Scan Based Side Channel Attack on Data Encryption Standard

Bo Yang Kaijie Wu Ramesh Karri
Electrical and Computer Engineering Department
Polytechnic University, Brooklyn, NY, 11201
yangbo, kaijie@photon.poly.edu, ramesh@india.poly.edu

Abstract

Scan based test is a double edged sword. On one hand, it is a powerful test technique. On the other hand, it is an equally powerful attack tool. In this paper we show that scan chains can be used as a side channel to recover secret keys from a hardware implementation of the Data Encryption Standard (DES).

By loading pairs of known plaintexts with one-bit difference in the normal mode and then scanning out the internal state in the test mode, we first determine the position of all scan elements in the scan chain. Then, based on a systematic analysis of the structure of the non-linear substitution boxes, and using three additional plaintexts we discover the DES secret key. Finally, some assumptions in the attack are discussed.

1. Introduction

Cryptographic algorithms are implemented as application-specific integrated circuits (ASICs) [1] [2] or as cryptographic coprocessors [3] [4] to meet high throughput requirements. Scan-based tests are used to validate the function of a hardware system at fabrication time and in field. Compared to built-in self test (BIST), scan-based tests provide high fault coverage and do not need hardware for test pattern generation and signature analysis [5].

A scan-based test constructs one or more scan chains in a chip by tying together some internal registers and flip flops and connecting them to the five-pin serial JTAG boundary scan interface [6]. TCK is the test clock signal while TMS selects normal mode or test mode. TRST is the reset signal for test controller. During testing, test vectors can be scanned in via the TDI pin and internal registers can be scanned out via TDO pin.

During test synthesis, a D flip-flop is replaced with its equivalent scan D flip-flop when it is included in a scan chain. As shown in Figure 1, a scan D flip-flop is a D flip-flop with a MUX at the D input. In normal mode (the mode signal is set to 0) it works like a D flip-flop. In test mode (the mode signal is set to 1) its contents can be scanned in and out. All scanned flip-flops are disconnected from the combinational circuit and connected to each other in a scan chain. During test synthesis, scan chains are automatically inserted into the design by the synthesis tool. A scan chain is organized according to the physical positions of the flip flops.

During chip packaging, the scan chains are connected to external JTAG interface pins to provide on-chip debug capability and maintenance in field [7], or left unbound to prevent further access. However, unbound scan chains can still be accessed by breaking the package open.

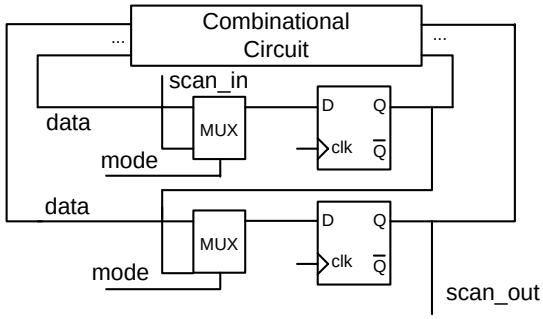

Figure 1 Scan chain structure

In this paper we show that scan chains can be used to discover the secret keys stored in a cryptographic device. Although we focus on DES, the approach is simple yet general and powerful and can be adapted to any cryptographic implementation on ASICs or FPGAs or general microprocessors. We organize the paper as follows. The DES algorithm is presented in section 2. We show how to discover the structure of internal scan chains and how to break the round key of DES using three selected plaintexts in section 3. In section 4, we discuss the complexity of the attack. We extend this attack to the Advanced Encryption Standard (AES) in section 5 and report conclusions in section 6.

2. Data Encryption Standard

2.1. Algorithm

The Data Encryption Standard (DES) is a symmetric encryption algorithm developed in the 1970s by IBM. DES encrypts 64-bit data blocks under the control of a 56-bit user key. DES decryption is the inverse of DES encryption and uses the same user key [8]. The DES encryption is performed in three phases as shown in Figure 2.

Phase 1: The 64-bit plaintext block is bit permuted and stored in two 32-bit registers L (Left) and R (Right).

Phase 2: A round operation composed of function f and exclusive-ored is performed 16 times. In the i^{th} round, the 32-bit R and the 48-bit round key Ki are inputs to the f function. The output of the f function is exclusive-ored with L to form R for the round i+1. The R used in round i becomes the L for round i+1. Function f shaded in Figure 2 performs following operations to generate a 32-bit output.

1. 32-bit value l is expanded into 48-bit value a by E.
2. Value a is exclusive-ored with the 48-bit round key and output value b.

3. Value b is partitioned into 8 groups with 6-bit for each group. Each group is input to an s-box, which substitutes a 6-bit input with a 4-bit output. All the eight 4-bit outputs are combined to form value c.
4. Value c is permuted to generate the 32-bit output of f function, value d by P.

Phase 3: In this phase the two 32-bit outputs of round 16 are concatenated and permuted using the inverse permutation and loaded into the output register.

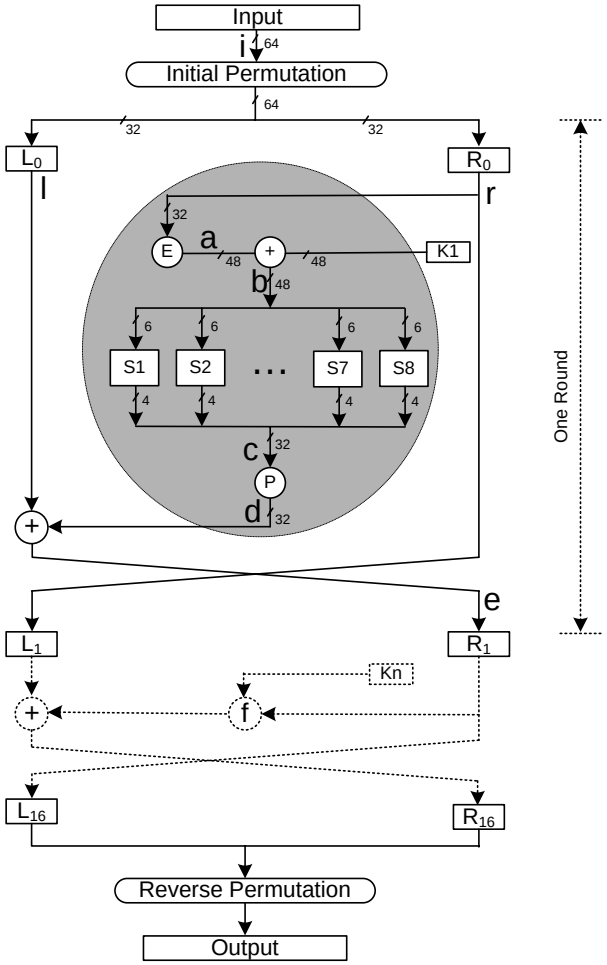

Figure 2 DES encryption algorithm

Round key generation: Since each of the sixteen rounds uses a 48-bit round key, a round key generation algorithm is used to generate the sixteen round keys K1, K2 …K16 from the 56-bit user key. The round key generation uses simple bit-permutation and shift operations. Each round key contains 48 bits of the 56-bit user key.

3. Mounting A Scan Based Attack

3.1. What does the attacker know?

1. We assume that the attacker knows the DES algorithm (it is public).
2. We assume that an attacker has access to high level timing diagrams provided by DES ASIC vendor. Based on the number of clock cycles for an encryption provided by the vendor, the attacker can infer the general structure as either iterative or pipelined implementation [1] [9]. In this paper we focus on the iterative DES encryption data path shown in Figure 3 where a DES round is performed every clock cycle. In the first clock cycle, the inputs are loaded to R and L registers with the permuted plain text. In the remaining 15 clock cycles, the intermediate cipher text is the input to the R and L registers. The output register has the valid cipher text at the end of the 16th clock cycle. Initial and reverse permutations and the E and P functions are all fixed one-to-one mappings. The s-boxes are implemented as either ROMs or combinational gates. The DES controller is a 4-bit counter. The controller generates 4-bit addresses which index into the round key RAM/ROM.

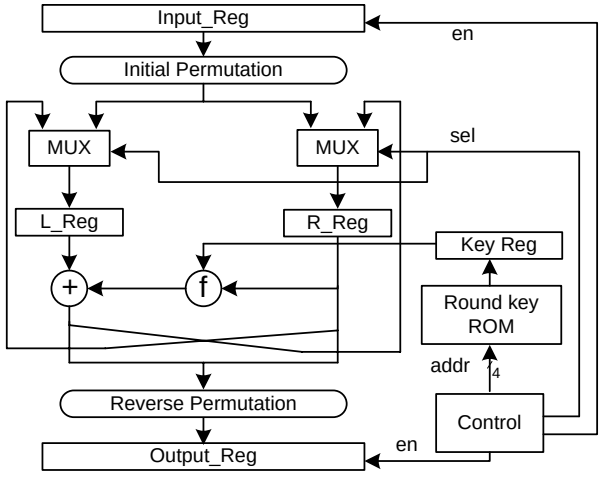

Figure 3 Iterative implementation of DES

3. Although the general implementation structure is known, the attacker does not know the exact number of registers used in the design.
4. It is reasonable that round keys are stored in a secure RAM/ROM.
5. Although it is reasonable that an attacker has direct access to scan chains via the JTAG port or he can break open the package and directly probe the buried JTAG ports, round key registers are not included in the scan chain; otherwise it will be easy to scan out the round key.
6. The attacker does not know the structure of the scan chain. To confirm this we designed the iterative DES encryption data path using Synopsys Design Compiler and inserted a scan chain using Synopsys Test Compiler. The 197 flip-flops (64 from input register, 64 from output register, 64 from L and R register and 4 from controller) in the scan chain shown in Figure 4 are not connected according to

their position in their respective registers. Rather, the Test compiler optimizes the scan chain according to the physical locations of the individual flip flops. For example, in this scan chain flip flop #12 of register L is connected to TDI and flip flop #3 of Input plaintext register is connected to TDO.

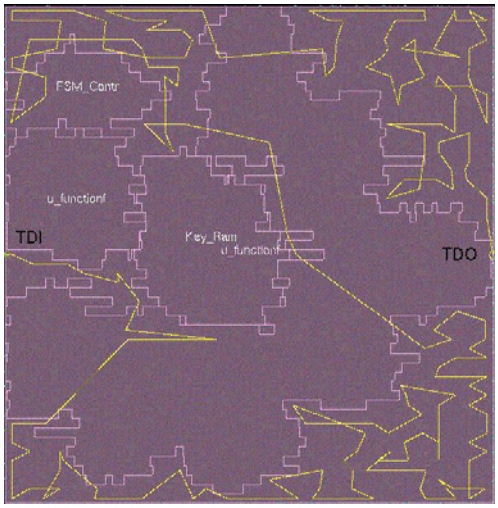

Figure 4 Scan chain structure of the DES data path.

Based on the above reasonable assumptions we mount a two-phase attack. In the first phase, we determine the structure of the scan chain and in the second phase we retrieve DES round keys and the DES user key.

3.2. Attack step 1: Determine scan chain structure

In this step, we will locate the flip-flops in input, L and R registers in the scan chain. The scan out pin TDO yields a serial bit stream that does not immediately reveal the correspondence between the bits in the registers and the bits in the scanned-out bit stream. By switching the DES circuit between normal mode and test mode, we can determine the structure of scan chain as follows.

1. Reset the DES chip and run it in normal mode for one clock cycle to load a known plaintext word into input register.
2. Switch to test mode and scan out the bit stream pattern 1.
3. Switch back to normal mode and run one clock cycle to load the plaintext into L or R registers.
4. Switch to test mode and scan out the bit stream pattern 2.
5. Repeat steps 1 to 3 using a plaintext that is different from the first plaintext in only one-bit position. Save the pattern 3 and pattern 4.

Pattern 1 and pattern 3 must have a one bit difference and that determines the location of an input register flip-flop in the scan chain. Pattern 2 and 4 must have two-bit difference, one of which is from input register and the other is from L or R registers. The location of this flip-flop in the scan chain is determined. By repeating step 5 63 times, we can determine the structure of the scan chain (Locations of all flip flops of input, L and R registers).

3.3. Attack step 2: Recover Round Key 1

Since we know the location of L and R registers in the scan chain, we can break DES algorithm by applying three known plaintexts by analyzing the DES algorithm. Referring to Figure 2, a DES round can be described as follows:

$L_1=R_0$ (1)

$R_1=L_0 \oplus d$; (2)

$d=permutation(c)$; (3)

$a=Expand(r)$; (4)

$b=a \oplus K1$; (5)

$c=$ S_box(b); (6)

Load a plaintext word (L_0 and R_0) and run 3 clock cycles. Switch to test mode and scan out the bit stream. Now L_1 and R_1 are known. Is this information enough to discover round key K_1? From equation (2), d is known ($=L_0 \oplus R_1$). From equation (3), c is known (c = inverse permutation(d)). From equation (4), a is known (expand (R_0)). From equation 5, if we know b, we can find round key K1 (K1 = a⊕b). Since we know a, we need to find b the input to the s-boxes, from their output c.

Each DES round uses eight different s-boxes S1, S2, …,S8. Let us look into the structure of S1 shown in Table 1 as an example. b is 48-bit wide. The 6 most significant bits $b_{47}b_{46}b_{45}b_{44}b_{43}b_{42}$ are inputs to s-box S1. While bits b_{48} and b_{43} select one-of-four rows of s-box S1, bits $b_{47}b_{46}b_{45}b_{44}$ select one-of-sixteen columns of s-box S1. For example, $b_{48}b_{47}b_{46}b_{45}b_{44}b_{43}=(100110)_2$ uniquely identifies the s-box S1 cell with row address 2 and column address 3 yielding 8 shown in Table 1 as the output.

Each s-box compresses the 6-bit input into a 4-bit output. A close look at the eight s-boxes reveals that every output value appears exactly four times. For example, in s-box S1 1 appears in locations (0,3), (1,7), (2,1) and (3,6) as shown shaded in Table 1. If the output of s-box S1 is 1, $b_{48}b_{47}b_{46}b_{45}b_{44}b_{43}$ can be either $(000110)_2$, $(001111)_2$, $(100010)_2$ or $(101101)_2$. Since each output value appears four times, we can not determine the input to an s-box by observing just one output. When we apply a value at point a in Figure 2, some bits in the round key K1 are masked. Based on our analysis of the row and column addresses and values stored in s-boxes we need to apply three values at point a to uniquely determine the round key from the outputs observed at c. Let us analyze S1 to show which values should be applied at point a to recover the six most significant bits of round key K1 from the outputs of S1 at point c.

Table 1: Substitution box S1

Address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

First apply $(000000)_2$ at point a, we have:

$$c^1_{32-29} = S1(K1_{48} K1_{47} K1_{46} K1_{45} K1_{44} K1_{43})$$

We notice that every output appears only once in each row, so we switch one-bit in the column address. Apply $(001000)_2$ at point a and we have:

$$c^2_{32-29} = S1(K1_{48} K1_{47} \overline{K1_{46}} K1_{45} K1_{44} K1_{43})$$

Then we switch two-bit in the input of S1, one in column address and the other in row address. Apply $(010001)_2$ at point a and we have:

$$c^3_{32-29} = S1(K1_{48} \overline{K1_{47}} K1_{46} K1_{45} K1_{44} \overline{K1_{43}})$$

From the three outputs C^1_{32-29}, C^1_{32-29} and C^1_{32-29} we can uniquely determine the value of $K1_{48...43}$.

For example, if c^1_{32-29} is 1, according to Table 1 [8], $K1_{48...43}$ could be $(000110)_2$, $(001111)_2$, $(100010)_2$ or $(101101)_2$. Then we apply $(001000)_2$ at point a and get c^2_{32-29}. If $K1_{48...43}$ is $(000110)_2$ or $(101101)_2$, c^2_{32-29} is 8; if $K1_{48...43}$ is $(001111)_2$, c^2_{32-29} is 4; if $K1_{48...43}$ is $(100010)_2$, c^2_{32-29} is 6. When c^2_{32-29} is 6 or 4, we can decide what $K1_{48...43}$ is. However, if we get 8 as c^2_{32-29}, we still can not determine is $K1_{48...43}$ is $(000110)_2$ or $(101101)_2$. We then apply $(010001)_2$ at point a to distinguish between them. If c^3_{32-29} is 11, $K1_{48...43}$ is (000110). If c^3_{32-29} is 5, $K1_{48...43}$ is (101101).

If c^1_{32-29} is a value other than 1 in the example, we still can disciver $K1_{48...43}$ using this method. S-boxes S2,…, S8 have this feature. We can use the following method to discover round key K_1.

Step1: Apply $(000000000000000000000000000000000000000000000000)_2$ = $(000000000000)_{16}$ to point a. Round key K1 is applied to all s-boxes. Save the output of s-box (c1).

Step2: Apply $(001000001000001000001000001000001000001000001000)_2$ = $(208208208208)_{16}$ to point a. Save the output of s-box (c2).

Step3: Apply $(010001010001010001010001010001010001010001010001)_2$ = $(451451451451)_{16}$ to point a. Save the output of s-box (c3).

Step4: Analyze c1, c2 and c3 to discover K1.

In practice, we can not directly apply these values at point a shown in Figure 2. We will determine plaintext inputs at point i so as to apply the above values at point a by considering the relation between a, r and i in Figure 2.

From the DES expand function specification [8]:

$a_{48...1}=r_{32}r_1r_2r_3r_4r_5r_4r_5r_6r_7r_8r_9r_8r_9r_{10}r_{11}r_{12}r_{13}r_{12}r_{13}r_{14}r_{15}r_{16}r_{17}r_{16}r_{17}$
$r_{18}r_{19}r_{20}r_{21}r_{20}r_{21}r_{22}r_{23}r_{24}r_{25}r_{24}r_{25}r_{26}r_{27}r_{28}r_{29}r_{28}r_{29}r_{30}r_{31}r_{32}r_1$ (7)

From DES initial permutation specification [8]:

$l_{32...1}=i_{58}i_{50}i_{42}i_{34}i_{26}i_{18}i_{10}i_2i_{60}i_{52}i_{44}i_{36}i_{28}i_{20}i_{12}i_4i_{62}i_{54}i_{46}i_{38}i_{30}i_{22}i_{14}i_6$
$i_{64}i_{56}i_{48}i_{40}i_{32}i_{24}i_{16}i_8$ (8)

$r_{32...1}=i_{57}i_{59}i_{41}i_{33}i_{25}i_{17}i_9i_1i_{59}i_{51}i_{43}i_{35}i_{27}i_{19}i_{11}i_3i_{61}i_{53}i_{45}i_{37}i_{29}i_{21}i_{13}i_5i$
$_{63}i_{55}i_{47}i_{39}i_{31}i_{23}i_{15}i_7$ (9)

Replacing r in (7) using i in (9), the value at point i can be derived from the value at point a:

$a_{48...1}= i_{57}i_7i_{15}i_{23}i_{31}i_{39}i_{31}i_{39}i_{47}i_{55}i_{63}i_5i_{63}i_5i_{13}i_{21}i_{29}i_{37}i_{29}i_{37}i_{45}i_{53}i_{61}i_3$
$i_{61}i_3i_{11}i_{19}i_{27}i_{35}i_{27}i_{35}i_{43}i_{51}i_{59}i_1i_{59}i_1i_9i_{17}i_{25}i_{33}i_{25}i_{23}i_{41}i_{59}i_{57}i_7$ (10)

In practice, we can not directly observe the values at point c in Figure 2. They can be calculated from the values at point e. Since e= l⊕d, we set l as all zero and then d=e. From the DES permutation function specification [8]:

$c_{32...1}=d_{12}d_{18}d_6d_{28}d_{26}d_{11}d_{21}d_1d_{14}d_{22}d_4d_{29}d_{30}d_8d_{19}d_{25}d_{32}d_{23}d_{13}d$
$_7d_{27}d_3d_{17}d_9d_{15}d_{31}d_5d_{20}d_2d_{10}d_{16}d_{24}$ (11)

Let us summarize the first three steps in this method as follows:

Step1: Apply $(0000000000000000)_{16}$ at point i. $(00000000)_{16}$ will be propagated to point r; $(000000000000)_{16}$ will be propagated to point a and $(000000000000)_{16}$ will be propagated to point l after the first clock cycle. Scan out the value at point e after the second clock cycle. Calculate c1 using (11).

Step2: Apply $(00002A8000005500)_{16}$ at point i. $(11111111)_{16}$ will be propagated to point r; $(208208208208)_{16}$ will be propagated to point a and $(000000000000)_{16}$ will be propagated to point l after the first clock cycle. Scan out the value at point e after the second clock cycle. Calculate c2 using (11).

Step3: Apply $(0000005500000055)_{16}$ at point i. $(22222222)_{16}$ will be propagated to point r; $(451451451451)_{16}$ will be propagated to point a and $(000000000000)_{16}$ will be propagated to point l after the first clock cycle. Scan out the value at point e after the second clock cycle. Calculate c3 using (11).

3.4. Attack step 3: Recover User Key

As we mentioned in section 2.1, each round key contains 48 bits of the 56-bit user key. How many round keys are needed to discover the user key? Analysis of the DES round key generation algorithm shows that Round key K1 consists of the following 48 bits in the user key:

(45,24,9,32,22,51,8,29,38,44,53,16,39,10,2,1,43,30,31,37, 36,3,52,15,54,4,14,27,12,42,21,6,11,26,55,5,33,25,13,35,4 8,56,19,47,18,34,28,7).

However, it does not contain bits 17, 20, 23, 40, 41, 46, 49 and 50. Bits 17, 20, 23, 40, 41, 49 and 50 can be obtained from round key K2 and bit 46 can be obtained from round key K3. To discover the user key, round keys K2 and K3 have to be recovered as well.

In an iterative architecture, all sixteen rounds are calculated using the same register L and R. Scan $(00000000)_{16}$ into L and $(00000000)_{16}$ into R after the first round as L_1 and R_1; run one cycle and scan out L_2 and R_2. Do the same work using L=0 and R= $(11111111)_{16}$ and then L=0 and R= $(22222222)_{16}$. From these (L_2, R_2) pairs, we can retrieve K2. Similarly, we can get K3.

4. Complexity of the attack

We simulated the attack steps on the iterative DES design discussed in section 2 using modelsim. It took 198 clock cycles (1 clock cycle for normal operation + 197 clock cycles for scan operations) to scan-out the first bit stream. 198 more clock cycles are necessary to locate a flip flop in the input register. This translates into 38214 clock cycles (192×198+198) to determine the structure of the entire scan chain (i.e. to locate all 192 flip flops in input plaintext register and the L and R registers). It takes 397 clock cycles (2 clock cycles for normal operation + 197 clock cycles for scan operation + 1 clock cycle for normal operation + 197 clock cycles for scan operation) for every input plaintext to reach R_0, L_0, R_1 and L_1. This translates into 1191 clock cycles (397×3) to discover round key K_1. Similarly, 1185 clock cycles each are required to discover round keys K_2 and K_3. Overall, 41775 clock cycles (38214+1191+1185×2) are required to discover the user key.

Once the structure of the scan chain is known, this attack requires only three known plaintexts to determine the user key. Compare this with the 2^{64} (plaintext, ciphertext) required to attack DES.

5. Discussion

5.1. Extension to a pipelined DES architecture

In a fully pipelined architecture, one-round unit is instantiated sixteen times. This 16-stage pipeline will have 17 pairs of (L, R) starting from (L_0, R_0) to (L_{16}, R_{16}). L_0 and R_0 can be located first. L_1 and R_1 can be located by observing that L_1= R_0 and R_1=$L_0 \oplus f$ (R_0, K1). If we only change the lowest bit in L_0, L_1 remains unchanged, then the lowest bit in R_1 will switch because f (R_0, K1) remains unchanged. Except for the flip-flops in the input plaintext register and the L_0 and R_0 registers the bit stream will only have one-bit difference. This bit locates $R_1(0)$. In a similar manner we can locate all flip-flops of R_1. Since L_1= R_0, we can switch the bit in R_0 to locate the positions of flip-flops in L_1. Now we can recover round key K1 from L_0, R_0, L_1 and R_1. Similarly round keys K2 and K3 can be retrieved.

5.2. Revisiting some simplifying assumptions

In determining the structure of a scan chain in section 2.1, we assumed that it takes one cycle to load values on the input pins into the input register. For some DES implementations this may take several cycles. Or there is no input register at all and the permuted input is in L and R registers after the first cycle. We describe a simple method to determine the start of encryption based on two important characteristics of crypto algorithms. First, they are data-driven; when we load different plaintexts into the data path the control logic performs the same action independent of the applied input. The flip-flops in the control logic do not contribute to the differences in scanned-out patterns. Second, cryptographic algorithms display the avalanche effect [11]. Due to the avalanche effect a one-bit difference in a round will translate into several bit changes in the next round. This determines start of encryption. The method is summarized as follows:

1. Load a plaintext, run one cycle in normal mode and scan out bit stream. Load same plaintext and run 2, 3,…clock cycles and scan out bit streams.
2. Modify one-bit in plaintext and repeat step 1.
3. Compare bit streams from steps 1 and 2. The clock cycle in which patterns from steps 1 and 2 are widely different is when encryption starts. This also determines the clock cycle when the plaintext is loaded into the input plaintext register and the L, R registers.

The reset operation does not necessarily imply resetting the chip thereby clearing the secret key RAM. If the round keys are stored in ROM, we can physically reset the chip without clearing the round keys. If the round keys are stored in RAM and the chip is in system, we do not use physical reset. Rather, we load the same input and run 16 cycles and the chip will be in a fixed state every time. This state can be the initial state.

6. Conclusions

Several side channel attacks have been proposed including timing analysis [10], differential fault analysis [11] and differential power analysis [12]. In this paper we show that scan chains and scan based tests are a potent side channel. We described a known plaintext attack using only 3 plaintexts to break DES.

References

[1] Helion Technology. Datasheet-High Performance DES and Triple DES core for ASIC, 2003. http://www.heliontech.com/downloads/des_asic_helioncore.pdf

[2] I. Verbauwhede, P. Schaumont and K. Kuo. Design and performance testing of a 2.29 Gb/s Rijndael processor. IEEE Journal of Solid-State Circuits, pp. 569-572, March 2003.

[3] J. Goodman and A.P. Chandrakasan, An Energy-Efficient Reconfigurable Public-Key Cryptography

Processor, IEEE Journal of Solid-State Circuits, pp. 1808-1820, November, 2001.
[4] Corrent. Datasheet-Product Brief of Corrent Packet Armor CR7110 Security Processor, 2003. http://www.corrent.com/pdfs/CR7110_%20Pbrief_v3.pdf
[5] D. Chang, K.T. Cheng, M. Sadowska, Functional Scan Chain Testing, pp278-284, DATE, 1998
[6] M.L. Bushnell and V.D. Agrawal, Essentials of Electronic Testing, Kluwer Academic Publishers, ISNB 0-7923-7991-8, 2000.
[7] D, Josephson, S. Poehhnan, Debug methodology for the McKinley processor, pp451-460, ITC, 2001
[8] National Bureau of Standards, Data Encryption Standard, Federal Information Processing Standards Publication FIPS PUB 46, 1977
[9] Atmel, 32-bit Embedded Core Peripheral: DES http://www.atmel.com/dyn/resources/prod_documents/doc1351.pdf
[10] P. Kocher. Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. pp.104-113, CRYPTO, 1996
[11] Biham and A. Shamir, Differential Fault Analysis of Secret Key Cryptosystems, pp. 156-171, CRYPTO, 1991
[12] P. Kocher, J. Jaffe, B. Jun, Differential Power Analysis, pp.388-397, CRYPTO, 1999