Exploiting the Incomplete Diffusion Feature: A Specialized Analytical Side-Channel Attack against the AES and its Application to Microcontroller Implementations

Shize Guo, Xinjie Zhao, Fan Zhang, Tao Wang, Zhijie Shi, Francois-Xavier Standaert, Chujiao Ma.

Abstract—Algebraic side-channel attack (ASCA) is a typical technique that relies on a general solver to solve the equations of a cipher and its side-channel leaks. It falls under analytical side-channel attack and can recover the entire key at once. Many ASCAs are proposed against the AES, and utilize the Gröbner basis-based, SAT-based or optimizer-based solver. The advantage of the general solver approach is its generic feature, which can be easily applied to different cryptographic algorithms. The disadvantage is that it is difficult to take into account the specialized properties of the targeted cryptographic algorithms. The results vary depending on what type of solver is used, and the time complexity is quite high when considering the error-tolerant attack scenarios. Thus, we were motivated to find a new approach that would lessen the influence of the general solver and reduce the time complexity of ASCA. This paper proposes a new analytical side-channel attack on AES by exploiting the *incomplete diffusion feature* in one AES round. We named our technique *incomplete diffusion analytical side-channel analysis (IDASCA)*. Different from previous ASCAs, IDASCA adopts a specialized approach to recover the secret key of AES instead of the general solver. Extensive attacks are performed against the software implementation of AES on an 8-bit microcontroller. Experimental results show that: 1) IDASCA can exploit the side-channel leaks in all AES rounds using a single power trace; 2) It has less time complexity and more robustness than previous ASCAs, especially when considering the error-tolerant attack scenarios; 3) It can calculate the reduced key search space of AES for the given amount of side-channel leaks. IDASCA can also interpret the mechanism behind previous ASCAs on AES from a quantitative perspective, such as, why ASCA can work under unknown plaintext/ciphertext scenarios and what are the extreme cases in ASCAs.

Index Terms—Incomplete diffusion, algebraic side-channel attack, AES, Hamming weight leakage model, microcontrollers.

1 Introduction

Security evaluation of cryptographic algorithms can be considered from two aspects. The algorithms can be considered from the point of view of mathematical security such as differential [4], linear [19], algebraic cryptanalysis [3], [10], [11] and their variants. And their implementations can be considered from the point of view of physical security (which makes any analysis quite device-dependent) with side channel-analysis (SCA) techniques where physical leakages from the target devices, such as execution time [16], power consumption [17], and electromagnetic emissions [25], are exploited to break the algorithms. Embedded systems are most vulnerable to SCAs as attackers often have direct physical accesses.

Typical SCA techniques include simple power analysis (SPA) [18], differential power analysis (DPA) [17], correlation power analysis (CPA) [7], mutual information analysis (MIA) [15], template analysis (TA) [9], stochastic SCA (SSCA) [31], side-channel cube analysis (SCCA) [13], algebraic side-channel collision analysis (ASCCA) [5] and algebraic SCA (ASCA) [8], [20], [21], [23], [26], [27], [36]. All these attacks exploit some model of the physical leakages to be compared with actual measurements. Assumptions regarding the model lead to two important classes of SCAs: profiled and non-profiled. A profiled SCA is composed of two phases: a profiling phase where the adversary is provided with a training device under test (TRDUT) that allows him to characterize physical leakages (in order to obtain a precise leakage model); and an online exploitation phase where the attack is mounted against a similar target device under test (TADUT) to perform a secret key extraction. Non-profiled SCA only requires the latter phase and assumes some (less precise) leakage model, typically obtained from engineering intuition. With respect to the key recovery procedure, SCAs fall into two categories:

Shize Guo is with the Institute of North Electronic Equipment, Beijing, China. E-mail: tigerone-gsz@vip.sina.com.
Xinjie Zhao is with the Institute of North Electronic Equipment, Beijing and the Department of Information Engineering, Ordnance Engineering College, Hebei, China. E-mail: zhaoxinjieem@163.com
Fan Zhang is with the Department of Information Science & Electrical Engineering, Zhejiang University, China. E-mail: fanzhang@zju.edu.cn
Zhiije Shi and Chujiao Ma are with the Department of Computer Science & Engineering, University of Connecticut, Storrs, Connecticut, USA. E-mail: {zshi,chujiao.ma}@engr.uconn.edu
Tao Wang is with the Department of Information Engineering, Ordnance Engineering College, Hebei, China. E-mail: twangdrsjz@yahoo.com.cn.
Francois-Xavier Standaert is with the UCL Crypto Group, Belgium. E-mail: fstandae@uclouvain.be

TABLE 1
SCAs classified by the leakages measurement and key recovery procedure.

SCA type	Non-profiled SCA	Profiled SCA
divide-and-conquer SCA	DPA [17], CPA [7], MIA [15]	SPA [18], TA [9], SSCA [31], SCCA [13]
analytical SCA	ASCCA [5]	ASCA [8], [20], [21], [23], [26], [27], [36]

divide-and-conquer SCA (which provide distinguishers for small key chunks that are then combined, e.g., using key enumeration [34]) and analytical SCA (which recover the entire key at once, e.g., by solving systems of equations). As a result, SCAs can be divided into the four types represented in Table 1. It should be noted that the list in Table 1 is admittedly incomplete (these are just examples of SCAs).

Analytical SCA is currently a very active area in the crypto community. Traditional SCAs [7], [9], [15], [17], [31] exploit a divide-and-conquer strategy and recover several pieces of a secret key independently. For analytical SCA, both the cipher and the leakages are represented with algebraic equations and the full secret key is recovered at once by solving these equations with different strategies. Since leakages of more rounds can be utilized, this attack has less measurement complexity than traditional SCAs.

The idea of analytical SCA is first introduced by Bogdanov *et al.* in 2007 [5] to attack the software implementation of AES on 8-bit microcontroller, where collision-based SCA [30] is combined with algebraic cryptanalysis. The attack is named algebraic side-channel collision analysis (ASCCA). In ASCCA, the adversary detects the internal collisions (if the values of two intermediate states are equal) in two AES rounds by comparing the patterns of the two sections of the power traces and then converts them into equations. The F4 Gröbner basis-based algorithm in MAGMA solver [14] is used to solve the equations. Under known plaintext scenario, ASCCA only requires five power traces to recover the master key of AES. The attack is independent of the leakage model.

In INSCRYPT 2009, Renauld *et al.* combined template attack with algebraic cryptanalysis [3], [10], [11] and proposed the algebraic side-channel attack (ASCA) against the software implementation of PRESENT on an 8-bit microcontroller [26]. In ASCA, template attack is used to deduce the Hamming weight (HW) or the accurate value of intermediates states. This can be done by detecting the external collisions between the targeted power trace in TADUT with the template power trace in TRDUT. The algebraic technique is used to represent both the cipher and the deductions. A SAT solver, ZChaff [24], is used to recover the secret key. In CHES 2009, Renauld *et al.* successfully extended ASCA to the software implementation of AES on an 8-bit microcontroller [27]. Compared with ASCCA and other SCA techniques, ASCA can exploit the side-channel leakages in all cipher rounds and can recover the key with a single trace even when both the plaintexts and ciphertexts are unknown [26], [27].

ASCA works well on the software implementation of AES [12] on an 8-bit microcontroller under the Hamming weight leakage model (HWLM) [27]. Recently, it has also been successfully applied to the hardware implementation of AES on a 65nm ASIC, under the template leakage model (TLM) [23] with a single power trace. The work in [28] studies the impact of representation dependence, leakage dependence and cipher dependence to ASCA. The work in [8] studies the impact of algebraic immunity to the resistance of block ciphers against ASCA. The original ASCAs assume that the correct deduction on the Hamming weight (HW), or the accurate value of intermediates states, can be profiled from analyzing the side-channel leakages. In practice, it was observed that noise is the main issue for robust ASCA. Because of this, multiple deductions have to be obtained from the leakage and utilized in the attack. To mitigate this issue, two types of solutions are provided. One solution is to group these multiple deductions together into sets, then convert them into algebraic equations. A SAT solver is normally used to recover the secret key. In this approach, there are many variants, such as multiple deductions-based ASCA (MDASCA) in COSADE 2012 [36] and improved ASCA (IASCA) in HOST 2012 [20]. The other solution is to include the imprecise deductions in the equation set and to deal with these imprecisions via an optimizer (e.g., the SCIP solver [1]). This technique is denoted as Tolerant ASCA (TASCA) in CHES 2010 [21] and in Eprint 2012/092 [22]. In [8], [20], [21], [27], [36], the probabilities of multiple deductions are treated as equal, which may lose some useful information. In CHES 2012, TASCA is further modified to cope with the different probabilities of multiple deductions named probabilistic TASCA (Prob-TASCA) [23]. Prob-TASCA can regain some information lost in other attacks.

In summary, most existing ASCAs adopt a general, sometimes off-the-shelf equation solver (e.g., the F4 Gröbner basis-based algorithms in MAGMA solver [8], SAT solver [26], [27], [20], [36], mixed integer programming solver [21], [22], [23]). The advantage of the general solver approach is its generic feature, which can be easily applied to different cryptographic algorithms. The disadvantage is that it is difficult to take into account the specialized structures or properties of the targeted cryptographic algorithms. The results vary depending on what type of solver was used, and the time complexity of the general solver is quite high when considering the error-tolerant attack scenario in ASCA on AES. When there is too much noise and the deduction sets are too large, there exist too many solutions for the equation system. If only the plaintext is included in the equation set, the general equation solver might output a satisfied or optimized solution but not the correct one, which reduces the success rate. If both the plaintext and ciphertext are included in the equation set, the output solution should be correct but the solver may not output the correct solution in a reasonable amount of time. Meanwhile, the exact reduced key search space of AES for the given amount of leakages is not studied in previous ASCAs [8], [20], [21], [22], [23], [27], [36].

It is critical to find a new approach that would lessen the influence of the solver and reduce the time complexity of existing ASCAs on AES, especially when considering the error-tolerant attack scenario. These are the motivations of this paper. Our main idea is inspired by the simple power attack technique in ICISC 02 [18] and the low data complexity attack technique in CRYPTO 11 [6]. The work in [18] utilized the incomplete diffusion feature in the AES key expansion [1] to recover the secret key of AES with a single power trace. The work in [6] utilized the customized solver approach instead of the general equation solver to solve the equations of Round-Reduced AES. It is interesting to exploit the incomplete diffusion feature in the AES encryption procedure [2] and utilize a specialized approach (construct a customized or specialized solver) instead of the general equation solver to improve ASCA. Since there are more leakages in the AES encryption procedure, the attack might work under unknown plaintext and ciphertext scenario. Meanwhile, as the incomplete diffusion feature is considered, this specialized attack may achieve better performance than existing ASCAs. We name our technique *incomplete diffusion analytical side-channel analysis (IDASCA)*.

Specifically, the main principle of IDASCA on AES is listed as follows.

1) For each AES round, considering the incomplete diffusion feature in one round, four state bytes of the round output are computed by transforming the fixed four state bytes of the AddRoundKey output with SubBytes, ShiftRows, MixColumns [3] operations.
2) Thus, the sixteen state bytes of the round output and the AddRoundKey output in each round can be divided into four state groups, each state group containing four bytes.
3) The 84 deduction sets on the intermediates states in one AES round can be profiled from analyzing the side-channel leakages. These deduced sets can also be divided into four groups. Each deduction set group contains 21 leaks in computing the four state bytes of the round output from the four state bytes of the AddRoundKey output.
4) By enumerating the 2^{32} candidates of one state group of the AddRoundKey output, the deduction on the 21 leaks can be predicted. For each candidate, it can be kept only when the value of each deduction in the predicted deduction group is among the practical deduction set group. Otherwise, the candidate can be discarded. Then, limited candidates of one state group of the AddRoundKey output can be kept.
5) By applying the above procedure to other three state groups, limited candidates of the AddRoundKey output can be calculated. Then, the same amount of candidates of the round output can also be computed.
6) The candidates of the i-th round-key in each AES round can be calculated by XORing the $i-1$-th round output with the i-th round AddRoundKey output.
7) The master key can be retrieved by analyzing the candidates of all the round-keys.

Extensive experiments of IDASCA are conducted against the software implementation of AES on an 8-bit microcontroller under the HWLM. The results show that IDASCA has the following features. As in previous ASCAs, IDASCA can exploit the leaks in all AES rounds and work even when both the plaintext and ciphertext are unknown. Compared with previous ASCAs on AES, IDASCA has lower attack complexities and better error tolerant capabilities. IDASCA can calculate the accurate reduced key search space of ASCA on AES, which interprets the results of previous ASCAs (e.g., the original ASCA [27], MDASCA [36], IASCA [20], TASCA [22], Prob-TASCA [23]) from a quantitative perspective. It can also be used to explain why ASCA can work under unknown plaintext/ciphertext scenario and what are the extreme scenarios in previous ASCAs.

The rest of this paper are organized as follows. Section 2 describes the design and the targeted implementation of AES. Section 3 shows how to predict the deduction set for different leaks with template attack technique. Section 4 presents the details of IDASCA and how to use it to recover the secret key of AES by analyzing the deduction set. Section 5 provides the experimental results of IDASCA on AES. Section 6 interprets the results of different previous ASCAs on AES with IDASCA. Section 7 concludes the paper.

2 Analysis on the Design and Implementation of AES

2.1 Description of AES

The full description of the AES algorithm can be referred to in [12]. The typical AES with 128-bit key length, AES-128, is described in this subsection. Both the 128-bit input plaintext (P) and the secret key (K) are arranged as a 4×4 array of bytes, known as the state matrix, and referred to as follows.

1. The calculation on one byte of the i-th round key does not rely on all sixteen bytes of the $i-1$-th round key.

2. For this, the incomplete diffusion feature means that the calculation on one byte of the i-th round output does not rely on all sixteen bytes of the i-th round AddRoundKey output. Comparatively, the full diffusion can be achieved by two AES rounds, which means that the calculation on one byte of the i-th round output relies on all sixteen bytes of the $i-1$-th round AddRoundKey output.

3. The last AES round does not have the MixColumns operation.

$$P = \begin{pmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{pmatrix}, \quad K = \begin{pmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{pmatrix}.$$

In AES-128, the input state matrix (plaintext) is transformed through 10 round functions. The state matrix evolves as it passes through the various operations of the algorithm and finally emerges in the form of a ciphertext. The first nine rounds are composed of the following four operations and the last round is the same but without the MixColumns operation.

1) AddRoundKey (AK). Each byte of the state matrix is XORed with a byte from a corresponding matrix of round subkeys.
2) SubBytes (SB). Each byte of the state matrix is updated by a lookup table using the S-Box.
3) ShiftRows (SR). Row i of the state is rotated i-byte to the left, for i = 0, 1, 2, 3.
4) MixColumns (MC). Each column of the state is updated by multiplying itself with the corresponding column of a fixed matrix M.

2.2 Target Implementation of AES

As in previous work [20], [21], [26], [27], [36], our attack considers the typical C software implementation of AES [12] on an 8-bit microcontroller ATMEGA324P for both TRDUT and TADUT. We assumed that no leaks from the key expansion process of AES are available to the adversary, e.g., the device performs the round key expansion in advance. The details of our implementation can be referred to in Algorithm 1. S[] denotes the S-Box lookup function and xt denotes the $xtime$ function [12].

In this application, the HW of an intermediate byte can be leaked and there are nine candidates for each byte. In the attack, we exploited the HW leaks from the device when the 8-bit operands on its data bus are processed. In total, it corresponds to 100 leaks per round (as noted in [23]) described below.

1) AK operation leaks information about the 16 state bytes after the XOR with the key, as well as information about the key bytes themselves, providing a total of 32 leaks per round.
2) SB operation is implemented as a table look-up and leaks information about its 16 output byte per round.
3) SR operation does not leak any information.
4) MC operation is implemented with 8-bit XTIME and XOR operations as specified in [12]. It leaks 36 bytes of internal state and 16 bytes of final state, resulting in a total of 52 leaks per round.

We noted that some previous ASCAs in [20], [27], [36] only consider 84 leaks for one AES round, where the 16 leaks of loading the round key are omitted. The technique in analyzing 100 and 84 leaks is identical, except that the key search space of AES can be further reduced by analyzing the 16 extra leaks on the round key. We will address this in Section 4.

Algorithm 1. C implementation of AES

```
void AddRoundKey (int $n$){
  for $i$=0 to 15 do
    key[$i$]=rkey[$16 \times n + i$];  //16 leaks
    state[$i$]^=key[$i$];  //16 leaks
  end for
}
void SubBytes ( ){
  for $i$=0 to 15 do  //16 leaks
    state[$i$]=S[state[$i$]];
  end for
}
void ShiftRows ( ){
  for $i$=0 to 15 do
    temp[$i$]=state[$R[i]$];
  end for
  for $i$=0 to 15 do
    state[$j$]=temp[$j$];
  end for
}
void MixColumns ( ) {  //52 leaks
  for $i$=0, 4, 8, 12 do
    a=state[$i$]^state[$i+1$]; b=state[$i+1$]^state[$i+2$];  //$2 \times 4$ leaks
    c=state[$i+2$]^state[$i+3$]; d=state[$i$]^state[$i+3$];  //$2 \times 4$ leaks
    e=a^c;  //$1 \times 4$ leakages
    state[$i$]^=xt(a); state[$i$]^=e;  //$2 \times 4$ leaks
    state[$i+1$]^=xt(b); state[$i$]^=e;  //$2 \times 4$ leaks
    state[$i+2$]^=xt(c); state[$i$]^=e;  //$2 \times 4$ leaks
    state[$i+3$]^=xt(d); state[$i$]^=e;  //$2 \times 4$ leaks
  end for
}
void AES (byte *in, byte *rkey, byte *out) {
  for $i$=0 to 15 do
    state[$i$]=in[$i$];
  end for
  for $i$=0 to 8 do
    AddRoundKey($i$); SubBytes(); ShiftRows(); MixColumns();
  end for
    AddRoundKey(9); SubBytes(); ShiftRows(); AddRoundKey(10)
  for $i$=0 to 15 do
    out[$i$]=state[$i$];
  end for
}
```

3 Predict the Deduction Set with Template Attack

Using the same method as previous ASCAs [8], [20], [21], [22], [23], [27], [36], we first predict the deduction set for each side-channel leakage point. We use the template attack technique [9] to decode the power trace into a series of deduction sets for all the leakage points. Let b denote an intermediate byte, $L()$ denote the leakage function mapping the *intermediate state* b to the *deduction*, d denote the correct deduction (Hamming weight in HWLM or specific value in TLM) on b ($d = L(b)$), D denote the deduction set on d, and d_i denote the i-th deduction on d.

In the attack, we calculate the probabilities of every deduction by template attack [9] with two steps. In the first step, we profile nine templates for every HW leak in an *intermediate state*. Each template assumes a Gaussian noise and is characterized by a mean value and a noise standard deviation. In the second step, we use the Bayesian inversion to simulate the classification probability from the templates for each deduction. Next, we present several parameters used in the attack.

1) μ denotes the size of the deduction set for one leak, where the deduction set $D = \{d_1, \ldots d_\mu\}$.
   The value of μ can be determined as either a global constant (fixed μ) for all leaks in one attack (as in [22], [23], [27]), or on a per-leak basis (dynamic μ) according to some heuristics (e.g., choosing the highest n deductions when the sum of their probabilities is over a fixed threshold T, as in [20], [36]). Under the ideal case, the adversary can deduce the single and correct deduction for each leak and $\mu = 1$, as assumed in [27]. This representation provides the most information, but it cannot tolerate any errors. To tolerate errors, more deductions can be predicted and $\mu \geq 1$. In previous ASCAs, the work in [21], [22], [23] considered the fixed μ scenario for one attack. Using the same method, when the value of μ is large, the complexity of the attack increases exponentially. To overcome this, the work in [36] considered the dynamic μ scenario where $1 \leq \mu \leq 3$ under HWLM and the work in [20] also considered the dynamic μ scenario where $1 \leq \mu \leq 5$ under HWLM, both of which achieved lower attack complexity than that in fixed μ scenario.
2) λ_i denotes the probabilities of $d = d_i$.
   Two scenarios can be considered here. The first is to assume all the deductions have the same probability. Multiple deductions are represented as equations regardless of the different probabilities and input into the machine solver for key recovery (as in [20], [21], [22], [36]). The second is to consider different values of λ_i, encode them into equations and send them to an optimizer (e.g., SCIP solver), as done in prob-TASCA [23]. Since more information is exploited, prob-TASCA can regain some information lost in MDASCA [36], IASCA [20] and TASCA [21], [22]. Recently, the work in [34] also considered using the probabilistic (soft) information to optimize the key enumerating in DPA, CPA and other SCAs [4].
3) ρ_d denotes the decoding success rate, the proportion of traces for which the correct deduction of all the leaks are included in the deduction sets provided by the decoder.
   The value of ρ_d depends both on the deduction size μ and the noise. Under fixed μ scenario, the work in [21], [22], [23] showed that ρ_d would approach 100% when $\mu = 3$. Under dynamic μ scenario, the work in [20] showed that ρ_d would approach 99% when the threshold $T \geq 0.90$, where $1 \leq \mu \leq 4$.

4. Since the target list to be optimized is only the 16 key bytes in standard DPA attacks, the complexity of the key enumeration algorithm in [34] is directly affordable in this case. Exploiting it in the context of ASCA, where there are 100 intermediate bytes to be analyzed per round, is not straightforward - and we leave it as an interesting scope for further research.

4 Incomplete Diffusion Analytical Side-channel Analysis (IDASCA) on AES

When the measurement is too noisy and the deduction sets are too large (the value of μ is large), there are too many solutions for ASCA. For SAT solver based ASCA, the solver may output a satisfiable solution that is incorrect (if plaintext and ciphertext are known) or run for too long (otherwise). For optimizer-based ASCA, the SCIP solver may output an optimized solution that is also incorrect. In these cases, both attacks would fail, and how to conduct more efficient attacks still needs to be further studied. Meanwhile, how to accurately estimate the reduced key search space for a given amount of leakages in ASCA on AES is also an interesting problem.

This section proposes a new technique named *Incomplete Diffusion Analytical Side-channel Analysis (IDASCA)*. We first analyze the incomplete diffusion feature in one AES round and then describe the core of IDASCA on AES, which is composed of three steps: divide the states and leaks in each AES round, conquer the state from leaks in each AES round and search for the master key of AES.

4.1 The Incomplete Diffusion Feature in One AES Round

Let X^i, K^i, A^i, B^i, Y^i denote the 128-bit input, round-key, output of AK, output of SB and the final output of the i-th round, respectively. Let x^i_j, k^i_j, a^i_j, b^i_j, y^i_j, denote the j-th byte of X^i, K^i, A^i, B^i, Y^i ($0 \leq i \leq 9$, $0 \leq j \leq 15$). Eq. (1) presents the equation of how to calculate $\{y^i_0, y^i_1, y^i_2, y^i_3\}$ from x^i and the round-key K^i for the first nine rounds ($0 \leq i \leq 8$). As to the final round ($i = 9$), MC is omitted and Eq. (1) should be modified accordingly.

$$\begin{aligned}
y^i_0 &= 02 \cdot S[x^i_0 + k^i_0] + 03 \cdot S[x^i_5 + k^i_5] + S[x^i_{10} + k^i_0] + S[x^i_{15} + k^i_{15}] \\
y^i_1 &= S[x^i_0 + k^i_0] + 02 \cdot S[x^i_5 + k^i_5] + 03 \cdot S[x^i_{10} + k^i_0] + S[x^i_{15} + k^i_{15}] \\
y^i_2 &= S[x^i_0 + k^i_0] + S[x^i_5 + k^i_5] + 02 \cdot S[x^i_{10} + k^i_{10}] + 03 \cdot S[x^i_{15} + k^i_{15}] \\
y^i_3 &= 03 \cdot S[x^i_0 + k^i_0] + S[x^i_5 + k^i_5] + S[x^i_{10} + k^i_{10}] + 02 \cdot S[x^i_{15} + k^i_{15}]
\end{aligned} \tag{1}$$

In Eq. (1), $\cdot$ denotes the Finite field multiplication in $GF(2^8)$ that corresponds with the multiplication of polynomials modulo an irreducible polynomial $x^8 + x^4 + x^3 + x + 1$; $+$ denotes the XORed function; $S[\ ]$ denotes the S-Box lookup function.

From Eq. (1), we can see that, in order to calculate one byte of Y^i, not all bytes of A^i need to be known and only four bytes of A^i are required. We call this the *incomplete diffusion feature*. Take the calculation of $\{y^i_0, y^i_1, y^i_2, y^i_3\}$ as an example, any byte in $\{y^i_0, y^i_1, y^i_2, y^i_3\}$ is calculated by $\{a^i_0, a^i_5, a^i_{10}, a^i_{15}\}$ (the XORed result between $\{x^i_0, x^i_5, x^i_{10}, x^i_{15}\}$ and $\{k^i_0, k^i_5, k^i_{10}, k^i_{15}\}$), which are only four bytes of A^i.

According to this feature and the C implementation in Algorithm 1, there are only 21 leaks in the calculation procedure of $\{y^i_0, y^i_1, y^i_2, y^i_3\}$ from $\{a^i_0, a^i_5, a^i_{10}, a^i_{15}\}$. These 21 leaks can be used to search for $\{a^i_0, a^i_5, a^i_{10}, a^i_{15}\}$.

4.2 Divide the states and leaks in Each AES Round

This section presents how to divide the state and 84 leaks in each AES round into several groups by exploiting the incomplete diffusion feature. First, we present some notations and definitions.

State group. Define $\beta_0 = \{y_0^i, y_1^i, y_2^i, y_3^i\}$ as a state group in Y^i and $\alpha_0 = \{a_0^i, a_5^i, a_{10}^i, a_{15}^i\}$ as a state group in A^i. Each state group contains four bytes. Similarly, there are three other state groups in Y^i, which are $\beta_1 = \{y_4^i, y_5^i, y_6^i, y_7^i\}$, $\beta_2 = \{y_8^i, y_9^i, y_{10}^i, y_{11}^i\}$, $\beta_3 = \{y_{12}^i, y_{13}^i, y_{14}^i, y_{15}^i\}$. There are three other corresponding state groups in A^i, which are $\alpha_1 = \{a_4^i, a_9^i, a_{14}^i, a_3^i\}$, $\alpha_2 = \{a_8^i, a_{13}^i, a_2^i, a_7^i\}$,$\alpha_3 = \{a_{12}^i, a_1^i, a_6^i, a_{11}^i\}$.

State group mapping. We define the calculation procedure of one state group in β_i from the state group in α_i as the state group mapping, which can be denoted as $\alpha_i \rightarrow \beta_i$.

Leak group. According to the software implementation of AES in Algorithm 1, there are 21 leaks in one state group mapping, which can form a leak group and be denoted as $L_{\alpha_i \rightarrow \beta_i}$.

Below lists the 21 leaks in calculating β_0 from α_0.

1) 4 leaks in AK when calculating $a_0^i, a_5^i, a_{10}^i, a_{15}^i$.
2) 4 leaks in SB when calculating $b_0^i = S[a_0^i], b_5^i = S[a_5^i], b_{10}^i = S[a_{01}^i], b_{15}^i = S[a_{15}^i]$.
3) 13 leaks in MC when calculating $b_0^i + b_5^i, b_5^i + b_{10}^i, b_{10}^i + b_{15}^i, b_0^i + b_{15}^i, b_0^i + b_5^i + b_{10}^i + b_{15}^i, b_0^i + xt(b_0^i + b_5^i), b_0^i + xt(b_0^i + b_5^i) + b_0^i + b_5^i + b_{10}^i + b_{15}^i, b_5^i + xt(b_5^i + b_{10}^i), b_5^i + xt(b_5^i + b_{10}^i) + b_0^i + b_5^i + b_{10}^i + b_{15}^i, b_{10}^i + xt(b_{10}^i + b_{15}^i), b_{10}^i + xt(b_{10}^i + b_{15}^i) + b_0^i + b_5^i + b_{10}^i + b_{15}^i, b_{15}^i + xt(b_0^i + b_{15}^i), b_{15}^i + xt(b_0^i + b_{15}^i) + b_0^i + b_5^i + b_{10}^i + b_{15}^i$.

The Leak group $L_{\alpha_0 \rightarrow \beta_0} = \{L(a_0^i), L(a_5^i), L(a_{10}^i), L(a_{15}^i), L(b_0^i), L(b_5^i), L(b_{10}^i), L(b_{15}^i), L(b_0^i + b_5^i), \ldots, L(b_{15}^i + xt(b_0^i + b_{15}^i) + b_0^i + b_5^i + b_{10}^i + b_{15}^i)\}$.

Deduction set group. The group of all the deduction set on the Leak group $L_{\alpha_i \rightarrow \beta_i}$, can be denoted as $D_{\alpha_i \rightarrow \beta_i}$.

Then $D_{\alpha_0 \rightarrow \beta_0} = \{D(a_0^i), D(a_5^i), L(a_{10}^i), D(a_{15}^i), D(b_0^i), L(b_5^i), D(b_{10}^i), D(b_{15}^i), D(b_0^i + b_5^i), \ldots, D(b_{15}^i + xt(b_0^i + b_{15}^i) + b_0^i + b_5^i + b_{10}^i + b_{15}^i)\}$.

Below describes how to divide the states (A^i and Y^i) and 84 leaks in each AES round into groups.

Step 1.1. Divide A^i into four state groups, $\alpha_0, \alpha_1, \alpha_2, \alpha_3$, Y^i into four state groups, $\beta_0, \beta_1, \beta_2, \beta_3$.

Step 1.2. Divide the 84 leaks in each AES round into four groups,$L_{\alpha_0 \rightarrow \beta_0}$,$L_{\alpha_1 \rightarrow \beta_1}$,$L_{\alpha_2 \rightarrow \beta_2}$,$L_{\alpha_3 \rightarrow \beta_3}$.

Step 1.3. Predict the deduction set on the 84 leaks from analyzing the side-channel leakages in each AES round with template attack technique.

Step 1.4. Divide the 84 deduction sets in each AES round into four groups,$D_{\alpha_0 \rightarrow \beta_0}$,$D_{\alpha_1 \rightarrow \beta_1}$,$D_{\alpha_2 \rightarrow \beta_2}$,$D_{\alpha_3 \rightarrow \beta_3}$.

The details on how to search for the candidates of the state group from analyzing the deduction set group will be presented in the next section.

4.3 Conquer the States from Leaks in Each AES Round

Let $E(X)$ denote the entropy of the state X. Let x denote a state byte, $D(x)$ denote the deduction set on the value of $L(x)$, $\overline{x}$ denote one possible candidate of $L(x)$, and $Is(\overline{x}, D(x))$ denote the function to judge whether $\overline{x}$ is in the deduction set $D(x)$. This section will describe how to search for the state group α_j by analyzing $L_{\alpha_j \rightarrow \beta_j}$ and $D_{\alpha_j \rightarrow \beta_j}$ $(0 \leq j \leq 3)$.

Next, we take the recovery of α_0 as an example. The general procedure is composed of the following steps.

Step 2.1. Enumerate one candidate of α_0.

Step 2.2. Compute the predicted leak group, denoted as $\overline{L_{\alpha_0 \rightarrow \beta_0}}$.

Step 2.3. The enumerated candidate of α_0 can be kept only when every element in $\overline{L_{\alpha_0 \rightarrow \beta_0}}$ is among the corresponding deduction set element in $D_{\alpha_0 \rightarrow \beta_0}$, which can be realized by calling the $Is()$ functions for 21 times. Else, it can be discarded.

Step 2.4. Repeat Step 1 to Step 3 for 2^{32} times. Then, both the candidates and entropy of α_0 can be calculated.

Above is a general way to enumerate and recover α_0, the maximal enumeration complexity is 2^{32}. In practice, the adversary can adopt a smart approach to enumerate $\alpha_0 = \{a_0^i, a_5^i, a_{10}^i, a_{15}^i\}$. The improved enumeration procedure is composed of the following steps.

Step 2.1. Enumerate a_0^i.

Step 2.1.1. Enumerate one candidate of a_0^i.

Step 2.1.2. Compute 2 predicted leaks related with a_0^i ($L(\overline{a_0^i})$,$L(\overline{b_0^i})$).

Step 2.1.3. If both predicted leaks are among the corresponding deduction set($L(\overline{a_0^i})$ is among $D(a_0^i)$ and $L(\overline{b_0^i})$ is among $L(\overline{b_0^i})$), the candidate of a_0^i can be kept and go to Step 2.2. Else, go to Step 2.1.1 and enumerate another candidate of a_0^i.

Step 2.2. Enumerate a_5^i.

Step 2.2.1. Enumerate one candidate of a_5^i.

Step 2.2.2. Compute 4 predicted leaks related with a_0^i, a_5^i ($L(\overline{a_5^i})$,$L(\overline{b_5^i})$,$L(\overline{b_0^i} + \overline{b_5^i})$, $L(\overline{b_0^i} + xt(\overline{b_0^i} + \overline{b_5^i}))$).

Step 2.2.3. If all predicted leaks are among the corresponding deduction set, the candidate of a_0^i, a_5^i can be kept and go to Step 2.3. Else, go to Step 2.2.1 and enumerate another candidate of a_5^i.

Step 2.3. Enumerate a_{10}^i.

Step 2.3.1. Enumerate one candidate of a_{10}^i.

Step 2.3.2. Compute 4 predicted leaks related with a_0^i, a_5^i, a_{10}^i ($L(\overline{a_{10}^i})$,$L(\overline{b_{10}^i})$,$L(\overline{b_5^i} + \overline{b_{10}^i})$, $L(\overline{b_5^i} + xt(\overline{b_5^i} + \overline{b_{10}^i}))$).

Step 2.3.3. If all these predicted leaks are among the corresponding deduction set, the candidate of a_0^i, a_5^i, a_{10}^i can be kept and go to Step 2.4. Else, go to Step 2.3.1 and enumerate another candidate of a_{10}^i.

Step 2.4. Enumerate a_{15}^i.

Step 2.4.1. Enumerate one candidate of a_{15}^i.

Step 2.4.2. Compute the extra 11 predicted leaks related with $a_0^i, a_5^i, a_{10}^i, a_{15}^i$.

Step 2.4.3. If all predicted leaks are among the corresponding deduction set, the candidate of $a_0^i, a_5^i, a_{10}^i, a_{15}^i$ can be kept. Else, discard it. Then, go to Step 2.4.1 and enumerate another candidate of a_{15}^i.

If there is no information leaked, we can see that the attack complexity of the above general enumeration procedure and the improved enumeration procedure are identical (2^{32} times). In practice, there are always some leaks on the intermediate states and the attack complexity of the improved enumeration procedure is much lower.

Algorithm 2 shows the pseudo-code of search for the candidates of α_0 by analyzing the 21 related leaks.

Algorithm 2. Search the space of $\alpha_0 = \{a_0^i, a_5^i, a_{10}^i, a_{15}^i\}$

```
1: int Is (int b, Set $D$){
2:  if $b \in D$
3:    return 1;
4:  else
5:    return 0;
6:  end if
7: }
8: void PartialStateSearch ( ) {
9:  for $\overline{a_0^i}$=0 to 255 do //   candidates of $a_0^i$
10:    $\overline{b_0^i} = S[\overline{a_0^i}]$;
11:    if (Is($L(\overline{a_0^i}), D(a_0^i)$) and Is($L(\overline{b_0^i}), D(b_0^i)$))
12:      for $\overline{a_5^i}$=0 to 255 do { //   candidates of $a_5^i$
13:        $\overline{b_5^i} = S[\overline{a_5^i}]$;
14:        if (Is($L(\overline{a_5^i}), D(a_5^i)$) and Is($L(\overline{b_5^i}), D(b_5^i)$) and Is($L(\overline{b_0^i} + \overline{b_5^i}), D(b_0^i + b_5^i)$)
             and Is($L(\overline{b_0^i} + xt(\overline{b_0^i} + \overline{b_5^i})), D(b_0^i + xt(b_0^i + b_5^i))$)))
15:          for $\overline{a_{10}^i}$=0 to 255 do //   candidates of $a_{10}^i$
16:            $\overline{b_{10}^i} = S[\overline{a_{10}^i}]$;
17:            if (Is($L(\overline{a_{10}^i}), D(a_{10}^i)$) and Is($L(\overline{b_{10}^i}), D(b_{10}^i)$) and Is($L(\overline{b_5^i} + \overline{b_{10}^i}),$
                 $D(b_5^i + b_{10}^i)$) and Is($L(\overline{b_5^i} + xt(\overline{b_5^i} + \overline{b_{10}^i})), D(b_5^i + xt(b_5^i + b_{10}^i))$)))
18:              for $\overline{a_{15}^i}$=0 to 255 do //   candidates of $a_{15}^i$
19:                $\overline{b_{15}^i} = S[\overline{a_{15}^i}]$;
20:                if (Is($L(\overline{a_{15}^i}), D(a_{15}^i)$) and Is($L(\overline{b_{15}^i}), D(b_{15}^i)$) and Is($L(\overline{b_{10}^i} + \overline{b_{15}^i}),$
                     $D(b_{10}^i + b_{15}^i)$) and Is($L(\overline{b_0^i} + \overline{b_{15}^i}), D(b_0^i + b_{15}^i)$) and
                     Is($L(\overline{b_0^i} + \overline{b_5^i} + \overline{b_{10}^i} + \overline{b_{15}^i}), D(b_0^i + b_5^i + b_{10}^i + b_{15}^i)$) and
                     Is($L(\overline{b_0^i} + xt(\overline{b_0^i} + \overline{b_5^i}) + \overline{b_0^i} + \overline{b_5^i} + \overline{b_{10}^i} + \overline{b_{15}^i}), D(b_0^i +$
                     $xt(b_0^i + b_5^i) + b_0^i + b_5^i + b_{10}^i + b_{15}^i)$) and
                     Is($L(\overline{b_5^i} + xt(\overline{b_5^i} + \overline{b_{10}^i}) + \overline{b_0^i} + \overline{b_5^i} + \overline{b_{10}^i} + \overline{b_{15}^i}), D(b_5^i +$
                     $xt(b_5^i + b_{10}^i) + b_0^i + b_5^i + b_{10}^i + b_{15}^i)$) and
                     Is($L(\overline{b_{10}^i} + xt(\overline{b_{10}^i} + \overline{b_{15}^i})), D(b_5^i + xt(b_{10}^i + b_{15}^i))$)) and
                     Is($L(\overline{b_{10}^i} + xt(\overline{b_{10}^i} + \overline{b_{15}^i}) + \overline{b_0^i} + \overline{b_5^i} + \overline{b_{10}^i} + \overline{b_{15}^i}),$
                     $D(b_{10}^i + xt(b_{10}^i + b_{15}^i) + b_0^i + b_5^i + b_{10}^i + b_{15}^i)$) and
                     Is($L(\overline{b_0^i} + xt(\overline{b_0^i} + \overline{b_{15}^i})), D(b_0^i + xt(b_0^i + b_{15}^i))$)) and
                     Is($L(\overline{b_0^i} + xt(\overline{b_0^i} + \overline{b_{15}^i}) +$
                     $\overline{b_0^i} + \overline{b_5^i} + \overline{b_{10}^i} + \overline{b_{15}^i}), D(b_0^i + xt(b_0^i + b_{15}^i) + b_0^i + b_5^i + b_{10}^i + b_{15}^i)$)))
21:                  kept $\overline{a_0^i}, \overline{a_5^i}, \overline{a_{10}^i}, \overline{a_{15}^i}$ as candidates of $a_0^i, a_5^i, a_{10}^i, a_{15}^i$;
22:                end if
23:              end for
24:            end if
25:          end for
26:        end if
27:      end for
29:    end if
30:  end for
31: }
```

Applying Algorithm 2, we can get limited candidates of α_0. Using a similar approach, we can conquer other three state groups α_j by analyzing $L_{\alpha_j \to \beta_j}$ and $D_{\alpha_j \to \beta_j}$ ($1 \le j \le 3$). Then, candidates of A^i can be computed ($A^i = \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\}$). According to the encryption procedure of AES, Y^i can be computed directly from A^i and their entropies are the same, $E(A^i) = E(Y^i)$. Applying the above approach to each round, candidates and entropies of A^i and Y^i can be recovered ($0 \le i \le 9$).

4.4 Search for the Master Key

In this section, we show how to search for the master key of AES with the recovered state A^i, Y^i and the 16 HW leaks of loading the round-key K^i in each AES round.

Step 3.1. Search for the preliminary space of the round-keys.

According to the AES algorithm, Eq.(2) holds:

$$Y^i = X_{i+1}, X_{i+1}^i + K^{i+1} = A^{i+1} \Rightarrow K^{i+1} = Y^i + A^{i+1}. \tag{2}$$

As to $0 \le i \le 8$, the candidates of round-key K^{i+1} can be calculated individually. If the plaintext is known, the candidates of the first round key K^0 can be calculated by:

$$K^0 = P + A^0. \tag{3}$$

If the ciphertext is known, the post-whitening key K^{10} can be calculated by:

$$K^{10} = C + A^9. \tag{4}$$

Step 3.2. Further reduce the search space of the round keys.

If there are 100 leaks per AES round, the entropy of K^i, $E(K^i)$, can be further reduced by analyzing 16 leaks of loading the round-key bytes.

If there are only 84 leaks per AES round (e.g., ASCAs in [20], [27], [36]), this step can be omitted. Obviously, the final entropy of the secret key would be larger in this scenario than that in using 100 leaks.

Step 3.3. Search for the master key.

According to the AES key schedule [12], recovering any round key is equivalent to recovering the master key. We can calculate the candidates of the master key by brute-force search of K^i for each round and then use the intersection of candidates in multiple rounds to compute the final search space of the master key. The complexity of computing the intersection varies with the candidate size in different rounds, affordable with small size and intensive with large size.

5 Experimental Results of IDASCA on AES

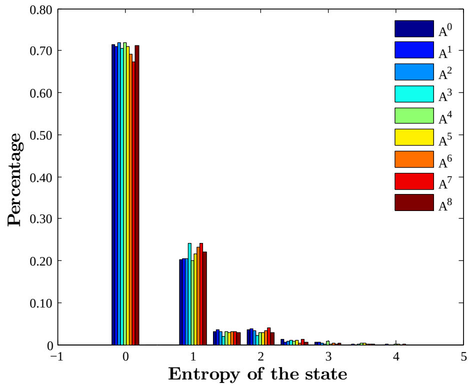
Let ρ_k denote the 2^{48}th-order key recovery success rate [32], the proportion of traces for which the entropy of the AES master key of ($E(K)$) can be reduced to less than 48 [5]. Next, we present results of extensive attacks on AES implemented on 8-bit microcontroller ATMEGA324P, which closely follows the Hamming weight leakage model (HWLM) [37] and is also widely studied in previous ASCAs [20], [21], [23], [27], [36]. In our experiment, IDASCA runs on a quad-core PC with Intel Core I5-2400, 3.10 GHz, 8G memory and Windows XP 32-bit OS.

Three cases of IDASCA are studied in this section. The first is the attack without Hamming weight deducing errors, which explains why ASCA can work under unknown plaintext and ciphertext scenario. The second and the third cases are error tolerant attacks with fixed and dynamic deduction set sizes, respectively.

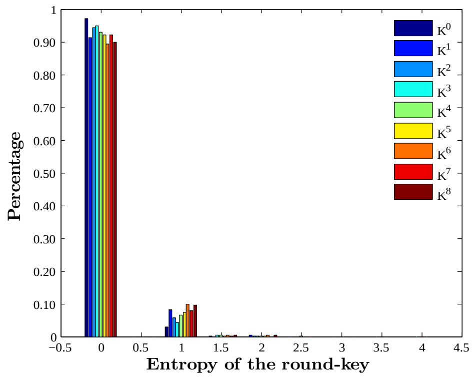
5. Most modern Intel CPUs have a native implementation of AES (AES-NI), which allows a sustained rate of more than 2^{31} AES operations per second. Thus, an attacker can use a single machine with an AES-NI implementation to enumerate 2^{48} key candidates within 40 hours [2].

5.1 Case 1: Attack Without Errors

In this case, it assumes that all of the HWs are deduced correctly, which can be achieved by averaging multiple power traces with the same plaintext and the secret key, as noted in [21]. We run 1000 attacks and calculate the frequency of $E(A^i)$ (Fig. 1(a)) for the first nine AES rounds. The average time required for the attack is 0.1 second. We can see that $E(A^i)$ is quite small and is 0.37 on average, which means that the almost correct value of A^i can be recovered directly.



(a) Entropy of A^i



(b) Entropy of K^i

Fig. 1. Entropy of A^i and K^i in IDASCA on AES (μ=1)

Next we present the analysis on why IDASCA can work under both known plaintext and unknown plaintext/ciphertext scenarios. No matter what scenario it is, both the output of AK and the round output in the i-th round can be recovered by analyzing the 84 leaks (excluding the 16 leaks in loading the round key K^i) in that round. As to analyzing the $i+1$-th round, since the final output of the i-th round, also an input of AK in the $i+1$-th round, is unknown, we cannot extract the round-key from any single round between Round 2 and 9. However, we can extract the preliminary search space of the $i+1$-th round key by analyzing any two consecutive rounds, e.g., the i-th round and the $i+1$-th round (by XORing the final output of the i-th round with the output of AK in the $i+1$-th round). Then, $E(K^{i+1})$ can be further reduced by analyzing the 16 leaks in loading K^{i+1}.

The difference comes in the first round when one input of the AK is the plaintext and the final round when the output of the 11-th AK is the ciphertext. Under known plaintext scenario, the secret key can be extract directly from the first round. Under known ciphertext scenario, the secret key can be extract directly from the last round.

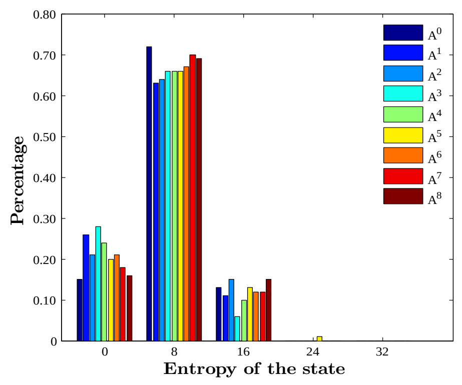
Fig. 1(b) demonstrates the frequency of $E(K^i)$ in the first 9 rounds, which is calculated by the output of Step 2 in Section 4.4. Under known plaintext scenario, $E(K^0)$ can be precalculated by XORing A^0 with the plaintext, and then further reduced by analyzing the 16 leaks in loading K^0. The average value of $E(K^0)$ is only 0.02, which means that the secret key can be recovered easily by analyzing the first round. Under unknown plaintext/ciphertext scenario, limited candidates of K^{i+1} ($0 \le i \le 7$) can be calculated by analyzing any two consecutive rounds (XORed Y^i with A^{i+1}), and then further reduced by analyzing the 16 leaks in loading K^{i+1}. According to the key schedule of AES, the master key can be recovered easily.

5.2 Case 2: Attack with Errors, Fixed Deduction Size

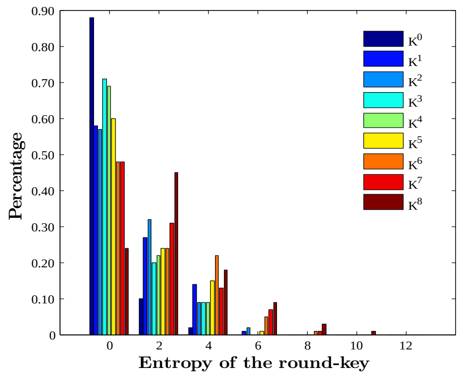
In this case, two different values of the deduction size μ are considered, the same as the prob-TASCA in CHES 2012 [23]. The first is $\mu = 2$, where we conduct 100 attacks on the first 9 rounds of AES and the average time required is 1 second. The results show that $\rho_d = 80\%$ and $\rho_k = 80\%$. The statistics on $E(A^i)$ and $E(K^i)$ are shown in Fig. 2. We can see that $E(K^i)$ can be reduced to less than 12, which means that the master key of AES can be recovered by attacking the first round under known plaintext scenario , or by attacking any two consecutive rounds under unknown plaintext/ciphertext scenario. Compared with 43.7 minutes required in TASCA and 56.7 minutes required in prob-TASCA [23], IDASCA is more efficient and requires less time complexity.

For $\mu = 3$, we also conduct 100 attacks on the first 9 AES rounds. The decoding success rate ρ_d is 100%. For entropy estimations of the 128-bit state, we divide the state into four groups (four bytes in each group). The entropy estimations are made independently for the four groups of four bytes and then combined together to calculate the entropy of the full 128-bit state. The statistics of $E(A^i)$ are shown in Fig. 3(a). We can see that $E(A^i)$ is quite large (on average 2^{48}). Under known plaintext scenario, we use candidates of A^0, the plaintext and the 16 HW leaks on the secret key K^0 to further reduce $E(K)$. The results are shown in Fig. 3(b). By attacking the first round of AES, $E(K)$ can be reduced to on average $2^{37.2}$ and the success rate ρ_k is 100%. The time required in searching for $E(K)$ is 7.42 seconds. After that, the master key of AES can be recovered with an additional exhaustive search within two minutes using a single machine with an AES-NI implementation [2].

In TASCA and prob-TASCA [23], the *error rate* denotes the percentage of the number of HWs that does not

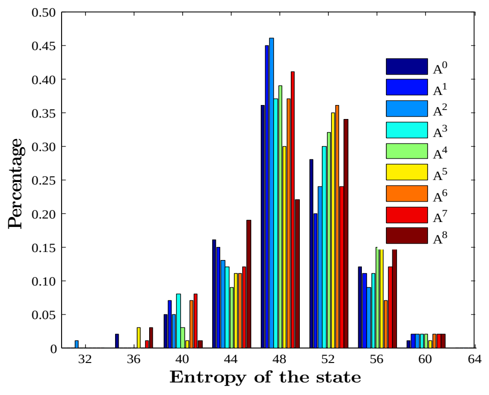


(a) Entropy of A^i

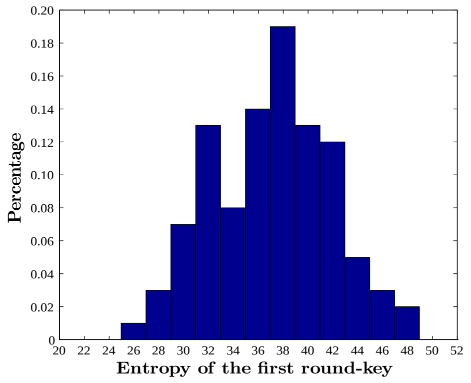


(b) Entropy of K^i

Fig. 2. Entropy of A^i and K^i in IDASCA on AES (μ=2)



(a) Entropy of A^i



(b) Entropy of K^i

Fig. 3. Entropy of A^i and K^i in IDASCA on AES (μ=3)

have the highest probabilities in the deduction set among all HWs used in one attack. The efficiency of IDASCA is less dependent on the *error rate* and the attack can be implemented successfully with 100% *error rate* when $\mu = 3$. Compared with the less than 20% error rate, 16.8 hours required in TASCA and 8.2 hours required in prob-TASCA [23], IDASCA is more robust and efficient.

When $\mu \geq 4$ from one power trace, $E(A^i)$ is very large and $E(A^i) > 64$. Since it is difficult to recover the master key by analyzing one sample, two strategies can be adopted.

The first is to average the power traces of encrypting the same plaintext multiple times and get a smaller value of μ. For our experiment setup, the results show that average of two power traces can reduce μ to 3.

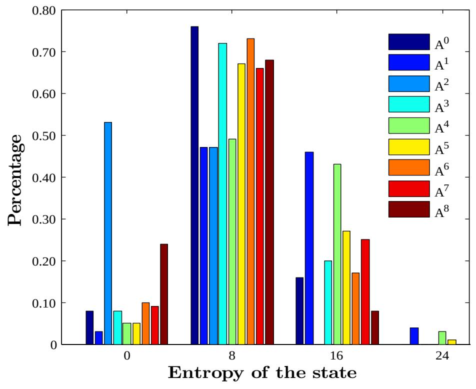
The second is to use multiple different plaintexts for the attack. We conducted IDASCA on AES under different large value of μ, two plaintexts are required when $\mu = 4$, three plaintexts are required when $\mu = 5$, seven plaintexts are required when $\mu = 6$, and fifteen plaintexts are required when $\mu = 7$. We can see that IDASCA can be used to bridge the gap between DPA (using many plaintexts) and ASCAs (using very few, sometimes only one). One advantage of IDASCA is that, as in DPA, increasing the data complexity (number of measurements) is an easy way to reduce the time complexity; but unlike DPA, IDASCA can exploit all of the leaks in one AES round, more than 16 leaks exploited in DPA on AES.

When $\mu \geq 4$, under unknown plaintext/ciphertext scenario, since both $E(A^i)$ and $E(A^{i+1})$ $(1 \leq i \leq 8)$ are quite large, the effort to calculate and further reduce $E(K^{i+1})$ is unaffordable (2^{96}) and IDASCA fails using a single power trace of one plaintext. If power traces of multiple plaintext/ciphertext pairs are used, since new unknown 256-bit variables (unknown plaintext/ciphertext) are introduced in each new pair, the complexity of IDASCA would be greater than single pair attack and IDASCA also fails. Next, we provide a better way to conduct IDASCA on AES for large value of μ.

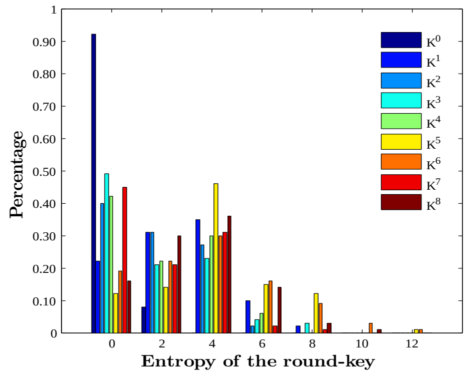
5.3 Case 3: Attack with Errors, Dynamic Deduction Size

In practice, an adaptive adversary can apply the dynamic μ approach and choose the highest n HW deductions for each leak when the sum of their probabilities is over a fixed threshold T, as noted in [20], [36]. When we apply IDASCA on AES for this scenario, the attack complexity is much lower than the attack with fixed μ.

In the attack, ρ_d increases with T (better robustness). Let $\eta(\mu)$ denote the percentage of leaks when the deduction size is μ. When $T = 0.85$, $\eta(1) = 25\%, \eta(2) =$



(a) Entropy of A^i



(b) Entropy of K^i

Fig. 4. Entropy of A^i and K^i in IDASCA on AES ($1 \leq \mu \leq 3$, $T = 0.90$)

$65\%, \eta(3) = 10\%$, and the decoding success rate $\rho_d = 90\%$. If $T = 0.90$, ρ_d would reach 99%, $\eta(1) = 16\%, \eta(2) = 51\%, \eta(3) = 33\%$. We run 100 attacks for $T = 0.90$. The attack success rate is $\rho_k = 99\%$ and the average time of the attack is 4.5 seconds. The results are shown in Fig. 4. We can see that $E(K^i)$ $(0 \leq i \leq 8)$ is less than 12. For the first time, IDASCA can be implemented successfully on AES under both the error tolerant and unknown plaintext/ciphertext scenarios with only single power trace, which is quite powerful.

To show the robustness of IDASCA, we also run 100 attacks for $T = 0.95$ on the first AES round under known plaintext scenario. The average time required for one attack is 25.50 seconds. In the attack, we can also get some deduction sets with four candidates, $\eta(1) = 9\%, \eta(2) = 29\%, \eta(3) = 44\%, \eta(4) = 18\%$ and $\rho_d = 100\%$. The results on the entropy of K^0 are shown in Fig. 5. K^0 can be reduced to $2^{29.84}$ by analyzing the first AES round.

6 Interpreting Previous ASCAs on AES with IDASCA

In this Section, we use IDASCA to interpret the results of different ASCAs under the same assumption as in previous work.

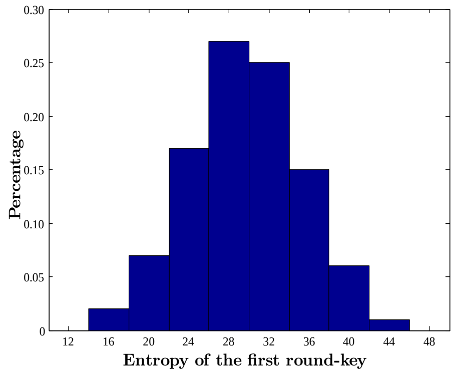


Fig. 5. Entropy of K^i in IDASCA on AES ($1 \leq \mu \leq 3$, $T = 0.95$)

6.1 Case 1: Interpreting the Original ASCA

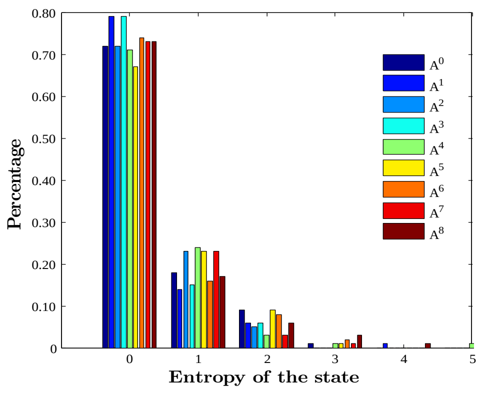
Under the assumption that 84 HWs were leaked per AES round and all the HWs can be deduced correctly, Renauld *et al.* [27] first applied ASCA on AES with wonderful results in CHES 2009. It has been shown that under the known plaintext scenario, the secret key of the AES can be recovered by analyzing the HW leakages in three consecutive rounds from a single power trace. ASCA is also the first attack that succeed on AES under unknown plaintext and ciphertext scenario.

We also conduct IDASCA 100 times on AES under this scenario. The frequency of $E(A^i)$ and $E(K^i)$ $(0 \leq i \leq 8)$ are shown in Fig. 6. We can see that $E(A^i)$ can be reduced to less than 1 on average and $E(K^i)$ can be reduced to less than 5. Under known plaintext scenario, the secret key of AES can be recovered by analyzing the 84 HW leaks in the first round, which is less than the three rounds required in [27]. Under unknown plaintext/ciphertext scenario, the secret key of AES can be recovered by analyzing the 168 HWs in any two consecutive rounds, which is less than the three rounds required in [27].

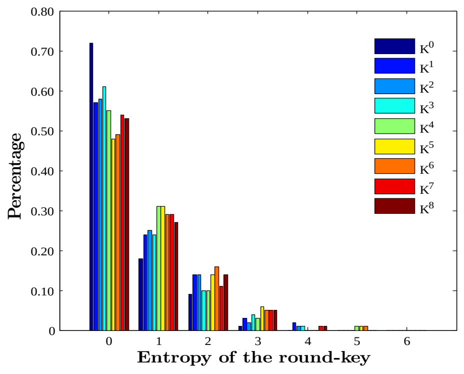
6.2 Case 2: Interpreting MDASCA and IASCA

The original ASCA [27] is sensitive to errors. To mitigate this, zhao *et al.* [36] predicted multiple deductions on the HWs and proposed the multiple deductions-based ASCA (MDASCA) technique to attack AES. Similar ideas are also proposed by Mohamed et al. in HOST 2012 [27] named as improved ASCA (IASCA).

In MDASCA [36], 84 HWs are assumed to be leaked per AES round. The result showed that, even if 80% of the HWs have more than one deduction ($\eta(1) = 20\%, \eta(2) = 30\%, \eta(3) = 50\%$), the master key of AES can be recovered by exploiting 252 HWs in three consecutive rounds. To interpret this, we also conduct IDASCA 100 times on AES under this scenario. The results are shown in Fig. 7. We can see that $E(K^0)$ and $E(K^i)$ $(1 \leq i \leq 8)$ can be reduced to 2^{15} and 2^{26} respectively. Also, $E(K)$

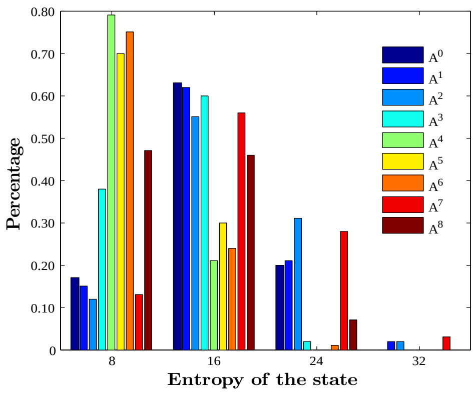


(a) Entropy of A^i

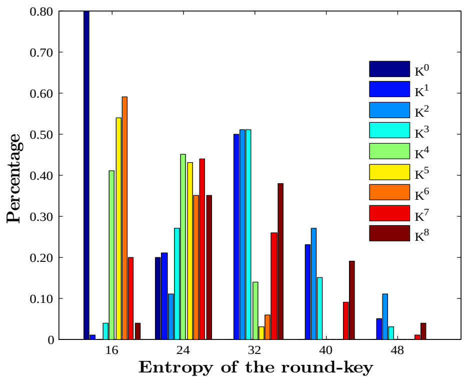


(b) Entropy of K^i

Fig. 6. Entropy of A^i and K^i in IDASCA on AES ($\mu = 1$, 84 leaks per round)



(a) Entropy of A^i



(b) Entropy of K^i

Fig. 7. Entropy of A^i and K^i in IDASCA on AES ($\mu > 1$ (80%), 84 leaks per round)

has high probability of being reduced to zero by analyzing the round-keys in three rounds, which make sure that the satisfied solution of the SAT solver in [36] is the correct one.

From Fig. 7, we can also see that the secret key of AES can be recovered by analyzing the HW leaks in the first round of AES under known plaintext scenario ($E(K^0)$ in Fig. 7(b)), and by analyzing the HW leaks in any two consecutive rounds of AES under unknown plaintext/ciphertext scenario ($E(K^1)$,...,$E(K^8)$ in Fig. 7(b)). The data complexity required in IDASCA on AES is less than that of MDASCA [36].

In IASCA [20], the dynamic μ approach was applied to attack AES under known plaintext/ciphertext scenario. Different threshold values are used, where $T = \{0.80, 0.85, 0.90, 0.95\}$. When $T \geq 0.90$, IASCA requires HW leakages from multiple rounds. To interpret this, under the same scenario in [20], we also conduct IDASCA on AES. The results are listed in Table 2.

When $T = \{0.80, 0.85\}$, ρ_d is much lower. IDASCA on the first round of AES, can reduce the search space of the 128-bit master key to less than 2^5. As both the plaintext and ciphertext are included in the equations, it is easy for the SAT solver to verify the 2^5 key candidates and output the correct key quickly. This explains why the work in [20] can succeed by analyzing the HW leakages in the first round of AES for $T = \{0.80, 0.85\}$.

When $T = \{0.90, 0.95\}$, ρ_d is much higher, which increases the robustness of the attack. Applying IASCA on the HW leakages in the first round of AES, on average $2^{13.63}$ key solutions exist when $T = 0.90$ and on average $2^{29.84}$ key solutions exists when $T = 0.95$. By only analyzing the HW leaks in the first round, it is difficult for the SAT solver to verify the correctness of these key candidates within a reasonable amount of time. To overcome this, IASCA [20] requires HW leakages of two or three rounds, so that the search space of the AES master key can be reduce to 1 and the SAT solver can find out the solution quickly.

6.3 Case 3: Interpreting TASCA and Prob-TASCA

In CHES 2010 [21], Oren *et al.* proposed the error tolerant ASCA (TASCA) and converted the key recovery to a pseudo-Boolean optimization problem. The errors (± 1 variance from the correct HW deduction) are encoded into the equations. The SCIP optimizer [1] was used to solve these equations. TASCA was applied to Keeloq in [21] and extended to AES in Eprint 2012/092 [22] with less than 20% error rate.

TABLE 2
Results of this paper and the work in [20] under HWLM, dynamic μ, $\eta = 84$

Attack	T	μ	ρ_d	ρ_k	required rounds	time	$E(K)$
IASCA[20]	0.80	1(35%), 2(47%), 3(18%), 4(0%), 5(0%)	82%	82%	1	2s	–
this paper	0.80	1(35%), 2(47%), 3(18%), 4(0%), 5(0%)	82%	82%	1	0.63s	$2^{1.95}$
IASCA[20]	0.85	1(23%), 2(64%), 3(13%), 4(1%), 5(0%)	94%	94%	1	3s	–
this paper	0.85	1(23%), 2(64%), 3(13%), 4(1%), 5(0%)	94%	94%	1	1.21s	$2^{4.32}$
IASCA[20]	0.90	1(14%), 2(45%), 3(36%), 4(5%), 5(0%)	99%	99%	2	3s	–
this paper	0.90	1(14%), 2(45%), 3(36%), 4(5%), 5(0%)	99%	99%	1	4.85s	$2^{13.63}$
IASCA[20]	0.95	1(9%), 2(29%), 3(44%), 4(18%), 5(0%)	100%	100%	3	84s	–
this paper	0.95	1(9%), 2(29%), 3(44%), 4(18%), 5(0%)	100%	100%	1	25.50s	$2^{29.84}$

In CHES 2012, Oren *et al.* exploited the different probabilities of deduction candidates and proposed prob-TASCA. The results showed that prob-TASCA can retain some information lost in TASCA. In both attacks on AES [22], [23], it assumes that 100 HWs are leaked per round and the attack is successful if less than four key bytes were incorrect. As the locations of the erroneous key bytes are unknown, the work in [22], [23] claimed that this would yield at most 2^{48} brute-force-search of the AES secret key. If the error rate is less than 20%, TASCA can recover 13.25 correct key bytes (the location of these correct key bytes are unknown) within 16.8 hours on average and prob-TASCA can recover 16 correct key bytes within 8.2 hours on average.

The IDASCA technique proposed in this paper can be sightly adapted to attack the first round of AES under the same scenario in TASCA and Prob-TASCA. In other words, TASCA and Prob-TASCA can be conducted by IDASCA without relying on the SCIP solver. To interpret TASCA on the first round of AES [22], we propose a variant of IDASCA to calculate the candidates of $\{k_0^0, k_5^0, k_{10}^0, k_{15}^0\}$ by exploiting the 25 leaks in the first AES round. We first generate the 3-sized deduction set for these 25 leaks. Then, we modify the Algorithm 2 according to the following steps:

1) Add four $Is()$ functions as filters to analyze the four HW leakages from loading the four round key bytes $\{k_0^0, k_5^0, k_{10}^0, k_{15}^0\}$: $Is(L(\overline{a_0^0} + p_0), D(k_0^0))$ at line 11, $Is(L(\overline{a_5^0} + p_5), D(k_5^0))$ at line 14, $Is(L(\overline{a_{10}^i} + p_{10}), D(k_{10}^0))$ at line 17, $Is(L(\overline{a_{15}^0} + p_{15})$, and $D(k_{15}^0))$ at line 20.
2) Let h_i denote the correct HW for the i-th leak among the 25 leaks ($0 \le i < 25$), $\overline{h_i}$ denote the predicted HW of h_i, which is computed by the candidate of $\{a_0^0, a_5^0, a_{10}^0, a_{15}^0\}$. Let $\widehat{h_i}$ denote the HW deduction of h_i with the highest probability in the 3-sized deduction set, which is profiled from the power traces, S_h denote the sum of 25 absolute values on $\overline{h_i} - \widehat{h_i}$. Then, at the end of the line 21 in Algorithm 2, we write a function to compute S_h.

$$S_h = \sum_{i=0}^{24} abs(\overline{h_i} - \widehat{h_i}) \tag{5}$$

where abs(x) denotes the absolute value of x.
3) Sort all the candidates of $\{a_0^0, a_5^0, a_{10}^0, a_{15}^0\}$ by S_h. The candidate with the smallest S_h has the highest probability to be the correct $\{a_0^0, a_5^0, a_{10}^0, a_{15}^0\}$. Since the plaintext is known, $\{k_0^0, k_5^0, k_{10}^0, k_{15}^0\}$ can be recovered.

Using similar approach, we can also recover $\{k_4^0, k_9^0, k_{14}^0, k_3^0\}$, $\{k_8^0, k_{13}^0, k_2^0, k_7^0\}$, and $\{k_{12}^0, k_1^0, k_6^0, k_{11}^0\}$. The concatenation of them is the secret key.

To interpret Prob-TASCA on the first round of AES [23], we also propose another variant of IDASCA to calculate the candidates of $\{k_0^i, k_5^i, k_{10}^i, k_{15}^i\}$ by exploiting the 25 leaks in the first AES round. We first predict the 3-sized deduction set from the power trace and calculate the different probabilities for each candidate. Then, we modify the Algorithm 2 by the following steps:

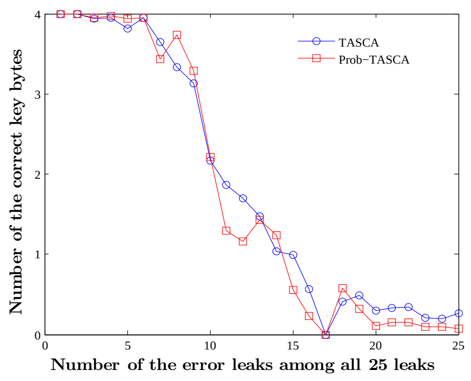
1) Add four $Is()$ functions as filters to analyze the four HW leakages on loading the four round key bytes $\{k_0^0, k_5^0, k_{10}^0, k_{15}^0\}$: $Is(L(\overline{a_0^0} + p_0), D(k_0^0))$ at line 11, $Is(L(\overline{a_5^0} + p_5), D(k_5^0))$ at line 14, $Is(L(\overline{a_{10}^i} + p_{10}), D(k_{10}^0))$ at line 17, $Is(L(\overline{a_{15}^0} + p_{15})$, and $D(k_{15}^0))$ at line 20.
2) Let $\overline{h_i}$ denote the predicted HW of h_i, which is computed by the candidate of $\{a_0^0, a_5^0, a_{10}^0, a_{15}^0\}$. Let $prob_i(\overline{h_i} = h_i)$ denote the probability of $\overline{h_i} = h_i$, which is profiled from the power traces. Let S_h denote the sum of 25 values on $prob_i(\overline{h_i} = h_i)$. Then, at the end of the line 21 in Algorithm 2, we write a function to compute S_h.

$$S_h = \sum_{i=0}^{24} prob_i(\overline{h_i} = h_i) \tag{6}$$

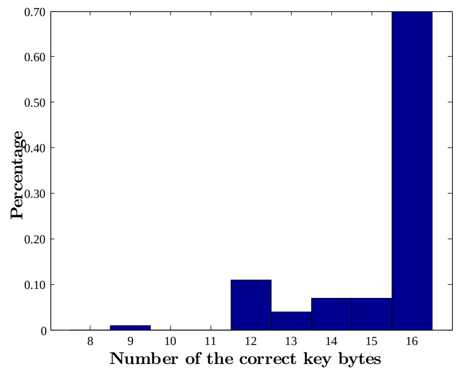
3) Sort all the candidates of $\{a_0^0, a_5^0, a_{10}^0, a_{15}^0\}$ by S_h. The candidate related with the largest S_h has the highest probability to be the correct $\{a_0^0, a_5^0, a_{10}^0, a_{15}^0\}$. Since the plaintext is known, $\{k_0^0, k_5^0, k_{10}^0, k_{15}^0\}$ can be recovered.

The above two types of attack can succeed within five seconds. We run 100 attacks for each type, and the number of the correct key bytes of $\{k_0^0, k_5^0, k_{10}^0, k_{15}^0\}$ under different number of error leaks among these 25 leaks for TASCA and Prob-ASCA is shown in Fig. 8(a).

We can see that when the number of error HW deductions in computing $\{y_0^0, y_1^0, y_2^0, y_3^0\}$ is less than five or six bytes (25% to 30% error rate), more than three key bytes are correct. When extending this to analyze the full AES key, more than twelve key bytes tend to be correct. From



(a) Entropy of A^i



(b) Statistics on the number of the correct key bytes in TASCA

Fig. 8. Entropy of A^i and K^i in IDASCA on AES (20% error rate)

Fig. 8(a), when the number of erroneous HW deductions is less than six, more key bytes can be recovered by Prob-ASCA, which means that Prob-ASCA can regain some information lost in TASCA by considering the probabilities of different deductions. Fig. 8(b) lists the number of correct key bytes from analyzing the 100 HWs in the first AES round with IDASCA to interpret TASCA (the error rate is 20%). The results show that more than twelve key bytes are correct for 99% of the cases. In the experiments of interpreting Prob-TASCA with IDASCA, if the error rate is less than 20%, full AES key can be recovered with 95% success rate. Compared with the average 16.8 hours required in TASCA [22] and 8.2 hours required in Prob-TASCA [23], IDASCA is also much more efficient.

7 Conclusions and Future Work

ASCA falls under analytical side-channel attack and can recover the entire secret key at once. Previous ASCAs on AES [8], [20], [21], [23], [26], [27], [36] mainly rely on a general equation solver, which makes it difficult to take into account the specialized structures or properties of AES. This paper proposes a new analytical side-channel analysis technique on AES by exploiting the incomplete diffusion feature in one AES round. We name our technique *incomplete diffusion analytical side-channel analysis (IDASCA)*. Extensive attacks are performed against the software implementation of AES on an 8-bit microcontroller, which conforms to HWLM. The results show that like ASCA, IDASCA can exploit the leakages in all AES rounds from a single power trace and works even under the unknown plaintext/ciphertext scenario. Compared with previous ASCAs, IDASCA is also much more efficient and robust. Meanwhile, IDASCA can be used to explain the results of previous ASCAs on AES from a quantitative perspective, such as, why ASCA can work under unknown plaintext/ciphertext scenario and what are the extreme cases in ASCA on AES. We also extend IDASCA to devices that do not conform to HWLM, e.g, TLM in CHES 2012 [23]. More details can be found in Appendix 1.

From IDASCA on AES, we can explain why ASCA is so efficient on block ciphers. Firstly, for the software implementation of block ciphers, there exists too many leakage points per round. Secondly, the intermediate state of each round is simple enough so that there is a high probability of being recovered by analyzing these leakages. Finally, the leakages of any two adjacent round can be used to recover the related round-keys, which in turn can be used to recover the master key.

The work in [8] introduced a new criterion for the effectiveness of ASCA. This criterion relies on a new notion of algebraic immunity (especially on the nonlinear component, e.g., S-Box). Assuming the Hamming weight (or Hamming distance) of the S-Box input and S-Box output are leaked, various experiments are performed on PRESENT and AES. The equations are solved using the F4 Gröbner basis-based algorithms in MAGMA solver. The results show that algebraic immunity can be used as a criterion to design S-boxes of block ciphers optimally resistant to ASCA. The work of IDASCA raises a very interesting question from the complementary view of [8], can diffusion in the linear component (e.g., the branch number of the MixColumns operation in the AES) of block ciphers be used as a good design criterion for improving its security against ASCA or analytical SCAs? IDASCA can also be adjusted and adapted to other block ciphers if diffusion was not complete in one round. We have successfully applied IDASCA to PRESENT, LED and MIBS by exploiting the incomplete diffusion feature and successfully recover the master key from a single power trace. The attacks can also be conducted under error tolerant and unknown plaintext/ciphertext scenarios. For IDASCA on ciphers where diffusion can be completed in one round (e.g., Khazad block cipher), attacks would be more difficult (computationally).

It should be noted that the readers should evaluate the impact of ASCA and IDASCA objectively considering the informativeness and the robustness in the side-channel information extraction. ASCA and IDASCA can exploit the leakages of all the cipher rounds (more than a single round in DPA), which provide a positive answer in search of methods that extract more information from the side-channel leakages. Thus, they can recover the secret key of AES with a single power trace even when both the plaintext and ciphertext are unknown, which is much more powerful and requires less power traces than DPA. From the robustness view, the success of ASCA and IDASCA depends highly on the ability of the adversary in deducing the accurate value of intermediates states from the power traces. This precondition is much stronger than that in DPA.

IDASCA can help to improve the understanding of different ASCA techniques and make analytical side-channel attacks more practical. Although there still remains a tradeoff between the robustness and informativeness for IDASCA at the moment, we believe that future research may further reduce this gap. The experimental results of the paper show that IDASCA should at least be taken into account for the security of AES implemented on small devices like microcontroller. We hope IDASCA can be used to evaluate the worst-case security level for cryptographic hardware products assuming the adversary has the most powerful attacking capability in practice.

When searching for the state in each AES round (Section 4.3), IDASCA in this paper mainly enumerates and computes the satisfied candidates of four state bytes. For these candidates, no rank information is exploited. This also raises several interesting problems for further research. The first is how to utilize the different probabilities of multiple deductions to generate the rank for each state candidate (four bytes). The second is how to apply the key ranking enumeration [34] and estimation algorithms [35] to IDASCA, and exploit the rank to estimate the remaining key strength in ASCA more accurately.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under the grants 60772082, 61173191, 61272491, 61202386, 61309021, the Major State Basic Research Development Program (973 Plan) of China under the grant 2013CB338004, and the US National Science Foundation under the grant CNS-0644188 and the ERC project 280141 on CRyptographic Algorithms and Secure Hardware (CRASH).

References

[1] T. Achterberg. SCIP: solving constraint integer programs, *Math. Prog. Comp.*, vol. 1, pp. 1-41, July 2009.

[2] K. Akdemir, M. Dixon, W. Feghali, P. Fay, V. Gopal, J. Guilford, E. Ozturk, G. Wolrich and R. Zohar. Breakthrough AES Performance with Intel AES New Instructions[Online], available at: http://software.intel.com/file/27067, October 2010.

[3] G.V. Bard. Algebraic Cryptanalysis, Springer, 2009.

[4] E. Biham and A. Shamir. Differential Cryptanalysis of the Data Encryption Standard, in *CRYPTO 1990*, LNCS 537, pp. 2-21, 1991.

[5] A. Bogdanov and A. Pyshkin. Algebraic Side-Channel Collision Attacks on AES[online], *Cryptology ePrint Archive*, available at: http://eprint.iacr.org/2007/477, 2007.

[6] C. Bouillaguet, P. Derbez, and P.-A. Fouque. Automatic Search of Attacks on Round-Reduced AES and Applications,in *CRYPTO 2011*, LNCS 6841, pp. 169-187, 2011.

[7] E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model, in *CHES 2004*, LNCS 3156, pp. 16-29, 2004.

[8] C. Carlet, J.-C. Faugère, C. Goyet and G. Renault. Analysis of the algebraic side channel attack, *Journal of Cryptographic Engineering*, vol. 2 No. 1, pp. 45-62, 2012.

[9] S. Chari, J. Rao and P. Rohatgi. Template Attacks, in *CHES 2002*, LNCS 2523, pp. 13-28, 2002.

[10] C. Cid, S. Murphy and M. Robshaw. Algebraic Aspects of the Advanced Encryption Standard, Springer, 2006.

[11] N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations, in *Asiacrypt 2002*, LNCS 2501, pp. 267-287, 2002.

[12] J. Daemen and V. Rijmen. AES Proposal: Rijndael[online], available at: http://www.cryptosoft.de/docs/Rijndael.pdf, 1998.

[13] I. Dinur and A. Shamir. Side Channel Cube Attacks on Block Ciphers[online], *Cryptology ePrint Archive*, available at: http://eprint.iacr.org/2009/127, 2009.

[14] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F4), *Journal of Pure and Applied Algebra*, Vol. 139, No. 1-3, pp. 61-88, 1999.

[15] B. Gierlichs, L. Batina, P. Tuyls and B. Preneel. Mutual Information Analysis, in *CHES 2008*, LNCS 5154, pp. 426-442, 2008.

[16] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems, in *CRYPTO 1996*, LNCS 1109, pp. 104-113, 1996.

[17] P. Kocher, J. Jaffe and B. Jun. Differential power analysis, in *CRYPTO 1999*, LNCS 1666, pp. 388-397, 1999.

[18] S. Mangard. A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion, in *ICISC 2002*, LNCS 2587, pp. 343-358, 2003.

[19] M. Matsui. Linear Cryptanalysis Method for DES Cipher, in *EUROCRYPT 1993*, LNCS 765, pp. 386-397, 1994.

[20] M.S.E. Mohamed, S. Bulygin, M. Zohner, A. Heuser, M. Walter and J. Buchmann. Improved algebraic side-channel attack on AES, in *HOST 2012*, pp. 146-151, 2012.

[21] Y. Oren, M. Kirschbaum, T. Popp and A. Wool. Algebraic Side-Channel Analysis in the Presence of Errors, in *CHES 2010*, LNCS 6225, pp. 428-442, 2010.

[22] Y. Oren and A. Wool. Tolerant Algebraic Side-Channel Analysis of AES[online], *Cryptology ePrint Archive*, Report 2012/092, available at: http://iss.oy.ne.ro/TASCA-eprint, 2012.

[23] Y. Oren, M. Renauld, F.-X. Standaert and A. Wool. Algebraic Side-Channel Attacks Beyond the Hamming Weight Leakage Model, in *CHES 2012*, LNCS 7428, pp. 140-154, 2012.

[24] Princeton University. zChaff[online], available at: http://www.princeton.edu/~chaff/, 2004.

[25] J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards, in *E-smart 2001*, LNCS 2140, pp. 200-210, 2001.

[26] M. Renauld and F.-X. Standaert. Algebraic Side-Channel Attacks, in *INSCRYPT 2009*, LNCS 6151, pp. 393-410, 2009.

[27] M. Renauld, F.-X. Standaert and N. Veyrat-Charvillon. Algebraic side-channel attacks on the AES: Why time also matters in DPA, in *CHES 2009*, LNCS 5747, pp. 97-111, 2009.

[28] M. Renauld and F.-X. Standaert. Representation-, Leakage- and Cipher- Dependencies in Algebraic Side-Channel Attacks, in industrial track of *ACNS 2010*, 2010.

[29] M. Renauld, F.-X. Standaert, N. Veyrat-Charvillon, D. Kamel and D. Flandre. A Formal Study of Power Variability Issues and Side-Channel Attacks for Nanoscale Devices, in *EUROCRYPT 2011*, LNCS 6632, pp. 109-128, 2011.

[30] K. Schramm, G. Leander, P. Felke and C. Paar. A collision-attack on AES, in *CHES 2004*, LNCS 3156, pp. 163-175, 2004.

[31] W. Schindler, K. Lemke and C. Paar. A Stochastic Model for Differential Side Channel Cryptanalysis, in *CHES 2005*, LNCS 3659, pp. 30-46, 2005.

[32] F.-X. Standaert, T.G. Malkin and M. Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks, in *EUROCRYPT 2009*, LNCS 5479, pp. 443-461, 2009.

[33] M. Soos, K. Nohl and C. Castelluccia. Extending SAT Solvers to Cryptographic Problems, in *SAT 2009*, LNCS 5584, pp. 244-257, 2009.

[34] N. Veyrat-Charvillon, B. Gerard, M. Renauld and F.-X. Standaert. An optimal key enumeration algorithm and its application to side-channel attacks, in *SAC 2012*, LNCS 7707, pp. 390-406, 2012.

[35] N. Veyrat-Charvillon, B. Gerard and F.-X. Standaert. Security Evaluations beyond Computing Power, in *EUROCRYPT 2013*, LNCS 7881, pp. 126-141, 2013.
[36] X.J. Zhao, F. Zhang, S.Z. Guo, T. Wang, Z.J. Shi, H.Y. Liu and K.K. Ji. MDASCA: An Enhanced Algebraic Side-Channel Attack for Error Tolerance and New Leakage Model Exploitation, in *COSADE 2012*, LNCS 7275, pp. 231-248, 2012.
[37] X.J. Zhao, S.Z. Guo, F. Zhang, T. Wang, Z.J. Shi, H.Y. Liu, K.K. Ji and J. Huang. Efficient Hamming Weight Based Side-Channel Cube Attacks on PRESENT, *The Journal of Systems and Software*, vol. 86, no. 3, pp. 728-743, 2012.

Appendix 1: IDASCA on AES under TLM

As noted in [23], the leakage of certain devices (e.g. in 65nm and smaller technologies [29]) cannot always be precisely expressed with HWLM, but can be with TLM. Under TLM, there are 256 candidates for the value of an intermediate byte.

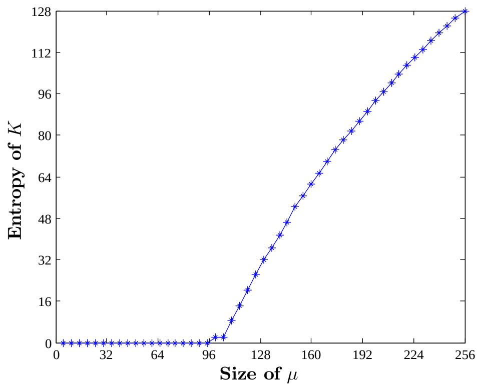
In CHES 12, Oren *et al.* first applied prob-TASCA to AES implemented on a 65nm ASIC under HWLM [23]. They simulated 100 leaks in the first round of AES according to the model of the 65nm ASIC implementing one AES S-Box and used prob-TASCA to recover the secret key with fixed μ approach. The results showed that when $\mu \geq 100$, the decoding success rate ρ_d and key recovery success rate ρ_k are all 100%. Even when $\mu = 256$, if the rank of the correct deduction for each leak is less than 14, prob-TASCA can still succeed on AES with 62254 seconds on average. It is interesting to investigate how IDASCA works with TLM and interpret prob-TASCA on AES in this scenario.

In the attack, we simulated the leaks from the PIC device as in [23]. A C program is used to generate the deduction set according to the value of μ, the value of the probability for every deduction of a byte are also simulated. Let δ denote the rank of the correct deduction in the μ deductions. We conduct three types of the attack on the first round of AES. 100 random attack instances are launched for each type.

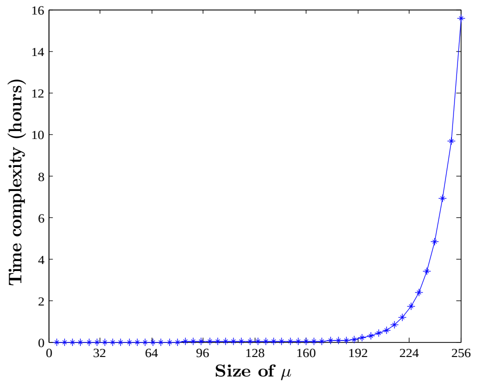
(1) Fixed μ, without consider the probability of different deduction candidates

In this scenario, we set $\delta = \mu$, which means that the correct deduction is always with the lowest probability and is the worst case in [23]. As in [23], we conduct the attacks under different μ. The entropy of K and time complexity required in IDASCA on AES for different μ are shown in Fig. 9(a) and Fig. 9(b), respectively .

From Fig. 9(a), the value of $E(K)$ is less than 48 when $\mu \leq 145$. In the attack, when μ is small, the attack complexity of IDASCA is lower, but the success rate of the attack is also smaller due to the errors. The decoding success rate, ρ_d, can reach 100% when $\mu \geq 100$. For $\mu = 100$, IDASCA can reduce the key search space of AES to $2^{1.26}$ within 0.01 second. From Fig. 9(b), the time complexity for $\mu \leq 145$ is less than one minute and is affordable in practice. When $\mu = 256$, the time complexity required for IDASCA is less than 16 hours, but it cannot recover any bit of the secret key. In our experiments, when $\mu = 100$, we were able to extend the attack to cover unknown plaintext/ciphertext scenario and successfully recover the master key of AES by exploiting the leaks in any two consecutive rounds.



(a) Entropy of K



(b) Time complexity

Fig. 9. Results of IDASCA on AES with different size of μ (TLM)

(2) Dynamic μ, without consider the probability of different deduction candidates

IDASCA can also be extended to attack the first round of AES under TLM and known plaintext scenario using the dynamic μ, which enable a smaller reduced key search space for large range on the value of μ. For example, when $64 < \mu \leq 90$ (15%), $90 < \mu \leq 120$ (22%), $120 < \mu \leq 150$ (23%), $150 < \mu \leq 200$ (22%), $200 < \mu \leq 256$ (18%) and $\delta = \mu$, IDASCA can reduce the key search space of AES to $2^{35.4}$ within one minute.

(3) Interpreting prob-TASCA on AES under HWLM with IDASCA (Fixed μ, consider the probability of different deduction candidates)

In prob-TASCA [23], for all the 100 leaks, δ is manually set to be less than 15 ($1 \leq \delta \leq 14$). In fact, such leakage is even more informative than the $\mu = 1$ under HWLM (which equals $\mu = 50$ under TLM). Since δ is so small in [23], the SCIP solver always enumerates the deduction from the highest probability to the lowest. This is why prob-TASCA can succeed even when $\mu = 256$. In fact, in this case, a smart adversary can simply set $\mu = 14$, and then both MDASCA [36], IASCA [20] and IDASCA can also succeed within one second.

IDASCA can also be used to exploit the probability of different deduction candidates by modifying Algorithm 2. Take recovering $\{k_0^0, k_5^0, k_{10}^0, k_{15}^0\}$ as an example, we first generate the 256-sized deduction set ($\mu = 256$) for these 25 leaks. Then, we modify Algorithm 2 using the similar idea in Section 6.3 and compute the maximized sum (S_h) on the values of the probabilities for all the candidates of $a_0^0, a_5^0, a_{10}^0, a_{15}^0$. The results show that if $\delta = 14$, the output optimized solution is always the correct $a_0^0, a_5^0, a_{10}^0, a_{15}^0$. Then, $\{k_0^0, k_5^0, k_{10}^0, k_{15}^0\}$ can be recovered. To recover full 16 bytes of K, the time complexity of our attack is 56046 seconds on average (the maximized solving time is 60234 seconds), which is comparable to 62254 seconds on average (the maximized solving time is 48+ hours) required in [23]. Moreover, we also extend our attack to when $\delta = 100$, $\rho_d = \rho_k = 95\%$, which is more robust than $1 \leq \delta \leq 14$ in prob-TASCA.

Shize Guo was born in 1964. He received his Ph.D. degree in Harbin Institute of Technology in 1989 and his M.S. and B.S. degrees from Ordnance Engineering College, China, in 1991 and 1988, respectively. He is currently a researcher in Institute of North Electronic Equipment and also a Professor in Beijing University of Post and Telecommunications. His main research interest includes information security and cryptography.

Xinjie Zhao received his Ph.D., M.S. and B.S. degrees in Department of Information Engineering, Ordnance Engineering College in 2012, 2009 and 2006, respectively. He is currently an Engineer in Institute of North Electronic Equipment. His main research interest includes side channel analysis, fault analysis and combined analysis of block ciphers. He won the best paper award in Darmstadt - the 3rd International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE 2012).

Fan Zhang was born in 1978. He received his Ph.D. degree in the Department of Computer Science and Engineering from University of Connecticut in 2012. His research interests include side channel analysis and fault analysis in cryptography, computer architecture, security in wireless sensor network, and more. Currently he is working in the Department of Information Science & Electrical Engineering, Zhejiang University, China.

Tao Wang was born in 1964. He received his Ph.D. degree in computer application from Institute of Computing Technology Chinese Academy of Sciences in 1996 and masters degree in computer application from Ordnance Engineering College in 1990. He is currently a Professor in Ordnance Engineering College. His research interests include information security and cryptography.

Zhijie Jerry Shi is currently an Associate Professor of Computer Science and Engineering at the University of Connecticut. He received his Ph.D. degree from Princeton University in 2004 and his M.S. and B.S. degrees from Tsinghua University, China, in 1996 and 1992, respectively. He is a member of IEEE and ACM. Dr. Shi received US National Science Foundation CAREER award in 2006. His current research interests include hardware mechanisms for secure and reliable computing, side channel attacks and countermeasures, primitives for cipher designs, embedded system designs, underwater sensor networks, and sensor network security.

Francois-Xavier Standaert was born in Brussels, Belgium in 1978. He received the Electrical Engineering degree and PhD degree from the Universite catholique de Louvain, respectively in June 2001 and June 2004. In 2004-2005, he was a Fulbright visiting researcher at Columbia University, Department of Computer Science, Network Security Lab and at the MIT Medialab, Center for Bits and Atoms. In March 2006, he was a founding member of IntoPix s.a. From 2005 to 2008, he was a post-doctoral researcher of the UCL Crypto Group and a regular visitor of the two aforementioned laboratories. Since September 2008, he is associate researcher of the Belgian Fund for Scientific Research (F.R.S.-FNRS) and professor at the UCL Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM). In 2010, he was program co-chair of CHES (IACR's flagship workshop on cryptographic hardware). In June 2011, he has been awarded a Starting Independent Research Grant by the European Research Council. His research interests include digital electronics, FPGAs and cryptographic hardware, low power implementations for constrained environments (RFIDs, sensor networks, ...), the design and cryptanalysis of symmetric cryptographic primitives, physical security issues in general and side-channel analysis in particular.

Chujiao Ma is currently a PhD Candidate of Computer Science and Engineering at the University of Connecticut. She received her Bachelor of Science degree from Franklin W. Olin College of Engineering in 2010. She was the recipient of Graduate Assistance in Area of National Needs (GAANN) fellowship from 2010 to 2013. Her research interests focus on computer security, more specifically side channel attacks and countermeasures, as well as network security.