Multilateral White-Box Cryptanalysis

- Case study on WB-AES of CHES Challenge 2016 -

Hyunjin Ahn[1], and Dong-Guk Han[1,2]

[1] Department of Financial Information Security, Kookmin University, Seoul, Korea
[2] Department of Mathematics, Kookmin University, Seoul, Korea
{ahz012, christa}@kookmin.ac.kr

Abstract. Security requirement of White-Box Cryptography (WBC) is that it should protect secret key from white-box security model permits an adversary who is able to entirely control execution of the cryptographic algorithm and its environment. It has already been demonstrated that most of the primitive is vulnerable to algebraic attacks in the white-box security perspective. In recently, a new Differential Computation Analysis (DCA) attack is proposed that thwarts White-Box AES (WB-AES) by monitoring accessed memory information during execution of the algorithm. Though it requires ability to estimate internal information of memory pattern, the attack retrieves secret key with a few attempts. In addition it is proposed that the existence of vulnerability on hardware implementation of WB-AES against to Differential Power Analysis (DPA) attack. In this paper, we propose DPA based attack which directly exploits intermediate value of WB-AES computation without effort to take memory data. And demonstrate its practicability with respect to public software implementation of WB-AES. Additionally, we investigate vulnerability of our target primitive on DPA by acquiring actual power consumption traces of software implementation.

Keywords: White-Box Cryptanalysis, Side-Channel Attack, Software Implementation

1 Introduction

Secret key management is as important as design of robust cryptographic algorithm to cryptanalysis. In order to protect key extraction, secure memory technique is introduced such as ARM TrustZone[1] which prevents leakage of sensitive information from the memory. However, high price is a drawback of it. As one of the attempts to solve the problem, white-box implementation is proposed which interleaves the secret key in the software program. The technique aims to hide the sensitive data in the cryptographic implementation to make it hard to discover the data from there.

In this concept, white-box security model is happened which ensures resistance to the assumption of an adversary who is able to fully control the device

[1] http://www.arm.com/products/processors/technologies/trustzone/

processing the cryptographic algorithm. In particular, he can take from source code to entire information corresponding to the algorithm computation. In 2002, concrete WBC implementation for Data Encryption Standard (DES) and Advanced Encryption Standard (AES) were proposed by Chow *et al.* in [4] and [5], respectively. However, a variety of studies demonstrate that they vulnerable to algebraic attacks [2, 10, 13, 14]. Xiao *et al.* propose a new design of WB-AES in [17] that is tolerant the BEG attack regarded as effective algebraic attack [2] against Chow's WB-AES implementation.

Recently, in [3], Bos *et al.* introduce a novel Side-Channel Attack (SCA) retrieving secret key by exploiting accessed memory information during Chow's WB-AES execution. The attack applies DPA using mean-difference on the memory data to distinguish correct key. Pascal *et al.* demonstrates DPA vulnerability of WB-AES hardware implementation through power consumption traces measured onto actual evaluation board embedding FPGA chip in [15]. Unlike previous one, the attack adopts correlation coefficient instead of mean-difference.

In this paper we introduce Differential Data Analysis (DDA) which reveals secret key by applying DPA on overall output value of table look-up operation while WB-AES computation. An adversary who has ability to achieve entire intermediate values within the WB-AES is able to readily perform the attack. We demonstrate practicability of this attack against to public WB-AES software implementation of CHES Challenge 2016[2]. From the attack, all of secret key bytes is successfully recovered with over 200 acquired traces. In addition, we verify the vulnerability of the WB-AES which identical with previous target on power consumption measured from XMEGA128D4 microprocessor. The attack retrieves 14 of 16 key bytes with only 2,000 acquired software traces.

The remainder of the paper is organized as follows. Section 2 describes basic design of WB-AES and existing SCA based attack methods. In Section 3, we introduce our DDA attack and investigate its performance. Taking into account ChipWhisperer-lite evaluation board, we experimentally demonstrate if the WB-AES has vulnerability on software power consumption trace in Section 4. Section 5 concludes this paper with mention of further work.

2 Preliminaries

2.1 White-Box AES Implementation

In this section we briefly introduce White-Box AES architecture of Chow *et al.* which is proposed in [5] and referred to basic design. The WB-AES computation is composed to a series of table look-up operations taking advantage of three different types of table as follows:

- TBoxTy table: $Ty_j \circ T_i^r(x) = Ty_j(T_i^r(x)) = Ty_j(Sbox(x \oplus \hat{k}_{r-1}[i]))$

[2] This contest is held on Conference on Cryptographic Hardware and Embedded Systems 2016 (CHES 2016) and verifies secret key recovery skill of participant. Available from https://ctf.newae.com/

- XOR table: $XOR(x, y) = x \oplus y$
- TBox table: $T_i^{10}(x) = Sbox(x \oplus \hat{k}_9[i]) \oplus k_{10}[i]$

where index of state byte $i \in \{0, ..., 15\}$, round $r \in \{1, ..., 9\}$, input index of MixColumns $j \in \{0, ..., 3\}$ and $\hat{k}$ indicates round key applied ShiftRows. The XOR table yields exclusive-or of two 4-bit inputs x and y. The TBox has 8-bit input and output value, and the TBoxTy table yields 32-bit output from 8-bit input. For the MixColumns, four Ty_j tables are exploited as if AES T-table implementation [7], which are defined as follows:

$$Ty_0(x) = x \cdot \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix}, Ty_1(x) = x \cdot \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix}, Ty_2(x) = x \cdot \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix}, Ty_3(x) = x \cdot \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix}.$$

Finally, for 4 input bytes x_0, x_1, x_2 and x_3, MixColums is identical with $Ty_0(x_0) \oplus Ty_1(x_1) \oplus Ty_2(x_2) \oplus Ty_3(x_3)$, where exclusive-or is fulfilled by combining multiple XOR tables. Round function of AES is performed with ShiftRows, TBboxTy and XOR tables in sequence while final round consists of ShiftRows and TBox table.

Since white-box security permits an attacker who is able to fully control WBC computation, it is easy to extract secret key from corresponding look-up table. Note that an adversary readily achieves contents of tables by using disassembler or debugger. Intuitively, a secret key byte is determined through investigation of an TBoxTy table with key candidates of 2^8. In order to protect the table based WB-AES implementation, internal encoding rule is applied. For a table T, we make protected new table $T^{'} = g \circ T \circ f^{-1}$ by determining both bijection functions input encoding f and output encoding g.

Fig 1 (a) depicts four result bytes of round 1 adopting internal encoding and Fig 1 (b) shows round 2. In the figure, L_0^r, L_1^r, L_2^r, L_3^r are the four 8-bit to 8-bit invertible linear transformations, so-called mixing bijection, in round r. The L^{r+1} is identical with $L_0^{r+1} \| L_{13}^{r+1} \| L_{10}^{r+1} \| L_7^{r+1}$ due to ShiftRows of round $r+1$. The MB is 32-bit to 32-bit mixing bijection and MB_0^{-1}, MB_1^{-1}, MB_2^{-1} and MB_3^{-1} are 8-bit to 32-bit tables. In addition, to thwarts *code lifting* attacks [6], external encoding rule is applied in many WBC implementation. Entire storage for look-up tables is of 508 KB and the WB-AES is slowdown 55 times than standard AES. We refer the interested reader to [5, 11].

2.2 State-of-the-art SCA on WBC

Recently, several white-box cryptanalysis are published, which exploit side-channel information emitted during WBC computation [3, 15]. In this section, we describe both existing white-box cryptanalysis attacks on side-channel analysis perspective. Those have both assumptions that the attacker is able to acquire a number of trace with randomly chosen plaintext and does not need to consider external

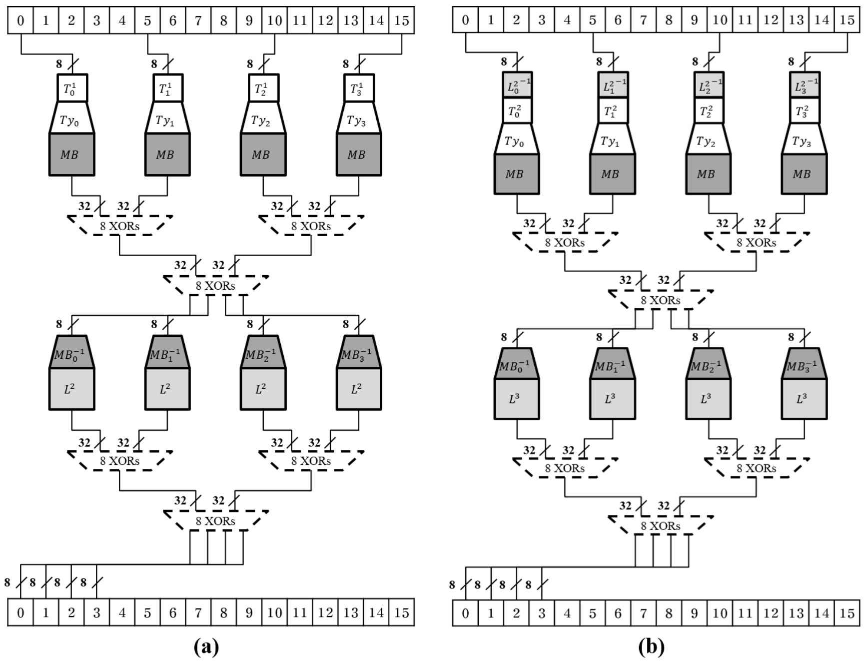

Fig. 1: WB-AES round structure applied internal encoding on round 1 (a) and 2 (b).

encoding of the target WBC. In other world, the target has not been applied the external encoding or the attacker should know the encoding rule if the WBC includes external encoding technique.

Differential Computation Analysis. Bos *et al.* proposes a novel attack method Differential Computation Analysis in [3] which thwarts WB-AES by using *software execution trace* consists of accessed memory address and data throughout the WBC operation. The DCA procedure is composed of 4 steps, an optional first step and three fundamental steps. In the first optional step, the attacker measures a *software execution trace* throughout overall the WBC computation, followed by identify where the WBC is manipulated by visualizing the trace with method presented in [12]. Now the attacker is able to acquire multiple *software execution traces* with diminished storage capacity by intensively collecting only portion of the WBC computation. In the second step, the attacker takes the number of traces with random plaintext and converts it to binary representation (zero or one) to make it suitable to conventional DPA tool in the third step. Finally, the attacker reveals secret key by using original DPA tool using mean-difference on the converted *software execution trace* instead of power consumption.

Differential Power Analysis on Hardware Implementation. In [15], Sasdrich *et al.* presents practical attack result of DPA using correlation coefficient on hardware implementation of the WB-AES operated onto FPGA platform. They implement the algorithm in hardware concept and demonstrate how much it is vulnerable to the DPA in gray-box security model. In the paper, they theoretically prove the existence of fraction in the structure of their target algorithm and examine with respect to SAKURA-X evaluation board. This is the first investigation of WBC weakness taking into account H/W power consumption as side-channel information.

3 Vulnerabilities Raising out of WBC Implementation

Existing SCA on WB-AES (described in Section 2.2) extract secret key from *software execution trace* consists of memory data and address, and power consumption for FPGA implementation by using DPA based distinguisher with the output of first round Sbox as intermediate value. Both vulnerabilities are results from correlation between considered side-channel information and intermediate value. These relations yield the fact that there exist some intermediate results of WB-AES which are significantly related to the Sbox output than the side-channel source. Note that most of the side-channel information includes noise as well as sensitive data. In conclusion, DPA for intermediate value of the WB-AES computation outperforms power consumption trace as side-channel information. Hereafter, for the sake of simplicity, we denote the DPA attack on computation data of WBC as Differential Data Analysis (DDA). In addition, though DCA applies mean-difference in [3], we adopt person's correlation coefficient on every attacks (DDA, DCA, DPA) as if Correlation Power Analysis (CPA) [1] instead of mean-difference to investigate in the identical manner.

We calibrate performance of DDA on public WB-AES of CHES Challenge 2016. Although it already has been demonstrated that the WB-AES has vulnerability, 20 participants recovered secret key of the target in the challenge, we exploit the implementation to merely estimate ability of DDA. The target implementation uses 4,048 look-up tables and 41 local variables (8-bit data) to store result of table. The WB-AES computation is composed to 4,080 table load and store operations, the loaded value is set to one of the variables. We denote the set of stored intermediate values during the WB-AES execution as a *data trace* in which consists of 4,080 samples for our target.

For DDA evaluation we acquire 5,000 *data traces* according to randomly chosen plaintext per every execution, followed by modify it to two different types. First one is binary representation, *Bit-data trace*, and the other consists of Hamming weight value of *data trace* elements, *HW-data trace*. Since $Ty_j \circ T_i^r$ yields Sbox output (S_i), two times polynomial multiplication of MixColumns ($\{02\} \cdot S_i$) and three times product ($\{03\} \cdot S_i$), we take into account each 3 results of $Ty_j \circ T_i^r$ as intermediate value in DDA. Remark that existing research for DCA and DPA demonstrate that both *software execution trace* and current trace for H/W implementation may be significantly related to 1-bit output of the Sbox. Intuitively, we can expect that the relation is result from $Ty_j \circ T_i^r$ yielding S_i even if the WB-AES has table for $MB \circ Ty_j \circ T_i^r$ instead of $Ty_j \circ T_i^r$. In the same context, both $\{02\} \cdot S_i$ and $\{03\} \cdot S_i$ also can refer to both DCA and DPA as intermediate value.

Fig 4 (a) in Appendix shows DDA results on *Bit-data trace* for 8 individual bit of three intermediate values and Fig 4 (b) presents *HW-data trace*'s results. To divide success or failure we impose Relative Distinguishing Margin[3] (*RelMarg*) which means successful attack when it has positive value. Table 1 and 2 shows summary of both results, respectively. Table element indicates the number of bit for each intermediate value, which is yield over *RelMarg* of 0.1 and marked with gray in Fig 4. In DDA on *Bit-data trace* with intermediate value S_i, 15 key bytes are revealed except for 13th byte while the others recover overall secret key. In general, attack results exploiting *HW-data trace* have low performance when compared with the other, there is not any intermediate value recovers overall key bytes. Nevertheless, the attack retrieves full secret key when combining results of three intermediate values.

inter. \ i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**rate**
S_i	1	2	2	2	2	1	3	3	3	1	3	3	2	0	4	1	15/16
$\{02\} \cdot S_i$	1	2	2	1	2	1	2	1	3	4	2	3	1	1	4	4	16/16
$\{03\} \cdot S_i$	3	4	4	4	3	1	2	5	4	5	2	3	3	2	2	2	16/16
total	5	8	8	7	7	3	7	9	10	10	7	9	6	3	10	7	-

Table 1: Summary of DDA on *Bit-data trace* with three intermediate values

inter. \ i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**rate**
S_i	2	0	1	1	2	1	0	2	3	0	3	3	2	0	2	2	12/16
$\{02\} \cdot S_i$	1	0	1	1	3	3	1	0	2	2	2	3	2	0	3	2	13/16
$\{03\} \cdot S_i$	0	3	2	2	2	1	3	4	2	2	0	2	4	2	1	1	14/16
total	3	3	4	4	7	5	4	6	7	4	5	8	8	2	6	5	-

Table 2: Summary of DDA on *HW-data trace* with three intermediate values

From DDA attack result on *Bit-data trace*, it is demonstrated that our target implementation (WB-AES of CHES Challenge 2016) has essential weakness for

[3] This distinguisher is proposed by Whitnall *et al.* in [16] and has positive value when correct key is revealed while negative if it fails to recover the key. $RelMarg = \frac{\rho(k^*) - max\{\rho(k)|k \neq k^*\}}{\sqrt{var\{\rho(k)|k \in \mathcal{K}\}}}$, where ρ is person's correlation coefficient, k^* is correct key, $var\{\cdot\}$ is variance of $\cdot$ and $\mathcal{K}$ is guess key space.

security of its design. Since DDA on *HW-data trace* successfully reveals overall secret key, it is likely that the WB-AES vulnerable to DPA on power consumption trace from software implementation following Hamming weight model.

4 Practical Experiments

In this section we show experimental results of DCA and DPA on our target WB-AES. By comparing DCA with DDA on *Bit data trace*, we identify how well the DCA is able to follow attack performance of the DDA. Through DPA on actual power consumption trace for software implementation we verify if the WB-AES has vulnerability in that environment. As previously mentioned, in this section, we apply correlation coefficient not mean-difference on both DCA and DPA.

4.1 DCA attack

Prior to DPA weakness verification of the WB-AES in software implementation environment, we investigate vulnerability on DCA. From executable file generation to *software execution trace* acquisition is run on Linux. We compile the WB-AES as 32-bit binary on 64-bit Debian 8 with Address Space Layout Randomization (ASLR) disabling. In order to collect memory usage information during the WB-AES computation, we exploit free downloadable public tool TracerPIN[4] which uses Intel's Dynamic Binary Instrumentation (DBI) tool Pin [9].

As stated in [15], there are three type of *software execution trace*, however, we only exploits accessed memory address. In fact, we experimentally identified that both *software execution traces* for address and accessed data are suitable to thwart our target with DCA while stack data is not. Beside former two traces has significantly similar attack performance. We record 5,000 *software execution traces* during operation of our compiled executable file with arbitrary plaintext per every execution. Table 3 shows summary of attack result under the identical condition with DDA of previous section and Fig 5 presents in detail. Though overall bits revealing secret key are not identical with ones of Fig 4 (a), attack performance is fairly similar each other. Both attacks recover 15 of 16 key bytes when intermediate value of S_i and full secret key for $\{02\} \cdot S_i$ or $\{03\} \cdot S_i$.

4.2 DPA attack

Our aim is to examine weakness of the WB-AES with respect to DPA attack on software implementation environment. To do so, we acquire multiple power consumption traces from ChipWhisperer-lite board [8] manipulating the WB-AES with randomly chosen plaintext per every execution. The board mainly consists of two parts, main board and target board, and measures power consumption

[4] https://github.com/SideChannelMarvels

i / *inter.*	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**rate**
S_i	2	2	2	2	3	2	3	2	3	1	3	2	2	0	4	2	15/16
$\{02\} \cdot S_i$	1	2	2	2	3	1	2	1	3	4	2	2	1	1	4	2	16/16
$\{03\} \cdot S_i$	3	4	3	4	3	1	2	6	4	4	1	3	3	1	2	1	16/16
total	6	8	7	8	9	4	7	9	10	9	6	7	6	2	10	5	-

Table 3: Summary of DCA on memory address data with three intermediate values

on target board equipped with Atmel XMEGA128D4-u processor having 128 KB Flash memory. Unfortunately, code size of the WB-AES is too large to be programmed into the board. Therefore we compile portion of the source code which leaks sensitive information helping key recovery. Remark, the purpose of this experiment is to investigate if we can retrieve secret key from S/W power consumption, not practicability examination. Since 1st round of WB-AES is computed per each column exploiting 4 Sbox outputs (cf. Fig 1), we take 4 trace types for each column. Concretely, first portion is composed of specific table look-up operations which have one of 4 plaintext bytes $plain[0]$, $plain[5]$, $plain[10]$ and $plain[15]$ as input. Fig 2 shows source code of the portion exploited to acquire the first type of trace. The code has size of 7,364 bytes for program and 4,336 bytes for data when compiling with option 's' optimizing code size. In the similar way, the remainder of portion type is decided.

```
void chow_aes3_encrypt_wb(const unsigned char plain[], unsigned char cipher[])
{
    unsigned char v0, v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, ... , v41;

    v0 = plain[0];   v1 = plain[4];   v2 = plain[8];    v3 = plain[12];
    v4 = plain[1];   v5 = plain[5];   v6 = plain[9];    v7 = plain[13];
    v8 = plain[2];   v9 = plain[6];   v10 = plain[10];  v11 = plain[14];
    v12 = plain[3];  v13 = plain[7];  v14 = plain[11];  v15 = plain[15];

    v16 = lookup_nibble(table_13648, v0);    v17 = lookup_nibble(table_13649, v0);
    v18 = lookup_nibble(table_13650, v5);    v19 = lookup_nibble(table_13651, v5);
    v18 = lookup_nibble(table_13652, v10);   v19 = lookup_nibble(table_13653, v10);
    v20 = lookup_nibble(table_13654, v15);   v21 = lookup_nibble(table_13655, v15);
    v18 = lookup_nibble(table_13656, v0);    v19 = lookup_nibble(table_13657, v0);
    v20 = lookup_nibble(table_13658, v5);    v21 = lookup_nibble(table_13659, v5);
    v20 = lookup_nibble(table_13660, v10);   v21 = lookup_nibble(table_13661, v10);
    v22 = lookup_nibble(table_13662, v15);   v23 = lookup_nibble(table_13663, v15);
    v20 = lookup_nibble(table_13664, v0);    v21 = lookup_nibble(table_13665, v0);
    v22 = lookup_nibble(table_13666, v5);    v23 = lookup_nibble(table_13667, v5);
    v22 = lookup_nibble(table_13668, v10);   v23 = lookup_nibble(table_13669, v10);
    v24 = lookup_nibble(table_13670, v15);   v25 = lookup_nibble(table_13671, v15);
    v22 = lookup_nibble(table_13672, v0);    v0 = lookup_nibble(table_13673, v0);
    v23 = lookup_nibble(table_13674, v5);    v5 = lookup_nibble(table_13675, v5);
    v5 = lookup_nibble(table_13676, v10);    v10 = lookup_nibble(table_13677, v10);
    v23 = lookup_nibble(table_13678, v15);   v15 = lookup_nibble(table_13679, v15); }
```

Fig. 2: Overall source code of the portion for the first type of trace. Four red variables are overwriting operation into plaintext bytes.

Now we are able to program each type of portion codes into our board and collect power consumption trace. However, there is a constraint to apply DPA attack on measured traces. In the portion code, the table look-up operation is processed through a user-defined function, *lookup_nibble*, which yields 4-bit output from a declared table corresponding to input value as follows:

$\#define\ lookup\_nibble\ (t, i)\ (t[i >> 1] >> ((i\&1) * 4)\&0xf)$.

If i is odd value then the function computes $t[i >> 1] >> 4\&0xf$, while it operates $t[i >> 1]\&0xf$ when even. Therefore, look-up function outputs through distinct operation process based on type of input value. Fig 3 shows both power consumption traces from ChipWhisperer-lite board manipulating the first portion code with odd (a) and even (b) plaintext, respectively. The portion code is performed during 1,643 samples for odd value and 1,003 samples when even plaintext. A look-up operation is conducted within approximately 48 and 28 samples, respectively. Therefore, if we acquire multiple power consumption traces with randomly chosen plaintext, we come up with misalignment problem. As previously mentioned, since we concentrate on investigation of vulnerability existence on S/W power consumption trace, we leave how to solve the problem out of the discussion instead measure both traces for each odd and even plaintext.

In the aggregate, we take 8 types of measured trace for each 4 portion codes and each 2 plaintext types (even and odd), and 50,000 traces are acquired per each trace types. Table 4 shows summary of attack result and Fig 6 presents in detail. The attack retrieves 14 of 16 key bytes with only 1,000 measured traces with respect to each plaintext types. In conclusion, DPA thwarts WB-AES of CHES challenge 2016 by acquiring overall 8,000 S/W traces, the number of 8 types of trace is 1,000.

inter. \ i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**rate**
S_i	1	0	0	1	0	0	1	0	3	0	3	3	1	0	1	0	8/16
$\{02\} \cdot S_i$	0	0	0	1	0	1	2	0	2	1	1	1	0	0	1	0	8/16
$\{03\} \cdot S_i$	1	2	2	2	1	0	1	2	2	0	1	3	2	0	0	0	11/16
total	2	2	2	4	1	1	4	2	7	1	5	7	3	0	2	0	-

Table 4: Summary of DPA on S/W trace with three intermediate values. Table element is sum of the results of both plaintext types.

5 Conclusions and Further Work

In this paper we proposed Differential Data Analysis (DCA) attack which recovers secret key on multiple entire intermediate value of WB-AES execution by applying DPA based distinguishment method. Through actual experiment, we verified feasibility of the attack with public WB-AES software implementation supported at CHES Challenge 2016. Our attack retrieved overall secret key

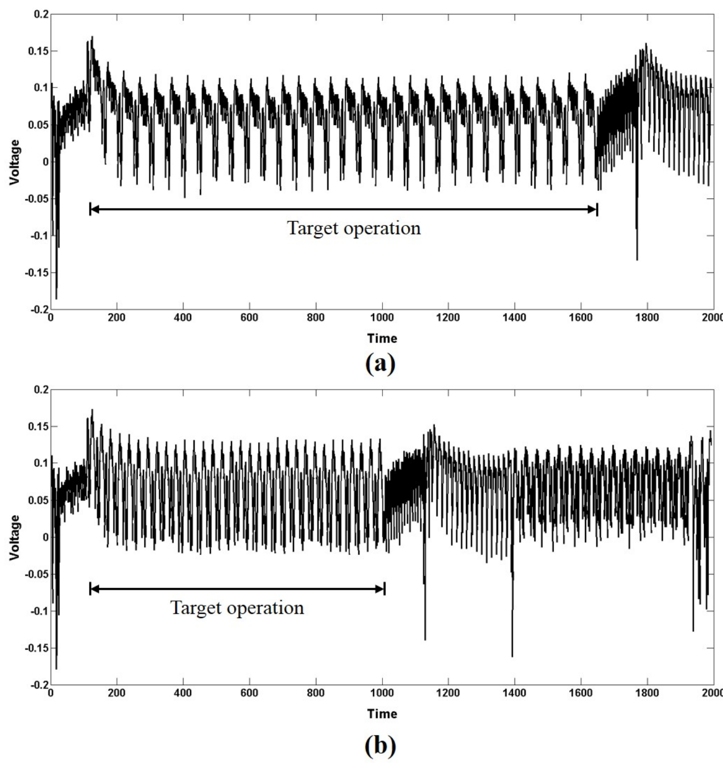

Fig. 3: Measured power consumption traces from ChipWhisperer-lite manipulating 1st portion code. (a) shows trace with respect to odd plaintext and (b) is acquired when even value.

from the target WBC with solely 200 acquisitions of intermediate data. Unlike the DCA, an adversary is available this attack without knowledge about memory information if he possess source code or knows look-up tables of the target.

In addition, we investigated availability of DPA on the WB-AES software implementation. In order to program our target onto ChipWhisperer-lite, we selected portion of source code which significantly leaks secret key information. From DPA on power consumption trace manipulating the portion code, we revealed 14 of 16 secret key bytes with 1,000 measured traces. However, as already mentioned in Section 4, we should solve alignment problem to make DPA feasible on our target and conduct DPA taking into account complete source code not portion. We leave this additional evaluation for future work.

References

1. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In: M. Joye, and J.-J. Quisquater (eds.) CHES 2004. LNCS, vol. 3156, pp. 16-29. Springer, Heidelberg (2004)
2. O. Billet, H. Gilbert, and C. Ech-Chatbi. Cryptanalysis of a White Box AES Implementation. In: H. Handschuh, and M. A. Hasan (eds.) SAC 2004. LNCS, vol. 3357, pp. 227-240. Springer, Heidelberg (2005)
3. J. W. Bos, C. Hubain, W. Michiels, and P. Teuwen. Differential Computation Analysis: Hiding your White-Box Designs is Not Enough. IACR Cryptology ePrint Archive, 2015, https://eprint.iacr.org/2015/753.pdf
4. S. Chow, P. A. Eisen, H. Johnson, and P. C. van Oorschot. A White-Box DES Implementation for DRM Applications. In: J. Feigenbaum (eds.) DRM 2002. LNCS, vol. 2696, pp. 1-15. Springer, Heidelberg (2003)
5. S. Chow, P. A. Eisen, H. Johnson, and P. C. van Oorschot. White-Box Cryptography and an AES Implementation. In: K. Nyberg, and H. Heys (eds.) SAC 2002. LNCS, vol. 2595, pp. 250-270. Springer, Heidelberg (2003)
6. C. Delerablée, T. Lepoint, P. Paillier, and M. Rivain. White-Box Security Notions for Symmetric Encryption Schemes. In: T. Lange, K. Lauter, and P. Lisoněk (eds.) SAC 2013. LNCS, vol. 8282, pp. 247-264. Springer, Heidelberg (2014)
7. J. Daemen, and V. Rijmen. AES Proposal: Rijndael. Technical Report. Available at http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf
8. C. OFlynn, and Z. D. Chen. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. In: E. Prouff (eds.) COSADE 2014. LNCS, vol. 2014, pp. 243-260. Springer, Switzerland (2014)
9. C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallance, V. J. Reddi, and K. Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In: V. Sarkar, and M. W. Hall (eds.) ACM 2005. pp. 190-200
10. T. Lepoint, M. Rivain, Y. D. Mulder, P. Roelse, and B. Preneel. Two Attacks on a White-Box AES Implementation. In: T. Lange, K. Lauter, and P. Lisoněk (eds.) SAC 2013. LNCS, vol. 8282, pp. 265-285. Springer, Heidelberg (2014)
11. J. A. Muir. A Tutorial on White-box AES. IACR Cryptology ePrint Archive, 2013, https://eprint.iacr.org/2013/104.pdf

12. C. Mougey, and F. Gabriel. Désobfuscation de DRM par attaques auxiliaires. In: Symposium sur la sécurité des technologies de l'information et des communications 2014. https://www.sstic.org/2014/presentation/dsobfuscation_de_drm_par_attaques_auxiliaires/
13. T. D. Mulder, P. Roelse, and B. Preneel. Revisiting the BGE Attack on a White-Box AES Implementation. IACR Cryptology ePrint Archive, 2013, https://eprint.iacr.org/2013/450.pdf
14. W. Michiels, P. Gorissen, and H. D. L. Hollmann. Cryptanalysis of a Generic Class of White-Box Implementations. In: R. M. Avanzi, L. Keliher, and F. Sica (eds.) SAC 2008. LNCS, vol. 5381, pp. 414-428. Springer, Heidelberg (2009)
15. P. Sasdrich, A. Moradi, and T. G´uneysu. White-Box Cryptography in the Gray Box A Hardware Implementation and its Side Channels . In: T. Peyrin (eds.) FSE 2016. LNCS, vol. 9783, pp. 185-203. Springer, Heidelberg (2016)
16. C. Whitnall, E. Oswald. A fair evaluation framework for comparing side-channel distinguishters. Journal of Cryptographic Engineering, 1(2): pp. 145-160. Springer, Verlag (August 2011)
17. Y. Xiao, and X. Lai. A Secure Implementation of White-Box AES. In: 2nd International Conference on Computer Science and its Applications, CSA 2009. pp. 1-6. IEEE 2009

A Synthesis of Attack Results on WB-AES of CHES Challenge 2016

Inter. value : S_i	Target bit							
Key Byte	c7	c6	c5	c4	c3	c2	c1	c0
0	8.298	-5.318	-4.067	-5.426	-1.328	-4.036	-5.021	-4.360
1	-5.261	-3.633	-6.269	8.533	2.298	-4.497	-5.164	-4.635
2	1.647	-5.161	-4.568	1.256	-2.624	-6.629	-6.639	-4.996
3	-5.389	-4.783	-3.615	11.556	-6.005	-6.470	-5.208	8.685
4	0.706	-3.964	-4.011	-4.911	-4.077	-0.855	9.755	1.999
5	-5.427	-5.653	0.404	0.847	-6.246	-3.440	-6.015	1.260
6	-4.806	-4.211	1.815	-4.964	1.295	-4.687	-5.166	1.404
7	8.959	-5.425	-5.615	-5.439	8.726	1.007	-5.093	-5.037
8	-5.432	0.625	-5.589	1.599	-3.165	9.029	8.767	-4.659
9	-5.184	-4.926	-4.393	7.325	-3.730	-4.506	-4.867	-5.494
10	8.296	0.247	-5.384	-4.236	11.531	-4.459	-5.590	11.558
11	-5.988	1.252	-3.156	-4.655	-5.210	-4.959	9.331	8.029
12	8.253	-4.710	-5.219	-4.960	-4.710	7.559	-4.895	-5.641
13	-6.995	-4.063	-4.132	-5.617	0.315	-4.520	-4.804	-0.467
14	2.927	-5.671	11.940	2.030	-6.436	1.797	-4.810	-4.429
15	0.179	-0.434	-5.868	-4.450	-5.266	-4.940	8.187	-4.437

Inter. value : $\{02\} \cdot S_i$	Target bit							
Key Byte	c7	c6	c5	c4	c3	c2	c1	c0
0	-5.318	-4.067	-5.426	-5.756	-6.787	-5.021	-5.417	8.298
1	-3.633	-6.269	8.533	-3.747	-4.732	-5.164	1.318	-5.261
2	-5.161	-4.568	1.256	-5.126	-4.678	-6.639	-4.842	1.647
3	-4.783	-3.615	11.556	0.945	-5.418	-5.208	-0.087	-5.389
4	-3.964	-4.011	-4.911	-3.928	8.108	9.755	-5.566	0.706
5	-5.653	0.404	0.847	-6.227	7.001	-6.015	-0.825	-5.427
6	-4.211	1.815	-4.964	-4.792	7.434	-5.166	-5.394	-4.806
7	-5.425	-5.615	-5.439	-0.048	-5.584	-5.093	-6.077	8.959
8	0.625	-5.589	1.599	-5.288	-5.815	8.767	1.863	-5.432
9	-4.926	-4.393	7.325	1.931	7.886	-4.867	1.497	-5.184
10	0.247	-5.384	-4.236	9.304	-4.673	-5.590	-5.212	8.296
11	1.252	-3.156	-4.655	-5.079	8.625	9.331	-6.468	-5.988
12	-4.710	-5.219	-4.960	-4.941	0.785	-4.895	-5.782	8.253
13	-4.063	-4.132	-5.617	-4.215	-7.966	-4.804	8.519	-6.995
14	-5.671	11.940	2.030	-4.626	-5.626	-4.810	1.882	2.927
15	-0.434	-5.868	-4.450	2.309	1.247	8.187	1.162	0.179

Inter. value : $\{03\} \cdot S_i$	Target bit							
Key Byte	c7	c6	c5	c4	c3	c2	c1	c0
0	-4.549	-6.355	-7.176	10.796	1.391	7.846	-4.327	-5.417
1	-4.495	8.782	0.951	-0.126	7.604	-6.816	11.546	1.318
2	7.493	8.295	1.370	0.345	7.873	-4.355	-5.169	-4.842
3	8.048	8.161	2.253	1.094	-5.381	0.336	-5.450	-0.087
4	-4.118	8.506	8.887	8.974	-4.569	-5.055	-5.749	-5.566
5	7.996	-6.277	-4.922	-4.971	-5.992	-4.295	-5.043	-0.825
6	-0.072	-6.880	-6.506	10.928	11.952	-5.535	-4.744	-5.394
7	7.872	0.773	6.989	8.498	8.400	-4.883	7.565	-6.077
8	-3.886	-6.665	1.043	1.368	-5.704	11.038	0.805	1.863
9	-4.186	8.460	10.715	-5.520	1.381	8.146	-5.134	1.497
10	-4.935	7.258	-4.746	-5.690	-6.384	-4.535	1.663	-5.212
11	8.774	8.210	-7.293	-5.086	6.938	-6.211	-5.205	-6.468
12	-1.046	-6.272	1.567	8.774	-5.328	-5.857	7.447	-5.782
13	0.939	1.241	-6.033	-5.223	-5.912	-4.674	-6.073	8.519
14	-4.586	7.396	-5.630	-6.339	-5.043	-5.902	-4.839	1.882
15	-5.600	0.205	7.519	-5.114	-5.650	-5.554	-5.787	1.162

(a)

Inter. value : S_i	Target bit							
Key Byte	c7	c6	c5	c4	c3	c2	c1	c0
0	5.614	-4.672	-3.599	-5.667	7.373	-4.905	-5.299	-1.355
1	-5.095	-5.244	-6.133	-2.845	0.905	-4.620	-4.827	-4.497
2	4.779	-5.152	-4.338	0.545	-4.847	-3.868	-4.902	-5.239
3	-5.883	-5.505	-5.023	4.872	-5.624	-4.588	-5.487	-2.407
4	1.136	-5.731	-4.964	-4.291	-5.192	0.224	4.649	-1.732
5	-5.006	-5.160	-4.471	1.297	-4.481	-4.241	-3.822	-5.296
6	-6.186	-3.317	-5.688	-6.312	-2.917	-6.638	-4.890	-0.065
7	-4.065	-5.123	-5.500	-4.788	5.470	1.161	-5.163	-5.117
8	-5.415	4.572	-4.962	-2.757	-3.360	7.172	1.427	-3.579
9	-5.457	-6.324	-2.652	0.969	-4.671	-5.161	-4.994	-3.211
10	1.422	-1.844	-5.720	-5.948	9.401	-4.646	-5.466	9.507
11	-5.436	4.653	-5.790	-5.022	-5.014	-4.107	4.668	7.500
12	4.723	-4.557	-4.779	-4.829	-4.856	4.643	-5.208	-5.568
13	-5.153	-3.850	-5.514	-3.130	-4.569	-3.879	-4.693	-4.828
14	1.897	-5.134	8.727	-2.718	-4.527	-3.902	-4.011	-5.288
15	4.263	-2.906	-4.856	-4.089	-4.692	-6.913	5.632	-5.334

Inter. value : $\{02\} \cdot S_i$	Target bit							
Key Byte	c7	c6	c5	c4	c3	c2	c1	c0
0	-4.672	-3.599	-5.667	-4.751	-5.320	-5.299	-4.156	5.614
1	-5.244	-6.133	-2.845	-3.582	-6.218	-4.827	-3.347	-5.095
2	-5.152	-4.338	0.545	-4.328	-6.434	-4.902	-3.981	4.779
3	-5.505	-5.023	4.872	-4.563	-4.708	-5.487	-3.361	-5.884
4	-5.731	-4.964	-4.291	-4.975	4.815	4.649	-1.396	1.136
5	-5.160	-4.471	1.297	-5.333	1.167	-3.822	2.066	-5.006
6	-3.317	-5.688	-6.312	-4.949	9.242	-4.890	-4.375	-6.186
7	-5.123	-5.500	-4.788	-2.173	-4.088	-5.163	-5.596	-4.065
8	4.572	-4.962	-2.757	-6.420	-4.834	1.427	-3.547	-5.415
9	-6.324	-2.652	0.969	1.759	4.450	-4.994	-2.901	-5.457
10	-1.844	-5.720	-5.948	1.655	-5.212	-5.466	-5.693	1.422
11	4.653	-5.790	-5.022	-5.481	1.593	4.668	-4.741	-5.436
12	-4.557	-4.779	-4.829	-5.495	5.055	-5.208	-4.721	4.723
13	-3.850	-5.514	-3.130	-3.413	-5.282	-4.693	-0.537	-5.153
14	-5.134	8.727	-2.718	-3.772	-4.703	-4.011	3.683	1.897
15	-2.906	-4.856	-4.089	-4.813	0.995	5.632	-2.726	4.263

Inter. value : $\{03\} \cdot S_i$	Target bit							
Key Byte	c7	c6	c5	c4	c3	c2	c1	c0
0	-0.812	-4.881	-5.033	-3.019	-5.816	-3.448	-5.619	-4.156
1	-5.823	7.379	-5.476	-2.934	4.502	-5.193	1.859	-3.347
2	4.718	-4.321	-4.811	-4.452	7.304	-5.359	-5.455	-3.981
3	3.690	-2.811	2.080	-2.383	-5.026	0.390	-4.615	-3.361
4	-4.996	1.145	-5.426	4.673	-3.864	-4.140	-4.842	-1.396
5	-3.581	-5.570	-2.050	-2.293	-5.174	-4.276	-4.060	2.066
6	4.727	-5.235	-4.679	5.270	5.816	-5.241	-5.105	-4.375
7	9.367	-3.175	-5.313	4.503	6.650	-4.662	1.433	-5.596
8	-2.188	-4.351	-2.431	4.398	-4.937	9.307	-3.431	-3.547
9	-3.685	-2.390	5.065	-5.148	1.418	-5.004	-5.477	-2.901
10	-5.303	-2.565	-5.290	-5.106	-3.142	-5.525	-2.972	-5.693
11	-2.894	4.872	-5.179	-6.064	7.534	-4.717	-3.719	-4.741
12	3.947	-2.509	3.651	4.089	-4.693	-4.969	9.984	-4.721
13	5.495	1.512	-3.794	-5.408	-5.055	-3.510	-5.013	-0.537
14	-5.230	-3.650	-5.302	-4.936	-4.158	-4.578	-5.827	3.683
15	-4.620	1.567	-3.290	-5.830	-5.544	-5.439	-4.977	-2.726

(b)

Fig. 4: (a) is DDA results on *Bit-data trace* with individual bit of each three intermediate values and (b) shows results on *HW-data trace*.

Inter. value : S_i		Target bit							
		c7	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	**c0**
Key Byte	**0**	8.478	-4.595	-5.106	-5.001	1.599	-4.793	-5.530	-4.685
	1	-5.504	-5.361	-4.709	7.822	1.876	-5.493	-4.734	-5.149
	2	1.083	-5.636	-4.618	1.094	-5.067	-4.724	-5.533	-4.710
	3	-3.713	-5.739	-4.792	8.001	-4.996	-6.166	-4.179	8.723
	4	1.379	-5.052	-3.098	-4.362	-5.116	1.376	9.300	0.797
	5	-5.941	-5.202	0.526	1.356	-5.895	-4.774	-5.375	1.357
	6	-6.853	-5.158	1.449	-6.129	2.436	-5.444	-6.132	1.422
	7	8.925	-4.783	-6.104	-4.697	7.487	0.377	-6.744	-5.293
	8	-5.020	0.554	-5.492	1.219	-4.024	8.510	8.955	-4.387
	9	-5.013	-3.748	-4.367	8.269	-4.331	-3.780	-4.412	-5.643
	10	7.089	0.518	-4.079	-4.404	11.623	-4.040	-4.936	11.807
	11	-5.440	0.015	-2.890	-5.369	-4.122	-4.974	8.504	7.334
	12	8.066	-5.348	-4.218	-7.153	-4.551	8.390	-5.705	-6.886
	13	-6.224	-3.692	-3.602	-5.676	0.332	-4.657	-5.298	0.898
	14	1.945	-5.686	12.282	1.515	-4.711	1.562	-5.409	-4.485
	15	1.022	0.500	-7.300	-6.374	-4.226	-4.892	7.015	-4.293

Inter. value : $\{02\} \cdot S_i$		Target bit							
		c7	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	**c0**
Key Byte	**0**	-4.595	-5.106	-5.001	-5.963	-5.529	-5.530	-5.737	8.478
	1	-5.361	-4.709	7.822	-3.946	-4.578	-4.734	1.420	-5.504
	2	-5.636	-4.618	1.094	-6.287	-5.108	-5.533	-5.014	1.083
	3	-5.739	-4.792	8.001	1.128	-5.313	-4.179	0.722	-3.713
	4	-5.052	-3.098	-4.362	-5.828	7.134	9.300	-6.809	1.379
	5	-5.202	0.526	1.356	-5.668	0.421	-5.375	-0.966	-5.941
	6	-5.158	1.449	-6.129	-6.062	7.511	-6.132	-5.153	-6.853
	7	-4.783	-6.104	-4.697	0.589	-4.374	-6.744	-4.971	8.925
	8	0.554	-5.492	1.219	-4.499	-5.552	8.955	1.798	-5.020
	9	-3.748	-4.367	8.269	1.346	7.325	-4.412	1.119	-5.013
	10	0.518	-4.079	-4.404	7.991	-4.975	-4.936	-5.079	7.089
	11	0.015	-2.890	-5.369	-3.907	8.912	8.504	-4.814	-5.440
	12	-5.348	-4.218	-7.153	-5.618	0.645	-5.705	-5.137	8.066
	13	-3.692	-3.602	-5.676	-4.327	-6.322	-5.298	8.913	-6.224
	14	-5.686	12.282	1.515	-4.887	-4.102	-5.409	1.983	1.945
	15	0.500	-7.300	-6.374	0.994	0.979	7.015	-5.115	1.022

Inter. value : $\{03\} \cdot S_i$		Target bit							
		c7	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	**c0**
Key Byte	**0**	-6.937	-5.202	-7.573	11.101	1.699	7.936	-5.219	-5.737
	1	-5.746	2.256	-0.099	-0.550	1.593	-5.615	10.884	1.420
	2	7.049	8.151	0.773	0.304	7.873	-4.968	-5.132	-5.014
	3	7.844	8.442	1.023	2.450	-5.269	-0.329	-4.923	0.722
	4	-5.691	7.407	9.391	8.723	-5.938	-4.902	-5.051	-6.809
	5	7.375	-6.335	-5.066	-6.811	-5.999	-5.019	-4.484	-0.966
	6	0.095	-5.789	-7.336	10.878	11.574	-4.315	-5.634	-5.153
	7	7.580	1.295	8.183	8.210	7.908	-5.366	8.395	-4.971
	8	-5.432	-6.397	1.257	0.884	-5.711	11.807	1.732	1.798
	9	-5.161	8.014	11.299	-5.755	0.444	8.721	-4.799	1.119
	10	-5.110	7.257	-5.298	-6.003	-6.820	-6.041	-0.111	-5.079
	11	7.763	7.828	-5.420	-5.119	7.494	-5.629	-4.121	-4.814
	12	-0.424	-6.161	2.398	8.152	-5.676	-6.811	7.936	-5.137
	13	0.593	0.726	-5.470	-4.913	-4.697	-3.958	-7.454	8.913
	14	-6.870	7.977	-5.999	-5.585	-6.575	-5.146	-4.681	1.983
	15	-5.784	0.860	7.721	-5.470	-5.804	-4.798	-5.187	-5.115

Fig. 5: DCA attack results on accessing memory address during WB-AES computation.

Inter. value : S_i		Target bit							
		c7	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	**c0**
Key Byte	**0**	-0.336	-3.462	-3.625	-4.026	-2.049	-4.640	-5.947	-2.497
	1	-4.361	-1.307	-4.003	-0.705	-3.417	-3.778	-4.057	-3.310
	2	-1.145	-2.315	-3.685	-3.731	-3.922	-3.795	-3.062	-5.852
	3	-4.779	-1.948	-4.728	6.837	-5.011	-1.846	-2.641	-6.813
	4	0.293	-1.746	-4.025	-3.272	-3.796	-0.942	-4.098	-2.665
	5	-5.162	-1.791	-4.360	-1.217	-1.698	-3.899	-3.627	-5.979
	6	-5.312	-4.393	2.525	-3.090	-2.839	-5.251	-4.363	0.061
	7	-7.342	-3.773	-3.467	-5.116	-0.663	-2.389	-5.110	-3.353
	8	-3.089	-2.650	-2.442	1.077	-3.948	2.446	6.875	-2.522
	9	-2.952	-2.418	-2.979	-2.263	-4.023	-4.066	-5.298	-2.645
	10	-2.675	-1.647	-5.396	-4.737	2.454	-3.893	-2.951	1.637
	11	-3.379	-2.824	-5.643	-4.681	-1.987	-4.276	-0.132	4.578
	12	-4.473	-3.366	-6.587	-3.165	-3.730	3.350	-4.751	-4.840
	13	-4.466	-2.196	-5.281	-4.178	-4.655	-4.219	-2.419	-4.417
	14	-1.373	-1.967	-0.198	-3.405	-2.896	0.118	-4.847	-5.395
	15	-0.833	-2.148	-7.907	-2.696	-3.396	-1.658	0.061	-4.357

Inter. value : $\{02\} \cdot S_i$		Target bit							
		c7	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	**c0**
Key Byte	**0**	-3.462	-3.625	-4.026	-5.377	-4.129	-5.947	-1.311	-0.336
	1	-1.307	-4.003	-0.705	-4.275	-2.501	-4.057	-1.470	-4.361
	2	-2.315	-3.685	-3.731	-1.192	-4.147	-3.062	-3.545	-1.145
	3	-1.948	-4.728	6.837	-0.849	-3.743	-2.641	0.475	-4.779
	4	-1.746	-4.025	-3.272	-1.250	0.446	-4.098	-4.489	0.293
	5	-1.791	-4.360	-1.217	-4.671	4.761	-3.627	-3.541	-5.162
	6	-4.393	2.525	-3.090	-2.050	2.730	-4.363	-3.098	-5.312
	7	-3.773	-3.467	-5.116	-5.071	-4.901	-5.110	-1.814	-7.342
	8	-2.650	-2.442	1.077	-4.232	-3.223	6.875	-1.788	-3.089
	9	-2.418	-2.979	-2.263	-2.015	1.787	-5.298	-4.142	-2.952
	10	-1.647	-5.396	-4.737	2.581	-3.010	-2.951	-4.335	-2.675
	11	-2.824	-5.643	-4.681	-3.606	-4.077	-0.132	-2.596	-3.379
	12	-3.366	-6.587	-3.165	-3.604	-1.834	-4.751	-2.508	-4.473
	13	-2.196	-5.281	-4.178	-3.525	-4.155	-2.419	-4.422	-4.466
	14	-1.967	-0.198	-3.405	-4.423	-4.931	-4.847	-1.238	-1.373
	15	-2.148	-7.907	-2.696	-1.542	-2.709	0.061	-2.727	-0.833

Inter. value : $\{03\} \cdot S_i$		Target bit							
		c7	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	**c0**
Key Byte	**0**	-6.563	-1.734	-4.361	6.152	-4.036	-4.742	-4.033	-1.311
	1	-3.436	1.169	-4.400	-2.412	4.519	-5.354	-4.634	-1.470
	2	3.559	-3.935	-4.702	-4.350	3.386	-5.859	-3.848	-3.545
	3	4.627	-3.315	-4.348	-2.238	-2.823	2.066	-1.788	0.475
	4	-3.683	-2.066	-4.888	-3.401	-2.650	-1.088	-4.342	-1.197
	5	-4.123	-4.948	-4.506	-4.547	-5.618	-3.817	-1.864	-3.541
	6	-1.043	-3.078	-4.290	-1.148	-1.818	-2.186	-4.519	-3.098
	7	2.713	-1.825	-5.241	5.829	-0.167	-4.814	-3.575	-1.814
	8	-4.490	-3.491	-5.033	-2.030	-5.621	3.553	-2.667	-1.788
	9	-3.388	-4.325	-0.282	-1.520	-4.368	-1.761	-3.797	-4.142
	10	-5.434	3.549	-4.786	-4.704	-5.287	-5.065	-2.355	-4.335
	11	-4.022	2.410	-5.190	-6.149	1.521	-5.123	-3.612	-2.596
	12	1.840	-1.380	0.859	4.626	-3.988	-3.208	-1.542	-2.508
	13	-0.429	-2.770	-5.060	-3.665	-2.243	-4.875	-3.471	-4.422
	14	-5.999	-4.948	-4.431	-4.765	-1.901	-4.247	-2.856	-1.238
	15	-3.820	-3.564	-5.710	-4.495	-4.293	-4.800	-3.544	-2.727

(a)

Inter. value : S_i		Target bit							
		c7	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	**c0**
Key Byte	**0**	-2.657	-1.135	-4.773	-2.026	1.839	-3.152	-2.785	-3.040
	1	-3.551	-3.092	-5.308	-3.545	-4.046	-3.440	-4.573	-3.064
	2	-1.165	-3.315	-3.206	-1.727	-4.369	-3.377	-3.830	-5.501
	3	-2.087	-3.662	-5.605	-2.356	-5.909	-1.325	-4.297	-2.469
	4	-5.266	-3.040	-4.338	-2.544	-4.298	-1.111	-4.070	-3.664
	5	-1.740	-2.839	-3.956	-2.909	-4.054	-3.255	-4.005	-1.467
	6	-3.906	-1.067	-3.289	-3.988	-1.972	-2.641	-4.936	-3.108
	7	-3.452	-1.583	-5.331	-3.636	-3.154	-3.442	-2.485	-3.149
	8	-5.097	-1.095	-5.635	-3.785	-2.135	-2.964	-2.305	-4.883
	9	-3.894	-2.976	-1.830	-1.718	-4.380	-1.716	-1.657	-1.976
	10	-3.591	-4.195	-1.831	-3.327	2.045	-4.575	-3.080	-0.808
	11	-3.917	-1.417	-5.241	-4.597	-3.387	-0.654	2.507	1.875
	12	-5.347	-1.009	-2.315	-4.363	-3.495	0.572	-4.105	-3.683
	13	-3.500	-1.882	-4.529	-2.645	-3.987	-1.080	-3.017	-3.746
	14	-5.844	-3.067	2.956	-2.841	-3.812	-3.479	-4.857	-4.047
	15	-2.972	-2.822	-5.696	-4.896	-4.241	-4.418	-4.431	-4.626

Inter. value : $\{02\} \cdot S_i$		Target bit							
		c7	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	**c0**
Key Byte	**0**	-1.135	-4.773	-2.026	-1.788	-2.207	-2.785	-3.108	-2.657
	1	-3.092	-5.308	-3.545	-5.273	-4.515	-4.573	-1.866	-3.551
	2	-3.315	-3.206	-1.727	-3.890	-2.055	-3.830	-1.769	-1.165
	3	-3.662	-5.605	-2.356	-2.451	-4.010	-4.297	-2.237	-2.087
	4	-3.040	-4.338	-2.544	-3.783	-3.821	-4.070	-1.197	-5.266
	5	-2.839	-3.956	-2.909	-2.989	-2.617	-4.005	-1.742	-1.740
	6	-1.067	-3.289	-3.988	-4.901	-3.993	-4.936	-4.506	-3.906
	7	-1.583	-5.331	-3.636	-3.066	-2.957	-2.485	-2.838	-3.452
	8	-1.095	-5.635	-3.785	-4.812	-0.697	-2.305	-3.375	-5.097
	9	-2.976	-1.830	-1.718	-4.047	-1.353	-1.657	-2.399	-3.894
	10	-4.195	-1.831	-3.327	-2.691	-6.188	-3.080	-0.991	-3.591
	11	-1.417	-5.241	-4.597	-5.321	-1.696	2.507	-2.448	-3.917
	12	-1.009	-2.315	-4.363	-4.359	-0.378	-4.105	-4.031	-5.347
	13	-1.882	-4.529	-2.645	-4.341	-3.888	-3.017	-1.918	-3.500
	14	-3.067	2.956	-2.841	-2.516	-4.824	-4.857	-1.122	-5.844
	15	-2.822	-5.696	-4.896	-3.341	-3.970	-4.431	-2.949	-2.972

Inter. value : $\{03\} \cdot S_i$		Target bit							
		c7	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	**c0**
Key Byte	**0**	-1.612	-3.718	-4.388	-3.255	-4.511	-3.527	-1.178	-3.108
	1	-1.242	-0.715	-1.864	-2.761	-1.976	-3.467	-3.441	-1.866
	2	-2.801	-3.099	-3.254	-3.698	-3.283	-2.178	-4.113	-1.769
	3	-0.411	-2.111	-1.812	-2.447	-3.285	-4.097	-3.908	-2.237
	4	-2.808	-2.920	-3.926	1.060	-2.855	-3.792	-3.062	-4.489
	5	-4.115	-2.558	-4.983	-3.126	-2.736	-2.987	-3.610	-1.742
	6	-0.756	-4.161	-1.362	-5.412	1.075	-4.294	-1.394	-4.506
	7	0.866	-3.768	-5.518	-2.028	-0.840	-4.873	-4.537	-2.838
	8	-1.780	-4.394	-3.890	-2.762	-3.218	2.788	-1.388	-3.375
	9	-4.098	-1.224	-2.126	-3.496	-3.254	-3.054	-3.847	-2.399
	10	-4.526	-1.624	-1.405	-3.175	-3.571	-2.939	-3.987	-0.991
	11	-4.115	-3.913	-1.176	-2.354	1.207	-4.418	-1.584	-2.448
	12	-1.749	-3.953	-3.215	-0.248	-2.407	-3.080	-0.195	-4.031
	13	-2.628	-2.154	-2.889	-4.283	-3.478	-2.967	-3.342	-1.918
	14	-4.479	-2.853	-3.972	-3.372	-1.354	-2.528	-3.307	-1.122
	15	-4.421	-0.999	-4.663	-3.793	-4.148	-2.868	-4.753	-2.949

(b)

Fig. 6: This shows DPA attack results on measured power consumption during WB-AES computation operated in Chipwhisperer-lite with two type of chosen plaintext. (a) is yield from odd plaintext measurement and (b) is result of even plaintext acquisition. The even and odd plaintext consists of only odd and even value per byte, respectively.