Implementation and improvement of the Partial Sum Attack on 6-round AES

Francesco Aldà
Ruhr-Universität Bochum, francesco.alda@rub.de
Riccardo Aragona
University of Trento, riccardo.aragona@unitn.it
Lorenzo Nicolodi
University of Trento, lo@hidden-bits.com
Massimiliano Sala
University of Trento, maxsalacodes@gmail.com

I. Introduction and Problem Statement

The Partial Sum Attack is one of the most effective attacks, independent of the key schedule, developed in the last 15 years against reduced-round versions of AES [1]. 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 show the performances we achieve.

II. Improvement

After the publication of the Partial Sum Attack [1], many researchers worked on the integral cryptanalysis of AES, finding new extensions or improvements for this class of attacks (see for example [2, 3, 4]). Our approach started from a different perspective, providing a full implementation of the attack as it is described in [1], trying to understand where some other potentialities could be exploited. We indeed believe that an effective implementation of this attack had not already been presented before this work and that this kind of practices can lead to better understand the dynamic of an attack.

Throughout the rest of the paper, we denote by $\bar{\Delta}$-set a group of 2^{32} plaintexts such that one state column of bytes at the input of *MixColumns* of the 1st round ranges over all possible values of $(\mathbb{F}_{2^8})^4$ and all other bytes are constant. We briefly recall that the Partial Sum Attack is a probabilistic attack based on the fact that the state bytes of a $\bar{\Delta}$-set at the end of the 4th round are balanced, i.e. fixing a position (i, j), $0 \leq i, j \leq 3$, the bytes in position (i, j) of the states after 4 rounds sum up to 0. Guessing only 5 key bytes and performing a partial decryption of 2 rounds, it is possible to compute the sum on a fixed position (i, j). If the sum is nonzero, then the guess for the key bytes is certainly wrong. Whenever the sum is zero, the key space (5 bytes) is expected to be reduced by a factor 256 (for further details we refer to [1]).

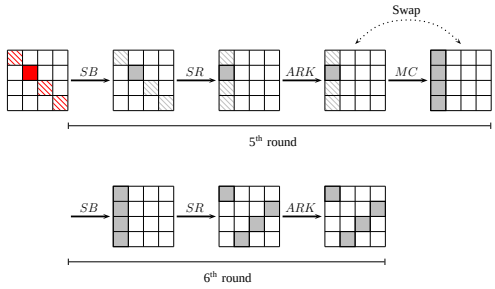

Figure 1: *The state bytes at the end of the 4th round (in red) which can be computed for a configuration according to which four ciphertexts bytes $(c_0^{(l)}, c_1^{(l)}, c_2^{(l)}, c_3^{(l)})$, for $1 \leq l \leq 2^{32}$, are extracted*

In the original paper [1], it is claimed that at least 6 sets of 2^{32} plaintexts, which form a $\bar{\Delta}$-set, are necessary in order to find a correct 5-tuple $(k_0, k_1, k_2, k_3, k_4)$, where k_0, k_1, k_2, k_3 are bytes of the 6th round key and k_4 is a byte of the 5th round key. We claim that only two $\bar{\Delta}$-sets are necessary in order

to find the correct $(k_0, k_1, k_2, k_3, k_4)$. In fact, fixing one configuration according to which four ciphertexts bytes $(c_0^{(l)}, c_1^{(l)}, c_2^{(l)}, c_3^{(l)})$, for $1 \leq l \leq 2^{32}$, are extracted (see [1]), one can compute the sum in four different state bytes at the end of the 4th round. In Figure 1, we provide an example.

If we consider each sum as independent, using only two $\bar{\Delta}$-sets, the probability that for a 4-tuple (k_0, k_1, k_2, k_3) exists for each row a value k_4 which gives a zero sum for both $\bar{\Delta}$-sets is $(256)^{-8}$. Therefore, checking the value of the sum for at least three bytes at the end of the 4th round, for which there exist values k_4 (dependent only on the row) which give zero sums for both $\bar{\Delta}$-sets, is expected to eliminate all but one 4-tuple (k_0, k_1, k_2, k_3), that is, the correct one. Note that the values of the 5th round key bytes k_4, which produce zero sums, may be different for each row, but, as for (k_0, k_1, k_2, k_3), their correctness follows by the crosschecking between the two $\bar{\Delta}$-sets. The hypothesis which bears our result consists of considering the sums on four rows as independent. We believe that this hypothesis substantially holds even in practice. In fact, 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 have made it negligible. Therefore we observed that the number of chosen plaintexts which are necessary in order to mount the attack can be reduced from 6 $\bar{\Delta}$-sets of 2^{32} elements to only 2.

In general, this reduction implies that more verification steps are needed, since one can check only two sums instead of six, which might increment the number of false positive key bytes, which are discarded with very high probability by the crosschecking on other rows. Therefore, in our first implementation we used 3 $\bar{\Delta}$-sets of 2^{32} elements (instead of 2 as claimed) and checked the value of the sum for four bytes at the end of the 4th round, in order to lower as much as possible the probability of false positives. We made this choice since using only 2 $\bar{\Delta}$-sets involves more verification steps (as already observed, there are more wrong key candidates which give a zero sum), which are time consuming operations in our current implementation. We simply observed that using 3 $\bar{\Delta}$-sets resulted slightly faster. Nevertheless, we plan to improve our implementation using only 2 $\bar{\Delta}$-sets.

This improvement has been reached independently from [4], where a quite similar result is described. In fact, our result is based on different observations and reaches different conclusions.

III. Experimental Results and Conclusions

We implemented a parallel version of the Partial Sum Attack combined with our slight improvement, which can be run on multiple core systems. To retrieve the whole 16-byte key, the attack has to be run 4 times, according to the four configurations shown in [1]. Since the attack requires the management of huge bit vectors, our implementation maps every Boolean vector's element to a bit inside an `unsigned char`'s array. This allow us to save space and time while writing and reading the encrypted arrays to and from the disk. Moreover, we aimed to speed up the partial decryption involved by means of look-up tables and `XOR` on 64-bit blocks.

The final outcome of this effort was interesting in terms of time and memory used. The attacks have been launched on 6 desktop PC, with 4 cores (Intel Pentium CPU G640 @2.80GHz) and 8GB of RAM each, using 25 processes in total. The first process coordinated the attacks, while the 24 workers performed the attacks in 11,5 days on average. Based on this result, we estimate that, on average, the 128-bit 6th round key can be retrieved in 25,8 hours, using 256 workers with less than 2GB of RAM used by each worker.

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

References

[1] N. Ferguson, J. Kesley, S. Lucks, B. Schneier, M. Stay, D. Wagner and D. Whiting, "Improved cryptanalysis of Rijndael", *Proc. of FSE 2001*, LNCS vol. 1978, pp. 103–111, 2001.

[2] S. Galice and M. Minier, "Improving Integral Attacks against Rijndael-256 up to 9 rounds", *Proc. of AFRICACRYPT 2008*, LNCS vol. 5023, pp. 1–15, 2008.

[3] Y. J. Li and W. L. Wu, "Improved Integral Attacks on Rijndael", *Journal of Information Science and Engineering*, vol. 27(6), pp. 2031–2045, 2011.

[4] M. Tunstall, "Improved "Partial Sums"-based Square Attack on AES", *Proc. of SECRYPT 2012*, pp. 25–34, 2012.