When One Bit Is (Almost) Enough: A Low Leakage Chosen Plaintext Side Channel Attack on AES

Antoine Wurcker

Serma Safety and Security
Pessac, France
a.wurcker@serma.com

Abstract. In order to thwart Side Channel Attacks (SCA) and keep secret safe, countermeasures such as masking can be implemented to hide information. However, when a defect occurs in software or in hardware, a portion of the information may still leak. If the leak is sufficient, the state-of-the-art techniques can still be performed up to a potential brute force search to cover a missing part. However, if only a short portion of the information leaks, the completion of the attack can be hard, and considered as failed.
In this paper, we describe a new chosen plaintext strategy on AES algorithm to recover the secret key even in very low leakage condition. As an example, but not limited to, we describe the methodology adapted to one of the worst case scenario: only one bit position leaks over the 128-bit output of *SubBytes* AES sub-operation along rounds. As a drawback, a heavier cost in terms of number of traces is required due to the chosen plaintext approach.

Keywords: AES · side-channel · SCA · Chosen plaintext · low leakage

1 Introduction

Advanced Encryption Standard (AES) is a symmetric cipher algorithm specified by United States of America National Institute of Standards and Technology in 2001[7]. Three verions are available depending on key size: AES-128, AES-192 and AES-256. For sake of clarity, the word AES will hereafter refer to AES-128 and the methodology will be described for this version of the algorithm. Adaptation to other sizes will be discussed in Section 5.

Side Channel Attacks (SCA) is a family of attacks that targets secret information hidden inside an electronic device. They exploit information leaking through physical measurements such as power consumption and electromagnetic emissions [4, 5, 3]. Statistical SCA, such as DPA and CPA [5, 1], target constant secret values through the leakage of intermediate value of a computation, when a combination between an unknown constant value and a known variable value occurs.

Chosen plaintext SCA consider an attacker that can control, at least partially, the input of the targeted algorithm. The choice of specific values or specific properties, such as fixed or variable, is required in order to obtain peculiar behaviors of intermediate values, allowing the attack to be performed. One drawback of this kind of attack is that numerous sets of traces must often be acquired. Indeed, each set uses a particular choice of message designed to target a specific part of the information, leading to having one set for each part of information to recover.

In order to protect from SCA, several methods to hide information were proposed [6, 2, 9], such as masking. This method consists in modifying the algorithm to be protected by adding and removing mask values during intermediate computation. These masks must be chosen randomly for each execution, and kept unpredictable by the attacker to ensure security. The attacker must then perform more complex attacks to bypass such protection [8].

In this paper, we consider a scenario where an attacker faces leakages that occur only on a reduced part of the intermediate values required for classical SCA to succeed. To our knowledge, classical SCA cannot be conducted in such conditions as soon as the missing information cannot be brute-forced in a reasonable effort. We then describe our new methodology that allows to exploit these few leakages to recover the complete secret AES key. This scenario of low available leakage can occur for various reasons, such as bad random generation, errors in implementation of countermeasures, hardware defects, ...

In order to show the potential of our attack method, we choose a scenario where only one bit out of the 128 bits of AES *SubBytes* output is leaking at each round. This choice was made to show the feasibility in a extremely difficult context for the attacker. However, the method described here might be adapted to any context where a wider leakage is available. Thus, the cost of our attack methodology might be reduced in a more favorable context for an attacker.

In this paper, notations and the leakage models considered are detailed in Section 2. Our main contribution is detailed in Section 3. Cost of attack considerations and optimizations are detailed in Section 4. Section 5 contains discussions about the presented methodology, to finally conclude in Section 6.

2 Notations

The notations used are as follows:

- $\oplus$ is the bitwise xor operation.
- $S()$ is the AES's byte look up table called S-box.
- $S^{-1}()$ is the inverse of AES's byte look up table.
- $a{\cdot}b$ (with a and b being any byte values) is the AES's multiplication done modulo an irreductible polynomial.
- p is the plaintext, k the key and c is the ciphertext of an AES execution.
- $p_{i,\forall i\in[0,\ldots,15]}$ are the 16 individual bytes of plaintext.
- k_0 is the sub-key used in the initial *AddRoundKey*.

- $k_{j,\forall j\in[1,\ldots,10]}$ is the sub-key used in the *AddRoundKey* of the j^{th} round.
- $k_{j,i,\forall j\in[0,\ldots,10],\forall i\in[0,\ldots,15]}$ are the 11×16 individual bytes of round keys.
- $x_{i,\forall i\in[0,\ldots,15]}$ are the 16 individual bytes of output of first round *SubBytes*.
- $y_{i,\forall i\in[0,\ldots,15]}$ are the 16 individual bytes of output of second round *SubBytes*.
- $z_{i,\forall i\in[0,\ldots,15]}$ are the 16 individual bytes of output of third round *SubBytes*.

Referring to all bytes described above:

- Lowercase/Red (e.g. p_5) represents an unknown variable.
- Underlined Lowercase/Green (e.g. $\underline{p_5}$) represents a known variable.
- Uppercase/Fuchsia (e.g. P_5) represents an unknown constant.
- Underlined Uppercase/Blue (e.g. $\underline{P_5}$) represents a known constant.

One AES round is composed of:

- *AddRoundKey*: The current 128-bit state is xored with a 128-bit sub-key.
- *SubBytes*: Each of the 16 bytes of current state is replaced by its value through a look-up table (S-box).
- *ShiftRows*: 12 bytes over the 16 of current state change of position.
- *MixColumns*: The 4 vectors of 4 bytes of current state are multiplied by a constant matrix.

As depicted on Figure 1, the AES is composed of 10 rounds with a prior *AddRoundKey* operation at the beginning and the last round does not contain *MixColumns* operation.

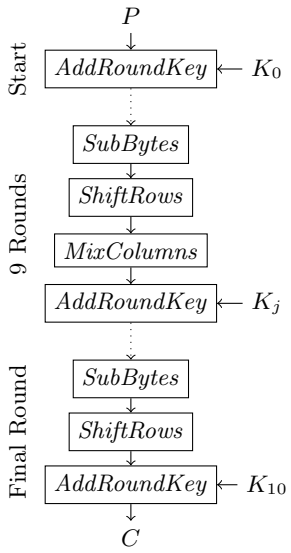


Fig. 1. AES-128 algorithm main steps.

2.1 Leakage Model

As stated in introduction, our restrictive leakage model consider only one bit over the 128-bit state as output of *SubBytes* operation is leaking for (at least) the three first rounds.

Despite the cost in terms of number of traces that can be higher due to the exploitation of a lower quantity of information, an attacker using the leakage of only one bit of output of S-box look-up eventually recovers the same information as an attacker that has access to a wider leakage of the byte where the bit lies. This is due to the non-linearity of the look-up table and the fact that each bit of output is related to every 8 input bits.

For sake of clarity, in the following, we will use the term leakage of a byte without recalling that it can be only a single bit that leaks or more.

Also for sake of clarity, in the following, we consider cases where the same bit position is leaking over the rounds but this is not required in practice. This is discussed in Section 5.3.

2.2 Attack Types

In the following of this paper two types of attacks will be considered. For sake of clarity we define them as A-Type and B-Type attacks.

A-Type attack are the ones that target only one secret byte (2^8 candidates) through a leakage at the output of an S-box. For example the leakage of $S(\underline{v} \oplus C)$ with $\underline{v}$ a known variable byte and C the targeted unknown constant byte.

B-Type attack are the ones that target two secret bytes simultaneously (2^{16} candidates), each one being involved in computation of two nested S-boxes. The first secret byte being involved inside the computation of a S-box whose result is involved in another S-box with the second secret byte. For example the leakage of $S(S(\underline{v} \oplus C_1) \oplus C_2)$ with $\underline{v}$ a known variable byte and C_1 and C_2 the two targeted unknown constant bytes.

Ratio in Attack Costs. Despite both attacks target the output of an S-box, B-Type attacks may be more costly in term of number of traces as the good candidate must emerge from 256 times more candidates. The cost ratio of the two attack types seems to highly depend on the conditions of measurement. Figure 2 shows the results of one simulation, chosen to be a simple illustration and not a rigorous quantification. It shows how the good candidate emerged from respectively 256 (A-Type) and from 65,536 candidates (B-Type). The x-axis is the number of traces used, the y-axis is the CPA score, the red line is the evolution of the good candidate progressively emerging from the surface containing the 255 wrong candidates in yellow and respectively 65,535 wrong candidates in blue.



Fig. 2. Illustration of emergence of the correct candidate from surface containing the wrong ones during a simulated attack of a byte leakage.

The simulation conditions are as follows: 3,000 traces of one point whose value corresponds to a normal noise added with the Hamming weight of $S(S(v \oplus C_1) \oplus C_2)$ with v a variable byte and C_1 and C_2 constant bytes. For the 256-candidate configuration the byte C_2 is known and C_1 is searched when the whole pair (C_1, C_2) is searched for the 65,536-candidate configuration. This simulation shows an example where, in same configuration, it takes nearly twice the number of traces for the correct candidate to emerge in a Type-B attack ($\sim 1,400$ traces) than in Type-A attack (~ 700 traces).

2.3 Attack Metric

DPA-like attacks have shown how to recover a secret constant value using variability of known values. For sake of clarity, we assume the ability of the attacker to recover a secret that will meet this condition. As all leakages targeted in this paper correspond to the same step in a round of the algorithm (i.e. output of *SubBytes* sub-operation) we assume that attacks of same type will have a similar cost. We also assume that each set of traces contains enough traces to recover the secret with regards to the conditions of leakage. If these assumptions are not verified, the attack can still be performed but the total cost must be computed accordingly.

Therefore, in this paper, the metric of the attack will not be the number of traces as it highly depends on the physical conditions of attack. Instead, the number of set of traces of each type (A-Type or B-Type) required to perform

an attack is given as the metric. Our objective is to reduce the number of sets required to perform an attack.

3 Our Contribution

This section describes the general methodology of our attack. It is meant to be as generic as possible to be adapted to the leakage location and the AES size. Therefore, the method is not the most efficient due to its genericness. Section 4 then proposes methods that can be applied individually to each configuration, reducing the cost of the attack.

The process of the attack differs slightly depending on the position of the leaking byte. If the leaking byte position is in the set $\{0, 4, 8, 12\}$ then the attack process described in Section 3.1 must be followed. For the 12 other leaking byte positions, Section 3.2 describes the variations needed at some steps to adapt the methodology. As a reminder, Section 5.3 details how to adapt our methodology when leaking byte position is not the same on each round.

3.1 Global Methodology

We consider here a leakage as described in Section 2.1 that appears for byte $i = 0$ as an example. This means that physical measurement values are related to byte values (x_0, y_0, z_0) but not to other bytes $(x_i, y_i, z_i)_{,\forall i \in [1,\dots,15]}$.

Step 1: First Round Leakage ($K_{0,0}$)

$$x_0 = S(\underline{p_0} \oplus K_{0,0})$$

The only leaking byte of first round is x_0. This leakage is used as in classical DPA-like attack. A set of trace is built with sole constraint on plaintext that $\underline{p_0}$ is variable. The variability of known $\underline{p_0}$ is used through a A-Type attack to recover the unknown constant $K_{0,0}$. This is the only key byte recoverable on first round as other bytes $x_{i,\forall i \in [1,\dots,15]}$ are not leaking, according to the considered leakage model.

Step 2: Second Round Leakage ($\mathscr{C}_0$)

$$y_0 = S\begin{pmatrix} 2{\cdot}\underline{x_0} \\ \oplus\ 3{\cdot}x_5 \\ \oplus\ 1{\cdot}x_{10} \\ \oplus\ 1{\cdot}x_{15} \\ \oplus\ K_{1,0} \end{pmatrix} = S\begin{pmatrix} 2{\cdot}S(\underline{p_0}\ \oplus\underline{K_{0,0}}) \\ \oplus\ 3{\cdot}S(\underline{p_5}\ \oplus K_{0,5}) \\ \oplus\ 1{\cdot}S(\underline{p_{10}} \oplus K_{0,10}) \\ \oplus\ 1{\cdot}S(\underline{p_{15}} \oplus K_{0,15}) \\ \oplus\ K_{1,0} \end{pmatrix}$$

The only leaking byte of second round is y_0. Its equation, obtained from AES algorithm specifications, implies four bytes of K_0 and one from K_1. $\underline{K_{0,0}}$ being known since attack Step 1, $\underline{x_0}$ is therefore also known. By building a set of traces

where p_0 is variable and other involved plaintext bytes are set to (any) constant value (i.e. P_5, P_{10}, P_{15}), then y_0 can be re-written as:

$$y_0 = S(2{\cdot}x_0 \oplus \mathscr{C}_0)$$

with an new constant $\mathscr{C}_0$:

$$\mathscr{C}_0 = 3{\cdot}X_5 \oplus X_{10} \oplus X_{15} \oplus K_{1,0}$$

where:

$$X_i = S(P_i \oplus K_{0,i})_{,\forall i \in \{5,10,15\}}$$

Now the only unknown element in this equation being $\mathscr{C}_0$, a A-Type attack can be conducted, targeting the leakage of y_0 and using the variability of p_0 to distinguish the correct value $\mathscr{C}_0$.

Step 3: Second Round Leakage ($K_{0,5}$, $K_{0,10}$, $K_{0,15}$) Now $\mathscr{C}_0$ is recovered, a new set of trace is run with p_5 variable, while P_{10} and P_{15} keep the very same constant used for previous step and no constraint on other bytes. Under this configuration, y_0 becomes:

$$y_0 = S\begin{pmatrix} 2{\cdot}x_0 \\ \oplus\ 3{\cdot}S(p_5 \oplus K_{0,5}) \\ \oplus\ 1{\cdot}X_{10} \\ \oplus\ 1{\cdot}X_{15} \\ \oplus\ K_{1,0} \end{pmatrix}$$

With x_0 a known value, p_5 a known variable, and $K_{0,5}$, X_{10}, X_{15}, $K_{1,0}$ unknown constants. The last three being part of known constant $\mathscr{C}_0$, they can be replaced as such:

$$X_{10} \oplus X_{15} \oplus K_{1,0} = 3{\cdot}X_5 \oplus \mathscr{C}_0$$

$$y_0 = S\begin{pmatrix} 2{\cdot}x_0 \\ \oplus\ 3{\cdot}S(p_5 \oplus K_{0,5}) \\ \oplus\ 3{\cdot}X_5 \\ \oplus\ \mathscr{C}_0 \end{pmatrix} = S\begin{pmatrix} 2{\cdot}x_0 \\ \oplus\ 3{\cdot}S(p_5 \oplus K_{0,5}) \\ \oplus\ 3{\cdot}S(P_5 \oplus K_{0,5}) \\ \oplus\ \mathscr{C}_0 \end{pmatrix}$$

Now the only unknown element in this equation being $K_{0,5}$, a A-Type attack can be conducted, targeting the leakage of y_0 and using the variability of p_5 to distinguish the correct value $K_{0,5}$.

The exact same method can be applied onto $K_{0,10}$ with a set where p_{10} is variable and P_{15} that keeps the very same constant used for previous step and no constraint over other bytes. This allows to retrieve the correct value $K_{0,10}$.

Then, the exact same method can be applied onto $K_{0,15}$ with a set where p_{15} is variable and no constraint on other bytes. This allows to retrieve the correct value $K_{0,15}$.

Finally, $K_{1,0}$ value is recovered for free (no additive set of traces required), with a simple computation now that $\underline{K_{0,5}}$, $\underline{K_{0,10}}$ and $\underline{K_{0,15}}$ are known:

$$\begin{aligned} K_{1,0} &= \underline{\mathscr{C}_0} \oplus 3{\cdot}\underline{X_5} \oplus \underline{X_{10}} \oplus \underline{X_{15}} \\ &= \underline{\mathscr{C}_0} \oplus 3{\cdot}S(\underline{P_5} \oplus \underline{K_{0,5}}) \oplus S(\underline{P_{10}} \oplus \underline{K_{0,10}}) \oplus S(\underline{P_{15}} \oplus \underline{K_{0,15}}) \end{aligned}$$

This induces that $\underline{y_0}$ is now known for any input value.

Step 4: Third Round Leakage ($\mathscr{C}_1$)

$$z_0 = S\begin{pmatrix} 2{\cdot}\underline{y_0} \\ \oplus\ 3{\cdot}y_5 \\ \oplus\ 1{\cdot}y_{10} \\ \oplus\ 1{\cdot}y_{15} \\ \oplus\ K_{2,0} \end{pmatrix}$$

The only leaking byte of third round is z_0. Its equation implies four bytes of second round *SubBytes* output and one byte of K_2. By developing the equation, one can see that all bytes of K_0, four of K_1 and one of K_2 are involved in total:

$$z_0 = S\left(\begin{array}{l} 2{\cdot}S\begin{pmatrix} 2{\cdot}S(\underline{p_0} \oplus \underline{K_{0,0}}) \\ \oplus\ 3{\cdot}S(\underline{p_5} \oplus \underline{K_{0,5}}) \\ \oplus\ 1{\cdot}S(\underline{p_{10}} \oplus \underline{K_{0,10}}) \\ \oplus\ 1{\cdot}S(\underline{p_{15}} \oplus \underline{K_{0,15}}) \\ \oplus\ \underline{K_{1,0}} \end{pmatrix} \oplus 3{\cdot}S\begin{pmatrix} 1{\cdot}S(\underline{p_4} \oplus K_{0,4}) \\ \oplus\ 2{\cdot}S(\underline{p_9} \oplus K_{0,9}) \\ \oplus\ 3{\cdot}S(\underline{p_{14}} \oplus K_{0,14}) \\ \oplus\ 1{\cdot}S(\underline{p_3} \oplus K_{0,3}) \\ \oplus\ K_{1,5} \end{pmatrix} \\ \oplus\ 1{\cdot}S\begin{pmatrix} 1{\cdot}S(\underline{p_8} \oplus K_{0,8}) \\ \oplus\ 1{\cdot}S(\underline{p_{13}} \oplus K_{0,13}) \\ \oplus\ 2{\cdot}S(\underline{p_2} \oplus K_{0,2}) \\ \oplus\ 3{\cdot}S(\underline{p_7} \oplus K_{0,7}) \\ \oplus\ K_{1,10} \end{pmatrix} \oplus 1{\cdot}S\begin{pmatrix} 3{\cdot}S(\underline{p_{12}} \oplus K_{0,12}) \\ \oplus\ 1{\cdot}S(\underline{p_1} \oplus K_{0,1}) \\ \oplus\ 1{\cdot}S(\underline{p_6} \oplus K_{0,6}) \\ \oplus\ 2{\cdot}S(\underline{p_{11}} \oplus K_{0,11}) \\ \oplus\ K_{1,15} \end{pmatrix} \oplus K_{2,0} \end{array}\right)$$

$\underline{y_0}$ being completely composed of known values also has this property. Using a set where all plaintext bytes are fixed, except at least one involved in $\underline{y_0}$, the equation can be re-written:

$$z_0 = S(2{\cdot}\underline{y_0} \oplus \mathscr{C}_1)$$

with a new constant $\mathscr{C}_1$:

$$\mathscr{C}_1 = 3{\cdot}Y_5 \oplus Y_{10} \oplus Y_{15} \oplus K_{2,0}$$

with:

$$\begin{aligned} X_i &= S(\underline{P_i} \oplus K_{0,i}), \forall i \in \{1,2,3,4,6,7,8,9,11,12,13,14\} \\ Y_5 &= S(1{\cdot}X_4 \oplus 2{\cdot}X_9 \oplus 3{\cdot}X_{14} \oplus 1{\cdot}X_3 \oplus K_{1,5}) \\ Y_{10} &= S(1{\cdot}X_8 \oplus 1{\cdot}X_{13} \oplus 2{\cdot}X_2 \oplus 3{\cdot}X_7 \oplus K_{1,10}) \\ Y_{15} &= S(3{\cdot}X_{12} \oplus 1{\cdot}X_1 \oplus 1{\cdot}X_6 \oplus 2{\cdot}X_{11} \oplus K_{1,15}) \end{aligned}$$

Now the only unknown element in this equation being $\mathscr{C}_1$, a A-Type attack can be conducted, targeting the leakage of z_0 and using the variability of $\underline{y_0}$ to distinguish the correct value $\underline{\mathscr{C}_1}$.

Step 5: Third Round Leakage (All remaining bytes) Now that $\underline{\mathscr{C}_1}$ is recovered, the following method can be applied in order to target the 12 remaining bytes. We describe here the method applied on y_5 in order to recover four other bytes of first sub-key ($K_{0,4}, K_{0,9}, K_{0,14}, K_{0,3}$) and one byte from second sub-key ($K_{1,5}$). The same method would then be applied onto y_{10} and y_{15} to recover the other remaining bytes. Only indexes of bytes and values of AES multiplicands will change. For each, we need an entry point to start with (Step 5.1) followed by the recovery of other implied bytes (Step 5.2).

Step 5.1: Getting an Entry Point The first byte targeted in y_5 can be more costly than the next ones as it requires a B-Type attack.

Any implied byte can be chosen to start with, here we arbitrarily choose $K_{0,4}$ byte as a target. A set is required with variability of plaintext byte $\underline{p_4}$, no constraint on bytes $\underline{p_0}, \underline{p_5}, \underline{p_{10}}, \underline{p_{15}}$ and other bytes must be at the constant value chosen during the recovery of $\underline{\mathscr{C}_1}$. Introducing the new constant $\mathscr{C}_2$ as:

$$\begin{aligned}\mathscr{C}_2 &= 2{\cdot}S(\underline{P_9} \oplus K_{0,9}) \oplus 3{\cdot}S(\underline{P_{14}} \oplus K_{0,14}) \oplus 1{\cdot}S(\underline{P_3} \oplus K_{0,3}) \oplus K_{1,5} \\ &= 2{\cdot}X_9 \oplus 3{\cdot}X_{14} \oplus 1{\cdot}X_3 \oplus K_{1,5}\end{aligned}$$

the intermediate value z_0 for this set of traces can be rewritten as:

$$z_0 = S\left(\begin{array}{l} 2{\cdot}\underline{y_0} \\ \oplus\ 3{\cdot}S(1{\cdot}S(\underline{p_4} \oplus K_{0,4}) \oplus \mathscr{C}_2) \\ \oplus\ Y_{10} \\ \oplus\ Y_{15} \\ \oplus\ K_{2,0} \end{array}\right)$$

With $\underline{y_0}$ a known value, $\underline{p_4}$ a known variable and $K_{0,4}$, $\mathscr{C}_2$, Y_{10}, Y_{15} and $K_{2,0}$ unknown constants. The last three being part of known constant $\underline{\mathscr{C}_1}$, they can be replaced:

$$\begin{aligned}Y_{10} \oplus Y_{15} \oplus K_{2,0} &= \underline{\mathscr{C}_1} \oplus 3{\cdot}Y_5 \\ &= \underline{\mathscr{C}_1} \oplus 3{\cdot}S(1{\cdot}X_4 \oplus 2{\cdot}X_9 \oplus 3{\cdot}X_{14} \oplus 1{\cdot}X_3 \oplus K_{1,5}) \\ &= \underline{\mathscr{C}_1} \oplus 3{\cdot}S(1{\cdot}X_4 \oplus \mathscr{C}_2) \\ &= \underline{\mathscr{C}_1} \oplus 3{\cdot}S(1{\cdot}S(\underline{P_4} \oplus K_{0,4}) \oplus \mathscr{C}_2)\end{aligned}$$

z_0, for this set, can then be re-written as:

$$z_0 = S\left(\begin{array}{l} 2{\cdot}\underline{y_0} \\ \oplus\ 3{\cdot}S(1{\cdot}S(\underline{p_4} \oplus K_{0,4}) \oplus \mathscr{C}_2) \\ \oplus\ 3{\cdot}S(1{\cdot}S(\underline{P_4} \oplus K_{0,4}) \oplus \mathscr{C}_2) \\ \oplus\ \underline{\mathscr{C}_1} \end{array}\right)$$

Now the only unknown element in this equation being the couple ($K_{0,4}$, $\mathscr{C}_2$), a B-Type attack can be conducted, targeting the leakage of z_0 and using the variability of $\underline{p_4}$ to distinguish the correct candidate for both values ($\underline{K_{0,4}}$, $\underline{\mathscr{C}_2}$).

Step 5.2: Exploiting the Entry Point Once the first key byte involved in y_5 is recovered, the other bytes can be targeted in the same way. However, as $\underline{\mathscr{C}_2}$ is now known, the cost is reduced to A-Type attacks instead of B-Type.

As an example, we detail here how to recover the next byte: $K_{0,9}$. A set is required with variability of plaintext byte $\underline{p_9}$, no constraints on bytes $\underline{p_0}, \underline{p_5}, \underline{p_{10}}, \underline{p_{15}}, \underline{p_4}$ and other bytes must be at the constant value chosen during the recovery of $\underline{\mathscr{C}_1}$. This configuration induces that the intermediate value z_0 can be rewritten as:

$$z_0 = S\left(\begin{array}{l} 2{\cdot}\underline{y_0} \\ \oplus\ 3{\cdot}S\left(\begin{array}{l} 1{\cdot}\underline{x_4} \\ \oplus\ 2{\cdot}S(\underline{p_9} \oplus K_{0,9}) \\ \oplus\ 3{\cdot}X_{14} \oplus 1{\cdot}X_3 \oplus K_{1,5} \end{array}\right) \\ \oplus\ Y_{10} \oplus Y_{15} \oplus K_{2,0} \end{array}\right)$$

As seen previously, some of the numerous unknown values can be merged. Indeed, Y_{10}, Y_{15} and $K_{2,0}$ are related to $\underline{\mathscr{C}_1}$, when X_{14}, X_3 and $K_{1,5}$ are related to $\underline{\mathscr{C}_2}$.

$$\begin{aligned} Y_{10} \oplus Y_{15} \oplus K_{2,0} &= \underline{\mathscr{C}_1} \oplus 3{\cdot}S(1{\cdot}\underline{X_4} \oplus \underline{\mathscr{C}_2}) \\ 3{\cdot}X_{14} \oplus 1{\cdot}X_3 \oplus K_{1,5} &= \underline{\mathscr{C}_2} \oplus 2{\cdot}S(\underline{P_9} \oplus K_{0,9}) \end{aligned}$$

This implies that z_0 can be re-written as:

$$z_0 = S\left(\begin{array}{l} 2{\cdot}\underline{y_0} \\ \oplus\ 3{\cdot}S\left(\begin{array}{l} 1{\cdot}\underline{x_4} \\ \oplus\ 2{\cdot}S(\underline{p_9} \oplus K_{0,9}) \\ \oplus\ 2{\cdot}S(\underline{P_9} \oplus K_{0,9}) \\ \oplus\ \underline{\mathscr{C}_2} \end{array}\right) \\ \oplus\ 3{\cdot}S(1{\cdot}\underline{X_4} \oplus \underline{\mathscr{C}_2}) \\ \oplus\ \underline{\mathscr{C}_1} \end{array}\right)$$

Now the only unknown element in this equation being $K_{0,9}$, a A-Type attack can be conducted, targeting the leakage of z_0 and using the variability of $\underline{p_9}$ to distinguish the correct value $\underline{K_{0,9}}$.

This can be adapted in order to target the two remaining bytes of K_0 involved in y_5 (i.e. $K_{0,14}$ and $K_{0,3}$), each one requiring a dedicated A-Type set with variant byte at its corresponding position. Once recovered, one byte from K_1 involved is also recovered for free (no additional set of traces required):

$$K_{1,5} = \underline{\mathscr{C}_2} \oplus 2{\cdot}\underline{X_9} \oplus 3{\cdot}\underline{X_{14}} \oplus 1{\cdot}\underline{X_3}$$

Now, Step 5.1 and then Step 5.2 can be repeated, targeting y_{10} to recover the four bytes of K_0 plus one additional from K_1 involved in its computation. The same can respectively be done on y_{15}. This attack finally reveals all bytes from $\underline{K_0}$ and four from K_1. Also, one byte from K_2 is recovered for free as being the only remaining unknown value in $\underline{\mathscr{C}_1}$ equation:

$$K_{2,0} = 3{\cdot}\underline{Y_5} \oplus \underline{Y_{10}} \oplus \underline{Y_{15}} \oplus \underline{\mathscr{C}_1}$$

3.2 Attack on Other Leaking Byte Positions

Section 3.1 details how to perform the attack when the leaking position is 0, 4 , 8 or 12. However, some steps need variations when the leaking position is one of the 12 others. Taking position 3 as an example (i.e. leakage only on (x_3, y_3, z_3)), we describe here how each step is impacted.

Step 1 No adaptation is required, this step reveal $K_{0,3}$ using the leakage of x_3:

$$x_3 = S(\underline{p_3} \oplus K_{0,3})$$

Step 2 This step needs an adaptation as $\underline{K_{0,3}}$ is not involved in computation of y_3, we therefore lack of an entry point:

$$y_3 = S\begin{pmatrix} 3{\cdot}x_0 \\ \oplus\ 1{\cdot}x_5 \\ \oplus\ 1{\cdot}x_{10} \\ \oplus\ 2{\cdot}x_{15} \\ \oplus\ K_{1,3} \end{pmatrix} = S\begin{pmatrix} 3{\cdot}S(\underline{p_0} \oplus K_{0,0}) \\ \oplus\ 1{\cdot}S(\underline{p_5} \oplus K_{0,5}) \\ \oplus\ 1{\cdot}S(\underline{p_{10}} \oplus K_{0,10}) \\ \oplus\ 2{\cdot}S(\underline{p_{15}} \oplus K_{0,15}) \\ \oplus\ K_{1,3} \end{pmatrix}$$

In order to pursue the attack as in Step 2 of Section 3.1, we have to recover one key byte involved. For sake of clarity we choose the same ($K_{0,0}$) but any involved byte could has been chosen. The same set of traces is used where one implied plaintext byte is variable (here $\underline{p_0}$) and other implied plaintext bytes are set to a constant value (here $\underline{P_5}$, $\underline{P_{10}}$ and $\underline{P_{15}}$):

$$\begin{aligned} y_3 &= S(3{\cdot}x_0 \oplus \mathscr{C}'_0) \\ &= S(3{\cdot}S(\underline{p_0} \oplus K_{0,0}) \oplus \mathscr{C}'_0) \end{aligned}$$

With $\mathscr{C}'_0$ a constant, that has the same role as $\mathscr{C}_0$ but adapted to changes in K_1 byte index and AES multiplicands variations:

$$\mathscr{C}'_0 = 1{\cdot}X_5 \oplus 1{\cdot}X_{10} \oplus 2{\cdot}X_{15} \oplus K_{1,3}$$

As the only unknown constants of the y_3 equation are $K_{0,0}$ and $\mathscr{C}'_0$ both can be recovered using a B-Type attack using the variability of $\underline{p_0}$.

Step 3 Once $\underline{K_{0,0}}$ and $\underline{\mathscr{C}'_0}$ recovered, Step 3 can be performed without need of adaptation leading to the successive recovery of $\underline{K_{0,5}}$, $\underline{K_{0,10}}$ and $\underline{K_{0,15}}$ and finally $\underline{K_{1,3}}$. This also gives knowledge of $\underline{y_3}$ for any input value. However, this will not be helpful for next step as $\underline{y_3}$ is not implied in z_3 computation.

Step 4 This step starts using third round leaking byte z_3:

$$z_3 = S\begin{pmatrix} 3{\cdot}y_0 \\ \oplus\ 1{\cdot}y_5 \\ \oplus\ 1{\cdot}y_{10} \\ \oplus\ 2{\cdot}y_{15} \\ \oplus\ K_{2,3} \end{pmatrix}$$

$$z_3 = S\left(\begin{array}{l} 3{\cdot}S\begin{pmatrix} 2{\cdot}S(\underline{p_0}\ \oplus\underline{K_{0,0}}) \\ \oplus\ 3{\cdot}S(\underline{p_5}\ \oplus\underline{K_{0,5}}) \\ \oplus\ 1{\cdot}S(\underline{p_{10}}\ \oplus\underline{K_{0,10}}) \\ \oplus\ 1{\cdot}S(\underline{p_{15}}\ \oplus\underline{K_{0,15}}) \\ \oplus\ K_{1,0} \end{pmatrix} \oplus 1{\cdot}S\begin{pmatrix} 1{\cdot}S(\underline{p_4}\ \oplus K_{0,4}) \\ \oplus\ 2{\cdot}S(\underline{p_9}\ \oplus K_{0,9}) \\ \oplus\ 3{\cdot}S(\underline{p_{14}}\ \oplus K_{0,14}) \\ \oplus\ 1{\cdot}S(\underline{p_3}\ \oplus K_{0,3}) \\ \oplus\ K_{1,5} \end{pmatrix} \\ \oplus\ 1{\cdot}S\begin{pmatrix} 1{\cdot}S(\underline{p_8}\ \oplus K_{0,8}) \\ \oplus\ 1{\cdot}S(\underline{p_{13}}\ \oplus K_{0,13}) \\ \oplus\ 2{\cdot}S(\underline{p_2}\ \oplus K_{0,2}) \\ \oplus\ 3{\cdot}S(\underline{p_7}\ \oplus K_{0,7}) \\ \oplus\ K_{1,10} \end{pmatrix} \oplus 2{\cdot}S\begin{pmatrix} 3{\cdot}S(\underline{p_{12}}\ \oplus K_{0,12}) \\ \oplus\ 1{\cdot}S(\underline{p_1}\ \oplus K_{0,1}) \\ \oplus\ 1{\cdot}S(\underline{p_6}\ \oplus K_{0,6}) \\ \oplus\ 2{\cdot}S(\underline{p_{11}}\ \oplus K_{0,11}) \\ \oplus\ K_{1,15} \end{pmatrix} \oplus K_{2,3} \end{array}\right)$$

The Step 4 process described in Section 3.1 can be used except that, this time, a B-Type attack will be necessary instead of a A-Type one. The attack can be performed thanks to re-writing the equation:

$$z_0 = S(3{\cdot}S\begin{pmatrix} 2{\cdot}\underline{x_0} \\ \oplus\ 3{\cdot}\underline{X_5} \\ \oplus\ 1{\cdot}\underline{X_{10}} \\ \oplus\ 1{\cdot}\underline{X_{15}} \\ \oplus\ K_{1,0} \end{pmatrix} \oplus \mathscr{C}'_1)$$

with a new constant $\mathscr{C}'_1$:

$$\mathscr{C}'_1 = Y_5 \oplus Y_{10} \oplus 2{\cdot}Y_{15} \oplus K_{2,3}$$

with:

$$\begin{aligned} X_i &= S(\underline{p_i} \oplus K_{0,i})_{,\forall i \in \{1,2,3,4,6,7,8,9,11,12,13,14\}} \\ Y_5 &= S(1{\cdot}X_4 \oplus 2{\cdot}X_9 \oplus 3{\cdot}X_{14} \oplus 1{\cdot}X_3 \oplus K_{1,5}) \\ Y_{10} &= S(1{\cdot}X_8 \oplus 1{\cdot}X_{13} \oplus 2{\cdot}X_2 \oplus 3{\cdot}X_7 \oplus K_{1,10}) \\ Y_{15} &= S(3{\cdot}X_{12} \oplus 1{\cdot}X_1 \oplus 1{\cdot}X_6 \oplus 2{\cdot}X_{11} \oplus K_{1,15}) \end{aligned}$$

Using the variability of $\underline{x_0}$ and the knowledge of constants $\underline{X_5}, \underline{X_{10}}, \underline{X_{15}}$, the couple of unknown constants $(K_{1,0}, \mathscr{C}'_1)$ can be recovered through a B-Type attack.

Step 5: No adaptation is required, the knowledge of $\underline{y_0}$ and $\underline{\mathscr{C}'_1}$ allowing to apply the very same strategy.

One can notice that one byte of the remaining ones is already known thanks to Step 1. Therefore, one of the byte does not need to be recovered and as it

offers an entry point, one of the B-Type Step 5.1 can be changed into a A-Type, or one A-Type from Step 5.2 can be skipped, depending of the most favorable scenario for the attacker.

4 Cost of Attack & Optimization

4.1 Cost Before Optimization

Without optimization, the number of sets of A-Type or B-Type required to perform an attack targeting an AES can be established depending of the leaking byte location.

The non-optimized attack for leaking position 0 can be summarized in the two sub-tables in Table 1. The left one lists all sets that have to be generated

Set ref.	Type	Varying bytes
$\mathscr{S}_1$	A	P_0
$\mathscr{S}_2$	A*	P_0
$\mathscr{S}_3$	A	P_5
$\mathscr{S}_4$	A	P_{10}
$\mathscr{S}_5$	A	P_{15}
$\mathscr{S}_6$	A*	P_0
$\mathscr{S}_7$	B	P_4
$\mathscr{S}_8$	A	P_9
$\mathscr{S}_9$	A	P_{14}
$\mathscr{S}_{10}$	A	P_3
$\mathscr{S}_{11}$	B	P_8
$\mathscr{S}_{12}$	A	P_{13}
$\mathscr{S}_{13}$	A	P_2
$\mathscr{S}_{14}$	A	P_7
$\mathscr{S}_{15}$	B	P_{12}
$\mathscr{S}_{16}$	A	P_1
$\mathscr{S}_{17}$	A	P_6
$\mathscr{S}_{18}$	A	P_{11}

Step	Set	Target	Free	Num. sets	Key bits
1	$\mathscr{S}_1$	$K_{0,0}$	$\emptyset$	1	120
2	$\mathscr{S}_2$	$\mathscr{C}_0$	$\emptyset$	2	120
3	$\mathscr{S}_3$	$K_{0,5}$	$\emptyset$	3	112
3	$\mathscr{S}_4$	$K_{0,10}$	$\emptyset$	4	104
3	$\mathscr{S}_5$	$K_{0,15}$	$K_{1,0}$	5	96
4	$\mathscr{S}_6$	$\mathscr{C}_1$	$\emptyset$	6	96
5.1	$\mathscr{S}_7$	$K_{0,4}$,$\mathscr{C}_2$	$\emptyset$	7	88
5.2	$\mathscr{S}_8$	$K_{0,9}$	$\emptyset$	8	80
5.2	$\mathscr{S}_9$	$K_{0,14}$	$\emptyset$	9	72
5.2	$\mathscr{S}_{10}$	$K_{0,3}$	$K_{1,5}$	10	64
5.1	$\mathscr{S}_{11}$	$K_{0,8}$,$\mathscr{C}_2'$	$\emptyset$	11	56
5.2	$\mathscr{S}_{12}$	$K_{0,13}$	$\emptyset$	12	48
5.2	$\mathscr{S}_{13}$	$K_{0,2}$	$\emptyset$	13	40
5.2	$\mathscr{S}_{14}$	$K_{0,7}$	$K_{1,10}$	14	32
5.1	$\mathscr{S}_{15}$	$K_{0,12}$, $\mathscr{C}_2''$	$\emptyset$	15	24
5.2	$\mathscr{S}_{16}$	$K_{0,1}$	$\emptyset$	16	16
5.2	$\mathscr{S}_{17}$	$K_{0,6}$	$\emptyset$	17	8
5.2	$\mathscr{S}_{18}$	$K_{0,11}$	$K_{1,15}$, $K_{2,0}$	18	0

Table 1. Summary of non-optimized version of attack when leaking byte position is 0 on all rounds.

giving them a reference ($\mathscr{S}_{i,\forall i\in[1,\ldots,18]}$), giving the type of Attack (A-Type or B-Type) and the exhaustive list of varying bytes (non-listed bytes are set to a constant value or have no constraint). The right table shows the order of the attack steps, the set used, the value targeted and recovered during the attack step, the values obtained for free at the end of the attack step, the total number of sets used so far and, finally, the remaining entropy of the targeted key at the end of the step.

This table is also valid for byte leakage positions 4, 8 and 12 up to a reordering of byte recovering order. The cost for full key recovery is therefore 18 sets (15 A-Type + 3 B-Type).

For other leaking byte positions the cost is 17 sets (12 A-Type + 5 B-Type)[1] as two adaptations occur:

- A-Type sets noted A* are sets that become B-Type.
- One A-Type of Step 5.2 is not required as it already recovered in Step 1. [2]

As made to be generic, the method described here implies a inefficient cost in number of sets that could be reduced thanks to optimization methods described in following sections. As discussed in Section 5, this method can also be applied if the three leaking bytes are not aligned.

4.2 Optimization: Re-Usage of Sets

Constraint on sets can be shared by different steps. This allows to use the same set, avoiding unnecessary acquisitions. As an example for byte leaking position 0, Step 1, Step 2 and Step 4 can all be performed onto a set where only p_0 byte varies when all others are fixed. This allows these three steps to be performed onto the same set of traces, saving the cost of two sets not required anymore.

4.3 Optimization: Planning of Sets

As detailed during attack process, the constraints on sets depend on the attack step and tend to become stricter as the attack progresses. However, it is possible to anticipate future needs and plan set construction to be re-usable in the future. As an example, for byte leaking position 0, Step 3 and Step 5 can be combined. Step 3 requires sets to be created with constraint only on some plaintext byte position (i.e. 0, 5, 10 and 15) when Step 5 requires no constraints over them at all. This implies that the three sets used in Step 3 can also carry a constraint for Step 5 and be used in both, reducing the cost in number of sets.

4.4 Optimization: Exploiting Relation in Sub-Keys

Until now the key bytes are taken as unrelated values. However, round keys are computed from the main key, thus, relations between bytes of different round keys appear, depending on the AES size.

[1] Or 17 sets (13 A-Type + 4 B-Type) if less costly. See Footnote 2.

[2] If B-Type/A-Type cost ratio is over 2, it is more effective to use this knowledge to convert one B-Type of Step 5.1 into a A-Type.

For example, for AES-128, the following equations lie between the three first sub-key bytes:

$$\begin{aligned}
K_{1,0} &= K_{0,0} \oplus S(K_{0,13}) \oplus 1 & K_{2,0} &= K_{1,0} \oplus S(K_{1,13}) \oplus 2 \\
K_{1,1} &= K_{0,1} \oplus S(K_{0,14}) & K_{2,1} &= K_{1,1} \oplus S(K_{1,14}) \\
K_{1,2} &= K_{0,2} \oplus S(K_{0,15}) & K_{2,2} &= K_{1,2} \oplus S(K_{1,15}) \\
K_{1,3} &= K_{0,3} \oplus S(K_{0,12}) & K_{2,3} &= K_{1,3} \oplus S(K_{1,12}) \\
K_{1,4} &= K_{0,4} \oplus K_{1,0} & K_{2,4} &= K_{1,4} \oplus K_{2,0} \\
K_{1,5} &= K_{0,5} \oplus K_{1,1} & K_{2,5} &= K_{1,5} \oplus K_{2,1} \\
K_{1,6} &= K_{0,6} \oplus K_{1,2} & K_{2,6} &= K_{1,6} \oplus K_{2,2} \\
K_{1,7} &= K_{0,7} \oplus K_{1,3} & K_{2,7} &= K_{1,7} \oplus K_{2,3} \\
K_{1,8} &= K_{0,8} \oplus K_{1,4} & K_{2,8} &= K_{1,8} \oplus K_{2,4} \\
K_{1,9} &= K_{0,9} \oplus K_{1,5} & K_{2,9} &= K_{1,9} \oplus K_{2,5} \\
K_{1,10} &= K_{0,10} \oplus K_{1,6} & K_{2,10} &= K_{1,10} \oplus K_{2,6} \\
K_{1,11} &= K_{0,11} \oplus K_{1,7} & K_{2,11} &= K_{1,11} \oplus K_{2,7} \\
K_{1,12} &= K_{0,12} \oplus K_{1,8} & K_{2,12} &= K_{1,12} \oplus K_{2,8} \\
K_{1,13} &= K_{0,13} \oplus K_{1,9} & K_{2,13} &= K_{1,13} \oplus K_{2,9} \\
K_{1,14} &= K_{0,14} \oplus K_{1,10} & K_{2,14} &= K_{1,14} \oplus K_{2,10} \\
K_{1,15} &= K_{0,15} \oplus K_{1,11} & K_{2,15} &= K_{1,15} \oplus K_{2,11}
\end{aligned}$$

Always related by trio, it means that when two bytes of any trio are known, then the third can be computed for free.

As an example, when the leaking byte is 0, at the end of Step 3 of the attack, the attacker knows, at least, $K_{0,0}$ and $K_{1,0}$ thanks to SCA attacks. This implies that he gains the knowledge of $K_{0,13}$ for free as:

$$K_{0,13} = S^{-1}(K_{0,0} \oplus K_{1,0} \oplus 1)$$

This byte does not have to be recovered during Step 5 and therefore avoid the creation of a set dedicated to its recovery[3].

4.5 Example of Optimized Attack

Optimization methods detailed above can benefit from various equations that depends on the bytes recovered during the attack. Anytime a choice can be made in the order of byte recovery during the attack described in Section 3 we must choose wisely in order to maximize the information gained and thus reduce the cost of attack by optimizing the number of sets required. Each leaking byte position induce different equations, implying that each configuration must have a proper optimized version.

Here is an example of such optimized attack on leaking byte position 0:

Step 1 need one A-Type set of traces with p_0 variable and all other bytes fixed to constant value in order to be shared by Step 2 and Step 4. It recovers $K_{0,0}$.

[3] Or allows to change a B-Type set into a A-Type one. See Footnote 2

Step 2 does not need a dedicated set as it uses the one created at Step 1. it recovers $\mathscr{C}_0$.

Step 3 needs three A-Type set of traces:

- One with p_5 and p_8 (planned for Step 5.2) variable when other plaintext bytes are constants. It recovers $K_{0,5}$.
- One with p_{10} and p_2 (planned for Step 5.2) variable when other plaintext bytes are constants. It recovers $K_{0,10}$.
- One with p_{15} and p_7 (planned for Step 5.2) variable when other plaintext bytes are constants. It recovers $K_{0,15}$.

Once these bytes are recovered, they can be combined with $\mathscr{C}_0$ in order to get $K_{1,0}$. Using key relation between $K_{0,0}$, $K_{0,13}$ and $K_{1,0}$, it allows to recover $K_{0,13}$.

Step 4 does not need a dedicated set as it uses the one created at Step 1. It recovers $\mathscr{C}_1$.

In Step 5, as $K_{0,13}$ is already known, the attacker can start by targeting y_{10} where this key byte is involved. It still requires a set where p_{13} is variable to perform Step 5.1 by targeting ($K_{0,13}$, $\mathscr{C}_2$). However, as $K_{0,13}$ is already known the set falls from B-Type into A-Type.

Step 5.2 is performed targeting $K_{0,8}$, it should have required another A-Type set with p_8 variable, however, as it was anticipated, it can use the first set of Step 3.

Step 5.2 is performed targeting $K_{0,2}$, it should have required another A-Type set with p_2 variable, however, as it was anticipated, it can use the second set of Step 3. The knowledge of $K_{0,2}$ and $K_{0,15}$ can be used with the key byte relations to acquire for free the knowledge of $K_{1,2}$.

Step 5.2 is performed targeting $K_{0,7}$, it should have required another A-Type set with p_7 variable, however, as it was anticipated, it can use the third set of Step 3. This step completes the attack on y_{10} revealing the value of $K_{1,10}$. Several equations of key byte relations are triggered in cascade:

- $K_{1,6}$ is retrieved thanks to $K_{0,10}$ and $K_{1,10}$
- $K_{0,6}$ is retrieved thanks to $K_{1,2}$ and $K_{1,6}$

As $K_{0,6}$ is involved in computation of y_{15}, the attacker can pursue the attack by targeting this element. Its Step 5.1 is also reduced from a B-Type attack into a A-Type one, thanks to the knowledge of $K_{0,6}$. One A-Type set is therefore required with p_6 varying in order to complete this step.

Step 5.2 is performed targeting $K_{0,12}$, it requires another A-Type set with p_{12} variable.

Step 5.2 is performed targeting $K_{0,1}$, it requires another A-Type set with p_1 variable.

Step 5.2 is performed targeting $K_{0,11}$, it requires another A-Type set with p_{11} variable. This step completes the attack on y_{15} revealing the value of $K_{1,15}$. Several equations of key byte relations are triggered in cascade:

- $K_{2,2}$ is retrieved thanks to $K_{1,2}$ and $K_{1,15}$

- $K_{2,6}$ is retrieved thanks to $K_{2,2}$ and $K_{1,6}$
- $K_{2,10}$ is retrieved thanks to $K_{2,6}$ and $K_{1,10}$
- $K_{1,11}$ is retrieved thanks to $K_{0,15}$ and $K_{1,15}$
- $K_{1,7}$ is retrieved thanks to $K_{0,11}$ and $K_{1,11}$
- $K_{1,3}$ is retrieved thanks to $K_{0,7}$ and $K_{1,7}$
- $K_{0,3}$ is retrieved thanks to $K_{0,12}$ and $K_{1,3}$

The knowledge of $K_{0,3}$ allows to reduce the cost of Step 5.1 performed on y_5 from a B-Type to a A-Type set. One A-Type set is required with p_3 variable allowing to recover the value of $\mathscr{C}_2$:

$$\mathscr{C}_2 = 1{\cdot}X_4 \oplus 2{\cdot}X_9 \oplus 3{\cdot}X_{14} \oplus K_{1,5}$$

Now, before pursuing to the next byte, a combination of $\mathscr{C}_1$ and $\mathscr{C}_2$ can be done, leading to the recovery of $K_{2,0}$. Indeed, the equation of $\mathscr{C}_1$ can be re-written:

$$K_{2,0} = \mathscr{C}_1 \oplus 3{\cdot}Y_5 \oplus Y_{10} \oplus Y_{15}$$

where $\mathscr{C}_1$, Y_{10} and Y_{15} are known from previous attack steps. Thus, only Y_5 status remain undetermined but can also be re-written:

$$\begin{aligned} Y_5 &= S(1{\cdot}X_4 \oplus 2{\cdot}X_9 \oplus 3{\cdot}X_{14} \oplus K_{1,5} \oplus 1{\cdot}X_3) \\ &= S(\mathscr{C}_2 \oplus 1{\cdot}X_3) \end{aligned}$$

$\mathscr{C}_2$ and X_3 (through $K_{0,3}$) being known, then Y_5 is also known, and so is $K_{2,0}$. Several equations of key byte relations are triggered in cascade:

- $K_{1,13}$ is retrieved thanks to $K_{1,0}$ and $K_{2,0}$
- $K_{1,9}$ is retrieved thanks to $K_{0,13}$ and $K_{1,13}$

Step 5.2 is performed targeting $K_{0,14}$, it requires another A-Type set with p_{14} variable.

The only two remaining byte $K_{0,4}$ and $K_{0,9}$ would have required one A-Type set for each, however, thanks to key relations, the attack is in fact already finished. Indeed, several equations of key byte relations are triggered in cascade:

- $K_{1,1}$ is retrieved thanks to $K_{0,1}$ and $K_{0,14}$
- $K_{1,5}$ is retrieved thanks to $K_{0,5}$ and $K_{1,1}$
- $K_{0,9}$ is retrieved thanks to $K_{1,5}$ and $K_{1,9}$

Once $K_{0,9}$ recovered, $\mathscr{C}_2$ equation reveal the last unknown K_0 byte: $K_{0,4}$:

$$\begin{aligned} & \mathscr{C}_2 = 1{\cdot}X_4 \oplus 2{\cdot}X_9 \oplus 3{\cdot}X_{14} \oplus K_{1,5} \\ \Rightarrow \quad & X_4 = \mathscr{C}_2 \oplus 2{\cdot}X_9 \oplus 3{\cdot}X_{14} \oplus K_{1,5} \\ \Rightarrow \quad & S(P_4 \oplus K_{0,4}) = \mathscr{C}_2 \oplus 2{\cdot}X_9 \oplus 3{\cdot}X_{14} \oplus K_{1,5} \\ \Rightarrow \quad & K_{0,4} = P_4 \oplus S^{-1}(\mathscr{C}_2 \oplus 2{\cdot}X_9 \oplus 3{\cdot}X_{14} \oplus K_{1,5}) \end{aligned}$$

Optimization made in this example reduces the cost of the attack from 18 sets (mixed A-Type and B-Type) to 11 sets (only A-Type). Alternative optimization could have removed A-Type sets instead of changing B-Type into A-Type.

Set ref.	Type	Varying bytes
$\mathscr{S}_1$	A	P_0
$\mathscr{S}_2$	A	P_5 & P_8
$\mathscr{S}_3$	A	P_{10}& P_2
$\mathscr{S}_4$	A	P_{15} & P_7
$\mathscr{S}_5$	A	P_{13}
$\mathscr{S}_6$	A	P_6
$\mathscr{S}_7$	A	P_{12}
$\mathscr{S}_8$	A	P_1
$\mathscr{S}_9$	A	P_{11}
$\mathscr{S}_{10}$	A	P_3
$\mathscr{S}_{11}$	A	P_{14}

Step	Set	Target	Free	Num. sets	Key bits
1	$\mathscr{S}_1$	$K_{0,0}$	$\emptyset$	1	120
2	$\mathscr{S}_1$	$\mathscr{C}_0$	$\emptyset$	1	120
3	$\mathscr{S}_2$	$K_{0,5}$	$\emptyset$	2	112
3	$\mathscr{S}_3$	$K_{0,10}$	$\emptyset$	3	104
3	$\mathscr{S}_4$	$K_{0,15}$	$K_{1,0}, K_{0,13}$	4	88
4	$\mathscr{S}_1$	$\mathscr{C}_1$	$\emptyset$	4	88
5.1	$\mathscr{S}_5$	$\mathscr{C}_2'$	$\emptyset$	5	88
5.2	$\mathscr{S}_2$	$K_{0,8}$	$\emptyset$	5	80
5.2	$\mathscr{S}_3$	$K_{0,2}$	$K_{1,2}$	5	72
5.2	$\mathscr{S}_4$	$K_{0,7}$	$K_{1,10}, K_{1,6}, K_{0,6}$	5	56
5.1	$\mathscr{S}_6$	$\mathscr{C}_2''$	$\emptyset$	6	56
5.2	$\mathscr{S}_7$	$K_{0,12}$	$\emptyset$	7	48
5.2	$\mathscr{S}_8$	$K_{0,1}$	$K_{1,2}$	8	40
5.2	$\mathscr{S}_9$	$K_{0,11}$	$\begin{cases} K_{1,15}, K_{2,2}, K_{2,6}, K_{2,10}, \\ K_{1,11}, K_{1,7}, K_{1,3}, K_{0,3} \end{cases}$	9	24
5.1	$\mathscr{S}_{10}$	$\mathscr{C}_2$	$K_{2,0}, K_{1,13}, K_{1,9}$	10	16
5.2	$\mathscr{S}_{11}$	$K_{0,14}$	$K_{1,1}, K_{1,5}, K_{0,9}$	11	0

Table 2. Summary of optimized version of attack when leaking byte position is 0 on all rounds.

5 Discussion

5.1 Reduced Cost

One can notice in Table 2 that thanks to optimization, only 11 A-Type sets are now sufficient to recover the whole key. If an attacker also owns at least one pair of corresponding (plaintext, ciphertext) then a brute force search can be conducted, reducing the number of sets required. If a demanding, but not unrealistic, 2^{56} brute force is possible, then the attack can be done using only 5 A-Type sets. This result is only applicable when leakage appears on byte at position 0 for all rounds. Our approach must be adapted to the observed leakage as each configuration will imply different equations.

A summary of an optimized attack for another one of the 16 positions is provided as an example in Table 3 in Annex A. It gives results close to the ones described in Table 2.

Furthermore, depending on the cost ratio between A-Type and B-Type attacks, choices can be adapted to reduce the total cost. Indeed, if this ratio is close to 1, reducing a B-Type attack into A-Type one is not relevant. In this situation, a skip of a A-Type attack (if possible) would be more effective in terms of attack cost reduction.

5.2 About Optimality

Optimization methods described here help to reduce the multiplicative factor applied to the number of set of traces required to perform the attack. However,

as a lot of variations and choices can be considered, we do not claim that the optimized version proposed here is the best one. Further research may find better solutions to each situation.

5.3 Adaptation to Any Leakage Location

Considering a scenario where the leaking byte location changes at each round, the attack can still be performed. Indeed, the first round always allows to recover one key byte thanks to x_{i_1}. If the recovered byte is (respectively is not) involved in y_{i_2} computation, then a A-Type (respectively B-Type) attack is required as described in Section 3.1 (respectively Section 3.2). If the K_1 byte recovered for free during round 2 attack is not the one used at the correct position in the third round equation z_{i_3}, then the attack requires the adaptation described in Section 3.2 to recover it before recovering the 12 remaining bytes. A summary of an optimized attack in such a configuration is provided as an example in Table 4 in Annex A. It gives results close to the ones described in Table 2.

5.4 Adaptation to AES Sizes

The non-optimized methodology described in Section 3.1 does not assume the AES size and can therefore be applied for each one. For AES-128, the attack ends after Step 5 when all 16 bytes of K_0 are recovered as it represents the whole main key. However, for higher sizes, bytes are missing and must be recovered. For AES-192, 8 more bytes from K_1 are required being $K_{1,i,\forall i\in[0,7]}$, when for AES-256, all 16 bytes from K_1 are required.

Our solution is to use the control gained over K_0 that gives control over the input of second round. Thus, if leakage of fourth round occurs, the very same methodology can be applied starting one round later to recover all or part of K_1 as it takes the role of K_0. One can notice that this second attack phase will cost less, as several bytes of K_1 and one of K_2 are already known through the attack process of K_0 disclosure. Optimization methods can also be applied, including using equations specific to the AES's size key schedule.

5.5 Simulation

Simulations were run in order to confirm the viability of our attack strategy. As we do not challenge the effectiveness of DPA-like attacks, we choose to consider a number of trace per set sufficient to have a high probability for the good candidate to emerge from wrong ones.

For each experiment: one 16-byte key is randomly chosen and one random value is chosen for each plaintext byte to be used in the sets where it needs to be set to a constant. Sets are generated according to the attack methodology and a CPA attack is performed. The candidate with the highest score is recorded as being the targeted constant. At the end, the recovered key is controlled to be the chosen one to control the success of the attack.

For each set: a set of 1,000 traces (10,000 for B-Type attacks) is generated using the key chosen for the experiment. The pattern of plaintext bytes: variable (draw at random) or constant (set to chosen fix value) is selected according to the attack step.

For each trace: 3×16 points are generated, being the concatenation of Hamming weights of values of the 16 bytes of x then the ones from y and z. All bytes are masked beforehand by a random value. The mask is draw at random for each byte and for each trace, except for the three bytes intended to leak that are not masked. Finally, a normal random noise (standard deviation of 1) is added on each point individually.

Experiments were run 10,000 times for each of the 16 aligned leakage positions $(x_i, y_i, z_i)_{,\forall i \in [0,15]}$ in non-optimized version of the attack. It was also run 10,000 times on optimized version of the attack for leakage at position 0 (x_0, y_0, z_0). All experiments lead to correct key recovery.

5.6 Further Work

Our chosen plaintext approach manages to recover secret key in very low leakage context for AES and might be considered for adaptation to other algorithms.

Further research can be considered in order to establish lower cost or prove optimality of strategies. Other leakage location in the algorithm can also be considered such as *SubBytes* input or during *MixColumns* computation.

6 Conclusion

In this paper, we show how, even in a low information leakage context, a chosen plaintext side channel attack can lead to recover the key of an AES. Indeed, only one leaking bit per round at output of *SubBytes* are sufficient to perform our attack. We create a general methodology and completed it with several methods allowing to optimize and reduce the attack cost. The cost of chosen plaintext attacks being the number of sets required, a naive approach lead to a cost of 18 sets to recover a full AES-128 key in the chosen example scenario. Using optimization techniques, we manage to reduce the cost to 11 sets. This can finally be reduced to only 5 sets in case the attacker can perform a brute force search of 2^{56} AES.

A Optimized Attack Examples in Other Configurations

Table 3 shows the summary of an optimized strategy of attack when leakages are located on byte 6 on every rounds, i.e. (x_6, y_6, z_6). When Table 4 shows the summary of an optimized strategy of attack when leakages are located on different byte on every rounds, here (x_9, y_2, z_{13}) was randomly chosen. This illustrates the fact that the attack also work and can be optimized in these kind of configurations.

Set ref.	Type	Varying bytes
$\mathscr{S}_1$	A	P_6
$\mathscr{S}_2$	B	P_4
$\mathscr{S}_3$	A	P_9 & P_{12}
$\mathscr{S}_4$	A	P_{14}& P_1
$\mathscr{S}_5$	A	P_3 & P_{11}
$\mathscr{S}_6$	A	P_{10}
$\mathscr{S}_7$	A	P_0
$\mathscr{S}_8$	A	P_5
$\mathscr{S}_9$	A	P_2

Step	Set	Target	Free	Num. sets	Key bits
1	$\mathscr{S}_1$	$K_{0,6}$	$\emptyset$	1	120
2	$\mathscr{S}_2$	$K_{0,4}, \mathscr{C}_0$	$\emptyset$	2	112
3	$\mathscr{S}_3$	$K_{0,9}$	$\emptyset$	3	104
3	$\mathscr{S}_4$	$K_{0,14}$	$\emptyset$	4	96
3	$\mathscr{S}_5$	$K_{0,3}$	$K_{1,6}, K_{1,2}$	5	88
4	$\mathscr{S}_2$	$K_{1,4}, \mathscr{C}_1$	$K_{1,0}$	5	88
5.1	$\mathscr{S}_1$	$\mathscr{C}_2$	$\emptyset$	5	88
5.2	$\mathscr{S}_3$	$K_{0,12}$	$K_{1,3}$	5	80
5.2	$\mathscr{S}_4$	$K_{0,1}$	$K_{1,1}$	5	72
5.2	$\mathscr{S}_5$	$K_{0,11}$	$K_{1,14}, K_{1,10}, K_{0,10}, K_{2,1}$	5	56
5.1	$\mathscr{S}_6$	$\mathscr{C}_2'$	$\emptyset$	6	56
5.2	$\mathscr{S}_7$	$K_{0,0}$	$K_{0,13}$	7	40
5.2	$\mathscr{S}_8$	$K_{0,5}$	$K_{1,5}, K_{1,9}, K_{1,13}, K_{2,0}, K_{2,4}, K_{2,5}, K_{2,9}, K_{2,13} K_{0,15}, K_{0,2}$	8	16
5.1	$\mathscr{S}_9$	$\mathscr{C}_2''$	$K_{2,6}, K_{2,2}, K_{1,15}, K_{1,11}, K_{1,7}, K_{0,7}, K_{0,8}$	9	0

Table 3. Summary of optimized version of attack when leaking byte position is 6 on all rounds.

Set ref.	Type	Varying bytes
$\mathscr{S}_1$	A	P_9
$\mathscr{S}_2$	B	P_0
$\mathscr{S}_3$	A	P_5 & P_4
$\mathscr{S}_4$	A	P_{10}& P_{14}
$\mathscr{S}_5$	A	P_{15} & P_3
$\mathscr{S}_6$	A	P_1
$\mathscr{S}_7$	A	P_{11}
$\mathscr{S}_8$	A	P_{12}
$\mathscr{S}_9$	A	P_2
$\mathscr{S}_{10}$	A	P_7

Step	Set	Target	Free	Num. sets	Key bits
1	$\mathscr{S}_1$	$K_{0,9}$	$\emptyset$	1	120
2	$\mathscr{S}_2$	$K_{0,0}, \mathscr{C}_0$	$\emptyset$	2	112
3	$\mathscr{S}_3$	$K_{0,5}$	$\emptyset$	3	104
3	$\mathscr{S}_4$	$K_{0,10}$	$\emptyset$	4	96
3	$\mathscr{S}_5$	$K_{0,15}$	$K_{1,2}, K_{0,2}$	5	80
4	$\mathscr{S}_2$	$K_{1,1}, \mathscr{C}_1$	$K_{1,5}, K_{1,9}$	5	80
5.1	$\mathscr{S}_1$	$\mathscr{C}_2$	$\emptyset$	5	80
5.2	$\mathscr{S}_3$	$K_{0,4}$	$\emptyset$	5	72
5.2	$\mathscr{S}_4$	$K_{0,14}$	$K_{0,1}$	5	56
5.2	$\mathscr{S}_5$	$K_{0,3}$	$\begin{cases} K_{1,6}, K_{0,6}, K_{1,10}, K_{1,14} \\ K_{2,1}, K_{2,5}, K_{2,9} \end{cases}$	5	40
5.1	$\mathscr{S}_6$	$\mathscr{C}_2'$	$\emptyset$	6	40
5.2	$\mathscr{S}_7$	$K_{0,11}$	$\emptyset$	7	32
5.2	$\mathscr{S}_8$	$K_{0,12}$	$K_{1,3}, K_{1,12}, K_{1,8}, K_{2,3}$	8	24
5.1	$\mathscr{S}_9$	$\mathscr{C}_2''$	$\begin{cases} K_{2,13}, K_{1,13}, K_{0,13}, K_{1,0} \\ K_{1,4}, K_{0,8}, K_{2,0}, K_{2,4} \\ K_{2,8}, K_{2,12} \end{cases}$	9	8
5.2	$\mathscr{S}_{10}$	$K_{0,7}$	$N.A.$	10	0

Table 4. Summary of optimized version of attack when leaking byte position is 9 on first round, 2 on second round and 13 on third round.

References

1. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3156, pp. 16–29. Springer (2004). https://doi.org/10.1007/978-3-540-28632-5"˙2, https://doi.org/10.1007/978-3-540-28632-5_2
2. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. pp. 398–412. Lecture Notes in Computer Science, Springer (1999)
3. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2162, pp. 251–261. Springer (2001). https://doi.org/10.1007/3-540-44709-1"˙21, https://doi.org/10.1007/3-540-44709-1_21
4. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Koblitz, N. (ed.) Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1109, pp. 104–113. Springer (1996). https://doi.org/10.1007/3-540-68697-5"˙9, https://doi.org/10.1007/3-540-68697-5_9

5. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1"˜25, https://doi.org/10.1007/3-540-48405-1_25
6. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Power analysis attacks of modular exponentiation in smartcards. In: Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings. pp. 144–157. Lecture Notes in Computer Science, Springer (1999)
7. NIST: Fips 197 advanced encryption standard (aes). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf (2001), updated: May 9 2023, Accessed: May 22 2026
8. Piret, G., Standaert, F.: Security analysis of higher-order boolean masking schemes for block ciphers (with conditions of perfect masking). IET Inf. Secur. **2**(1), 1–11 (2008)
9. Schramm, K., Paar, C.: Higher order masking of the AES. In: Pointcheval, D. (ed.) Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings. pp. 208–225. Lecture Notes in Computer Science, Springer (2006)