Vulnerability of Dynamic Masking in Test Compression

Yogendra Sao, Debanka Giri, Soham Saha, and Sk Subidh Ali(✉)

Indian Institute of Technology Bhilai, Durg 491001, India
{yogendras,debankagiri,sohamsaha,subidh}@iitbhilai.ac.in

Abstract. Scan-based Design for Testability (DfT) ensures the testability of chips while providing observability and high fault coverage. In the case of security-critical applications, an attacker can misuse the scan-based DfT of a chip as a backdoor and reveal the secret information embedded inside the chip. Even advanced test infrastructures such as X-compactor and X-masking are vulnerable to such an attack. In this work, we perform a detailed security analysis of one of the DfT techniques known as the Embedded Deterministic Test (EDT), which is used in the test compression tool Tessent TestKompress. EDT uses dynamic masking along with an XOR-based compactor to achieve test compression. The existing state-of-the-art attack is shown to be effective against dynamic masking. However, the attack success rate is highly constrained by the scan chain configuration, with a 20.53% success rate in the worst-case scenario. In this paper, we propose an improved attack by leveraging signature analysis. The advantage of our attack is that it's a deterministic attack on dynamic masking, which can retrieve the secret key with a 100% success rate. The attack is independent of the internal scan infrastructure and can work even in the presence of a compactor.

Keywords: AES · Security · Scan Attack · Design for Testability · Scan Chain · Static Masking · Dynamic Masking · Compactor · XOR Compression

1 Introduction

Scan-based DfT is a popular technology that is associated with the field of circuit testing, for examining manufacturing-related defects, providing high testability and high fault coverage. In the test mode, the internal flip-flops of the chip are converted into fully accessible scan cells and connected to a scan chain. An attacker can exploit it to get the intermediate response of a cipher to reveal the secret key of the cipher embedded inside the crypto chip [18]. The attack by which an attacker bypasses the weakness of these scan infrastructures is known as a scan attack.

The traditional scan attacks [18,19] rely on mode switching. Therefore, a mode-reset countermeasure [7] was proposed to prevent scan-based attacks by

F. Regazzoni et al. (Eds.): SPACE 2023, LNCS 14412, pp. 101–116, 2024.
https://doi.org/10.1007/978-3-031-51583-5_7

flushing the data in the round register of Chip-Under-Test (CUT) while switching from normal mode to the test mode. Later on, test-mode-only attack [1] was proposed, which is performed using only the test mode without mode switching. Another kind of countermeasure was proposed in [19] by introducing a mirror key register (MKR), which is used in test mode at the time of testing and contains a dummy key value. However, it does not support online testing.

VIm-Scan [9] authenticates the testing process using a pattern matching through the first M consecutive test vectors each of N bits. The security of VIm-Scan can be increased by increasing the value of M and N at the cost of area overhead. There are other secure and interesting countermeasures based on obfuscation of scan data [2,3] and encryption of scan data [17]. While encryption-based countermeasures are secure, they incur a huge area overhead, whereas the obfuscation techniques with low area overhead are not proven to be secure.

Most of the testing time is consumed for shifting in and out the scan data, to speed up the testing, multiple scan chains are introduced. Furthermore, to reduce the test data, multiple scan chains with decompressor and compactor are used in advanced DfT infrastructure to achieve time and space compaction for reducing test time and cost. Moreover, the compaction is often combined with additional logic, such as X-tolerance, X-masking, etc., to remove the effect of some unknown states (X-states or don't care) from the compacted output.

These advanced DfT structures were considered secure against scan-based attacks [8]. Later on, advanced attacks are shown against these advanced DfT structures [4–6,13,14,16]. The signature-based attack was proposed in [4,13] against X-masking. In [5,6], a Hamming weight-based attack is proposed on different test compression techniques, such as X-tolerance, static masking, and dynamic masking, used in commercial EDA tools provided by popular EDA vendors: Synopsys, Cadence, and Siemens. The attacks proposed in [5,6] are probabilistic, having a lower success rate. Thus, a deterministic attack on static masking has been proposed recently in [16], which is successful whenever at least 6 bits corresponding to each AES word are observable. However, a deterministic attack on dynamic masking is still unexplored.

Embedded deterministic test (EDT) [12] is a widely used advanced DfT technique based on dynamic masking offered by Siemens. An in-depth security analysis of EDT composed of dynamic masking was done in [5], and they suggested the designer to have a lower number of active slices to provide security to their chips. We refine the security analysis of EDT by proposing an improved attack on dynamic masking, which is successful even when at least one of the scan chains is unmasked. Our contributions to this paper are:

1. We perform a security analysis of the Embedded Deterministic Test employed with the dynamic masking.
2. We propose a state-of-the-art attack on dynamic masking with compaction having a 100% success rate.
3. We perform an analysis of our attack against different levels of masking and compaction for different scan architectures to validate our results.

2 Background

2.1 AES

Advanced Encryption Standard (AES) is a symmetric key encryption algorithm, which is available in different key sizes, AES-128, AES-192, and AES-256 having 128, 192, and 256 bit key, respectively. Figure 1 shows the AES operations with n rounds, where n can be 10, 12, 14. Each round is composed of the following four operations, except the last round, which does not have the *MixColumns* operation:

1. *SubBytes:* This is the only non-linear substitution procedure where each byte of the 4×4 state matrix is replaced using an 8-bit substitution box.
2. *ShiftRows:* In this step, each row of the state matrix is shifted to the left where shift operation involves shifting 0, 1, 2, and 3 number of bytes in respect of 4 rows of the input matrix.
3. *MixColumns:* This is a mixing operation, where the state matrix is multiplied with a 4×4 constant matrix.
4. *AddRoundKey:* This is the XOR operation between the state matrix and the round key.

There is a key whitening phase before the above four operations, where the input plaintext is XORed with the AES key.

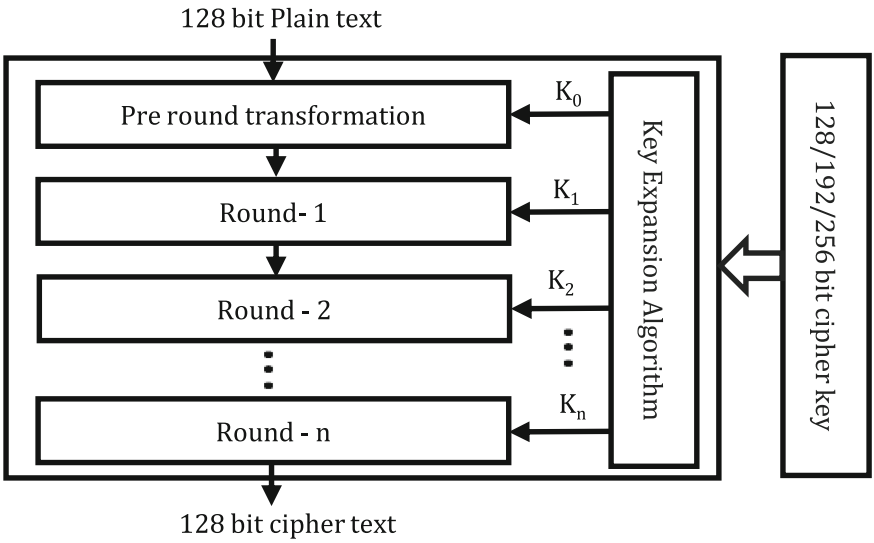

Fig. 1. AES Block Diagram.

2.2 Scan Based DfT

In scan-based DfT, the internal flip-flops of the CUT are converted into fully accessible scan cells (Scan Flip-flops (SFFs) in Fig. 2) and connected in scan chains which can be treated as a configurable shift register. The scan chain infrastructure shown in Fig. 2 has a Test Control (TC) pin connected to a MUX used to control the mode of CUT. The scan in (SI) and Scan out (SO) pins are used to shift in test vectors and shift out the captured responses to and from the scan chain, respectively. As shown in the figure, depending on the test control input line, the input to a scan cell can either be from the round function (in normal mode) or from the previous scan cell (test mode). Scan enables greater access to the chip's internal logic, leading to high test coverage.

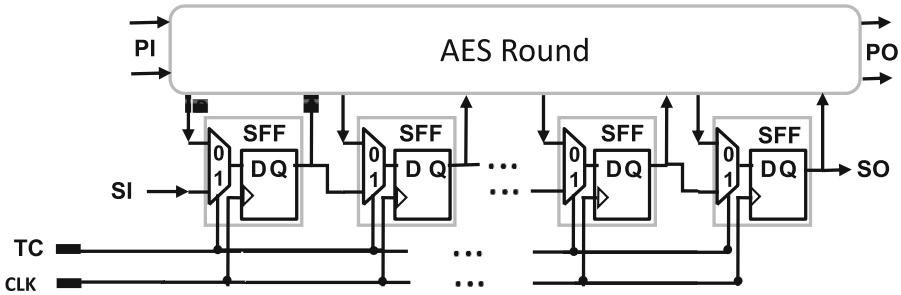

Fig. 2. Scan Architecture.

2.3 Advanced DfT Infrastructure

In the scan chain, there may be some flip-flops holding some unpredictable values, such as the previous state, unknown values of buses, etc. These unpredictable values are called X-States or unknown states. There are two methods available to handle X-states: X-masking and X-tolerance.

X-Masking. In X-masking, a mask is added to the CUT to filter the X-states. Masked CUT includes a mask decoder and a mask input. The mask is achieved by adding the desired number of AND gates to the scan chain, and these AND gates are connected to a mask decoder as shown in Fig. 3. The mask can be either static or dynamic.

– *Static masking:* The static masking always generates a fixed mask value for the entire test with the help of a mask decoder. In order to generate a static mask for a scan chain, the input vector supplied to the mask decoder remains static for the entire test.

– *Dynamic masking:* Dynamic masking always generates different mask values at each clock cycle throughout the test process with the help of a mask decoder. In order to generate a dynamic mask for a scan chain, the input vector supplied to the mask decoder is changed every time for the entire test.

Compaction. Compaction is used to compress the output of the scan chain. In order to achieve the compaction scheme, the output of multiple scan chains is XORed with each other to produce a single output.

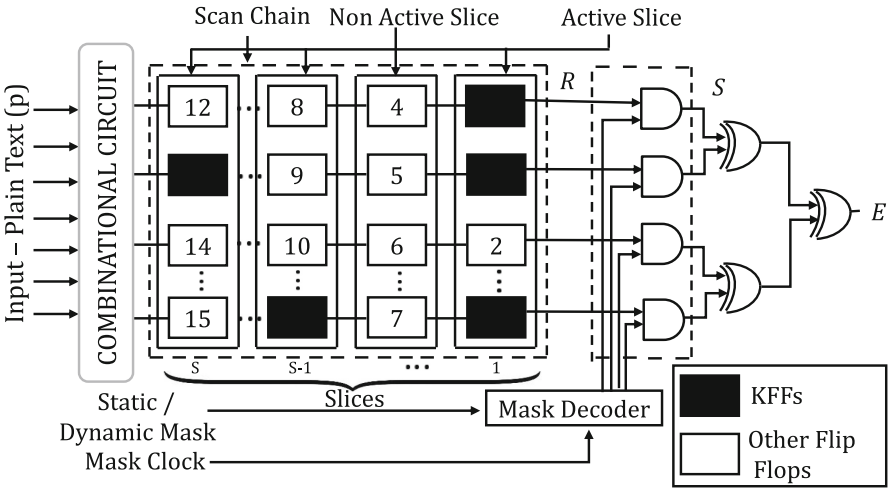

Fig. 3. X-Masking along with the Compactor: P is input plaintext, R is the round output, S is the masked round output, E is the compacted test output, KFFs are the Key Flip Flop containing information related to the target key byte, the number of active scan chains is 3, and the number of active slices is 3, where active scan chain and active slice contains at least one KFF.

3 Proposed Attack Principle

A typical X-masking scheme is shown in Fig. 3. There can be multiple scan chains (scan cells in rows) and multiple slices (scan cells in columns). A flip-flop whose value depends on the targeted key byte (i.e., flip-flops of the round register containing differential information) is known as a Key Flip-flop (KFF) [5,6], and there can be at most 32 KFFs at a time in the scan chain involved in the differential analysis of AES one-round response for a one-byte input difference. A scan chain or a slice with at least one KFF is known as an active scan chain or active slice. There are 3 active scan chains and 3 active slices in Fig. 3. There can be numerous combinations of active scan chains and active slices. To

explain our attack analysis and for the sake of simplicity, we consider multiple scan architecture as shown in Table 1 with each of the active slices or active scan chains completely filled with KFFs. Let us consider one of the scenarios of these scan architectures, where all 32 significant bits of the round register (an AES word) are in a single slice with a 32 number of active scan chains. In this case, 32 bits of the round response will be shifted out in just one shift cycle and will be masked with a 32-bit mask value. The mask value will determine which scan chain will be blocked and which one will pass. In the case of static masking, if a scan chain is blocked due to masking, it will remain blocked throughout the test process as the mask value is static. Whereas in dynamic masking, the value of the mask depends on the test input and can change dynamically at each clock cycle (or at regular intervals), blocking different scan chains in each shift cycle.

The test response corresponding to a test input may be distributed over multiple slices containing don't care (X-states). Dynamic masking provides a more sophisticated way of masking by dynamically changing the mask for each slice to block only don't care bits. However, the same sequence of mask patterns will be applied for the same test response if the same test input is applied again. Therefore, a repetitive application of the same test input vector will fix the effective mask for a test response, which can be considered a special case of static masking, where all of the scan cells are placed in a single slice. Now, by varying the first byte of the plaintext for all possible 256 values, a partial one-round response can be observed, and a differential attack can be launched on this partial information as follows:

1. A sequence of plaintext pairs is formed with a varying difference ranging from 0 to 255 in their first byte. Here one plaintext P' is kept fixed, such that the first byte of the plaintext P' is set to 0, the other plaintext P is varied by varying its first byte from 0 to 255 to form a sequence $(0,0)$, $(1,0)$, $(2,0)$..., $(255,0)$, then differences in their outputs are arranged in the same sequence.
2. Any one of the scan cells is identified for which a difference is observed. Note that there can be only a maximum of 32 such scan cells (bit positions) on which a difference is observed. In the case of masking with compaction, the difference can be observed in the compacted output.
3. The sequence of 1-bit differences in the identified bit position in step (2) can be used as the signature of CUT, which needs to be matched in the signature table(s) containing 256×32 signatures which increases up to 256×255 (generated in Sect. 4) in the presence of compaction, to retrieve the first byte of the key.
4. Similarly, the input difference is applied in the rest of the 15 bytes of the plaintext in step (1), and by repeating step (1) to (3), the other 15 bytes of the key can be retrieved.

4 Attack on Dynamic Masking with Compaction

We consider a CUT with dynamic masking with compaction as shown in Fig. 3 . The detailed attack procedure based on the attack principle (Sect. 3) is explained

in this section. Before proceeding to the attack part, we consider the following assumptions similar to [6]:

1. Dynamic masking scheme is applied on an iterative implementation of AES-128 similar to Fig. 3.
2. The attacker has full control over the SI and TC pin, can load the scan chains with any test vector through the SI line and can apply any desired plaintext through the chip's primary inputs (PI).
3. As the value of the mask depends on the test inputs, the attacker can load the same test vector multiple times to apparently fix the mask value for multiple test responses.
4. The scan chain includes the complete 128-bit round register, out of these 128 scan cells, there will be 32 KFFs based on the targeted key byte.

4.1 Basics of Signature-Based Attack

A basic signature attack is a two-phase attack. In the first phase, a signature table is generated based on some observable output difference. For example, one can observe only n-bits of the output difference. In the best case, complete output difference may be observable, which can uniquely identify the key. In the case of partial information, a single output difference may not be sufficient. Therefore, we can apply 256 plaintext pairs with a varying difference in their first byte similar to Sect. 3 and observe 256 partial output differences. These 256 output differences form a signature. Now, in the signature table generation phase, we'll set a key byte value, and apply 256 possible plaintext pairs to observe the output differences to generate one of the signatures. Similarly, we'll generate the entire table for all 256 values of a key byte. In the second phase, we'll apply the same set of 256 plaintext pairs to the CUT and observe the output differences as a signature. Then, we can match this signature in the signature table to get a key byte. This attack works even in the worst case when only 1 bit of the output difference is observable. This makes it suitable against partial scan designs, where the exact bit positions of the round register in the scan cell are unknown.

The proposed attack uses the above signature-based attack, in two phases (online and offline phase). The online phase is the only phase that requires CUT to collect test responses corresponding to 256 desired plaintexts applied to the CUT by varying only one byte of the plaintext. Whereas, offline phase is used to create a signature from CUT responses and match it in the signature table(s) for key recovery. In the rest of the paper, we show how to recover the first byte of the AES key by varying the first byte of plaintexts in the online phase. Similarly, the other 15 bytes of the key can be recovered by targeting the remaining 15 bytes of the plaintexts in the online phase.

4.2 Online Phase: Collecting *CUT* Responses

The main challenge in dynamic masking is to identify the unmasked bits from CUT responses. If the mask values are changed for each of the CUT responses,

it is very difficult to identify the unmasked bits used in the key recovery process. Therefore, a test vector is kept fixed in CUT to fix the mask value for multiple CUT responses as discussed in Sect. 3.

Algorithm 1 shows the sequence of steps followed in the online phase. Initially, 256 different plaintexts are created from a random plaintext P by varying only its first byte for all 256 possible values ($P_i = P$, where $P_i^0 = i$ and $0 \leq i \leq 255$). These 256 plaintexts are used as the inputs for the Algorithm 1. Before that, a test vector is chosen randomly and stored in a variable TV at the start of the Algorithm 1, and is kept fixed as long as the algorithm runs. In each iteration, the chip is first switched to the test mode ($TC = 1$), and the above-chosen test vector (TV) is shifted into the scan chain (SC) as shown in steps 1 and 1. Then, the chip is switched to normal mode ($TC = 0$), and one of the plaintexts (P_i) is applied through primary input pins PI. The CUT is run for one round of AES to capture its one round masked and compacted response in the scan chain (SC) as shown in steps 1 to 1. $ENC\_CUT()$ in step 1 shows the one-round encryption operation performed by CUT on a given plaintext, where the key used for encryption is the embedded key of the CUT. Finally, the chip is switched to the test mode, and CUT response loaded in the scan chain (SC) is shifted out and stored in a variable E_i corresponding to a plaintext P_i, which is shown in steps 1 and 1. The above steps are repeated for 256 plaintexts in the loop, and at the completion of the loop, 256 CUT responses $E_0, E_1, \ldots, E_{255}$ for each of the 256 plaintexts are obtained as the output of the Algorithm 1, which becomes inputs for the offline analysis. Figure 4 shows how a CUT response is obtained by applying a plaintext with a fixed test vector when CUT is run for one round of AES.

Algorithm 1: Online Phase

Input : $P_i (0 \leq i \leq 255)$, P_i is the plaintext applied at CUT, where $P_i^j \leftarrow i$ and considering the default value of $j = 0$.

Output: $E_i (0 \leq i \leq 255)$, where E_i is the one round response from CUT with respect to Plaintext $P_i (0 \leq i \leq 255)$

$TV \leftarrow Random()$

for $i \leftarrow 0$ **to** 255 **do**

- $TC \leftarrow 1$ /*Switching to the test mode*/
- $SC \leftarrow TV$ /*Consecutive shift cycles to load the test vector*/
- $TC \leftarrow 0$ /*Switching to the normal mode*/
- $PI \leftarrow P_i$ /*Apply plaintext through Primary Input*/
- $SC \leftarrow ENC\_CUT(PI)$ /*Capture one round response*/
- $TC \leftarrow 1$ /*Switching to the test mode*/
- $E_i \leftarrow SC$ /*Consecutive shift cycles to Shift out CUT response */

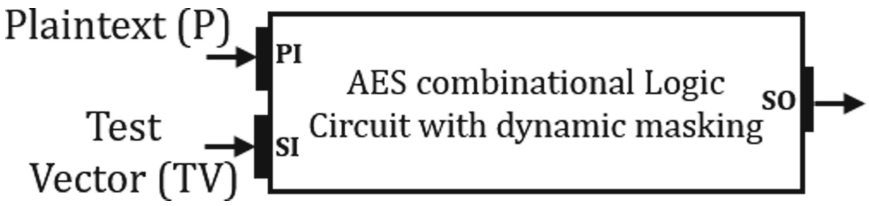

Fig. 4. Online Procedure on CUT

4.3 Offline Phase

The offline phase consists of three steps. In the first step, the required signature tables are built by simulating the first round of AES with predetermined 256 plaintexts, as mentioned in the online phase. In the second step, the same sets of plaintext are applied to the CUT, and the corresponding signature is observed. In the third step, the key byte is recovered by matching the CUT signature with the signature tables.

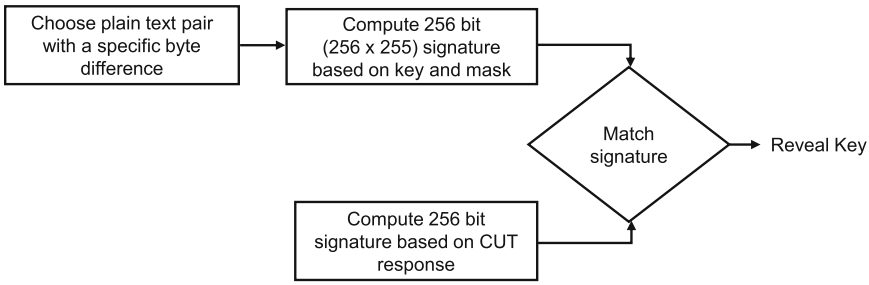

Fig. 5. Offline Procedure

Signature Table Generation. The first step in the offline phase is to build signature tables, where the row of the table will have a unique signature corresponding to each possible key byte value. Let us assume that the signature is corresponding to only one mask, whose value is m.

If we consider the dynamic masking without compaction, anyone unmasked bit from CUT response will be sufficient to reveal the key by matching at most 256×32 signatures. However, if the masked response is further compacted using XOR-tree as shown in Fig. 3, then only the parity of round output will be observable. Thus, the signature obtained using 1-bit parity of CUT responses may not match in 32 signature tables corresponding to each of the individual bit positions of an AES word, as the signature created using 1-bit parity is a combination of 32 signatures from 32 different signature tables corresponding to individual bit

positions. Since there can be a maximum of 32 KFFs in the scan chain, which can be masked with 2^{32} different values of mask. Hypothetically, there can be 2^{32} possible signature tables corresponding to 2^{32} mask values for a fixed value of the key. However, experimentally, it is found that there are repetitions of signatures for 2^{32} different masks, and only 2^8 masks applied to the first byte of round output is sufficient to generate 256 unique signatures, where one of the signatures is for mask 0, where no output is observed and can not be used to recover a key. So, using 255 masks (except mask 0), 255 different signature tables can be generated, each having 256 signatures corresponding to 2^8 possible values of the key, with a total of 256×255 unique signatures. These signatures can be matched with the signature created from CUT responses to get the key.

Now, the signature table is generated using Algorithm 2. Initially, all 16 bytes of the mask M are set to zero. Then, the first loop is run for all 2^8 possible values of the first byte K^0 of the key K while keeping other bytes of the key constant. The second loop is for mask M, where the first byte M^0 of mask M is varied from 1 to 255 running the loop 255 times. The third loop is for plaintexts, where 256 different plaintexts are created by varying their first byte P_i^0 of plaintext P_i, while the other 15 bytes of the plaintexts are kept constant. Using this newly created plaintext along with the key K, AES is run for one round using the function $ENC()$, and the one round output is stored in a variable R, while the second plaintext P' from plaintext pair is chosen as P_0, effectively selecting R_0 as R'. The mask M is applied on both R and R' to get S and S' as masked output, respectively. S and S' are further compressed using a function $PARITY()$ to get their parities, which are stored in E and E', respectively. By performing an XOR operation between 1-bit E and E', one bit of the signature is stored in the signature corresponding to a key byte and a mask value (in the first byte). On completion of the innermost loop, one complete signature is generated, which is of size 256 bits. At the completion of the second loop, 255 signatures corresponding to a fixed value of key byte will be stored in the 255 different signature tables for 255 values of mask M. And, finally, at the end of the first loop, 256×255 signatures are generated for 255 signature tables, where one of the signatures represented as $SIG\_TAB_{k,m}$ is the signature for a key byte value k, and a mask value m in the first byte of mask M keeping other bytes of M as zero.

Signature Generation. The second part of the offline phase is to create a signature using CUT responses. To generate a signature from CUT responses in the offline phase, we can directly create a signature using 1-bit CUT responses, as shown in Algorithm 3. The obtained signature can be stored in the variable SIG_CUT, which is matched in the signature table to reveal the key.

Key Recovery. Now, SIG_CUT can be matched with 256×255 signatures of 255 signature tables. If a signature $SIG\_TAB_{k,m}$ matches with SIG_CUT, then the first key byte can be recovered as k, as shown in Algorithm 4. Similarly, other 15 bytes of the key can be obtained by targeting other bytes of the plaintext in the online phase. Note that the signature tables generated by targeting the first

byte of the key and plaintext are sufficient and reusable and can reveal any of the key bytes using the signature from CUT responses, targeted at any byte of the plaintext in the online phase. The flowchart for the offline phase is shown in Fig. 5, where 256×255 signatures will be generated corresponding to a key byte and mask value.

A complete picture of the attack with an example is shown in Fig. 6, where a signature from CUT responses is matched in the signature table, matching it with one of the signatures corresponding to a key byte $k = 255$.

Algorithm 2: Signature Table Generation

Output: SIG_TAB as a Signature Table of size 256×255
$M^n \leftarrow 0, \forall n \in]0, 15[$ /*M^n is the n^{th} byte of M*/
for $k \leftarrow 0$ **to** 255 **do**
  $K^0 \leftarrow k$ /*K^j is the j^{th} byte of K*/
  for $m \leftarrow 1$ **to** 255 **do**
    $M^0 \leftarrow m$
    for $i \leftarrow 0$ **to** 255 **do**
      $P_i^0 \leftarrow i$
      $R_i \leftarrow ENC(K, P_i)$
      $S \leftarrow R_i \ \& \ M$
      $S' \leftarrow R_0 \ \& \ M$
      $E \leftarrow PARITY(S)$
      $E' \leftarrow PARITY(S')$
      $SIG\_TAB^i_{k,m} \leftarrow E \oplus E'$

Algorithm 3: Generating Signature from compacted CUT Response

Input : $E_i (0 \leq i \leq 255)$, where E_i is the response of CUT with respect to Plaintext $P_i (0 \leq i \leq 255)$;
Output: SIG_CUT
for $i \leftarrow 0$ **to** 255 **do**
  $SIG\_CUT^i \leftarrow (E_i \oplus E_0)$

Algorithm 4: Match Signature of *CUT* in Signature Table

Input : SIG_CUT, SIG_TAB
Output: KEY_BYTE

for $k \leftarrow 0$ **to** 255 **do**
  for $m \leftarrow 1$ **to** 255 **do**
    if $(SIG\_CUT = SIG\_TAB_{k,m})$ **then**
      $KEY\_BYTE \leftarrow k$

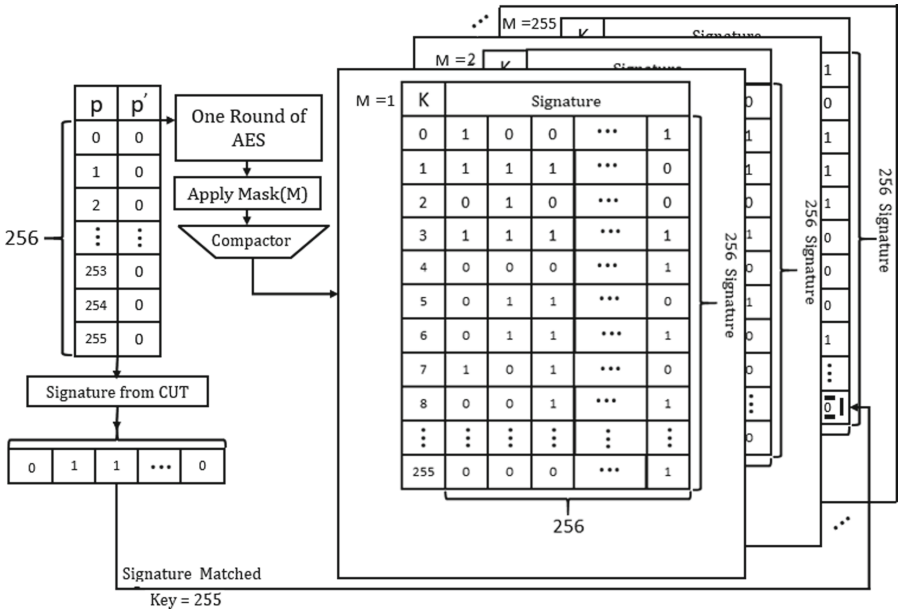

Fig. 6. Attack on dynamic masking with compaction: *CUT* signature is matched in the signature table, and the recovered key is 255.

5 Security Analysis of Embedded Deterministic Test (EDT)

Dynamic masking with an XOR-based compaction is used in Embedded Deterministic Test (EDT) [10–12], as shown in Fig. 3. Mentor Graphics test compression tool Tessent TestKompress uses EDT [5], where XOR-tree is used for space compaction. The value of the mask depends on the test inputs and can vary frequently as per the mask clock. In the worst case, it can vary at each shift cycle, masking differently for each of the slices.

The main reason for the successful attack is the input-dependent mask, which can be fixed by applying the same test vector multiple times. The proposed attack can be thwarted if the mask values are generated randomly and cannot be controlled by the attacker. Thus, the designer should not depend completely on

the EDT logic of dynamic masking and should adopt additional countermeasures to provide security against scan-based attacks. One simple solution is to block primary inputs in test mode; in this case, the attacker would not be able to apply the desired plaintexts through the primary inputs and must apply the plaintexts through the scan input pin SI as a test vector. For differential analysis, it has to apply different plaintexts, therefore changing the mask values. In this case, the proposed attack could be thwarted. To block the primary inputs in test mode, the 128 primary input lines can be controlled using 128 AND gates, where the other input of the AND gate is the output of a NOT gate fed by TC (Test Control) line in 2. In normal mode ($TC = 0$), the one input of AND gates will be 1, and the primary input will pass through it and will not affect the chip's working in normal mode. In the test mode ($TC = 1$), one input of AND gates will be 0, restricting primary inputs in the test mode. It will need only 128 AND gates and one NOT gate, with a total 129 number of additional gates requirement. The other solution could be to link the test vector and the plaintext (from primary input) for mask generation. In this case, the input test vector can be concatenated with the inputs from the primary inputs for mask generation. For a different plaintext, different masks will be generated, and the proposed attack can be thwarted. To implement this, no additional gate will be required for concatenation. However, it may require minor changes in mask generation logic.

6 Results and Comparison

We simulated our proposed attacks on dynamic masking with compaction with the help of C programming language on a system having the configuration of Intel(R) Core(TM) i5-8250U CPU @ 1.60 8 core, 8 GB RAM, loaded with Ubuntu 20.04.2 LTS operating system. The attack was launched for 6 different combinations of active scan chains and active slices as shown in Table 1. We tried to simulate the CUT for the online phase similar to [5], we implemented one round of AES in C. The mask pattern for each of the slices was chosen randomly using the pseudo-random function and is kept fixed assuming a fixed test vector is applied while applying each of the 256 plaintexts. This mask is applied on AES one-round output using bitwise AND operation. The masked output is further compacted with different compression ratios using bitwise XOR operation. For the sake of simplicity, we consider scan chains consisting of only KFFs. One round of compacted masked outputs is collected, corresponding to 256 plaintexts. Then, a signature SIG_CUT was created and matched in the signature table using the methods proposed in Sect. 4, and the first byte of the AES key was recovered correctly. Other 15 bytes of the key were recovered by targeting other bytes of the plaintexts in the online phase.

The attack result is shown in Table 1. Although Fig. 3 shows the worst-case scenario of the compaction, where only parity of the masked output is observable. As the proposed attack requires only a one-bit compacted output of a partial one-round response, in the case of multiple bits after compaction, any

one of the bits can be targeted to generate a signature from CUT responses. Different compression ratios can be achieved using multiple slices or by XORing a group of scan chains. This attack is equally applicable in both of the cases, as both the cases are equivalent, either mask some of the KFFs before compaction or ignore compacted outputs from those KFFs. Therefore, the attack results for different compression ratios for different combinations of active scan chains and active slices have a 100% success rate, as shown in Table 1. However, the existing state-of-the-art attack [5,6], where a detailed security analysis of EDT was also performed, is probabilistic in nature and has a worst-case success rate of 20.53% for 32 active scan chains and 16 active slices. The reason behind the low success rate is that the Hamming weight-based attack [5] is not suitable for a partial scan. Suppose a unique Hamming weight 9 is targeted, and one of the bits participating in the Hamming weight calculation is masked. Then, after masking, the resultant Hamming weight will be reduced by one producing Hamming weight 8 [16]. Therefore, the actual plaintext pairs may shift to lower Hamming weights. Since the actual plaintext pair for Hamming weight 9 is shifted to 8, an incorrect plaintext pair from some other Hamming weight may produce Hamming weight 9, and a wrong key will be recovered with an attack failure. Similarly, the distortion of hamming weight distribution due to compaction can be seen in [15]. Therefore, a basic scan attack using Hamming weights on advanced DfT structures [5,6] is probabilistic in nature and has a low success rate for partial scan. Whereas the proposed attack is signature-based and outperforms the partial scan. To recover all 16 bytes of the key, our proposed attack took only 4096 plaintexts and 255 signature tables containing 256×255 signatures with space complexity of $256 \times 255 \times 256 \approx 2^{24}$ bits (using Algorithm 2) to recover all 16 bytes of the key. The time taken for signature table generation was only 94 s. For mask generation, Algorithm 2 needs around $16 \times 256 \times 255 \times 256 \times 7 \approx 2^4 \times 2^8 \times 2^8 \times 2^8 \times 2^3 = 2^{31}$ number of operations.

Table 1. Success for attack on Dynamic Masking with Compaction

Sl. No.	#Active Slices	#Active Scan Chains	Success rate for different compression ratios					
			1:1	2:1	4:1	8:1	16:1	32:1
1	1	32	100%	100%	100%	100%	100%	100%
2	2	16	100%	100%	100%	100%	100%	
3	4	8	100%	100%	100%	100%		
4	8	4	100%	100%	100%			
5	16	2	100%	100%				
6	32	1	100%					

7 Conclusion

In this paper, an attack on dynamic masking with compaction is proposed. Hypothetically, it requires 2^{32} signature tables corresponding to 2^{32} different values

of mask. We have experimentally shown that only $2^8 - 1$ masks are sufficient to generate 255 unique signature tables, with an overall requirement of 256×255 signatures. The results show that the attack has a 100% success rate for any combination of active scan chains and active slices, thus making it independent of the internal structure of the scan chain. At the same time, the existing state-of-the-art attack has a 20.53% success rate in its worst case. Based on the attack, we provided a security analysis of EDT employed with dynamic masking, which shows the vulnerability of EDT offered by Siemens against scan-based attacks. The main vulnerability is its input-dependent mask, which can be controlled by an end user by applying a fixed test vector. To thwart the proposed attack, we propose to block the primary inputs in the test mode.

References

1. Ali, S.S., Saeed, S.M., Sinanoglu, O., Karri, R.: Novel test-mode-only scan attack and countermeasure for compression-based scan architectures. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **34**(5), 808–821 (2015)
2. Cui, A., Li, M., Qu, G., Li, H.: A guaranteed secure scan design based on test data obfuscation by cryptographic hash. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **39**, 4524–4536 (2020)
3. Cui, A., Luo, Y., Chang, C.H.: Static and dynamic obfuscations of scan data against scan-based side-channel attacks. IEEE Trans. Inf. Forensics Secur. **12**(2), 363–376 (2017)
4. DaRolt, J., Natale, G.D., Flottes, M.L., Rouzeyre, B.: Are advanced DfT structures sufficient for preventing scan-attacks? In: VTS, pp. 246–251. IEEE (2012)
5. Das, A., Ege, B., Ghosh, S., Batina, L., Verbauwhede, I.: Security analysis of industrial test compression schemes. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(12), 1966–1977 (2013)
6. Ege, B., Das, A., Ghosh, S., Verbauwhede, I.: Differential scan attack on AES with X-tolerant and X-masked test response compactor. In: DSD, pp. 545–552. IEEE (2012)
7. Hely, D., Bancel, F., Flottes, M.L., Rouzeyre, B.: Test control for secure scan designs. In: ETS, pp. 190–195 (2005)
8. Liu, C., Huang, Y.: Effects of embedded decompression and compaction architectures on side-channel attack resistance. In: VTS, pp. 461–468 (2007)
9. Paul, S., Chakraborty, R.S., Bhunia, S.: VIm-Scan: a low overhead scan design approach for protection of secret key in scan-based secure chips. In: VTS, pp. 455–460. IEEE (2007)
10. Rajski, J., Kassab, M., Mukherjee, N., Tamarapalli, N., Tyszer, J., Qian, J.: Embedded deterministic test for low-cost manufacturing. IEEE Des. Test Comput. **20**(5), 58–66 (2003)
11. Rajski, J., Tyszer, J., Kassab, M., Mukherjee, N.: Embedded deterministic test. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **23**(5), 776–792 (2004)
12. Rajski, J., et al.: Embedded deterministic test for low cost manufacturing test. In: ITC, pp. 301–310. IEEE (2002)
13. Rolt, J.D., Natale, G.D., Flottes, M.L., Rouzeyre, B.: A novel differential scan attack on advanced DFT structures. ACM Trans. Des. Autom. Electr. Syst. (TODAES) **18**(4), 58 (2013)

14. Sao, Y., Ali, S.S.: Security analysis of scan obfuscation techniques. IEEE Trans. Inf. Forensics Secur. **18**, 2842–2855 (2023). https://doi.org/10.1109/TIFS.2023.3265815
15. Sao, Y., Ali, S.S., Ray, D., Singh, S., Biswas, S.: Co-relation scan attack analysis (COSAA) on AES: a comprehensive approach. Microelectron. Reliab. **123**, 114216 (2021)
16. Sao, Y., Pandian, K.S., Ali, S.S.: Revisiting the security of static masking and compaction: discovering new vulnerability and improved scan attack on AES. In: 2020 (AsianHOST), pp. 1–6. IEEE (2020)
17. Vaghani, D., Ahlawat, S., Tudu, J., Fujita, M., Singh, V.: On securing scan design through test vector encryption. In: ISCAS, pp. 1–5. IEEE (2018)
18. Yang, B., Wu, K., Karri, R.: Scan based side channel attack on dedicated hardware implementations of data encryption standard. In: ITC, pp. 339–344. IEEE (2004)
19. Yang, B., Wu, K., Karri, R.: Secure scan: a design-for-test architecture for crypto chips. In: Jr., W.H.J., Martin, G., Kahng, A.B. (eds.) DAC, pp. 135–140. ACM (2005)