Ease of Side-Channel Attacks on AES-192/256 by Targeting Extreme Keys

Antoine Wurcker

eshard, France, antoine.wurcker@eshard.com

Abstract. Concerning the side-channel attacks on Advanced Encryption Standard, it seems that majority of studies focus on the lowest size: AES-128. Even when adaptable to higher sizes (AES-192 and AES-256), lots of state-of-the-art attacks see their complexity substantially raised. Indeed, it often requires to perform two consecutive dependent attacks. The first is similar to the one applied on AES-128, but a part of the key remains unknown and must be retrieved through a second attack directly dependent on the success of the first.
This configuration may substantially raise the complexity for the attacker, especially if new signal acquisitions with specific input, built using the first key part recovered, must be performed. Any error/uncertainty in the first attack raise the key recovery complexity.
Our contribution is to show that this complexity can be lowered to two independent attacks by the mean of attacking separately first and last round keys. We show that the information is enough to recover the main key (or a very small list of candidates) in a negligible exploratory effort.

Keywords: Advanced Encryption Standard, AES, AES-192, AES-256, key recovery, Side-Channel Analysis, SCA

1 Introduction

Side-channels research field concerns the malicious usage of an involuntary leaked information during the execution of an algorithm, with the objective to retrieve a secret such as a cryptographic key. This was first introduced in [Koc96], where the processing time of operations, that was secret-dependent, was revealing the secret information. Once the side-channels potential discovered, numerous channels where used to extract secret information, such as: power consumption [KJJ98], electromagnetic emissions [GMO01], acoustic emissions [GST14] (extension of preliminary work of 2004), light emission [FH08]. The subject is wide, as in 2014 was shown a new leak source can be used: the ground of a laptop could leak sensitive information along cables (USB, Ethernet, ...) in [GPT14].

Side-channels might not leak directly the secret itself but be an information related to the secret, e.g. leakage model linear with the Hamming weight[1] of the processed data or with the Hamming distance[2] between two successive states are commonly considered. In order to exploit those kind of leakages, statistical methods where developed that can reveal the secret information from the leakage, such as: Differential Power Analysis (DPA) [KJJ99], Correlation Power Analysis (CPA) [BCO04], Mutual Information Analysis (MIA) [GBTP08], `SCATTER` [TGWC18]. All consist in using the variations of a known data (e.g. plaintext, ciphertext) to recover a constant secret data (e.g. key). The guesses of candidates for all, or a part of, the key are ranked from the most fitting to the leakages to the least one. Errors might occur in this ranking, leading to partial recoveries of the targeted secret as, sometimes, the good candidate might not be judged the most fitting one. Those errors might be due to various causes such as: countermeasures, noise on the leakage acquisition, error in leakage model estimation.

Due to such errors, an exploration in probable solutions might be necessary. The guessing entropy represents the number of bit of information that is missing to attacker at the end of an attack. Using the side-channel attacks results, one can build an estimate ranking of the full secret candidates and test each candidate against a plaintext/ciphertext pair to confirm/infirm its validity (under some computational limit in depth). The determination of an efficient order of candidates to be tested are explored in [VGRS12,PSG16]. We speak about guessing *depth* of an attack to indicate the complexity to reach the good candidate. This depth can be underestimated at 2 to the guessing entropy.

Some attacks might not be concerned by our method, such as blind attacks originally introduced in [LDL14] and further in [lB14,CR17,CRW18] that do not need to know the round inputs to attack intermediate data within, even in case of some masking countermeasures. This methodology giving the attacker the ability to recover any round keys independently.

AES, for Advanced Encryption Standard [Nat01], is the NIST[3] symmetric cipher standard. Established in 2001 after a contest to find a successor to the DES (Data Encryption Standard), being the previous standard established in 1977 [Nat77].

[1] Number of bits to "1" in a binary word.
[2] Number of bits that differ between two binary words.
[3] U.S. National Institute of Standards and Technology

In this paper we focus on the key scheduling part of the algorithm, that consists in transforming the 128-bit main key (respectively 192-bit and 256-bit) into 11 (respectively 13 and 15) 128-bit round keys. This configuration of key schedule implies that a side-channel attack targeting the first (respectively last) round is sufficient for AES-128 but only retrieves a part of the main key for higher sizes, thus requiring an additional attack, directly depends on the success of the first one, onto the second (respectively penultimate) round to complete the key recovery. In this paper we propose to solve this potential difficulty by applying two independent attacks onto extreme (first and last) rounds instead of two consecutive rounds. We describe how such information is enough for an attacker to retrieve the main key.

This paper is organized as follows. The Section 2 introduces the context of our attack with scenarios of application and the notations used in the paper. The Section 3 describes our contribution on how the knowledge of extreme round keys gives the ability to the attacker to retrieve the main key, or a reduced set of candidates, in a negligible exploration time for both AES-256 and AES-192. The Section 4 gives a conclusion to this paper and further work perspectives.

2 Context and Notations

As stated previously, the side-channel attacks might be imprecise, we measure this imprecision by the guessing entropy N. Meaning that the good key is then in a set of 2^N candidates. In the following we denote by N_1 the guessing entropy remaining after the first attack applied on one round key and N_2 the guessing entropy remaining after the second attack.

The methodology consist in confronting those candidates against a plaintext/ciphertext pair . This has a low incidence on AES-128 size but a great on higher sizes. Indeed this implies that the second attack might have to be replayed until reaching the good candidate, at the guessing depth of the first one. Depending of the cost of the second attack, this might become tedious or infeasible.

Our extreme key attack does not reduce the error level of the key recovery attacks, but it may i) reduce the candidate exploration cost or ii) make feasible attacks when consecutive rounds cannot be attacked.

One can note that our main key recovery using extreme keys method is trivial in DES algorithm. As defined in its specification in [Nat77], the key schedule description implies that 48-bit round keys are composed of unmodified bits from the 56-bit main keys. A first round key attack would reveal all main key bits except 8, those missing bits can all be found in the last round key. Then recovering first and last DES round keys gives the main key without need for specific exploration.

2.1 AES Key Schedule

The Figures 1, 2 and 3 show the different size key computation processes, where 4-byte columns W_i are represented with the computations that relate them.

The main key is used as first round key(s), then next round keys are derived using several transformations: RW stands for `RotWord` that will rotate the 4-byte column from one byte up, SW stands for `SubWord` that will apply the AES **SBox** look-up table on each of the 4 bytes independently and finally, RC stands for `RCON` that consists in applying a xor operation between the first byte of the column and a round dependent constant byte.

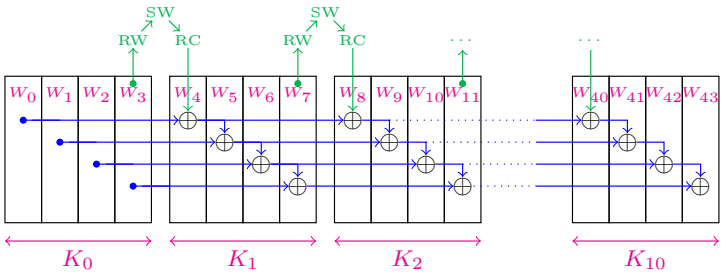

Fig. 1. AES-128 key schedule structure.

For the AES-128 (Figure 1), the main key fits in the first round key K_0. The next key (K_1) columns are computed as:

$W_4 = W_0 \oplus RC \circ SW \circ RW(W_3)$ $\quad$ $W_6 = W_2 \oplus W_5$

$W_5 = W_1 \oplus W_4$ $\quad$ $W_7 = W_3 \oplus W_6$

Each round key being computed using columns from the previous one in the same way. This bring a property of AES-128: an attacker recovering any round key is able to compute the main key from it.

For the AES-192 (Figure 2), the same procedure is applied but onto blocks of 6 columns instead of 4. One can remarks that the main key fills

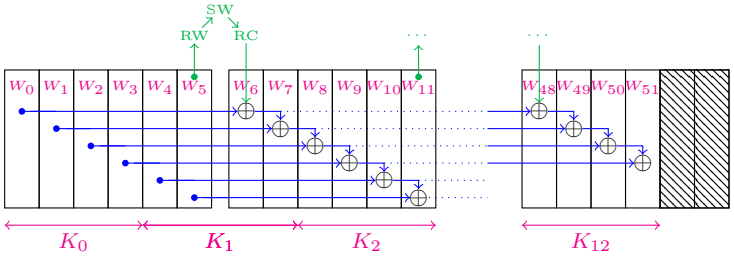

Fig. 2. AES-192 key schedule structure.

K_0 and the half of K_1, the other half being completed by the first two columns of next treated block. As only 13 keys are needed, the last two columns of the last block are not computed.

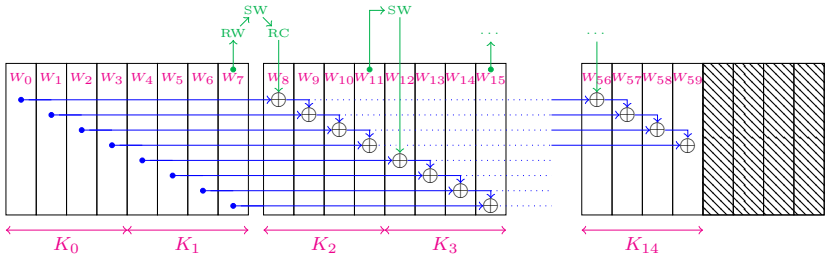

Fig. 3. AES-256 key schedule structure.

For the AES-256 (Figure 3), the same procedure is applied but onto blocks of 8 columns this time. One supplementary operation consists in applying of SW transformation operation onto fourth column before being used to compute the fifth. One can remark that the main key fills both K_0 and K_1. As only 15 keys are needed, the last four columns of the last block are not computed.

2.2 Attacker Scenarios

Here are given some scenarios as example of configurations where our attack can ease or enable an attack onto higher sizes of AES algorithm. In those scenarios we consider cases where several attacks must be done, each recovering a part of the full key. We show here that this unavoidable depth to reach can be very costly depending on the operations to perform

during its exploration, and then how our method can reduce its impact by lowering this cost.

Scenario 1: Small Ease of an Attack In this scenario, the attacker knows the plaintext and the ciphertext and the leakage is linear to the Hamming weight of intermediate data.

Without our method: The full key recovery consists in running an attack on first round key (guessing depth: D_1) and then use this gained knowledge to compute the input of second round and attack it the same way to recover the second round key (guessing depth: D_2).

This creates a dependence between the success of the first attack and the complexity of the second. In a worse case this became challenging as the attacker have to run as many times the second attack as the guessing depth of first one (D_1). Each one of $D_1 \times D_2$ full key proposed would be tested against a plaintext/ciphertext pair to be confirmed/infirmed.

With our method: The attacker can target independently the first round key and the last one. The remaining action is to run our key recovery process on the $D_1 \times D_2$ (first key, last key) pairs, the returned full key candidates would be tested against a plaintext/ciphertext pair to be confirmed/infirmed.

The main time cost without our method is to run $1 + D_1$ attacks on the whole trace set, when our method needs only two attacks on the trace set and then $D_1 \times D_2$ key candidate exhaust.

Scenario 2: Huge Ease of an Attack In this scenario, the attacker can choose the plaintext and knows the ciphertext and the leakage is linear to the Hamming distance between successive data, e.g. output of current round erase the register with previous round in it.

Without our method: The first step consists in running a chosen plaintext attack on first round key (guessing depth: D_1). This gained knowledge allows to forge a new set of plaintext inducing chosen second round inputs. This require a new trace acquisition phase. The new trace set is used to recover the second round key (guessing depth: D_2).

This time the dependence is much worse than in scenario 1. The attacker have to acquire a new trace set and run a new attack on it as many times as the guessing depth of first attack (D_1).

With our method: The attacker can apply independently, and on the same trace set, chosen plaintext attack to target first round key and known ciphertext to target the last round key. As in previous scenarios the remaining action is to run our key recovery process on the $D_1 \times D_2$ (first key,last key) pairs.

The main time cost without our method is to generate $1 + D_1$ trace sets and run $1 + D_1$ attacks, when our method needs only two attacks on a single trace set and the $D_1 \times D_2$ key exhaust.

Scenario 3: Enable an Attack In this scenario, the attacker knows the plaintext and ciphertext, and can observe encrypt or decrypt and the leakage is linear to the distance of Hamming between successive data.

Without our method: As stated in previous scenario the attack may recover only the last round key, that is enough in AES-128 but not for higher sizes. No easy attack path is known in this configuration. Attacker might start to wait for known plaintext to be in specific condition to build subset that got the same properties than chosen plaintext attack but its represent a huge cost.

With our method: The attacker build two independent trace sets, one with encryption, the second with decryption. Respectively last and first[4] round keys are recovered, targeting the last round execution of encryption and decryption sets.

Without our method the attack cannot be performed, while our method needs only two attacks on a two trace sets and the $D_1 \times D_2$ key candidate exhaust.

Remark 1. If a countermeasure against faults is implemented that compute $C = \text{cipher}(P)$ and then control that $P = \text{cipher}^{-1}(C)$ both encryption and decryption can be acquired in a row. This countermeasure then simplify the attack scenario as it requires only one acquisition set instead of two and the countermeasure relaxes the constraint to be able to run decipher on the device.

2.3 Notations

The notations used in this paper are defined as follows:

[4] The decryption uses the same keys as encryption but in the reverse order: the key used in last round of decryption is the first round key.

- $K_i, i \in [0, r]$ is the i^{th} 128-bit round key. $r = 10/12/14$ for AES-128/192/256.
- $W_i, i \in [0, n[$ is the i^{th} 4-byte column of the expanded key:
  - e.g. first round key K_0 is composed of $\{W_0, W_1, W_2, W_3\}$,
  - $n = 44/52/60$ for AES-128/192/256,
  - each W_i can be decomposed in 4 bytes $w_i^j, j \in [0, 3]$.
- We denote by $T_{a,b,c}$ The xor of several atomic elments sharing the same letter: $T_{a,b,c} = T_a \oplus T_b \oplus T_c$.
- $f_1()$ is the invertible column transformation function that apply AES non linear `SubWord` operation onto input column.
- $f_{2,r}()$ is the invertible column transformation function that apply `RotWord`, `SubWord` and `RCON` addition.
- $\mathcal{L}$ letters represent a set of candidates for a 32-bit word and L_i will be one of the candidates of the set $\mathcal{L}_i$.
- **SBox**[ ] is the AES byte table lookup.

When the equation needs the distinction, over-lined elements are unknown to the attacker. e.g. $\overline{X_{3,7}} = Y_0 \oplus Z_{5,7}$ means:

- $X_{3,7}$ is initially unknown, because X_3, or X_7 or both are unknown.
- Y_0 is known.
- $Z_{5,7}$ is known but it does not assume that Z_5 and Z_7 are known.
- $X_{3,7}$ became now known as being a combination of known values.

3 Our Contribution

3.1 Recover AES-256 Key

Attacker Model We consider an attacker that managed to recover the first round key (K_0) and last round key (K_{14}) of an AES-256 implementation. The main key is the concatenation of K_0 and K_1 so we explain here how to recover a short list of candidates for K_1 in a negligible computational effort using the knowledge of (K_0, K_{14}) pair.

A naive approach to solve this question would have been to use the brute force method: guess the 128-bits of K_1 and apply the key schedule on it to check if the last key corresponds to the expected value or not, but such a 2^{128} exhaust is considered infeasible in practice.

First Observations Table 1 is giving $W_i, i \in [0, 60[$ equations of the AES-256 key schedule process. First 8 columns are initialized with the 256-bit main key. Other values are computed as a combination of initial values $\{W_0, \ldots, W_7\}$. Those expressions require the introduction of

$P_i, i \in [0,5]$ and $T_i, i \in [0,6]$ columns, resulting from application of $f_1()$, respectively $f_{2,r}()$ transformation functions onto some W columns.

One can remark different behaviors between the round key of even or odd rounds. Indeed, the even ones are using a cyclic combination of values from set $\{W_0, \ldots, W_3\}$, use the T values and produce the P values, when the odd ones are using a cyclic combination of values from set $\{W_4, \ldots, W_7\}$, use the P values and produce the T values.

Table 1. AES-256 key schedule equations

K_0	$W_0 = W_0$		K_1	$W_4 = W_4$	
	$W_1 = W_1$			$W_5 = W_5$	
	$W_2 = W_2$			$W_6 = W_6$	
	$W_3 = W_3$			$W_7 = W_7$	$T_0 = f_{2,0}(W_7)$
K_2	$W_8 = W_0 \oplus T_0$		K_3	$W_{12} = W_4 \oplus P_0$	
	$W_9 = W_{0,1} \oplus T_0$			$W_{13} = W_{4,5} \oplus P_0$	
	$W_{10} = W_{0,1,2} \oplus T_0$			$W_{14} = W_{4,5,6} \oplus P_0$	
	$W_{11} = W_{0,1,2,3} \oplus T_0$	$P_0 = f_1(W_{11})$		$W_{15} = W_{4,5,6,7} \oplus P_0$	$T_1 = f_{2,1}(W_{15})$
K_4	$W_{16} = W_0 \oplus T_{0,1}$		K_5	$W_{20} = W_4 \oplus P_{0,1}$	
	$W_{17} = W_1 \oplus T_1$			$W_{21} = W_5 \oplus P_1$	
	$W_{18} = W_{0,2} \oplus T_{0,1}$			$W_{22} = W_{4,6} \oplus P_{0,1}$	
	$W_{19} = W_{1,3} \oplus T_1$	$P_1 = f_1(W_{19})$		$W_{23} = W_{5,7} \oplus P_1$	$T_2 = f_{2,2}(W_{23})$
K_6	$W_{24} = W_0 \oplus T_{0,1,2}$		K_7	$W_{28} = W_4 \oplus P_{0,1,2}$	
	$W_{25} = W_{0,1} \oplus T_{0,2}$			$W_{29} = W_{4,5} \oplus P_{0,2}$	
	$W_{26} = W_{1,2} \oplus T_{1,2}$			$W_{30} = W_{5,6} \oplus P_{1,2}$	
	$W_{27} = W_{2,3} \oplus T_2$	$P_2 = f_1(W_{27})$		$W_{31} = W_{6,7} \oplus P_2$	$T_3 = f_{2,3}(W_{31})$
K_8	$W_{32} = W_0 \oplus T_{0,1,2,3}$		K_9	$W_{36} = W_4 \oplus P_{0,1,2,3}$	
	$W_{33} = W_1 \oplus T_{1,3}$			$W_{37} = W_5 \oplus P_{1,3}$	
	$W_{34} = W_2 \oplus T_{2,3}$			$W_{38} = W_6 \oplus P_{2,3}$	
	$W_{35} = W_3 \oplus T_3$	$P_3 = f_1(W_{35})$		$W_{39} = W_7 \oplus P_3$	$T_4 = f_{2,4}(W_{39})$
K_{10}	$W_{40} = W_0 \oplus T_{0,1,2,3,4}$		K_{11}	$W_{44} = W_4 \oplus P_{0,1,2,3,4}$	
	$W_{41} = W_{0,1} \oplus T_{0,2,4}$			$W_{45} = W_{4,5} \oplus P_{0,2,4}$	
	$W_{42} = W_{0,1,2} \oplus T_{0,3,4}$			$W_{46} = W_{4,5,6} \oplus P_{0,3,4}$	
	$W_{43} = W_{0,1,2,3} \oplus T_{0,4}$	$P_4 = f_1(W_{43})$		$W_{47} = W_{4,5,6,7} \oplus P_{0,4}$	$T_5 = f_{2,5}(W_{47})$
K_{12}	$W_{48} = W_0 \oplus T_{0,1,2,3,4,5}$		K_{13}	$W_{52} = W_4 \oplus P_{0,1,2,3,4,5}$	
	$W_{49} = W_1 \oplus T_{1,3,5}$			$W_{53} = W_5 \oplus P_{1,3,5}$	
	$W_{50} = W_{0,2} \oplus T_{0,1,4,5}$			$W_{54} = W_{4,6} \oplus P_{0,1,4,5}$	
	$W_{51} = W_{1,3} \oplus T_{1,5}$	$P_5 = f_1(W_{51})$		$W_{55} = W_{5,7} \oplus P_{1,5}$	$T_6 = f_{2,6}(W_{55})$
K_{14}	$W_{56} = W_0 \oplus T_{0,1,2,3,4,5,6}$				
	$W_{57} = W_{0,1} \oplus T_{0,2,4,6}$				
	$W_{58} = W_{1,2} \oplus T_{1,2,5,6}$				
	$W_{59} = W_{2,3} \oplus T_{2,6}$				

Attack Step 1: Guess-Free Knowledge The knowledge of W_0, W_1, W_2, W_3, W_{56}, W_{57},W_{58} and W_{59} allows to use K_{14} equations and revert them to gain new effortless knowledge:

$$\overline{T_{0,1,2,3,4,5,6}} = W_0 \oplus W_{56} \qquad \overline{T_{1,2,5,6}} = W_1 \oplus W_2 \oplus W_{58}$$
$$\overline{T_{0,2,4,6}} = W_0 \oplus W_1 \oplus W_{57} \qquad \overline{T_{2,6}} = W_2 \oplus W_3 \oplus W_{59}$$

Now this 4 combinations of T values are known, we can combine them in order to reduce the number of inner elements. So we gained, without need to guess, one atomic element: T_3 and three composed: $T_{0,4}$, $T_{1,5}$ and $T_{2,6}$ from which we do not know the atomics yet.

The gained knowledge continues to give us guess-free elements: $\overline{W_{35}}$, $\overline{W_{42}}$, $\overline{W_{43}}$, $\overline{W_{49}}$, $\overline{W_{50}}$ and $\overline{W_{51}}$.

Some of those values are involved into the computation of P values that we can also learn:

$$\overline{P_3} = f_1(W_{35}) \qquad \overline{P_4} = f_1(W_{43}) \qquad \overline{P_5} = f_1(W_{51})$$

Attack Step 2: Identify Where to Guess Here we describe how to identify where to make a pertinent guess on a value in order to confront those candidates to constraints and then reduce the size of exploration tree. We observe that some composed T rely on the same combination of K_1 columns, e.g.:

$T_{0,4}$ is known and:

$$\overline{T_0} = f_{2,0}(\overline{W_7}) = f_{2,0}(\overline{W_7}) \quad (1)$$
$$\overline{T_4} = f_{2,4}(\overline{W_{39}}) = f_{2,4}(\overline{W_7} \oplus P_3) \quad (2)$$

The above equations can be confronted to their relative known data, for example a guess of $\overline{W_7}$ gives a value to $\overline{T_0}$ thanks to Equation 1 an gives a value to $\overline{T_4}$ thanks to Equation 2. We can invalidate all $\overline{W_7}$ values that do not lead to a pair ($\overline{T_0}$,$\overline{T_4}$) respecting the known $T_{0,4}$. An attacker is then able to build:

- a reduced list $\mathcal{L}_0$ of candidates for $\overline{W_7}$ thanks to $T_{0,4}$
- a reduced list $\mathcal{L}_1$ of candidates for $(\overline{W_{4,5,6,7}} \oplus \overline{P_0})$ thanks to $T_{1,5}$
- a reduced list $\mathcal{L}_2$ of candidates for $(\overline{W_{5,7}} \oplus \overline{P_1})$ thanks to $T_{2,6}$
- a list of one candidate $\mathcal{L}_3$ for $(\overline{W_{6,7}} \oplus \overline{P_2})$ thanks to T_3

Attack Step 3: Reduce the Guessing Cost The guess over 32-bit words is feasible but costly. In our situation we are able to reduce this cost. Indeed, all our pairs of equations, used to create candidate lists, can be split at the byte level. Due to the AES key schedule design, our pairs of equations contains the same guessed data and the $f_{2,r}$ function are applied at byte level. Then, the guesses list and the confrontations to known data can be established at a byte level too. As an example, we can guess the byte w_7^3 (the last byte of W_7) and confront it to the known value of $t_{0,4}^2$ through the equation system:

$$\left.\begin{array}{l} \overline{t_0^2} = \mathbf{SBox}[w_7^3] \\ \overline{t_4^2} = \mathbf{SBox}[w_7^3 \oplus p_3^3] \end{array}\right\} t_{0,4}^2 \stackrel{?}{=} t_0^2 \oplus t_4^2$$

Attack Step 4: Building the List of Keys Each quadruplet (L_0, L_1, L_2, L_3) is transformed into a valid candidate for K_1 that is composed of $\{W_4, W_5, W_6, W_7\}$:

1. L_0 gives the knowledge of $\overline{P_0} = f_1(W_{0,1,2,3} \oplus f_{2,0}(L_0))$
2. L_1 gives the knowledge of $\overline{P_1} = f_1(W_{1,3} \oplus f_{2,1}(L_1))$
3. L_2 gives the knowledge of $\overline{P_2} = f_1(W_{2,3} \oplus f_{2,2}(L_2))$
4. L_0 gives the knowledge of $\overline{W_7} = L_0$
5. L_3, P_2 and W_7 gives the knowledge of $\overline{W_6} = W_7 \oplus P_2 \oplus L_3$
6. L_2, P_1 and W_7 gives the knowledge of $\overline{W_5} = W_7 \oplus P_1 \oplus L_2$
7. L_1, P_0 and $W_{5,6,7}$ gives the knowledge of $\overline{W_4} = W_{5,6,7} \oplus P_0 \oplus L_1$

This corresponds to the exhaustive list of keys that have the same pair of extreme keys: (K_0, K_{14}), even if they have different second round key K_1.

Table 2. Occurrences of sizes of AES-256 key candidate set over 10 000 random keys

Remaining keys (2^x)	12	13	14	15	19	20	21	26
Encounter (%)	78.94	15.09	1.17	0.08	3.83	0.77	0.03	0.09
Average Time (s)	< 1	< 1	< 1	< 1	< 1	< 1	< 1	15.4

Simulation Results In order to estimate the number of keys that remains indistinguishable at the end of our constraint application, we run it onto a set of 10 000 random AES-256 keys. The number of candidates is 4 096 in $\sim 79\%$ of cases as depicted in Table 2. The executions take place onto a standard desktop computer with only one core used. The average time needed is ~ 0.03 second so we consider it negligible.

Considering the scenario of uncertainty in side-channel results, the negligible execution time for both candidate list establishment and good candidate extraction, allows the attacker to consider guessing depth of K_0 and K_{14}.

Remark 2. Errors in extreme keys may lead to inconsistency in equations and then to empty full key solution sets, discarding it without the need to check it against a plaintext/ciphertext pair.

3.2 Recover AES-192 Key

Attacker Model We consider here an attacker that managed to recover the first round key (K_0) and last round key (K_{12}) of an AES-192 implementation. The main key is the concatenation of K_0 and first half of K_1 so

Table 3. AES-192 key schedule equations

K_0	$W_0 = W_0$		K_6	$W_{24} = W_0 \oplus R_{0,1,2,3}$	
	$W_1 = W_1$			$W_{25} = W_1 \oplus R_{1,3}$	
	$W_2 = W_2$			$W_{26} = W_2 \oplus R_{2,3}$	
	$W_3 = W_3$			$W_{27} = W_3 \oplus R_3$	
K_1	$W_4 = W_4$		K_7	$W_{28} = W_{0,4} \oplus R_{0,1,2,3}$	
	$W_5 = W_5$	$R_0 = f_{2,0}(W_5)$		$W_{29} = W_{1,5} \oplus R_{1,3}$	$R_4 = f_{2,4}(W_{29})$
K_2	$W_6 = W_0 \oplus R_0$		K_8	$W_{30} = W_0 \oplus R_{0,1,2,3,4}$	
	$W_7 = W_{0,1} \oplus R_0$			$W_{31} = W_{0,1} \oplus R_{0,2,4}$	
	$W_8 = W_{0,1,2} \oplus R_0$			$W_{32} = W_{0,1,2} \oplus R_{0,3,4}$	
	$W_9 = W_{0,1,2,3} \oplus R_0$			$W_{33} = W_{0,1,2,3} \oplus R_{0,4}$	
	$W_{10} = W_{0,1,2,3,4} \oplus R_0$			$W_{34} = W_{1,2,3,4} \oplus R_{1,2,3,4}$	
	$W_{11} = W_{0,1,2,3,4,5} \oplus R_0$	$R_1 = f_{2,1}(W_{11})$		$W_{35} = W_{2,3,4,5} \oplus R_{2,4}$	$R_5 = f_{2,5}(W_{35})$
K_3	$W_{12} = W_0 \oplus R_{0,1}$		K_9	$W_{36} = W_0 \oplus R_{0,1,2,3,4,5}$	
	$W_{13} = W_1 \oplus R_1$			$W_{37} = W_1 \oplus R_{1,3,5}$	
	$W_{14} = W_{0,2} \oplus R_{0,1}$			$W_{38} = W_{0,2} \oplus R_{0,1,4,5}$	
	$W_{15} = W_{1,3} \oplus R_1$			$W_{39} = W_{1,3} \oplus R_{1,5}$	
K_4	$W_{16} = W_{0,2,4} \oplus R_{0,1}$		K_{10}	$W_{40} = W_{2,4} \oplus R_{2,3,4,5}$	
	$W_{17} = W_{1,3,5} \oplus R_1$	$R_2 = f_{2,2}(W_{17})$		$W_{41} = W_{3,5} \oplus R_{3,5}$	$R_6 = f_{2,6}(W_{41})$
K_5	$W_{18} = W_0 \oplus R_{0,1,2}$		K_{11}	$W_{42} = W_0 \oplus R_{0,1,2,3,4,5,6}$	
	$W_{19} = W_{0,1} \oplus R_{0,2}$			$W_{43} = W_{0,1} \oplus R_{0,2,4,6}$	
	$W_{20} = W_{1,2} \oplus R_{1,2}$			$W_{44} = W_{1,2} \oplus R_{1,2,5,6}$	
	$W_{21} = W_{2,3} \oplus R_2$			$W_{45} = W_{2,3} \oplus R_{2,6}$	
	$W_{22} = W_{0,3,4} \oplus R_{0,1,2}$			$W_{46} = W_{3,4} \oplus R_{3,4,5,6}$	
	$W_{23} = W_{0,1,4,5} \oplus R_{0,2}$	$R_3 = f_{2,3}(W_{23})$		$W_{47} = W_{4,5} \oplus R_{4,6}$	$R_7 = f_{2,7}(W_{47})$
			K_{12}	$W_{48} = W_0 \oplus R_{0,1,2,3,4,5,6,7}$	
				$W_{49} = W_1 \oplus R_{1,3,5,7}$	
				$W_{50} = W_2 \oplus R_{2,3,6,7}$	
				$W_{51} = W_3 \oplus R_{3,7}$	

we explain here how to recover the exact value of K_1 first half in a negligible computational effort using the knowledge of (K_0, K_{12}) pair. The brute force approach to try every 2^{64} possibility is considered hard/infeasible in practice.

Attack Step 1: Guess-Free Knowledge The equations $W_i, i \in [0, 52[$ are redefined for AES-192 in Table 3, expressed as a combination of current main key columns: $\{W_0, W_1, \ldots, W_5\}$. We introduce $R_i, i \in [0, 7]$ columns, resulting from application of $f_{2,r}()$ functions onto some W columns.

The knowledge of $\{W_0, \ldots, W_3, W_{56}, \ldots, W_{59}\}$ allows to use K_{14} equations and revert them to gain effortless knowledge: $R_{0,4}$, $R_{1,5}$, $R_{2,6}$ and $R_{3,7}$ from which we do not know the atomics. Those new knowledge continue to give us guess-free elements: $\overline{W_{33}}$, $\overline{W_{38}}$, $\overline{W_{39}}$, $\overline{W_{43}}$, $\overline{W_{44}}$ and $\overline{W_{45}}$.

Attack Step 2: Guess Pattern to Reach Constraints The Figure 4 is describing the path a guessed value G for a W_5 byte will follow and how it sets the five constraints $\{C_1, \ldots, C_5\}$ that will validate or invalidate the guess G. The known operations/data are in blue, constraints are in red. 4-byte functions considered here (i.e. $f_{2,i}()$ family and $\oplus$) can be split at a byte level, then, for a sake of clarity, applying such a function onto only one byte in a 4-byte word means here that we applied the part of the

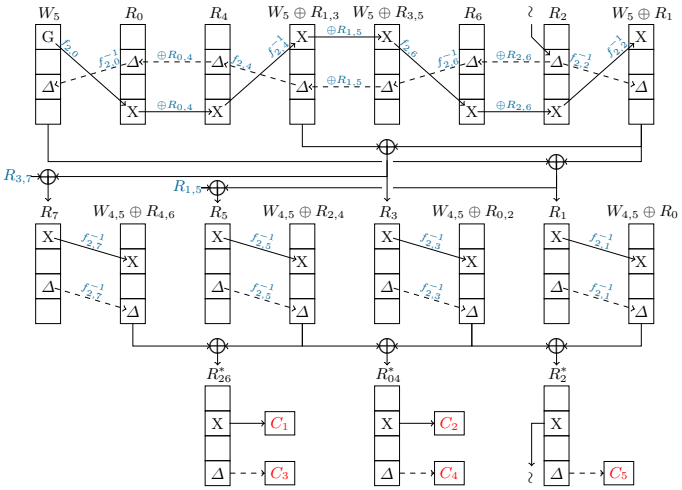

Fig. 4. AES-192 Exploration Path.

function concerning this byte as it do not involves the 3 other bytes. For sake of clarity again we removed the set of known values: W_0, W_1, W_2, W_3 from the equations. Original equations are given in Table 3.

One can follow the guessed byte G that allows to compute, through plain arrows, the X values (Δ should be first be ignored). Once column $W_5 \oplus R_1$ (last of first stage) is reached we can combine existing X values, and known $R_{3,7}$ and $R_{1,5}$, vectors to compute the X values of the second stage. Those are combined again to compute the X in third stage. As $R_{2,6}$ and $R_{0,4}$ are known, we can confront it to the proposed $R_{2,6}*$ and $R_{0,4}*$. We keep only G values that fit with those constraints (C_1 and C_2). The concerned byte of R_2* is not known, so we cannot use it as a constraint, but we can re-inject it in R_2 on first stage. We choose to change the letter from X to Δ at this point in order to differentiates the paths. Now we can compute Δ in both directions on the whole first stage. The Δ on second stage and then third can then be computed, leading to new constraints (C_3 and C_4 in $R_{2,6}*$ and $R_{0,4}*$) and a last one (C_5) in R_2* as, this time, it corresponds to a known value obtained during X path.

This set of five constraints apply on G but also on another byte of the W_5 column obtained during the process. All simulation cases considered show only one solution for the pair of bytes (G, Δ) in W_5. The two other bytes can be recovered by the same process, one just have to apply a rotation of one byte of all column of Figure 4. Once W_5 recovered we see that $W_{4,5} = (W_{4,5} \oplus R_0) \oplus R_0$ is recovered too, so is W_4.

Simulation Results

We run our recovery algorithm on 10 000 randomly chosen AES-192 keys. Every case considered lead to only one solution (the good one) in $\sim 0,002$ second of exploration in average.

4 Conclusion

In this paper we shown how the recovery of first and last round keys of AES-192 or AES-256 might be used to recover the main key. Some existing attacks designed onto AES-128 and that cannot be trivially adapted to higher sizes might became available. This method can also reduce the complexity of attacks very sensitive to errors, because constituted of several dependent chained attacks, by allowing to replace them by several independent attacks. Despite the fact that the attacker scenarios are limited, we question the strength of the AES key scheduling process. It might have been stronger, especially for bigger sizes.

Further works might be considered as determining the existence (or not) of AES-192 key that shares same extreme keys could be interesting. Applying this method to others algorithms than AES or DES as an overview of security of the key scheduling processes might be valuable.

References

[BCO04] Éric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES '04*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer-Verlag, 2004.

[CR17] Christophe Clavier and Léo Reynaud. Improved Blind Side-Channel Analysis by Exploitation of Joint Distributions of Leakages. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES '17*, volume 10529 of *Lecture Notes in Computer Science*, pages 24–44. Springer, 2017.

[CRW18] Christophe Clavier, Léo Reynaud, and Antoine Wurcker. Quadrivariate improved blind side-channel analysis on boolean masked AES. In *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings*, pages 153–167, 2018.

[FH08] Julie Ferrigno and Martin Hlavác. When AES blinks: introducing optical side channel. *IET Information Security*, 2(3):94–98, 2008.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES '08*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '01*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer-Verlag, 2001.

[GPT14] Daniel Genkin, Itamar Pipman, and Eran Tromer. Get Your Hands Off My Laptop: Physical Side-Channel Key-Extraction Attacks on PCs. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES '14*, volume 8731 of *Lecture Notes in Computer Science*, pages 242–260. Springer, 2014.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 444–461. Springer, 2014.

[KJJ98] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Introduction to Differential Power Analysis and Related Attacks. Technical report, Cryptography Research Inc., 1998. http://www.cryptography.com/resources/whitepapers/DPATechInfo.pdf.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996.

[lB14] Hélène le Bouder. *Un formalisme unifiant les attaques physiques sur circuits cryptographiques et son exploitation afin de comparer et rechercher de nouvelles attaques.* PhD thesis, École Nationale Supérieure des Mines de Saint-Étienne, 2014.

[LDL14] Yanis Linge, Cécile Dumas, and Sophie Lambert-Lacroix. Using the Joint Distributions of a Cryptographic Function in Side Channel Analysis. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design – COSADE '14*, volume 8622 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2014.

[Nat77] National Bureau of Standards. Data Encryption Standard. Federal Information Processing Standard #46, 1977.

[Nat01] National Institute of Standards and Technology. Advanced Encryption Standard (AES). Federal Information Processing Standard #197, 2001.

[PSG16] Romain Poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 61–81, 2016.

[TGWC18] Hugues Thiebeauld, Georges Gagnerot, Antoine Wurcker, and Christophe Clavier. SCATTER: A new dimension in side-channel. In *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop,*

COSADE 2018, Singapore, April 23-24, 2018, Proceedings, pages 135–152, 2018.

[VGRS12] Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, pages 390–406, 2012.