UNIVERSITY OF BERGEN
DEPARTMENT OF INFORMATICS

Side Channel Analysis on Bitsliced AES

Author: Stian Johannesen Husum
Supervisors: Martijn Stam and Øyvind Ytrehus

UNIVERSITY OF BERGEN
Faculty of Mathematics and Natural Sciences

May 30, 2022

c8a195b5 (main)

Abstract

The current power analysis attacks on bitsliced AES only take into account the power consumption of a single bit. We present a way of combining multiple Differential Power Analysis attacks using different power models to target multiple bits. The new attacks require less than half the amount of traces to reach a success rate of 99% compared to the current state of the art.

Acknowledgements

I would like to thank Martijn Stam and Nicolas Costes for their incredible guidance and support during the writing of this thesis. Thanks also to my friends, classmates, and everyone at Simula UiB for giving me some much-appreciated distractions, and giving me the best social environment for when I needed a break. And lastly, I would like to thank my family for supporting me in my studies, keeping me fed when I needed it the most, and always offering their help.

Contents

Glossary

Symbols

Notation	**Description**	Size
$\lvert\square\rvert$	Absolute value	
$\overline{\square}$	Arithmetic mean	
$\mathbf{c}$	Vector of scores for key guesses	K
$\mathbf{D}$	Matrix containing the data corresponding to traces with parallel encryptions	$P \times N$
$\mathbf{d}$	Vector containing the data corresponding to the traces	N
$\mathbf{g}$	Vector of key guesses sorted by score	K
k^*	Value of the correct key	
K	The number possible values for the part of the key that the intermediate value depends on	
N	The total number of traces	
P	The number of parallel encryptions	
$\mathbf{R}$	Matrix of scores as a result of a DPA attack	$K \times T$
$\mathbf{T}$	Matrix of power meassurement traces	$N \times T$
T	The number of power meassurement points in one trace	

Acronyms

Notation	**Description**
AES	Advanced Encryption Standard
CPA	Correlation Power Analysis
DES	Data Encryption Standard
DPA	Differential Power Analysis
LSB	Least Significant Bit
SCA	Side Channel Analysis
SNR	Signal-to-Noise Ratio
SPN	Substitution-Permutation Network
SR	Success Rate

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Introduction to Cryptography

The field of cryptography is interested in keeping communication secure. There are different aspects to security. One aspect is keeping information private such that an eavesdropper can not understand it. Another aspect is keeping the information intact to make sure nothing changes during transmission, either by tampering or an insecure channel. And thirdly, the aspect of verification of the identities of all parties. These aspects are known as confidentiality, integrity, and authentication, respectively, and are some of the most important aspects of cryptography.

We focus on the first of these aspects, confidentiality. To achieve confidentiality, messages are *encrypted* with a *key* such that they appear random to anyone other than the intended recipient. The intended recipient can then *decrypt* the message with a matching key to restore the original message. The original message is known as the *plaintext*, and the encrypted message is known as the *ciphertext*.

Encryption can be divided into two main categories: symmetric and asymmetric. For symmetric encryption, the keys used for encryption and decryption are equal. One very common symmetric algorithm is the Advanced Encryption Standard (AES), used, for example, as a part of TLS to secure the internet. Meanwhile, for asymmetric algorithms, encryption and decryption use different keys. Asymmetric encryption allows for public keys to be broadcast. Senders can encrypt their message using the recipient's private key, and then only the recipient can decrypt the message. Asymmetric algorithms are more versatile than symmetric algorithms, but asymmetric algorithms tend to be slower. Therefore, before sending data, an asymmetric algorithm is often used to exchange a shared key, and then that key can be used for symmetric encryption for the rest of the exchange.

1.2 Introduction to Side Channel Analysis

While cryptography is interested in constructing secure algorithms, *cryptanalysis* tries to break those algorithms. Breaking algorithms is a crucial part of security because it is the only way to know if something is secure, by discovering weaknesses before an adversary does. There are several ways to break an algorithm, and different attacks require different assumptions about what the attacker knows. For example, an attacker can try to recover the plaintext without knowing the key, or they can try to recover the key knowing both the plaintext and the ciphertext.

In Side Channel Analysis (SCA), it is often assumed that the attacker has physical access to the device. It incorporates additional information from the physical device into the attack. Such information is known as *side channels*. When encryption is run on a physical device, the behavior of that device depends on the secret key, and that behavior can leak information about that key. That behavior can be the time used, the power consumption, or even the electromagnetic radiation emitted by the device.

In cryptanalysis, the mathematical description of the algorithm is under attack, so the implementation is not relevant. But for SCA, both the physical characteristics of a device and the implementation become important. Different implementations of the same algorithm can have different leakage and require different attacks.

In this thesis, we will focus on SCA based on power consumption, called *power analysis*. For power analysis, the power consumption of a device is measured during encryption. Such a power measurement is known as a *trace*. Traces are often collected using fast oscilloscopes to measure the power several times per clock cycle. While the attacker might have physical access to the device, they might not have the key that is stored on it, so the goal is usually to recover the key. The power consumption can be dependent on the key in multiple ways. For instance, the algorithm could do different operations depending on the value of the key. Different operations often look distinct in the power measurements, and the key could therefore be recovered from looking at a single trace. Protecting from such attacks is done by avoiding any conditional branching that depends on any secret value.

The power consumption also depends on the value of the data being operated on, but the variation in power usage is much smaller. This leads to an attack introduced by Kocher, Jaffe, and Jun [KJJ99] called Differential Power Analysis (DPA). DPA attacks need a large number of traces that are analyzed using statistical functions, and the effectiveness of the attack can be measured in the amount of traces needed for a successful attack.

1.3 Bitslicing

Bitslicing is an implementation technique for encryption algorithms. Instead of storing the values in the state as whole bytes, the bits of the state are spread across multiple registers. By filling the rest of the registers with bits from other computations of the same cipher, multiple encryptions can be done in parallel.

Depending on the platform and algorithm, bitslicing can come at a great cost to speed. However, bitslicing can help with protecting against side-channel attacks. For instance, it protects against cache timing attacks, and it becomes easier to add countermeasures against DPA attacks.

1.4 Our Contribution

Since bitsliced implementations spread the secret data over multiple registers, the leakage is no longer at a single point in time. Therefore, the DPA attacks normally used for AES are no longer efficient. The goal of this thesis is to improve the current state-of-the-art attacks on bitsliced AES proposed by Balasch et al. [Bal+15]. Their attack is a DPA attack that only considers the power usage of a single bit. By considering more bits and exploiting multiple timestamps in the traces, we attempt to reduce the number of traces needed for a successful attack.

In Chapter 3 we go into detail about DPA attacks and introduce success rates as a metric for comparing the effectiveness of attacks.

Then, in Section 4.1, we establish a baseline for benchmarking by applying the attack by Balasch et al. [Bal+15] on a new platform and implementation. That implementation is the currently fastest bitsliced implementation by Adomnicai and Peyrin [AP20] running on a ChipWhisperer-Lite. We then compare the success rates for DPA attacks targeting all bits separately. Each of these attacks gives us a score for the possible keys. In Section 4.2 we compare multiple methods for combining these scores to increase the efficiency of the attack. By combining the scores with either sum or product we get a new attack requiring less than half the amount of traces as the original attack to reach a success rate of 99%.

Lastly, as bitsliced implementations are parallelized, in Section 4.3 we compared the success rates of having the parallel plaintexts equal or different. We discovered that the success rates are almost the same, with equal plaintexts being slightly better in some cases.

Chapter 2
Advanced Encryption Standard

In the late 1990s, the American National Institute of Standards and Technology (NIST) realized that their previous standard for encryption, Data Encryption Standard (DES), was getting to the end of its lifetime. The key size used in DES had become too small to be secure against modern computers, and in 1997 the first exhaustive brute-force attack on DES was achieved [Smi21]. To encourage more trust in a new standard, NIST decided to hold a competition to choose a replacement. This competition consisted of the international cryptographic community both submitting proposals for new algorithms and scrutinizing the algorithms. Efficiency, security, and flexibility were all high priorities for the algorithms. There were 15 valid submissions to the competition, which over several rounds were reduced down to 5 finalists [Smi21]. In October 2000, NIST announced the winner to be the Rijndael algorithm [Bul00]. The next year, in 2001, the standard was completed, and Rijndael was accepted as FIPS 197, The Advanced Encryption Standard (AES) [NIST01]. AES has since been accepted as a standard in the International Organization for Standardization. It is included in the Transport Layer Security (TLS) protocol used to secure the Internet, as well as in Wi-Fi to secure wireless networks [Smi21].

Advanced Encryption Standard (AES) AES is a symmetric block cipher and is a subset of the original Rijndael cipher. The full details of the cipher are in *Advanced Encryption Standard (AES)* [NIST01], but the following is a brief overview.

A block cipher works on a fixed-size input, referred to as a block. The Rijndael cipher allowed for a range of key- and blocksizes, however, AES only allows a blocksize of 128-bits and keys of 128-bits, 192-bits, and 256-bits. As a convention, AES-128 means AES using a 128-bit key. The same applies to 192-bits and 256-bits. This thesis will focus on AES-128. Therefore, if the size of the key is omitted, assume it to be 128-bits.

The AES state is structured as a 4×4 array of bytes. The cipher is constructed as a Substitution-Permutation Network (SPN). An SPN cipher consists of several rounds. Each round consists of a substitution of bytes, the addition of a round key, and a linear layer permuting the state.

In AES, the number of rounds depends on the size of the key. There are either 10, 12, or 14 rounds, with more rounds for larger keys. A key scheduler is used to expand the primary key into round keys for each of the rounds.

The first step of the cipher is adding the first round key to the state using bitwise xor. It is worth noting that the main key is used directly here, i.e., for a 128-bit key, the first round key is equal to the main key. Then follows $(n-1)$ rounds of SubBytes, ShiftRows, MixColumns, and AddRoundKey, where n is the number of rounds. The last round skips the MixColumns operation. It only contains SubBytes, ShiftRows, and the addition of the key.

The substitution layer, SubBytes, applies a function on each byte of the state. This function is referred to as the S-Box. It has *a byte*, eight bits, as input and output. The S-box is invertible, as the inverse is used for decryption. In addition, S-Boxes should be as non-linear as possible as it is the only part of the cipher that is not linear. S-boxes are commonly implemented using either lookup tables or logic circuits. Lookup tables are precomputed tables where the element at index i is the output of the S-box for input i. A logic circuit, on the other hand, consists of many logic gates, and performance is dependent on the number of gates needed to construct it. Much work has been put into finding circuits for AES S-boxes with fewer gates, and the current best is 113 gates[CMT20].

The linear layer consists of two operations, MixColumns and ShiftRows, and together they provide the permutation in the algorithm. MixColumns applies a linear transformation on each column of the state. The details of this operation will be omitted here but can be found in the NIST standard[NIST01]. ShiftRows does a cyclic right shift on each row of the state. The first row is not shifted, the second row is shifted once, the third row twice, and the fourth row is shifted three times.

The full structure of encryption with AES is described in Algorithm 2.1.

2.1 T-Table Implementation

T-tables are an implementation strategy for AES proposed in the original Rijndael proposal [DR02]. The T-table implementation precomputes four large lookup tables. These tables take into account the linear layer as well as the S-Box. Therefore the entire round can be reduced to lookups and XOR. Because the last round has a different linear layer from the rest, missing the MixColumns, this round must be handled separately. The T-table implementation is faster than the naive way of implementing AES on platforms that can fit the lookup tables. However, AES instruction sets on modern processors are even faster than using T-tables. Therefore T-tables are more popular on devices that do not have access to a dedicated AES instruction set.

Algorithm 2.1 AES-128

Require: 128-bit key k, plaintext as 4×4 matrix of bytes $\mathbf{S}$ numbered 1-16
Ensure: $\mathbf{S}$ contains the ciphertext

```
function AES($k, \mathbf{S}$)
    for $i \leftarrow 1$ to $10$ do
        $rk_i \leftarrow$ key schedule($k, i$)
    $\mathbf{S} \leftarrow k \oplus \mathbf{S}$
    for $i \leftarrow 1$ to $9$ do
        for $j \leftarrow 1$ to $16$ do
            $\mathbf{S}_j \leftarrow$ S-Box($\mathbf{S}_j$)
        $\mathbf{S} \leftarrow$ ShiftRows($\mathbf{S}$)
        $\mathbf{S} \leftarrow$ MixColumns($\mathbf{S}$)
        $\mathbf{S} \leftarrow rk_i \oplus S$
    for $j \leftarrow 1$ to $16$ do
        $\mathbf{S}_j \leftarrow$ S-Box($\mathbf{S}_j$)
    $\mathbf{S} \leftarrow$ ShiftRows($\mathbf{S}$)
    $\mathbf{S} \leftarrow rk_{10} \oplus \mathbf{S}$
    return $\mathbf{S}$
```

2.2 Bitslicing

Bitslicing is an implementation approach for block ciphers where the bits in the state are spread over multiple registers. It was first used for an efficient DES implementation by Biham [Bih97]. The simplest way to do this is by putting each bit of the state in a separate register. The rest of the register can then be filled with bits from different computations of the same cipher, allowing as many blocks enciphered in parallel as there are bits in the register.

Since S-Box computations can be represented as logic circuits, the computation for all the parallel blocks can be done at the same time using bitwise logic operators on the whole register.

Unfortunately, this approach is not very efficient for AES on devices with fewer than 128 registers. This is due to the overhead incurred by loading and storing the extra registers. The first bitsliced AES implementation by Matsui [Mat06] used this approach regardless. A better approach introduced by Könighofer [Kön08] uses the 4×4 structure of AES to its advantage and performs byte-level bitslicing. Instead of spreading the entire state over the registers, the bits for each byte are split instead. Multiple bytes of the same block are then packed into the same register. This allows for fewer blocks processed in parallel, as 16 bits are needed for each block instead of only one, but by using fewer registers, it becomes more efficient on platforms without many

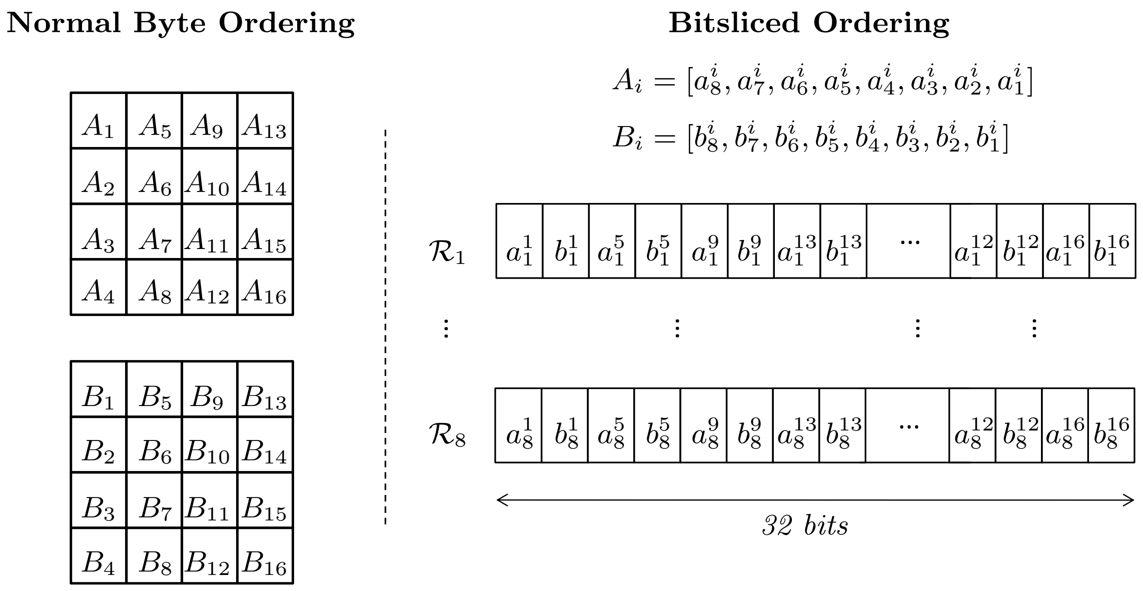
Normal Byte Ordering
Bitsliced Ordering
$A_i = [a_8^i, a_7^i, a_6^i, a_5^i, a_4^i, a_3^i, a_2^i, a_1^i]$
$B_i = [b_8^i, b_7^i, b_6^i, b_5^i, b_4^i, b_3^i, b_2^i, b_1^i]$
$\mathcal{R}_1$
$\mathcal{R}_8$
32 bits

Figure 2.1 / Layout of the registers in bitsliced AES [Bal+15].

registers. The layout for the registers using such byte-level bitslicing is in Figure 2.1.

An advantage of bitsliced implementations over T-tables is resistance against cache timing attacks due to not using lookup tables. In addition, due to the circuit structure of bitsliced implementations, adding Boolean masking to protect against side-channel attacks is straightforward.

2.3 Speed Comparison

As mentioned, T-table implementations are generally faster on platforms without dedicated hardware. One such platform, and the platform that will be the focus of this thesis, is the ChipWhisperer-Lite. The ChipWhisperer-Lite has an STM32F303 microcontroller with an Arm Cortex-M4 core. For Cortex-M, Schwabe and Stoffelen [SS17] introduced both a fast T-table and a fast bitsliced implementation. The bitsliced implementation was then further improved by Adomnicai and Peyrin [AP20].

The improvement made by Adomnicai and Peyrin [AP20] is using four specialized linear layers, one for every fourth round. By combining MixColumns and ShiftRows, the total number of operations needed is reduced, increasing performance. This implementation is referred to as the fixsliced implementation and will be the bitsliced implementation used in this thesis.

Choosing a bitsliced implementation comes at a cost to speed when T-table implementations are an option. To quantify the difference in performance between the fastest bitsliced and T-table implementations, we counted the number of cycles used by each of them to perform encryption on the ChipWhisperer-Lite. We ran the encryption several times, but the number of cycles did not change between different plaintexts and keys. The T-table implementation by Schwabe and Stoffelen [SS17] spends 672 cycles to encrypt one block. Meanwhile, the fixsliced implementation uses 2780 cycles to encrypt two blocks in parallel. As long as there are two plaintexts to encrypt, the fixsliced implementation uses 1390 cycles per block. This comparison does not take into account the key scheduler. In conclusion, the fixsliced implementation is about half as efficient as the T-table implementation.

Chapter 3
Side-Channel Analysis

When a cryptosystem is implemented on a physical device, it might leak different information than just the plaintext, and ciphertext. Such information might come, for example, from the power consumption of the system, or the electromagnetic radiation emitted by the system. This information is known as a side-channel. In Side Channel Analysis (SCA), this information is used to extract secrets and attack the cryptosystem.

SCA based on power consumption, called power analysis, collects power consumption values over time for the system while encrypting. Such measurements are called traces. Figure 3.1 shows the scaled power consumption during the first one and a half rounds of AES using a naive lookup table implementation. This trace was collected on a ChipWhisperer-Lite. The ChipWhisperer-Lite is a board containing an STM32F303 target that runs the encryption and an analog-to-digital converter to measure the voltage over a shunt resistor that is in line with the power of the target. The power consumption changes depending on what operation is being performed. By recognizing repeated patterns, an attacker can make assumptions about what operation is being performed. The trace shows a repeating pattern starting with a large negative peak, and each of these patterns is one round of AES. Power Analysis can also be used when a side-channel behaves similar to power consumption, for instance for measurements of electromagnetic radiation.

Power consumption is affected by the operation being performed. The simplest form of power analysis looks at patterns in a single trace that can reveal what operation is being performed. An example of this is when the square and multiply algorithm is used in encryption, like in the asymmetric encryption RSA. Square and multiply is used to compute exponentiation. It goes through each of the bits in the exponent, if the bit is zero only squaring is done, but if the bit is one multiplication is also done. The rounds with squaring will look different on the trace than the rounds with both squaring and multiplying, and therefore an attacker can read the exponent from the trace. For RSA this exponent is the secret key. To protect against these types of attacks, the choice of operation performed should not depend on the data.

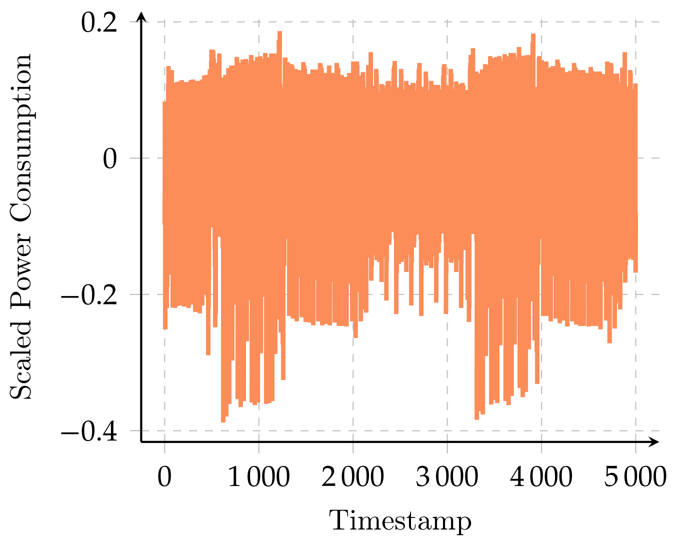

Figure 3.1 / Trace from naive AES implementation.

The power consumption also depends on the value of the data of the computation. In a CMOS circuit, flipping a bit requires more power than keeping the bit the same. For example, going from 1 to 0 requires more power than going from 1 to 1. Thus the power consumption when writing to a register depends on the number of bits that are different between the data already stored in the register and the new data. This difference is known as the Hamming distance. $\mathrm{HD}(a, b)$ denotes the number of bits that are different between a and b. A similar concept to Hamming distance is the Hamming weight, which describes the number of set bits. $\mathrm{HW}(a)$ denotes the number of one bits in a. Hamming distance and Hamming weight are related as follows: $\mathrm{HD}(a, b) = \mathrm{HW}(a \oplus b)$ where $\oplus$ is bitwise XOR.

3.1 Differential Power Analysis

Instead of just exploiting the information in one trace, Differential Power Analysis (DPA) uses many traces to exploit the data dependency described above. The traces are collected using different plaintexts encrypted under an unknown fixed key. DPA was originally proposed to attack DES by Kocher, Jaffe, and Jun [KJJ99]. Many improvements and variations have been made since then, and we will use the term DPA to refer to a wide range of attacks, including Correlation Power Analysis (CPA). The following approach is based on the generalized DPA attack described by Mangard, Oswald, and Popp [MOP08, pp. 119–123].

DPA starts with picking an intermediate value in the encryption that depends on both some known data and the unknown key. The known data is usually either a part of the plaintext or the ciphertext. Since DPA depends on trying all possible values for the unknown key, the intermediate value should only depend on a small part of the key, usually only one byte. The number of possible values for that part of the key is denoted K. If d is the known data and k is a part of the unknown key, then the intermediate value should be a function $f(d,k)$.

Next, several traces are collected with changing values for the known data. The number of traces is denoted N. The data is stored in a length N vector $\mathbf{d}$. We will denote the data corresponding to trace i by d_i. Each of the traces has a length of T, and the traces are stored in a $N \times T$ matrix $\mathbf{T}$. As notation, $\mathbf{t}_i$ refers to the i'th column of $\mathbf{T}$, and $\mathbf{t'}_i$ refers to row i of $\mathbf{T}$. Hence, $\mathbf{t'}_i$ refers to the trace for the i'th run of the encryption.

With the known data values collected, *hypothetical intermediate values* for each possible guess of k must be computed. For each value of k and each value of d, $v = f(d,k)$ is computed. Then these hypothetical intermediate values are mapped to *hypothetical power consumption values*. Let this mapping be the function 'model(v)', where v is a hypothetical intermediate value. This function should return a unitless estimate of the power consumption and is known as the *power model*.

Lastly, a *distinguisher* function is used to compare how closely the hypothetical power consumption under a given key matches the observed power consumption at a given timestamp. The distinguisher function takes in all hypothetical power consumptions for a key guess k, $\mathbf{h}_k$, and all power measurements at a timestamp i, $\mathbf{t}_i$, and returns a *score*. We denote this function as 'distinguisher$(\mathbf{h}_k, \mathbf{t}_i)$'. The scores for all key guesses and all timestamps are computed and collected in a $K \times T$ matrix $\mathbf{R}$. Higher scores in the matrix indicate that the key guess at that index is more likely to be the correct key.

DPA can take many forms, and the attacker has many choices in the parameters for the attack. The attacker has to choose the intermediate value, the power model, and the distinguisher. Algorithm 3.1 gives an overview of a DPA attack, and Algorithm 3.2 describes how to turn a matrix $\mathbf{R}$ into a length K vector $\mathbf{c}$ containing just the scores for each key guess.

Algorithm 3.1 Generalized DPA attack

Require: $N \times T$ matrix $\mathbf{T}$ of traces, length N vector $\mathbf{d}$ of data.
Ensure: $K \times T$ matrix $\mathbf{R}$ of scores.

```
1: function DPA($\mathbf{T}$, $\mathbf{d}$)
2:     for $n \leftarrow 1$ to $N$ do
3:         for $k \leftarrow 1$ to $K$ do
4:             $h_{k,n} \leftarrow \text{model}(f(d_n, k))$
5:     for $k \leftarrow 1$ to $K$ do
6:         for $i \leftarrow 1$ to $T$ do
7:             $r_{k,i} \leftarrow \text{distinguisher}(\mathbf{h}_k, \mathbf{t}_i)$
8:     return $\mathbf{R}$
```

Algorithm 3.2 Compute a vector of keyscores

Require: $K \times T$ matrix $\mathbf{R}$ of scores

Ensure: Length K vector $\mathbf{c}$ of one score for each keyguess.

```
1: function keyscores($\mathbf{R}$)
2:     for $k \leftarrow 1$ to $K$ do
3:         $c_k \leftarrow \max_{i \in \{1,..,T\}} r_{k,i}$
4:     return $\mathbf{c}$
```

3.1.1 Intermediate Values

The possible intermediate values depend on the encryption algorithm used, and what data is known. For AES the most commonly used intermediate value is the output of one of the first round S-boxes and it depends on knowing the plaintext. If d is one byte of the plaintext and k is one byte of the key, then this intermediate value is $f(d,k) = \text{S-Box}(d \oplus k)$. Prouff [Pro05] showed that the properties of S-boxes often help DPA attacks. This intermediate value is also the last value that only depends on one byte of the key because after MixColumns any intermediate value will depend on multiple keybytes.

If only the ciphertext is known, an alternative value is the input of one of the last round S-Box. Here the value depends on the last round key, but if the full last round key is found, the original main key can be recovered using the inverse of the key expansion.

3.1.2 Power Models

The choice of power model depends on the implementation and platform. Based on the observation about power consumption earlier, the Hamming distance or Hamming weight are common choices. The Hamming distance requires more knowledge of the system but often gives a better estimate than the Hamming weight. Both Hamming weight and Hamming distance are only efficient power models if the whole intermediate value is stored in the same register and operated on at the same time.

When neither Hamming weight nor Hamming distance is efficient, the simplest power model, and the one used in the original DPA paper[KJJ99], is the single-bit model. In this model, just one bit of the intermediate value is used. Commonly this is the Least Significant Bit (LSB). The choice of which bit to look at can change the effectiveness of the attack, as different bits can leak differently.

3.1.3 Correlation Power Analysis

Lastly is the choice of distinguisher, of which we will only focus on the Pearson correlation coefficient. DPA using Pearson correlation coefficient as distinguisher is called

Correlation Power Analysis (CPA). For two random variables X and Y, Pearson correlation coefficient is defined as:

$$\rho(X, Y) := \frac{\mathrm{Cov}(X, Y)}{\sqrt{\mathrm{Var}(X) \cdot \mathrm{Var}(Y)}} \tag{3.1}$$

However, this value needs to be estimated in the case of CPA. Given $\mathbf{x}$ and $\mathbf{y}$ as vectors of measurements, using sample variance and sample covariance, ρ is estimated as:

$$\hat{\rho}(\mathbf{x}, \mathbf{y}) := \frac{\sum_{n=1}^{N}(x_n - \bar{\mathbf{x}}) \cdot (y_n - \bar{\mathbf{y}})}{\sqrt{\sum_{n=1}^{N}(x_n - \bar{\mathbf{x}})^2 \cdot \sum_{n=1}^{N}(y_n - \bar{\mathbf{y}})^2}} \tag{3.2}$$

Let $\mathbf{t}_i$ and $\mathbf{h}_k$ be length N vectors. Here, $\mathbf{t}_i$ corresponds to the power value of all traces at timestamp i and $\mathbf{h}_j$ corresponds to the hypothetical power value with keyguess j for those traces. Then Pearson correlation coefficient can be used as a distinguisher as $\hat{\rho}(\mathbf{t}_i, \mathbf{h}_j)$.

3.2 Measuring The Performance of Attacks

As seen in this chapter, there are many variations of attacks. There are even more different devices, algorithms, and implementations. To compare how these factors impact the efficiency of the attacks, some metrics are needed. The definitions here are based on the definitions used by Cagli [Cag18, pp. 31–32].

The most common metrics are the *success rate of order* n(SR_n) and the *average keyrank*.

Each of the attacks described in this chapter can be used to create a sorted vector of key guesses, described in Algorithm 3.3, where a key guess is closer to the start if it is a more likely key. Let this vector be $\mathbf{g}$, g_i is the i'th most likely key guess, and k^* is the correct key. The *rank* of the correct key is its index in this vector, i.e. if $g_i = k^*$ then the rank of k^* is i.

Algorithm 3.3 Compute Keyguesses

```
Require: Length $K$ vector $\mathbf{c}$ of scores.
Ensure: Length $K$ vector $\mathbf{g}$ of sorted keyguesses.

1: $\mathbf{g} \leftarrow \arg\mathrm{sort}_{k \in \{1,..,K\}} c_k$
2: return $\mathbf{g}$
```

Success Rate The Success Rate (SR) of order n, SR_n, is the likelihood of the correct key, k^*, having a rank less than or equal to n. For instance, the success rate of order 1 is the likelihood of the correct key being the first guess in $\mathbf{g}$. In other words, it is the likelihood of the correct key being the key with the highest score. The success rate of order 1 is also called the *first order success rate*. When the order of the success rate is not specified, it is assumed to be of order 1.

Average Key Rank The average key rank is the average rank of the correct key. That is, the average index of k^* in $\mathbf{g}$.

These values are usually estimated empirically by repeating the attack a large number of times. The SR_n is then the ratio between the attacks where the correct key is within n and all attacks. And the average key rank is the mean index of the correct key in all the attacks.

3.3 Signal-to-Noise Ratio for SCA

A Signal-to-Noise Ratio (SNR) plot can be used to estimate the amount of leakage and where the leakage occurs. The SNR is the ratio between the signal and the noise, first introduced in the context of SCA by Mangard [Man04], defined as:

$$\mathrm{SNR} = \frac{\mathrm{Var}(\mathrm{Signal})}{\mathrm{Var}(\mathrm{Noise})} \tag{3.3}$$

The signal is the part of the power consumption that is caused by the chosen intermediate value, and the noise is the power consumption caused by algorithmic and electric noise.

To estimate the SNR we use the approach described by Mangard, Oswald, and Popp [MOP08, pp. 73–75]. First, we need to collect several traces, along with the data needed for the intermediate value. If the intermediate value depends on the key, we also assume the key to be known. We also need to choose a power model. The traces are grouped by their expected power value, i.e. the output of the power model on the intermediate values.

For each group, the mean of the traces in that group is calculated. To compute the signal, we can replace every trace with the mean trace for its group. The noise is calculated by subtracting the group mean from every trace. Algorithm 3.4 gives a summary of how to compute the SNR for a given time. By repeating this process for all timestamps, we can get a vector of SNR values. We can use that vector to locate where possible leakage occurs by looking at the highest values.

Algorithm 3.4 Compute an estimated SNR

```
Require: Length  vector $\mathbf{t}_j$ of traces at time $j$.
Ensure: Estimated SNR at time $j$.

1: for $i \leftarrow$ possible expected power values do
2:     $\mathbf{m}_i \leftarrow$ Values of $\mathbf{t}_j$ where the expected power value for that trace is $i$
3: for $n \leftarrow 1$ to $N$ do
4:     $i \leftarrow$ The expected power value of trace $n$
5:     $\text{Signal}_n \leftarrow \overline{\mathbf{m}_i}$
6:     $\text{Noise}_n \leftarrow t_{j,n} - \overline{\mathbf{m}_i}$
7: return $\frac{\sum_{n=1}^{N}(\text{Signal}_n - \overline{\text{Signal}})^2}{\sum_{n=1}^{N}(\text{Noise}_n - \overline{\text{Noise}})^2}$
```

Chapter 4

Attacks on Bitsliced AES

The current state of the art for attacks on bitsliced AES is the attacks by Balasch et al. [Bal+15]. Their attack only exploits one moment in the traces, corresponding to a single bit. The goal of this chapter is to improve these attacks by combining leakage from multiple timestamps, and therefore multiple bits, to improve their attack.

4.1 Correlation Power Analysis

Balasch et al. [Bal+15] uses CPA for their attack, therefore we will use it as the basis for our attacks and we chose to focus on CPA for this thesis.

As described in Section 3.1.1, the chosen intermediate value for AES is usually the output of one of the first round S-boxes and that will be the intermediate value we use. CPA using this intermediate value is described in Algorithm 4.1. In AES implementations using lookup tables, the entire intermediate value is stored in a single register. Therefore, when the intermediate value leaks, the leakage depends on all bits of the intermediate value. In this case, we can use Hamming weight as the power model as described in Section 3.1.2.

For bitsliced implementations, the intermediate value is no longer stored in its entirety in a single register. Instead, each bit of the intermediate value is stored in a separate register. Therefore, the Hamming weight model is no longer effective and we fall back to the single-bit model described in Section 3.1.2.

In addition, bitsliced implementations are inherently parallelized. If only the intermediate value corresponding to one of the plaintexts is used, and the plaintexts are different, the other plaintexts will only add algorithmic noise and make the attack less efficient. But, for each of the plaintexts, the same bit of the intermediate value is stored in the same register. Therefore, we can exploit the leakage from multiple plaintexts by taking the Hamming weight between bits from different intermediate values. Figure 4.1 shows an example of how intermediate values can be stored in

Algorithm 4.1 CPA on AES

```
Require: $N \times T$ matrix $\mathbf{T}$ of traces, length $N$ vector $\mathbf{d}$ of plaintexts, target S-box $b$, the
    power model used.
Ensure: $256 \times T$ matrix $\mathbf{R}$ of correlations.

1: function CPA($\mathbf{T}$, $\mathbf{d}$, $b$, model)
2:     for $n \leftarrow 1$ to $N$ do
3:         for $k \leftarrow 0$ to $255$ do
4:             $p_n \leftarrow$ choose byte $b$ of $d_n$
5:             $h_{n,k} \leftarrow$ model(S-Box($p_n \oplus k$))
6:     for $k \leftarrow 0$ to $255$ do
7:         for $i \leftarrow 1$ to $T$ do
8:             $r_{k,i} \leftarrow |\hat{\rho}(\mathbf{h}_k, \mathbf{t}_i)|$
9:     return $\mathbf{R}$
```

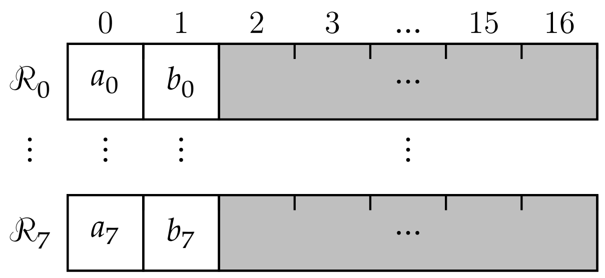

Figure 4.1 / Layout of intermediate values in registers for bitsliced AES with two parallel encryptions.

different registers for a bitsliced implementation with two parallel encryptions. Here a_i is the i'th bit of the intermediate value corresponding to the first plaintext, and b_i is the i'th bit of the intermediate value corresponding to the second plaintext. Then, a power model to exploit the i'th bit from each intermediate value is the Hamming weight of a_i and b_i.

Algorithm 4.2 describes an algorithm for an attack exploiting the intermediate value of multiple plaintexts, where P is the number of plaintexts encrypted in parallel. Balasch et al. [Bal+15] uses the Hamming weight of the least significant bits of two plaintexts as their power model.

The implementation we will use for our measurements is by Adomnicai and Peyrin [AP20]. It expects two plaintexts in parallel, therefore we will use the Hamming weight of two bits, one from each intermediate value, as our power model. In addition, for the following experiments, the two plaintexts will be different. There are multiple candidates for which bit to pick for the model. To compare the leakage of each of the bits, we can use each of them as the power model for an SNR plot. Figure 4.2 shows an

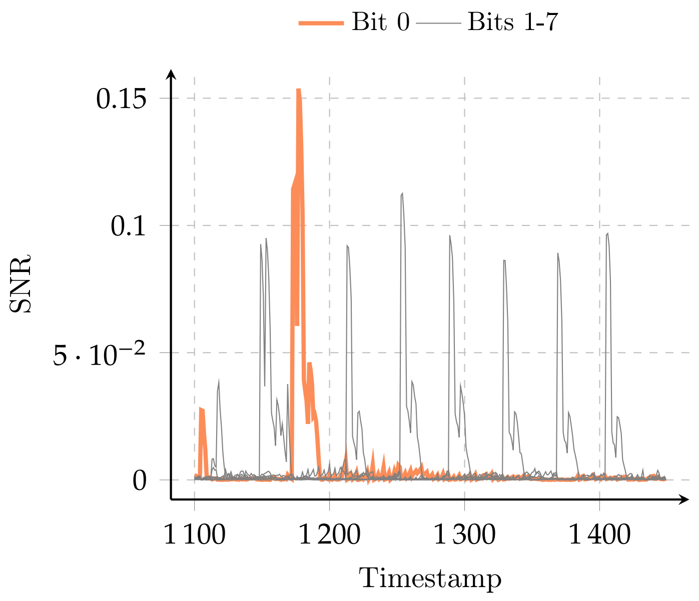

Figure 4.2 / SNR plot for each bit of intermediate value during the first S-box computation.

SNR plot with each potential power model. Each of the larger peaks corresponds to each of the power models. It clearly shows why the Hamming weight model for the full intermediate value is bad, as each of the bits is operated on separately and sequentially. Since the height of each peak is different, different bits also have a varying amount of leakage, with bit 0, the LSB, having the highest SNR for this implementation.

As the different bits leak differently, we can run an attack with each different bit as the power model to compare how they perform. We will use Algorithm 4.2 for these attacks, with $P = 2$. The details for the experiments are described in Section 4.5. Figure 4.3 shows the success rate for each different model. It clearly shows that the choice of bit can greatly impact the effectiveness of the attack. The least and most significant bits both have a high success rate, making the bit chosen by [Bal+15] a good choice also for this implementation.

4.2 Combining Scores

To exploit the information in multiple bits we can run an attack for each bit and combine the scores. For each attack we get a matrix of correlations, that matrix can be turned into a vector of scores for every key guess with Algorithm 3.2. Then we can use a combining function to combine the scores for each key guess. Such an attack is described in Algorithm 4.3.

There are a few options for combining functions, but our first attempt is just taking the maximum of every score. This is an improvement on just looking at one bit as

Algorithm 4.2 CPA on bitsliced AES implementations with parallel encryption

Require: $N \times T$ matrix $\mathbf{T}$ of traces, $P \times N$ matrix $\mathbf{D}$ of plaintexts, target S-box b, power model.

Ensure: $256 \times T$ matrix $\mathbf{R}$ of correlations.

```
1:  function CPA Bitsliced($\mathbf{T}$, $\mathbf{D}$, $b$, model)
2:      for $n \leftarrow 1$ to $N$ do
3:          for $k \leftarrow 0$ to $255$ do
4:              $h_{n,k} \leftarrow 0$
5:              for $i \leftarrow 1$ to $P$ do
6:                  $p \leftarrow$ choose byte $b$ of $d_{i,n}$
7:                  $h_{n,k} \leftarrow h_{n,k} + \text{model}(\text{S-Box}(p \oplus k))$
8:      for $k \leftarrow 0$ to $255$ do
9:          for $i \leftarrow 1$ to $T$ do
10:             $r_{k,i} \leftarrow |\hat{\rho}(\mathbf{h}_k, \mathbf{t}_i)|$
11:     return $\mathbf{R}$
```

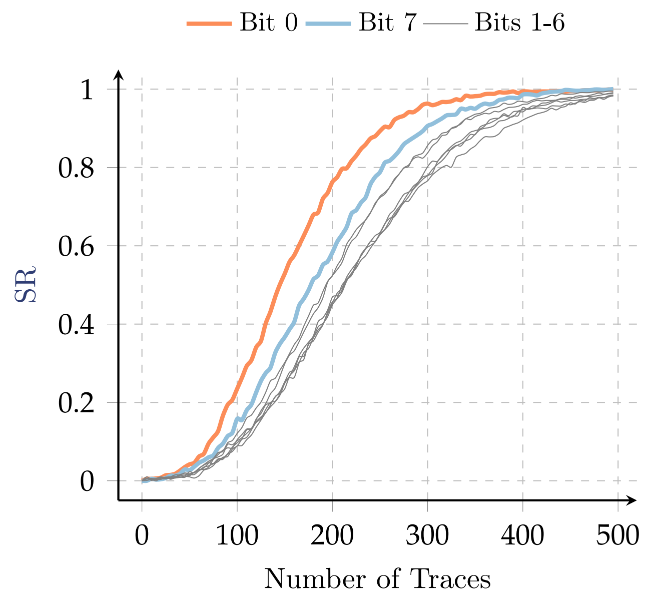

Figure 4.3 / Success rates for attacks on each bit.

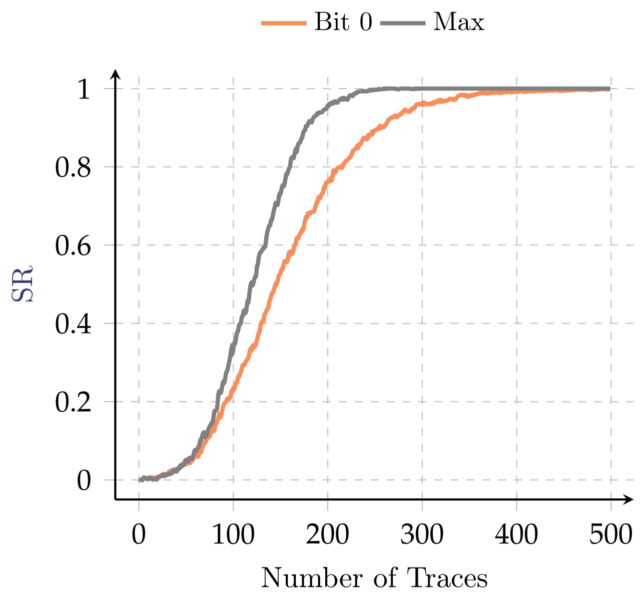

Figure 4.4 / Success rates for attacks using maximum correlation.

shown in Figure 4.4.

Algorithm 4.3 CPA attack with score combiner

```
Require: $N \times T$ matrix $\mathbf{T}$ of traces, $P \times N$ vector $\mathbf{D}$ of plaintexts, target S-box $b$, score
    combiner.
Ensure: Length 256 vector $\mathbf{c}$ of scores.

1: for $i \leftarrow 0$ to 7 do
2:     $\mathbf{c}_i \leftarrow$ keyscores(CPA Bitsliced($\mathbf{T}$, $\mathbf{D}$, $b$, $i$'th bit power model))
3: for $k \leftarrow 0$ to 255 do
4:     $c'_k \leftarrow$ combiner $\left(c_{i,k} | i \in \{0, \ldots, 7\}\right)$
   return $\mathbf{c}'$
```

To improve the attack further we need to retain the information from all the bits, and not just pick the best score. For this, we chose to use sum and product as combiners. Elaabid et al. [Ela+11] chose to use product to combine correlation. In their paper, they try to combine the correlation at several timestamps on a non-bitsliced implementation, and unlike our attack, they do not use different power models.

Figure 4.5 shows that the success rate for both the product and sum are more efficient than both attacking a single bit and taking the maximum. Sum and product are very close in performance. This similarity is noted by Yang et al. [Yan+17] in a related setting, where the correlation is combined across different side channels, instead of different models on the same side channel. The reason for this close similarity has not

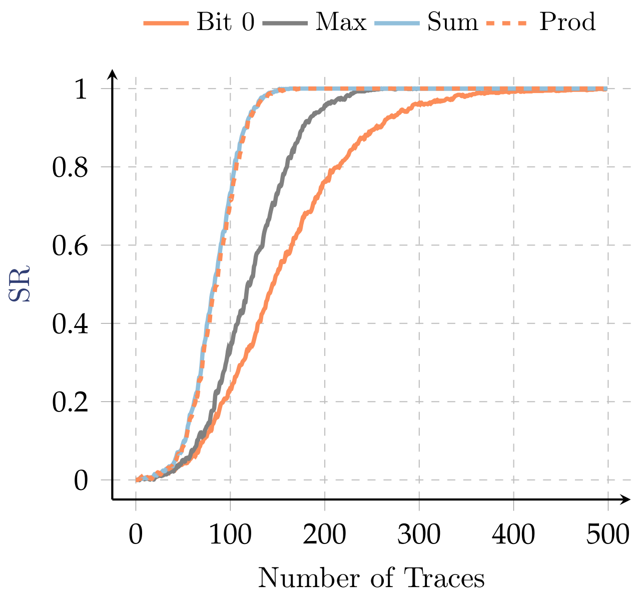

Figure 4.5 / Success rates for attacks using sum or product.

been explored further in this thesis.

4.3 Choice of Plaintexts

So far we have only looked at the case where the two plaintexts in the plaintext pair are different but does the efficiency of the attack change if they are equal?

In many cases, if the attacker does not control the plaintexts, we expect the two plaintexts to be different. But, in some cases, they could be similar enough for most of the bytes to be equal. AES can be used in a mode of operation known as counter mode[Dw001]. In counter mode, AES is used to generate a keystream, this keystream is then XORed with the plaintext to create the ciphertext. To generate the keystream, a random initial value is encrypted for the first block, then the initial value is incremented for each block. Since only the last byte changes between most increments, the other bytes stay the same. Another instance where the two plaintexts could be equal is for added redundancy and to protect against fault attacks.

Figure 4.6 compares the success rates of an attack using just LSB(Algorithm 4.2) and an attack using sum as a combiner (Algorithm 4.3) using either traces where the plaintexts are equal or traces where the plaintexts are different. It shows that the success rates are almost the same, with equal plaintexts being slightly better in some cases. If the attacker does not control the plaintexts they are not at any significant disadvantage.

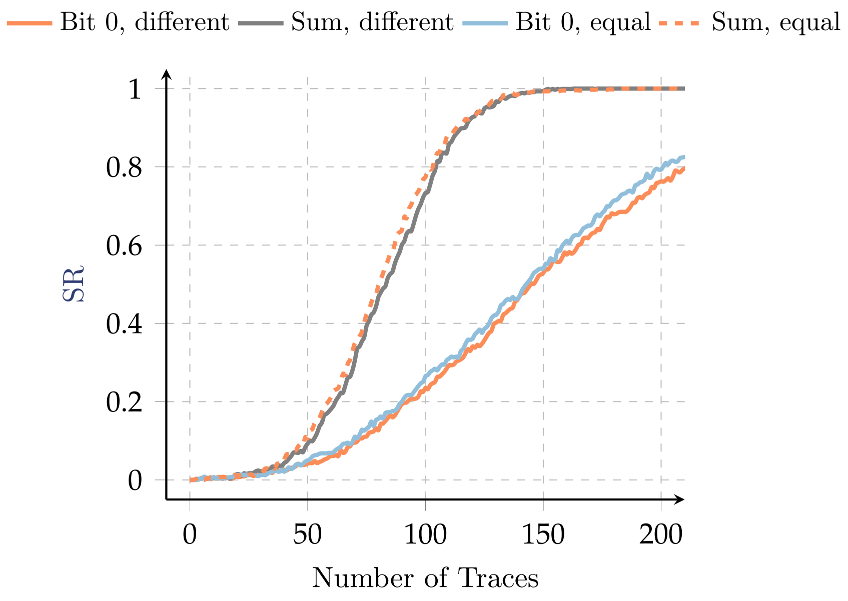

Figure 4.6 / Success rates for attacks using either traces with equal plaintexts or different plaintexts.

Table 4.1 / The number of traces needed to have $SR_0 \geq 0.99$.

Trace method \ Attack	Sum	Product	Max	Bit 0
Plaintext pairs equal	144	146	228	334
Plaintext pairs differ	143	145	231	364

4.4 Overall Results

We have presented several new attacks and experimentally evaluated their success rate using both equal plaintext pairs and plaintext pairs with different plaintexts. The bit 0 attack with different plaintexts is equivalent to the attack used by Balasch et al. [Bal+15] in their paper and is the current state of the art. To compare the success rates more directly, Table 4.1 compares the traces needed to reach a success rate of 99% for the different attacks. It shows that the sum and product attack presented in this thesis requires less than half the amount of traces compared to the current state of the art. On different plaintexts, the current state of the art requires 364 traces, and our best attack requires only 143 traces.

4.5 Experimental Setup

To get the success rates we used in this thesis, we collected power traces on a ChipWhisperer-Lite using the implementation by Adomnicai and Peyrin [AP20]. The traces were cut down to the area around first-round SubBytes. Each type of attack was performed on up to 500 traces and 1000 times, requiring 500000 traces.

Since algorithms in this thesis only describe how to perform *one* attack, speed improvements had to be made to make the computation of all the attacks feasible. These improvements were based on the algorithms by Bottinelli and Bos [BB17]. They describe different algorithms for computing CPA that make tradeoffs between speed and memory. We opted to aim for the highest speed, but at the cost of high memory usage. They describe that $\hat{\rho}(\mathbf{h}_k, \mathbf{t}_i)$, Equation 3.2, can be rewritten as:

$$\frac{Ns_5 - s_1 s_3}{\sqrt{(Ns_2 - s_1^2)(Ns_4 - s_3^2)}} \qquad [4.1]$$

Where N is the number of traces, $s_1 = \sum_{n=1}^{N} \mathbf{h}_{k,n}$, $s_2 = \sum_{n=1}^{N} \mathbf{h}_{k,n}^2$, $s_3 = \sum_{n=1}^{N} \mathbf{t}_{i,n}$, $s_4 = \sum_{n=1}^{N} \mathbf{t}_{i,n}^2$ and $s_5 = \sum_{n=1}^{N} \mathbf{h}_{k,n}\mathbf{t}_{i,n}$. The tradeoffs between speed and memory are which of these sums to precompute, and we chose to precompute all of them. We also wanted to compute the correlation for every added trace, not just the final correlation, i.e. we want the correlation for n traces, $n+1$ traces, $n+2$ traces, and so on, and other than the newly added trace, the set of traces stay the same. So, instead of just precomputing the final sums, we stored the cumulative sums as well.

The full code for both the trace collection and experiments can be found at github.com/Simula-UiB/bitsliced-aes-cpa.

Chapter 5

Conclusions and Future Work

In this thesis, we have presented several new attacks on bitsliced AES. The current state of the art only exploited a single bit of information, but we presented a way to use the information in multiple bits. These attacks are based on combining the scores from multiple CPA attacks, each attack targeting a different bit. The scores are combined using sum, product, or maximum. Only half as many traces were needed for the same performance with sum and product, compared to attacking only one bit.

There are potential improvements to be made to the combiner. For instance, we have not investigated how much each bit contributes to the leakage, and if a weighted sum/product would be more efficient. The combiner could also potentially be used for other attacks, such as for higher-order DPA to target masked implementations and for profiled attacks.

In addition to new attacks, we have also looked at how the choice of plaintext pairs affects the performance of the attacks. This investigation showed that collecting traces having the two plaintexts equal can be slightly more efficient, but the difference is minimal compared to differing plaintexts. We did not compare the performance of looking at only one plaintext and ignoring the other as we expect that to be significantly less efficient, however, it would be interesting to quantify how much the difference is.

Lastly, we made a speed comparison between a bitsliced and T-table implementation of AES, showing the bitsliced implementation is about half as slow. Comparing the side-channel security between those implementations would also be interesting.

Bibliography

[AP20] Alexandre Adomnicai and Thomas Peyrin. “Fixslicing AES-like Ciphers: New Bitsliced AES Speed Records on ARM-Cortex m and RISC-V.” In: IACR Transactions on Cryptographic Hardware and Embedded Systems 2021.1 (Dec. 2020), pp. 402–425 (cit. on pp. 3, 7, 17, 23).

[Bal+15] Josep Balasch et al. “DPA, Bitslicing and Masking at 1 GHz.” In: Cryptographic Hardware and Embedded Systems CHES 2015. Ed. by Tim Güneysu and Helena Handschuh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 599–619 (cit. on pp. 3, 7, 16–18, 22).

[BB17] Paul Bottinelli and Joppe W Bos. “Computational Aspects of Correlation Power Analysis.” In: Journal of Cryptographic Engineering 7.3 (2017), pp. 167–181 (cit. on p. 23).

[Bih97] Eli Biham. “A Fast New DES Implementation in Software.” In: Fast Software Encryption. Ed. by Eli Biham. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 260–272 (cit. on p. 6).

[Bul00] Philip Bulman. Commerce Department Announces Winner of Global Information Security Competition. NIST. Oct. 2, 2000. url: https://www.nist.gov/news-events/news/2000/10/commerce-department-announces-winner-global-information-security (visited on 05/04/2022) (cit. on p. 4).

[Cag18] Eleonora Cagli. “Feature Extraction for Side-Channel Attacks.” PhD thesis. Sorbonne Université, Dec. 5, 2018 (cit. on p. 13).

[CMT20] Cagdas Calik and Circuit Minimization Team. Circuit Minimization Work. 2020. url: http://cs-www.cs.yale.edu/homes/peralta/CircuitStuff/CMT.html (visited on 09/23/2021) (cit. on p. 5).

[DR02] Joan Daemen and Vincent Rijmen. The Design of Rijndael: AES the Advanced Encryption Standard. Springer-Verlag, 2002 (cit. on p. 5).

[Dwo01] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Special Publication (SP) 800-38A. National Institute of Standards and Technology, Dec. 1, 2001 (cit. on p. 21).

[Ela+11] M. Abdelaziz Elaabid et al. “Combined Side-Channel Attacks.” In: Information Security Applications. Ed. by Yongwha Chung and Moti Yung. Vol. 6513. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 175–190 (cit. on p. 20).

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. “Differential Power Analysis.” In: Advances in Cryptology CRYPTO 99. Ed. by Michael Wiener. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1999, pp. 388–397 (cit. on pp. 2, 10, 12).

[Kön08] Robert Könighofer. “A Fast and Cache-Timing Resistant Implementation of the AES.” In: Topics in Cryptology CT-RSA 2008. Ed. by Tal Malkin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 187–202 (cit. on p. 6).

[Man04] Stefan Mangard. “Hardware Countermeasures against DPA A Statistical Analysis of Their Effectiveness.” In: Topics in Cryptology CT-RSA 2004. Ed. by Tatsuaki Okamoto. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 222–235 (cit. on p. 14).

[Mat06] Mitsuru Matsui. “How Far Can We Go on the X64 Processors?” In: Fast Software Encryption. Ed. by Matthew Robshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 341–358 (cit. on p. 6).

[MOP08] S. Mangard, E. Oswald, and T. Popp. Power Analysis Attacks: Revealing the Secrets of Smart Cards. Advances in Information Security. Springer US, 2008 (cit. on pp. 10, 14).

[NIST01] Advanced Encryption Standard (AES). National Institute of Standards and Technology, Nov. 2001 (cit. on pp. 4, 5).

[Pro05] Emmanuel Prouff. “DPA Attacks and S-Boxes.” In: Fast Software Encryption. Ed. by Henri Gilbert and Helena Handschuh. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 424–441 (cit. on p. 12).

[Smi21] Miles Smid. “Development of the Advanced Encryption Standard.” In: 126 (Aug. 16, 2021) (cit. on p. 4).

[SS17] Peter Schwabe and Ko Stoffelen. “All the AES You Need on Cortex-M3 and M4.” In: Selected Areas in Cryptography SAC 2016. Ed. by Roberto Avanzi and Howard Heys. Cham: Springer International Publishing, 2017, pp. 180–194 (cit. on pp. 7, 8).

[Yan+17] Wei Yang et al. “Multi-Channel Fusion Attacks.” In: IEEE Transactions on Information Forensics and Security 12.8 (Aug. 2017), pp. 1757–1771 (cit. on p. 20).

Further Reading

The following references were used in this work but not cited in the text body; they are provided here as-is.

Wouter de Groot et al. “Bitsliced Masking and ARM: Friends or Foes?” In: Lightweight Cryptography for Security and Privacy. Ed. by Andrey Bogdanov. Cham: Springer International Publishing, 2017, pp. 91–109.

Emilia Käsper and Peter Schwabe. “Faster and Timing-Attack Resistant AES-GCM.” In: Cryptographic Hardware and Embedded Systems - CHES 2009. Ed. by Christophe Clavier and Kris Gaj. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–17.

Mitsuru Matsui and Junko Nakajima. “On the Power of Bitslice Implementation on Intel Core2 Processor.” In: Cryptographic Hardware and Embedded Systems - CHES 2007. Ed. by Pascal Paillier and Ingrid Verbauwhede. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 121–134.

Stefan Mangard, Elisabeth Oswald, and Francois-Xavier Standaert. One for All - All for One: Unifying Standard DPA Attacks. 449. 2009.

Luke Mather, Elisabeth Oswald, and Carolyn Whitnall. “Multi-Target DPA Attacks: Pushing DPA Beyond the Limits of a Desktop Computer.” In: Advances in Cryptology ASIACRYPT 2014. Ed. by Palash Sarkar and Tetsu Iwata. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2014, pp. 243–261.

Ambuj Sinha, Zhimin Chen, and Patrick Schaumont. “A Comprehensive Analysis of Performance and Side-Channel-Leakage of AES SBOX Implementations in Embedded Software.” In: Proceedings of the 5th Workshop on Embedded Systems Security. WESS ’10. New York, NY, USA: Association for Computing Machinery, 2010.