the DES encryption: assignment operation, bit-by-bit addition modulo two (XOR) operation, shift operation and indexing operation. In our approach, we do not mask all the operations, but only the operations that use the secret key and those operations that use the data generated from prior secure operations. The compiler analyzes the code and identifies how these variables are used within the code. Then, for the operators that work on these variables, the compiler employs secure versions of the corresponding instructions. It should be emphasized that it is not sufficient to protect only the sensitive variables annotated by the programmer. This is because the variables whose values are determined based on the values of the protected variables can also be exploited to leak information. Consequently, such variables also need to be protected. We achieve this using a technique called forward slicing [11]. In forward slicing, given a set of variables and/or instructions (called *seeds*), the compiler determines all the variables/instructions whose values depend on the seeds. The complexity of this process is bounded by the number of edges of the control flow graph of the code being analyzed. After all the variables whose values are affected by the seeds are determined, the compiler uses secure instructions to protect them.

Figure 2 shows how we modified the DES operations. Figure 2(a) shows the original DES operations. The first step is initial permutation of the plaintext. This operation does not use any secret key and hence does not require being secure.

The next operation is the key permutation. This operation obviously needs to be secure. Figure 2(b) shows how we modify this operation. In this figure, the symbol "=" corresponds to the original assignment (i.e., insecure assignment), and the symbol "←" indicates that the assignment is secure.

The next step contains the operations within each round. Since some of these operations require the secret key, and the operations are repeated in every round using the data generated from the previous round, we need to secure all operations inside this block. Note that the modified left side operation uses a secure assignment operation, although it does not operate on the secret key directly. This is because it uses the data generated from the previous round (for ≥2nd round) that uses the secret key. In the right side operation, all the instructions need to be secure. Each round uses four types of secure operations: they are secure assignment, secure shift, secure bit-by-bit addition modulo two and secure indexing. Note that the S symbol in the figure represents the S-Box operation.

The last operation is the output inverse permutation. This operation does not need any secure instruction although it uses data generated from secure instructions as it reveals only the information already available from the output cipher. The following section explains how our secure instructions are implemented.

4.2 Implementation of Secure Instructions

Our target 32-bit embedded processor has five-pipeline stages (fetch, decode, execute, memory access and write back) and implements the integer instructions from the Simplescalar instruction set architecture [16]. Its ISA is representative of current embedded 32-bit RISC cores used in smart cards such as the ARM7-TDMI RISC core. We augment our target instruction set architecture with secure versions of select instructions. To support these secure operations, the hardware should be modified as explained below.

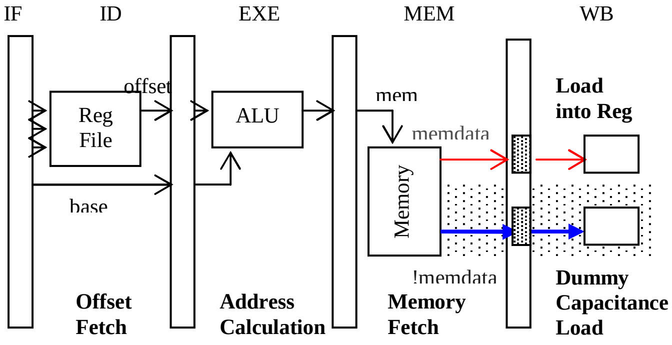

Figure 3. Secure Load Architecture (dotted portion is the augmented part)

First, we provide an overview of the underlying reasons for the differences in the power consumption because of data dependencies when executing these instructions. An assignment operation typically involves loading a variable and storing it into another variable. We will consider the parts of the load operation that are of interest. All stages of our pipeline (see Figure 3) till the memory access stage are independent of the loaded data (note that revealing the address of data is not considered as a problem). The memory access itself is not sensitive to the data being read due to the differential nature of the memory reads. However, the output data bus switching depends on the data being transmitted. For example, let us consider the different scenarios for the 1st bit (d0) of the 32-bit data read from the cache. If the values of d0 in two successive cycles are 0 and 1, it consumes more power than the case when the values are 0 and 0 in these two cycles. Specifically, for an internal wire of 1pF and a supply voltage of 2.5V, the first case consumes 6.25pJ more energy than the second case. The output from the memory access stage is fed to the pipeline register before being forwarded for storing the data in the register file. Thus, based on whether a bit value of one or zero is stored in the pipeline register bits, a different amount of energy is consumed. Finally, the energy consumed in writing to a register is independent of the data as the register file can be considered as another memory array.

The secure version of the load operation will need to mask all these energy differences due to bit dependences. This is achieved by the following modifications to the architecture. The buses carrying the data from a secure load are provided in both their normal and complementary forms. Thus, instead of a 32-bit bus, we use a 64-bit bus. Thus, the number of 1s and 0s transmitted in the bus will both be 32. However, this is not sufficient for masking the energy differences that depend on the number of transitions across the bus. But this modification along with a pre-charged bus can mask this difference. All the 64 bus lines are pre-charged to a value of one in the first phase of the clock. In the next evaluating phase, the bus settles to its actual value. Exactly, 32 of the bus lines will discharge to a value of zero. In subsequent cycles, energy is consumed only in pre-charging 32 lines independent of the input activity. The next modification involves propagating the normal and complementary values until the write back stage. The complementary values are terminated using a dummy capacitive load. The required enhancements to the underlying processor architecture are illustrated in Figure 3. Similarly, a secure version of the store operation involves passing along both the normal and complementary forms of the data read from the register file in the decode stage to the memory access stage.

```
………….
// Left Side Operation
for (i=0; i<32; i++)
         newL[i] = oldR[i]
……..
```

```
.................      .................
$L12:                  $L12:
  ...............        ...............
$L15:                  $L15:
  lw      $2,i           lw      $2,i
  ...............        ...............
  la      $4,newL        la      $4,newL
  addu    $3,$2,$4       addu    $3,$2,$4
  move    $2,$3          move    $2,$3
  lw      $3,i           lw      $3,i
  move    $4,$3          move    $4,$3
  sll     $3,$4,2        sll     $3,$4,2
  la      $4,oldR        la      $4,oldR
  addu    $3,$3,$4       addu    $3,$3,$4
  move    $4,$3          move    $4,$3
  lw      $3,0($4)  ⇨    slw     $3,0($4)
  sw      $3,0($2)       ssw     $3,0($2)
$L14:                  $L14:
  lw      $3,i           lw      $3,i
  addu    $2,$3,1        addu    $2,$3,1
  move    $3,$2          move    $3,$2
  sw      $3,i           sw      $3,i
  j       $L12           j       $L12
$L13:                  $L13:
.................      .................
```

(a) Original Assembly Code **(b) Modified Assembly Code**

Figure 4. Code level representation of the left side operation

A secure assignment uses a combination of both the secure load and the secure store to mask the energy behavior of the sensitive data. Figure 4 shows a specific elaboration of the use of the secure assignment in assembly code for the assignment performed during the "left side operation". The high-level assignment statement leads to a sequence of assembly instructions. The critical operations (the load and store instructions highlighted) whose energy behavior needs to be made data independent are then converted to secure versions in our implementation by the optimizing compiler.

The secure 32-bit XOR instruction is implemented using complementary pre-charged circuit (see Figure 5) that will ensure that for every XOR bit that discharges in the required circuit, the complementary circuit will not discharge and vice-versa. In the first clock phase (when v =0), all (64 = 32 original + 32 complementary) the output nodes of the XOR circuit are pre-charged to one. In the next phase (when v=1), half of them will discharge and the other half of them will remain at one. In subsequent cycles that use the XOR, the energy is consumed only for charging 32 output nodes immaterial of the data activity.

During the S-Box operation, a 6-bit value is used to index a table. This operation is performed by a load operation with the 6-bit value serving as the offset in our underlying architecture. Note that our current secure load operation does not mask the energy difference due to differences in the offset. As these 6-bits are derived from the key, it is also important to hide the value of this offset. When the 6-bit value is added as an offset to the base address of the table, the addition operation will consume an energy based on the 6-bit value. In order to avoid this, we align the base address of the table such that the 6-bit value serves as the least significant bits of the lookup and the most significant bits are determined at compile time. Further, the inverted value of this 6-bit index is propagated to mask the energy consumption. Thus, the load operations used for indexing are replaced by the secure indexing that generates the memory address using our secure version.

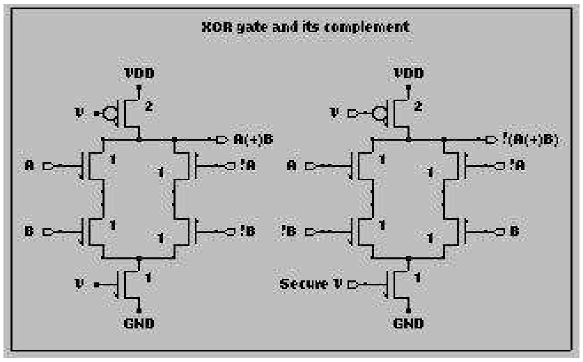

Figure 5. XOR circuit and its complement. v is the clock. A and B are the inputs to the XOR function

In order to utilize these augmented architectural features, the compiler tags selected operations as secure. Secure instructions can be implemented using either the unassigned opcodes (bits in the instruction identifying the operation) in the processor architecture or by augmenting the original opcodes with an additional secure bit per operand. In our implementation, we resort to the second option to minimize the impact on the decoding logic. Whenever a secure version of the instruction is identified both the normal and complementary versions of the appropriate segments of the processor become active. For example, for the secure XOR operations, the data values (both source data and result data) are present in normal and complementary forms in the internal data buses. Further, the required and complementary versions of the circuit operate together. Since the additional parts consume extra power, the clock to the complementary versions is gated to reduce energy consumption. The details of the gating (note that the complementary version of the circuit is provided with a clock v gated with secure signal – secure v - for the evaluation phase) for the XOR unit implementation are shown in Figure 5. Thus, as opposed to energy consumption of 0.06pJ in the secure mode, the XOR unit consumes only 0.03pJ in the normal mode. Additional savings in energy also accrue during the execution of normal versions due to gating of the additional buses and the pipeline registers.

4.3 Evaluation

To evaluate the effectiveness of our approach, we have implemented the DES algorithm in software and captured the energy consumption in each cycle using a customized version of the publicly available SimplePower [16], a cycle-accurate energy simulator. We focus only on the processor and buses in this work, as memory power consumption is largely data-independent. The simulator uses validated transition-sensitive energy models for both the buses and functional units obtained through detailed circuit simulation and is within 9% of actual values [16]. It is able to accurately capture the differences in energy consumption due to data transitions. The flexibility of working with the simulator provides us the ability to monitor the energy consumed in every cycle (along with details of actual instructions executed) and also helps us in quickly identifying the benefits or (otherwise) in modifying the underlying processor architecture. Current measurement based approaches would be limited by the sampling speed of the measuring devices and would also be more difficult to correlate the operations and sources of energy consumption. The processor modeled for our simulation results is based on 0.25micron technology using 2.5V supply voltage.

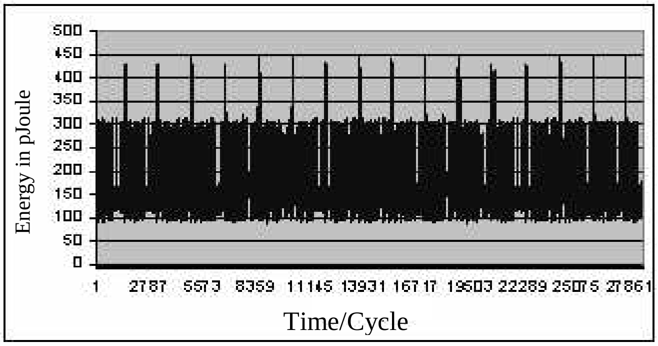

Figure 6. Energy consumption trace of encryption (every 10 cycles)

First, we show the energy behavior of the original DES algorithm to demonstrate the type of information that it leaks. Figure 6 shows the energy profile of the original encryption process revealing clearly the 16 rounds of operation. This result reiterates that the energy profile can show what operations are being performed. Next, we present a (differential) energy consumption trace for two different secret keys to demonstrate that the energy consumption profiles can reveal more specific information.

Figure 7 illustrates the difference in energy consumption profiles generated for two different secret keys using the same plaintext. This example illustrates that it is possible to identify differences in even a single bit of the secret key. Similar observations on energy differences can also be made using differences in one of key-related variables generated internally.

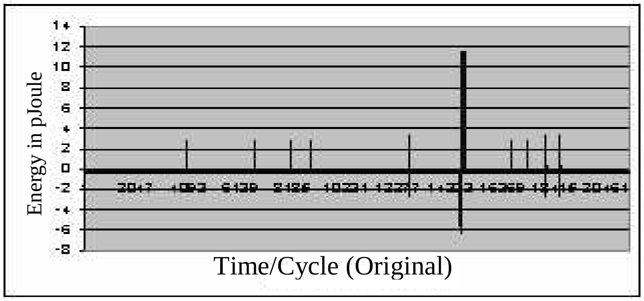

Figure 7. Difference between energy consumption profiles generated using two different secret keys (vary in bit 10), 1st round

Figures 8 and 9 show the difference between the two energy consumption traces generated using two different secret keys and the same plaintext before and after the energy masking. These traces are shown only for the first round of DES algorithm for clarity. The graphs clearly demonstrate that using secure instructions can mask the energy behavior of the key related operations. While the effectiveness of the algorithm is shown using differences between profiles generated from two different keys, the results hold good for other key choices as well. Specifically, the mean of the energy consumption traces which generate different internal (key related) bits will not exhibit any differences that can be exploited by DPA attacks.

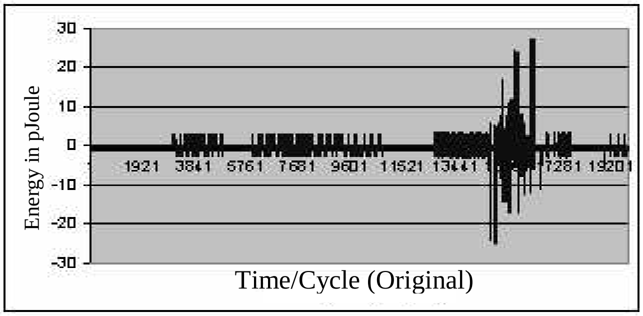

Figure 8. Difference between energy consumption profiles generated using two different keys before masking process

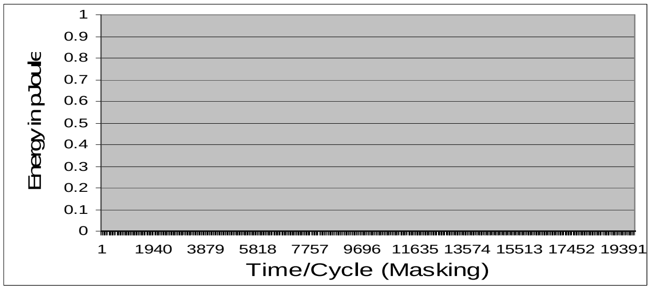

Figure 9. Difference between energy consumption profiles generated using two different keys after masking process

Figures 10 and 11 depict the difference between the energy consumption traces generated using two different plain texts but the same secret keys.

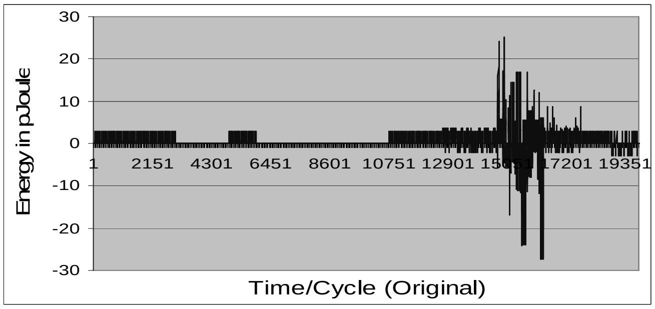

Figure 10. Difference between energy consumption profiles generated using two different plaintexts before masking process

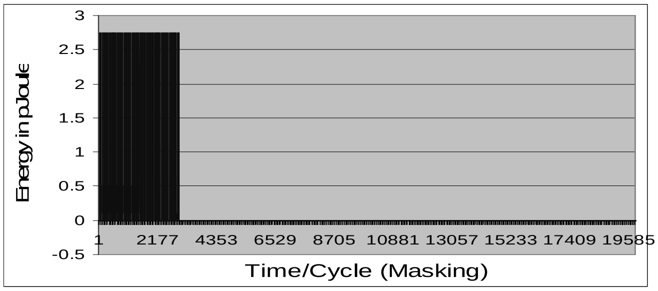

Figure 11. Difference between energy consumption generated using two different plaintexts after masking process

The first operation in the DES is plaintext permutation. Since this process is not operated in a secure mode, the differences in the input values result in the difference in both the energy masked and original versions. The other operations in the first round are secure; as a result, there are energy consumption power differences.

However, the proposed solution is not without its drawbacks. The energy masking requires that the same amount of energy be consumed independent of the data. Thus, additional energy is consumed in the circuits added for the complementary portion of the circuit as shown in Figure 12. However, this additional energy is 45 pJ per cycle (as compared to an average energy consumption of 165 pJ per cycle in the original application). Note that we add excessive energy even in places where the differential profile in Figure 8 shows no difference.

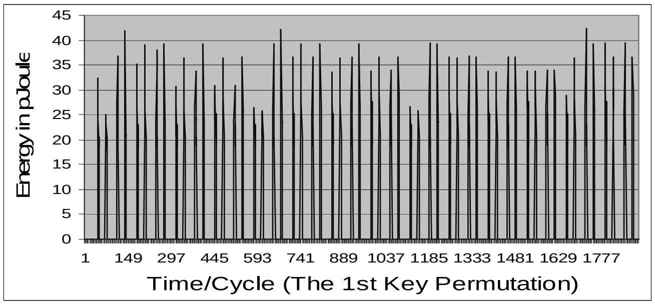

Figure 12. Additional energy consumed due to the energy masking operation during the 1st key permutation

This is because the same secure instruction is used for parts of the input that are the same for both the runs. Of course, in portions where the data was identical we have nothing to mask but we need to be conservative to account for all possible inputs in a statistical test using large samples. It must also be observed that our approach of using selective secure instructions helps to reduce the energy cost as compared to a naïve implementation that balances energy consumption of all operations. For example, looking at code segment shown earlier in Figure 4, we increase the energy cost of only one of the four load operations executed in the segment. On the other hand, the naïve approach would convert all the four load operations into secure loads thereby consuming significantly more energy than our strategy.

The total energy consumed without any masking operation is 46.4 uJoule. Our algorithm consumes 52.6 uJoule while the naïve approach consumes 63.6 uJoule (all loads and stores are secure instructions). When all instructions are secure instructions, it will consume almost as twice as much as the original, 83.5 uJoule. This scheme is the one used in current dual-rail solutions [4].

5. CONCLUSIONS

Smart cards, unlike magnetic stripe cards, can carry all necessary functions and information on the card. Therefore, recent years have witnessed a significant increase in smart card use throughout the world. In fact, Data Monitor predicts that over 3 billion smart cards are in circulation worldwide. As a result, ensuring secure use of smart cards is receiving a lot of attention.

In this paper, we focus on addressing the problem of information leakage using power analysis and propose a solution. The uniqueness of our solution comes from the fact that, unlike many previous techniques, we approach the problem from an architectural perspective and consider adding secure instructions to a given architecture. The purpose of these secure instructions is to hide the energy behavior of sensitive variables in the application (e.g., key values). Our experiments with the DES application demonstrate that the proposed solution is very effective in preventing the information leakage due to power analysis.

This work has certain limitations that need to be addressed. The use of complementary values and dual rail logic alone will not be sufficient in the future. This is because power consumption differences will also arise due to signal transitions on adjacent lines of on-chip buses [8]. Current dual-rail encoding schemes do not mask the key leakage arising due to these differences. Thus, more work is required in addressing power analysis attacks.

6. REFERENCES

[1] Data Encryption Standard (DES). Federal Information Processing Standards Publication 46-2. 1993 December 30.

[2] E. Biham, A. Shamir. Power Analysis of The Key Scheduling of The AES Candidates. *Proceedings of the second AES Candidate Conference*, March 1999, pp. 115-121.

[3] J. Coron. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. *Ç.K Koç and C. Paar, Eds., Cryptographic Hardware and Embedded Systems*, vol. 1717 of *Lecture Notes in Computer Science*, pp. 292-302, Springer-Verlag, 1999.

[4] J.-F. Dhem, N. Feyt. Hardware and Software Symbiosis Helps Smart card Evolution. IEEE Micro, Vol. 21, Issue. 6, pp. 14-25, November – December 2001.

[5] L. Goubin, J. Patarin. DES and Differential Power Analysis The "Duplication" Method. *Proceeding of CHES'99, Springer, Lecture Notes in Computer Science, Vol. 1717, August 1999*.

[6] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and other Systems, *Advances in Cryptology, Proceedings of Crypto'96*, LNCS 1109, N.Koblitz, Ed., Springer-Verlag, 1996, pp.104-113.

[7] P. Kocher, J. Jaffe, and B. Jun. Introduction to Differential Power Analysis and Related Attacks. *http://www.cryptography.com/dpa/technical,1998*.

[8] P. Sotiriadis, A.P. Chandrakasan. Low Power Bus Coding Techniques Considering Interwire Capacitance. *IEEE Custom Integrated Circuits Conference*, April 2000.

[9] P. Wayner. Code Breaker Cracks Smart Cards' Digital Save. *New York Times*, June 22 1998.

[10] Secure Hash Standard. Federal Information, Processing Standards Publication 180-1, April 17 1995.

[11] S. Horwitz, T. Reps, and D. Binkley. Interprocedural Slicing Using Dependence Graphs. *ACM Transactions on Programming Languages and Systems 12*, 1 (January 1990), 26-60.

[12] S. Moore, R. Anderson, and M. Kuhn. Improving Smart card Security Using Self-Timed Circuit Technology. *The Eight IEEE International Symposium on Asynchronous Circuit And Systems, Manchester, UK*, April 8 – April 11 2002.

[13] S. Skorobogatov, R. Anderson. Optical Fault Induction Attacks. *IEEE Symposium on Security and Privacy*, 2002.

[14] S.W. Smith, E.R. Palmer, S. Weingart. Using a High-Performance, Programmable Secure Coprocessor. *Proceedings of the Second International Conference on Financial Cryptography. Springer-Verlag Lecture Notes in computer Science, 1998*.

[15] O. Kommerling and M. G. Kuhn. Design Principles for Tamper-Resistant Smart card Processors. *USENIX Workshop on Smart card Technology, Chicago, IL*, May 10 – May 11 1999.

[16] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool". *Design Automation Conference, June 2000*.

[17] S. S. Muchnick. Advanced Compiler Design Implementation. Morgan Kaufmann Publishers, San Francisco, CA, 1997.