A Comparison of Single-Bit and Multi-Bit DPA for Attacking AES128 on an ATmega328P

Michael Yonli
Email: michiy@gmx.net

Abstract

Side channel attacks have demonstrated in the past that it is possible to break cryptographic algorithms by attacking the implementation rather than the algorithm. This paper compares an adaptation of Paul Kocher's Differential Power Analysis (DPA) for AES with a multi-bit variant by attacking an AES128 implementation for an ATmega328P microcontroller board. The results show that the use of multi-bit DPA can significantly reduce ghost peaks and allow for the recovery of a key with far fewer traces.

1 Background

1.1 Side Channel Attacks

While widely used algorithms have successfully been designed to be resistant against attacks, side channel attacks which target implementations have had a bigger impact on the security of cryptographic systems. Popular implementations of widely used algorithms have been shown to be vulnerable to such attacks. These side channel attacks utilise implementation specific characteristics in order to attack cryptographic systems [6]. Examples include Bernstein's timing attack [2] on the Advanced Encryption Standard (AES), in which he demonstrates how an attacker can recover the key of an algorithm within a few hours, and Kocher's Diffrential Power Analysis (DPA) [6], which demonstrates how power draw can be used to recover the key being used for the Data Encryption Standard (DES). What sets these attacks apart is that they are completely independent of the security of the attacked algorithm. An algorithm with no known attacks may still be vulnerable to side channel attacks. This complicates the design of secure systems, since the implementational aspects have to be considered. The algorithm itself may be vulnerable, the software system that was written to implement the algorithm may be vulnerable or the underlying hardware may be vulnerable.

1.2 Original DPA Attack

In his publication Kocher describes an attack which uses the correlation between the power dissipation of a processor and the data it is operating on to recover the cryptographic key of a smartcard running DES [6]. Fundamentally, his attack uses the correlation in order to create an oracle which allows the attacker to bruteforce each possible key block separately. Rather than guessing the whole key at once, the attacker can bruteforce the key chunk by chunk.

The first step of the attack is to recover a set of power traces [6]. Each trace represents the power dissipation of the system as it encrypts a piece of data with a fixed key [6]. Kocher attacks the last round of the encryption process and makes use of a selection function [6]: $D(C, b, K_s)$. C denotes the ciphertext (note that we need to capture the ciphertext along with the power traces) [6]. b selects the bit of the DES intermediate L at the beginning of the last round, which is the value of the function [6]. And K_s is a guess for the key block that enters the S-box corresponding to bit b of the intermediate [6]. The attack assumes that an incorrect guess of K_s results in D computing the correct value of bit b in half of all cases [6].

The attacker now captures m traces with k samples each [6]: $T_{1\ldots m}[1\ldots k]$ and computes the differential trace $\Delta_D[1\ldots k]$ [6].

$$\Delta_D[j] = \frac{\sum_{i=1}^{m} D(C_i, b, K_s) T_i[j]}{\sum_{i=1}^{m} D(C_i, b, K_s)} - \frac{\sum_{i=1}^{m} (1 - D(C_i, b, K_s)) T_i[j]}{\sum_{i=1}^{m} (1 - D(C_i, b, K_s))} \tag{1}$$

If the correct K_s is chosen, then the differential trace will correspond to the average power consumption of having that one bit set [6]. For an incorrect K_s, there is no correlation between the differential trace and the average power consumption of the chosen bit [6]. As D is going to compute the correct value in half of all cases, we are effectively looking at a function which randomly assigns traces into two sets, for an incorrect K_s. Thus:

$$\lim_{m\to\infty} \Delta_D[j] \approx 0 \tag{2}$$

for an incorrect K_s [6]. We can calculate $\Delta_D[j]$ for every possible K_s, in the case of DES that would be 64 possibilities [11]. The differential trace with the biggest peak will then correspond to the correct key block.

1.3 Problems with DPA

One of the issues which was hinted at by Kocher and also encountered by various other researchers are so called ghost peaks [6][8]. Differential traces for incorrect key guesses are not always close to 0 [6], they may display peaks and these peaks can be even bigger than the ones for the correct key guess [8]. Ghost peaks are partially caused by a bias of the S-boxes [3].

Some researches attempted to leverage these ghost peaks by predicting where they would occur for different roundkeys and then comparing them to the measured peaks [8].

Another possibility of approaching this issue is to make use of what is called multi-bit DPA [7].

2 Multi-Bit DPA

As the name suggests, multi-bit DPA uses multiple bits rather than a single bit to calculate the differential trace. But first, we have to adjust Kocher's DPA from DES to AES. Instead of attacking the intermediate L, we will be attacking a bit of the S-box input. The last round of AES consists of the *SubBytes*, *ShiftRows* and *AddRoundKey* operations in that order [4]. We simply guess the key byte and invert *SubBytes*; *ShiftRows* can be ignored since the position of the targeted byte in the state does not matter. We then use the value of the S-box input bit b to partition the traces into two sets. What makes this version distinct from ordinary DPA is that we perform this partition for every possible b. This reduces the impact of S-box biases and other unwanted correlations.

The new formula is as follows:

$$\Delta_D[j] = \frac{\sum_{b=1}^{8}(\sum_{i=1}^{m} D(C_i, b, K_s)T_i[j])}{\sum_{b=1}^{8}(\sum_{i=1}^{m} D(C_i, b, K_s))} - \frac{\sum_{b=1}^{8}(\sum_{i=1}^{m}(1 - D(C_i, b, K_s))T_i[j])}{\sum_{b=1}^{8}(\sum_{i=1}^{m}(1 - D(C_i, b, K_s)))} \tag{3}$$

Note that the formula assumes that we are dealing with an 8-bit S-box.

3 Measurement Setup

3.1 Device under Attack

In order to minimise the time period that needed to be captured per trace, a highly optimised assembly implementation of AES-128 was chosen [9]. The attack would also work with a less efficient implementation, but the limited number of sample points that can be stored on the oscilloscope would require a lower sampling frequency. Additionally, less sample points per trace reduce both the file size of the traces and make the attack less time-consuming.

This implementation had to be modified since it violates the avr-gcc application binary interface (ABI), which makes an integration with high level programming languages difficult. The author did not follow the conventions regarding the use of registers and parameter passing. As a result, wrapper functions were added which store and restore certain registers and translate the passed parameters from the high level language ABI to the interface used by the AES implementation. In addition, some assembly instructions were added in order to toggle a pin for the duration of the encryption routine. This ensures that the oscilloscope can measure the process efficiently and simplifies synchronisation.

The platform to run the cryptographic algorithm was the Arduino Uno, a microcontroller board. The Arduino Uno uses an ATmega328P with a clock frequency of 16MHz to execute uploaded programs [1]. One of the main reasons

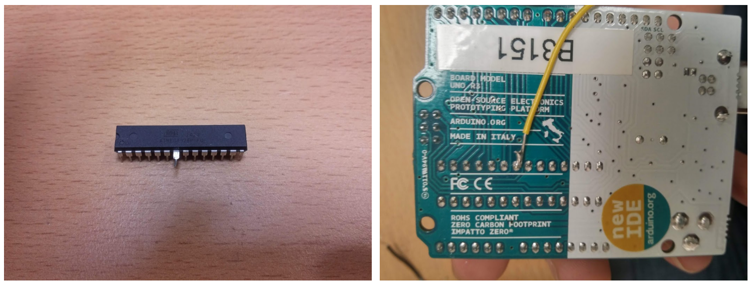

Figure 1: Left: ATmega328P microcontroller with its VCC pin bent in order to avoid making contact with the socket in the package. Right: Back of the Arduino board with a wire soldered to a contact corresponding to the VCC socket in the package.

why this platform was chosen is because of its low clock frequency. A higher clock frequency would have required us to capture signals that have a higher frequency, which in turn would have required more expensive equipment that we did not have access to. Another benefit is the microcontroller's deterministic behaviour. Table lookups are executed in constant time, since this simple architecture does not make use of data caches. This ensured that an encryption always took the same amount of time and made it easier to align traces. The only feature which added variation to the process were interrupts, these were simply disabled for the duration of the encryption.

The firmware that ran on the Arduino read one block (16 bytes) over a universal serial bus (USB) connection, encrypted it and sent the ciphertext back over the USB connection. The cipher mode being used was electronic code book (ECB).

Measuring the current draw of the processor requires the addition of a shunt resistor. This resistor was added between the VCC contact of the DIP-package and the VCC pin of the microcontroller by bending the microcontroller pin so that no contact with the socket is made on that pin and soldering the other part of the resistor to the package contact. The microcontroller is capable of running without any current coming through the VCC pin as it has a second pin, $AVCC$, for the analogue parts which can fully supply the microcontroller on its own. We verified this by completely disconnecting the VCC pin and running a program that would modify pins on the microcontroller. The change in state of the pins showed that the program still ran as expected. However, the voltage of the digital pins was reduced and programs which require the output of signals with higher voltages may run into issues.

Using USB to power the board resulted in strong fluctuations of the power supply which could have impacted the attack. Hence, we decided to power the board with a dedicated power supply, which resulted in a fairly stable supply voltage.

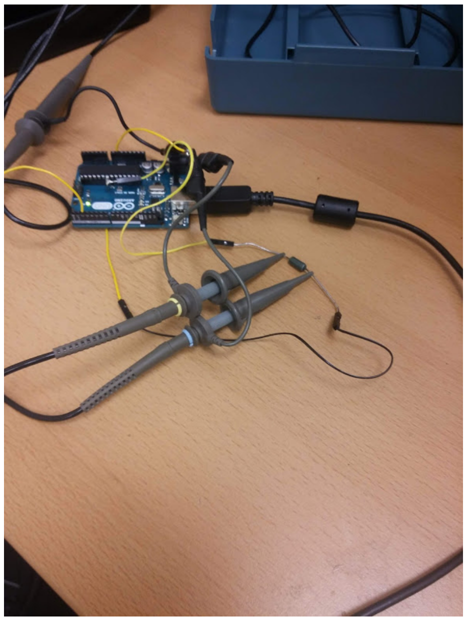

Figure 2: The DPA setup: two probes are connected to the shunt resistor and the third probe which is not fully in the picture is connected with the trigger pin.

3.2 Oscilloscope

A Tektronix TDS3034B oscilloscope was used to capture traces from the device under attack. Two probes were attached to the shunt resistor and one probe was attached to the trigger pin. This pin was manipulated in order to indicate the state of the Arduino. The ground clamps of all three probes were attached to the ground pin of the board. The roughly fixed value of the shunt resistor would allow us to calculate the current that is currently flowing through the processor. However, this calculation is essentially a multiplication by a constant factor and can therefore be neglected for our purposes. The attack may be carried out by using the voltage drop across the resistor rather than the actual current flowing through it. In order to recover the voltage difference across the two channels, an internal function of the oscilloscope which calculates the difference between two inputs was used. This allowed us to half the amount of data that needed to be transferred from the oscilloscope, when compared with sending both signals and calculating the difference on another platform. The fact that the microcontroller is capable of running without a voltage being applied to VCC means that we may use resistors with an arbitrarily high resistance for our shunt resistor. Empirically, we determined that a resistor with a higher resistance resulted in bigger voltage differentials. But this effect disappeared as we moved towards higher resistances, for example no clear difference between 1k

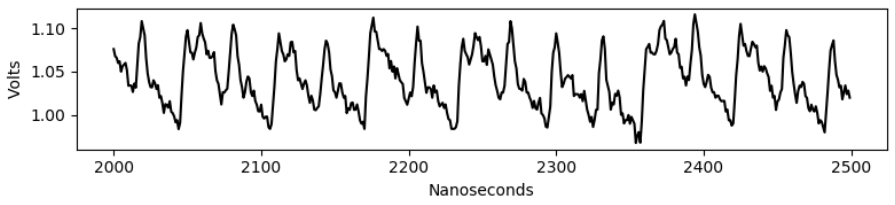

Figure 3: A small portion of a trace. A clock period corresponds to 62.5 nanoseconds.

Ohm and 1M Ohm were found. The final resistance of the resistor was chosen to be 10M Ohm.

The oscilloscope offers a bandwidth of 300 MHz which is sufficient, considering that our target uses a clock frequency of 16 MHz. The whole captured spectrum is affected by noise and limiting the bandwidth to the frequencies of the signal can result in a higher signal-to-noise ratio [5]. We experimented with halving the bandwidth in an attempt to remove noise and recover a higher quality signal, but the effects were marginal and the full bandwidth was used.

A sampling frequency of 1 GS/s was used for the experiment since this is the highest frequency supported when considering the storage constraints and the time needed for an encryption. The oscilloscope is capable of storing a total of 10 000 sample points, which turned out to be a limiting factor and made the experiment more difficult. According to the official documentation of the AES implementation, the encryption of a block of data takes 2474 clock cycles [9]. Hence, we would be left with about 4 samples per clock period if we were to capture the full encryption. This translates to a sampling frequency of roughly 64MS/s. However, we are using 1GS/s which allows us to capture about 160 clock cycles. In order for the attack to succeed, only a specific part of the whole encryption process is needed. Namely, the beginning of the last round during which the final S-box lookup happens. We were able to limit that space to these 160 cycles by adding assembly instructions to raise a pin during the parts of the encryption that needed to be captured and then measuring the time. These instructions were removed during the actual measurement, since there was some concern that they could influence the supply voltage. Driving or lowering a pin is an expensive action which leaves a clearly visible peak in the power trace. The results were verified by examining the source code and counting the clock cycles that the processor would take to execute the program.

3.3 Laptop

The whole experiment is orchestrated from a laptop which is controlling the oscilloscope via an Ethernet connection and the Arduino over an USB connection. The oscilloscope exposes a programming interface over a protocol called VXI-11. This interface allows us to download data and to configure the settings of the oscilloscope. A library was developed, with the help of the developer's handbook, in order to implement the commands that are essential for the experiment. These are mostly related to the downloading of data and managing the trace acquisitions.

As part of the initialisation, some parameters such as the format of the downloaded data and trigger settings are set. This is followed by the arming of the trigger, which enables the oscilloscope to capture data. We now proceed to send a block of random data to the Arduino which encrypts the data, toggles the trigger pin in the process and sends the ciphertext back over the USB connection. At this point the oscilloscope will have captured a trace of the encryption process and will disarm the trigger to avoid synchronisation issues. The program running on the laptop now verifies the encryption of the Arduino by using a different AES implementation and comparing the results. A deviation results in the abortion of the experiment and an error message. So far no deviations have occurred, which indicates that the AES implementation is correct and that no corruption occurs as part of the measurement process. The ciphertext is saved in a file and the trace is downloaded from the oscilloscope. We have experimented with using a relational database for the data, but code profiling revealed that the database increases the time taken to store each measurement by a significant amount while offering no clear benefits. This download of data turned out to be the bottleneck in the process of taking measurements and care was taken to optimise this. As part of the optimisation the internal oscilloscope data format is used for the data. This format needs the least amount of data for a trace and is also the fastest according to the documentation [10]. A custom parser for the internal format was written as part of the project and verified with a sine wave. This parser converts the traces from the internal format to a format known as comma separated value (CSV). Finally, the CSV data is stored in a flat file. This whole process takes about one second per trace with the bulk of the time being spent on waiting for the oscilloscope.

The next step is to process the captured data and perform the attack described in the next chapter. First we need to parse the CSV file, which is done by Panda's CSV-parser. It was chosen since it allows to parse the file in chunks as opposed to reading the whole file into memory at once and then parsing it. This approach allowed us to reduce the memory usage and resulted in a better overall performance. The size of the file is going to be around 1.8 GB for about 10 000 traces. NumPy is used to implement the algorithm itself, since arithmetic on NumPy arrays is far more efficient than on Pandas Dataframes. This was determined by using a Python profiler. Once we have the set of differential traces, we sort it by the sample point with the biggest magnitude in each differential trace.

4 DPA Results

The original DPA algorithm failed to recover the AES key with 10 000 traces and in general appeared to be inferior to the modified multi-bit version. In order to account for errors in the measurement process, additional sets of data were captured and the differential traces were compared. These traces showed, as expected, some minor differences, but overall the results were similar. Increasing the number of traces to 15 000 offered little improvement and the number of traces would likely need to be orders of magnitudes bigger.

However, when ranking the differential traces by their peaks, the original algorithm consistently places the correct subkey in the upper half. This suggests that the algorithm can, at the very least, be used in order to guess the key in an educated manner, which is more efficient than an exhaustive search.

The modified version fares far better and is capable of recovering the full key with 10 000 traces. It should be noted that the modifications come with an increase in time complexity. The original algorithm takes about 24 minutes to attempt to recover the roundkey, while our version recovers the roundkey in 140 minutes. Of these times roughly one minute is spent just reading the file. However, the scripts used to extract the key could still be further optimised by parallelising operations.

Metrics described in [8] were used in order to evaluate and compare the performance of the algorithms. The results in table 1 were acquired by running the DPA attack multiple times on a full key and then treating each subkey attack as an independent result. The numbers are averages and the number of guesses needed refers to attacking a byte of the key, not the whole key.

The table makes it clear that our modified approach is more efficient even when we are only using 1000 traces. It should also be noted that the original method reaches a peak ratio with 8000 traces comparable to the modified approach with 1000 traces, but it still fails to reach a comparable number of guesses needed. Another interesting insight is that we actually experience a temporary drop in correctly identified key bytes as we add more traces. The modified algorithm is capable of instantly identifying all key bytes when using 8000 traces but fails to do so when presented with 9000. This is likely just a result of noise, the overall trend is clearly that the attacks improve as more traces are used.

5 Conclusion

We have demonstrated that AES128 can be easily broken by using multi-bit DPA and that this version offers significant benefits over the original DPA algorithm. The fact that the Arduino offers no hardware implementation of AES requires authors to mitigate any attacks in software. This likely decreases the performance of the code and makes the development of cryptographic routines more difficult. However, physical access and tampering is required to perform these attacks and contrary to devices such as smartcards, the Arduino is not commonly deployed in scenarios where attacks with these requirements pose a threat.

Kocher's original DPA algorithm fares fairly poorly when modified to run with AES and was unable to recover the key. A possible explanation might be that the original algorithm was published on DES, which uses different S-

Traces	Ratio		Guesses	
	Original	Modified	Original	Modified
1000	0.6373	1.0736	71.1875	7.7500
2000	0.7983	1.4482	52.3750	2.7500
3000	0.8627	1.7709	41.9375	1.7500
4000	0.9640	1.9955	43.1250	1.3750
5000	1.0174	2.1958	36.1250	1.1250
6000	1.0390	2.2209	35.6250	1.1250
7000	1.0354	2.2792	34.0000	1.1250
8000	1.0730	2.3096	34.2500	1.0000
9000	1.0559	2.3493	32.1875	1.0625
10000	1.0574	2.3586	30.8125	1.0000

Table 1: Comparison of original and modified DPA. Traces is the number of traces used for the algorithm. Ratio is the average ratio of the correct peak to the highest incorrect peak. Guesses is the average number of guesses needed in order to find the correct subkey.

boxes from AES and that the original attack targeted a smartcard while we are targeting a microcontroller. The modified mutli-bit variant that this project produced coped far better and achieved better performance with one tenth of the data. This variant takes roughly six times as long as the original algorithm to recover the key, but has no other drawbacks and can be applied in the same scenarios as the original algorithm, which is an advantage when compared to other algorithms proposed by the research community. The performance gap may be further narrowed by parallelising the calculation of differential traces for different bits.

Hardware was one of the limiting factors for the project and care should be taken that the available instruments are capable of capturing the desired data in a quick and efficient manner. This also implies that only an attacker with access to high-end equipment would be capable of performing this attack on devices which have a significantly higher clock frequency, such as desktop computers.

References

[1] Arduino. *Arduino Uno Rev3*. 2019. URL: `https://store.arduino.cc/arduino-uno-rev3`.

[2] Daniel J. Bernstein. 'Cache-timing attacks on AES'. 2005.

[3] Eric Brier, Christophe Clavier and Francis Olivier. 'Correlation Power Analysis with a Leakage Model'. In: *Lecture Notes in Computer Science Cryptographic Hardware and Embedded Systems - CHES 2004* (2004), pp. 16–29. DOI: `10.1007/978-3-540-28632-5_2`.

[4] Joan Daemen and Vincent Rijmen. *AES Proposal: Rijndael*. 1999.

[5] Tektronix Experts. *Scope tip: Use Bandwidth Limiting to Reduce Noise on Captured Signals*. 2014. URL: `https://www.tek.com/blog/scope-tip-use-bandwidth-limiting-reduce-noise-captured-signals`.

[6] Paul Kocher, Joshua Jaffe and Benjamin Jun. 'Differential Power Analysis'. In: *Advances in Cryptology — CRYPTO' 99 Lecture Notes in Computer Science* (1999). `https://www.paulkocher.com/doc/DifferentialPowerAnalysis.pdf`, pp. 388–397. DOI: `10.1007/3-540-48405-1_25`.

[7] Thanh-Ha Le et al. 'A Proposition for Correlation Power Analysis Enhancement'. In: *Lecture Notes in Computer Science Cryptographic Hardware and Embedded Systems - CHES 2006* (2006), pp. 174–186. DOI: `10.1007/11894063_14`.

[8] Jing Pan et al. 'Improving DPA by Peak Distribution Analysis'. In: *Selected Areas in Cryptography Lecture Notes in Computer Science* (2011), pp. 241–261. DOI: `10.1007/978-3-642-19574-7_17`.

[9] B. Poettering. *AVRAES: The AES block cipher on AVR controllers*. Mar. 2007. URL: `http://point-at-infinity.org/avraes/`.

[10] Tektronix. *Digital Phosphor Oscilloscopes Programmer Manual*. 2008. URL: `https://www.tek.com/oscilloscope/tds3014b-manual/tds3000-tds3000b-tds3000c-series`.

[11] U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology. *DATA ENCRYPTION STANDARD (DES)*. Gaithersburg, MD, USA: U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, Oct. 1999. URL: `https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf`.