A Comprehensive Analysis of Performance and Side-Channel-Leakage of AES SBOX Implementations in Embedded Software

Ambuj Sinha
ECE Department
Virginia Tech
ambuj87@vt.edu

Zhimin Chen
ECE Department
Virginia Tech
chenzm@vt.edu

Patrick Schaumont
ECE Department
Virginia Tech
schaum@vt.edu

ABSTRACT

The Advanced Encryption Standard is used in almost every new embedded application that needs a symmetric-key cipher. In such embedded applications, high-performance as well as resistance against implementation attacks is mandatory. In this paper, we compare and contrast three different software implementations of AES. The first two are based on cryptographic lookup tables, while the third uses bit-slicing. We analyze the performance and side-channel resistance of each implementation on two different FPGA platforms, one based on a PowerPC processor, and the second based on a LEON-3 soft-core processor. Our measurements show that, on embedded platforms, a bit-sliced AES implementation does not always outperform a lookup-table based AES implementation. We also present a detailed analysis of the side-channel resistance and the source of side-channel leakage, and show that our bit-sliced implementation has eight times more side-channel leakage than the lookup-table implementations. Hence, we conclude that a variation on the implementation style for embedded software implementation of AES will not only affect performance, but also embedded system security.

General Terms

-

Keywords

Side Channel Attack (SCA), Bitsliced AES

1. INTRODUCTION

The Advanced Encryption Standard, which was selected by the National Institute of Standards and Technology in 2001, is the dominant symmetric-key algorithm in use today. The algorithm has been intensively studied, and a large amount of results with respect to performance and security have been published. However, often it is unclear how these different optimizations behave when compared to one another. For example, optimizing for performance may have an adverse effect on security, and vice versa. In this paper, we compare three different implementation techniques for the software implementation of AES, and we compare them under two orthogonal concerns: performance and (power-based) side-channel resistance. We show that such a comprehensive analysis provides a deeper understanding of the design space of embedded system security.

Our experiments are targeted to two 32-bit processor architectures, the LEON-3 (soft-core) processor [7], and the PowerPC 405 processor [17]. These 32-bit architectures are representative for typical embedded systems.

The AES implementations under study differ primarily in the way they perform cryptographic substitutions. Such substitutions are part of the substitute-and-permute network (SPN) kernel within AES. A software implementation can implement these substitutions in different ways. The most straightforward way is to use a lookup table, called an S-box, which is a byte-wide 256-element table. By optimizing the AES algorithm for execution on 32-bit architectures, the S-box can be merged with some of the permute-steps of the SPN network, resulting in a so-called T-box [5]. The T-box is a word-wide 256-element table. Besides the S-box and T-box lookup-table based formulation of AES, we also study a bit-sliced implementation AES. In bit-slicing, the individual bits of a word are computed separately, using bit-level operations [3]. In this case, the lookup tables need to be decomposed into elementary operations; this works for AES S-box because the input and output of the S-box are mathematically related. Indeed, the substitution can be expressed using finite field arithmetic operations [11].

Given these two processor platforms, and three different software implementations of AES, we analyze these designs from two perspectives: performance and power-based side-channel leakage. Performance is evaluated by measuring the execution time of the AES kernel. Power-based side-channel leakage is evaluated using differential power analysis (DPA) [10]. These experiments are performed on a side-channel analysis setup consisting of FPGA prototyping boards, a sampling oscilloscope, and post-processing software.

The main contributions of the paper are as follows. First, we show that bit-slicing on 32-bit platforms does not outperform a T-box implementation of AES. While it is assumed that bit-slicing requires a wide processor wordlength to be effective [13], this was never demonstrated. In contrast, on high-end processors with wide execution units (64-

WESS'10, October 24–29, 2010, Scottsdale, Arizona, USA.

bit or higher), bit-slicing is known to provide superior AES performance [9]. Second, we also demonstrate that memory-based table-lookups have smaller power-based side-channel leakage than the equivalent bit-sliced implementation. We attribute this to the fact that a single-step memory access performs the same amount of work that requires many different instructions for the bit-sliced implementation. Third, we describe several practical techniques to analyze the side-channel resistance of software implementations. We explain how to discriminate the correct secret key from the side-channel measurements. In addition, we present a technique to identify the most sensitive part of a software implementation.

In our evaluation, we did not consider side-channel resistant designs, such as masking or hiding [12]. Initially, we wanted to compare the algorithms by themselves, since that gave already an interesting design space. The exploration of side-channel resistant designs is a useful extension of this work, but it is outside of the scope of this paper.

The remainder of this paper is organized as follows. In the next section, we briefly review the implementation of AES using the S-box, T-box, and bit-sliced formulation on a 32-bit processor. We also discuss the current state of the art for AES implementations in software, considering performance as well as side-channel resistance. In Section 3, we describe the platforms that we have used for our experiments, and we present performance results for our three AES implementations in Section 4. Section 5 presents and in-depth discussion of the power-based side-channel analysis of these designs. We conclude the paper in Section 6.

2. THREE SOFTWARE IMPLEMENTATIONS OF AES

In this section, we describe the three different implementations of AES that we evaluated. We emphasize the dataflow of the algorithm, and we highlight specific optimizations for software.

2.1 Overview of AES

The Advanced Encryption Standard is well known, and detailed descriptions of the algorithm have been published [5]. Here, we briefly review the major steps in the algorithm. AES operates on a block of 128 bits at one time. It takes a block of 128 bits of plaintext and iterates that data through several rounds to produce ciphertext. Each round includes 4 transformations: SubBytes, ShiftRows, MixColumns, AddRoundKey. We refer to the literature for a description of these operations. AES organizes a block of 128 bit data as a 4x4 matrix of 16 bytes. Figure 1 illustrates the dataflow within a single AES round. The AES-128 algorithm, which is investigated in this paper, includes 10 such rounds.

2.2 AES S-Box and the AES T-box

Each of the *s*-operations in Figure 1 is a 256-entry substitution table. This substitution is mathematically defined: an S-box operation on an input byte a is found as a finite-field inversion (over $GF(2^8)$) followed by an affine transformation [5]. In each AES-128 round, there are 16 S-box substitutions. Such a formulation of AES leads to byte-wide computations over the entire round, which is inefficient on a 32-bit processor.

The S-box is therefore frequently implemented as a T-box, which includes each of the shaded operations in Figure 1. A T-box transforms one byte into four bytes, each of which represent part of a MixColumn result. In each AES round, four T-box operations can be combined to obtain a single row of the AES state matrix. Due to the detailed implementation of MixColumn, this approach requires four different T-box lookup tables. However, this approach provides a significantly better utilization of the 32-bit datapath of a processor.

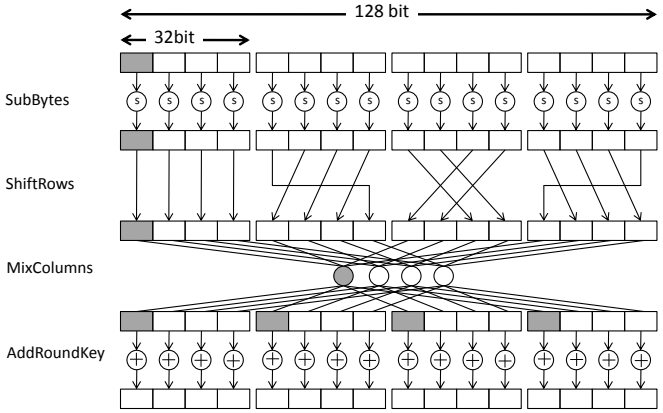

Figure 1: Dataflow in AES. A single T-tbox operation includes the shaded operations.

2.3 Bit-sliced SBOX

The concept of bit-slicing for symmetric key ciphers was first inroduced by Biham et al. for the DES algorithm [3]. In bit-slicing, an algorithm is broken down into bit-level computations (AND, OR, NOT, XOR) and each of these bit-level operations is implemented as a machine instruction. For example, a bit-level AND operation will be implemented with an AND instruction. Since the wordlength of a processor is n bits (e.g. 32 bits), this implies that n parallel copies of the same bit-level AND operation can be evaluated in parallel. Hence, using bit-slicing, n parallel copies of the algorithm can be evaluated in parallel. The suitability of an algorithm for bit-slicing depends on the *hardware-level* complexity of the algorithm. Essentially, bitslicing treats an algorithm as netlist of combinational logic, and simulates this netlist using processor instructions.

Bit-slicing of AES has been discussed extensively in literature [11, 3, 13, 9, 14, 16]. There are two motivations for it: bitslicing improves performances, and it provides timing based side-channel resistance [15]. The latter property is achieved when lookup tables (memory-load operations) are replaced with computations (logical operations). In that case, data-dependent cache-behavior is avoided, resulting in AES constant-time execution. In this paper however, we are investigating power-based side-channel resistance, which is not prevented by bitslicing.

Figure 2 illustrates two approaches to bitslicing of the AES state. In state-level bitslicing, each bit of the AES state is mapped into a different register [13]. 32 AES states can then be represented in 128 processor registers. This approach suffers from three drawbacks. First, it requires a large amount of processor registers, and may result in spilling, which reduces efficiency. Second, when less then 32 AES blocks need to be encrypted or decrypted, this method results in suboptimal processor utilization [2]. Finally, conversion of data

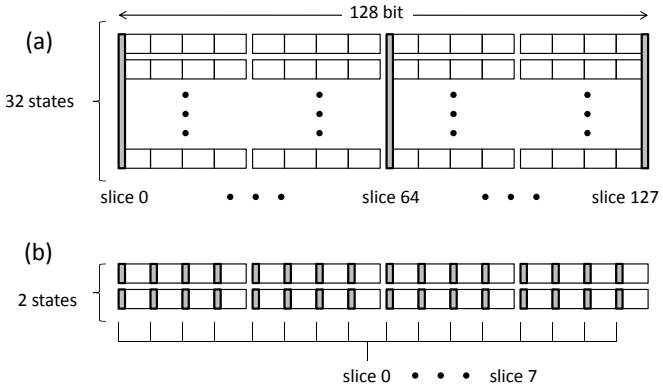

Figure 2: Bitslicing of AES. (a) State-level bitslicing allocates each bit of an AES state to a different processor register, mapping 32 states in 128 registers. (b) Byte-level bitslicing allocates each bit of a byte to a different processor register, mapping 2 states in 8 registers.

blocks into bitsliced format and back requires additional overhead. However, the method results in an encryption speed of 9.2 cycles per byte on a Core2 processor [14].

Byte-level bitslicing, illustrated in Figure 2b, takes a more compact approach. The corresponding bits of each byte in two AES states (32 bytes in total) are mapped into the bits of a register [9]. Eight 32-bit processor registers thus can capture two entire AES states. In contrast to the previous method, this method needs a specific bit-level organization in order to obtain an efficient implementation of the AES permutation operations (such as ShiftRows and MixColumns). Nevertheless, on a Core2 processor, a byte-level bitsliced AES was shown to execute at 7.59 cycles/byte [9].

For our experiments, we implemented a byte-level bitsliced version of AES.

3. PLATFORMS AND SCA SETUP

In this section, we describe two embedded platforms on which the 3 different AES implementations are built. After that, we discuss the side-channel analysis setup on these two platforms.

3.1 Platforms

We chose two different platforms. The first platform is the SASEBO-G board [1], which contains a Virtex 2 pro FPGA (XC2VP30-5FG676C) with two embedded ASIC 32-bit PowerPC 405 RISC processor blocks inside. AES software is implemented on one PowerPC processor. In the rest of this paper, we use `PPC` to represent this platform. Figure 3a illustrates the first platform.

The second platform is the Digilent Spartan 3E-1600 Development Board [6] with a Spartan 3E-1600 FPGA (S3E1600). The AES software runs on a 32-bit Leon3 SPARC V8 processor [7] built with the FPGA resources. We use `LEON3` to represent this platform. Figure 3b gives an overview of the second platform.

The above two platforms are different in several major features. The `PPC` platform supports both the software's instructions and data to be stored in on-chip memories, as shown in Figure 3a. The PowerPC processor and the on-chip

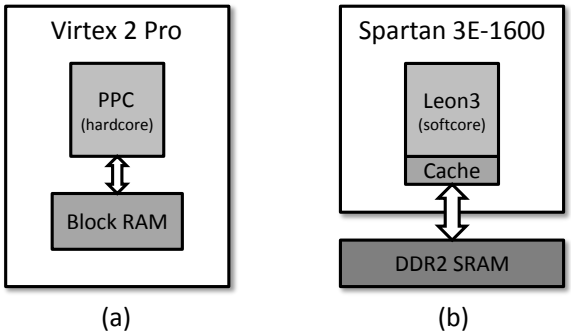

Figure 3: Embedded platforms: (a) PowerPC hardcore embedded in Virtex 2 Pro FPGA; (b) Leon3 softcore implemented in Spartan 3E-1600 FPGA.

Table 1: Differences between two platforms

`PPC`		`LEON3`
ASIC	processor	**FPGA**
NO	cache	**YES**
on-chip	code location	**off-chip**
on-chip	data location	**off-chip**
constant	exec. time	**data-dependent**

memory are clocked at the same speed of 24 Mhz. Both instructions and data can be read or written in one clock cycle. In this case, there is no need to implement a cache. From an architectural perspective, the `PPC` platform enables the software to run at a fastest speed. In addition, the execution time of each AES implementation can be a constant.

The `LEON3` platform uses the FPGA resources to implement the Leon3 processor, clocked at 50 Mhz. The instructions and data of the software are stored in an external DDR2 SRAM memory. Due to the lower speed of the external memory, both instruction cache and data cache are implemented within the FPGA. Figure 3b depicts the architecture of the second platform. Due to the cache effects, execution time of each AES implementation is data-dependent. Table 1 summarizes the differences of these two platforms.

The AES implementations follow the same design flow in both platforms. All 3 AES implementations are designed as 3 functions and are integrated to the testing program (Algorithm 1), shown in Section 3.2. The S-box and T-box based implementations are both designed in C, while the bit-sliced implementation is written in Assembly. The pure-C designs can be simply compiled with 'powerpc-eabi-gcc' and 'sparc-elf-gcc'. When compiling the Bit-sliced AES, we link the hand-written assembly to the object file and obtain the executable. At any of the above stages, the compilers' optimization option is '-O2'.

3.2 SCA setup

The side-channel analysis setup is the same for both platforms. Hence, we only use `PPC` platform as an example for explanation. The setup contains the `PPC` platform, an oscilloscope (Agilent DSO5032A) and a PC, shown in Figure 4.

The three parts of the setup are connected in a circular fashion. A RS232 cable connects the `PPC` platform and the

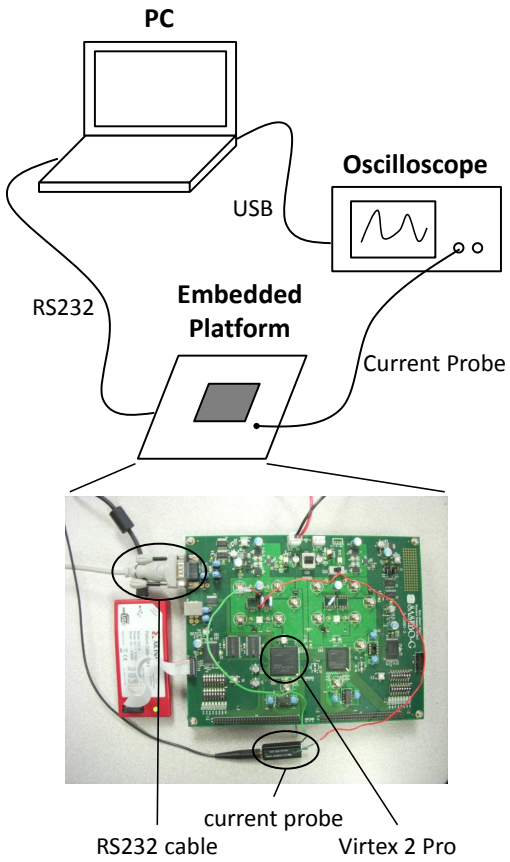

Figure 4: Side-channel attack system setup.

PC. Between the oscilloscope and the PC is a USB cable, through which the PC is able to send commands to and get sampling waveform from the oscilloscope. The oscilloscope uses a current probe (Tektronix CT-2) to monitor the current flowing into the PPC platform. The sampling rate and precision are set to be the same for all 3 different AES implementations. We use the current to represent the power consumption of the embedded system. Side-channel analysis requires a number of measurements with different inputs (plaintexts for AES). In our experiment, the result of one measurement is the current trace of the PPC platform and the corresponding random plaintext block for encryption. Each measurement consists of the following 4 steps. A side-channel analysis that requires n measurements needs to repeat these 4 steps for n times.

- Step 1: The PC sends a random plaintext block (16 bytes) to the embedded platform through the RS232 cable.
- Step 2: The PowerPC processor in the platform receives the plantext and encrypts it with the AES software. The program running on the PowerPC processor is shown in Algorithm 1. *Cipher* is one of the 3 AES implementations.
- Step 3: After sending out one block of plaintext, the PC sends command to the oscilloscope to sample the current trace when PowerPC is running the encryption.
- Step 4: After sampling is done, one current trace is sent back to PC for side-channel analysis.

Algorithm 1 Program running on PowerPC

```
exp_key ⇐ KeyExpansion(key)
if Cipher is bit-sliced AES then
  exp_key ⇐ BitsliceConvertion(exp_key)
end if
loop
  if UART is not empty then
    wait until 16 bytes received
    in[15 : 0] ⇐ UART_FIFO
  end if
  out[15 : 0] ⇐ Cipher(in, exp_key)
end loop
```

4. PERFORMANCE ANALYSIS

In this section we present a performance based comparison between the three different AES implementations.

Table 2 compares the three implementations based on latency, throughput, and memory footprint. In terms of speed, T-box AES is by far the best implementation on both the platforms. It has a slightly large memory footprint because of the large lookup table that it uses (4 kB).

The performance of bit-sliced AES is second best on the PowerPC. On a 32-bit processor, the bit-sliced implementation processes two AES blocks at once. Hence, the effective execution time for encryption is half the latency. Bit-sliced AES incurs a large code footprint on both the platforms. This is because the different functions like SubBytes and ShiftRows, which are simple loads and stores from memory in a standard implementation, are expanded into a much larger set of instructions in the bit-sliced domain. It can also be seen that the performance of the bit-sliced implementation is comparatively worse on the Leon3 than on the PowerPC. This can possibly be attributed to the large code footprint of bit-sliced AES. Because of large code size, instructions are not present in the instruction cache, and have to be fetched frequently from main memory. This is not a problem on the PowerPC, because all the code resides on the on-chip memory.

The standard S-box implementation has the lowest throughput of the three implementations. However, it has a smaller code footprint as compared to the other two implementations.

It can be seen from these results that, in terms of performance, T-box AES is one of the best choices for implementation on embedded cores. Though bit-sliced AES has made speed records on other platforms [9], it may not be the best option to use on 32-bit embedded processors. As argued by Matsui [13], bit-sliced implementations can offer an advantage when the following conditions are met: the target processor has a large number of registers, and the register length of the target processor is long. These conditions may not always be available on embedded platforms. In addition, bit-slicing benefits greatly from more complex instructions present on high end general purpose cores (eg. SSE3 and

Table 2: Performance Summary

	PPC (24MHz)			Leon3 (50MHz)		
	S-box	T-box	Bitsliced	S-box	T-box	Bitsliced
latency (μs)	310	60	284	210	30	440
throughput (kbits/s)	412	2133	450*2=900	609	4266	290*2=580
footprint (kB)	1.9	5.2	7.0	1.0	5.3	11.2

SSE4 ISE [8]). These instructions are generally not present on embedded platforms.

5. SIDE-CHANNEL ANALYSIS

In this section, we describe our experiments to evaluate and analyze the side-channel leakage of the three AES implementations under consideration. We are using differential power analysis (DPA) for our side-channel analysis. In this technique, power measurements are compared with a hypothetical power model. We discuss two important aspects in performing a practical DPA. The first is the selection of the power model, and the second is the definition of the attack success metric (measurements to disclosure, MTD). Next, we present the DPA results for the three AES implementation. Finally, we perform a refined analysis, in order to precisely locate the source of side-channel leakage for the chosen power model.

5.1 Power Model Selection for DPA

The objective of DPA is to compare measurements, obtained from a prototype, with power estimates, generated using a power model. The power model is chosen so that it has a dependency on a part of the secret key. For example, in the first round of AES encryption, the output of the substitution step is given as follows.

$$out[i] = subbytes(in[i] \oplus key[i]); \quad (1)$$

where $in[i]$ is the i-th byte of the 128-bit plaintext, $key[i]$ is the i-th byte of the 128-bit key, and $out[i]$ is the i-th byte of the 128-bit AES state. In this formula, in is known, while key and out are unknown. However, out is indirectly observable through the power consumption of the algorithm. Hence, we can create a hypothesis test for a key byte, using a power model for out. By making a guess for the i-th key byte, we can infer the value of $out[i]$. The power model used in the DPA is an estimate for the power consumption of the hypothesized $out[i]$. The actual hypothesis test, explained in the next section, will compare the estimated power consumption with the measured power consumption to identify the most likely key byte hypothesis. In this section, we discuss the selection of the power model.

In CMOS technology, where power is consumed as a result of state transitions, it is common to choose the Hamming Distance (the toggle count) as a power model [12]. For example, when performing power analysis on a processor, we could use the transitions of a processor register. However, in a software implementation of AES, we do not know what other data will share the same register location as $out[i]$. As a result, the Hamming Distance model is of limited use. Instead, we use the Hamming Weight (the bit count) as the power model.

A second difficulty in selecting the power model is that a software implementation of AES typically may take many instructions. As a result, the power trace of an AES encryption may cover several hundred clock cycles. We cannot predict, in general, at which clock cycle $out[i]$ will be computed. To address this issue, we simply extend our power model over all the clock cycles of a power trace, by repeatedly using the Hamming Weight of $out[i]$ for every clock cycle that we wish to analyze.

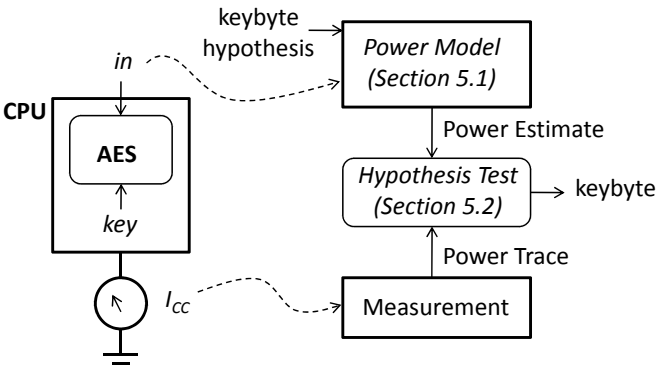

Figure 5: Differential Power Analysis: Power Model and Hypothesis Test

Finally, when attacking an unknown AES implementation, the specific variation of the algorithm that is executing is unknown to an attacker. In this paper, we compare three common implementations of AES (S-box based, T-box based, and bit-sliced). The S-box based and bit-sliced implementation produce the intermediate value $out[i]$ as specified above, and therefore the Hamming Weight of $out[i]$ is a valid power model. The T-box based implementation, on the other hand, does not have such an intermediate value, because it collapses the S-box substitution with several other AES operations. For a T-box, a more appropriate power model is one that is based on a T-box lookup table. This raises the obvious question how an attacker can select the power model. In practice, this is a minor issue. An attacker can simply try out multiple power models, and select the one for which the strongest hypothesis conclusions can be made (Section 5.2). Even 8 possible implementations for AES would not add more than 3 bit of uncertainty, and require 8 times more hypothesis tests. Therefore, in our tests, we used the most appropriate power consumption model for each implementation.

5.2 MTD: Quantifying side-channel leakage

Unlike the current or voltage, the side-channel leakage is not directly measurable. Therefore, researchers usually use indirect approaches to quantify the side-channel leakage, for example to use the attacking effort needed for successful attacks to reflect the amount of side-channel leakage. One of the commonly used approaches is called *Measurements to Disclosure* (MTD). The basic idea is that the more mea-

surements that are required to successfully attack a cryptographic design, the more secure that design is. From a statistical point of view, the more measurements means the more samples. Since the side-channel analysis is basically a statistical process, more samples usually lead to more accurate result.

This paper uses the same approach to compare 3 different AES software implementations. In particular, we mount a *Correlation Power Attack* (CPA) [4] on each of the implementations. With the same amount of measurements, the less key bytes are discovered from a design, the more secure that design is. In the rest of this section, we discuss our method to quantify the side-channel leakage based on MTD and CPA.

Suppose we have a set of random plaintext blocks ($pt[1 \dots n]$). Each of them is used for one measurement. Correspondingly, we obtain n power traces, each of which contain m sampling points ($tr[1 \dots n][1 \dots m]$). In our experiment, the CPA takes pt and tr as inputs, and discovers AES's key byte by byte by focusing on the first round of AES. The details are presented in Algorithm 2. The basic process is to take $pt[1 \dots n]$ and a guess value of a key byte (key_guess) to calculate an intermediate value of the AES software $iv[1 \dots n]$ ($iv[i] = f(key\_guess, pt[i])$). Sampling a complete power trace guarantees that the operations on the intermediate value occur during the sampling process. We use the Hamming weight of iv ($iv\_hw[1 \dots n]$) to approximate the power dissipated by iv. Then we calculate the correlation coefficient of iv_hw and the actual measured power traces at each sampling point. As a result, we obtain a correlation coefficient trace $corr[1 \dots m]$ ($corr[i]$ is the correlation coefficient of $iv\_hw[1 \dots n]$ and $tr[1 \dots n][i]$). By now, we have obtained one coefficient trace $corr[1 \dots m]$ which is corresponding to one guess value of the key byte. Since there are 256 possible values for one byte of key, we can, therefore, obtain 256 coefficient traces. By grouping these coefficient traces together, we are able to identify one coefficient trace from all the other 255 traces. In particular, at some points (when the operations on the intermediate value occur), the abstract value of the coefficient trace corresponding to the correct key guess is much larger than all the other traces. Figure 6 gives an example of the correlation coefficient traces. We can see that around time 100μs, one trace emerged from all the other traces. And it turned out that the key guess corresponding to this trace is the correct key, which means the CPA on this key byte is successful.

Algorithm 2 Correlation power attack on one key byte.

Require: $pt[1 \dots n]$ contains the random plaintext for encryption; $tr[1 \dots n][1 \dots m]$ contains the sampled power traces; f maps the inputs to the intermediate value iv; g calculates the correlation coefficient.
Ensure: $key\_gap[j]$ is the CPA-attacked key byte.

```
/* Obtain correlation coefficient traces corr */
for key_guess = 0 to 255 do
  for i = 1 to n do
    iv[i] = f(key_guess, pt[i])
    iv_hw[i] = HammingWeight(iv[i])
  end for
  for i = 1 to m do
    corr[key_guess][i] = g(iv_hw[1...n], tr[i][1...n])
  end for
end for
/* Find the correct key byte */
for i = 1 to m do
  find |corr[key1][i]| = max(|corr[0...255][i]|)
  find |corr[key2][i]| = second_max(|corr[0...255][i]|)
  gap[i] = corr[key1][i] − corr[key2][i]
  key_gap[i] = key1
end for
find gap[j] = max(gap[1...m])
return key_gap[j] as the correct key byte
```

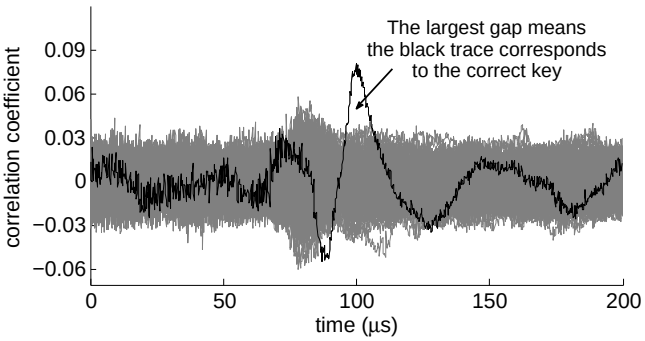

Figure 6: An example of 256 correlation coefficient traces. Around time 100us, the black trace which corresponds to the correct key byte emerges from all the other 255 traces.

5.3 SCA Results

Table 3 presents a comparison between three AES implementations in terms of resistance to side channel attack.

An attack was mounted on all the three implementations of AES on the two platforms (`PPC` and `Leon3`). The attacks were based on the techniques described previously.

It can be clearly observed from the table that the bit-sliced implementation is the weakest in terms of side channel resistance. Using the S-box implementation as a reference, the bit-sliced version requires 5-8 times fewer power traces to find all bytes of the key. The T-box implementation is strongest, requiring maximum number of traces to break the key. This is expected, because the T-Box implementation combines SubBytes, ShiftRows and part of MixColums into one highly compressed lookup table. This means that sensitive data, that can be attacked using a power analysis attack, is present in the system for a very short time as compared to the other two implementations. Figures 7 and 8 display correlation coefficient plots for the three implementations on both the platforms. It can be seen that the correlation peak for bit-sliced AES is higher than those for S-box and T-box based implementations.

A significant fact can be observed from the data in table 3. Among the two platforms, implementations on the `Leon3` are much more easier to attack as compared to those on the `PPC`. A standard S-box implementation required 40960 power traces to break completely on the `PPC`, while this number for the `Leon3` was 6400 (an increase by a factor of 6.4). Similar trend was observable for the other implementations as well. The same observation can be made by comparing the correlation coefficient plots for the different AES implementations on the two platforms. For the same number of

Table 3: Attack Results Summary - Showing number of bytes discovered

PPC				Leon3			
Measurements	Bitsliced	S-box	T-box	Measurements	Bitsliced	S-box	T-box
2048	13	2	2	512	9	0	0
5120	16	4	4	768	12	1	0
10240	16	8	6	1024	14	4	0
25600	16	11	8	1280	16	8	0
40960	16	16	12	5120	16	14	0
51200	16	16	13	6400	16	16	0

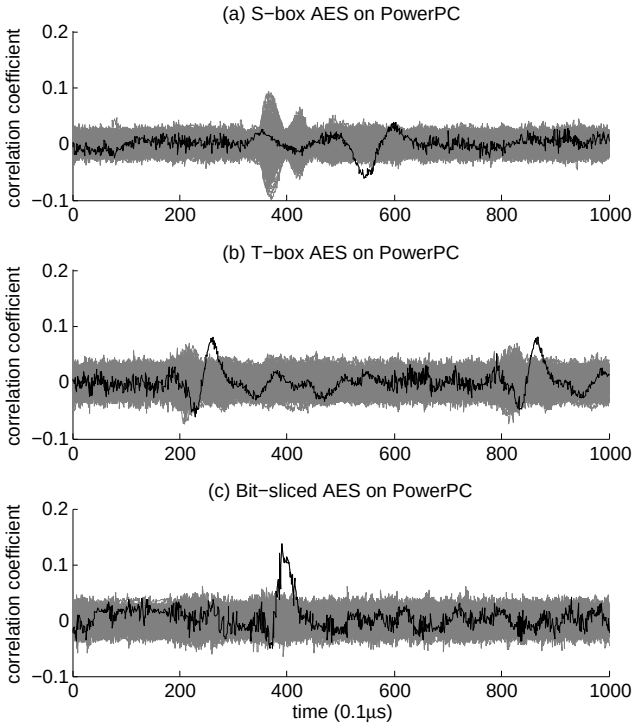

Figure 7: Correlation Coefficient plots for Side-channel Attack (number of measurements = 5120) on three different AES implementations on PPC platform.

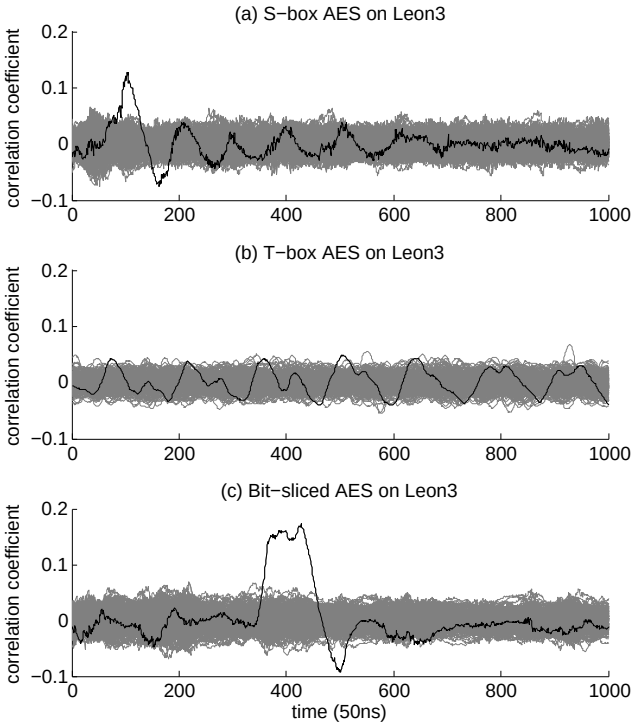

Figure 8: Correlation Coefficient plots for Side-channel Attack (number of measurements = 5120) on three different AES implementations on Leon3 platform.

measurements, the correlation peak for implementations on the Leon3 are larger than their counterparts for PPC. This shows that FPGA based soft-cores leak much more sensitive information as compared to ASIC cores.

5.3.1 The Effect of Cache

It can be seen from table 3 that the T-Box implementation cannot be broken on the Leon3. The correlation coefficient plot for T-box AES on Leon3 (figure 8) also does not show any clear peak which can be identified as the correct key. The reason for this is as follows. While mounting an attack on the T-Box implementation on the Leon3, our first observation was that the waveform was highly unstable and noisy, even when the beginning of every encryption was aligned. We tried to attack this particular waveform but were unsuccessful. We found that the cache in the Leon3 was causing such behavior. Because of the large lookup tables used by the T-Box, the effect of cache becomes noticeable, and the execution time for different encryptions differs depending on the plaintext. We verified this by disabling the cache on the Leon3 and doing the attack again. With the cache tunrned off, a very stable waveform was observed with constant execution time, and the T-box implementation could be easily broken.

This particular effect is absent in the PPC because it does not have cache and uses on-chip memory instead.

5.4 Location Analysis

Finally, we describe a technique that helps to identify which part of the algorithm in software is most sensitive to a power analysis attack.

We first identify which part of the algorithm corresponds to which part of the power trace. This can be done by inserting a series of 'nop' instructions at specific points in the program. The 'nop' instructions have a power profile which is easily distinguishable on the power trace. Thus,

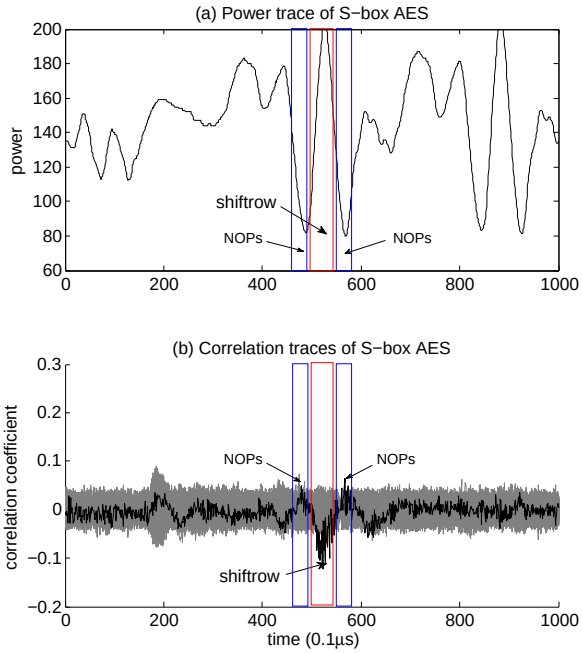

Figure 9: (a) Power trace of S-box AES on PPC platform; (b) Correlation coefficient plots for Side-channel Attack on S-box AES on PPC platform.

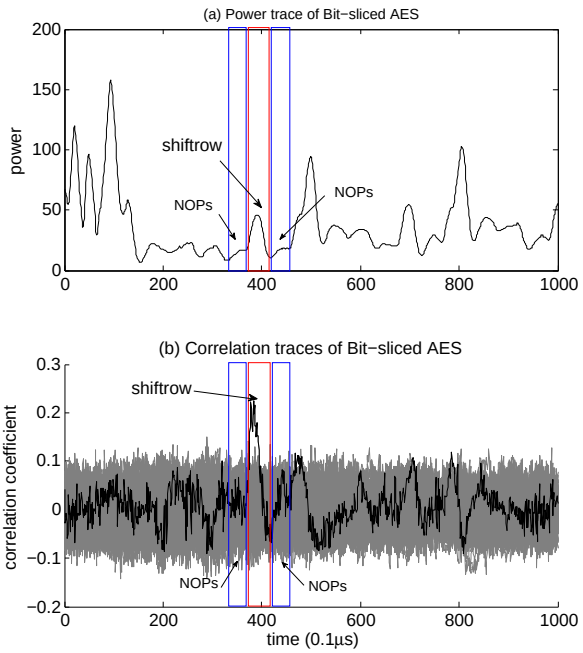

Figure 10: (a) Power trace of Bit-sliced AES on PPC platform; (b) Correlation coefficient plots for Side-channel Attack on Bit-sliced AES on PPC platform.

by inserting 'nops' at appropriate places in the program, we can separate different parts of the algorithm from one another in the power trace. Next, we perform an attack on the algorithm with these 'nop' instructions present in the program. The sensitive points in the algorithm can now be identified by seeing where in the power trace a correlation peak occurs.

For the S-box implementation, the correlation peak was found to be occurring around the ShiftRows function of the first round. Figure 9 shows the correlation plot along with the power trace for the S-box implementation on the PPC. The ShiftRows function has been isolated from other parts of the algorithm by inserting a set of 'nops' before and after it. The 'nops' are easily distinguishable in the power trace as large drops in the power. The peak between the two drops corresponds to the ShiftRows function. The correlation peak can be clearly seen to be occurring over this part of the waveform. Figure 10 shows a similar plot for bit-sliced AES. Here too, the correlation peak occurs around the ShiftRows function. The sensitivity of ShiftRows is expected, as it operates directly on the sensitive data. The T-box implementation shows sensitivity around the first round, as shown in Figure 11. It is difficult to localize the correlation peak from the T-box implementation any further, as each round is compressed into a memory lookup and a set of XOR operations.

The plots presented in Figures 9, 10 and 11 are for attacks using small number of measurements (768 for bit-sliced AES and 5120 for S-box and T-box AES). Hence, the sensitive locations that we have pinpointed for the different implementations correspond to those parts of the algorithm that break the earliest under an attack. For an attack using a large number of measurements, correlation peaks will often be spread out over a large area of the power trace, and not

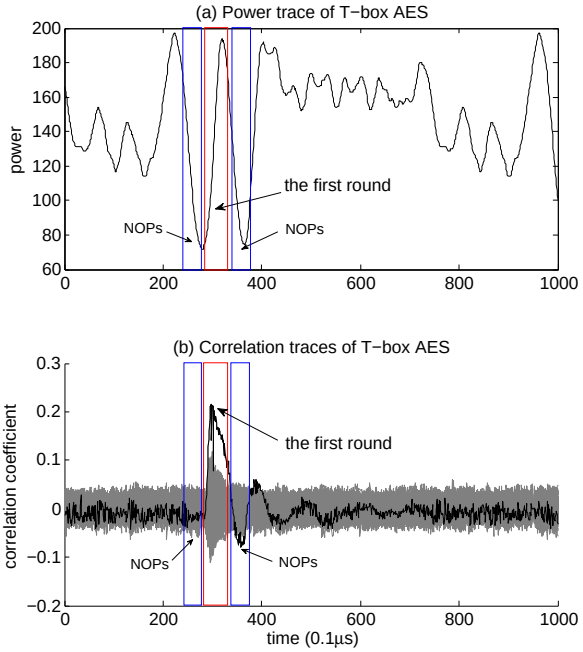

Figure 11: (a) Power trace of T-box AES on PPC platform; (b) Correlation coefficient plots for Side-channel Attack on T-box AES on PPC platform..

be localized to just one function or part of the algorithm.

6. CONCLUSIONS

In this paper, we compared three different implementations of AES from the viewpoint of performance and power-

based side-channel leakage. We concluded that large variations can be found in terms of performance as well as in terms of security. We have also shown that there are many factors that affect performance and side-channel resistance. The contribution of our work is in putting these implementations side-by-side, and demonstrating a practical and comprehensive evaluation. There are many extensions possible on this work. For example, one could consider the impact of side-channel resistant design techniques (such as masking) on performance and side-channel resistance. Or, one can consider additional attacks, such as those based on timing or on faults.

7. ACKNOWLEDGEMENTS

This project was funded in part by NSF Award no 0644070.

8. REFERENCES

[1] Side-channel Attack Standard Evaluation Board. `http://www.rcis.aist.go.jp/special/SASEBO/SASEBO-G-en.html`.

[2] D. J. Bernstein and P. Schwabe. New AES Software Speed Records. In *INDOCRYPT '08: Proceedings of the 9th International Conference on Cryptology in India*, pages 322–336, Berlin, Heidelberg, 2008. Springer-Verlag.

[3] E. Biham. A Fast New DES Implementation in Software. In *FSE '97: Proceedings of the 4th International Workshop on Fast Software Encryption*, pages 260–272, London, UK, 1997. Springer-Verlag.

[4] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. *CHES 2004*, LNCS 3156:page 16–29, 2004.

[5] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[6] Digilent Inc. Spartan 3E-1600 Development Board. `http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,793&Prod=S3E1600`.

[7] A. Gaisler. The LEON3 Multiprocessing CPU Core. Product Sheet, 2010. `http://www.gaisler.com/doc/leon3_product_sheet.pdf`.

[8] Intel Inc. Intel 64 and IA-32 Architectures Optimization Reference Manual. 2007. `http://www.intel.com/design/processor/manuals/248966.pdf`.

[9] E. Käsper and P. Schwabe. Faster and Timing-Attack Resistant AES-GCM. In *CHES*, pages 1–17, 2009.

[10] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[11] R. Könighofer. A Fast and Cache-Timing Resistant Implementation of the AES. In *CT-RSA'08: Proceedings of the 2008 The Cryptopgraphers' Track at the RSA conference on Topics in cryptology*, pages 187–202, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[13] M. Matsui. How Far Can We Go on the x64 Processors? In *FSE*, pages 341–358, 2006.

[14] M. Matsui and J. Nakajima. On the Power of Bitslice Implementation on Intel Core2 Processor. In *CHES '07: Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, pages 121–134, Berlin, Heidelberg, 2007. Springer-Verlag.

[15] D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. In *CT-RSA*, pages 1–20, 2006.

[16] C. Rebeiro, D. Selvakumar, and A. Devi. Bitslice Implementation of AES. *CANS*, LNCS 4301:pages 203–212, 2006.

[17] Xilinx. Virtex-II Pro Platform FPGAs: Module 2: Functional Description. Data Sheet, 2010. `http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf`.