JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 33, 1085-1100 (2017)
DOI: 10.6688/JISE.2017.33.4.15

# Short Paper

# Another Security Evaluation of SPA Countermeasures for AES Key Expansion in IoT Devices

YANG LI, MENTING CHEN AND JIAN WANG
*College of Computer Science and Technology*
*Nanjing University of Aeronautics and Astronautics*
*Nanjing, 211106 P.R. China*
*E-mail: {li.yang; chenmengting; wangjian}@nuaa.edu.cn*

Internet of things (IoT) devices are easily exposed to physical attackers for their easy access. Therefore, the cryptographic algorithms should be implemented carefully considering the key recovery attacks such as side-channel attacks and fault attacks. This work focuses on the simper power analysis against AES key expansion in the attack scenario of the IoT device. We mainly focused on the power analysis countermeasure applied to AES key expansion proposed and evaluated by Clavier *et al.* in CHES 2014. Their proposed column-wise random order countermeasure showed certain resistance against power analysis. Clavier *et al.* then analyzed the improved key recovery attack that combines power analysis with fault injections. In this work, we argue that extracting power information of AES state is more preferred than performing fault injections for practical attackers. This work first comprehensively evaluates the random order countermeasure assuming the attackers use the power consumptions of AES state to accelerate the key recovery. The relationship between the key recovery result and the amount of used information are verified with both theoretical analysis and key recovery simulations. The results demonstrate a set of effective key extractions with no fault injections. The most effect attack uses the Hamming weight of 12 bytes for 2 AES executions, whose key extraction finishes in 1 minute. This work also considers to use algebraic side-channel attack to construct a general security evaluation method for variant countermeasures. We explain the successful key recovery of algebraic side-channel attack on AES key expansion and discuss some observations.

***Keywords:*** power analysis, AES key expansion, random order, algebraic side channel attack, SAT solver

## 1. INTRODUCTION

With the goal of improving quality of life, the smart city uses the information and communication technology (ICT) and Internet of Things solutions to integrate and manage the assets of the modern city such as transportation system, education system, government service system *etc.* To sense the real-time information, many small IoT devices which gather and transmit the sensed data, will be employed around people's daily life and connected to the network. According to the Gartner, Inc, 64 million IoT device will be used over the world in 2016. In 2020, the IoT, which excludes PCs, tablets and smart-

Received August 3, 2016; revised September 26, 2016; accepted October 21, 2016.
Communicated by Zhe Liu.

phones, will grow to 26 billion units installed. The security and privacy of these small devices have the key importance to keep the system as controlled and good smart.

As the basic of security of IoT devices, the collected data needs to be encrypted before transmission. The cryptographic protocols and primitives are the basis for constructing, maintaining and providing the security in Internet of things. To keep the devices low cost and responsive, the symmetric key cryptographic system such as block ciphers and hash functions are mostly used.

In this recent decade, it is realized that even for a well-designed cryptographic application, the attackers can bypass the mathematical resistance and use physical attacks to practically break it. Non-invasive physical attacks including side-channel attacks [1] and fault attacks [2] have practically broken the implementations of cryptographic primitives in the real word. Thus, evaluating the resistance of an implemented cryptographic primitive against the physical attacks becomes a necessity. The IoT devices in smart cities are especially vulnerable to physical attacks for their easy access.

This work focused on the simper power analysis against Advanced Encryption Standard (AES) [3] key expansion in the attack scenario of an IoT device. The physical security of AES implementation has been largely discussed in academic [4-8]. In CHES 2014, the security of AES key expansion against simple power analysis was discussed considering two possible countermeasures [9]. The first type of countermeasure is Boolean masking, and the second countermeasure is random order for the storage of the AES round keys. Based on the work of [9], this work mainly has two main contributions towards more accurate security evaluations of the power attack countermeasures for AES key expansion.

First, this work extends the security evaluation for the random order countermeasure. As for the security of random order countermeasure, previous work found that the key extraction using only power traces could fail after all kinds of optimization. Then it was proposed to use fault injection in additional with power traces to accelerate the key recovery. We believe fault injection discussed in [9] is not the first choice of a side-channel attacker. A more practical way is try to extract more information from the power traces, for example the Hamming weight of AES state, to accelerate the key recovery. In detail, we point out that using Hamming weight of AES state can help attackers to accelerate the key recovery that does not require fault injections. We study the possible techniques and variations that the attackers could apply in practice. All the discussed attack scenarios are verified via simulations. The results show that with the Hamming weight of an AES state from 1 or 2 AES executions, the key recovery can be accelerated to different extend. We discuss the trade-off between the additional information extracted from power traces and the key recovery efficiency via key recovery simulations. A cost-effective key recovery uses Hamming weights of 12 bytes for 2 AES executions, whose average key recovery time is less than 1 minute. This work shows that without fault injections, the attacker can efficiently extract AES key against random order countermeasure. A preliminary version of this part was presented in the 2nd international conference on cloud computing and security (ICCCS 2016), Nanjing, China in July 2016, which was sponsored by NUIST, MSU, NCCU, NDHU and NUAA.

Second, we consider to use the algebraic side channel attacks as a general method for evaluating various parameters in countermeasures. As the first step, this work verified the successful key recovery result for the basic attack scenario. The algebraic side chan-

nel attack (ASCA) is an important attack method, which is the combination of algebraic attacks and side channel attacks. ASCA uses the side channel leakage (for example, Hamming weight, Hamming distance, *etc.*) and algebraic presentation of the cipher to feed the SAT solver for the key recovery. Since the key recovery is fully automatic without human factor, we consider it is a general and fair approach to evaluate various possible countermeasures. In this paper, we study the principle and application of the algebraic attack, and prove the possibility of the application of the ASCA in the AES key expansion analysis. We assume that attackers can obtain the Hamming weight of each byte of the round keys, through the analysis of AES key expansion algorithm and use the Hamming weight nonlinear equations are established, the equations are transformed to the SAT problem. Finally, we use SAT solver to solve the problem and recover the original key. Through the experiment proved this method can correctly recover the key, and in ten times the minimum only know four rounds keys Hamming weight can successfully recover the key. The possibility of the application of the algebraic side channel attack to the AES key expansion analysis is verified.

The rest of this paper is organized as follows. In section 2, we review AES and briefly introduce the existing power analysis attacks against AES key expansion and their countermeasures. In section 3, we propose a new fast key recovery method for random order countermeasure. Section 4 shows the experimental verification of the proposed attack. Section 5 explains our idea of a general security evolution framework and the first result using algebraic side channel attack to successfully recover the AES secret key. Section 6 concludes this paper.

## 2. PRELIMINARIES

### 2.1 Review AES Key Expansion

Advanced Encryption Standard (AES) was selected by the U.S. National Institution of Standard and Technology (NIST) in 2001 to replace the Data Encryption Standard (DES). The block size of AES is fixed to 128 bits, and the key size can be 128, 192 or 256 bits, corresponding to 10, 12 or 14 rounds of encryption and decryption. The 128-bit data block in encryption, decryption and round keys are usually considered as a 4×4 byte matrix. The byte matrix of intermediate date of 128 bits, which is called state, represents a block of data with 128 bits or 16 bytes.

This paper mainly discusses the 128-bit version of AES as AES-128. Fig. 1 depicts the encryption flow of AES-128. Each AES round consists of four operations including SubBytes, ShiftRows, MixColumns and AddRoundKey except the final round which is without MixColumn.

We denote $K$ as the initial key. Each round uses its round key $K_r$ for $r = 0, 1, \ldots, 10$ and we have $K_0 = K$. All the round keys are the results of the key expansion based on the initial key $K$. The 16 key bytes of $K_r$ are expressed as $k_{r,i}$ and $i = 0, 1, \ldots, 15$. The key expansion of AES-128 is based on the following formulas

$$k_{r,i} = k_{r\text{-}1,i} \oplus S(k_{r\text{-}1,12+((i+1) \bmod 4)}), \text{ for } i = 0, \ldots, 3 \quad (1)$$

$$k_{r,i} = k_{r\text{-}1,i} \oplus k_{r,i\text{-}4}, \text{ for } i = 4, \ldots, 15 \quad (2)$$

where $S$ is the AES substitution-box (S-box) and $c_r$ is the round constant with variable $r$. Note that S-box is the only non-linear operation in the AES key expansion. For AES key expansion, each round key, *e.g.* $K_0$, can be expanded to all the other round keys.

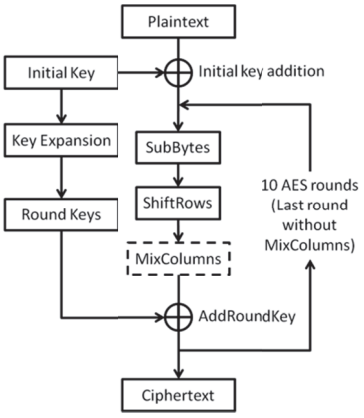


Fig. 1. AES encryption flow.

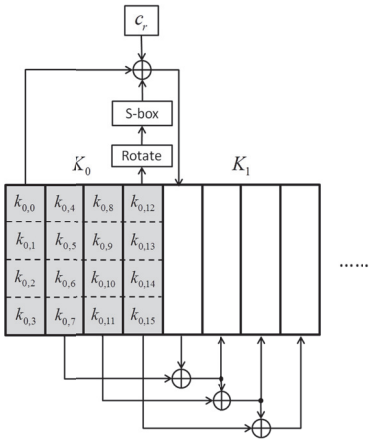


Fig. 2. AES key expansion.

The process of key expansion from $K_0$ to $K_1$ is shown in Fig. 2. The round keys can be computed column by column. For a round key, the computation of the first column is different from the other three columns, which involves S-box, byte rotate and round constant $c_r$.

### 2.2 AES Key Expansion Under Simple Power Analysis

When considering the security of AES against physical attacks, most of the literatures focus on the round function of AES. However, there are some works mentioned that the AES key can be extracted with Simple Power Analysis (SPA) from the key expansion part of AES. In contrast to Differential Power Analysis (DPA), SPA observes only a few or only one power trace for the key extraction. Among the SPA attacks on AES key expansion, most of them assumed the target to be an AES implementation on an 8-bit micro controller. Furthermore, the leakage model for an 8-bit micro controller is usually considered to fit the Hamming weight model. In other words, the attacker can extract the Hamming weight of the operated 8-bit value from the power traces. As for the key expansion, it is assumed that the Hamming weight of key bytes of all the round keys can be extracted [10-12].

In 2003, Mangard [13] first described this key recovery problem as the SPA attack on AES key expansion. It is found that the Hamming weight extracted from the power consumption of smart cards can be used to reveal the initial key of AES. The key recovery process is essentially a continuously reduction of the key space using Hamming weight information and the internal computational relations between round key bytes. To make the key recovery effective and efficient, the key recovery algorithm has to use several techniques in accelerating the process of key recovery.

In 2005, VanLaven *et al.* [14] improved the key recovery process for the SPA attack

on AES key expansion. They proposed a key search algorithm as the guess-compute-and-backtrack algorithm. After guessing a key byte value, some related key bytes could be computed with guessed keys, then all the computed key byte values can be checked with the Hamming weight information. If all the key byte values are correct, the next key byte value will be guessed. Otherwise, the algorithm backtrack to a previous stage. Another important contribution of [14] is that the author discussed how to optimize the guess sequence of the key bytes to maximize the number of computable key bytes during recover process. As the result of the optimized guess sequence, the attack efficiency is largely improved. In [14], the key of AES-128 can be recovered as fast as 16ms.

### 2.3 Random Order as a Countermeasure for AES Key Expansion

In 2014, Clavier *et al.* [9] first conducted the security evaluation of AES key expansion considering two approaches of countermeasures. As the countermeasure increases the difficulty of SPA key recovery, Boolean masking and a column-wise random order storage of key bytes have been considered. It is shown that for the Boolean masking countermeasure, a 11-byte entropy masking and a 16-byte entropy masking are still vulnerable to the improved SPA attacks. In this paper, we concentrate on the random order countermeasure, since it seems to be more effective than Boolean masking as a power analysis countermeasure.

The applied random order countermeasure in [9] is a column-wise random order. As mentioned in Section 2.1, AES round keys could be computed column by column. Based on this property, it is reasonable to randomly shuffle the key bytes which are inside the same column of a round key. After applying the column-wise countermeasure, the sequence of the 4 key bytes is in a random order. For each column, although 4 Hamming weight could still be obtained using SPA, the attacker cannot ensure the exact position for the 4 non-ordered Hamming weight. Fig. 3 is an example of the available information of SPA attack for random order countermeasure. The attackers get 4 non-ordered Hamming weight for each column of round key, *e.g.* $K_{10}$.

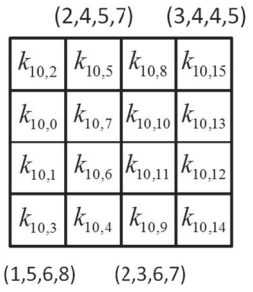


Fig. 3. Example of random order countermeasure.

**Basic Key Recovery Attack Against Random Order Countermeasure** In [9], Clavier *et al.* first tried to apply the key recovery using Hamming weight information of random ordered key bytes as the basic attack. Basic attack adopts the idea of a booking system combined with the guess-compute-and-backtrack algorithm. That is, to every position, attackers will book a candidate Hamming weight for it and then determine its val-

ue either by guessing or computing. If one position could be directly calculated with already determined key bytes, then compute it and compare it with the candidate Hamming weight list. If couldn't, guess the value of it should satisfy the constraint of its booked Hamming weight. Once the Hamming weight of the computed position is not in the candidate list, or all available values of the guessed position are exhausted, it will backtrack to the previous guessed position. For instance, Fig. 3 shows the Hamming weight of $K_{10}$ that the attackers get. Among the 4 candidates (1, 5, 6, 8), first book $HW(k_{10,0}) = 1$ and guess its value $k_{10,0} = 1$. Similarly, book $HW(k_{10,4}) = 2$ and guess its value $k_{10,4} = 3$. Then compute the $k_{9,4} = k_{10,0} \oplus k_{10,4} = 2$ and check it with its candidate Hamming weight list. If $HW(k_{9,4}) = 1$ is in the list, then book value 1 as the Hamming weight of $k_{9,4}$ and update the candidate Hamming weight list of the other 3 key bytes in the same column with $k_{9,4}$. If $HW(k_{9,4}) = 1$ is not in the list, backtrack to $k_{10,4}$ and re-guess another value satisfying $HW(k_{10,4}) = 2$. If no more available value satisfies $HW(k_{10,4}) = 2$, then book Hamming weight of $k_{10,4}$ with another candidate.

Table 1 shows the result of basic attack over 100 runs of experiments in [9]. About 27% of the key recovery experiments cannot finish within a reasonable time limit, which implies that the computation effort of the key recovery for random order countermeasure is non-negligible. Only 41% cases ended in 1 hour and we estimate the average runtime is as long as 2 hours.

**Table 1. Basic attack result against random order in [9].**

| Time Elapsed | ≤ 30 min | ≤ 1h | ≤ 2h | ≤ 3h | ≤ 4h | ≤ 5h | ≤ 6h | + 6h |
|---|---|---|---|---|---|---|---|---|
| # over 100 runs | 6 | 25 | 41 | 55 | 66 | 71 | 73 | 27 |

**Fault Injection Key Recovery Attack** In order to accelerate the key recovery, in [9] Clavier *et al.* considers an attack that combines fault attack (FA) with SPA. The idea is that the attackers could randomly induce a fault into a key byte. The fault model is a random modification of the chosen byte value. Due to key expansion, the induced fault will cause differential in some round key bytes and also their corresponding Hamming weight. By SPA, attackers could observe the differentials in Hamming weight and infer the position of these changed key bytes. For example, Fig. 4 shows the changed positions in $K_{10}$ when $k_{9,0}$ is modified. The attackers will compare the original Hamming weight to the fault-induced Hamming weight and observe the changes. Then, attackers could deduce out the position of the changed key bytes and reduce the number of candidate Hamming weight of other 3 bytes in the same column. Afterwards, attackers will retrieve the round keys using the idea of basic attack, that is the booking system with guess-compute-and-backtrack algorithm.

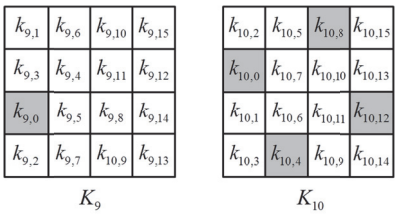


Fig. 4. Fault propagation in round keys.

The experiment result of fault injection key recovery attack indicates that the average attack time is reduced to 20 minutes using 1 fault injection. However, we notice that this attack pattern uses two kinds of attack techniques as SPA and FA. It requires more knowledge from attackers and increases the cost and equipment difficulty of the attack.

## 3. FAST KEY RECOVERY WITHOUT FAULT INJECTION AGAINST COLUMN-WISE RANDOM ORDER COUNTERMEASURE

In [9], two types of key recovery attacks have succeeded in attacking the AES key expansion with the column-wise random order countermeasure. The basic attack completes the key recovery using 2 hours in average and there are still 27% of experiments cannot finish within a reasonable timeout. Then, fault injection is combined with basic attack so that the average key recovery time is reduced to 20 minutes and the success rate is improved. However, the usage of two kinds of attack techniques, SPA and FA, inevitably require more expertise from attackers and increases the cost of the attack. Furthermore, since there are key recovery attacks that use the fault injections to key expansion only [15], the attack method that combines fault analysis and power analysis makes less sense.

In this paper, we propose an attack method that only uses the power consumption information to perform the security evaluation of column-wise random order. The basic idea is to use additional Hamming weight information of AES state to help recover the round keys. Our attack pattern is based on the circumstance that attackers could get Hamming weight of some bytes of the AES state. Hereafter, from an attacker's perspective, we discuss the practical key recovery strategy including the choice of start point of the attack, the choice of the AES state and the key guessing sequence.

### 3.1 Choice of Start Point

For a target chip that executes the AES encryption, the attackers could have access to both the plaintext and the corresponding ciphertext. The start point of our attack must start near these public data, *i.e.* plaintext and ciphertext. The power traces of the calculations near the public data are easier to extract even with random delay countermeasure that adds random delays during AES execution.

Under the assumption that plaintext, ciphertext and the related power consumption can be achieved with the same difficulty, we select ciphertext as the start point for a faster key recovery. To explain the reasons, we review the graphed representation of AES-128 key expansion from [14] to describe the computational relationship between round key bytes. In [14], the optimized key guessing sequence that maximized the key recovery efficiency was discussed using this graph.

Fig. 5 shows the graphed representation of key expansion for AES-128, in which each circle represents a key byte. The label $k_{N,i}$ on the left side stands for the position of every key byte, where $N$ is the round number and $i$ represents the key byte position. For example, the 11 circles of the top line from left to right correspond to $k_{0,0}$, $k_{1,0}$,…until $k_{10,0}$. For each grey triangle, the 3 vertexes have an exclusive-or computational relationship, while for each black triangle, the 3 vertexes have the computational relationship using S-box. Note that given any two vertexes of a triangle, the other one can be computed. For

example, the second circle of the top line is $k_{1,0}$ and the first two circles of the second line are $k_{0,4}$ and $k_{1,4}$. Since $k_{1,4} = k_{1,0} \oplus k_{0,4}$, knowing any two of them, the other one can be directly computed.

In key recovery process, the attackers guess key byte values according to the Hamming weight in a certain sequence, while obtaining the key bytes which can be calculated with the initially identified key bytes. When the directly computed key byte value does not match the Hamming weight of its position, the current guessed key byte is wrong, the key recovery algorithm backtracks to the previous stage. For each guess of a key byte, itself and the key bytes that can be calculated from it can be used to check the Hamming weight information. Therefore, for each key guess, the more key bytes can be calculated with known key bytes, the faster the key space shrinks and the higher the key recovery efficiency is.

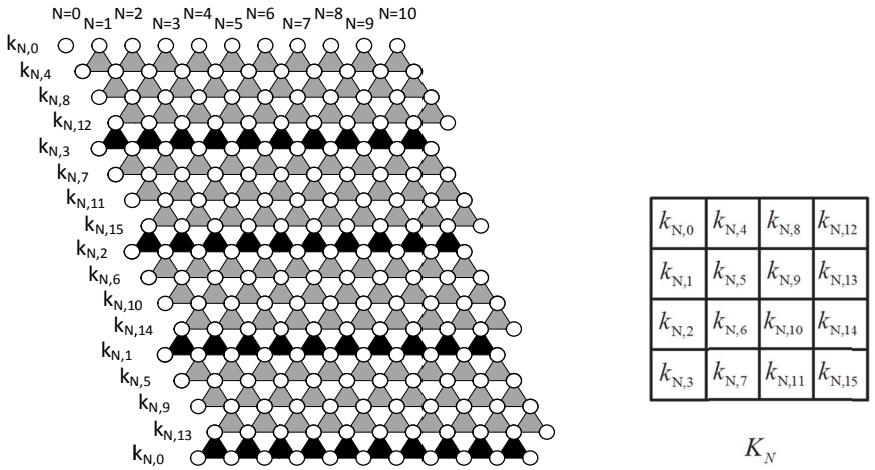


Fig. 5. Graphed representation of key expansion for AES-128.

We use Fig. 5 to explore the optimized recovery sequence for the start point near plaintext or ciphertext. If the start point is set to plaintext, $K_0$ is the main target for key recovery. When guessing each byte of $K_0$, one of the best guess sequence is $k_{0,13}$, $k_{0,0}$, $k_{0,4}$, $k_{0,8}$, …, $k_{0,5}$, $k_{0,9}$. Starting with guessing $k_{0,13}$, the number of key values that can be additionally calculated for each key value guess is 1,1,1,1,2,2,2,2,3,3,3,3,4 and 4. After guessing $k_{0,9}$, all remaining key bytes can be calculated.

If the start point is set to ciphertext, $K_{10}$ is the main target for key recovery. When guessing each byte of $K_{10}$, one of the best guess sequence is $k_{10,0}$, $k_{10,4}$, $k_{10,8}$,…, $k_{10,9}$, $k_{10,13}$. Starting with guessing $k_{10,0}$, the number of key values that can be additionally calculated for each key value guess is 1,2,3,3,4,5,6,6,7,8,9,9,10 and 10. After guessing $k_{10,13}$, all remaining key bytes can be calculated.

Based on the above analysis, it is clear that the number of key bytes that can be calculated in the key recovery process for $K_{10}$ is much larger than that for $K_0$. Therefore, the key space is shrinking much faster and the key recovery efficiency is much higher when using ciphertext as the start point.

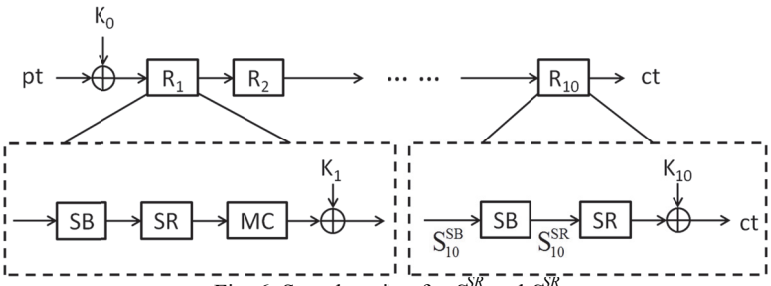


Fig. 6. State location for $S_{10}^{SR}$ and $S_{10}^{\widetilde{SR}}$.

### 3.2 Choice of AES State

There are two AES states near ciphertext can be chosen, *i.e.* the state before Sub Bytes $S_{10}^{SR}$ and the state before ShiftRows $S_{10}^{\widetilde{SR}}$, as shown in Fig. 6. Note that the state after ShiftRow is essentially equivalent to $S_{10}^{SR}$ since they can be linearly transformed into each other. If the Hamming weight of $S_{10}^{SR}$ is used, some candidates of Hamming weight ordering directly fail the linear calculation among $K_{10}$, ciphertext and $S_{10}^{SR}$. If the Hamming weight of $S_{10}^{SR}$ is used, the non-linear part S-box could enable a faster restriction of Hamming weight than that of $S_{10}^{SR}$. The reason is that S-box is the only non-linear part in AES key expansion, which confuses the relationship between input bits and output bits. By a preliminary experiment, we find out that the key space restricted using $S_{10}^{SR}$ is about 49.4% smaller than that of $S_{10}^{SR}$. Thus, we consider the attackers will choose Hamming weight of $S_{10}^{SR}$ to pursue the best key recovery complexity.

### 3.3 Key Recovery Algorithm with AES State Hamming Weight

The key recovery problem becomes a restriction process of key space using 176 Hamming weight of round keys with column-wise random order denoted as $HW_K$, the Hamming weight of AES state $S_{10}^{SR}$ denoted as $HW_S$ and the ciphertext denoted as CT. In the key recovery process, first a function GETHW is called to return all the possible Hamming weight sequences of $K_{10}$ with exact positions as $HWSET_{K10}$. Then, for every possible Hamming weight sequence $HW_{K10}^t$ in $HWSET_{K10}$, two functions KeyRecover and KeyVerify are used to recover the round keys. The function KeyRecover uses, $HW_{K10}^t$ which is a possible Hamming weight sequence of $K_{10}$, Hamming weight of AES state $HW_S$ and CT as inputs to obtain the possible values of $K_{10}$. The function KeyVerify verifies the correctness of $K_{10}$ with the byte-wise Hamming weight of other 10 round keys. Algorithm 1 outlines the process of recovering AES round keys.

**Algorithm 1:** Key Recovery Algorithm with AES State Hamming Weight

**Input:** $HW_S$, $HW_K$, CT
**Output:** $K$

1: $HWSET_{K10} \leftarrow GetHW(HW_K)$
2: for all $HW_{K10}^t \in HWSET_{K10}$ do
3: $K^t \leftarrow KeyRecover(HW_S, HW_{K10}^t, CT)$
4: $K^t \leftarrow KeyRecover(HK_{10}^t, CT\ HW_K)$
5: end for

In function KeyRecover, it should be noticed that not all 16 Hamming weight of state $S_{10}^{SR}$ are necessary in recovery process. We consider the situation that attackers probably only get a part of Hamming weight of state $S_{10}^{SR}$. Thus, first the number and position of the obtained Hamming weight of $S_{10}^{SR}$ should be confirmed. For every confirmed position of the state, the corresponding position's possible value set of $K_{10}$ could be initially determined with its Hamming weight. With the relationship $HW(CT_i \oplus K_{10,i}) = HW(SR(SB(S_{10}^{SR})))$, we could further shrink the value set of $K_{10}$. Compared with the "brute force" technique, the value set is relatively acceptable to deal with. In order to further improve the computational efficiency, in function KeyVerify we use the optimized sequence, *i.e.* $k_{10,0}$, $k_{10,4}$, $k_{10,8}$,…, $k_{10,9}$, $k_{10,13}$, to verify the correctness of each possible value of $K_{10}$. When guessing each byte of $K_{10}$ in the possible value set, the optimized sequence could maximize the number of computable bytes and therefore, more quickly to remove false key values.

## 4. EXPERIMENTAL VERIFICATION

Our experiments are implemented on a normal PC with Intel Core I3-3220 CPU and 4.0 GB of RAM. We simulated the situation of getting 1 power trace and 2 power traces, separately. Note that, we consider 2 traces since at least two AES executions are necessary to get the differential when fault injection is used to accelerate the key recovery. Our experiments verify the relationship between the key recovery results and the number of key bytes whose Hamming weight are used in the key recovery. The key recovery algorithm is written in python. The keys are randomly generated and 176 Hamming weight of round key bytes are used as the input to the key recovery algorithm. Table 2 summarizes the result of the performed key recovery simulations. Note that the last 5 lines of Table 2 are the fault injection attack results from [9] for comparison.

**Table 2. Result of fast key recovery without fault injection.**

| No. of Traces | No. of Bytes | No. of runs(N) | Simulation timeout(s) | Percentage finished runs | Average time(*s*) | Average residual entropy(bits) |
|---|---|---|---|---|---|---|
| 2 | 16 | 500 | 600 | 100 | 37.33 | 0.009 |
| 2 | 15 | 500 | 600 | 99.8 | 55.98 | 0.014 |
| 2 | 14 | 500 | 600 | 99.8 | 56.96 | 0.012 |
| 2 | 13 | 500 | 600 | 99.6 | 56.01 | 0.012 |
| **2** | **12** | **500** | **600** | **99.6** | **59.08** | **0.13** |
| 2 | 11 | 100 | 2000 | 95 | 314.83 | 0.12 |
| 2 | 10 | 100 | 2000 | 91 | 308.85 | 0.15 |
| 2 | 9 | 100 | 2000 | 91 | 363.98 | 0.15 |
| 2 | 8 | 100 | 2000 | 86 | 382.80 | 0.19 |
| 1 | 16 | 100 | 2000 | 90 | 519.05 | 0.08 |
| 1 | 15 | 100 | 2000 | 48 | 860.31 | 0.12 |
| 1 | 14 | 100 | 2000 | 39 | 892.97 | 0 |

| No. if Fault Injection in [9] | Average time(s) |
|---|---|
| 0 fault, 1 AES execution | 7200 |
| 1 fault, at least 2 AES execution | 1200 |
| 5 fault, at least 6 AES execution | 300 |
| 10 fault, at least 11 AES execution | 180 |
| 20 fault, at least 21 AES execution | 120 |

As shown in Table 2, we set the simulation timeout as 600 or 2,000 seconds. Each line represents the key recovery result for an attack scenario. For example, line 1 means that 500 times of runs on 16 bytes for 2 traces are executed, 100% of them terminated in 600 seconds and the average run time of terminated runs is 37.33 seconds. Similarly, with 15 Hamming weight of state bytes extracted, the recover success rate decreases to 99.8% and the average of recovery time rises to 55.98 seconds. When the knowing state bytes decrease to 8, the recover success rate decreases to only 86% and the average of recovery time rises to 382.80 seconds. When only 1 power trace is used, the success rate and average time of 16 bytes are 90% and 519.05 seconds, respectively.

The experiment results show that 2 traces will extremely increase the efficiency of key recovery. Among all simulations, getting 12 bytes for 2 traces is a good option for the attackers, which uses relatively less number of key bytes to achieve a high success rate, a low running time and acceptable residual entropy. Compared with the fault injection attack in [9], we can see that getting 2 traces 10 bytes of our attack pattern is close to 5 fault injections of [9]. Getting 12 bytes for 2 traces in our attack pattern is much better than all the attack result in [9].

## 5. ALGEBRAIC SIDE CHANNEL ATTACK AGAINST AES KEY EXPANSION

### 5.1 Motivation

In this section, we explain our effects toward a general security evaluation platform for side channel attack countermeasures in AES key expansion. In [9], the author proposed and evaluated several reasonable countermeasures. However, all of them are not secure enough. Thus, new countermeasures should be considered. Both Boolean masking and random order could easily be transformed to many possible variations, *e.g.* to use more randomness. Accordingly, the accurate security evaluation against their overhead for all the possible countermeasures become a difficult work to achieve.

The algebraic side channel attack (ASCA) is an important attack method, which is the combination of algebraic attacks and side channel attacks [16]. ASCA uses the side channel leakage (for example, Hamming weight, Hamming distance, *etc.*) and algebraic presentation of the cipher to feed the SAT solver for the key recovery. Since the key recovery is fully automatic without human factor, we consider it is a general and fair approach to evaluate various possible countermeasures.

We consider to use algebraic side channel attack (ASCA) as a tool to achieve the general security evaluation. ASCA is an important attack method, which is the combination of algebraic attacks and side channel attacks. ASCA has been applied to AES in many papers [17-20]. However, to the best of our knowledge, this work is the first one to apply ASCA to specifically the AES key expansion. After feed the solver with the algebraic representations of the cipher and the leakage, the key recovery of ASCA is fully automatic without human interference. Thus ASCA could become a general and fair approach to evaluate various possible countermeasures. In this work, we explain the verification of a successful key recovery in a basic attack scenario where no countermeasures are used.

### 5.2 ASCA Attack Design against AES Key Expansion

Fig. 7 shows the process of a key recovery experiment we performed in this work. The whole process can be divided into 3 steps. First, obtain the algebraic representation of AES key expansion. Second, obtain the algebraic representation of the side-channel leakage, *i.e.* the Hamming weight of key bytes. Third, using the SAT solver to solve the unknown variables, *i.e.* the secret key, in all the equations.

To obtain the algebraic representation of AES key expansion, the only difficult part is the AES S-box for its non-linearity. In order to obtain the nonlinear equations of S-box, we will start with C language lists all 256 equations. We use symbolic computation to generate 256 non-linear equations with 256 unknown coefficients, which is equivalent to the AES S-box. In our work, we use CryptoMiniSat as the SAT solver for its support to XOR calculations and strong solve ability [21]. In order to use CryptoMiniSat to determine all the coefficients, all the equations of AES S-box is converted to a standard SAT solver format named DIMACS. Finally, CryptoMiniSat gives the solutions of all the coefficients. It is much easier to obtain the algebraic representations for the rest calculations in AES key expansion.

Then, the leakage from the side-channel measurement, *i.e.* the Hamming weight of each key byte can be transformed to Hamming weight equations as well. In this experiment, we consider the Hamming weights of all the key bytes are known to attackers. For each byte, the Hamming weight is a 4-bit binary value, which can be presented as the formats using the 8-bit digits in the original value as the input.

All previously obtained equation are then transformed into a conjunctive normal and XOR equations, and finally to DIMACS format. After conversion, the entire algebraic representation system has 1408 1st order key bits, 4928 2nd order items, 2240 3rd order items, 12320 4th order items, 2240 5th order items, 1120 6th order items and 320 7th order items.

After all format conversion equation described above, all written to a file to feed the CryptoMiniSat to solve the recovered key. Hamming weight of all the key bytes together with the equation set of AES key expansion is feed to CryptoMiniSat to verify the successful key recovery. The key recovery experiment is performed on a laptop with the Intel Core 2 CPU and 3Gb of RAM. The SAT solver we used is CryptoMiniSat4.5.3 running in Linux Ubuntu 14.0.03 as the operating system.

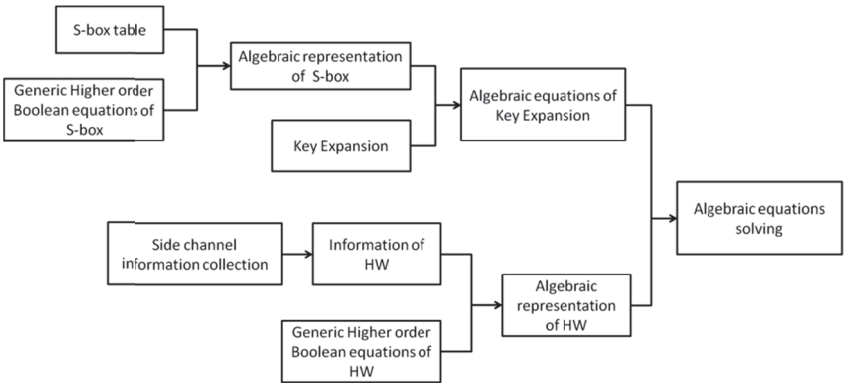


Fig. 7. Key recovery of algebraic SCA against AES key expansion.

### 5.3 ASCA Attack Result and Analysis

For each randomly selected key, *e.g.* 0x2b7e151628aed2a6abf7158809cf4f3c, we first confirm the successful key recovery using all the Hamming weight information. Then we gradually reduce the amount of the known Hamming weight round by round, to verify the key recovery and the calculated time. We tested randomly 10 keys, and the experiment result is shown in Table 3.

**Table 3. Result of key recovery solving.**

| Round No. / Exper No. | Round 11 | Round 10 | Round 9 | Round 8 | Round 7 | Round 6 | Round 5 | Round 4 |
|---|---|---|---|---|---|---|---|---|
| 1 | 14.57s | 16.02s | 5.81s | 5.15s | 8.27s | 15.64s | – | – |
| 2 | 38.22s | 13.50s | 9.41s | 23.12s | 9.43s | 7.47s | – | – |
| 3 | 48.43s | 12.50s | 11.07s | 39.24s | 8.99s | 32.39s | – | – |
| 4 | 10.01s | 4.75s | 10.50s | 4.89s | 5.66s | 5.30s | 5.11s | 7.13s |
| 5 | 30.11s | 9.77s | 13.07s | 13.88s | 9.54s | 30.48s | 15.76s | – |
| 6 | 12.30s | 6.62s | 29.30s | 11.46s | 32.81s | 33.61s | 32.50s | – |
| 7 | 14.98s | 33.75s | 14.42s | 9.14s | 9.65s | 16.06s | 39.31s | – |
| 8 | 14.97s | 5.64s | 6.88s | 6.01s | 30.87s | 12.19s | – | – |
| 9 | 9.82s | 11.59s | 7.34s | 7.67s | 38.58s | 13.81s | 41.02s | – |
| 10 | 9.69s | 8.21s | 12.62s | 10.05s | 7.94s | 9.62s | 8.33s | 7.98s |

−: recovery failed

Our experiment found that at least leakage from 4 round keys are required for the successful key recovery. After obtaining the leakage of 6 round keys, all the tested key can be successfully recovered. Our experiments successfully verified the feasibility of the application of algebraic side-channel attack to the AES key expansion.

A special case of the power analysis on AES key expansion is that multiple keys have the exactly the same side-channel leakage. For SAT solver, as long as it obtains a solution to meet all the conditions, the solving will stop. In order to get all the solutions from the SAT solver, one can run the SAT solver multiple times until the key recovery is failed. For each found key, it will be denoted as an incorrect solution in the next run. By a simple experimentation, we verified the mentioned method can recover the possible multiple solutions.

This part of work applied ASCA to key recovery attack on AES key expansion. With the known Hamming weight of each round key byte, we verify the key recovery is easy to achieve. The future work is combine the countermeasures into the key solution problem and use ASCA as a general platform for evaluating the variations of different countermeasures.

## 6. CONCLUSION

This paper focused on the key recovery problem for AES key expansion when random order countermeasure is applied. In [9], the random order countermeasure shows

certain resistance against the basic power analysis. Then, fault injection is used together with power analysis to further discuss the security of random order countermeasure. As a more practical approach, we consider that the attackers could use power traces to get a few Hamming weight of an AES state to accelerate the power analysis. This work studied the most possible strategy of the attack and quantitatively evaluated the relationship between the additional information from the AES state and the key recovery efficiency. When the attacker could get Hamming weights of 12 bytes for 2 AES traces, a very fast key recovery can be achieved with a high success rate even though no fault injection is required.

Second, we considered using the algebraic side channel attacks as a general method for evaluating various parameters in countermeasures. As the first step, this paper studied the principle and application of the algebraic attack, and verified the possibility of the application of the algebraic side channel attack in the AES key expansion analysis. Our result shows that in minimum only leakage from four rounds keys can successfully recover the key. The future work is to combine the countermeasures into the key solution problem and use ASCA as a general platform for evaluating the variations of different countermeasures.

## REFERENCES

1. J. Goodwin and P. R. Wilson, "Advanced encryption standard (AES) implementation with increased dpa resistance and low overhead," in *Proceedings of IEEE International Symposium on Circuits and Systems*, 2008, pp. 3286-3289.
2. G. Piret and J. J. Quisquater, "A differential fault attack technique against SPN structures, with application to the AES and KHAZAD," in *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems*, LNCS, Vol. 2779, pp. 77-88.
3. National Institute of Standards and Technology: Advanced Encryption Standard, NIST FIPS PUB 197, 2001.
4. P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proceedings of Annual International Cryptology Conference*, LNCS, Vol. 1666, 1999, pp. 388-397.
5. E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems*, LNCS, Vol. 3156, 2004, pp. 16-29.
6. C. Herbst, E. Oswald, and S. Mangard, "An AES smart card implementation resistant to power analysis attacks," in *Applied Cryptography and Network Security*, Springer, 2006, pp. 239-252.
7. S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems*, 2002, pp. 13-28.
8. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, "Mutual information analysis," in *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems*, LNCS, Vol. 5154, 2008, pp. 426-442.
9. C. Clavier, D. Marion, and A. Wurcker, "Simple power analysis on AES key expansion revisited," in *Proceedings of International Workshop on Cryptographic Hard-*

*ware and Embedded Systems Cryptographic Hardware and Embedded Systems*, 2014, pp. 279-297.
10. R. Mayer-Sommer, "Smartly analyzing the simplicity and the power of simple power analysis on smartcards" in *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems*, 2000, pp. 78-92.
11. E. Biham and A. Shamir, "Power analysis of the key scheduling of the AES candidates," in *Proceedings of the 2nd AES Candidate Conference*, 1999, pp. 115-121.
12. T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Investigations of power analysis attacks on smartcards," *USENIX Workshop on Smartcard Technology*, 1999, pp. 151-161.
13. S. Mangard, "A simple power-analysis (spa) attack on implementations of the AES key expansion," in *Proceedings of International Conference on Information Security and Cryptology*, 2002, pp. 343-358.
14. J. VanLaven, M. Brehob, and K. J. Compton, "A computationally feasible spa attack on AES via optimized search," in *Security and Privacy in the Age of Ubiquitous Computing*, Springer, 2005, pp. 577-588
15. S. S. Ali and D. Mukhopadhyay, "A differential fault analysis on AES key schedule using single fault," in *Proceedings of IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2011, pp. 35-42.
16. M. Renauld, "Algebraic side-channel attacks," in *Proceedings of International Conference on Information Security and Cryptology*, 2010, pp. 393-410.
17. Y. Oren, M. Renauld, and F.-X. Standaert, "Algebraic side-channel attacks beyond the hamming weight leakage model," in *Proceedings of the 14th International Conference on Cryptographic Hardware and Embedded Systems*, 2012, pp. 140-154.
18. Z. Guo, *et al.*, "Exploiting the incomplete diffusion feature: A specialized analytical side-channel attack against the AES and its application to microcontroller implementations," *IEEE Transactions on Information Forensics and Security*, Vol. 9, 2014, pp. 999-1014.
19. M. S. E. Mohamed, S. Bulygin, M. Zohner, A. Heuser, M. Walter, and J. Buchmann, "Improved algebraic side-channel attack on AES," in *Proceedings of IEEE International Symposium on Hardware-Oriented Security and Trust*, 2012, pp. 146-151.
20. M. S. E. Mohamed, S. Bulygin, M. Zohner, A. Heuser, M. Walter, and J. Buchmann, "Improved algebraic side-channel attack on AES," *Journal of Cryptographic Engineering*, Vol. 3, 2012, pp. 139-156.
21. CryptominiSAT2, http://www.msoos.org/cryptominisat2/.

**Yang Li (李阳)** received his B.E. degree in Electronic and Information Engineering from Harbin Engineering University, the M.E. degree in Information and Communication Engineering and the Ph.D. degree in faculty of Informatics and Engineering from the University of Electro-Communications, Tokyo. He is currently an Associate Professor in College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China. His main research interest includes security evaluation and improvement for cryptographic implementation in hardware and embedded systems.

**MengTing Chen (陈梦珽)** received her B.E. degree in College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China. She is now a master student in Software Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, China.

**Jian Wang (王箭)** received his Ph.D. degree from Department of Computer Science and Computer Application Technology of Nanjing University He is a Postdoctoral Fellow from 2001 to 2003 in University of Tokyo in 2004. He is now a Professor in College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. His main research interest includes key management, cryptographic protocols, privacy protection, malicious tracking, security assessment.