Towards a Software Approach to Mitigate Correlation Power Analysis

Ibraheem Frieslaar[1,2] and Barry Irwin[2]
[1]*Modelling and Digital Science, Council for Scientific and Industrial Research, Pretoria, South Africa*
[2]*Department of Computer Science, Rhodes University, Grahamstown, South Africa*

Keywords: Software Countermeasure, AES, CPA, Threads, Task Scheduler, Resistance.

Abstract: In this research we present a novel implementation for a software countermeasure to mitigate Correlation Power Analysis (CPA). This countermeasure combines pseudo controlled-random dummy code and a task scheduler using multi threads to form dynamic power traces which obscures the occurrence of critical operations of the AES-128 algorithm. This work investigates the use of a task scheduler to generate noise at specific areas in the AES-128 algorithm to mitigate the CPA attack. The dynamic power traces have shown to be an effective contermeasure, as it obscures the CPA into predicting the incorrect secret key. Furthermore, the countermeasure is tested on an ATmega and an ATxmega microcontroller. The basic side channel analysis attack resistance has been increased and in both scenarios the proposed countermeasure has reduced the correlation accuracy and forced the CPA to predict the incorect key. The correlation accuracy decreased from 97.6% to 53.6% on the ATmega microntroller, and decreased from 82% to 51.4% on the ATxmega microcontroller.

1 INTRODUCTION

Since the introduction of side channel analysis (SCA) (Kocher et al., 1999), SCA has grown to be a powerful technique used to attack various embedded systems that employ cryptographic algorithms for security such as the AES algorithm. These cryptographic algorithms assist in protecting vital information of an individual or a system. The AES algorithm has become the main standard for encrypting data, therefore the research community have focused much of their attention on attacking the AES implementations on embedded devices (Brier et al., 2004; Schramm et al., 2004; Kocher et al., 2011; O'Flynn and Chen, 2015).

Kocher *et al.* have shown that there is a correlation between the power usage of a cryptographic system and its secret key-dependent intermediate variables. SCA uses this information to retrieve the secret key that was used to encrypt the data. SCA consists of three main types of attacks, simple power analysis (SPA), differential power analysis (DPA), and correlation power analysis (CPA). To mitigate these SCA attacks, various software countermeasures have been introduced such as boolean and arithmetic masking (Blömer et al., 2004; Oswald and Schramm, 2005); hiding (Tillich et al., 2007); random precharging (Hoogvorst et al., 2011) and shuffling (Veyrat-Charvillon et al., 2012).

This work aims to combine the knowledge of the hiding technique within the time frequency domain and combines it with mullti threads and a task scheduler to generate dynamic power traces on each execution of the AES-128 algorithm to create a software solution for a hardware problem. The best possible location to insert the pseudo controlled-random dummy code into the algorithm and the amount of threads that are required to be inserted to generate a noise pattern that hides information and forms a mask for the power traces are investigated.

1.1 Our Contribution

This research presents a novel implementation of a software countermeasure to mitigate the CPA attack on microcontrollers. The implementation consists of combining pseudo controlled-random dummy code that hides the occurrence of critical operations of the algorithm. Additionally, time dilation is used in combination with the task scheduler and multi threads to generate dynamic power traces. To the best of our knowledge we have not come across a software countermeasure that makes use of a threaded approach. The use of a task scheduler to generate noise at specific areas in the AES-128 algorithm to mitigate the CPA attack is investigated. The dynamic power traces has shown to be an effective measure, as it obscures the CPA into predicting the incorrect secret key. The countermeasure is tested on an ATmega

Frieslaar, I. and Irwin, B.
Towards a Software Approach to Mitigate Correlation Power Analysis.
DOI: 10.5220/0005955604030410
In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 4: SECRYPT*, pages 403-410
ISBN: 978-989-758-196-0

and an ATxmega microcontroller, in both scenarios the countermeasure has reduced the correlation accuracy significantly and has prevented the correct secret key from being predicted on both microcontrollers. Furthermore, the countermeasure have increased the attack resistance from $p = \frac{1}{(T+1)\cdot 16}$ to $p = \frac{1}{(T+1)^N \cdot 17}$ with a minimal execution overhead.

1.2 Organization

The remainder of this paper is organized as follows: Sections 2 discusses the research carried out in the field of software countermeasures against SCA; Section 3 elaborates on the equipment and techniques used to capture and analyze the data; Section 4 details the proposed countermeasure the system implements; followed by Section 5 and Section 6 which presents on the results and analysis of the results; and finally, the paper is concluded in Section 7.

2 SOFTWARE COUNTERMEASURES

This section discusses the most prominent existing techniques used as a software countermeasures against SCA. These techniques are random precharging, masking, hiding and shuffling. Random precharging in a software environment requires the datapath to be filled with random operand instructions before and after an important value is executed (Tillich and Großschädl, 2007).

The masking countermeasure generates a mask to camouflage the intermediate values of the algorithm with a random value not known to the attacker. Therefore the intermediate values will become independent from the power consumption of the device.

Hiding can possibly occur in two domains. These are the amplitude and time domains. Hiding information in the amplitude domain attempts to diminish the power consumption of various executions of the algorithm to reduce the overall power consumption whereas, the time domain requires the use of randomization of a specific execution of the algorithm to occur at different positions in time for every execution of the algorithm.

It is naturally easier to implement the hiding technique in the time domain as a software countermeasure. This type of hiding reduces the correlation between the power usage and its secret key-dependent intermediate variables. To achieve the randomization that leads to the specific execution of the algorithm occurring at various time locations, two methods can be used. These two methods are inserting dummy operations and shuffling the operations of algorithm. It is important that the dummy code not be recognized from the normal code, as the attacker could ignore the dummy code.

A good SCA resistance is achieved by using both the shuffling and insertion of dummy operations in the AES algorithm. (Tillich et al., 2007). The most effective location in the AES algorithm for this approach is at the S-box round of encryption. The AES round consists of 16 S-boxes (Daemen and Rijmen, 2002) thus the probability to locate a specific value at a specific point in time is $p = \frac{1}{16}$. Dummy operations (T) can be inserted at the S-box round, this leads to T operations added to 16 true states. On each execution of the algorithm the randomizer would integrate the true states with the dummy operations randomly. Therefore, the probability for finding a specific value at a specific point in time becomes.

$$p = \frac{1}{(T+1)\cdot 16} \qquad (1)$$

The protection against SCA attacks is determined by the value of p. Therefore, the greater the probability the more resistant the algorithm becomes against SCA attacks (Tillich et al., 2007).

3 EXPERIMENT SETUP

This section is separated into three subsections. Subsection 3.1 will discuss the equipment that was used to carry out a CPA attack, Subsection 3.2 will elaborate on the CPA attack algorithm, followed by Subsection 3.3 which discusses the experiments that were carried out.

3.1 Equipment

This subsection describes the equipment and settings used to carry out a successful CPA attack on the microcontrollers. In order to capture the data, we make use of a complete platform called the ChipWhisperer kit (O'Flynn and Chen, 2014). This kit serves as a platform for SCA attacks as it consists of the main elements required for an attack: a target device, measuring equipment, capturing software, and attack software. The hardware consists of a field-programmable gate array (FPGA). The ZTEX FPGA Module uses a Spartan 6 LX25 FPGA (ZTEX, 2016).

The hardware allows the attacker to use synchronous sampling. This grants the ability to synchronize the sample clock with the device clock.

It is demonstrated that sampling at 96 MS/s synchronously achieves the same results as sampling at 2 GS/s asynchronously (O'Flynn and Chen, 2012). Once the system clock and the device clock is synchronized it is possible to multiple the digital signal.

The attack is carried out by using the provided board known as the multi-target board. The multi-target board is connected to the FPGA module. The multi-target board consists of various areas that can target different devices such as an AVR microprocessor or a smartcard. For this research the AVR section of the multi-target board was used as the target area.

In this research the countermeasure has been tested on two different types of microcontrollers (MCU). The two MCUs that were used are the ATmega328p and the ATxmega128D4 MCUs. Additionally, two versions of the ChipWhisperer were used as well. The ChipWhisperer Capture Rev2 was used to capture data for the ATmega328p MCU and the ChipWhisperer Lite was used to capture data for ATxmega128D4 MCU. Both versions of the ChipWhisperer uses the same FPGA.

These MCUs are part of the 8-bit microprocessors from Atmel. They offers on-chip Flash, SRAM and internal EEPROM memories. These types of MCU have been used in various scientific research and in industrial applications (Kunikowski et al., 2015). Therefore, this research makes use of these two MCUs.

Both ChipWhisperers were configured to use the same variables when an attack was carried out. These variables are as follows: the gain was set to a low setting and the signal amplitude was increased by 34.5039 decibel (dB); the trigger was set to use the rising edge, thus the power traces would be captured at a rising edge logic level; and finally, the base clock frequency of 7.375 MHz was multiplied by 4 to give a frequency of 29.5 MHz. These set variables assisted in increasing the power output and remained constant for all experiments.

3.2 Attack Procedure

This subsection explains the type of SCA attack that will be used in this research. In this research the CPA attack would be used (Brier et al., 2004). It is known that the CPA out performs the standard DPA attack. The CPA is much faster in processing data since it only requires a few power traces whereas the DPA is slower and needs thousands of power traces. Furthermore, CPA is more accurate at predicting the correct subkeys as it looks at the correlation between all the key guesses (Brier et al., 2004).

Firstly, the mathematical approach of the CPA would be explained followed by an implementation of using it to retrieve the secret keys from the AES algorithm.

It is assumed that the attacker has captured a power trace $t_{d,j}$ with $d = 1,2,3\ldots D$ being the total amount of traces, and $t = 1,2,3\ldots T$ is the time index per power trace. Therefore, the attack system takes D measurements with each measurement being T points long. If the attacker has the knowledge of the exact point the encryption process occurs, then only a single point would be measured such that $T = 1$ for each d trace. The plaintext that corresponds to the power trace is also known by the attacker and is defined as P_d.

It is assumed that the microcontroller's power consumption is dependent on the intermediate value's hamming weight. $h_{d,i} = l((P_d,i))$ is defined, where a given intermediate value's leakage model is $l(x)$, and $w(p,i)$ produces and intermediate value based on the given input plaintext with a guess number $i = 1,2,3\ldots I$.

In terms of AES, the intermediate value would be selected as follows: each byte of the plaintext is XOR'd with each subkey byte of the secret key. Therefore we have:

$$\begin{aligned} l(x) &= HammingWeight(x) \\ w(p,i) &= p \oplus i \end{aligned} \tag{2}$$

This implies that a single byte of the plaintext p is being attacked at a time. Therefore the AES key is being attacked a single byte at a time. It is known that AES-128 algorithm consists of 16 subkeys (Daemen and Rijmen, 2002) thus, there is only 16×2^8 possibilities, instead of 2^{128} possibilities.

The next procedure is to establish a linear relationship between the captured power traces $t_{d,j}$ and the predicted power consumption model $l(x)$ using the correlation coefficient. The aim is to ensure that there is a non-linear relationship between $w(p,i)$ and either p or i. For this case it is possible to attack the the AES-128 algorithm at the point of the non-linear substitution boxes (S-Boxes). The final step is to calculate the correlation coefficient for each of the possible subkey values I over all traces D for each points of j, the equation is as follows:

$$r_{i,j} = \frac{\sum_{d=1}^{D}[(h_{d,i} - \bar{h_i})(t_{d,j} - \bar{t_j})]}{\sqrt{\sum_{d=1}^{D}(h_{d,i} - \bar{h_i})^2 \sum_{d=1}^{D}(t_{d,j} - \bar{t_j})^2}} \tag{3}$$

Where h is the hypothetical values produced by the power consumption model.

Based on the mathematical explanation of the CPA in order to obtain the secret key of the AES-128 algorithm, four steps needs to be accomplished. These four steps are as follows:

1. Capturing the power trace, along with the input text when the encryption process is executed.
2. Implement a power leakage model, where the known input text is used with a guess of the key byte.
3. Implement the correlation equation that loops through all the captured power traces.
4. Create a ranking procedure that determines the most likely key based on the correlation equation.

Figure 1 illustrates a snippet of the AES algorithm at the first round of the algorithm.

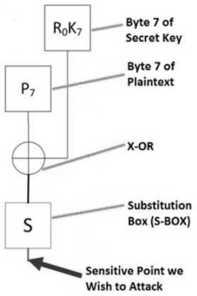

Figure 1: A snippet of the AES algorithm with an arrow poitning at the point of attack.

It is depicted that each input byte of text is XOR'd with the secret key which passes through an S-Box (a lookup table). The output of the S-Box is the target area where the attack will take place, this can be seen by the red arrow in the figure.

After the data is acquired, the next step is to perform a guess and create a power leakage model. A single subkey is attacked at a time, thus the attack system loops through a single subkey and guesses every possibility for that subkey. The guessed values range from 0 – 255. Upon acquiring the guess, the intermediate value corresponding to the guess needs to be calculated. Therefore, a single byte of input and a single byte of the guessed key, is used to return the output of the S-Box. For each guess, the number is converted to binary and the total number of 1's of that binary output is accumulated to determine the *weight*, known as the hamming weight power model (Mestiri et al., 2013).

The correlation is calculated by substituting all the acquired data into Equation 3. It is possible to obtain a negative correlation. However, only the value is required, thus the absolute value of the correlation is used. Furthermore, only the maximum correlation across all points in the trace are stored. In order to achieve a ranking system the maximum correlation is used to find which hypothetical key caused that maximum correlation, this intern is the subkey of the secret key.

3.3 Experiments

The experimental setup can be broken down into two phases the capture data and analyze data phases as seen in Figure 2.

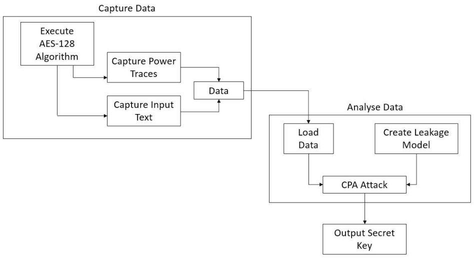

Figure 2: The flow diagram of the experimental setup.

In order to capture the data, the Atmel ATmega328p and ATxmega128D4 MCUs was programmed to execute the AES-128 cryptographic algorithm. The ATmega328p was placed into the multi-target board, the target board was connected to the FPGA which was connected to a computer. Python is used to send commands from the computer to the FPGA, issuing the commands to MCU to execute the AES-128 algorithm. While the algorithm is being executed the power traces along with its corresponding input text are captured. The ATxmega128D4 is an embedded MCU and it is embedded onto a pcb, known as the device under test as seen Figure 3. The device under test is connected to the FPGA of the ChipWhisperer Lite and the same procedure to capture data from the ATmega328p is used.

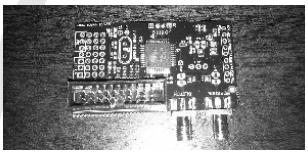

Figure 3: The device under test with the embedded MCU of the Atmel ATxmega128D4.

Once the data has been acquired it is sent to a Python program where it determines the secret key. This program consists of the leakage model and CPA attack discussed in the previous subsection to guess the correct secret key.

The first experiment is to create a baseline for future experiments. The baseline is determined by implementing the AES-128 on both MCUs with no countermeasures in place and using the CPA technique to retrieve the secret key. While the MCUs are executing the AES-128 algorithm, the power traces and its corresponding input text would be collected. The data collection will be repeated 50 times, on each occasion the same secret key and different random input text would be used. Therefore, each set would consist of 50 power traces with the same secret key and different input text. Upon acquiring the data, the CPA attack would be performed on these power traces.

In order to determine the best location to insert the noise threads two pre-experiments were carried. Experiment A inserted the noise threads at the odd subkeys and Experiment B inserted the noise threads at all the subkeys.

The second and third experiments would consist of implementing the countermeasure on the ATmega328p and ATxmega128D4, respectively. The same procedure would be used as discussed previously. Additionally, each experiment consists of an additional 25 test cases. These 25 test cases used different secret keys. The secret keys were generated by RANDOM.ORG which uses atmospheric noise as a True Random Number Generator (TRNG) to generate their data (RANDOM.ORG, 2016).

The robustness and scalability of the countermeasure will be tested in experiment four and five, where the number of trace would be increased from 50 to 200. On each increment of 50 power traces, the data would be used as input for the attack procedure for both MCUs. To test the added overhead introduced by the countermeasure in experiment four and five, the time it takes to execute one run of the algorithm will be recorded for all test scenarios.

4 PROPOSED COUNTERMEASURE

This section discusses the proposed countermeasure that the system will implement. The countermeasure aims to improve on the basic hiding and shuffling techniques and introduces a new approach of using a multi threads and a task scheduler.

The countermeasure makes use of the library implemented by Dean Ferreyra, known as the AVR Threads Library.(Ferreyra, 2008) The AVR Threads Library provides basic preemptive multitasking/multi-threading to the Atmel AVR family of microcontrollers. The library was implemented in assembly and in C language. A simple round-robin style task switcher is implemented. Originally, the library was designed for the ATmega128 and AT90S8515 MCUs, the library has been modified and recompiled to work with ATmega328p and ATxmega128D4 MCUs.

The basic idea is to have the AES-128 algorithm executing on one thread, while multiple other threads would be executing at the same time. These multiple threads would consist of dummy code, and will be known as noise threads. These noise threads would be executed at the S-boxes. Furthermore, on each execution of the code, the noise threads would vary.

A random number (x) would be generated between 0 – 15 this would server as the amount of threads to use. The next step in the procedure is to use that number and generate noise threads at random subkeys $y[x]$. To calculate at which location of the subkey the noise would appear each $y[\,]$ value was randomized between 0 – 15, a check is performed to make sure that none of the $y[\,]$ values are the same. Thus, the noise would be placed at different locations of the subkeys and generate dynamic power traces. Therefore, on each execution of the algorithm the power traces would differ. This approach gives us a resistance of:

$$p = \frac{1}{(T+1)\cdot 16} \tag{4}$$

Where T is the number of threads. The resistance to SCA attacks is the same to that of Equation 1. The resistance is further increased by having different types of noise threads. Each noise thread can execute a different set of dummy code. These dummy sets of code are various mathematical instructions. These instructions were placed inside the *printf* method. This generates a more robust power signal.

Below is a code snippet that illustrates at line 7 – 9, the *Noise* method making a call to the arithmetic method which is located at line 1 – 5. In order to encapsulate the *Noise* method into the thread framework it is called at line 9.

```
Begin
1    int ArthmPlus(int A, int B){
2      int x;
3      x = A+B;
4      return x;
5    }
6    void Noise(void){
7       printf("%d\n",ArthmPlus(10,20));
8    }
9   avr_thread_start(&fn_context,Noise,
fn_stack, sizeof(fn_stack));
End.
```

The resistance is increased by having each noise thread perform a different instruction. The type of noise generated by the noise thread is randomly selected. Therefore the new resistance becomes:

$$p = \frac{1}{(T+1)^N \cdot 16} \tag{5}$$

Where N is the number of different types of noise that are produced. It is noted that at the execution location of a subkey, there can be more than one noise thread. In this research only one noise thread would be generated at a subkey, since the ChipWhisperer has a hardware limitation of only be able to capture 24000 points. Therefore, this technique already shows it has the capabilities of deterring attackers that use low cost equipment to attack microcontrollers.

On the initial startup a value was stored into the EEPROM of the MCUs. This value is used to ensure that the algorithm is generating a different sequence of random values on each execution. Before the S-box procedure is called, the value is pulled from the EEPROM and used by the *srand()* method to change the random sequence. After the S-box procedure has been completed this value is incremented and stored back into the EEPROM to be used for the next program execution. Furthermore, to create more confusion and increase the resistance, this value is XOR'D and gets passed through the same S-box the AES algorithm uses. By inserting an additional XOR into the algorithm, locating a specific value at a specific point in time is no longer $\frac{1}{16}$ but now $\frac{1}{17}$. Therefore, the resistance becomes:

$$p = \frac{1}{(T+1)^N \cdot 17} \tag{6}$$

5 RESULTS

This section examines the results of the experiments explained in Subsection 3.3. The first step is to analyze the findings from experiment one. Table 1 shows the results for the CPA attack against both microcontrollers, while no countermeasures are in place. Furthermore, the table indicates the results for experiments A and B. These results will serve as the baseline that the other experiments are compared to. The table illustrates the correlation accuracy that the attack predicts for each subkey of the AES-128 algorithm, followed by a total average accuracy for all the subkeys.

Referring to Table 1, it is indicated that using 50 traces as input, CPA achieves an average accuracy of 97.66% while the ATmega328p MCU is under test and an average accuracy of 82% as the ATxmega128D4 is under test. In both experiments, the secret key is successfully retrieved. Furthermore, the table illustrates the results of experiments A and B. A slight reduction in accuracy is observed in comparison to the base results. However, the CPA attack still predicts the correct secret key. This is due to the fact that the CPA can handle power traces with the same type of noise even at different locations.

Tables 2 consists of five columns, with the first column referring to the microcontroller under test. The second column denotes whether the secret is recoverable. The third column indicates the number of recoverable subkeys over all cases, followed by its predication rate for recovering a subkey, in column four. In order to calculate the correct subkey prediction rate, the number of correct subkeys is divided by the total number of subkeys and multiplied by 100. Finally, column five indicates the average correlation accuracy. As explained in Subsection 3.2, the correlation accuracy is determined by the ranking system. The ranking system list all possible subkeys with their corresponding correlation values. The correlation values of the real subkeys are summed up and averaged, to give the average correlation accuracy. Furthermore, Table 3, second column depicts the number of traces used as input, followed by, if the secret key is recoverable, number of subkeys recoverable, and the correlation accuracy columns.

The next discussion will elaborate on the findings of experiment two and three, as the software countermeasure is on both microcontrollers. It is observed in Table 2, the CPA predicts the secret keys incorrectly and thus the countermeasure is working. It has only predicted two subkeys out of 400 subkeys correctly. The correct subkey prediction rate on the ATmega328p MCU is 0.5%. Therefore, 0.5% of the time CPA predicts a correct subkey. Furthermore, the average correlation accuracy is only 53.6%. Therefore the base correlation accuracy of 97.66% using the ATmega328 MCU has seen a reduction of 43.96% in correlation accuracy. The results further indicate that the countermeasure prevents the secret keys from being correctly predicted on the ATxmega128D4. The CPA attack correlation drops from 82% to an average of 51.4% in correlation accuracy. Furthermore, only 1 subkey out of 400 subkeys is predicted correctly. The system has a correct subkey prediction rate of 0.25% on the ATxmega128D4 MCU.

The next evaluation of the system is testing the scalability of the countermeasure as explained earlier in Subsection 3.3.

Table 3 illustrates the results for experiments four and five. Observing the table, the first noticeable finding is that the CPA attack predicts the incorrect secret keys on all four occasions. As the number of sample traces increase the correlation accuracy decreases. By adding an additional 150 traces, the correlation accuracy decreases to 26.8% from 53.6% as the ATmega328p is in use. A similar result is obtained as the ATxmega128D4 is in use. The correlation accuracy decreases from 51.4% to 26.3%.

The final experiment is to determine, the extent of extra overhead the countermeasure has on the algorithm. Table 4, depicts the average time it took to execute one instance of the AES-128 algorithm on the ATmega328p and ATxmega128D4 MCUs with and

Table 1: The results for the CPA attack against both microcontrollers, when no countermeasure is in place. All subkey values are in percentage(%)

	Subkey																
MCU	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
ATmega328p	98,0	98,8	98,7	98,1	96,8	98,3	94,4	98,6	95,9	98,5	98,5	98,8	97,5	98,2	97,2	96,2	97,6
ATxmega128D4	71,7	86,4	88,7	85	84,7	84,8	75,8	91,0	82,7	71,3	80,2	85,3	75,4	79,4	81,8	87,1	82,0
Experiment A	91,5	88,4	95,2	90,2	93,0	88,0	91,7	82,0	92,7	74,4	91,4	95,7	95,2	89,0	93,8	90,2	90,1
Experiment B	90,4	98,6	97,5	92,9	91,4	98,4	93,3	97,8	91,5	97,7	89,9	81,4	92,6	98,0	91,0	92,0	93,4

Table 2: Result of the CPA attack using various trace samples on the ATmega328p.

MCU	Secret Key Predicted	Num. of SubKeys Predicted	SubKey Prediction Rate (%)	Correlation Accuracy (%)
ATmega328p	No	2	0.5	53.6
ATxmega128D4	No	1	0.25	51.4

Table 3: Result of the CPA attack using various trace samples on the ATmega328p.

MCU	Traces	Secret Key Predicted	Num. of SubKeys Predicted	Correlation Accuracy(%)
ATmega328p	100	No	0	38.9
ATmega328p	200	No	0	27.7
ATxmega128D4	100	No	0	37.4
ATxmega128D4	200	No	0	26.3

Table 4: Time taken to execute the code with and without the countermeasure

MCU	Time (s)
ATmega328p	0.102
ATmega328p + Countermeasure	0.170
ATxmega128D4	0.115
ATxmega128D4 + Countermeasure	0.198

without the countermeasure.

It is observed that the difference in time on the ATmega328p MCU with and without the countermeasure is only 0,068 milliseconds and the difference on the ATxmega128D4 MCU is 0,084 milliseconds. Thus, by adding the countermeasure code with the existing AES-128 algorithm the execution time difference has a minimal increase on both the ATmega328p and ATxmega128D4 MCUs.

6 ANALYSIS

As we know the CPA attack only require less than a 100 power traces to carry out a successful attack. This is evident in experiment one and two where 50 power traces were needed to predict the correct secret key on two different microcontrollers. This is due to the strength of the CPA attack. Before the proposed countermeasure was implemented it was seen from the results in experiment A and B that inserting the same type of noise at every subkey or at every odd subkey in the substitution round, the CPA attack still predicted the correct secret key. This is due to the fact that the power traces had little change in each execution of the algorithm and each subkey was attacked.

Once the proposed countermeasure was in place, the CPA attack started loosing correlation accuracy tremendously. The power leakage model of the CPA relies on the power traces to have small changes when each subkey is attacked and since the proposed countermeasure produces dynamic power traces where the power trace is different on each occasion, causing confusion to the CPA attack model. Therefore, having more permutations of the dynamic power trace would lead to the CPA producing an even lower accuracy. This is further evident by the fact that when more power traces were added to the input, the correlation accuracy decreased and the attack predicted the incorrect secret keys.

These dynamic power traces are very effective against attackers that make use of low cost equipment, such as the ChipWhisperer, since it is possible to program the algorithm to use more noise threads at each subkey location. This will force the CHipWhisperer to only capture half or less of the actual runtime of the encryption process. This prevents the attackers from gaining useful data to carry out an attack. Furthermore, the proposed countermeasure has a minimal execution overhead. This leads to a slowdown in performance. However, the tradeoff is a more secured implementation.

7 CONCLUSION

In this work we present a novel software countermeasure to mitigate CPA. The implementation consists of combining pseudo controlled-random dummy code to hide the occurrence of critical operations of the AES-128 algorithm. Additionally, time dilation is used in combination with multi threads and a task scheduler to generate dynamic power traces. We have investigated the use of a task scheduler to generate noise at specific areas in the AES-128 algorithm to mitigate the CPA attack. The dynamic power traces have shown to be an effective countermeasure, as it obscures the CPA into predicting the incorrect secret key. Furthermore, the countermeasure was shown to work on an ATmega and an ATxmega microcontroller.

In both scenarios the countermeasure reduced the correlation accuracy significantly and prevented the correct secret key from being predicted. The research has also displayed that the extra overhead introduced by the countermeasure is minimal in execution time and that the basic SCA resistance has increased when using this countermeasure. Therefore, this research has introduced a novel low overheard software solution that uses multi threads and a task scheduler for a hardware security problem.

8 FUTURE WORK

Although, the countermeasure has demonstrated that it is able to mitigate the CPA attack, the issue of performing instructions sequentially still remains on these microcontrollers. It is intended to improve on this work by implementing the countermeasure on an embedded device that supports true multi threading functionality where it would be able to execute the noise and the AES threads in parallel. Additionally, it is aimed to create a system where the algorithm learns to manipulate the data such that it never produces the same power trace twice.

ACKNOWLEDGEMENTS

The authors would like to thank the department of Modelling and Digital Science at CSIR for providing funding and support.

REFERENCES

Blömer, J., Guajardo, J., and Krummel, V. (2004). Provably secure masking of AES. In *Selected Areas in Cryptography*, pages 69–83. Springer.

Brier, E., Clavier, C., and Olivier, F. (2004). Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 16–29. Springer.

Daemen, J. and Rijmen, V. (2002). The design of rijndael: AES. *The Advanced Encryption Standard*.

Ferreyra, D. (2008). AVR development. http://www.bourbonstreetsoftware.com/AVRDevelopment.html.

Hoogvorst, P., Duc, G., and Danger, J.-L. (2011). Software implementation of dual-rail representation. *COSADE, February*, pages 24–25.

Kocher, P., Jaffe, J., and Jun, B. (1999). Differential power analysis. In *Advances in CryptologyCRYPTO99*, pages 388–397. Springer.

Kocher, P., Jaffe, J., Jun, B., and Rohatgi, P. (2011). Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27.

Kunikowski, W., Czerwiński, E., Olejnik, P., and Awrejcewicz, J. (2015). An overview of ATmega AVR microcontrollers used in scientific research and industrial applications. *Pomiary, Automatyka, Robotyka*, 19.

Mestiri, H., Benhadjyoussef, N., Machhout, M., and Tourki, R. (2013). A comparative study of power consumption models for CPA attack. *International Journal of Computer Network and Information Security*, 5(3):25.

O'Flynn, C. and Chen, Z. (2012). A case study of side-channel analysis using decoupling capacitor power measurement with the OpenADC. In *Foundations and Practice of Security*, pages 341–356. Springer.

O'Flynn, C. and Chen, Z. D. (2014). Chipwhisperer: An open-source platform for hardware embedded security research. In *Constructive Side-Channel Analysis and Secure Design*, pages 243–260. Springer.

O'Flynn, C. and Chen, Z. D. (2015). Side channel power analysis of an AES-256 bootloader. In *Electrical and Computer Engineering (CCECE), 2015 IEEE 28th Canadian Conference on*, pages 750–755. IEEE.

Oswald, E. and Schramm, K. (2005). An efficient masking scheme for AES software implementations. In *Information Security Applications*, pages 292–305. Springer.

RANDOM.ORG (2016). Introduction to randomness and random numbers. https://www.random.org/randomness/.

Schramm, K., Leander, G., Felke, P., and Paar, C. (2004). A collision-attack on AES. In *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 163–175. Springer.

Tillich, S. and Großschädl, J. (2007). *Power analysis resistant AES implementation with instruction set extensions*. Springer.

Tillich, S., Herbst, C., and Mangard, S. (2007). Protecting AES software implementations on 32-bit processors against power analysis. In *Applied Cryptography and Network Security*, pages 141–157. Springer.

Veyrat-Charvillon, N., Medwed, M., Kerckhof, S., and Standaert, F.-X. (2012). Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *Advances in Cryptology–ASIACRYPT 2012*, pages 740–757. Springer.

ZTEX (2016). Spartan 6 LX9 to LX25 FPGA board. http://www.ztex.de/usb-fpga-1/usb-fpga-1.11.e.html.