A comprehensive study of multiple deductions-based algebraic trace driven cache attacks on AES

Xinjie Zhao\textsuperscript{a,b}, Shize Guo\textsuperscript{b}, Fan Zhang\textsuperscript{c,*}, Tao Wang\textsuperscript{a}, Zhijie Shi\textsuperscript{c}, Zhe Liu\textsuperscript{d}, Jean-François Gallais\textsuperscript{d}

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

ARTICLE INFO
Article history:
Received 13 June 2012
Received in revised form
15 May 2013
Accepted 2 July 2013

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

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 TDCAs 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.

© 2013 Elsevier Ltd. All rights reserved.

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 a cache hit. Cache attacks demonstrated fall into three categories, depending on the channels used to collect the leakages. These channels are spy
processes (Osvik et al., 2006), timing information (Bernstein, 2004; Bonneau and Mironov, 2006) and power/electromagnetic (EM) traces (Acıı¨ c¸mez and Koç, 2006a, 2006b; Bertoni et al., 2005; Bonneau, 2006; Fournier and Tunstall, 2006; Gallais et al., 2011; Gallais and Kizhvatov, 2011; Lauradoux, 2005). 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, TDCA monitors cache hit and miss misses from power/EM traces, and recover the secret key used in the computation. The number of traces required in TDCA is much less than in the conventional differential power attacks (DPAs) (Kocher et al., 1999), correlation power attacks (CPAs) (Brier et al., 2004) or other types of cache attacks (Osvik et al., 2006; Bernstein, 2004; Bonneau and Mironov, 2006). Considering AES for example, only 30 cache traces are required in TDCA (Gallais et al., 2011; Gallais and Kizhvatov, 2011) instead of hundreds (or thousands) of power traces in DPAs, CPAs (Brier et al., 2004), hundreds of cache traces in access driven cache attacks (Osvik et al., 2006), and millions of cache traces in timing driven cache attacks (Bernstein, 2004; Bonneau and Mironov, 2006).

AES was targeted in many TDCA’s (Acıı¨ c¸mez and Koç, 2006a, 2006b; Bertoni et al., 2005; Bonneau, 2006; Fournier and Tunstall, 2006; Gallais et al., 2011; Gallais and Kizhvatov, 2011; Lauradoux, 2005). Throughout this paper, AES refers to AES-128 by default. Bertoni et al. (2005) 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 implemented with a table of 256 bytes were estimated from power simulations and analyzed in Bertoni et al. (2005).

Further research in TDCA on AES splits into two directions. One is about exploiting new and real leakages in TDCA’s, where cache traces were collected from real power consumptions in Fournier and Tunstall (2006) and Gallais et al. (2011) and from EM in Gallais and Kizhvatov (2011). The other is improving the efficiency of TDCA’s on different AES implementations. In the attacks on AES with large lookup tables (e.g., 1 K bytes), TDCA can exploit cache events in the first round (Lauradoux, 2005), the first two rounds (Acıı¨ c¸mez and Koç, 2006a) or the last round (Acıı¨ c¸mez and Koç, 2006b; Bonneau, 2006). For AES implemented with a compact table (256 bytes), TDCA can exploit the cache events in the first round (Bertoni et al., 2005), or the first two rounds (Fournier and Tunstall, 2006; Gallais et al., 2011; Gallais and Kizhvatov, 2011). This paper is under the latter direction and tries to improve TDCA on AES.

In the aforementioned TDCA’s 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 TDCA’s with algebraic techniques and conducting algebraic side-channel attacks (ASCA) (Renauld and Standaert, 2009; Renauld et al., 2009; Oren et al., 2010; Oren and Wool, 2012) is a very promising way to improve TDCA. Previous ASCA mainly focused on power based Hamming weight leakage model (Renauld and Standaert, 2009; Renauld et al., 2009; Oren and Wool, 2012) or Hamming distance leakage model (Oren et al., 2010). The original ASCA (Renauld and Standaert, 2009; Renauld et al., 2009) can only work when the deduction on the targeted states is single and correct. The error tolerant ASCA in Oren et al. (2010) and Oren and Wool (2012) 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.

In COSADE 2012, Zhao et al. proposed the multiple deductions-based ASCA (MDASCA) (Zhao et al., 2012). The work in Zhao et al. (2012) 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 ASCA’s (Renauld and Standaert, 2009; Renauld et al., 2009; Oren et al., 2010; Oren and Wool, 2012). They showed that MDASCA can utilize the leakages in the first three rounds of AES in TDCA’s. Under error-free attack scenario, 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 1 KB, 2 KB tables in OpenSSL cryptography library (OpenSSL, 2012)), different attack scenarios (e.g., error-tolerant scenario (Gallais and Kizhvatov, 2011), partial preloaded cache scenario (Bonneau, 2006; Gallais et al., 2011; Gallais and Kizhvatov, 2011), different AES key lengths (e.g., AES-192/256), how many rounds of leakages can MDASCA exploit, how much can MDASCA improve TDCA’s 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 TDCA’s (MDATDCA’s). 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 (Acıı¨ c¸mez and Koç, 2006a, 2006b; Bertoni et al., 2005; Bonneau, 2006; Fournier and Tunstall, 2006; Gallais et al., 2011; Gallais and Kizhvatov, 2011). Then we categorize previous TDCA’s 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

---

1 S-Box is usually implemented with lookup table. In this paper, S-Box lookup is sometimes written as table lookup.

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.
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. 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, \( i \) 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 \( U_j \in P \), \( V_j \in K \). Let \( f(\cdot) \) be the function that computes \( y_j \) from \( U_j \) and \( V_j \), \( y_j = f(U_j, V_j) \). Suppose each entry in the table has \( 2^b \) bytes and each cache line has \( 2^d \) bytes. Let \( \{y_j\}_b \) be the \( b \) most significant bits (MSBs) of \( y_j \) leaked in \( q_j \), \( b = 1 - (d - e) \). Assume \( q_j \) is the \( t \)-th targeted cache events in TDCA and \( y_j \) 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 \( S_k^t \) be the set of the \( b \) MSBs of the indexes \( y_{M_1}, \ldots, y_{M_n} \), i.e., \( S_k^t = \{\{y_{M_1}\}_b, \{y_{M_2}\}_b, \ldots, \{y_{M_n}\}_b\} \).

3. The TDCA problem

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

TDCA on a block cipher is illustrated in Fig. 1, where \( \lambda \) 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 number of clock cycles is distinctly different). The goal of TDCA is to extract the value of all \( k \) in \( K \) (the secret key) from the knowledge of the \( p_s \) (known public variables) and \( q_s \) (cache events).

Suppose the cache contains no data from the table before each encryption. As to the analysis of the cache event in \( q_0 \), suppose \( q_1 \) is the only cache miss before \( q_0 \). A cache hit of \( q_1 \) means both \( y_1 \) and \( y_j \) access the same cache line. Eq. (1) holds if \( q_1 \) is a cache hit.

\[
\langle y_1 \rangle_b = \langle y_j \rangle_b \Rightarrow \langle f(U_1, V_1) \rangle_b = \langle f(U_j, V_j) \rangle_b
\] (1)

The key technique of TDCA is to use Eq. (1) to reduce the search space of \( U_j \cup V_j \), which converges to \( K \) if \( t \) is large enough. Since both \( U_j \) and \( V_j \) are known, the adversary can check all the assignments to those \( k_i \) in \( V_j \cup U_j \) 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 \( q_i \) is a cache miss, Eq. (2) can be used in the key search.

\[
\langle y_i \rangle_b \neq \langle y_j \rangle_b \Rightarrow \langle f(U_i, V_i) \rangle_b \neq \langle f(U_j, V_j) \rangle_b
\] (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 \( q_i \) is a hit, only one of \( q_{M_1}, \ldots, q_{M_n} \) accesses the same cache line as \( q_i \) (because if two of them are in the same cache line as \( q_i \), one of them must be a hit). In this case, Eq. (3) holds.

\[
\exists j \in O_M^t \setminus \{i\} : \langle f(U_i, V_i) \rangle_b = \langle f(U_j, V_j) \rangle_b
\] (3)

If \( q_i \) is a miss, Eq. (4) holds.

\[
\forall j \in O_M^t : \langle f(U_i, V_i) \rangle_b \neq \langle f(U_j, V_j) \rangle_b
\] (4)

Using the \( n \) equations (inequations) in (3) or (4), more assignments to key bits in \( V_{M_1} \cup \ldots \cup V_{M_n} \cup V_i \) can be verified. The key recovery is converted into the problem of how to converge \( V_{M_1} \cup \ldots \cup V_{M_n} \cup V_i \) 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 (Acııçmez and Koç, 2006a, 2006b; Bertoni et al., 2005; Bonneau, 2006; Fournier and Tunstall, 2006; Gallais et al., 2011; Gallais and Kizhvatov, 2011; Lauradoux, 2005; Peddar et al., 2011; Rebeiro and Mukhopadhyay, 2010, 2011; Rebeiro et al., 2011; Zhao and Wang, 2010). Different attack techniques can be developed to solve this problem, such as traditional TDCA technique (Acııçmez and Koç, 2006a, 2006b; Bertoni et al., 2005; Bonneau, 2006; Fournier and Tunstall, 2006; Gallais et al., 2011; Gallais and Kizhvatov, 2011; Lauradoux, 2005), MDASC technique (Zhao et al., 2012) 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) \( g_o \), the number of the lookup tables; (2) \( g_s \), the size of the lookup tables; (3) \( g_t \), the number of lookups in one round that access the same table; (4) \( g_c \), the size of the cache line, where \( g_c = 2^d \). 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 (Fournier and Tunstall, 2006).

**Type A.** A single table is used in all rounds, $g_i = 1$. Each round has 16 table lookups, $g_i = 16$. The size of the lookup table can be 256 B (256 bytes) or 2 KB (2 K bytes). The examples of this type include the implementation studied in Fournier and Tunstall (2006), Gallais et al. (2011), Gallais and Kizhvatov (2011) and FIPS 197 (2001) where $g_i = 256$ B, $g_i = 16$ B, and $b = 4$, and an implementation in OpenSSL v1.0.0d (OpenSSL, 2012), where $g_i = 2$ KB, $g_i = 64$ B, and $b = 5$. Recall $b$ is the number of bits revealed from one table lookup.

**Type B.** There are four 1 KB tables, $g_i = 4$. Each round has 16 table lookups, where each table is accessed four times, $g_i = 4$. For example, AES in OpenSSL v0.9.8j uses four 1 KB 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 1 KB table in the last round ($g_i = 1$), where it is accessed 16 times ($g_i = 16$). 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 TDCA.

### 4.2. TDCA on AES of Type A

Bertoni et al. performed the first TDCA on an implementation of Type A (Bertoni et al., 2005), 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 Fournier and Tunstall (2006), Gallais et al. (2011) and Gallais and Kizhvatov (2011) also utilized some cache events in the second round.

In Fournier and Tunstall (2006), equations are generated only from the cache hits as shown in Eq. (3). In the first round, $U_i = p_j$, $V_i = k_i$, and $U_i = p_j$, $V_i = k_i$, $(0 \leq j < t \leq 15)$, $f^t(\cdot)$ is the bitwise XOR and $(k_i \oplus k_j)$ can be derived. It is shown in Fournier and Tunstall (2006) that the search space of the AES key can be reduced to $2^{48b}$ with at most 240 adaptive chosen plaintexts. To improve the attack, the work in Fournier and Tunstall (2006) 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 $f^t(\cdot)$ becomes more complicated. Considering $q_{16}$ as an example ($t = 16$), suppose $n = 8$, and $O_{16} = \{m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\} = \{0, 2, 4, 5, 7, 8, 10, 14\}$. For each $q_t$, $(1 \leq i \leq 8)$, the key space of $V_{16} \cup V_{15}$ can be reduced via Eq. (5).

### 4.3. TDCA on AES of Type B

Aciícmez and Koç (2006a) 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 second round, only four groups of $(k_i \oplus k_j)$ ($j \equiv t \mod 4$) can be derived and the key search space can be reduced to $2^{128-4\times(4-1)x}$. More key bits can be derived via the analysis of the second round. In Aciícmez and Koç (2006a), $q_{16–q_{17}}$ are added. They showed that about 40 and 55 power traces were required to reduce the key search space to $2^{33.50}$ and $2^{44.26}$ for $b = 4$ and 5 respectively. Aciícmez and Koç (2006a) 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 Aciícmez and Koç (2006b) and Bonneau (2006) 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 $k_i \oplus k_j$, $(0 \leq j < t \leq 15)$ are leaked. While

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Type</th>
<th>$g_i$</th>
<th>$g_b$</th>
<th>$g_i$</th>
<th>$g_b$</th>
<th>TDCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard NIST implementation (FIPS, 2001)</td>
<td>Type A</td>
<td>1</td>
<td>256 B</td>
<td>16</td>
<td>16 B, 32 B, 64 B etc</td>
<td>Bertoni et al. (2005), Fournier and Tunstall (2006), Gallais et al. (2011) and Gallais and Kizhvatov (2011)</td>
</tr>
<tr>
<td>AES in OpenSSL v0.9.8a (OpenSSL, 2012)</td>
<td>Type B,C</td>
<td>5</td>
<td>1 KB</td>
<td>4,16</td>
<td>16 B, 32 B, 64 B etc</td>
<td></td>
</tr>
<tr>
<td>AES in OpenSSL v0.9.8j (OpenSSL, 2012)</td>
<td>Type B</td>
<td>4</td>
<td>1 KB</td>
<td>4</td>
<td>16 B, 32 B, 64 B etc</td>
<td></td>
</tr>
<tr>
<td>AES in OpenSSL v1.0.0d (OpenSSL, 2012)</td>
<td>Type A</td>
<td>1</td>
<td>2 KB</td>
<td>4</td>
<td>16 B, 32 B, 64 B etc</td>
<td></td>
</tr>
</tbody>
</table>

* a Type B for TDCA on the first nine rounds and C for TDCA on the 10-th round.
* b 4 for the first nine rounds, and 16 for the 10-th round.
in the last round, \( f(t) \) becomes a complicated nonlinear function. As a result, all the 8-bits of \( k_j^{10} \) and \( k_j^{10} \) \((j = \lfloor j \mod 16 \rfloor)\) can be derived where \( j \) is the \( j \)-th byte in the ciphertext. The work in Acıımez and Koç (2006b) and Bonneau (2006) 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 \( y_i \) in different rounds. As noted in Acıımez and Koç (2006a), the full avalanche effect is achieved in the third round, where \( f(U_i, V_i) \) becomes complicated if a manual analysis is conducted. Because of this, current traditional TDCAs on AES (Acıımez and Koç, 2006a; Bertoni et al., 2005; Bonneau, 2006; Fournier and Tunstall, 2006; Gallais et al., 2011; Gallais and Kizhvatov, 2011; Lauradoux, 2005) can at most analyze the cache events in the first two rounds, more precisely, from the first 20 lookups. Moreover, all current TDCAs work 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 TDCAs 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 TDCAs seems to be interesting and promising. The MDASCA proposed by Zhao et al. (2012) 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 TDCAs, 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 \( q_{M1}, \ldots, q_{Mn} \) before \( q_t \). As to the set of the \( b \) MSBs of the indexes, \( S_M = \{y_{M1}/b, y_{M2}/b, \ldots, y_{Mn}/b\} \), let \( d \) be the correct value (or deduction) of \( y_{Mj}/b \), \( d^j \) be the \( j \)-th bit of \( d \), \( d_j \) be the \( i \)-th element in \( S_M \), and \( d_j \) be the \( j \)-th bit of \( d \). If \( q_t \) is a cache hit, the cache line that includes the index \( y_i \) 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 \( S_M \). If \( q_t \) is a cache miss, the cache line that includes \( y_i \) has not been accessed, which means that \( d \) is not equal to any element of the multiple deductions in \( S_M \).

From above, we can see that how to represent the relations between \( y_i \) and its multiple possible or impossible deductions for cache miss event is the most challenging part in TDCAs. The work in Zhao et al. (2012) proposes a generic method to convert the multiple deductions into algebraic equations and applies it to TDCAs. For convenience, we will refer to the MDASCA-based trace driven cache attack or so called multiple deductions-based algebraic TDCAs as MDATDCAs. 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 (Faugère, 2007; Soos et al., 2009). More details about MDASCA can be found in Zhao et al. (2012).

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 \( D = S_M \). A one-bit variable \( c_i \) is introduced to represent whether \( d_i \) is equal to \( d \) or not. Another one-bit variable \( e_i \) is also introduced to represent whether \( d_i \) is equal to \( d' \). Then \( c_i \) can be represented with Eq. (6), where \( \neg \) denotes the NOT operation.

\[
e_i = \neg(d_i \oplus d') \quad c_i = \prod_{j=1}^{b} e'_j
\]

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

\[
c_1 \lor c_2 \lor \ldots \lor c_{s_p} = 1, \quad \neg c_i \lor \neg c_j = 1, \quad 1 \leq i < j \leq s_p
\]

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

5.2. Representing a cache miss

Let \( D \) denotes the impossible deduction set of \( \langle y_i \rangle \) and \( s_p \) be the size of \( D \). Then \( D = \overline{S_M} \). Then \( e'_i \) is also introduced as in Section 5.1. None of \( d_i \) in \( D \) is equal to \( d \), which can be represented as:

\[
e_i = \neg(d_i \oplus d') \quad c_i = \prod_{j=1}^{b} e'_j = 0
\]

According to Eq. (8), \((b + 1) \times s_p\) variables and \((b + 1) \times s_p\) ANF equations are introduced to represent \( D \). 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 (Soos et al., 2009), 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, \( D \) 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 \( D \), \( m = 2^k \). Let \( K = V_{M} \cup \overline{V_{M}} \cup V_{f} \).
E denotes the maximal number of rounds that can be utilized in MDATDCA. As long as $D$ 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 Acı́cmez and Koç (2006a, 2006b) and Zhao et al. (2012), 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 $(m - 1/m)^t$. Then, after $t$ lookups to the same table, the expected number of loaded cache lines, $n_t$, can be calculated as

$$n_t = m \left(1 - \frac{m - 1}{m}\right)^t$$  \hfill (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., $D$ is filled up). For Type C, all the 16 lookups in the last round can be used for key recovery.

6.2. The number of cache traces required

The work in Zhao et al. (2012) 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) $p_t$: the ratio between the size of the search space of $K_i$ after and before analyzing $q_t$. The probability that $q_t$ is a cache hit is $n_t/m$. If $q_t$ is a hit, the expected number for the candidates of $(y_i|X_i)$ can be reduced from $m$ to $n_t$. It is also easy to check the probability when $q_t$ is a miss. Then $p_t$ can be calculated as

$$p_t = \left(\frac{n_t}{m}\right)^2 + \left(1 - \frac{n_t}{m}\right)^2$$  \hfill (10)

Fig. 3 shows how $p_t$ changes with $t$ and $b$. We choose $p_t \leq 0.9$ 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) $\pi_t$: the number of key bits that can be derived from $q_t$

$$\pi_t = -\log_2(p_t)$$  \hfill (11)

Fig. 4 shows how $\pi_t$ changes with $t$ and $b$. We can see that $q_0$ 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 $p_t \geq 0.5$ for all the curves.

(3) $s_t$: 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 $s_1$ and $s_2$. For Type C, we calculate $s_9$ in the final round. Note that there are some intersects among $K_t$ for different table lookups in practice, thus $s_t$ satisfies

$$s_t \leq \sum_{z=2t}^{2t+15} \pi_z$$  \hfill (12)

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

Let $t_0$, $t_1$, and $t_9$ 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, $t_0 = 15b$ and $t_1 = 128$ for Type A, $t_0 = 12b$ and $t_1 = 128$ for Type B, and $t_0 = 128$ 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 $t_i$ bits in the $i$-th round, denoted as $N_i$. We calculate $N_0$ and $N_9$ as

$$N_i \geq \frac{t_i}{s_t}$$  \hfill (13)

As $t_0$ bits are recovered in the first round, we only need to recover the remaining $128 - t_0$ bits in the second round. So we roughly calculate $N_1$ as
Let $E_N$ denote the estimated minimal number of cache traces needed in order to recover the master key. $E_N = \max\{N_0, N_1\}$ for Type A and B. $E_N = N_9$ 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.

6.3. Comparisons with previous work

The comparisons of this paper with previous evaluations of TDCA on AES (Acıınc¸mez and Koç, 2006a, 2006b; Bertoni et al., 2005; Bonneau, 2006; Fournier and Tunstall, 2006; Gallais et al., 2011; Gallais and Kizhvatov, 2011; Lauradoux, 2005; Zhao et al., 2012) are listed as follows.

1. For the first time, we generalize the TDCA attacking strategy regardless of the different AES implementations. Our strategy can estimate the maximal number of rounds that can be exploited in the key recovery. Finally, we apply it to nine different AES implementations considering the cache line size and the table size.

2. Previous work is constrained within the first 20 cache events in the first round. We propose a new method regardless of the number of rounds. Our attack can actually utilize the cache events in all rounds of AES. Based on the information-theoretic approach, four metrics are introduced to estimate the minimal number of cache traces required for a successful MDATDCA on nine different AES implementations.

It should be noted that, we provide an approach to calculate the leaks from every round-key bits in all rounds. We use this approach to roughly estimate the minimal number of traces required in TDCA. Our estimation can be verified by the experiments (to be addressed in Sections 7 and 8). There could be some future work to quantify the joint cache leaks from many key bits in more rounds, and to estimate the lower bound more accurately.

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 Knudsen and Miolane (2010) 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.

7.2. Profile the cache traces

This paper mainly focuses on the analysis part of MDATDCA. Details of profiling the events can be found in Gallais et al. (2011), Gallais and Kizhvatov (2011) and Zhao et al. (2012). In Sections 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, \( D \) of 8/16/32 cache lines cannot be filled up within the first 2/3/6 rounds. For Type B, \( D \) of 8/16/32 cache lines cannot be filled up within the first 5, 10, 10 rounds. For Type C, \( D \) 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 Oren et al. (2010) and Oren and Wool (2012). 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

### Table 2 – \( N_i \) for different AES implementations.

<table>
<thead>
<tr>
<th>Type</th>
<th>%b</th>
<th>( N_0 )</th>
<th>( N_1 )</th>
<th>( N_9 )</th>
<th>( E_N )</th>
<th>%b</th>
<th>( N_0 )</th>
<th>( N_1 )</th>
<th>( N_9 )</th>
<th>( E_N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type A</td>
<td>3</td>
<td>4.25</td>
<td>34.69</td>
<td>35</td>
<td>4</td>
<td>5.06</td>
<td>6.88</td>
<td>7</td>
<td>5</td>
<td>8.72</td>
</tr>
<tr>
<td>Type B</td>
<td>3</td>
<td>4.89</td>
<td>5.93</td>
<td>6</td>
<td>4</td>
<td>11.77</td>
<td>6.49</td>
<td>12</td>
<td>5</td>
<td>28.43</td>
</tr>
<tr>
<td>Type C</td>
<td>3</td>
<td>12.10</td>
<td>13</td>
<td>4</td>
<td>10.79</td>
<td>11</td>
<td>5</td>
<td>14.88</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

![Fig. 5 – Cache hit and miss sequences for Types A, B and C.](image-url)
Fig. 6 – Data and time complexity. (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).

Fig. 7 – Overheads introduced in MDATDCAs. (a) Type A, (b) Type B, (c) Type C.
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 (Faugère, 2007), or SAT-based solver (Soos et al., 2009). We use a SAT-based solver, CryptoMiniSat 2.9.0 (Soos et al., 2009), on an AMD Athlon 64 Dual core 3600+ processor clocked at 2.0 GHz.

In Sections 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-128

In this section, we conduct MDATDCA on AES-128 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 (Acıı¨c¸mez and Koç, 2006a, 2006b; Bertoni et al., 2005; Bonneau, 2006; Fournier and Tunstall, 2006; Gallais et al., 2011; Gallais and Kizhvatov, 2011; Lauradoux, 2005).

#### 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\( b \) key bits. Fig. 6(a)–(i) show the distribution of the different solving times (in seconds) for the nine cases by analyzing \( N \) cache traces. The bold number is \( E_N \).

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 s). Only a few require the longer time (more than 200 s as shown at the right end of each subfigure). Similar observations are also reported in Renauld and Standaert (2009) and Renauld et al. (2009).

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

Note that we set 200 s 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 s, only 5 cache traces are required (Zhao et al., 2012). 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 \( N_r \) and \( N_v \), respectively. For the lookup \( q_t \), the overhead introduced can be calculated as in Sections 5.1 and 5.2. The number of variables and ANF equations in the

![Table 3 - Comparisons of error-free MDATDCAs on AES with previous work.](image)

<table>
<thead>
<tr>
<th>Type</th>
<th>Scenario</th>
<th>b</th>
<th>Attacks</th>
<th>Attack rounds</th>
<th>Cache traces</th>
<th>Key space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type A</td>
<td>Known plaintext</td>
<td>3</td>
<td>MDATDCA</td>
<td>2</td>
<td>35</td>
<td>1</td>
</tr>
<tr>
<td>Type A</td>
<td>Chosen plaintext</td>
<td>4</td>
<td>Fournier and Tunstall (2006)</td>
<td>1</td>
<td>14.5</td>
<td>2^48</td>
</tr>
<tr>
<td>Type A</td>
<td>Known plaintext</td>
<td>4</td>
<td>Gallais et al. (2011)</td>
<td>1.125</td>
<td>30</td>
<td>2^30</td>
</tr>
<tr>
<td>Type A</td>
<td>Known plaintext</td>
<td>4</td>
<td>Gallais and Kizhvatov (2011)</td>
<td>1.25</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>Type B</td>
<td>Known plaintext</td>
<td>4</td>
<td>MDATDCA</td>
<td>2–3</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Type B</td>
<td>Known plaintext</td>
<td>5</td>
<td>MDATDCA</td>
<td>2–6</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Type B</td>
<td>Known plaintext</td>
<td>3</td>
<td>MDATDCA</td>
<td>2–5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Type B</td>
<td>Known plaintext</td>
<td>4</td>
<td>Acıı¨c¸mez and Koç (2006a)</td>
<td>1.25</td>
<td>40</td>
<td>2^33.50</td>
</tr>
<tr>
<td>Type B</td>
<td>Known plaintext</td>
<td>4</td>
<td>MDATDCA</td>
<td>2–10</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Type B</td>
<td>Known plaintext</td>
<td>5</td>
<td>Acıı¨c¸mez and Koç (2006a)</td>
<td>1.25</td>
<td>55</td>
<td>2^34.26</td>
</tr>
<tr>
<td>Type B</td>
<td>Known plaintext</td>
<td>5</td>
<td>MDATDCA</td>
<td>2–10</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>Type C</td>
<td>Known ciphertext</td>
<td>3</td>
<td>Bonneau (2006)</td>
<td>1</td>
<td>10</td>
<td>2^30</td>
</tr>
<tr>
<td>Type C</td>
<td>Known ciphertext</td>
<td>3</td>
<td>MDATDCA</td>
<td>1</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>Type C</td>
<td>Known ciphertext</td>
<td>3</td>
<td>Acıı¨c¸mez and Koç (2006a)</td>
<td>1</td>
<td>20</td>
<td>2^24.22</td>
</tr>
<tr>
<td>Type C</td>
<td>Known ciphertext</td>
<td>4</td>
<td>Bonneau (2006)</td>
<td>1</td>
<td>14</td>
<td>2^30</td>
</tr>
<tr>
<td>Type C</td>
<td>Known ciphertext</td>
<td>4</td>
<td>MDATDCA</td>
<td>1</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>Type C</td>
<td>Known ciphertext</td>
<td>5</td>
<td>Acıı¨c¸mez and Koç (2006a)</td>
<td>1</td>
<td>20</td>
<td>2^23.97</td>
</tr>
<tr>
<td>Type C</td>
<td>Known ciphertext</td>
<td>5</td>
<td>Bonneau (2006)</td>
<td>1</td>
<td>25</td>
<td>2^30</td>
</tr>
<tr>
<td>Type C</td>
<td>Known ciphertext</td>
<td>5</td>
<td>MDATDCA</td>
<td>1</td>
<td>15</td>
<td>1</td>
</tr>
</tbody>
</table>
9. Case 2: error-tolerant MDATDCAs on AES-128

In this section, we will describe how to conduct MDATDCA on AES-128 in practice, where there are errors when deducing cache events.

Similar to Gallais and Kizhvatov (2011), 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-tolerance 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 \( V_H \leq V_t \leq V_M \), \( q_t \) is considered as an 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 \( d(\langle y_i \rangle_b) \), 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

\[
\begin{align*}
\ell_1 \lor c_2 \lor \ldots \lor c_n &= 1 \\
\end{align*}
\]

2. \( q_t \) is a miss.

Then the impossible deduction set of \( d(\langle y_i \rangle_b) \) 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 \( e = N_e/N_A \), 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 Gallais and Kizhvatov (2011) which is the only literature about TDCA on AES with error tolerance. We can see that, our error-tolerant

![Fig. 8 — Number of required cache traces for a full key recovery of error-tolerant MDATDCA on AES.](image-url)
MDATDCA can analyze the cache events of the first three rounds and require less cache traces than (Gallais and Kizhvatov, 2011). Moreover, when the error rate is as high as 90%, MDATDCA still works with 80 cache traces, which is better than 80% in Gallais and Kizhvatov (2011).

### 10. Case 3: MDATDCAs on AES-128 with preloaded cache

The MDATDCAs in Sections 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 (Bonneau, 2006; Gallais et al., 2011; Gallais and Kizhvatov, 2011). This section presents the cache analysis strategy and experimental results of MDATDCAs on AES-128 with partially preloaded cache (Table 4).

#### 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 (Gallais and Kizhvatov, 2011). Even when ten

---

**Table 4 – Comparisons of error-tolerant MDATDCAs on AES with previous work.**

<table>
<thead>
<tr>
<th>Attacks</th>
<th>Attack rounds</th>
<th>Error rate</th>
<th>Number of cache traces</th>
<th>Key search space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gallais and Kizhvatov (2011)</td>
<td>1.25</td>
<td>0.0</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>MDATDCA</td>
<td>3</td>
<td>0.0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Gallais and Kizhvatov (2011)</td>
<td>1.25</td>
<td>0.2</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>MDATDCA</td>
<td>3</td>
<td>0.2</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Gallais and Kizhvatov (2011)</td>
<td>1.25</td>
<td>0.4</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>MDATDCA</td>
<td>3</td>
<td>0.4</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>Gallais and Kizhvatov (2011)</td>
<td>1.25</td>
<td>0.6</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>MDATDCA</td>
<td>3</td>
<td>0.6</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>Gallais and Kizhvatov (2011)</td>
<td>1.25</td>
<td>0.8</td>
<td>158</td>
<td>10</td>
</tr>
<tr>
<td>MDATDCA</td>
<td>3</td>
<td>0.8</td>
<td>48</td>
<td>1</td>
</tr>
<tr>
<td>MDATDCA</td>
<td>3</td>
<td>0.9</td>
<td>80</td>
<td>1</td>
</tr>
</tbody>
</table>

---

**Table 5 – Comparisons of partially preloaded MDATDCA on AES with previous work.**

<table>
<thead>
<tr>
<th>Preloaded lines</th>
<th>Required traces in Gallais and Kizhvatov (2011)</th>
<th>Required traces in this paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>61</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>–</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>119</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
<td>40</td>
</tr>
<tr>
<td>8</td>
<td>296</td>
<td>80</td>
</tr>
<tr>
<td>10</td>
<td>–</td>
<td>120</td>
</tr>
</tbody>
</table>

---

**Fig. 9 – Leakages in TDCA on AES-128.**
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 Gallais and Kizhvatov (2011).

11. Extensions of MDATDCA 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 ($\{i\}$ 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 $K^2$ (when computing $X_i$) 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 $K^2$ 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 Gallais and Kizhvatov (2011) only used the first 4 table lookups, in which 128 bits of $K^0$ and 16-bit of $K^1$ can be leaked.

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.
Take TDCA on AES of Type A when $b = 4$ as an example. With the most efficient TDCA technique in Gallais and Kizhvatov (2011), 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 $2^{112}$.

In MDATDCA on AES-256, 60 bits of $K^0$ can be recovered if analyzing the first round, 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 min 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 $K^1$.

Take TDCA on AES of Type A when $b = 4$ as an example. With the TDCA technique in Gallais and Kizhvatov (2011), 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 min 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 TDCCAs. 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 TDCAs 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 Bonneau and Mironov (2006). When such a masking scheme is used, our attacks will outperform higher order DPA or CPA attacks that typically require thousands of traces. The countermeasures of MDATDCA is the same as TDCAs, which are widely discussed in the previous works (Aciı̈mez and Koç, 2006a, 2006b; Bertoni et al., 2005; Bonneau, 2006; Fournier and Tunstall, 2006; Gallais et al., 2011; Gallais and Kiazhatov, 2011; Lauraudoux, 2005). They include pre-fetching the lookup table into the cache prior to encryption and shuffling the order of table lookup computations. Meanwhile, Intel and other processor manufacturers have started to implement AES-NI and used on-chip atomic instructions that do not use the cache to implement AES (WATER PAPER, 2010), which can be used to prevent TDCAs efficiently.

Note that MDATDCA can also be extended to improve TDCAs on other block ciphers, such as Camellia (Poddar et al., 2011) and CLEFIA (Rebeiro and Mukhopadhyay, 2010, 2011; Rebeiro et al., 2011; Zhao and Wang, 2010). 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 master 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. This work was supported in part by the National Natural Science Foundation of China under the grants 60772082, 61173191, 61272491, and the US National Science Foundation under the grant CNS-0644188.

Appendix 1. Building the AES equation set.

We adopt the technique in Knudsen and Miolane (2010) to derive every S-Box output bit (denoted as $y_1 y_2 \ldots y_8$) with high-degree equations (degree 7) from the 8 S-Box input bits (denoted as $x_1 x_2 \ldots x_8$), which provides an explicit representation of the dependence of the S-Box output on the input. An example of $y_1$, represented by $x_1 x_2 \ldots x_8$ is shown as below.

$$y_1 = x_1 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_9 \oplus x_{10} \oplus x_{11} \oplus x_{12} \oplus x_{13} \oplus x_{14} \oplus x_{15} \oplus x_{16}$$

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)$^a$ | 128        |
| SubBytes                       | $262 \times 16 = 4192$ | $254 \times 16 = 4064$ |
| ShiftRows                      | 128     | 128        |
| MixColumns                     | 128     | 128        |
| One encrypt                    | 4704 (4832, 4576)$^b$ | 4448 (4320)$^c$ |
| Round                          | 1184 (1312)$^d$ | 1144        |

a 384 for the first AddRoundKey, and 256 for the others.

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

c 4448 for the first 9 rounds and 4320 for the final round.

d 1312 for the first round and 1184 for the others.

References


Bernstein DJ. Cache-timing attacks on AES. Available at: http://cr.yp.to/papers.html#cachetiming; 2004.
Xinjie Zhao received his B.E. degree and M.E. degree in Department of Computer Engineering, Ordonez Engineering College, Shijiazhuang, China, in 2006 and 2009, respectively. He is currently a Ph.D. student in Department of Information Engineering, Ordonez Engineering College, Shijiazhuang, China. His main research interest includes side channel analysis, fault analysis and combined analysis of block ciphers. He won the best paper in the 3rd International Workshop on Constructive Side-channel Analysis and Secure Design (COSADE 2012).
Zhizhe Shi is an Associate Professor of Computer Science and Engineering at the University of Connecticut (UConn). 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. Professor Shi has received US National Science Foundation CAREER award in 2006. His research interests include hardware mechanisms for secure and reliable computing, side-channel attacks and countermeasures, primitives for ultra-efficient cipher designs, sensor network security, application specific instruction-set instruction processor (ASIP) and embedded system designs, and underwater sensor networks.
Zhe Liu received the master degree from University of Luxembourg, Luxembourg, in October 2010, and excellent graduated from Shandong University for B.E. degree and M.E. degree, China, in June 2008 and 2011, respectively. Since 2011, he is a PhD student at Laboratory of Algorithmics, Cryptology and Security (LACS), University of Luxembourg. His supervisor is Professor Jean-Sebastien Coron. His research interests include cryptography and computer security. Specifically, he is major in elliptic curve cryptography and pairing-based cryptography on smart-card,
including fast implementation, side-channel attack and countermeasures.

Jean-François Gallais is a Ph.D. candidate at the University of Luxembourg. He received his Bachelor and Master Degree in Mathematics at the University of Caen (France) respectively in 2006 and 2008. His supervisor is Prof. Volker Müller. His research efforts concentrate on side-channel information leakages induced by micro-architectural mechanisms featured in embedded cryptographic implementations.