Project Code: BS21003, WJM5001

Advanced Cryptographic Power Analysis

A Major Qualifying Project Report

Submitted to The Faculty

of the

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree of Bachelor of Science in
Computer Science
and
Electrical and Computer Engineering

by

Christopher Kevorkian

Jacob Tanenbaum

Approved:

Dr Berk Sunar

Dr William J. Martin

Abstract

This project seeks to advance the differential power analysis (DPA) work of Hnath and Pettengill [4]. Using their work as a foundation, we implement a similar power analysis attack on the Advanced Encryption Standard (AES). We perform a rigorous evaluation of the effectiveness of the attack according to the criteria used by the DPA Contest [3]. In addition, we seek to improve the effectiveness of the attack through the consideration of different power models and trace alignment strategies. These attempts result in slight improvements in attack effectiveness in some cases.

In addition, we seek to attack AES by adapting the differential template attack (DTA) presented by Karakoyunlu and Sunar [5]. Although originally designed to attack physically unclonable functions, we show the DTA can be used against AES. We create an implementation of the DTA to attack AES and find our implementation effective, but unable to reduce the key space with enough certainty to be useful in its current state. We also present some theories on how to improve the DTA to make it more reliable and effective.

Contents

List of Figures

List of Tables

1 Introduction

Cryptography has become an almost ubiquitous part of everyday life. Every time someone logs into a computer, shops online or connects to a secure wireless network, cryptographic protocols are used to protect sensitive information from third parties. Many of these digital communication systems are secured using symmetric key encryption. Symmetric key encryption is a commonly used encryption method in which the two communicating parties share a predetermined key used to encrypt and decrypt the original message (plaintext). This method of encryption is widely used to secure digital communication channels. If an attacker were to compromise a cryptosystem by acquiring the key, the attacker could send fraudulent messages, clone a device, or listen to the messages sent on the communication link thought to be secure. Since the key is of finite length, an attacker could attempt to acquire the key by trying to decrypt a message with all possible values for the key. To protect against this, the key is designed to have a large enough set of possible values (key space) such that exhaustive search of those values could take decades [8].

In this project, we sought to explore the susceptibility of an FPGA implementation of Advanced Encryption Standard (AES) to a class of side-channel attack known as differential power analysis (DPA). A side-channel attack seeks to acquire information about the key of a cryptosystem by exploiting information leaked through physical channels [8], the three most important being timing attacks, power analysis attacks, and electromagnetic attacks [7]. This side-channel leakage can come from flaws in implementation or limitations of hardware [6] and allows an attacker to acquire the encryption key by analyzing the device as it operates. DPA involves gathering information about the power consumed by a device performing encryptions and analyzing that information to glean information about the encryption key.

DPA requires an attacker to have physical access to a device in order to collect the necessary power information, but this is not an uncommon occurrence. Consider the scenario of an ATM communicating with a server at a bank. The ATM encrypts the PIN numbers and financial data which are decrypted by the banks server. Sending the encrypted message (ciphertext) protects the information even if it is intercepted by a third party. However, many ATMs are in locations with limited physical security. If an attacker were able to insert a probe and monitor the power consumed by the device he could then use DPA to attempt to extract the cryptographic key. This would enable him to decrypt all the communications between the ATM and the server.

In this project, we acted as attackers and attempted to extract information from a device. We used the information provided by the DPA contest v2 organized by the VLSI research group from the Communications and Electronics Department of Telecom ParisTech. The contest provides participants with large sets of power information and reference documentation on the cryptographic implementation [3]. We used this data to perform two DPA side-channel attacks and analyzed the effectiveness of those attacks. The first attack was a standard DPA attack similar to the original DPA attack designed by Kocher et al. to attack DES [6]. The second attack was a Template Attack, based on the work of Karakoyunlu and Sunar used to attack a physically unclonable function [5]. The information gathered from performing these attacks may someday help to design implementations that are not susceptible to DPA and can better secure sensitive information.

2 Background

2.1 Advanced Encryption Standard

The Advanced Encryption Standard is a symmetric-key encryption algorithm. Symmetric-key encryption requires the two parties sharing data to first establish a shared cryptographic key. AES consists of specialized implementations of the Rijndael algorithm designed by Rijmen and Daemen. There are three variations of the algorithm, differing only by the length of the encryption key. The different key lengths are 128, 192 and 256 bits, called AES-128, AES-192 and AES-256 respectively. The larger key size means a larger key space which makes the algorithm less susceptible to exhaustive search attacks. As this project deals solely with AES-128, future references to AES will be referring to AES-128. The AES algorithm consists of ten rounds of repeated mathematical operations that make use of the cryptographic key to convert the plaintext to ciphertext. Figure 1 shows the steps performed within each round to encrypt the plaintext.

The inputs to AES are a 128-bit plaintext and a 128-bit key. The plaintext is stored as a 4x4 array of bytes and referred to as the state. Each successive round of AES modifies the state, eventually producing the ciphertext. Each round, the state undergoes four operations: SubBytes, ShiftRows, MixColumns and AddRoundKey. The first three are operations performed solely using the state, but the final operation combines the state with the round key. The Rijndael key schedule is used to generate each round key from the original 128-bit key.

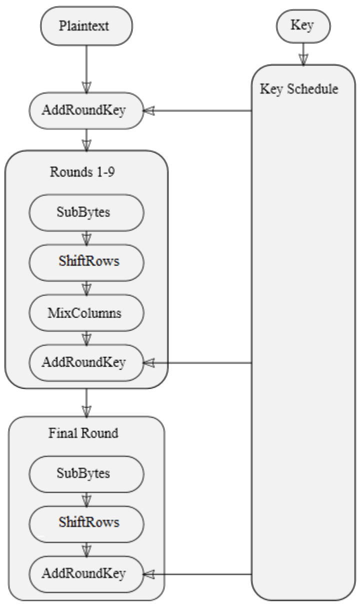


Figure 1: Flowchart Depicting the steps taken by AES to encrypt a plaintext.

Here we provide a breakdown of each of the transformations performed within each round of AES.

2.1.1 SubBytes

The SubBytes transformation modifies each byte of the state according to a substitution table (S-box). Each byte of the state is used as an index to the S-box and replaced with the value corresponding to that index. Due to the construction of the S-box, this results in a non-linear transformation. Non-linearity must be assured so that AES is not vulnerable to attacks based on simple algebra. Additional information on the construction of the S-box can be found in the FIPS AES publication [1].

2.1.2 ShiftRows

The ShiftRows transformation cyclically left shifts each row of the state. Each row is shifted r bytes where r is the index of that row. For the first row, r = 0 and the first row is therefore not shifted. This shift causes each column of the state to contain a single byte from each of the original columns.

2.1.3 MixColumns

The MixColumns transformation operates on the state column-by-column. Each column undergoes matrix multiplication with a fixed matrix. This matrix multiplication produces new column values as a linear combination of the original column values. More detail regarding this calculation can be found in the FIPS AES publication [1].

2.1.4 AddRoundKey

The AddRoundKey transformation adds the round key to the state through a bitwise XOR operation. The round key is generated according to the Rijndael key scheduler and has the same number of bytes as the state. For the initial AddRoundKey transformation performed prior to the first round, the round key used is the original key.

2.2 Differential Power Analysis

DPA is a type of side-channel attack that analyzes information about the power consumption of a device over multiple encryptions to obtain the encryption key. For DPA to work, the value of the key must influence the power consumption of the device. Modern cryptographic devices, including microcontrollers and FPGAs, are implemented using CMOS logic. Due to the nature of transistors that make up the logic blocks for these devices, the power consumption of the device correlates to the values of the bits the device is processing. This correlation, combined with information about the inputs and/or outputs of specific encryptions, makes it possible for DPA to extract secret keys [6].

2.2.1 Power Trace

The first step of any DPA attack is to acquire information about the power consumption of the device as it performs encryptions. A common method of doing this is to use a digital oscilloscope to measure the instantaneous voltage drop across a small chosen resistor which is connected in series from the power supply of the device to ground. The set of all the instantaneous voltage measurements for a given encryption form a plot of voltage measurements as a function of time. We can use the voltage data and resistance to calculate the power consumed. This set of data when viewed as a function of time is known as a power trace. An example of a power trace can be seen in Figure 2.

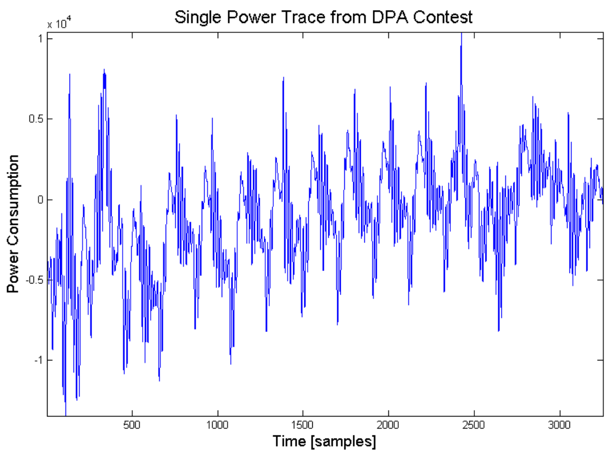


Figure 2: Example of a power trace from the DPA contest v2

$$P_{pow} = P_{data} + P_{opt} + P_{noise} + P_{const}$$

Figure 3: Sum that describes the components of the power consumption for an instantaneous power measurement.[7].

Visual inspection of the power trace in Figure 2 provides some limited information about the encryption algorithm. Of particular interest is the regularly repeating pattern that corresponds to the repeated cycles of AES. This is just one example of how information about the encryption is leaked through the power consumption of the device. This leaked information is what makes DPA possible. However, the signal in Figure 3 is not a very clean signal and appears to be a combination of signals. Prior research has shown that we can consider each point of the power trace to be modeled by the sum in Figure 3.

The total power consumption P_{pow} is the sum of the data dependant P_{data}, operation dependant P_{opt}, noise dependant P_{noise}, and constant P_{const} power contributions to the overall power consumption. Because we are dealing with power analysis the most important elements of the sum are P_{data}, P_{opt}, and P_{noise}. P_{opt} and P_{data} provide the link between power consumption and the data being processed, while P_{noise} acts as a barrier to our attack. P_{noise} adds randomness to the power traces. Because of P_{noise} two power traces will not be the same even if operating on the same data. Electronic noise comes from numerous sources. It could come from the nearby electronics, ambient radiation, changing temperature, or switching characteristics of the internal transistors to name a few. It is an important goal in trace acquisition to eliminate as much noise as possible. The main reason power analysis attacks need to use many traces is to effectively zero out the contribution made by the noise component to the power consumed. P_{const} is not important because it represent unexploitable information such as the power needed for the device to be operational and does not help or hinder the process of power analysis.

2.2.2 Standard DPA

The DPA attack presented by Kocher et al. provides us with a way to acquire the full 128-bit key by determining a single byte of the key at a time [6]. The correlation between power consumption and each byte of the key makes it possible to determine if a single byte guess matches the correct value of the key byte. This fact makes it possible to test all possible values of the key in many fewer tests than exhaustive search. Instead of testing 2^{128} possible keys as would be required by exhaustive search, we only need to test 2^8 possible values for each byte of the key. Since there are 16 bytes, there are a total of 2^{12} combinations to be tested instead of the original 2^{128}.

This benefit stems from the ability to use the power traces to confirm whether a byte guess is correct or not. To do this, we first must establish how the value of the key affects the power consumption. As stated in Section 2.2.1, the power consumption of a device is correlated to the values of the data being operated upon. If we consider an arbitrary cycle in the AES encryption, there is a state S being modified. The intermediary values of S therefore have some correlation to the power consumption of the device. To perform DPA, we must make an assumption about this correlation. For example, we could assume that an intermediate value with more high bits will consume more power than an intermediate value with more low bits. This assumption about the relationship between the intermediate values and the power consumption is an example of a "power model". To determine if a particular key byte guess is a candidate for the correct value of the key byte, we determine if our key byte guess results in correlation between our power model and the measured power consumption. If there is a strong correlation between our predicted power model and the measured power consumption, it is likely our key byte guess is correct.

To determine if this correlation exists, we can calculate an intermediate value and use it to calculate the result of the power model. Using our key byte guess and the known ciphertext or plaintext, we are able to calculate the intermediate value. With the intermediate value in hand, our chosen power model can be used to classify a particular trace as either high power consumption or low power consumption. All traces classified as high are placed in group A and all traces classified as low are placed in group B. With the traces grouped accordingly, we calculate the difference between the groups where diff= $|mean(power(A)) - mean(power(B))|$. If diff is a large value, we can conclude that a correlation exists between the power model and the measured power consumption for the current key byte guess.

If our guess for the key byte is not correct, then the intermediate value will have been calculated incorrectly, and the traces would have been placed randomly into the lower and higher power consumption groups. If we calculate diff= $|mean(power(A)) - mean(power(B))|$ and diff is close to 0, then we conclude that the key byte guess is incorrect. Experiments have shown that these are reasonable conclusions regarding the key byte guess and the power model correlation [6][4].

This method of determining if a particular guess is correct enables us to test all 256 possible values of a particular byte of the key and locate the correct value. This process can then be repeated for every single byte of the key until the entire key has been recovered.

2.2.3 Template Attack

Template attacks are another type of side-channel attack. Template attacks attempt to recover the cryptographic key in two stages. An attacker first uses a device identical to the one under attack to build a profile. The attacker then compares this profile to side-channel information gathered from the device under attack in order to acquire the key [2][9]. Template attacks target cryptographic protocols in which the key may be temporary or there is a limit on the number of times an operation can be performed [9]. Limiting the number of encryptions performed during an attack will cripple the abilities of DPA which seeks to reduce the impact of noise on the attack traces through averaging [6][2]. Template attacks seek to build a model of the device before an attack is performed so that no matter the key the attack can compromise it with relatively few traces.

Template attacks can be performed using different types of side-channel data (power, electromagnetic radiation, etc), but for the purpose of our project, we will deal solely with power-based attacks. When profiling a device, an attacker acquires side-channel information in the same manner as when performing a DPA attack (see Section 2.2.2 Standard DPA). Since the attacker is in control of the profiling device, he knows the plaintext, key, and ciphertext of each encryption. By combining this with the gathered side-channel data, the attacker creates a detailed model of the side-channel noise [2]. This model is referred to as a template. After the attacker has gathered enough templates [9], he uses them to perform the attack.

To perform the attack, the attacker classifies traces according to the acquired templates [9]. When a power trace is acquired from the device under attack, it is compared to each template to determine which template most closely matches the captured trace. By performing this classification, the attacker seeks to significantly reduce the number of possible key values or possibly even determine *the* used key value [9].

2.3 DPA Contest

2.3.1 Overview

In 2008, the VLSI research group from the Communications and Electronics Department of Telecom ParisTech officially launched the first edition of the DPA Contest. Their goal was to provide an objective way for researchers to compare DPA attack algorithms [3]. In the past, a number of factors made it quite difficult to compare independently designed attacks, including differences in the trace acquisition hardware, cryptographic implementation, noise, etc. The DPA contest seeks to remove these uncontrollable variables that exist when attempting to compare and benchmark independently created DPA attacks. To accomplish this, the research group collected large sets of power traces from a device under controlled conditions and made these power traces publicly available. They also created a testing framework which included detailed evaluation criteria to quantify the effectiveness of an attack and made this testing framework available along with the power traces. By taking care of the trace collection and providing detailed evaluation criteria, the research group made it possible for independent researchers to develop attacks and compare them objectively.

The DPA contest has had two iterations, and two more are scheduled for the near future. In the first, the cryptographic algorithm under attack was the Data Encryption Standard (DES). In the second, the cryptographic algorithm under attack was AES-128. We used the power traces provided in version two to form our DPA implementations.

Version two of the DPA contest included three databases of traces. Each database contained a large number of power traces along with the key, plaintext and ciphertext associated with each power trace. When performing attacks, only the plaintext and ciphertext are used, but the keys are provided so results can be verified. Two of the databases, the public and template databases, were made publicly available for use in designing, testing, and optimizing proposed DPA implementations. The third database was reserved and only used by the organizers of the DPA contest to test and verify submissions. The private database was kept private to prevent the inclusion of trace specific optimizations and other forms of cheating in the attack software.

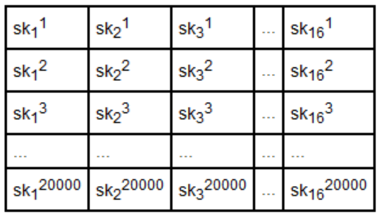
The public database contains power traces for developers to test their DPA attacks. It contains a total of 640,000 power traces, divided into 32 groups of 20,000 traces. Each group was generated using a single unique random key to encrypt the 20,000 random plaintexts. To test a DPA implementation, a developer would run their attack on a single group in an attempt to acquire the associated key. Having 32 different groups makes it possible to verify if an attack performs equally well on multiple sets of data.

The template database contains power traces for developers to perform profiling for template attacks. The database contains a total of 1,000,000 power traces, each acquired through encrypting a random plaintext with a random key. As described in Section 2.2.3, template attacks require profiling a device with a large variety of inputs, and this database provides the necessary traces.

The private database has a structure identical to that of the public database. However, the private database is never made available to DPA contest participants. When attacks are submitted to the contest, the private database is used as the input to the DPA implementation. The performance of the implementation when run on the private database is how each attack is judged.

2.3.2 Format of Attack Results

In order to make it possible to objectively compare independently developed attacks, attack results must be formatted according to strict guidelines. These guidelines were developed to enable comparison of several aspects of the performance of an attack. After processing each trace, an attack must provide all of the information the attack has acquired so far about the secret key. Specifically, for each byte of the secret key a 256 by 1 matrix must be provided. This matrix must contain all 256 possible values of that byte, sorted in order of likelihood, with the most likely first. If the attack has processed all 20,000 traces associated with a single key, it should have an output matrix in the form shown in Figure 4.



sk_1^1	sk_2^1	sk_3^1	...	sk_{16}^1
sk_1^2	sk_2^2	sk_3^2	...	sk_{16}^2
sk_1^3	sk_2^3	sk_3^3	...	sk_{16}^3
...	...	...	...	...
sk_1^{20000}	sk_2^{20000}	sk_3^{20000}	...	sk_{16}^{20000}

Figure 4: Result Matrix that would contain the probability for every key after each trace is analyzed.

Each cell of this matrix contains a sk_s^t. Each sk value is the 256 by 1 vector described above containing all possible values for a particular subkey (single byte). The subscript s corresponds to the index of the byte, and the superscript t corresponds to the number of traces that have been processed thus far. This result matrix provides essential information for evaluating attacks submitted to the DPA contest.

2.3.3 Evaluation Criteria

In order to compare the attack results from independently created attacks, the contest uses several standardized performance metrics. These metrics are as follows: partial success rate, partial guessing entropy, global success rate, execution time and memory footprint. The metrics are calculated from the output matrices and data returned from running an attack. If an attack is run on all 32 keys within either the public or private databases, there are 32 result matrices available to calculate the performance metrics.

The partial success rate shows the efficiency of an attack at determining the correct value of a single byte of the key. This metric is calculated independently for each byte of the key. Information from all 32 result matrices is combined to determine the probability that an attack has acquired the correct byte value after processing some number of traces. Figure 5 shows the partial success rate from one of the submissions to the DPA contest [10].

The partial guessing entropy seeks to capture the average number of guesses it would take to guess the value of a single byte. Just like partial success rate, this metric is calculated independently for each byte of the key. Information from the 32 result matrices are combined to determine the average number of guesses it would take after n traces. Figure 6 shows the partial guessing entropy from one of the submissions to the DPA contest [10].

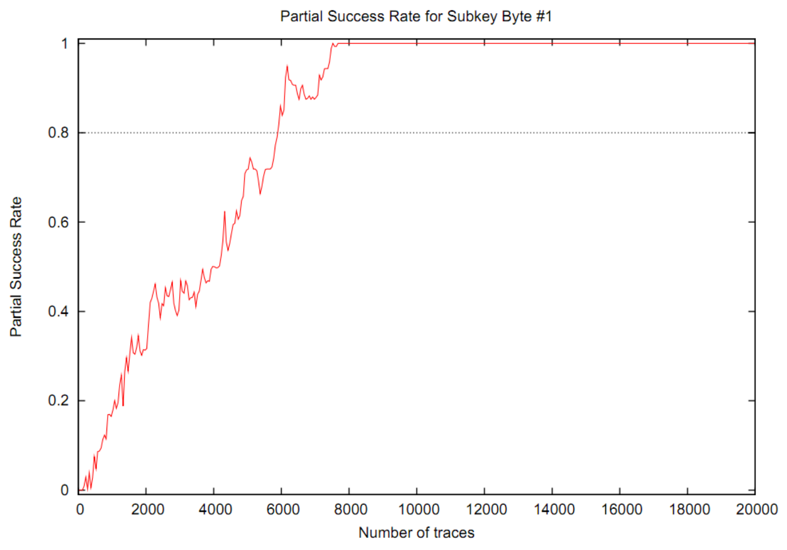


Figure 5: Example of a graph showing Partial Success Rate

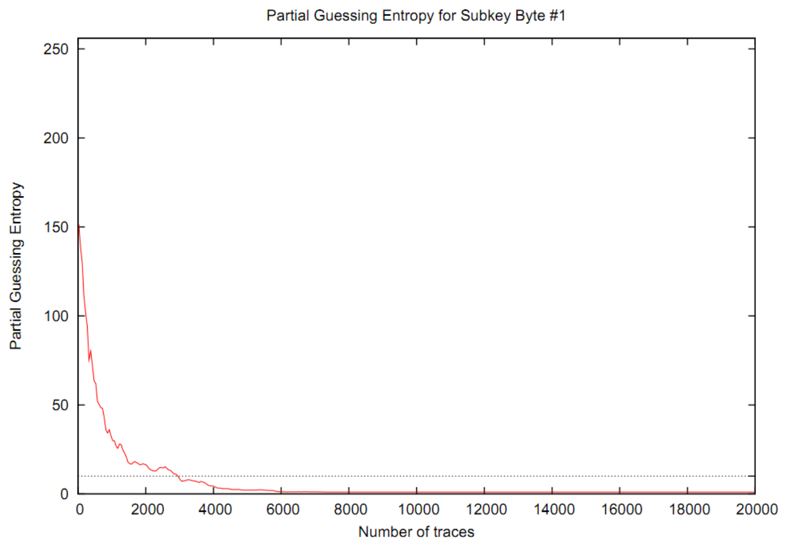


Figure 6: Example of Partial Guessing Entropy

3 Standard DPA Attack

3.1 Summary of Existing Attack

In the previous MQP done by Hnath and Pettengill[4], they designed and implemented a DPA attack according to the DPA method presented by Kocher et al.[6]. Their attack targeted the final round of AES, specifically the transition of the state from the output of the ninth round to the final ciphertext. The final round was chosen over one of the middle rounds as it only required the reversal of a single round of AES. Furthermore, the final round omits the MixColumns operation, further simplifying the reversal of the round.

Their attack made use of the MATLAB framework provided by the DPA contest to interface with the power traces provided by the contest. This framework provided a single for-loop that would iterate over each of the 20,000 traces associated with a single key value from the Public database. Each iteration, a new trace would be available along with the plaintext and ciphertext associated with the trace. Since their attack targeted the value of the state coming into the 10th round, the ciphertext was used along with each key guess to calculate potential values for a single byte of the state. This byte was then compared to the corresponding byte in the ciphertext to determine the number of bit transitions between the two bytes. The power model they used is based on the knowledge that when a bit transitions, it consumes more power than a bit that stays the same value. Traces which had more bit transitions were therefore qualified as having higher power consumption than traces with fewer bit transitions.

One manner in which their implementation deviated from the original DPA attack described by Kocher et al. was that they chose to use the Pearson correlation coefficient to correlate the number of bit transitions to the power consumption of the traces. This likely provided them with a more accurate correlation than merely dividing traces based on whether they had more or fewer than 4 bit transitions.

For the purpose of their research, Hnath and Pettengill determined an attack to be successful after their attack produced the correct attack byte 100 traces in a row. Each iteration of the primary loop, the key guess that produced the greatest correlation between the power consumption and the number of bit transitions was considered the most likely candidate for the key value. This candidate value often fluctuates significantly until enough traces have been processed and averaged to eliminate inaccuracies caused by noise. When this candidate value remained the same after processing 100 sequential traces, they considered the candidate to be the correct value for the current byte of the key. After testing this on the entire public database, they determined that this method produced the correct key byte for all of the bytes of each key. Furthermore, their attack was able to reach this point after processing an average of 1000 power traces.

3.2 Power Models

CMOS devices depend on the constant transitions of internal bits from zero to one and vice versa. The device requires variable amounts of power to perform these transitions. As shown in Figure 3 every instantaneous power measurement includes leaked side-channel information that directly correlates to the operation and data being operated on.

We use the intermediate value to determine if there is a correlation between our power model and the measured power consumption. If there exists some correlation between the power model and power consumption there is a high probability that the key guess is the same as the actual key. We examine Hamming weight and Hamming Distance as viable power models in this project.

3.3 Hamming Weight

Hamming weight is the simpler of the two power models that we examined. The hamming weight of a bit string is the number of '1' bits in the string. This would mean that all the appearances in the number '1' in the bit string of the intermediate value contributed to power consumption regardless of the value of that bit in the key guess. The use of this model assumes that the power consumption of the device is based on the number of bits switched from the zero bit string. Since all operations performed do not transition from the zero string this model has a relatively weak correlation to the power actually consumed by the device. Due to this generally weak correlation to the actual power consumption, this model often oversimplifies the circuit under attack and other power models should be considered. A major benefit of this model is the lack of implementation information needed to successfully implement this power model.

3.4 Hamming Distance

Hamming distance is the measurement of the difference between two bit strings. It is a logical step forward from hamming weight as it is a measure of the bits flipped from the previous value as opposed to the zero string The hamming distance and hamming weight are the same if the previous value happens to be the zero string. It models the power consumption of a device by the transitions that occur on observed data. We XOR the the previously observed data value with he current data value add the number of 1's in the result string and that is the Hamming distance. See Figure 7.

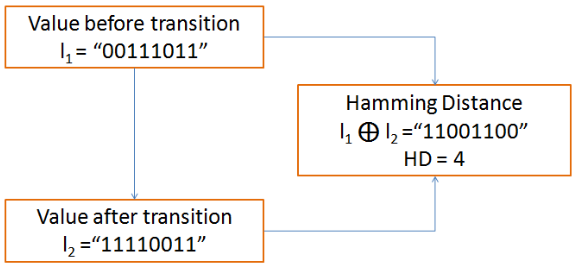


Figure 7: Hamming distance between two transitions on a bus

The main drawback of this model is the assumption that values that remain unchanged during the periods of measurement do not contribute to the overall power consumption of this device. In our tests, the Hamming distance power model offered a much better approximation of the power consumption of the device and when implemented greatly increased the effectiveness of our power analysis attack. This model can be further enhanced with knowledge of the device such as weighting the transitions of selected bits so they have a larger effect on the power consumption.

3.5 Trace Alignment

Methods of collecting power traces from devices under test are imperfect. These imperfect collection methods can lead to unwanted time shifts. In real systems there are several reasons that power traces could be misaligned. A device under attack could have counter measures that add dummy operations, trigger signals that activate the power trace could start data collection at a different point relative to processing the data [7], or temperature variations that slightly alter clock speeds could shift the position of critical sections of the power trace in time.

As the DPA contest implementation has no countermeasures, we believe that the observed timing offsets come from misfired trigger signals and temperature variation. These shifted regions introduce error that can be mitigated in DPA through either trace alignment or a larger trace set. As the name suggests, trace alignment is using information embedded in the trace to remove the jitter caused in data collection. The sample data provided by the DPA contest were generally well aligned with time shifts of a couple of samples in a majority of cases.

We attempted two basic trace alignment strategies to realign the curves. The initial idea was to take the max value of the whole trace and assume that the location of max power consumption is fairly characteristic across all traces in a given set. That method proved ineffective; the global max of the curves are so far apart that it not only did nott provide an increase in performance but no trace set was able to produce a result. The traces were different enough that the global maximum was not a good way to determine an alignment to the traces or it could be possible that regions are out of alignment because of differences in times to do specific operations. The more successful strategy was to align based on the max of the sensitive area that our attack is targeting. Most trace sets see a boost in performance because the aligned location is a local max and we find that aligning a small segment is better than finding a location in which to align the whole trace.

3.6 Results

The method of trace alignment used was very naive in that it used some information derived from a small sample of trace sets and attempted to use them across all the trace sets. We believe the strategy simplistic because some trace sets saw efficiency gains and others became less efficient. Referring to Table 1, we see the number of traces needed to complete the basic attack and the aligned attack. The subset that saw a performance benefit from this alignment strategy saw a decrease in the number of traces required to retrieve the key of about 12%.

Hence there are some sets of traces that alignment helped, some that it is detrimental, and some that there is no change. referring to table 1 shows the number of traces in the basic attack and the number of traces in the aligned attack. The places that the alignment did improve the number of traces about 12%. It turns out that whenever a trace set does not complete on the general attack it also does not complete with the trace alignment.

The naive trace alignment method implemented generally had no effect on the running on the trace sets which would suggest that there is a common property to the keys or plaintext of those trace sets that improved.

3.7 Coding Practices

One goal of this project is to produce code that will be used by future students to investigate and improve our attack implementations. This requires writing good, maintainable code that future users can quickly pick up and understand in order to modify easily without introducing undesired side effects. In the context of software systems that means having good code structure and documentation. Ideally someone that is not intimately familiar with the specific attacks or the inner workings of AES should be able to quickly identify what each piece of code is doing. We accomplished these goals by clearly delineating and commenting the major steps of our attack, which we will describe shortly.

The existing DPA implementation did not follow some basic coding practices that convey understanding and readability. The structure was a single monolithic file that performed the entire attack, as well as initial setup. There were some general comments, but much of the code required very detailed analysis to determine its intended purpose. The monolithic structure made it difficult to make changes to aspects of the attack without causing unintended side effects. Converting the existing code to a more manageable and documented form was an important part of this project. The primary actions done to improve the existing code were segmentation and documentation.

After fully understanding the existing implementation, it was possible to break the system down into several separate components. The basic sections of the DPA attack are the power model, performing correlation, and trace management. We identified these as the major components of performing DPA and split them each into separate scripts. The power model determines the relationship used to compare power consumption to the intermediate AES values. The method of correlation determines how the information from that comparison is used to obtain the key. Finally, the trace management handles acquiring data from the trace database and looping through each trace in turn.

In this project, several different power models were examined. Having separate compo-

Table 1: Shows the number of traces needed to recover the key with the basic DPA attack and with the basic trace alignment for all trace sets.

Trace Set	Basic DPA Attack	Basic Trace Alignment
1	7187	N/A
2	15791	13008
3	8786	7271
4	12480	12892
5	N/A	N/A
6	10592	10615
7	11003	12333
8	8480	9054
9	7995	8440
10	N/A	N/A
11	8130	8103
12	7790	7078
13	7536	7059
14	12086	12086
15	9738	9979
16	11936	12016
17	N/A	N/A
18	4817	4060
19	7995	8008
20	10704	9252

nents allowed for different power models and correlation methods to be substituted with relative ease. Clearly abstracting these blocks of code also allows for better understanding of the flow of the code and should make it easier for others to use and maintain.

4 Differential Template Attack

4.1 Description

Our implementation of a template attack is modeled after an attack by Karakoyunlu and Sunar called a differential template attack (DTA) [5]. The original DTA was designed to recover the output of a physically unclonable function (PUF) being transferred over an internal bus. We sought to implement a DTA and use it to recover the AES key. Similar to DPA, our goal was to acquire a suitable key guess for a single byte of the key with the assumption that a similar process could be duplicated in a "divide and conquer" fashion until all bytes of the key have a calculated key guess.

We decided to focus our attack on the initial round of AES as the original DTA was designed to focus on an XOR operation performed at the start of a cryptographic algorithm. Our goal was to profile the device for all possible outputs of this initial XOR operation. To ensure coverage of all possible values of the first byte, we took all the traces from the template database in which the first byte of the plaintext was 0x00. This enabled us to test all possible outputs of the XOR by choosing traces out of this subset with the appropriate key values. To perform the profiling, we chose the section of the power trace corresponding to the first AES round after the initial AddRoundKey operation. All computations performed in this round directly depend on the output of the initial AddRoundKey, meaning the power consumption is likely heavily dependent upon this value. We chose the subsection of the power trace with the largest distribution of values as this indicates likely dependence between power consumption and the processed data.

After gathering the power traces for all possible byte values of the output of the initial AddRoundKey operation, we created a template from each trace. To create the template, we calculated the l^2 norm [11] of the previously chosen section of the power trace. From the set of all templates, we then chose a subset of "distinguished templates". We chose the distinguished templates based on which templates had l^2 norms most different from the l^2 norm of all other templates. The original DTA sought to accomplish two things through using distinguished templates: 1) To increase the probability that we correctly match traces to templates. 2) To reduce the computation requirements of the attack by reducing the number of templates.

When performing the attack we determined if each collected trace matched to a distinguished template. A trace was said to match a distinguished template if the l^2 norm of the trace was closer to the l^2 norm of that distinguished template than it was to the l^2 norm of any other distinguished template. In order to prevent false-positive matches, we placed a maximum threshold on the allowable distance between a matching trace and template. Each collected trace that we determined matched a distinguished template was then grouped with that distinguished template. Figure 8 shows the process of matching traces, including the associated plaintext, to the distinguished templates, which contain a known key and plaintext. Once a trace has been matched to a distinguished template, we assume that the intermediate value of the trace, the output of the initial AddRoundKey, is the same as that of the template. With this assumption, we are able to calculate a potential key byte associated with each trace. The potential key byte is calculated according to Equation 1 where k_t and p_T are the key and plaintext byte associated with template, and p_t is the plaintext associated with the power trace.

$$k = k_T \oplus p_T \oplus p_t \tag{1}$$

After all of the available traces have been matched with an appropriate distinguished template, there are two primary methods we considered to form our key guess. The first is through calculating the intersection of the different groups of potential keys. If K_{T_i} is all potential keys grouped with distinguished template T_i, then our key guess would be $k = K_{T_1} \cap K_{T_2} \cap ... \cap K_{T_n}$ where n is the total number of distinguished templates. The second method of forming a key guess is through determining the most frequently occurring key value of all potential keys. If the templates are matching too inconsistently against the appropriate traces, it is possible that the correct key will not be the only value in the intersection nor the most frequently occurring guess value. In this case, as long as the correct value is within the top most frequent values it is still possible to acquire the key. For this to happen, the DTA must be able to narrow the possible values enough for brute force to be feasible on all bytes of the key. At the same time, it needs to maintain a high level of certainty that the correct key is within the reduced set of possible key values.

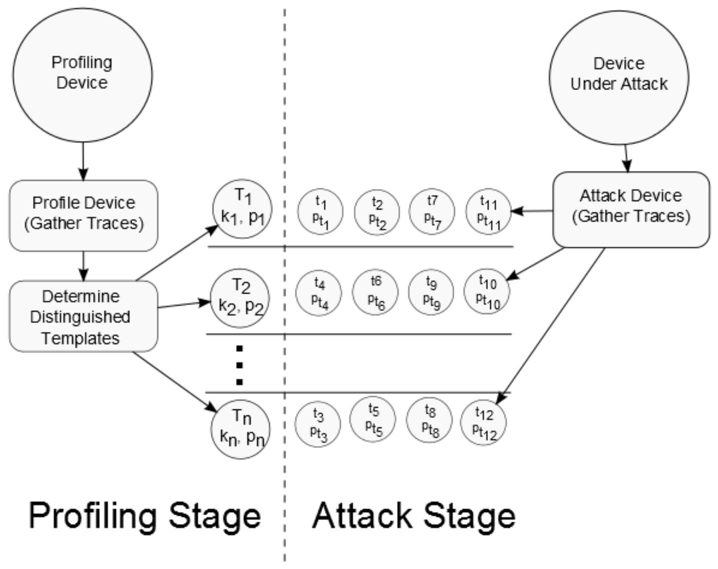


Figure 8: Flowchart Showing Traces Matching to Distinguished Templates

4.2 Feasibility Study

Due to the nature of template attacks there needed to be specific data available to perform the profiling. Specifically we needed all 256 possible values of the first byte of the key to be present when the first byte of the plaintext is fixed at zero. To be reasonably assured all of the necessary traces were available in the template database we turned to the coupon collector's problem. According to the coupon collector's problem, the expected number of traces needed to acquire at least one trace with each desired value can be given by $O(n \ln(n))$ where n is the number of unique values needed. Using this guideline, we need approximately 617 total traces with 0x00 as the first plaintext byte to be relatively certain that every possible value of the first key byte has appeared at least once. Assuming the keys are randomly distributed in the template database, the expected total number of traces in which the first byte of the key is zero can be given by $n/256 = 1,000,000/256$ or approximately 4,000. Using this estimate we can be reasonably assured that the information needed to perform the profiling of the attack is present in the template database.

Knowing that the database contains the necessary information is useless unless we identify a section of the trace to attack. Our implementation uses the l^2 norm of a subset of the power trace and in order to perform any initial analysis it is necessary to determine a specific section of the trace to analyze. Through direct observation of several power traces it is relatively easy to discern each of the ten cycles of AES. Figure 9 shows a plot of 1000 power traces. Looking at these traces it is easy to see the recurring pattern that forms the separation of the ten rounds. Figure 10 shows the same power traces with the 10 rounds separated by orange lines. Since our DTA implementation focuses on the first round of AES, we needed to find a subset of the first round that contains a significant amount of variation. Figure 11 highlights the subset of the first round with the highest degree of variation. Since the plaintext is fixed, we conclude that the variation in the traces is highly influenced by the differences in the key values.

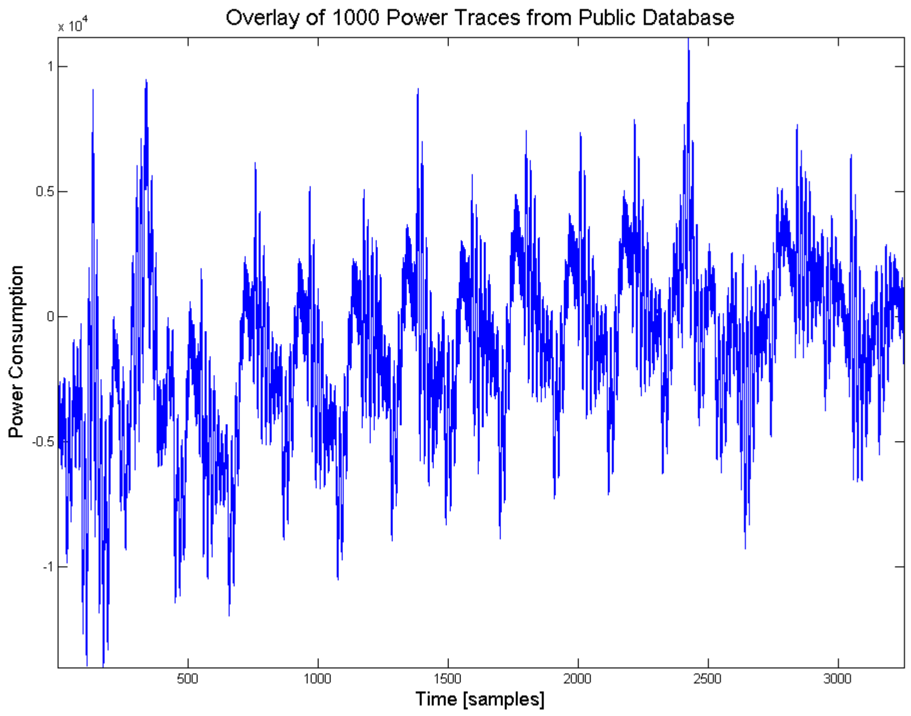


Figure 9: Overlay of 1000 power traces from unique AES encryptions.

Having identified the critical section of the power traces, some basic tests could then be run on the trace data to determine if our distinguished templates would be unique enough. If the distinguished templates are too similar, it would weakin our attack. First the l^2 norm of the critical section was calculated for all traces where the first byte of the plaintext was

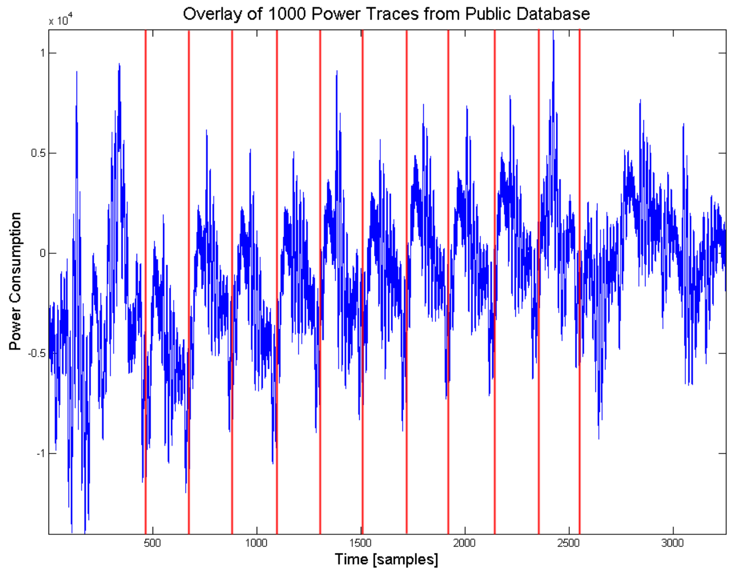


Figure 10: Overlay of 1000 power traces emphasizing the transitions between rounds.

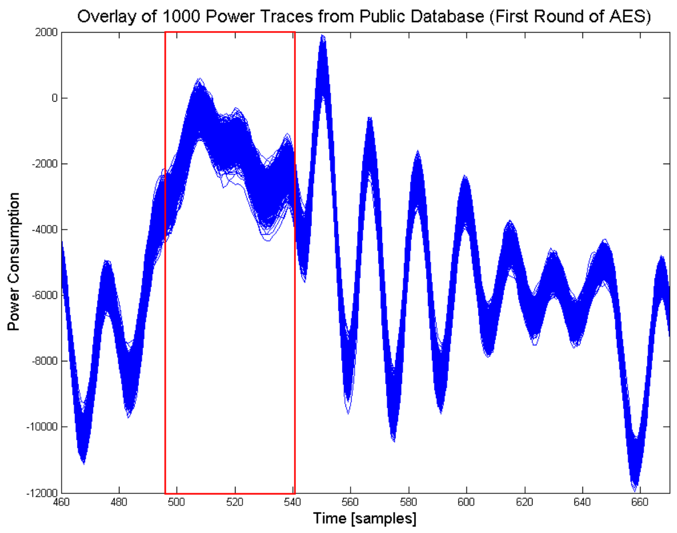


Figure 11: Zoomed in overlay of 1000 power traces emphasizing the first round of AES

zero. Then, taking these values as representing points along a line, the distance of each trace from every other trace was calculated. As can be seen in Figure 12, there are a relatively small number of traces with a significantly higher total distance than the rest of the traces, and these will end up becoming the distinguished templates. The existence of a relatively small number of relatively distant templates indicated a strong likelihood of the distinguished templates being unique enough to be effective in running the attack.

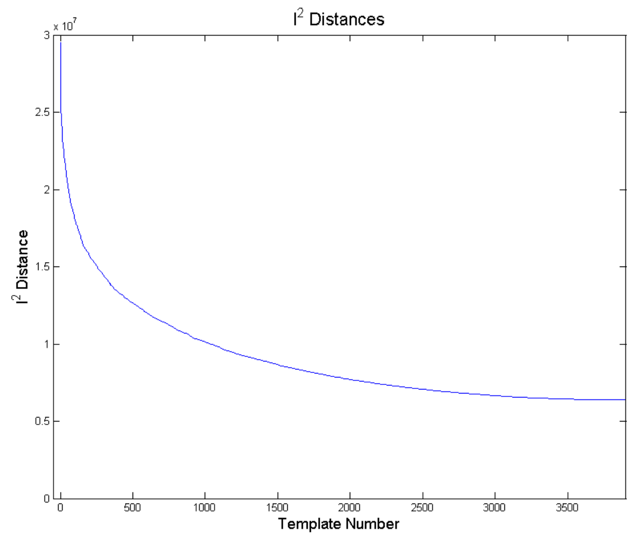


Figure 12: Total l^2 Distance of each Template to all other Templates

4.3 Results

With the attack implemented, the template and public databases provided by the DPA contest were used to test the effectiveness of the attack. To perform the profiling step, we used the template database as described above to acquire a set of templates from which we then extracted the distinguished templates. From the 256 templates, 25 distinguished templates were chosen. These became the templates used to perform the actual attack.

The attack itself was performed using the first 20,000 traces in the public database. These traces corresponded to a key with the first byte 0x00. This was chosen intentionally as it would simplify the process of calculating the key for a trace associated with a particular differential template. After matching the traces to the distinguished templates, we examined the groupings of traces. One observation was that the vast majority of traces matched against the same three distinguished templates. Specifically, out of the 25 distinguished templates, numbers 16, 20 and 25 matched against 21, 673 and 211 traces respectively. No other distinguished template matched against more than 5 traces. This observation was not very promising as we had originally expected to get relatively even distribution among the distinguished templates. Traces should ideally match against templates corresponding to the same intermediate value. Since these intermediate values are relatively evenly distributed, we expected that the number of traces matched to each distinguished template would be relatively evenly distributed. Therefore, this observation indicated that our method of matching traces to templates might need improvement.

In the Differential Template Attack described by Karakoyunlu and Sunar [5], the authors performed an intersection of the calculated keys associated with each distinguished template. Keys that were found associated with all of the distinguished templates were determined to be very likely candidates for the correct key value. In our attack, due to only a very small number of differential templates matching the majority of traces, the method used by Karakoyunlu and Sunar was not very effective at identifying candidates for the key byte. Therefore, we decided to consider the total number of occurrences of each calculated key value. For the first set of traces tested, it was found that the correct key always remained within the top 20 most common calculated keys after the first 7000 traces. If this were to hold true when testing other sets of traces, then this DTA could be used to narrow the possible key space to 20 possible values for each byte instead of 256. Unfortunately, this did not hold true after testing the other 31 sets of traces available in the public database. This led us to the conclusion that our current implementation of the DTA cannot effectively be used to significantly reduce the possible key space of a device under attack.

Due to the nature of our results, we believe that the failure of our DTA lies in how traces are matched to distinguished templates. We reached this conclusion because of the drastic variation in how attack traces matched against the distinguished templates. Although Karakoyunlu and Sunar found that the l^2 norm was effective at characterizing traces for their attack [5], it does not appear to be as effective when characterizing AES power traces. The l^2 norm condenses the sequence of points along a trace to a scalar value. In doing so, it ignores the order of the points and the output is only dependent upon the values of the points. An alternative matching method that does not ignore the order of the points may be more effective for our DTA.

5 Conclusion

Through our research, we were able to provide slight improvements to an existing DPA attack and implement a different variety of template attack on AES. With our relatively naive trace alignment process, we were able to acquire the key with up to 12% (around 2000) fewer traces. However, there were several cases with no improvement or slightly negative improvement, and this will need to be remedied before this improvement can be truly effective. The DTA implementation also shows promise but some challenges were encountered that prevent it from being immediately effective.

Since the original DTA was designed to attack both a different platform and a different type of cryptosystem, it is to be expected that the exact methods used may not prove particularly effective. We found this to be true with using the l^2 norm to match traces to templates. However, even with this limitation, the attack was able to mildly reduce the key space in a few cases. In the best case, it was able to reduce the key space of a single byte from 2^8 to less than 2^5 after processing 7000 traces. When considered in terms of the entire key, this could result in a possible 2^{36} reduction of the key space. Although this is not enough to make exhaustive search of AES tractable, it is a good start.

The existing DPA attack showed the effectiveness of DPA at compromising a cryptographic implementation. As such, it is important to consider DPA vulnerabilities when implementing AES. Our slight improvements through trace alignment show that it can be even more dangerous when an attacker is able to compensate for environmental factors. Our DTA implementation shows the potential of DTA in compromising the FPGA implementation of AES, but its performance at this time is far below that of our DPA implementation.

6 Future Work

The initial limited success of the Differential Template Attack leaves a number of avenues that could be taken to improve upon this work. First of all, replacing the l^2 norm with an alternative method of matching traces might improve the attack. There will be some computational consequences if the complexity in matching traces is increased, but template attacks by their nature often take longer to process individual traces than DPA. However, this is what enables them to acquire more useful information from each trace and reduce the number of traces necessary to perform an attack.

Additionally, designing and implementing this DTA would be greatly assisted by having full control over the inputs to the device when performing profiling. The provided Template database provides a very large number of traces, but it is still not as powerful as having complete control over the inputs. The DPA contest makes available the implementation used to acquire the traces in the databases, so a future project team could create an identical AES implementation on an FPGA and acquire new traces. This comes with its own set of challenges, but the complete control of the device could make it possible to determine what is required to make the Differential Template Attack successful.

References

[1] Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[2] Chari, S., Rao, J. R., and Rohatgi, P. Template attacks. In *CHES* (2002), B. S. K. Jr., Çetin Kaya Koç, and C. Paar, Eds., vol. 2523 of *Lecture Notes in Computer Science*, Springer, pp. 13–28.

[3] Guilley, S. Dpa contest v2 2009/2010, May 2010. `http://www.dpacontest.org`.

[4] Hnath, W., and Pettengill, J. *Differential Power Analysis Side-Channel Attacks in Cryptography.* Major Qualifying Project, Worcester Polytechnic Institute, April 2010.

[5] Karakoyunlu, D., and Sunar, B. Differential template attacks on puf enabled devices. In *Proceedings of IEEE Workshop on Information Security (WIFS)* (2010), pp. 1–6.

[6] Kocher, P. C., Jaffe, J., and Jun, B. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology - CRYPTO '99* (1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 388–397.

[7] Mangard, S., Oswald, E., and Popp, T. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, 1st ed. Springer, New York, 2007.

[8] Paar, C., and Pelzl, J. *Understanding cryptography : a textbook for students and practitioners*, 1st ed. Springer-Verlag, Heidelberg, 2010.

[9] Rechberger, C., and Oswald, E. Practical template attacks. In *WISA* (2004), C. H. Lim and M. Yung, Eds., vol. 3325 of *Lecture Notes in Computer Science*, Springer, pp. 440–456.

[10] Walle, M. *DPA contest v2 Evaluation Results.* white paper, August 2010. `http://www.dpacontest.org/v2/evals/2010-07-13_Thales_Walle/9T/results.pdf`.

[11] WEISSTEIN, E. W. L2-norm. From MathWorld - A Wolfram Web Resource. `http://mathworld.wolfram.com/L2-Norm.html`.