Chapter 12
Implementation and Improvement of the Partial Sum Attack on 6-Round AES

Francesco Aldà, Riccardo Aragona, Lorenzo Nicolodi and Massimiliano Sala

Abstract The Partial Sum Attack is one of the most powerful attacks, independent of the key schedule, developed in the last 15 years against reduced-round versions of AES. In this chapter, we introduce a slight improvement to the basic attack which lowers the number of chosen plaintexts needed to successfully mount it. Our version of the attack on 6-round AES can be carried out completely in practice, as we demonstrate providing a full implementation. We also detail the structure of our implementation, showing the performances we achieve.

12.1 Introduction

The research on the cryptanalysis of block ciphers partly deals with studying and proposing attacks on their reduced-round versions. Results on reduced versions are very interesting, since they help to better understand the behavior of a cipher, pointing out weaknesses in its structure which can eventually lead to attacks on the full version or characterize the security margin of the cipher.

In 2000, Ferguson et al. [5] introduced one of the most effective attacks, independent of the key schedule, developed in the last 15 years against reduced-round

F. Aldà (✉)
Horst Görtz Institute for IT Security and Faculty of Mathematics,
Ruhr-Universität Bochum, Universitätsstraße 150, 44801 Bochum, Germany
e-mail: francesco.alda@rub.de

R. Aragona · M. Sala
Department of Mathematics, University of Trento, Via Sommarive 14,
38123 Povo, Trento, TN, Italy
e-mail: riccardo.aragona@unitn.it

M. Sala
e-mail: maxsalacodes@gmail.com

L. Nicolodi
Independent Researcher, Lavis, TN, Italy
e-mail: lo@hidden-bits.com

versions of the Advanced Encryption Standard [3, 4], the *Partial Sum Attack*. Specifically, they developed attacks against AES reduced to 6, 7 and 8 rounds. The attack on 6-round is particularly powerful and its complexity is in the range which is referred to as practicable in the literature. It improves a previous attack which was first described in [3]. The latter is based on *integral cryptanalysis*, a general technique which is applicable to a large class of SPN block ciphers. This technique was originally designed by Lars Knudsen in the paper presenting the block cipher Square [2], as a specific attack against its byte-oriented structure. This is the reason why this class of attacks is commonly known as *Square Attack*. Since AES inherits many properties from Square, this attack can be easily extended to reduced-round versions of the Advanced Encryption Standard.

In this chapter, we introduce a slight theoretical improvement to the Partial Sum Attack on 6-round AES which lowers the number of chosen plaintexts needed to successfully mount it, and we describe the structure of our full implementation. After examining the literature which was developed after the publication of [5], we are not aware of any effective implementation of this attack. Therefore, we strongly believe that our implementation is the very first and, mostly, we show that it is completely practicable. Moreover, we believe that our effort allows a deeper understanding of the attack workflow and can point out some other weaknesses neither discovered nor exploited so far.

We would like to underline that a remark similar to the observation which our improvement is based on can be found in [12], although we achieved this result independently. Nevertheless, we believe that our analysis is more careful and detailed. In fact, the hypotheses which lead to this theoretical result are inherently strong, since they require the reduced-round cipher to "behave" like a random permutation. However, the attack we are dealing with strongly exploits the fact that AES can be easily distinguished from a random permutation. Therefore, it was not clear *a priori* whether these properties, or a good approximation of them, were actually satisfied in a real scenario. Thanks to our implementation which exploits the aforementioned improvement, we investigated these assumptions and explored how well the theoretical model describes an actual execution of the attack. In particular, the experimental results show that the number of false positives obtained closely matches that which was expected from the theoretical analysis. For a detailed explanation, we refer to Sect. 12.3.2.

The rest of the chapter is organized as follows: in Sect. 12.2, we briefly introduce the Square Attack and its extensions and we subsequently describe the Partial Sum Attack in detail. In Sect. 12.3, we present our main results. First, we explain our slight theoretical improvement, pointing out the issues that its implementation involves. We then detail our implementation and provide the results of our computations. In particular, we achieved to recover a full 6-round key in less than 12 days with 25 cores.

12.2 Preliminaries

We recall that the encryption process of AES-128, -192 and -256 consists of an initial key addition followed by the application of 10, 12 and 14 round transformations, respectively. The initial key addition and every round transformation take as input an intermediate result, called the *state*, and a round key which is derived from the cipher key through the key schedule. The state is always treated as a 4×4 matrix whose coefficients belong to $\mathbb{F}_{2^8}$. The output of any round is another state. The round transformation is a sequence of four processing steps: *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*. The *SubBytes* (*SB*) step is the only non-linear transformation of the cipher. It is an invertible byte substitution that operates independently on each byte of the state, according to an S-box. The S-box, which is henceforth indicated as γ, consists of the multiplicative patched inversion over $\mathbb{F}_{2^8}$, followed by an invertible affine transformation. The *ShiftRows* (*SR*) step is a byte transposition that cyclically shifts the rows of the state over different offsets. Specifically, let $s_{i,j}$ and $s'_{i,j}$ be the state bytes in position (i, j) before and after the *ShiftRows* transformation, respectively. Then $s'_{i,j} = s_{i,(j+i) \bmod 4}$ for $i, j \in \{0, 1, 2, 3\}$. The *MixColumns* (*MC*) step is a linear transformation which operates on the state column-by-column, treating each column as a polynomial over $\mathbb{F}_{2^8}[x]$. This polynomial is then multiplied modulo $x^4 + 1$ with the fixed polynomial $m(x) = (\alpha + 1)x^3 + x^2 + x + \alpha$, where $\alpha \in \mathbb{F}_{2^8}$ is such that $\alpha^8 = \alpha^4 + \alpha^3 + \alpha + 1$. Finally, in the *AddRoundKey* (*ARK*) transformation, the state is bitwise XORed with the corresponding round key. By *SubBytes*$^{-1}$, *ShiftRows*$^{-1}$, *MixColumns*$^{-1}$ and *AddRoundKey*$^{-1}$, we denote the inverses of the aforementioned steps. The final round differs from the others since the *MixColumns* step is removed. For further details on the structure of AES, we refer to [3, 4].

In the following sections, we first give an overview on the Square Attack on 4-round AES and we briefly introduce its extensions. We then describe the Partial Sum Attack in detail.

12.2.1 Square Attack

The Square Attack is a chosen plaintext attack, which is independent of the specific choices of the S-box of the *SubBytes* function, the multiplication polynomial of the *MixColumns* transformation and the key schedule. For the sake of clarity, however, we will often refer to the specific parameters used in AES.

In order to explain how this attack can be performed, we first introduce the following definition.

Definition 12.1 A Δ-set is a set of 256 AES states that differ in one of the state bytes (called *active* byte) and are equal in the other state bytes (called *passive* bytes). In other words, if U is a Δ-set, for every $x, y \in U$ we have

$$\begin{cases} x_{i,j} \neq y_{i,j} & \text{if } (i,j) \text{ is active} \\ x_{i,j} = y_{i,j} & \text{if } (i,j) \text{ is passive} \end{cases}$$

where $i, j \in \{0, 1, 2, 3\}$.

As it is explained in [4], the Square Attack on 4-round AES is heavily based on the following property.

Proposition 12.1 *Let $b_{i,j}^{(l)}$ be the byte in position (i, j), $i, j \in \{0, 1, 2, 3\}$, of the lth state of a Δ-set after three rounds. Then*

$$\sum_{l=1}^{256} b_{i,j}^{(l)} = 0. \tag{12.1}$$

In other words, the states at the end of the third round are *balanced*, i.e. all bytes at the input of the fourth round sum to zero. Note that the initial key addition is implicitly assumed and not counted in the number of rounds.

Let us consider a 4-round reduced AES, in which the fourth round is a final round, i.e. it does not include *MixColumns*. This implies that every byte of the ciphertext only depends on one byte of the input of the fourth round. The Square Attack on 4-round AES can then be mounted as follows. For any lth state of a Δ-set, $1 \leq l \leq 256$, let $c_{i,j}^{(l)}$, where $i, j \in \{0, 1, 2, 3\}$, be the ciphertext byte in position (i, j). Let $k_{i,j}^{(4)}$ be a guess for the byte in position (i, j) of the 4th round key (which is the last key used). For any (i, j), if the value of $k_{i,j}^{(4)}$ is correct, the following equation holds:

$$\sum_{l=1}^{256} \gamma^{-1}\left(c_{i,j}^{(l)} + k_{i,j}^{(4)}\right) = \sum_{l=1}^{256} b_{i,(j+i) \bmod 4}^{(l)} = 0, \tag{12.2}$$

where $b_{i,j}^{(l)}$ is the byte in position (i, j) of the lth state of a Δ-set after the application of three rounds, and γ^{-1} is the S-box of *SubBytes*$^{-1}$.

If Eq. (12.2) does not hold, the assumed value for the key byte must be wrong. This check is expected to eliminate all wrong key bytes, except for one value that could satisfy (12.2) by chance. To be more precise, the following result holds.

Proposition 12.2 *If $(X^{(l)})_{1 \leq l \leq 256}$ is a sequence of independent uniformly distributed random variables with values in $\mathbb{F}_{2^8}$, then the probability*

$$\mathbb{P}\left[\sum_{l=1}^{256} X^{(l)} = 0\right] = 2^{-8}.$$

Proof Let X and Y be two discrete independent random variables, with density functions $f_1(x)$ and $f_2(x)$ respectively. The convolution $f_3(x) = [f_1 * f_2](x) = \sum_y f_1(y) f_2(x - y)$ is the density function of the random variable $Z = X + Y$. Since X

Table 12.1 Estimated probability to obtain a zero sum for a random set of plaintexts and for a Δ-set at the end of the 3rd round

Random set	Δ-set
0.003904	0.007794

Number of trials: $2 \cdot 10^4$

and Y take values in $\mathbb{F}_{2^8}$, their sum Z takes values in $\mathbb{F}_{2^8}$ too. Therefore, the density function of Z is an uniformly distributed random variable, since it is the circular convolution of two independent uniformly distributed random variables. This result can be easily extended to the sum of an arbitrary number of random variables.   □

Before proceeding with the analysis of the attack, we would like to stress that the hypotheses of Proposition 12.2 are inherently strong. In particular, the bytes of the state at the end of the 3rd round are assumed to be independent and uniformly distributed. Although these are natural assumptions for modeling the attack, it is not clear a priori whether they hold even in practice. We thus performed some tests which aimed to estimate the probability to obtain a zero sum for a random set of 256 plaintexts and for a Δ-set at the end of the 3rd round. The values reported in Table 12.1 were obtained by averaging the estimates we collected using $2 \cdot 10^4$ random sets and $2 \cdot 10^4$ different Δ-sets, encrypted through an equal number of random keys, respectively.

As Table 12.1 shows, the tests we performed give evidence that Proposition 12.2 well describes the behavior of the cipher even at the end of the 3rd round. As expected, for a random set of 256 plaintexts there exists (on average) only one value which satisfies Eq. (12.2) by chance. In the case of a Δ-set, the estimate is roughly 1/128, since both the correct key byte and another random value satisfy (12.2).

Since checking Eq. (12.2) for a single Δ-set is expected to leave only 1 over 256 of the wrong key assumptions as a possible candidate, the 4th round key can be found with a sufficiently large confidence using two different Δ-sets. Henceforth, this crosscheck will be referred to as *verification step*.

All in all, two Δ-sets have to be used, and all 16 bytes of the 4th round key need to be recovered. Therefore, the working factor consists of 2^9 encryptions and $2^9 \cdot 2^4 = 2^{13}$ evaluations of Eq. (12.2).

In [4], Daemen et al. describe how this attack can be extended adding one round at the end or one round at the beginning. Combining the basic attack on 4 rounds with both extensions yields the Square Attack on 6-round AES. We can sketch this attack as follows. For the extension by one round at the end, the attacker has to perform a partial decryption of two rounds instead of only one, implying that four more bytes of the final round key need to be guessed. The idea for the extension by one round at the beginning consists of choosing a set of 256 plaintexts which, at the end of the first round, results in a Δ-set with a single active byte. This requires to guess four bytes of the initial round key $k^{(0)}$. We refer to [4] for further details on these two extensions. In both cases, we need to guess five key bytes instead of one. By combining these two methods, nine bytes need to be guessed.

12.2.2 Partial Sum Attack

Without considering the verification steps, the Square Attack on 6-round AES requires the storage of 2^{32} chosen plaintexts and the corresponding ciphertexts. Moreover, $(2^8)^9 = 2^{72}$ steps are needed for guessing nine key bytes, when it is applied to only recover 4 bytes of the 6th round key. Therefore, it is completely out of reach for current computing resources.

The Partial Sum Attack [5] significantly improves the Square Attack on 6-round AES. Ferguson et al. introduced two main ideas. First, instead of guessing four bytes of the initial round key $k^{(0)}$, one can use 2^{32} plaintexts such that one column of the states at the input of *MixColumns* of the first round ranges over all possible values of $(\mathbb{F}_{2^8})^4$ and all other bytes are constant. Throughout the rest of the chapter, we denote by $\bar{\Delta}$-set such a group of 2^{32} plaintexts. For any value of the initial round key, the corresponding ciphertexts consist of 2^{24} groups of 2^8 encryptions that vary in a single active byte at the end of the first round. In fact, imposing a particular linear combination which ranges over all possible values of $\mathbb{F}_{2^8}$ and three other linear combinations which are constant for all 256 states, we can uniquely determine a set of plaintexts which results in a Δ-set with a single active byte at the end of the first round. In particular, one has 2^{24} ways to choose the values for these three linear combinations.

Therefore, all an attacker has to do is guess four bytes of the 6th round key and one byte of the 5th round key, perform a partial decryption to a single state byte at the end of the 4th round, sum this value over all 2^{32} encryptions, and check whether the result is zero. Compared to the Square Attack on 6 rounds, the attacker needs to guess 40 bits instead of 72.

The further idea behind the improvement introduced by Ferguson et al. consists in organizing the partial decryption on *partial sums*. In order to properly understand what *partial sums* are and how one can use them, we introduce the following notation, where the pair (i, j) is used to denote the state entry (with $i, j \in \{0, 1, 2, 3\}$), and the index l (with $1 \le l \le 2^{32}$) denotes the lth element of a $\bar{\Delta}$-set:

$b_{i,j}^{(l)}$ is a byte at the end of the 4th round;

$a_{i,j}^{(l)}$ is a byte of the state at the 5th round before the application of *MixColumns*;

$a_s^{(l)}$ is the sth column of the lth state at the 5th round before the application of *MixColumns*. Thus $a_j^{(l)} = \left(a_{0,j}^{(l)}, a_{1,j}^{(l)}, a_{2,j}^{(l)}, a_{3,j}^{(l)} \right)^{\top}$;

$c_{i,j}^{(l)}$ is a byte at the end of the 6th round, which we refer to as the ciphertext byte;

$k^{(h)}$ is the hth round key and $\bar{k}^{(h)} = MixColumns^{-1}(k^{(h)})$;

$\bar{k}_{i,j}^{(h)}$ is a byte of $\bar{k}^{(h)}$.

It is easy to show that, in order to compute the partial decryption to a state byte at the end of the 4th round, we need to consider four bytes in each ciphertext and guess the corresponding bytes of the 6th round key, according to one of the configurations shown in Fig. 12.1. Observe that each configuration has exactly one byte per state row and one byte per state column.

1 st set config.   2nd set config.   3rd set config.   4th set config.
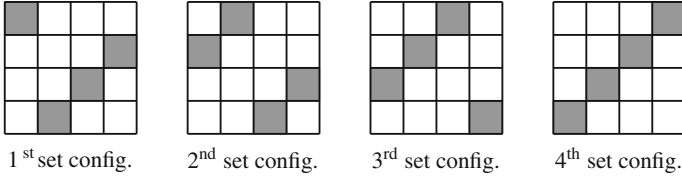
Fig. 12.1 The set of 4 bytes of the 6th round key (resp. ciphertexts) for the Partial Sum Attack on 6-round AES

In the following computations, with abuse of notation, we denote by *Mix-Columns*$^{-1}$ and *SubBytes*$^{-1}$ the inverse of *MixColumns* and *SubBytes* applied to a single column of the state. The relations between the $a^{(l)}$'s, the $c^{(l)}$'s and the $k^{(h)}$'s are easily established:

$$
a_j^{(l)} = \begin{bmatrix} a_{0,j}^{(l)} \\ a_{1,j}^{(l)} \\ a_{2,j}^{(l)} \\ a_{3,j}^{(l)} \end{bmatrix} = MixColumns^{-1} \left(SubBytes^{-1} \begin{pmatrix} c_{0,j}^{(l)} + k_{0,j}^{(6)} \\ c_{1,(j-1)\bmod 4}^{(l)} + k_{1,(j-1)\bmod 4}^{(6)} \\ c_{2,(j-2)\bmod 4}^{(l)} + k_{2,(j-2)\bmod 4}^{(6)} \\ c_{3,(j-3)\bmod 4}^{(l)} + k_{3,(j-3)\bmod 4}^{(6)} \end{pmatrix} \right),
$$

where $j \in \{0, 1, 2, 3\}$. When j is understood, we will remove it; for example we denote

$$
\xi^{(l)} = \begin{bmatrix} \xi_0^{(l)} \\ \xi_1^{(l)} \\ \xi_2^{(l)} \\ \xi_3^{(l)} \end{bmatrix} := SubBytes^{-1} \begin{pmatrix} c_{0,j}^{(l)} + k_{0,j}^{(6)} \\ c_{1,(j-1)\bmod 4}^{(l)} + k_{1,(j-1)\bmod 4}^{(6)} \\ c_{2,(j-2)\bmod 4}^{(l)} + k_{2,(j-2)\bmod 4}^{(6)} \\ c_{3,(j-3)\bmod 4}^{(l)} + k_{3,(j-3)\bmod 4}^{(6)} \end{pmatrix},
$$

for $1 \leq l \leq 2^{32}$. Let N be the byte matrix of *MixColumns*$^{-1}$. Working out the product, we have

$$
a_j^{(l)} = \begin{bmatrix} N_0 \cdot \xi_0^{(l)} + N_1 \cdot \xi_1^{(l)} + N_2 \cdot \xi_2^{(l)} + N_3 \cdot \xi_3^{(l)} \\ N_3 \cdot \xi_0^{(l)} + N_0 \cdot \xi_1^{(l)} + N_1 \cdot \xi_2^{(l)} + N_2 \cdot \xi_3^{(l)} \\ N_2 \cdot \xi_0^{(l)} + N_3 \cdot \xi_1^{(l)} + N_0 \cdot \xi_2^{(l)} + N_1 \cdot \xi_3^{(l)} \\ N_1 \cdot \xi_0^{(l)} + N_2 \cdot \xi_1^{(l)} + N_3 \cdot \xi_2^{(l)} + N_0 \cdot \xi_3^{(l)} \end{bmatrix},
$$

where, in the specific case of AES (see Sect. 12.2),

$$
N_0 = \alpha^3 + \alpha^2 + \alpha
$$
$$
N_1 = \alpha^3 + \alpha + 1
$$
$$
N_2 = \alpha^3 + \alpha^2 + 1
$$
$$
N_3 = \alpha^3 + 1.
$$

Thus we can compute a state byte at the end of the 4th round as follows:

$$b_{i,(j+i)\bmod 4}^{(l)} = \gamma^{-1}\left(a_{i,j}^{(l)} + \bar{k}_{i,j}^{(5)}\right), \tag{12.3}$$

where $i \in \{0, 1, 2, 3\}$ and γ^{-1} is the S-box of $SubBytes^{-1}$, as usual. Observe that in (12.3) γ^{-1} is applied to $a_{i,j}^{(l)} + \bar{k}_{i,j}^{(5)}$ rather than to $a_{i,j}^{(l)} + k_{i,j}^{(5)}$. The latter would be wrong, since $k_{i,j}^{(5)}$ is added *after* the application of *MixColumns*.

In order to identify a possible right guess, we have to check if $\sum_{l=1}^{2^{32}} b_{i,(j+i)\bmod 4}^{(l)} = 0$. This sum can be expressed as

$$\sum_{l=1}^{2^{32}} \gamma^{-1}\left(N_{-i} \cdot \xi_0^{(l)} + N_{1-i} \cdot \xi_1^{(l)} + N_{2-i} \cdot \xi_2^{(l)} + N_{3-i} \cdot \xi_3^{(l)} + \bar{k}_{i,j}^{(5)}\right), \tag{12.4}$$

where the indices $-i, 1-i, 2-i, 3-i$ are all meant to be reduced modulo 4, giving a remainder in $\{0, 1, 2, 3\}$.

If we trivially execute this summation, given 2^{32} ciphertexts and 2^{40} possible key guesses, we have to sum 2^{72} different values, which does not significantly improve the basic Square Attack. As it is pointed out in [5], Expression (12.4) can be organized in a more efficient manner. Once the row i is fixed, for each $t \in \{0, 1, 2, 3\}$, it is possible to associate a partial sum $x_t^{(l)}$ to each set $\{\xi_0^{(l)}, \ldots, \xi_t^{(l)}\}$, defined as follows:

$$x_t^{(l)} := \sum_{z=0}^{t} N_{z-i} \cdot \xi_z^{(l)}.$$

In particular,
$$x_2^{(l)} = x_1^{(l)} + N_{2-i}\xi_2^{(l)} \text{ and } x_3^{(l)} = x_2^{(l)} + N_{3-i}\xi_3^{(l)}.$$

In order to simplify the notation, let $(c_0^{(l)}, c_1^{(l)}, c_2^{(l)}, c_3^{(l)})$ be the 4-tuple formed by the lth ciphertext's bytes, extracted according to one of the configurations described above. Guessing the key values and using the partial sums, we can define the following maps

$$(c_0^{(l)}, c_1^{(l)}, c_2^{(l)}, c_3^{(l)}) \longmapsto (x_1^{(l)}, c_2^{(l)}, c_3^{(l)}) \longmapsto (x_2^{(l)}, c_3^{(l)}) \longmapsto x_3^{(l)}.$$

Using a similar notation, let (k_0, k_1, k_2, k_3) be four values for the 6th round key, which we want to guess, arranged in the same configuration chosen for the ciphertexts, and let k_4 be a guess for the 5th round key byte $\bar{k}_{i,j}^{(5)}$. The Partial Sum Attack is organized as follows.

- We start with the list of 2^{32} 4-tuples $(c_0^{(l)}, c_1^{(l)}, c_2^{(l)}, c_3^{(l)})$. Guessing k_0 and k_1, we can compute each triple $(x_1^{(l)}, c_2^{(l)}, c_3^{(l)})$.
- We then guess k_2, and compute each pair $(x_2^{(l)}, c_3^{(l)})$.
- Similarly, we guess k_3, and compute each value of $x_3^{(l)}$.
- Finally, guessing the value of k_4, we can compute Expression (12.4) and check whether the result is zero.

12.2.3 Complexity

In the first phase one guesses 2 bytes and processes 2^{32} ciphertexts bytes. For each choice of k_0 and k_1, one more byte has to be guessed, but only 2^{24} triples have to be processed. In the third phase, k_3 has to be guessed but one has only to process 2^{16} pairs. This holds similarly for the other two phases. Summing up all the contributions, we obtain that 2^{50} operations are required for a single $\bar{\Delta}$-set of 2^{32} elements.

12.3 Implementation and Improvement

The results described in this work started from Aldà's Master's thesis [1], where he developed a C++ code of the Partial Sum Attack and introduced (independently of [12]) the improvement specified in Sect. 12.3.2.

12.3.1 High-Level Scheme of the Implementation

To the best of our knowledge, this is the very first implementation of the Partial Sum Attack on 6-round AES. In this section, we explain the main ideas and principles we used in our implementation. We refer to Sect. 12.3.3 for further technical details on our implementation.

As it is displayed in Fig. 12.2, the steps involved in the attack are very simple. At the beginning of the attack, a $\bar{\Delta}$-set with 2^{32} elements has to be encrypted. In this way, we can obtain and store the 4-tuples $(c_0^{(l)}, c_1^{(l)}, c_2^{(l)}, c_3^{(l)})$, formed by the lth ciphertext's bytes, extracted according to one of the configurations described in Sect. 12.2.2. Extending the idea introduced in [5], it is sufficient to count how often each 4-tuple appears during the computation. As there are only 2^{32} possible 4-tuples, we do not have to store all $(c_0^{(l)}, c_1^{(l)}, c_2^{(l)}, c_3^{(l)})$ values. Since Expression (12.4) has to be computed in a field of characteristic 2, it suffices to count modulo 2. In fact, only the summands which appear an odd number of times give a non-zero contribution. Hence, a single bit suffices for each count and it is possible to store our list of 4-tuples
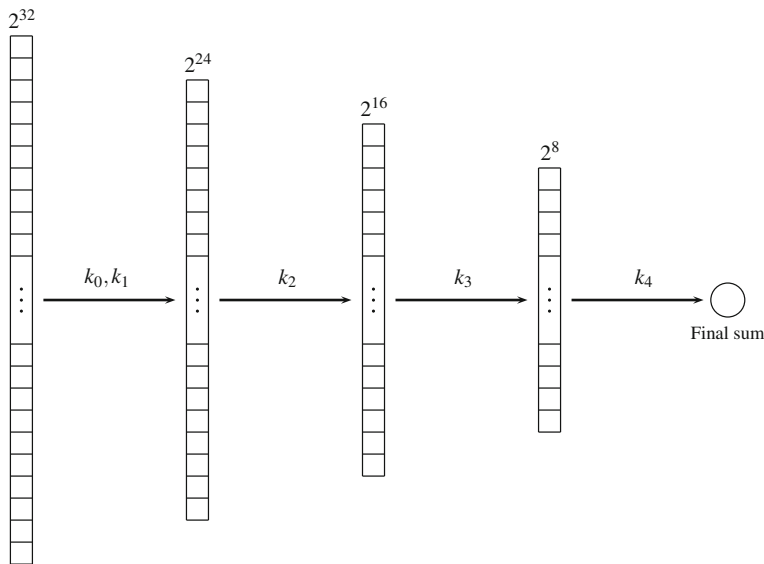
Fig. 12.2 The workflow of the Partial Sum Attack

in a 2^{32}-bit vector. Therefore, the space requirement for 2^{32} counters is just 2^{32} bits, which correspond to 0.5GB.

We then start a loop over 2^{16} possible values of k_0, k_1. For each pair (k_0, k_1), we compute the partial sums $x_1^{(l)}$ and store the triples $(x_1^{(l)}, c_2^{(l)}, c_3^{(l)})$. Using the same rationale, it suffices to count the parity of times each triple occurs. Again, we store all parities in a 2^{24}-bit vector. Moreover, we observe that, using an appropriate sorting, it suffices to compute the value $x_1^{(l)}$ every 2^{16} elements: in fact, this value only depends on $c_0^{(l)}, c_1^{(l)}, k_0$ and k_1. Thus, if $1 \leq l, h \leq 2^{32}$, we have

$$\begin{cases} c_0^{(l)} = c_0^{(h)} \\ c_1^{(l)} = c_1^{(h)} \end{cases} \implies x_1^{(l)} = x_1^{(h)}.$$

This observation significantly reduces the number of computations involved in this step, allowing entire blocks of bits to be updated at the price of very few calculations (see Sect. 12.3.3 for further details).

The same ideas can be similarly applied to the second step. For each value k_2, one computes the partial sums $x_2^{(l)}$, counts the parity of times each pair $(x_2^{(l)}, c_3^{(l)})$ occurs and stores it in a 2^{16}-bit vector. As before, it suffices to compute the value $x_2^{(l)}$ every 2^8 elements: in fact, this value only depends on $x_1^{(l)}, c_2^{(l)}$ and k_2.

In the third step, for each value k_3, we compute the partial sums $x_3^{(l)}$, count how many times each $x_3^{(l)}$ occurs and store its parity in a 2^8-bit vector. Unlike the previous

steps, this must be done scanning every entry of the 2^{16}-bit vector, since both $x_2^{(l)}$ and $c_3^{(l)}$ must be used in the computation of $x_3^{(l)}$. Finally, looping over the value k_4, it is possible to compute the final sum and check whether the result is zero.

As it was explained for the Square Attack, checking this sum for a single $\bar{\Delta}$-set is expected to eliminate 255 of the wrong key assumptions $(k_0, k_1, k_2, k_3, k_4)$. It is therefore necessary to verify their correctness using different $\bar{\Delta}$-sets (*verification steps*). At each positive verification, the key space is reduced by a factor 2^{-8}. Apparently, this implies that 6 different $\bar{\Delta}$-sets (or more) are needed to find the correct 5-tuple $(k_0, k_1, k_2, k_3, k_4)$ with overwhelming probability. This result can be improved, as it is explained in the following section.

12.3.2 Improvement

As it was already underlined, when it was published, the Partial Sum Attack represented one of the best cryptanalytic results on reduced-round versions of the Advanced Encryption Standard. After its publication, many other researchers worked on the integral cryptanalysis of Rijndael (and its specification AES), finding new extensions or improvements for this class of attacks (see for example [7, 9, 12]). Our approach started from performing a full implementation of the attack as it is described in Sect. 12.3.1, trying to understand where some other potentialities could be exploited.

In the original paper [5], it is claimed that at least 6 sets of 2^{32} plaintexts, which form a $\bar{\Delta}$-set, are necessary in order to find the correct 5-tuple $(k_0, k_1, k_2, k_3, k_4)$. However, we observed that only two $\bar{\Delta}$-sets suffices to determine the correct 4-tuple (k_0, k_1, k_2, k_3) with high probability. In fact, fixing one configuration according to which the ciphertexts bytes $(c_0^{(l)}, c_1^{(l)}, c_2^{(l)}, c_3^{(l)})$ are extracted, one can compute the sum in four different state bytes at the end of the 4th round (we can choose $i \in \{0, 1, 2, 3\}$ in Eq. (12.3)). We provide a visual example in Fig. 12.3.

If we consider each sum as independent and make use of Proposition 12.2, using only two $\bar{\Delta}$-sets, the probability that for a 4-tuple (k_0, k_1, k_2, k_3) there exists for each row a value k_4 which gives a zero sum for both $\bar{\Delta}$-sets is $(1/256)^8$. Note that the bytes of the 5th round key, which produce zero sums, may be different for each row, but, as for (k_0, k_1, k_2, k_3), their correctness should follow by the crosschecking between the two $\bar{\Delta}$-sets. Therefore, checking the value of the sum on four rows at the end of the 4th round is expected to determine, with sufficiently high confidence, the correct 4-tuple (k_0, k_1, k_2, k_3). More specifically, only one false positive (k_0, k_1, k_2, k_3) is expected to survive to all verification steps.

The hypothesis which this result is mainly based on consists of considering the sums on four rows as independent. As pointed out in Sect. 12.2.1, there is no certainty that this assumption holds perfectly in practice. Intuitively, even though the bytes involved in the sums belong to the same state and their correlation is hence nonzero, the diffusion and confusion introduced by the round transformations should make
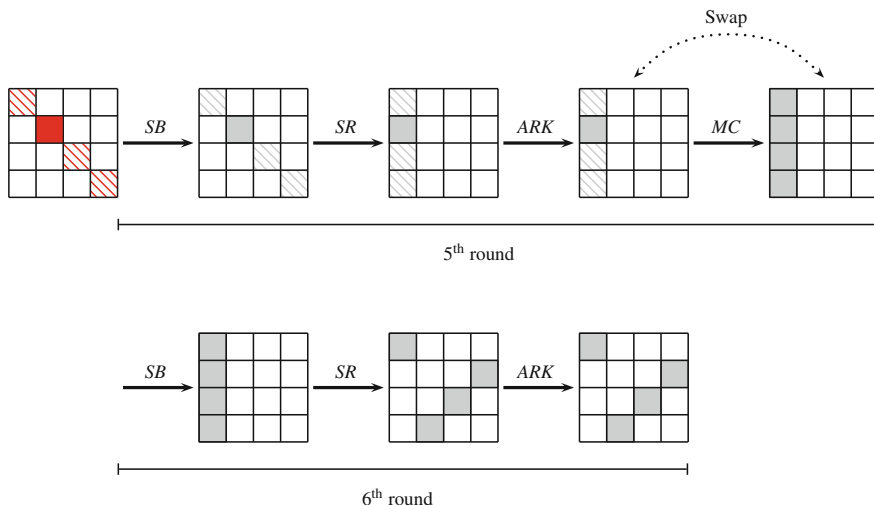
Fig. 12.3 The state bytes at the end of the 4th round (in the *top-left matrix*) which can be computed for a configuration according to which the ciphertexts bytes $(c_0^{(l)}, c_1^{(l)}, c_2^{(l)}, c_3^{(l)})$, for $1 \le l \le 2^{32}$, are extracted

it negligible after few rounds. The experimental results we performed using our implementation (which exploits the aforementioned improvement) show that using only two $\bar{\Delta}$-sets and computing the sum on four rows do not eliminate all wrong guesses, as we expected. In particular, besides the correct 4-tuple, we obtained on average one false positive, independently of the configuration chosen. Although more tests are needed in order to provide a better estimate, our results already indicate that the probability of false positive closely matches the expected one. Moreover, we believe that future analyses in this direction could point out further properties of the cipher, which may lead to other improvements of the attack.

All in all, we observed that the number of chosen plaintexts which are necessary in order to mount the attack (with high confidence) can be reduced from 6 $\bar{\Delta}$-sets of 2^{32} elements to only 2. In order to lower the probability of false positives (but still enhancing the basic attack described in [5]), we also performed some attacks using 3 $\bar{\Delta}$-sets, checking the sum on four rows at the end of the 4th round. As expected, in this setting we did not observe any false positive.

Although we reached this conclusion independently, we would like to point out that a remark similar to our observation can be found in [12]. As already observed, we believe that our analysis is more careful and detailed, since we supported the applicability of the hypotheses which this result is based on by means of experimental analyses on AES. Specifically, we provided a full implementation which strongly exploits the aforementioned improvement, and the results we obtained running the attack showed that the number of false positives closely matches the one which was expected from the theoretical analysis.

Among other speed-ups we introduced, this improvement allowed us to achieve optimal performances, showing the complete practicability of the attack, as it will be presented in the following section.

12.3.3 Implementation's Details

First of all, we ported Aldà's code [1] to C, to reduce the overhead of C++ abstractions, which are useful but not essential for this kind of application. During this phase, we decided to map every Boolean vector's element to a bit inside an `unsigned char`'s array. On one hand, this process forced us to create some ancillary functions to toggle and mask bits as necessary but, on the other hand, it had the side effect of accelerating some functions where shifting and masking were required, because we did it byte by byte, instead of bit by bit. Moreover, it allowed us to save space and time while writing and reading the encrypted arrays to and from the disk, storing every 2^{32} array in a 512 MB file and saving time while testing the attack. After completing the porting and introducing the new memory management concepts, we started focusing on how the memory management operations could be accelerated and we ended up managing every group of 8 `unsigned char` array elements as an `unsigned long long int` array, where possible. This allowed us to deal with the allocated memory as a set of 64 bit blocks, reducing the time needed to complete, for example, some `XOR` operations between these arrays. The resulting implementation was satisfactory.

We also decided to allow the parallelization of the attack on multiple core systems and, for this purpose, we needed to exchange information between each process. We chose `OpenMPI` [6, 8] because we appreciated its documentation and the maturity of the open source project supporting it. Porting the code from a linear to a parallel paradigm presented no real difficulties because the attack is mainly composed by loops, repeated for values from 0 to 255, so we decided to execute the 5 most inner loops on each worker (a worker is a parallel process running the attack), assigning to each of them a range of values of k_0 to go through in the outer of these 5 loops. Moreover, we shared the encrypted vectors, using `NFSv4`, on every system running the attack and using the same share storage to save the guessed partial keys and to check the status of the attack from each worker.

Our code works as follows. The master process coordinating the attack distributes the values to each worker using the `Round-Robin algorithm` [11] and then waits for replies from each of them. After finishing the attack with one of the assigned values, every worker reports the result to the master, if successful. If the attack with that value was not successful, the worker checks the shared storage looking if the current partial key has been guessed and, if so, it stops the attack, otherwise it starts the attack with the next assigned value.

To retrieve the whole 16-byte key, the attack has to be run 4 times, according to the four configurations shown in Fig. 12.1. The master writes 4 files that contain the

Table 12.2 Experimental results obtained running our implementation of the Partial Sum Attack on 6-round AES

Number of $\bar{\Delta}$-sets	Average time (days)	Memory (GB)
2	12.1	1.028
3	11.5	1.542

The keys were chosen according to the example vectors provided in [10]

partial keys guessed, and it also writes the whole key in another file when the attack is completed for every configuration.

The final outcome of this effort was interesting in terms of memory and time used. The attacks have been launched on 6 desktop PC, with 4 cores (Intel Pentium CPU G640 @2.80 GHz) and 8 GB of RAM each, using 25 processes. The first process coordinated the attacks, while the remaining 24 workers actually performed the attacks. The results we obtained are summarized in Table 12.2.

From these experimental results, we can note that the attacks which use 3 $\bar{\Delta}$-sets are generally slightly faster, though they obviously require more memory to be performed. This is not too surprising, since using only 2 $\bar{\Delta}$-sets triggers more verification steps on different rows (as observed in Sect. 12.3.2, there are more wrong key candidates which give a zero sum on a fixed row), which are time consuming operations in our current implementation.

Based on the results of Table 12.2, we estimate that, on average, the 128-bit 6th round key can be retrieved in 25.8 h using 256 workers.

The source code of our implementation of the Partial Sum Attack is available on http://tdsoc.org.

References

1. Aldà F (2013) The Partial Sum Attack on 6-round reduced AES: implementation and improvement. Master's thesis (laurea magistrale), University of Trento, Department of Mathematics
2. Daemen J, Knudsen L, Rijmen V (1997) The block cipher SQUARE. Fast software encryption. Springer, Heidelberg, pp 149–165
3. Daemen J, Rijmen V (1998) AES proposal: Rijndael. In: First advanced encryption standard (AES) conference
4. Daemen J, Rijmen V (2002) The design of Rijndael. In: AES—the advanced encryption standard. Information security and cryptography. Springer, Berlin
5. Ferguson N, Kelsey J, Lucks S, Schneier B, Stay M, Wagner D, Whiting D (2001) Improved cryptanalysis of Rijndael. In: Fast software encryption. Springer, Berlin, pp 213–230
6. Gabriel E, Fagg GE, Bosilca G, Angskun T, Dongarra JJ, Squyres JM, Sahay V, Kambadur P, Barrett B, Lumsdaine A, Castain RH, Daniel DJ, Graham RL, Woodall TS (2004) Open MPI: goals, concept, and design of a next generation MPI implementation. LNCS, vol 3241. Springer, Heidelberg, pp 97–104

7. Galice S, Minier M (2008) Improving integral attacks gainst Rijndael-256 up to 9 rounds. In: Progress in cryptology—AFRICACRYPT 2008. Springer, Heidelberg, pp 1–15
8. Graham RL, Woodall TS, Squyres JM (2006) Open MPI: a flexible high performance MPI. LNCS, vol 3911. Springer, Heidelberg, pp 228–239
9. Li YJ, Wu WL (2011) Improved integral attacks on Rijndael. J Inf Sci Eng 27(6):2031–2045
10. Pub NF (2001) 197: advanced encryption standard (AES), vol 197, pp 441–0311
11. Silberschatz A, Galvin PB, Gagne G (2008) Operating system concepts. Wiley, New York
12. Tunstall M (2012) Improved "partial sums"-based square attack on AES. In: International conference on security and cryptography—SECRYPT 2012. INSTICC Press, pp 25–34