A Comprehensive Study of Multiple Deductions-based Algebraic Trace Driven Cache Attacks on AES

Xinjie Zhao[a,*], Shize Guo[b], Fan Zhang[c,**], Tao Wang[a], Zhijie Shi[c], Zhe Liu[d], Jean-François Gallais[d]

[a]*Department of Computer Engineering, Ordnance Engineering College, Shijiazhuang 050003, China*
[b]*The Institute of North Electronic Equipment, Beijing 100083,China*
[c]*Department of Computer Science and Engineering,University of Connecticut, Storrs 06269, USA*
[d]*Laboratory of Algorithmics, Cryptology and Security (LACS), University of Luxembourg, L-1359, Luxembourg.*

Abstract

Existing trace driven cache attacks (TDCAs) can only analyze the cache events in the first two rounds or the last round of AES, which limits the efficiency of the attacks. Recently, Zhao et al. proposed the multiple deductions-based algebraic side-channel attack (MDASCA) to cope with the errors in leakage measurements and to exploit new leakage models. Their preliminary results showed that MDASCA can improve TDCAs and attack the AES implemented with a compact lookup table of 256 bytes. This paper performs a comprehensive study of MDASCA-based TDCAs (MDATDCA) on most of the AES implementations that are widely used. First, the key recovery in TDCA is depicted by an abstract model regardless of the specific attack techniques. Then, the previous work of TDCAs on AES is classified into three types and its limitations are analyzed. How to utilize the cache events with MDATDCA is presented and the overhead is also calculated. To evaluate MDATDCA on AES, this paper constructs a mathematical model to estimate the maximal number of leakage rounds that can be utilized and the minimal number of cache traces required for a successful MDATDCA. Extensive experiments are conducted under different implementations, attack scenarios and key lengths of AES. The experimental results are consistent with the theoretical analysis. Many improvements are achieved. For the first time, we show that TDCAs on AES-192 and AES-256 become possible with the MDATDCA technique. Our work attests that combining TDCAs with algebraic techniques is a very efficient way to improve cache attacks.

Keywords: Multiple deductions, Algebraic side-channel attack, Trace driven, Cache attack, Error-tolerant, AES-128/192/256.

1. Introduction

Cache attacks are a class of Side-channel attacks (SCAs) that extract the secret from the behavior of cache in the processors. These attacks utilize the fact that a cache miss has a different profile of leakages from

*Corresponding author. Email: zhaoxinjieem@163.com.
**Corresponding author. Email: fan.zhang@engineer.uconn.edu.

a cache hit. Cache attacks demonstrated fall into three categories, depending on the channels used to collect the leakages. These channels are spy processes [1], timing information [2, 3] and power/electromagnetic (EM) traces [4, 5, 6, 7, 8, 9, 10, 11]. The focus of this paper is trace driven cache attack (TDCA), which exploits the power or electromagnetic traces.

With direct access to the cryptographic device, the adversaries can monitor the power/EM traces, which minimizes the invasion to the device. As the name suggests, TDCAs monitor cache hits and misses from power/EM traces, and recover the secret key used in the computation. The number of traces required in TDCAs is much less than in the conventional differential power attacks (DPAs) [12], correlation power attacks (CPAs) [13] or other types of cache attacks [1, 2, 3]. Considering AES for example, only 30 cache traces are required in TDCAs [9, 10] instead of hundreds (or thousands) of power traces in DPAs, CPAs [13], hundreds of cache traces in access driven cache attacks [1], and millions of cache traces in timing driven cache attacks [2, 3].

AES was targeted in many TDCAs [4, 5, 6, 7, 8, 9, 10, 11]. Throughout this paper, AES refers to AES-128 by default. Bertoni et al. [6] showed that the cache traces manifested in the power profiles can be used to reveal the secret key. The cache events in the first round of AES S-Box lookups[1] implemented with a table of 256 bytes were estimated from power simulations and analyzed in [6]. Further research in TDCAs on AES splits into two directions. One is about exploiting new and real leakages in TDCAs, where cache traces were collected from real power consumptions in [8, 9] and from EM in [10]. The other is improving the efficiency of TDCAs on different AES implementations. In attacks on AES with large lookup tables (e.g., 1K bytes), TDCAs can exploit cache events in the first round [11], the first two rounds [4] or the last round [5, 7]. For AES implemented with a compact table (256 bytes), TDCAs can exploit the cache events in the first round [6], or the first two rounds [8, 9, 10]. This paper is under the latter direction and tries to improve TDCA on AES.

In the aforementioned TDCAs on AES, the cache events utilized are limited to the first 20 table lookups in the first two rounds because of the avalanche effect. Since the traces are captured for entire encryptions, exploiting the cache events in the third and later rounds can improve the efficiency of TDCA. Combining TDCAs with algebraic techniques and conducting algebraic side-channel attacks (ASCA) [14, 15, 16, 17] is a very promising way to improve TDCA. Previous ASCA mainly focused on power based Hamming weight leakage model [14, 15, 17] or Hamming distance leakage model [16]. The original ASCA [14, 15] can only work when the deduction on the targeted states is single and correct. The error tolerant ASCA in [16, 17] can only work with limited deductions where the variance of the error is small and fixed. Previous ASCA cannot be directly and easily combined with TDCA because in practice, there are multiple deductions and the variance of the errors are large and uncertain.

[1] S-Box is usually implemented with lookup table and S-Box lookup is identical with the table lookup throughout the paper.

In COSADE 2012, Zhao et al. proposed the multiple deductions-based ASCA (MDASCA) [18]. The work in [18] showed that, due to the inaccurate measurements or the interferences from other components in the cryptosystem, the deduction on the targeted intermediate state from SCA is not always correct. As a result, attacks have to deal with the fact that the correct value is among multiple candidates obtained during the process, which are also referred to as multiple deductions. How to represent and utilize these multiple deductions is critical to improving the error tolerance and exploiting new leakage models for ASCAs [14, 15, 16, 17]. They showed that MDASCA can utilize the leakages in the first three rounds of AES in TDCAs. Under error-free attack scenario[2], the number of cache traces required to attack the AES implemented with a compact lookup table of 256 bytes can be reduced to only five. However, there remain some questions to be answered. For different AES implementations (e.g., AES implemented with 1KB, 2KB tables in OpenSSL cryptography library [19]), different attack scenarios (e.g., error-tolerant scenario [10], partial preloaded cache scenario [7, 9, 10]), different AES key lengths (e.g., AES-192/256), how many rounds of leakages can MDASCA exploit, how much can MDASCA improve TDCAs in terms of the data complexity, and what are the new scenarios where we can apply MDASCA in cache attacks?

This paper aims to answer these questions and gives a systemic and comprehensive study of the multiple deductions-based algebraic TDCAs (MDATDCAs). The rest of this paper is organized as follows. Section 2 describes the notations used throughout the paper. In Section 3, we formalize the key recovery problem in TDCA with an abstract model, which is independent of specific TDCA techniques [4, 5, 6, 7, 8, 9, 10, 11, 18]. Then we categorize previous TDCAs on AES into three types and study their limitations in Section 4. In Section 5, we describe the detailed procedure of MDATDCA and analyze the overhead. To evaluate the efficiency of MDATDCA on AES, we build a mathematical model in Section 6 to estimate the maximal number of leakage rounds that can be exploited and the minimal number of cache traces required in a successful MDATDCA. Unlike previous work that can only analyze the cache events in the first round, this paper can analyze any cache events. The attack setup is described in Section 7. The preliminary results under an error-free scenario are presented in Section 8 to verify the theoretical analysis results. The results with different error rates are showed in Section 9. MDATDCAs on AES with partially preloaded cache are described in Section 10. To demonstrate the power of MDATDCA, we extend the attack to AES-192/256 in Section 11. Finally, we conclude the paper in Section 12.

[2]In TDCA, *error* means that the deduction for the cache events (hit or miss) from side channel leakages is incorrect. *error-free attack scenario* means that the adversary can deduce all the cache events correctly in one attack. Comparatively, *error-tolerant attack scenario* means that there exists errors when deducing some cache events in an attack.

2. Notation

Throughout the paper, P denotes the public variable (plaintext or ciphertext) and K denotes the targeted secret variable (the master key or equivalent key). Variables p_i and k_i denote the i-th ($i \geq 0$) part in P and K, respectively. Each part contains l bits. Let q_j denote the j-th table lookup in the execution of block ciphers, λ denote the number of table lookups considered in the attack, $0 \leq j < \lambda$. H and M denote whether q_j is a cache hit or miss respectively. y_j denotes the index of the lookup q_j. U_j and V_j are the set of p_i and k_i that represent y_j, where $U_j \subseteq P, V_j \subseteq K$. Let $f^j(\cdot)$ be the function that computes y_j from U_j and V_j, $y_j = f^j(U_j, V_j)$. Suppose each entry in the table has 2^e bytes and each cache line has 2^δ bytes. Let $\langle y_j \rangle_b$ be the b most significant bits (MSBs) of y_j leaked in q_j, $b = l - (\delta - e)$. Assume q_t is the t-th targeted cache events in TDCA and y_t denotes the related table lookup index. Suppose among the first $t-1$ lookups, there are n cache misses in n different table lookups $q_{M_1}, \ldots, q_{M_n}$, which form a set $O_M^t = \{M_1, \ldots, M_n\}$ $(n < t)$. Let S_M^t be the set of the b MSBs of the indexes $y_{M_1}, \ldots, y_{M_n}$, i.e., $S_M^t = \{\langle y_{M_1} \rangle_b, \langle y_{M_2} \rangle_b, \ldots, \langle y_{M_n} \rangle_b\}$.

3. The TDCA Problem

In this section, we propose an abstract model which can be used to generalize all TDCAs.

TDCA on a block cipher is illustrated in Fig.1, where λ lookups are considered. Observing power or EM traces, one can detect whether q_j is a cache hit (H) or miss (M), $0 \leq j < \lambda$. From Fig.1 we can see that a cache miss has a distinct amplitude peak than a cache hit (Note also that the amount of clock cycles is distinctly different). The goal of TDCA is to extract the value of all k_i in K (the secret key) from the knowledge of the p_is (known public variables) and q_js (cache events).

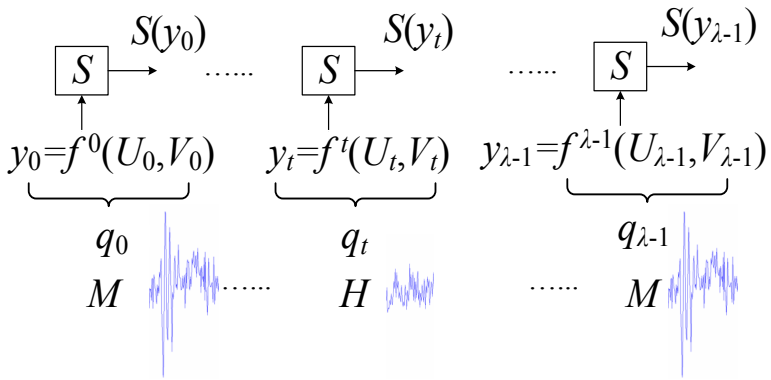


Figure 1: S-Box (Table)Look-up structure targeted in TDCA

Suppose the cache contains no data from the table before each encryption. As to the analysis of the cache event in q_t, suppose q_j is the only cache miss before q_t. A cache hit of q_t means both y_t and y_j access the same cache line. Eq.(1) holds if q_t is a cache hit.

$$\langle y_t \rangle_b = \langle y_j \rangle_b \Longrightarrow \quad \langle f^t(U_t, V_t) \rangle_b = \langle f^j(U_j, V_j) \rangle_b \tag{1}$$

The key technique of TDCA is to use Eq.(1) to reduce the search space of $V_t \bigcup V_j$, which converges to K if t is large enough. Since both U_t and U_j are known, the adversary can check all the assignments to those k_i in $V_t \bigcup V_j$ with Eq.(1). If Eq.(1) is satisfied, the assignment is a possible value for k_i. Otherwise, the assignment is an incorrect guess. Similarly, if q_t is a cache miss, Eq.(2) can be used in the key search.

$$\langle y_t \rangle_b \neq \langle y_j \rangle_b \Longrightarrow \quad \langle f^t(U_t, V_t) \rangle_b \neq \langle f^j(U_j, V_j) \rangle_b \tag{2}$$

From Section 2, there are n cache misses $q_{M_1}, \ldots, q_{M_n}$ before the first $t-1$ lookups. The set $O_M^t = \{M_1, \ldots, M_n\}$ $(n < t)$ can be used to build n additional equations (or inequations). If q_t is a hit, only one of $q_{M_1}, \ldots, q_{M_n}$ accesses the same cache line as q_t (because if two of them are in the same cache line as q_t, one of them must be a hit). In this case, Eq.(3) holds

$$\begin{aligned} \exists j \in O_M^t : \quad & \langle f^t(U_t, V_t) \rangle_b = \langle f^j(U_j, V_j) \rangle_b \\ \forall j^* \in O_M^t \wedge (j^* \neq j) : \quad & \langle f^t(U_t, V_t) \rangle_b \neq \langle f^{j^*}(U_{j^*}, V_{j^*}) \rangle_b \end{aligned} \tag{3}$$

If q_t is a miss, Eq.(4) holds

$$\forall j \in O_M^t : \quad \langle f^t(U_t, V_t) \rangle_b \neq \langle f^j(U_j, V_j) \rangle_b \tag{4}$$

Using the n equations (inequations) in (3) or (4), more assignments to key bits in $V_{M_1} \bigcup \cdots \bigcup V_{M_n} \bigcup V_t$ can be verified. The key recovery is converted into the problem of how to converge $V_{M_1} \bigcup \cdots \bigcup V_{M_n} \bigcup V_t$ to the master key K with cache events. In TDCA, the adversary can analyze different table lookups and traces until the search space of K is reduced to a level where a brute-force attack is feasible.

The above abstract model can help us to understand the TDCA problem and is generic to block ciphers using the S-Box (table) lookup structure [4, 5, 6, 7, 8, 9, 10, 11, 24, 25, 26, 27, 28]. Different attack techniques can be developed to solve this problem, such as traditional TDCA technique [4, 5, 6, 7, 8, 9, 10, 11], MDASCA technique [18] or others to be proposed in the future.

4. Analysis of Previous Work

4.1. AES implementations

All the AES implementations can be categorized into three types based on (1) g_t, the number of the lookup tables; (2) g_s, the size of the lookup tables; (3) g_l, the number of lookups in one round that access the same table; (4) g_c, the size of the cache line, where $g_c = 2^\delta$. Note that the scope of this paper is about AES implementations that use one or more lookup tables for the sole S-Box, and not the lookup tables for the field multiplication in the MixColumns operation of AES [8].

Type A. A single table is used in all rounds, $g_t = 1$. Each round has 16 table lookups, $g_l = 16$. The size of the lookup table can be 256B (256 bytes) or 2KB (2K bytes). The examples of this type include the implementation studied in [8, 9, 10, 20] where g_s=256B, g_c=16B, and b=4, and an implementation in OpenSSL v1.0.0d [19], where g_s=2KB, g_c=64B, and b=5. Recall b is the number of bits revealed from one table lookup.

Type B. There are four 1KB tables, $g_t = 4$. Each round has 16 table lookups, where each table is accessed four times, $g_l = 4$. For example, AES in OpenSSL v0.9.8j uses four 1KB tables, T_0, T_1, T_2, T_3, in all the 10 rounds, and OpenSSL v0.9.8a uses them in the first 9 rounds. In this paper, as to *Type B*, we mainly focus on AES in OpenSSL v0.9.8j.

Type C. There is only one 1KB table in the last round ($g_t = 1$), where it is accessed 16 times ($g_l = 16$). For example, OpenSSL v0.9.8a uses table T_4 in the 10-th round.

Table. 1. depicts the list of widely used AES implementations targeted in TDCAs.

Table 1: List of widely used AES implementations targeted in TDCAs

implementation	Type	g_t	g_s	g_l	g_c	TDCAs
standard NIST implementation [20]	*Type A*	1	256B	16	16B, 32B, 64B etc	[6, 8, 9, 10]
AES in OpenSSL v0.9.8a [19]	*Type B,C*[1]	5	1KB	4,16[2]	16B, 32B, 64B etc	[5, 7]
AES in OpenSSL v0.9.8j [19]	*Type B*	4	1KB	4	16B, 32B, 64B etc	[4]
AES in OpenSSL v1.0.0d [19]	*Type A*	1	2KB	4	16B, 32B, 64B etc	

4.2. TDCA on AES of Type A

Bertoni et al. performed the first TDCA on an implementation of *Type A* [6], which focused on the cache events of the first round of AES. To further reduce the key search space and the number of plaintexts (or power traces) required, attacks in [8, 9, 10] also utilized some cache events in the second round.

In [8], equations are generated only from the cache hits as shown in Eq.(3). In the first round, $U_j = p_j, V_j = k_j$ and $U_t = p_t, V_t = k_t$, $(0 \leq j < t \leq 15)$. $f^j(\cdot)$ is the bitwise XOR and $\langle k_j \oplus k_t \rangle_4$ can be derived. It is shown in [8] that the search space of the AES key can be reduced to 2^{68} with at most 240 adaptive chosen plaintexts. To improve the attack, the work in [8] also considered the case where the first two lookups in the second round (q_{16} and q_{17}) are cache hits. Consequently, more variables are used in Eq.(3) and $f^j(\cdot)$ becomes more complicated. Considering q_{16} as an example (t=16), suppose n=8, and

[1] *Type B* for TDCA on the first nine rounds and *C* for TDCA on the 10-th round

[2] 4 for the first nine rounds, and 16 for the 10-th round.

$O_M^{16} = \{M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8\} = \{0, 2, 4, 5, 7, 8, 10, 14\}$. For each q_{M_i} $(1 \leq i \leq 8)$, the key search space of $V_{M_i} \bigcup V_{16}$ can be reduced via Eq.(5).

$$\begin{aligned} &\langle y_{M_i} \rangle_4 = \langle y_{16} \rangle_4 \Longrightarrow \langle f^{M_i}(U_{M_i}, V_{M_i}) \rangle_4 = \langle f^{16}(U_{16}, V_{16}) \rangle_4 \\ &\Longrightarrow \langle p_{M_i} \oplus k_{M_i} \rangle_4 = \langle 02 \cdot S[p_0 \oplus k_0] \oplus 03 \cdot S[p_5 \oplus p_5] \oplus S[p_{10} \oplus k_{10}] \oplus \\ &\qquad S[p_{15} \oplus k_{15}] \oplus S[k_{13}] \oplus 01 \oplus k_0 \rangle_4 \end{aligned} \tag{5}$$

From Eq.(5), $U_{16} = \{p_0, p_5, p_{10}, p_{15}\}$ and $V_{16} = \{k_0, k_5, k_{10}, k_{13}, k_{15}\}$, so $f^{16}(U_{16}, V_{16})$ becomes more complicated than that in the first round. The result in [8] is that 1280 chosen plaintexts are required to reduce the key search space to 2^{24}.

In WISA 2010, under known plaintext scenarios, Gallais et al. analyzed the cache misses along with the hits in [8]. They showed that on average 19.43 known plaintexts can reduce the key search space to 2^{68} if q_j $(0 \leq j \leq 15)$ is analyzed. 30 plaintexts can further reduce the search space to 2^{30} if q_{16} and q_{17} are added. Later in COSADE 2011, Gallais et al. [10] extended the analysis to q_{18}, q_{19}, and showed that 30 known plaintexts can reduce the key search space to 10.

4.3. TDCA on AES of Type B

Acıiçmez [4] presented the first TDCA on AES for such implementation, in which four lookup tables are used for each round and each is accessed four times. In TDCA on the first round, only four groups of $\langle k_j \oplus k_t \rangle_b$ $(j \equiv t \bmod 4)$ can be derived and the key search space can be reduced to $2^{128-4\times(4-1)\times b}$. More key bits can be derived via the analysis of the second round. In [4], $q_{16} \ldots q_{19}$ are added. They showed that about 40 and 55 power traces were required to reduce the key search space to $2^{33.50}$ and $2^{34.26}$ for $b = 4$ and 5 respectively. Acıiçmez [4] also pointed out that TDCA on the third or the deeper rounds was an open problem.

4.4. TDCA on AES of Type C

The work in [5, 7] showed that under *Type C* implementations, TDCA on the final round of AES is much more effective than in the first round. As to the first round of AES, $f^t(\cdot)$ is a linear function and only the higher b bits of $k_j \oplus k_t$, $(0 \leq j < t \leq 15)$ are leaked. While in the last round, $f^t(\cdot)$ becomes a complicated nonlinear function. As a result, all the 8-bits of $k_{j'}^{10}$ and k_t^{10} $(j' \equiv 5j \bmod 16)$ can be derived where j' is the j-th byte in the ciphertext. The work in [5, 7] requires only 14 and 25 traces to reduce the key search space to 2^{30} when $b = 4$ and 5 respectively.

4.5. Limitation of previous TDCAs

Traditional TDCAs rely on the representation of y_t in different rounds. As noted in [4], the full avalanche effect is achieved in the third round, where $f^t(U_t, V_t)$ becomes complicated if a manual analysis is conducted.

Because of this, current traditional TDCAs on AES [4, 5, 6, 7, 8, 9, 10, 11] can at most analyze the cache events in the first two rounds, more precisely, from the first 20 lookups. Moreover, all current TDCA works are for AES-128. As to AES with longer key lengths (e.g., AES-192 and AES-256), the key expansion algorithms become more complicated and the first 20 lookups only leak partial bits of the master key. How to recover more key bits and conduct effective TDCA on them are still open problems.

The manual representation of table indexes is awkward. It is imperative to provide a tool for the analysis of the leakages in deeper rounds in order to improve the attacks. Combining algebraic techniques with TDCA seems to be interesting and promising. The MDASCA proposed by Zhao et al. [18] in COSADE 2012 is a generic method to exploit many types of side-channels leakages with algebraic techniques.

5. MDASCA-based Trace Driven Cache Attacks (MDATDCAs)

In TDCA, the key issue is to obtain the cache events related to table lookups and to represent the possible (and/or impossible) candidates of lookup indexes with equations. We will use the same notations in Section 2 in the following discussion. Recall that there are n misses $q_{M_1}, \ldots, q_{M_n}$ before q_t. As to the set of the b MSBs of the indexes, $S_M^t = \{\langle y_{M_1}\rangle_b, \langle y_{M_2}\rangle_b, \ldots, \langle y_{M_n}\rangle_b\}$, let d be the correct value (or deduction) of $\langle y_t\rangle_b$, d^j be the j-th bit of d, d_i be the i-th element in S_M^t, and d_i^j be the j-th bit of d_i. If q_t is a cache hit, the cache line that includes the index y_t has been loaded into the cache by earlier table lookups, which means that d is equal to only one element of the multiple deductions in S_M^t. If q_t is a cache miss, the cache line that includes y_t has not been accessed, which means that d is not equal to any element of the multiple deductions in S_M^t.

From above, we can see that how to represent the relations between y_t and its multiple possible or impossible deductions for cache miss event is the most challenging part in TDCA. The work in [18] proposes a generic method to convert the multiple deductions into algebraic equations and applies it to TDCA. For convenience, we will refer to *the MDASCA-based trace driven cache attack* or so called *multiple deductions-based algebraic TDCA* as MDATDCA. In this type of attack, the cipher is first represented with a system of algebraic equations. Then the cache hit/miss events are profiled via power/EM measurements and then convert the multiple deductions on b MSBs of the lookup index for each cache event into equations, which are added into the original equation system of the cipher. Finally, the secret key is recovered by solving the whole equation system [21, 22]. More details about MDASCA can be found in [18]. Next, we will describe the core of MDATDCA, which is to represent cache hit and miss events with algebraic equations.

5.1. Representing a cache hit

Let D be the possible deduction set of d and s_p be the size of D, where $D = S_M^t$. A one-bit variable c_i is introduced to represent whether d_i is equal to d or not. Another one-bit variable e_i^j is also introduced

to represent whether ${d_i}^j$ is equal to d^j. Then c_i can be represented with Eq.(6), where $\neg$ denotes the NOT operation.

$$e_i^j = \neg(d_i^j \oplus d^j), \quad c_i = \prod_{j=1}^{b} e_i^j \tag{6}$$

Only one d_i is equal to d (c_i is 1 then), which can be represented as:

$$c_1 \vee c_2 \vee \ldots \vee c_{s_p} = 1, \quad \neg c_i \vee \neg c_j = 1, \quad 1 \leq i < j \leq s_p \tag{7}$$

According to Eq.(6) and Eq.(7), $(b+1) \times s_p$ variables and $(b+1) \times s_p + \binom{s_p}{2}$ Algebraic Normal Form (ANF) equations are introduced to represent D.

5.2. Representing a cache miss

Let $\overline{D}$ denotes the impossible deduction set of $\langle y_t \rangle_b$ and s_n be the size of $\overline{D}$. Then $\overline{D} = S_M(y_t)$. e_i^j is also introduced as in Section 5.1. None of d_i in $\overline{D}$ is equal to d, which can be represented as:

$$e_i^j = \neg(d_i^j \oplus d^j), \quad c_i = \prod_{j=1}^{b} e_i^j = 0 \tag{8}$$

According to Eq.(8), $(b+1) \times s_n$ variables and $(b+1) \times s_n$ ANF equations are introduced to represent $\overline{D}$. As shown in this section, the algebraic equations for new constraints are quite simple. They can be easily fed into a solver, e.g., the SAT solver CryptoMiniSAT [22], to recover the key.

6. Evaluation of MDATDCAs on AES

For simplicity, this section only estimates the number of rounds that can be exploited, and the number of cache traces required in MDATDCAs on AES-128 under the error-free attack scenario, where the cache does not contain any AES data prior to each encryption. Extending these estimations to AES-192/256 is straightforward.

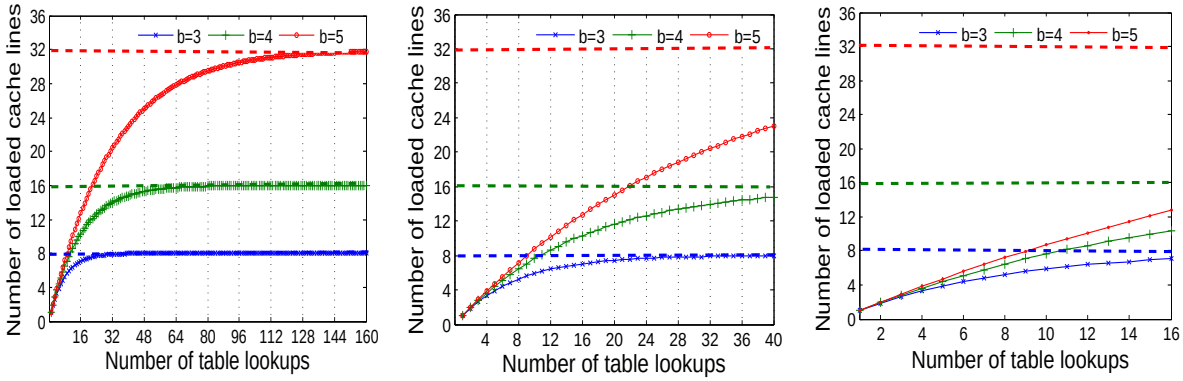
6.1. The Number of rounds that can be exploited

For convenience, $\mathbb{D}$ is used to denote the set of cache lines that will be filled up with data from lookup tables. Let m be the number of cache lines in $\mathbb{D}$, $m = 2^b$. Let $\mathbb{K}_t = V_{M_1} \bigcup \cdots \bigcup V_{M_n} \bigcup V_t$. E denotes the maximal number of rounds that can be utilized in MDATDCA. As long as $\mathbb{D}$ is not filled up, there may exist some cache misses (before q_t) that can be used for key recovery. To estimate E in MDATDCA, we introduce the metric n_t as in [4, 5, 18], which is the number of cache lines loaded after t table lookups.

For a cache line, the probability of not being filled after t lookups is $(\frac{m-1}{m})^t$. Then, after t lookups to the same table, the expected number of loaded cache lines, n_t, can be calculate as

$$n_t = m(1 - (\frac{m-1}{m})^t) \tag{9}$$

Fig. 2 shows how n_t changes with t and b in different types of implementations. The solid curves in blue, green and red are for the cases where $b = 3$, 4, and 5, respectively. We can see that, for *Type A* (or *B*), after the first 2/3/6 rounds (or 5/10/10 rounds), n_t is approaching to m (i.e., $\mathbb{D}$ is filled up). For *Type C*, all the 16 lookups in the last round can be used for key recovery.



(a) n_t of *Type A* (b) n_t of *Type B* (c) n_t of *Type C*

Figure 2: n_t for different AES implementations

6.2. The Number of cache traces required

The work in [18] presents a preliminary study of estimating the minimal number of cache traces required in TDCA. In this section, we introduce four metrics and adopt the information-theoretic approach to optimize the estimations on the minimal number of cache traces required for a successful MDATDCA.

(1) ρ_t: the ratio between the size of the search space of $\mathbb{K}_t$ after and before analyzing q_t

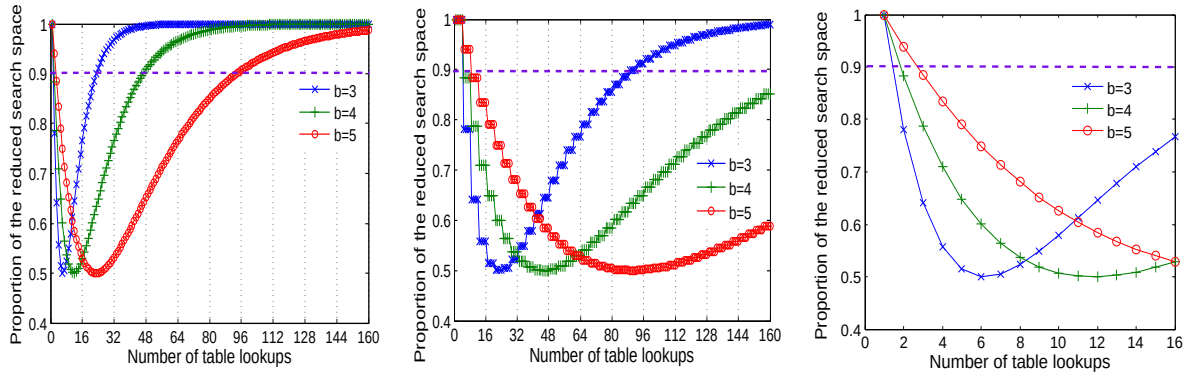
The probability that q_t is a cache hit is $\frac{n_t}{m}$. If q_t is a hit, the expected number for the candidates of $\langle y_t \rangle_b$ can be reduced from m to n_t. It is also easy to check the probability when q_t is a miss. Then ρ_t can be calculated as

$$\rho_t = (\frac{n_t}{m})^2 + (1 - \frac{n_t}{m})^2 \tag{10}$$

Fig. 3 shows how ρ_t changes with t and b. We choose $\rho_t \leq 0.9$ as the threshold (marked as a purple line in each subfigure). Note that the number of lookups read from the intersection between the purple and other lines can also be used to calculate the maximal number of rounds that can be utilized and the result is consistent with the earlier analysis.

(2) π_t: the number of key bits that can be derived from q_t

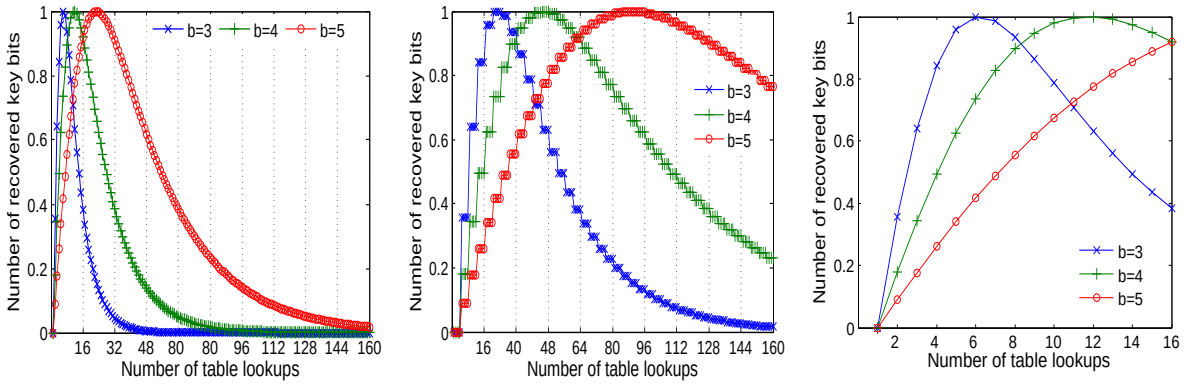
$$\pi_t \quad = -log_2(\rho_t) \tag{11}$$



(a) ρ_t of *Type A* (b) ρ_t of *Type B* (c) ρ_t of *Type C*

Figure 3: ρ_t for different AES implementations

Fig. 4 shows how π_t changes with t and b. We can see that (1) q_0 is always a cache miss. $\pi_0 = 0$. (2) $\pi_t \leq 1$ for all the values of t, which means that the number of key bits that can be extracted from adding one lookup is less than one. This can also be observed in Fig. 3 where $\rho_t \geq 0.5$ for all the curves.



(a) π_t of *Type A* (b) π_t of *Type B* (c) π_t of *Type C*

Figure 4: π_t for different AES implementations

(3) σ_i: the average number of key bits recovered in the i-th round of a trace

For AES of *Types A* and *B*, since all the key variables can be extracted using the cache events in the first two rounds, we just need to calculate σ_0 and σ_1. For *Type C*, we calculate σ_9 in the final round. Note that there are some intersects among $\mathbb{K}_t$ for different table lookups in practice, thus σ_i satisfies

$$\sigma_i \leq \sum_{z=16i}^{z=16i+15} \pi_z \tag{12}$$

(4) τ_i: the maximal number of key bits recovered in the i-th round

Let τ_0, τ_1, and τ_9 denote the maximal number of the key bits recovered in the first, second and last round. Since no information is leaked on the first access to each lookup table, according to Section 4, $\tau_0 = 15b$ and $\tau_1 = 128$ for *Type A*, $\tau_0 = 12b$ and $\tau_1 = 128$ for *Type B*, and $\tau_9 = 128$ for *Type C*.

With the introduction of all the aforementioned variables, we can now roughly estimate the minimal number of cache traces required to get τ_i bits in the i-th round, denoted as N_i. We calculate N_0 and N_9 as

$$N_i \geq \frac{\tau_i}{\sigma_i} \tag{13}$$

As τ_0 bits are recovered in the first round, we only need to recover the remaining 128-τ_0 bits in the second round. So we roughly calculate N_1 as

$$N_1 \geq \frac{128 - \tau_0}{\sigma_1} \tag{14}$$

Let E_N denote the estimated minimal number of cache traces needed in order to recover the master key. $E_N = \lceil max\{N_0, N_1\} \rceil$ for *Type A* and *B*. $E_N = \lceil N_9 \rceil$ for *Type C*. Table 2 lists the value of N_0, N_1, N_9 and E_N for different AES implementations. The value of E_N can help us to determine the number of traces needed in practical attacks. We will verify it with experiment results in Sections 7.

Table 2: N_i for different AES implementations

Type	b	N_0	N_1	N_9	E_N	b	N_0	N_1	N_9	E_N	b	N_0	N_1	N_9	E_N
Type A	3	4.25	34.69		**35**	4	5.06	6.88		**7**	5	8.72	3.42		**9**
Type B	3	4.89	5.93		**6**	4	11.77	6.49		**12**	5	28.43	9.45		**29**
Type C	3			12.10	**13**	4			10.79	**11**	5			14.88	**15**

7. Experiment Setup

The overall process of MDATDCA has been described in Section 5. Due to the page limit, here we only list a few important details about the setup. Each run of MDATDCA with different parameters is referred to as a *case*. Each case will be repeated many times and referred to as *instances*.

7.1. Build the AES equation set

How to represent the S-Box is the most difficult part in algebraic analysis. We adopt the technique in [23] to derive every S-Box output bit with high-degree equations (degree 7) from the eight S-Box input bits. More details can be found in Appendix 1.

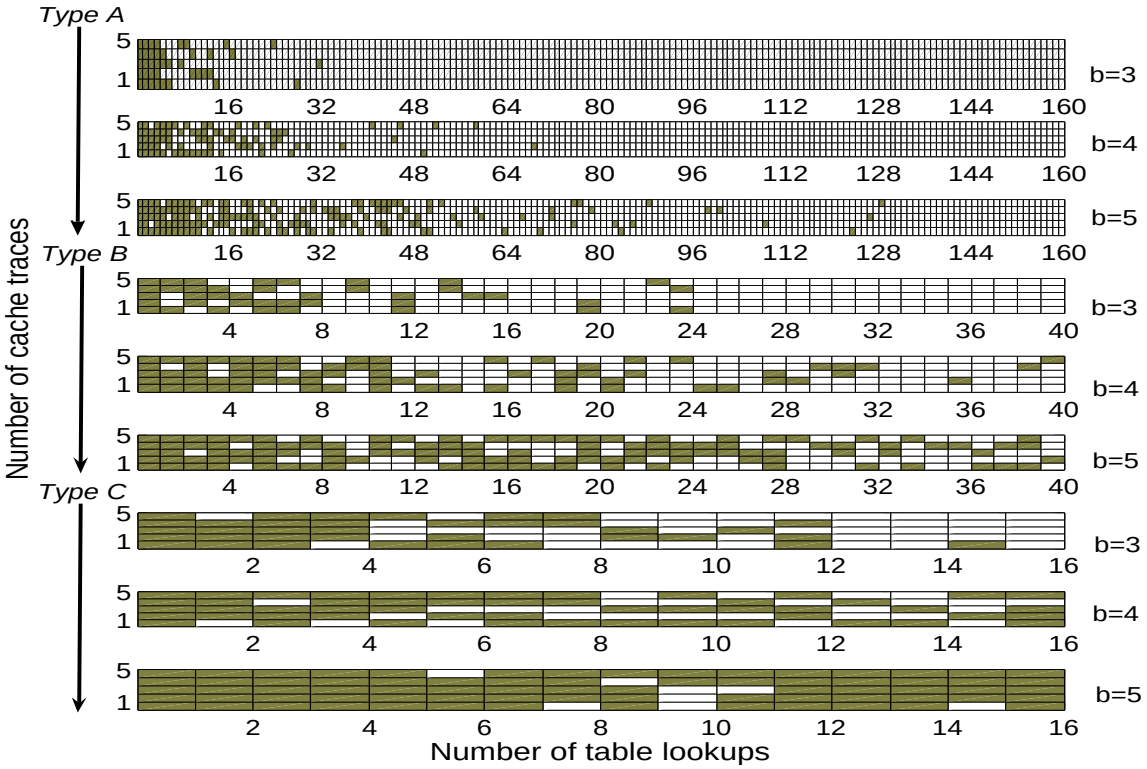


Figure 5: Cache hit and miss sequences for *Types A, B and C*

7.2. Profile the cache traces

This paper mainly focuses on the analysis part of MDATDCAs. Details of profiling the events can be found in [9, 10, 18]. In Section 8, 10, and 11, we assumed that the cache hits and misses are distinguishable in the EM traces. This can be achieved by modifying the AES source code in OpenSSL and generate the sequences of cache events under different configurations. Fig. 5 shows five traces for each implementation of *Types A B and C* where $b = 3, 4, 5$. A cyan rectangle stands for a cache miss and a white for a hit. The figure shows that for *Type A*, $\mathbb{D}$ of 8/16/32 cache lines cannot be filled up within the first 2/3/6 rounds. For *Type B*, $\mathbb{D}$ of 8/16/32 cache lines cannot be filled up within the first $5, 10, 10$ rounds. For *Type C*, $\mathbb{D}$ of 8/16/32 cache lines cannot be filled up within the final round. This is consistent with the theoretical analysis in Section 6.1.

To prove the feasibility of MDATDCA, in Section 9, we conduct concrete MDATDCA experiments against AES implemented with 256B compact table on 32-bit ARM microprocessor NXP LPC2124. In practice, the cache hits and misses are not always distinguishable from the EM traces, which are treated as uncertain cache events or errors. We propose a method to adapt MDATDCA to exploit these errors. More details can be found in Section 9.

7.3. Utilize the cache traces

We build additional equations from the generated cache events. When the leakage information is not enough, there may exist multiple solutions in solving for the key. The SAT solver may output a wrong but satisfied solution, which foils the whole attack, as noted by [16, 17]. In order to verify these multiple solutions, we append a set of new equations which describes a full AES encryption with a pair of known plaintext and ciphertext. Then, with this new approach, the correct key can always be derived within a reasonable time.

For each instance, we first try to solve the generated equations directly. However, some instances cannot be solved within a day. To accelerate the solving process, we give the guesses to n_k key bits first and run the exhaustive search for all the 2^{n_k} guesses. If the guess is correct, the solver can output the correct key within a reasonable amount of time. Otherwise, it will output "unsatisfiable" very quickly. As to *Types A,B,C*, we set $n_k = 4, 8, 4$, respectively. We can see that n_k is relatively small and the exhaustive search is affordable.

7.4. Solve the equation system

Many automatic tools can be used, such as Gröbner basis-based [21], or SAT-based solver [22]. We use a SAT-based solver, CryptoMiniSat 2.9.0 [22], on an AMD Athlon 64 Dual core 3600+ processor clocked at 2.0GHz.

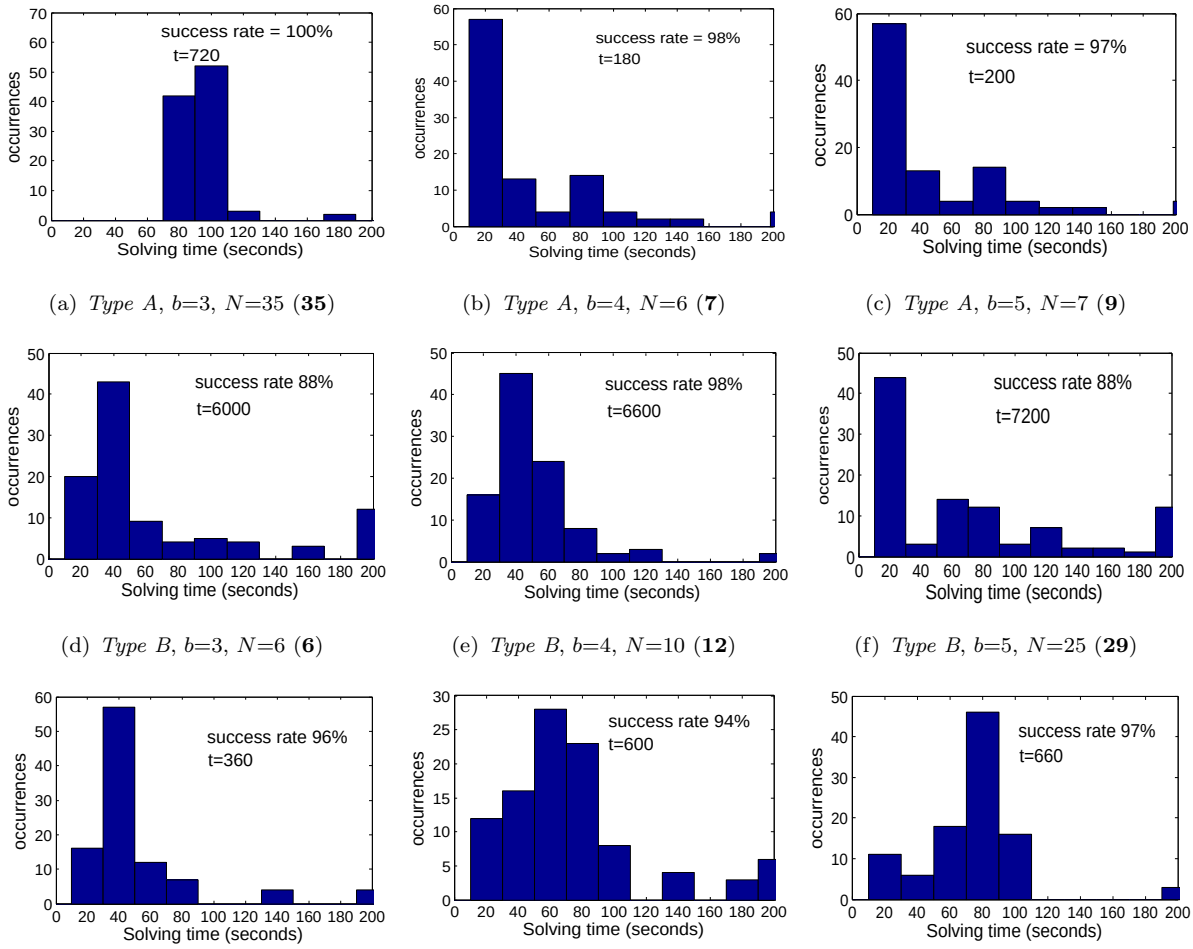
In Section 8, 9, and 10, three case studies are performed in MDATDCA on AES-128 considering different attack scenarios.

8. Case 1: Error-free MDATDCAs on AES

In this section, we conduct MDATDCA on AES under two assumptions. The first is that the cache does not contain any AES data prior to each encryption. The second is that the adversary can distinguish the cache miss event from the cache hit event precisely. We name this scenario as *MDATDCA on AES with error free*, which is also widely used in previous TDCA work [4, 5, 6, 7, 8, 9, 10, 11].

8.1. Data and time complexity

We conduct nine cases of MDATDCAs for *Types A, B, C* and $b = 3, 4, 5$. Then, for each case, we randomly generate a secret key and collect N cache traces for N different pairs of plaintexts and ciphertexts (N is chosen based on E_N in Table 2). For each case, we run 100 instances where the correct values of n_k key bits are fed into the equation set first). Let t denote the average full attack time (in seconds) when the correct key is retrieved in the exhaustive search procedure of guessing the 2^{n_k} key bits. Fig. 6(a)-6(i) show the distribution of the different solving times (in seconds) for the nine cases by analyzing N cache traces. The bold number is E_N.



(a) *Type A*, b=3, N=35 (**35**) (b) *Type A*, b=4, N=6 (**7**) (c) *Type A*, b=5, N=7 (**9**)

(d) *Type B*, b=3, N=6 (**6**) (e) *Type B*, b=4, N=10 (**12**) (f) *Type B*, b=5, N=25 (**29**)

(g) *Type C*, b=3, N=13 (**13**) (h) *Type C*, b=4, N=11 (**11**) (i) *Type C*, b=5, N=15 (**15**)

Figure 6: Data and time complexity

From Fig. 6, we can see that

(1) The number of cache traces required in practice, N, is consistent with the estimated value of E_N. For *Types A* and *B*, sometimes N is smaller than E_N. This is because the calculation of E_N only considers the leakages in the first two rounds or in the last round, while in real attacks, the cache events in the deeper rounds can also contribute to the attacks.

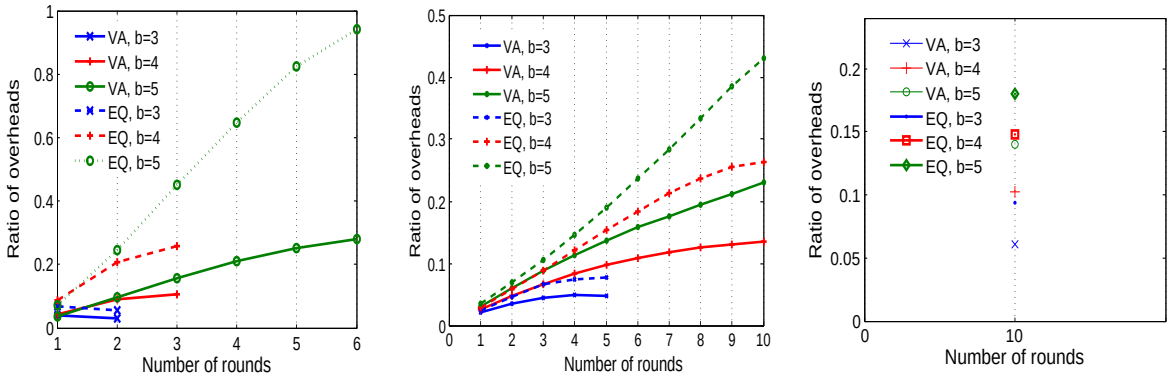
(2) The equation solving time seems to follow an exponential distribution. Most instances can be solved in a short time (less than 100 seconds). Only a few require the longer time (more than 200 seconds as shown at the right end of each subfigure). Similar observations are also reported in [14, 15].

(3) The time complexity of the full attack on AES is affordable. Most attacks can succeed within 7200 seconds. The time required in attacking AES for *Type A* and *Type C* is less than *Type B*.

Note that we set 200 seconds as the threshold of the equation solving time for a successful MDATDCA. If the adversary has more computation power, the attack may require fewer cache traces. For example, d=4 for *Type A*, but if we set the threshold to 3600 seconds, only 5 cache traces are required [18]. How to find a good tradeoff between the data complexity (the number of cache traces) and the time complexity (the solving time) is a very interesting problem.

8.2. Overhead for the equation system

The original AES with r rounds can be represented with a set of equations. Suppose the number of equations and variables to represent this set are N_e^r and N_v^r respectively. For the lookup q_t, the overhead introduced can be calculated as in Section 5.1 and 5.2. The number of variables and ANF equations in the r-th round (denoted as M_e^r and M_v^r) is the sum of the number of equations and variables for each lookup in the round. The ratio of $\frac{M_e^r}{N_e^r}$ and $\frac{M_v^r}{N_v^r}$ are denoted as EQ_r and VA_r respectively. Fig. 7 shows how EQ_r and VA_r change with r.



(a) *Type A* (b) *Type B* (c) *Type C*

Figure 7: Overheads introduced in MDATDCAs

We can observe that:(1) Both EQ_r and VA_r are small (less than 1); (2) Both EQ_r and VA_r increase linearly with r (as the size of the deduction set, s_p or s_n, for q_t increases); (3) EQ_r is a little larger than VA_r for the same r.

8.3. Comparisons with previous work

The comparisons of MDATDCAs with previous work are listed in Table 3. The first three columns describe the AES implementations. The next three columns list the attacks, and the number of traces and rounds that are required. The last column lists the reduced key search space. We can see that MDATDCAs have better performances than all previous work in terms of both data and time complexity.

Table 3: Comparisons of error-free MDATDCAs on AES with previous work

Type	Scenario	b	Attacks	Attack rounds	Cache traces	Key space
Type A	known plaintext	3	MDATDCA	2	35	1
Type A	chosen plaintext	4	[8]	1	14.5	2^{68}
Type A	known plaintext	4	[9]	1.125	30	2^{30}
Type A	known plaintext	4	[10]	1.25	30	10
Type A	known plaintext	4	MDATDCA	2-3	6	1
Type A	known plaintext	5	MDATDCA	2-6	7	1
Type B	known plaintext	3	MDATDCA	2-5	6	1
Type B	known plaintext	4	[4]	1.25	40	$2^{33.50}$
Type B	known plaintext	4	MDATDCA	2-10	10	1
Type B	known plaintext	5	[4]	1.25	55	$2^{34.26}$
Type B	known plaintext	5	MDATDCA	2-10	25	1
Type C	known ciphertext	3	[7]	1	10	2^{30}
Type C	known ciphertext	3	MDATDCA	1	13	1
Type C	known ciphertext	3	[4]	1	20	$2^{24.22}$
Type C	known ciphertext	4	[7]	1	14	2^{30}
Type C	known ciphertext	4	MDATDCA	1	11	1
Type C	known ciphertext	5	[4]	1	20	$2^{33.97}$
Type C	known ciphertext	5	[7]	1	25	2^{30}
Type C	known ciphertext	5	MDATDCA	1	15	1

9. Case 2: Error-tolerant MDATDCAs on AES

In this section, we will describe how to conduct MDATDCA on AES in practice, where there are errors when deducing cache events.

Similar to [10], we implemented unprotected AES software implementations on a 32-bit ARM microprocessor NXP LPC2124 and profiled the cache collisions via EM probe. We reset the cache to clear the AES data prior to each encryption. The acquisition was performed with Langer RF-B 3-2 probe, Langer PA303N 30 dB preamplifier and Tektronix DPO 4104 oscilloscope.

In the attack, for most of the time, a cache miss related EM trace is likely to have a distinct peak compared to a cache hit, which is also illustrated in Fig.1. However, for some table lookups, it is hard to tell whether they are cache miss or hit because the peak is not high enough. We consider such cache events as uncertain ones or errors. Under this scenario, the key issue is to deal with the uncertain cache events. Next, we describe the error-tolerant strategy and present the experimental results on AES.

9.1. Error tolerance strategy

In the attack, we set two thresholds of the amplitude peak value to deduce the cache events, the upper bound threshold V_M and the lower bound threshold V_H. Suppose V_t is the amplitude peak value of q_t in the trace. If $V_t < V_H$, the targeted cache event q_t is considered as a hit; if $V_t > V_M$, q_t is considered as a miss; if $V_H \leqslant V_t \leqslant V_M$, q_t is considered as a uncertain event. We adopt the following strategy to analyze each cache event.

1. q_t is a hit.

Then D, the possible deduction set of d ($\langle y_t \rangle_b$), is composed of the index set related to both previous cache miss events and uncertain cache events. Thus, the set size s_p is much larger than the one in error-free MDATDCA. Note that as some uncertain cache events might be cache hit in reality, there might exist two or more deductions which are both equal to d. Thus, we need to update Eq.(6) of Section 5 to

$$c_1 \vee c_2 \vee \ldots \vee c_{s_p} = 1 \tag{15}$$

2. q_t is a miss.

Then the impossible deduction set of $d(\langle y_t \rangle_b)$ is only composed of the index set related to previous cache miss events. Note that as some cache miss events in practice may be considered as uncertain cache events, the set size s_n is much smaller than the one in error-free MDATDCA.

3. q_t is an uncertain cache event.

Then no analysis is performed on this cache event.

9.2. Experimental results and comparisons

In the error tolerant MDATDCA experiments, we denote the error rate as $e = \frac{N_E}{N_A}$, where N_E is the number of uncertain cache events, and N_A is the number of all the utilized cache events.

For simplicity, we only perform the MDATDCA on AES for *Type A* when $b = 4$. The extensions to other cases are straightforward. To investigate the number of cache traces required for a successful MDATDCA for different e, we first conduct several attacks with different e and repeat the attacks for 100 random keys in every case. The results are plotted in Fig. 8.

Fig. 8 shows the number of traces required for MDATDCA changes with e. It is clear to see that the number of required cache traces increases linearly with the error rate. In practice, the error rate is about 40%. Only 12 cache traces are required to break AES. The online complexity of data acquisition is comparable to DPA, CPA and other types of cache attacks. The offline complexity is also affordable. Recovering the full key from a set of cache traces takes less than an hour on a computer mentioned in Section 7.4.

Table 6 lists the comparisons of MDATDCA to [10] which is the only literature about TDCA on AES with error tolerance. We can see that, our error-tolerant MDATDCA can analyze the cache events of the

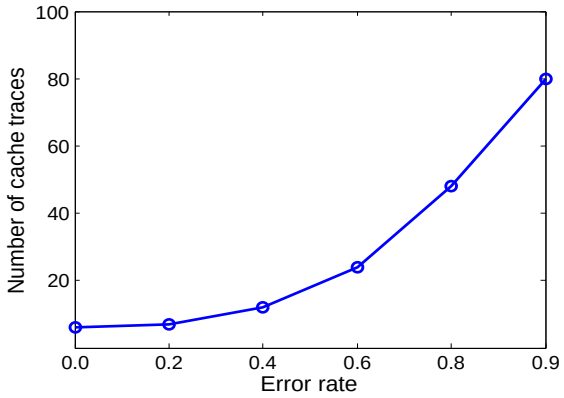


Figure 8: Number of required cache traces for a full key recovery of error-tolerant MDATDCA on AES

first three rounds and require less cache traces than [10]. Moreover, when the error rate is as high as 90%, MDATDCA still works with 80 cache traces, which is better than 80% in [10].

Table 4: Comparisons of error-tolerant MDATDCAs on AES with previous work

Attacks	Attack rounds	Error rate	Number of cache traces	Key search space
[10]	1.25	0.0	30	10
MDATDCA	3	0.0	6	1
[10]	1.25	0.2	≥ 30	10
MDATDCA	3	0.2	7	1
[10]	1.25	0.4	≥ 30	10
MDATDCA	3	0.4	12	1
[10]	1.25	0.6	≥ 30	10
MDATDCA	3	0.6	24	1
[10]	1.25	0.8	158	10
MDATDCA	3	0.8	48	1
MDATDCA	3	0.9	80	1

10. Case 3: MDATDCAs on AES with Preloaded Cache

The MDATDCAs in Section 8 and 9 are all conducted assuming the cache is cleaned before the attack. In practice, the cache might be partially filled with some lines of the lookup table, which is also named as TDCA in the partially preloaded cache scenario and widely studied in previous work [7, 9, 10]. This section presents the cache analysis strategy and experimental results of MDATDCAs on AES with partially preloaded cache.

10.1. Cache analysis strategy

Under this scenario, since some data of AES lookup table are already filled in the cache, more cache hit events can be observed for a single cache trace in practice. Then, the cache hits that occur may correspond to preloaded lines, and no valuable information can be provided to the attack. We utilized the cache miss events in our MDATDCA on AES.

10.2. Experimental results and comparisons

The comparisons of our results with previous work are depicted in Table 5. We can see that, under partially preloaded cache scenario, less cache traces are required to break AES by MDATDCA than [10]. Even when ten of sixteen cache lines are preloaded into cache before the AES encryption, MDATDCA can still succeed within 120 cache traces, which is better than eight preloaded cache lines reported in [10].

Table 5: Comparisons of partially preloaded MDATDCA on AES with previous work

Attacks	Preloaded lines	Traces
[10]	0	61
MDATDCA	0	10
MDATDCA	2	14
[10]	4	119
MDATDCA	4	24
MDATDCA	6	40
[10]	8	296
MDATDCA	8	80
MDATDCA	10	120

11. Extensions of MDATDCAs to AES-192/256

11.1. Different difficulties in TDCAs on AES-128/192/256

All previous TDCA work targets AES-128 and can at most analyze 16 lookups in the first round and first 4 lookups in the second round. As far as we know, there is no published work of TDCA on AES-192/256 and there exists more difficulties in TDCAs on AES-192/256 than on AES-128.

Let P denote the plaintext, K^0, K^1, K^2 be the round key of the first three rounds, and X^1,X^2 be the output of the first two rounds ($f(\cdot)$ be the round function). The key leakages in TDCA on AES-128 are depicted in Fig.9. We can see that, $16 \times b$ bits of K^0 are associated with the 16 leaked indexes of the first round, $16 \times b$ bits of K^1 and 128 bits of K^0 (when computing X_1) are involved in the 16 leaked indexes of the second round.

Take TDCA on AES of *Type A* when $b = 4$ as an example. As only the XORed results of the high 4 bits of these indexes are leaked, the maximal number of key bits of K^0 recovered from the 16 lookups in the first round is 60. Due to the avalanche effect of the S-Box function, 128 bits of K^0 and 64 bits of K^1 can also be leaked. According to the key expansion algorithm of AES-128, K^1 can be represented by the K^0 directly and also can be used to recover K^0. The work in [10] only used the first 4 table lookups, in which 128 bits of K^0 and 16-bit of K^1 can be leaked.

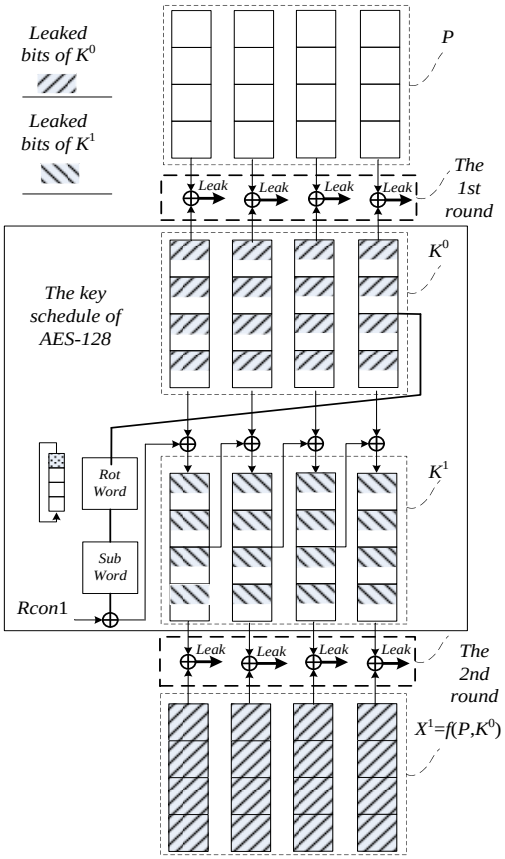


Figure 9: Leakages in TDCA on AES-128

However, such preponderance does not exist when attacking AES-192 and AES-256, in which the key expansion algorithm is much more complicated and the second round key has little (e.g., AES-192) or no relation (e.g., AES-256) with the first round key. How to conduct TDCA becomes an open problem when facing the difficulty of analyzing the cache leakages in more rounds. Next, we show that why and how MDATDCA can be used to attack AES-192 and AES-256.

11.2. MDATDCA on AES-192

The key leakages in TDCA on AES-192 are depicted in Fig.10. The right 64 bits of K^1 and 128 bits of K^2 are computed by 128 bits of K^0 and the left 64 bits of K^1, which is exactly the master key.

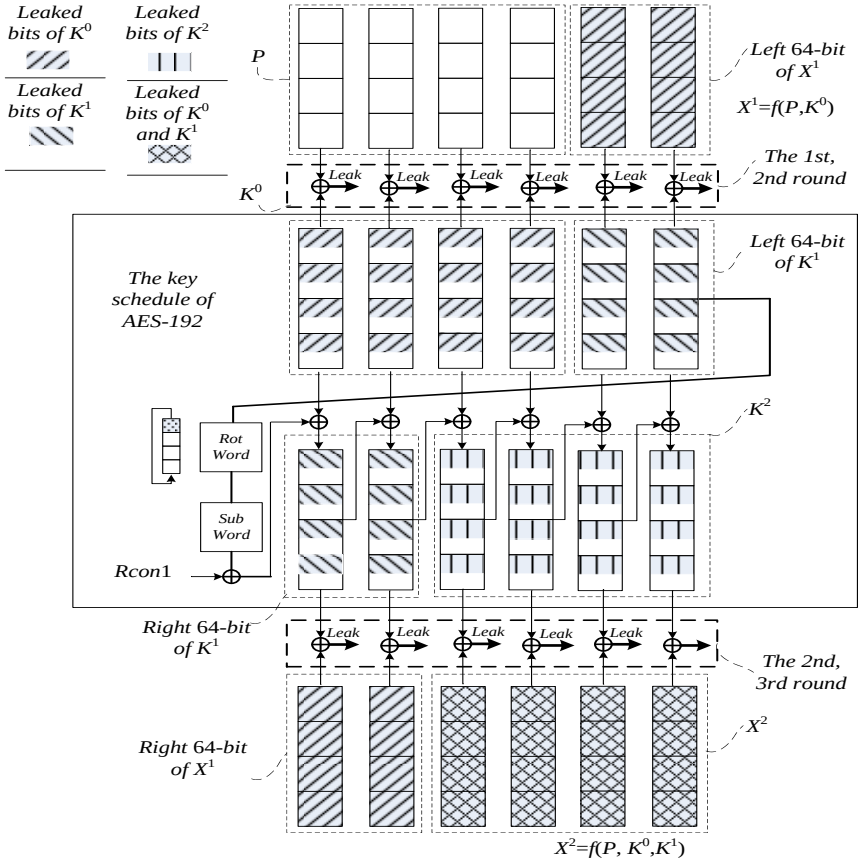


Figure 10: Leakages in TDCA on AES-192

Take TDCA on AES of *Type A* when $b = 4$ as an example. With the most efficient TDCA technique in [10], at most 60 bits of K^0 can be recovered if the first round is analyzed, 128 bits of K^0 and 16 bits of K^1 can be recovered if the first 4 lookups of the second round are analyzed. In total 144 key bits can be retrieved , which reduce the search space of the master key to 2^{48}.

In MDATDCA on AES-192, 60 bits of K^0 can be recovered if analyzing the first round, 128 bits of K^0 and 48 bits of K^1 (the lower 4-bits of the first column in K^1 cannot be leaked) can be recovered if analyzing the second round, 128 bits of K^0 and 64 bits of K^1 can be recovered in analyzing the third round. We can see that, in order to recover the full 192 bits of the master key, three rounds of cache leakages have to be analyzed, which can be done with MDATDCA. We show that 10 cache traces can recover AES key successfully within minutes on average under known plaintext and error-free scenario for the full attack.

11.3. MDATDCA on AES-256

The key leakages in TDCA on AES-256 are depicted in Fig.11. We can see that, K^1 is independent of K^0, K^2 is computed by 128 bits of K^0 and the right 32 bits of K^1.

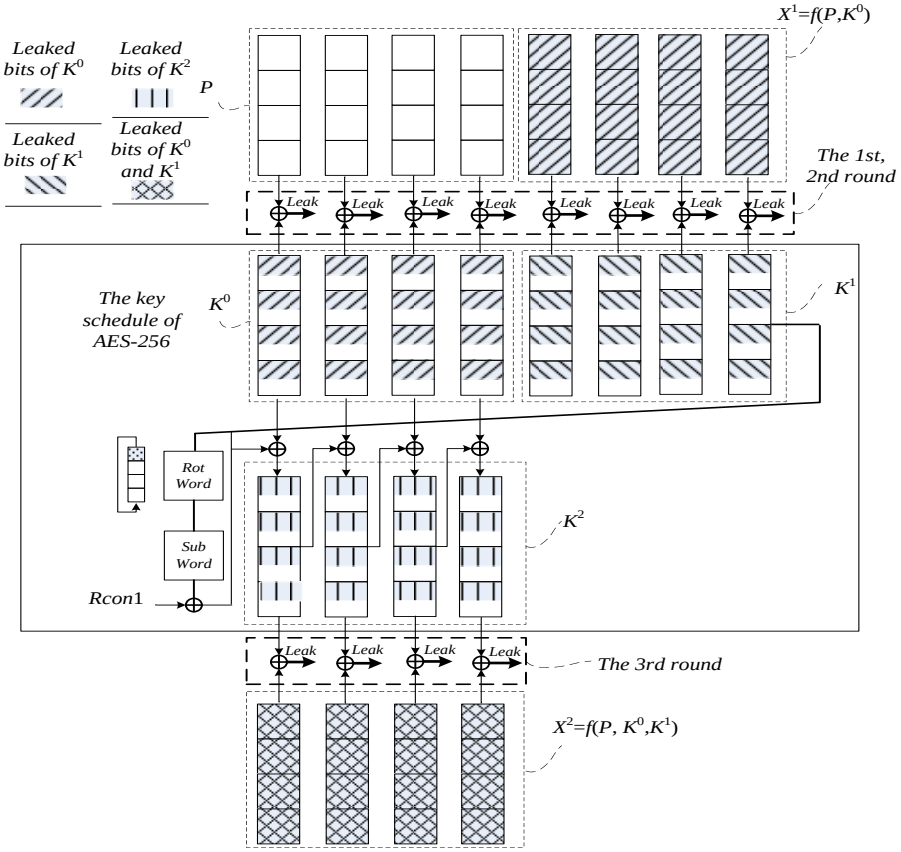


Figure 11: Leakages in TDCA on AES-256

Take TDCA on AES of *Type A* when $b = 4$ as an example. With the TDCA technique in [10], at most 60 bits of K^0 can be recovered if the first round is analyzed. 128 bits of K^0 and 16 bits of K^1 can be recovered if the first 4 lookups of the second round are analyzed. In total 144 key bits can be retrieved and reduce the search space of the master key to 2^{112}.

In MDATDCA on AES-256, 60 bits of K^0 can be recovered if the first round is analyzed, 128 bits of K^0 and 64 bits of K^1 can be recovered if the second round is analyzed, 128 bits of K^0 and 128 bits of K^1 can be recovered if the third round is analyzed. According to the key schedule of AES-256, the master key is just the concatenation of K^0 and K^1. To break AES-256, analyzing at least the cache events of the first 3 rounds has to be considered and MDATDCA works well for this. We show that 15 cache traces can recover the AES key within 30 minutes on average under known plaintext and error-free scenario for the full attack.

12. Conclusions and Future Work

This paper gives a comprehensive study on MDATDCA, the MDASCA-based trace driven cache attacks on AES under different AES implementations, attack scenarios and key lengths. We show that MDATDCA can exploit the cache hit/miss leakages in more rounds than the traditional TDCAs. Thus, the data and time complexity in both the online and offline phases can be significantly reduced. For the first time, we show that TDCAs on AES-192 and AES-256 become possible with the MDATDCA technique. We have achieved many improvements of TDCA on AES compared to previous work. Combining algebraic cryptanalysis with cache attacks is efficient for fully utilizing the leakages and improving cache attacks.

We stress that MDATDCAs are resistant to Boolean masking of software AES implementations in the case where all S-Boxes share the same random mask, as detailed in [3]. When such a masking scheme is used, our attacks will outperform higher order DPAs or CPAs that typically require thousands of traces. The countermeasures of MDATDCA is the same as TDCA, which are widely discussed in the previous works [4, 5, 6, 7, 8, 9, 10, 11]. They include pre-fetching the lookup table into the cache prior to encryption and shuffling the order of table lookup computations.

Note that MDATDCA can also be extended to improve TDCAs on other block ciphers, such as Camellia [24] and CLEFIA [25, 26, 27, 28]. The study of the trade-off between the data and time complexity in online and offline phases of MDATDCA, how to further quantized evaluating MDATDCA in the contributions of the leaked key bits from cache events to the recovery of the maser key of AES, how to evaluate MDATDCA on AES in case of error-tolerant and pre-loaded cache attack scenarios, how to develop new attack techniques to solve the TDCA problem might also be interesting problems in the future. We hope this paper can bring the understanding of both ASCA and TDCA to a new level, and help to evaluate the physical security of block cipher implementations.

Acknowledgments.

The authors would like to thank the anonymous reviewers of CHES 2012 for their helpful comments, Siwei Sun, Ruilin Li for fruitful discussions and their great help in improving the quality of the paper.

References

References

[1] D.A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: The case of AES. In the proceedings of CT-RSA 2006, LNCS, vol. 3860, pp. 1-20, 2006.

[2] D. J. Bernstein. Cache-timing attacks on AES. Available at `http://cr.yp.to/papers.html#cachetiming`, 2004.

[3] J. Bonneau, I. Mironov. Cache-collision timing attacks against AES. In the proceedings of CHES 2006, LNCS,vol 4249, pp. 201-215, 2006.

[4] O. Aciiçmez, Ç. Koç, Trace-Driven Cache Attacks on AES. Cryptology ePrint Archive, available at `http://eprint.iacr.org/2006/138.pdf`, 2006

[5] O. Acıiçmez, Ç. Koç, Trace driven cache attack on AES. In the proceedings of ICICS 2006, LNCS, vol 4296, pp. 112-121, 2006.

[6] G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, G. Palermo. AES Power Attack Based on Induced Cache Miss and Countermeasure. In the proceedings of ITCC 2005, IEEE Computer Society, pp. 586-591, 2005.

[7] J. Bonneau. Robust Final-Round Cache-Trace Attacks Against AES. Cryptology ePrint Archive, available at `http://eprint.iacr.org/2006/374.pdf`, 2006.

[8] J. Fournier and M. Tunstall. Cache based power analysis attacks on AES. In the proceedings of ACISP 2006, LNCS, vol 4058, pp. 17-28, 2006.

[9] J.-F. Gallais, I. Kizhvatov, and M. Tunstall. Improved Trace-Driven Cache-Collision Attacks against Embedded AES Implementations. In the proceedings of WISA 2010, LNCS 6513, pp. 243-257, 2011.

[10] J.-F. Gallais, and I. Kizhvatov. Error-Tolerance in Trace-Driven Cache Collision Attacks. In the proceedings of COSADE 2011, pp. 222-232, 2011.

[11] C. Lauradoux. Collision Attacks on Processors with Cache and Countermeasures. In the proceedings of WEWoRC 2005. LNI, vol. 74, pp. 76-85,2005.

[12] P.C. Kocher, J. Jaffe, B. Jun. Differential power analysis. In the proceedings of CRYPTO 1999, LNCS, vol. 1666, pp. 388-397, 1999.

[13] E. Brier, C. Clavier, F. Olivier. Correlation power analysis with a leakage model. In the proceedings of CHES 2004, LNCS, vol. 3156, pp. 16-29, 2004.

[14] M. Renauld, F.-X. Standaert. Algebraic Side-Channel Attacks. In the proceedings of INSCRYPT 2009, LNCS, vol 6151, pp. 393-410, 2009. Also available at `http://eprint.iacr.org/2009/279`, 2009.

[15] M. Renauld, F. Standaert, N. Veyrat-Charvillon. Algebraic side-channel attacks on the AES: Why time also matters in DPA. In the proceedings of CHES 2009, LNCS, vol 5747, pp. 97-111, 2009.

[16] Y. Oren, M. Kirschbaum, T. Popp, et al. Algebraic Side-Channel Analysis in the Presence of Errors. In the proceedings of CHES 2010, LNCS, vol 6225, pp. 428-442, 2010.

[17] Y. Oren and A. Wool. Tolerant Algebraic Side-Channel Analysis of AES. Cryptology ePrint Archive, available at `http://eprint.iacr.org/2012/092.pdf`, 2012.

[18] X.J. Zhao, F. Zhang, S.Z. Guo, et al. MDASCA: An Enhanced Algebraic Side-Channel Attack for Error Tolerance and New Leakage Model Exploitation. In the proceedings of COSADE 2012, LNCS, vol. 7275, pp. 231-248, 2012.

[19] OpenSSL: The Open Source toolkit for SSL/TLS. `http://www.openssl.org/`, 2012.

[20] FIPS 197, Advanced Encryption Standard, Federal Information Processing Standard, NIST, U.S. Dept. of Commerce, November 26, 2001.

[21] J.-C. Faugère, Gröbner Bases. Applications in Cryptology. In the proceedings of FSE 2007 Invited Talk, available at: http://fse2007.uni.lu/slides/faugere.pdf.

[22] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT Solvers to Cryptographic Problems. In the proceedings of SAT, LNCS, vol 5584, pp. 244-257, 2009.

[23] L.R. Knudsen, C.V. Miolane. Counting equations in algebraic attacks on block ciphers. International Journal of Information Security , vol. 9, No. 2, pp. 127-135, 2010.

[24] R. Poddar, A. Datta, and C. Rebeiro. A Cache Trace Attack on CAMELLIA. In the proceedings of InfoSecHiComNet 2011, LNCS, vol. 7011, pp. 144-156, 2011.

[25] C. Rebeiro, D. Mukhopadhyay. Differential Cache Trace Attack Against CLEFIA. Cryptology ePrint Archive, available at `http://eprint.iacr.org/2010/012.pdf`, 2010.

[26] C. Rebeiro, D. Mukhopadhyay. Cryptanalysis of CLEFIA Using Differential Methods with Cache Trace Patterns. In the proceedings of CT-RSA 2011. LNCS, vol. 6558, pp. 89-103, 2011.

[27] C. Rebeiro, R. Poddar, A. Datta, and D. Mukhopadhyay. An Enhanced Differential Cache Attack on CLEFIA for Large Cache Lines. In the proceedings of INDOCRYPT 2011, LNCS, vol. 7107, pp. 58-75, 2011.

[28] X.J. Zhao, T Wang. Improved Cache Trace Attack on AES and CLEFIA by Considering Cache Miss and S-box Misalignment. Cryptology ePrint Archive, available at `http://eprint.iacr.org/2010/056.pdf`, 2010.

Appendix 1: Building the AES Equation Set

We adopt the technique in [23] to derive every S-Box output bit (denoted as $y_1y_2\dots y_8$) with high-degree equations (degree 7) from the 8 S-Box input bits (denoted as $x_1x_2\dots x_8$), which provides an explicit representation of the dependence of the S-Box output on the input. An example of y_1 represented by $x_1x_2\dots x_8$ is shown as below.

$$y_1 = x_1 \oplus x_3 \oplus x_4 \oplus x_6 \oplus x_1x_8 \oplus x_1x_7 \oplus x_2x_6 \oplus x_4x_6 \oplus x_1x_3 \oplus x_6x_7 \oplus x_2x_4 \oplus x_2x_8 \oplus x_6x_8 \oplus x_3x_5 \oplus x_2x_3x_8 \oplus x_3x_5x_6 \oplus x_3x_7x_8 \oplus x_3x_4x_8 \oplus x_1x_4x_7 \oplus x_1x_5x_8 \oplus x_1x_2x_8 \oplus x_4x_7x_8 \oplus x_2x_3x_6 \oplus x_3x_5x_7 \oplus x_3x_6x_8 \oplus x_2x_7x_8 \oplus x_1x_2x_4 \oplus x_2x_6x_7 \oplus x_2x_5x_7 \oplus x_1x_2x_6 \oplus x_5x_6x_8 \oplus x_1x_3x_5 \oplus x_2x_4x_6 \oplus x_3x_4x_5 \oplus x_1x_6x_8 \oplus x_3x_5x_8 \oplus x_5x_6x_7 \oplus x_2x_3x_4 \oplus x_2x_5x_8 \oplus x_1x_3x_4 \oplus x_1x_2x_4x_7 \oplus x_1x_2x_5x_7 \oplus x_1x_2x_3x_6 \oplus x_1x_2x_3x_7 \oplus x_2x_3x_4x_5 \oplus x_1x_4x_5x_6 \oplus x_4x_5x_6x_7 \oplus x_5x_6x_7x_8 \oplus x_4x_6x_7x_8 \oplus x_4x_5x_7x_8 \oplus x_3x_6x_7x_8 \oplus x_1x_2x_3x_4 \oplus x_1x_5x_6x_8 \oplus x_2x_3x_5x_8 \oplus x_1x_3x_4x_7 \oplus x_3x_5x_6x_7 \oplus x_1x_5x_6x_7 \oplus x_3x_4x_6x_7 \oplus x_2x_4x_5x_8 \oplus x_1x_4x_7x_8 \oplus x_1x_3x_5x_8 \oplus x_1x_2x_5x_8 \oplus x_1x_4x_5x_8 \oplus x_1x_4x_6x_8 \oplus x_2x_4x_6x_8 \oplus x_1x_2x_4x_6 \oplus x_1x_6x_7x_8 \oplus x_1x_4x_5x_7 \oplus x_1x_2x_4x_8 \oplus x_2x_5x_7x_8 \oplus x_3x_5x_6x_8 \oplus x_2x_5x_6x_8 \oplus x_2x_4x_6x_7 \oplus x_1x_2x_3x_5x_7 \oplus x_1x_2x_3x_4x_7 \oplus x_1x_2x_3x_7x_8 \oplus x_1x_2x_3x_4x_5 \oplus x_1x_2x_3x_6x_7 \oplus x_2x_3x_4x_6x_7 \oplus x_1x_2x_4x_6x_7 \oplus x_1x_2x_4x_5x_8 \oplus x_2x_3x_5x_6x_7 \oplus x_1x_2x_3x_5x_8 \oplus x_4x_5x_6x_7x_8 \oplus x_2x_3x_4x_5x_6 \oplus x_2x_3x_4x_7x_8 \oplus x_1x_3x_4x_7x_8 \oplus x_3x_4x_5x_7x_8 \oplus x_1x_4x_5x_7x_8 \oplus x_2x_3x_4x_5x_8 \oplus x_2x_4x_5x_6x_8 \oplus x_1x_3x_6x_7x_8 \oplus x_3x_4x_5x_6x_8 \oplus x_1x_2x_6x_7x_8 \oplus x_1x_4x_5x_6x_8 \oplus x_2x_4x_6x_7x_8 \oplus x_2x_5x_6x_7x_8 \oplus x_1x_2x_4x_5x_6x_8 \oplus x_1x_2x_3x_4x_5x_7 \oplus x_1x_2x_5x_6x_7x_8 \oplus x_1x_2x_4x_5x_7x_8 \oplus x_1x_3x_4x_5x_7x_8 \oplus x_1x_2x_3x_5x_7x_8 \oplus x_1x_2x_3x_5x_6x_8 \oplus x_2x_3x_4x_5x_6x_8 \oplus x_1x_2x_3x_4x_6x_8 \oplus x_1x_3x_4x_5x_6x_8 \oplus x_1x_3x_4x_6x_7x_8 \oplus x_1x_2x_3x_4x_5x_7x_8 \oplus x_1x_2x_3x_4x_5x_6x_8$$

Each S-Box can be represented by 254 ANF equations with 262 variables. The number of the new variables and ANF equations introduced for different AES operations are listed in Table 6.

Table 6: The number of the new variables and ANF equations that introduced

Operation	Variables	ANF Equations
AddRoundKey	256 (384) [1]	128
SubBytes	262 × 16=4192	254× 16=4064
ShiftRows	128	128
MixColumns	128	128
One encrypt Round	4704 (4832,4576)[2]	4448 (4320)[3]
One key expansion Round	1184 (1312)[4]	1144

[1] 384 for the first AddRoundKey, and 256 for the others.

[2] 4832 for the first round (including the 128-bit plaintext), 4704 for the 2-nd to the 9-th round, 4576 for the last round.

[3] 4448 for the first 9 rounds and 4320 for the final round.

[4] 1312 for the first round and 1184 for the others.