# Abstract

Scan chain test technique is used as an attacking method to retrieve User Key from cryptographic hardware device. With the establishment of PRESENT, an ultra-lightweight block cipher fit for hardware environment, the security level from scan based side-channel attacks should be improved because this attack is a powerful analyzing method by injecting into encryption flow of ICs.

This project is a mix of Type I and Type II. It has to construct simulation software of the hardware device to be the experiment environment. It also has investigatory of existing scan based attack methods and countermeasures.

The contributions and achievements in my project are listed as follows:

1. I implemented two software programs, which perform like DES IC and PRESENT IC from the scan chain view. The DES IC software is programming based on source code in [1]. The PRESENT IC software is programming based on source code in [2].
2. I simulated a successful scan based side-channel attack on DES and proposed a new calculation for reversing of $SBox$ in DES algorithm.
3. I developed a novel scan based attack on PRESENT and simulated a specific attack on PRESENT IC successfully.
4. I simulated four countermeasures in PRESENT IC in the software environment and compared them from aspects of efficiency and security.

# Contents

# Chapter 1

# Introduction

Recently, as Large Scale Integrated Circuit (LSIC) is ubiquitous in modern life, security issue has become a critical concern. Especially with cryptographic Integrated Circuits (IC) widely used in areas such as hardware encryption, data integrity verification, signature verification and store of sensitive information, people would always worry the stolen of credits in bank card, leakage of personal information in SIM card and so on. Attacks on hardware become a serious threat for integrity and confidentiality.

There are many categories of different attacking methods on hardware: fault attack, reverse attack, side-channel attack, etc. These attacks are divided into active and passive attacks according to the behaviors of attackers. They are divided into invasive, semi-invasive and non-invasive attacks according to the performance of attacking techniques.

Firstly, scan based attack is a kind of side-channel attack because it recover key information by observing the physical leakage from hardware device. Secondly, scan chain based side-channel attacks are non-invasive (some are semi-invasive) because it is to use legal method to access secret inside hardware device. Thirdly, it is a kind of active attack because attackers should gain key information from cryptographic ICs using additional mathematical analysis and captured physical data [3].

As side-channel attacks perform well in cryptanalysis and many researchers have proposed effective attacks to reveal secret information from hardware, side-channel attacks become a serious threat to hardware security. Attackers can recover key

material from hardware by analyzing the leakage of side channel, such as power, timing, etc.

Platforms like Microsoft and OS X all have access for users (potential attackers) to overwrite some pieces of codes. This is a potential vulnerability to exploit and analyze by hackers. Many companies have suffered from such immense damage by vulnerability attacking. Similarly, scan chain test technique is provided as a physical access for users, although it is designed as a convenient tool for test engineering. So scan based attacks would cause a great loss for companies. As shown in [4], this kind of leakage is a reality today; and it will be a challenge tomorrow. [5] gives a data breach investigation in 2012, where physical attack is a powerful attacking method compared with social engineer and privilege misuse.

In 2004, [6] proposed a new side-channel attack to recover user key from DES IC by injecting into scan chains. More attacks are developed since then. Existing scan based attacks are very robust against cryptographic ICs. Besides, it is a very practical attacking method.

Since scan chain technique is a widely accepted test method for ICs, the security for cryptographic hardware is more serious. It is convenient for attackers to develop a side-channel attack by abusing scan chains. Effective countermeasures should be urgently proposed against such attacks. Therefore, this area needs more attention and research.

From the view of hardware manufacturers, they want to design low-cost, high-performance and secure hardware devices for a mass production. Although there are a trade-off among these things, this paper will give some valid ideas and suggestions to decide the implementation after weighing the pros and cons.

This project is to simulate existing attack, develop a novel attack and proposes effective countermeasures based on previous research. The analysis on the countermeasures simulated in this project will focus on non-invasive attack. This paper not only has theoretical meaning in scan based side-channel attack area but also has essential value in practice. Therefore, it is necessary and significant to research in this field.

## 1.1 Objectives and Aims

The ultimate aim of this project is to develop a new side-channel attack on existing cipher fit for hardware using scan chain test technique and give some effective countermeasures. To achieve this aim, this project is divided into the several following objectives:

1. Research on scan test technique and implement cryptographic ICs from view of scan chain.
2. Study the existing attacks on cryptographic ICs using scan chain and simulate the attacks on such algorithms such as DES.
3. Propose the algorithm of a novel attack on PRESENT and simulate it.
4. Simulate existing countermeasures and compare them to give the most appropriate one.

## 1.2 Achievements

This project is to research on scan chain based side-channel attack and propose effective countermeasures. It is a mixed type, so it has theoretical analysis and software implementation. Existing attack on DES has been simulated successfully in this project. Beyond this, a novel, valid scan based attack on PRESENT, an ultra-lightweight block cipher, has been developed and simulated in this project. Additionally, different countermeasures have been tried as prevention schemes against such attack; and the results are compared after analyzing extended attacks. The achievements are listed as follows:

**Analysis Achievements**

1. Investigate and analyze existing cryptographic algorithms and existing scan based attack.
2. Develop a new scan based attack on PRESENT by analyzing algorithm PRESENT and existing scan based attack on DES.
3. Compare some effective countermeasures against such attack on PRESENT.

**Software Achievements**

1. Simulate DES hardware implementation and its relevant scan based side-channel attack.
2. Simulate PRESENT hardware implementation and its relevant attack using Analysis 2.
3. Simulate four different countermeasures.
4. Simulate reset attack against Inverter Chain prevention scheme.

## 1.3 Organization

This main body of this paper is organized into six chapters. They are listed as follows: Chapter 2 gives a detailed description about the technique background about the scan based attack. Chapter 3 describes the attacking steps of existing attack on DES and its simulation. Chapter 4 proposes scan based attack on PRESENT and its simulation, which is the main part of this project. Chapter 5 presents different countermeasures after they are added into oracle algorithm. Chapter 6 gives further attack against these countermeasures, compares these countermeasures and gives the result, which is the most effective prevention scheme. Finally, Chapter 7 concludes the evaluation of this paper and discusses some future work.

# Chapter 2

# Technique Background

This section is to give a detailed description about scan chain technique, which is the base of scan based side-channel attack and the previous work about such attacks.

## 2.1 Scan Test

With the fast development of LSIC, testing becomes much more difficult for test engineers. They have less access to the entire IC, so traditional test methods cannot meet the test requirements of such LSICs with more pins, smaller size and greater density. Therefore, a new test method, Joint Test Action Group (JTAG) is developed, which is called scan test technique.

Scan test is developed in the 1990s with the emergence of LSIC, and it soon becomes a widely accepted Design-For-Test (DFT) technique. It obviously improves two important properties of testability: controllability and observability. The state of every flip-flop in the IC is conveniently tested by connecting them into a scan chain, like a large shift register. Test engineers can use Test Access Port (TAP) to access complex IC. TAP controller recognizes the communication protocol and generates internal signals [7]. Test engineers can communicate with the IC using three controlling pins: Test Clock (TCK), Test Mode Select (TMS) and Test Reset (TRST), (which is optional) and using two observing pins: Test Data In (TDI) and Test Data Out (TDO). TCK can control each clock cycle of the inside logic while TMS can switch the IC between normal mode and test mode. Then in each clock cycle in test mode, the states of every flip-flop can be scanned in via TDI and scanned out via TDO.

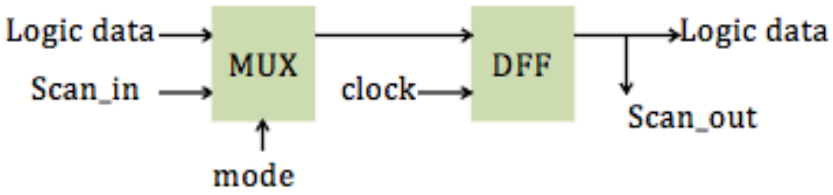


Figure 2.1: A DFF with MUX.

The input of a DFF in the IC is connected with a MUX during the test synthesis, which is shown in Figure 2.1. The MUX can be switched to a normal mode, in which the MUX with DFF performs like a normal DFF; and the MUX can be switched to a test mode, in which the data can be scanned in and out. For example, if the mode is set to normal mode, in which the MUX with DFF performs like a normal DFF; the data is processed in Combinational Circuit and scanned in the DFFs via MUX directly. If the mode is set to test mode, testers can choose test data and input it into DFFs via MUX through TDI pin. Then data flows over all DFFs in the scan chain by several clocks. After data is processed in the Combinational Circuit in normal mode, it can be scanned out from TDO pin in test mode.

A basic scan chain is shown in Figure 2.2. This scan chain is shown from a macro view where all DFFs are connected ignoring the effect of MUXs. In this chain, all D flip-flops (DFF) are connected as a large register, and they can be simply tested using TAP. In this example, Counter Register and temporary Data Register are connected as a whole chain. Test engineers control the flow of IC by scanning data into the chain, and observe the states of IC by scanning data out of the chain. The scan chain performs as a communicating tool between test engineers and internal logic. In this method, faults are easily discovered by programming using scan-in data and scan-out data. This reduces duplication of time and labor.

The convenience to scan in and out data of every DFF for the attackers gives an effective method to observe the structure of cryptographic IC and acquire sensitive, intermediate data in an encryption.

## 2.2 Hardware Security

Information technology has affected many aspects of modern life more widely. The importance of information security has become increasingly prominent. As an

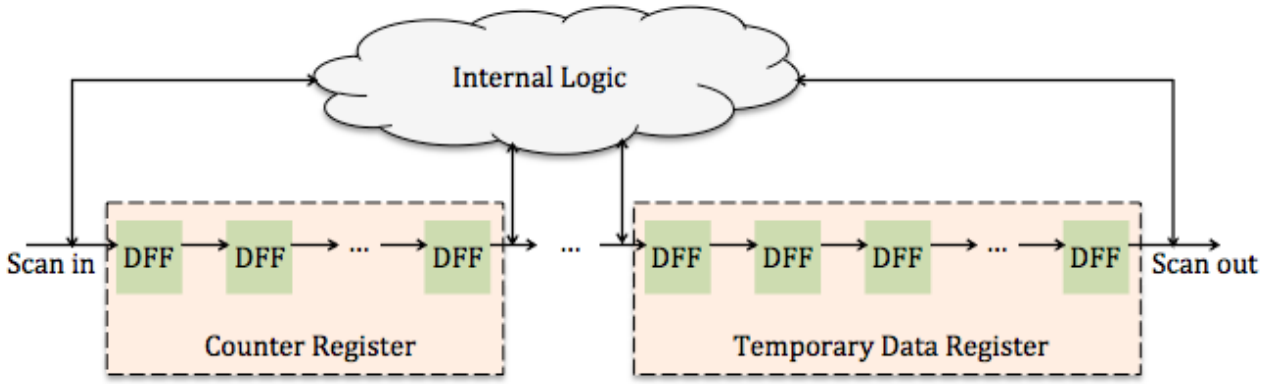


Figure 2.2: Scan Chain.

important part of information security, the cryptographic ICs have been widely used to apply security in the digital field.

Recently, side-channel attack on hardware is a great threat to such cryptographic ICs. Attackers analyze leakage data of the device to recover key material. Fault attack, Differential Power Analysis (DPA), and Time Attack are most effective attacks on ICs. Federal Information Processing Standards 140-2 [8] has given the definition of security issues for cryptographic module, which is cited by [9], "A cryptographic module shall employ physical security mechanisms in order to restrict unauthorized physical access to the contents of the module and to deter unauthorized use or modification of the module (including substitution of the entire module) when installed. All hardware, software, firmware, and data components within the cryptographic boundary shall be protected. " This definition gives a serious alarm for hardware designers because any leakage of hardware device can reveal the key material and lead to grave consequences.

[10] has exploited some categories of existing side-channel attacks on current cryptographic algorithms. In fault attack, attackers make the IC in an abnormal environment to acquire additional access to the inner logic by generating malfunctions. In DPA, attackers analyze the instantaneous power consumption of a cryptographic IC by observing the data it processes and the operations it performs and then exploit key material. In time attack, attackers make a statistical hypothesis using timing data, divide samples into two sets depending on the hypothesis and then give the result after comparing two sets.

However, with the development of side-channel attack, researchers have realized the necessity to propose countermeasures against such attacks. For example, researchers modify hardware structure to make it less effective to generate an abnormal environment; the frequency of hardware operations is increased significantly to generate more noise in power traces; programmers write codes using secure methods to avoid hypothesis differences developed by timing data.

Although some countermeasures are proposed; there are still many security problems and leakages existing hardware security. Scan test based side-channel attack is one of the most effective attacking methods to recover key from hardware device and this project just focuses on scan based side-channel attack.

## 2.3 Scan Based Attack

As described in Section 2.1, scan test is a powerful test technique for LSICs. However, scan test can be abused as an attacking method for attackers. Attackers can control data flow in and out of the IC by injecting into scan chains. The bit of every DFF is the leakage of the cryptographic device. The scan chain stores intermediate data of the operations of an encryption. So scan based attack is a kind of side-channel attacks.

Scan based side-channel attack was firstly developed by Yang et al. in 2004. In [6], the authors described detailed attacking phases on DES IC using scan test. In 2005, Yang et al. mounted attack on AES IC again, together with a new countermeasure, Mirror Key Register (MKR) proposed at this time [9]. From then on, many papers about scan based attack and countermeasures are proposed.

Recently, there are four papers, which have developed some methods to attack different cryptographic ICs. Despite the attacks on DES and AES, which are all proposed by Yang et al., there are attack on Trivium, a stream cipher by Agrawal et al. in 2008 and attack on RSA by Rolt et al. in 2012. This project has simulated the attack on DES and proposed a novel attack on PRESENT, a block cipher fit for hardware environment.

As to countermeasures, in 2006, [11] presented different techniques to secure scan chain and gave the comparison of their pros and cons. In 2007, [12] used the mind of random accessing and proposed a flexible prevention scheme, Lock&Key technique.

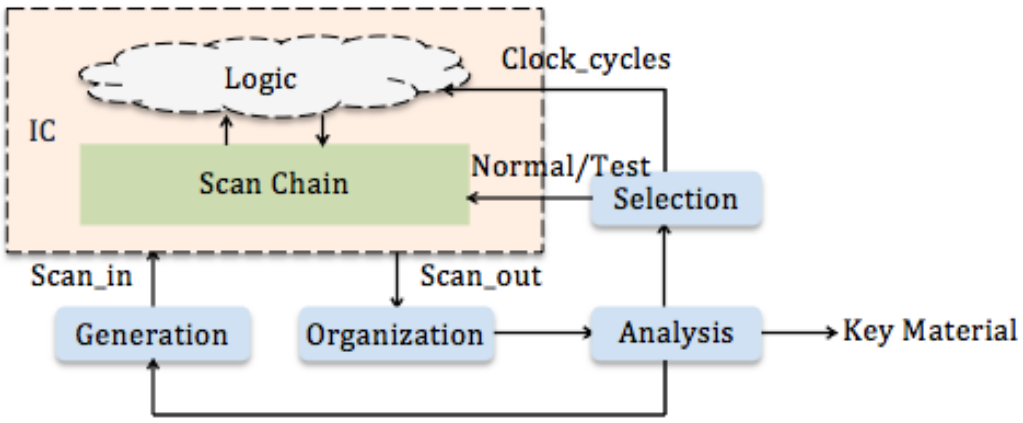


Figure 2.3: Scan Based Attack.

In 2008, [13] developed attack on stream ciphers and gave XOR Chain countermeasures based on countermeasure Inverter Chain proposed by [14]. In 2009, Inoue proposed a partial scan approach protection scheme [15]. However, it is proved not secure in [16], which is the latest paper about the scan based attack on RSA and its analysis after using some existing prevention schemes.

This project will choose four typical countermeasures and compare them from various views. Finally, a most appropriate countermeasure for PRESENT IC will be given.

Although to different cryptographic algorithms; there are slightly different attacking details, all scan based attacks can be described in fixed phases, which are shown in Figure 2.3. An attacker firstly gives some analysis about the attack on specific cryptographic IC and generates some scan-in data. Then the attacker selects different mode and certain clock cycles depending on the analysis and receives scan-out data from scan chain. Attacker organizes the whole data from scan chain and analyzes it again to generate scan-in data until finding out the key material.

# Chapter 3

# Attack on DES

## 3.1 Implementation of DES IC

Data Encryption Standard (DES) algorithm is a symmetric block cipher with 64-bit data block 64-bit key. Actually, only 56 bits of the secret key participate in the DES encryption and decryption. Each 64-bit block of plaintext is mixed into a 64-bit block of ciphertext based on 16 rounds of balanced feistel structure.

Recently, DES is widely used in hardware with iterative structure. The overall structure of DES algorithm is shown in Figure 3.1. The DES encryption performed using three operations: $InitialPermutation$ $(IP)$, $Function(F)$ and $FinalPermutation$ $(FP)$ where $F$ has an iterative structure for 16 rounds. In a DES IC, plaintext, Left $(L)$, Right$(R)$ and ciphertext are stored in their corresponding registers, and these registers can be connected into a scan chain. The operations between $RoundKey$ $(RK)$ and registers in a DES round are described as follows:

$$l = R_0 \tag{3.1}$$

$$r = L_0 \oplus d \tag{3.2}$$

$$a = Expand(l) \tag{3.3}$$

$$b = a \oplus RK_1 \tag{3.4}$$

$$c = SBox(b) \tag{3.5}$$

$$d = Permutation(c) \tag{3.6}$$

$$e = d \oplus l \tag{3.7}$$

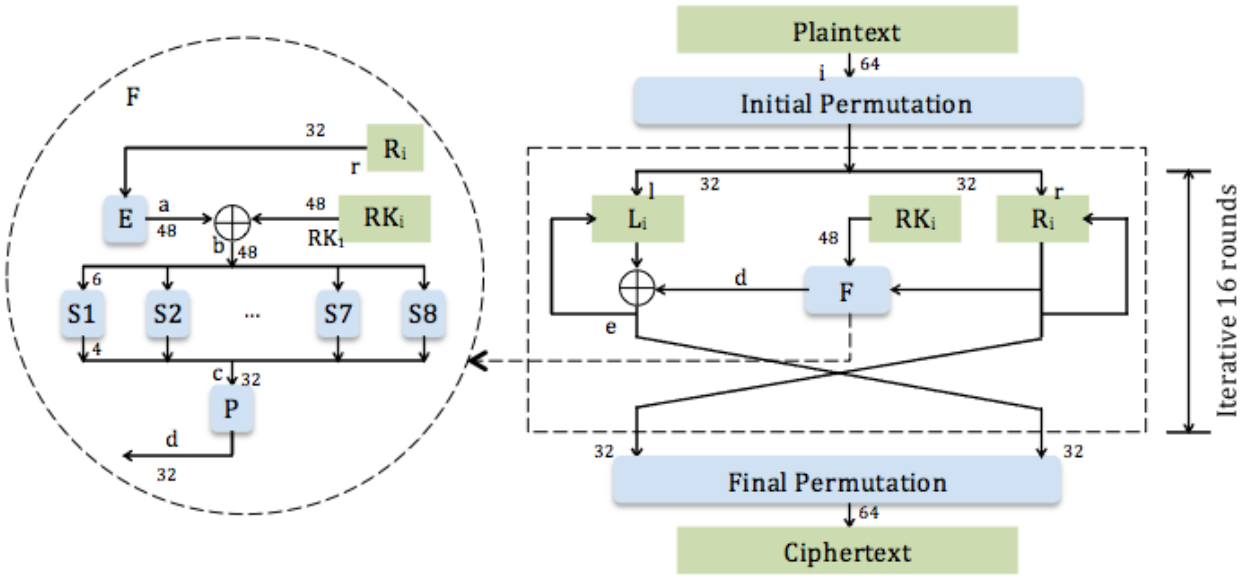


Figure 3.1: DES Encryption Structure.

In addition, there is one more operation $IP$ in the first round and one more operation $FP$ in the last round.

The scan chain in DES IC includes five registers, Counter Register, Input Register, Left Register, Right Register and Output Register. The data of scan chain can be scanned in or scanned out every clock cycle. The DES IC implementation is shown in Figure 3.2. Tester can control scan-in data and observe scan-out data from scan chain. Besides, test engineers can select clock cycles and switch between normal mode and test mode. This figure also clearly shows although $RoundKey$ is stored in RK Register and the RK Register is not included in the scan chain. The generation of $RKs$ and operations of encryption such as $IP$, $Sbox$ and $FP$ are all described in [17] in details. An entire encryption of DES needs 20 clock cycles to get the ciphertext. They are listed as follows:

1. Cycle 1: Load plaintext into Input Register via scan chain.
2. Cycle 2: Load plaintext into Left Register and Right Register.
3. Cycle 3: Load data into Left Register and Right Register after $IP$.

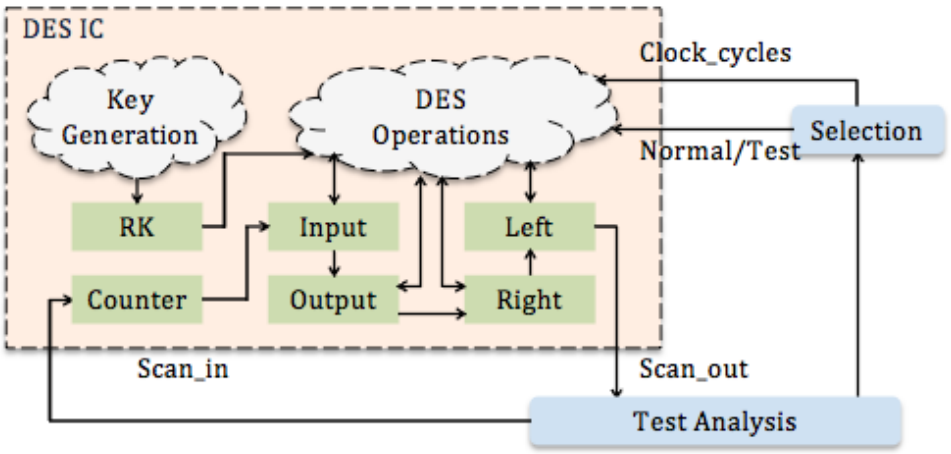


Figure 3.2: DES IC Implementation.

4. Cycle 4 - 19: Load data into Left Register and Right Register after Round 1-16.

5. Load data into Output Register after $FP$.

Therefore, this simulation can perform all functions of DES IC and scan chain test technique.

## 3.2 Simulation of DES Attack

[6] proposed a three-phase side-channel attack on DES using scan test: determine the location of scan chain, recover round keys and recover user key. This section is to describe a detailed, successful simulation of a DES attack using scan chain.

This simulation is in a particular scenario. There is a DES IC with a fixed key embedded inside and this IC is used to encrypt only one block plaintext (64 bits). Initially, the plaintext is loaded into Input Register. In the encryption operations, Counter Register, Left Register and Right Register are loaded with processing bits. Finally, the ciphertext, which is encrypted after using DES algorithm with the fixed key, is loaded into Output Register. A successful attack is that attackers use scan test technique to recover the fixed key.

### 3.2.1 Determine location of Scan Chain

Scan chain, which is shown in Figure 2.2, is connected of many registers where each register is connected of many DFFs. Before attackers mount an attack, they need to know the location of each register in the scan chain. For example, in Figure 3.2, scan chain of this DES IC is connected in this order: Counter Register (4 bits), Input Register (64 bits), Output Register (64 bits), Right Register (32 bits) and Left Register (32 bits). One bit is stored in one DFF, so this scan chain is connected using 196 DFFs.

Before a real attack, attackers can scan out 196 bits from the scan chain of any clock cycle, but they must have doubts what are these bits corresponding to which register. So the first phase is to determine the location of scan chain, which means attackers need to know each bit in the scan chain is belonged to which register.

From the 20 clock cycles described in Section 3.1, the steps to determine the location of scan chain are shown as follows.

**Input Register** Input Register includes 64 DFFs, which represents a DES plaintext block. The plaintext (64 bits) is loaded into Input Register in the first clock cycle. So after a one clock cycle, only the locations where are DFFs of Input Register will change and other locations in the scan chain still remain 0.

1. Select normal mode, input plaintext 0000000000000001 (in hexadecimal) and run 1 cycle. Switch to test mode, scan out the bit stream. There should be only one bit, which is 1 and other 195 bits are all 0. The location, which is bit 1, represents the LSB of Input Register. For example, in Figure 3.3, the 64 bit in the scan out bit stream is the location of Least Significant Bit (LSB) of Input Register.

2. Repeat step 1 63 times with changing plaintext one bit different from previous plaintext. There should be only one bit, which is 1 and other 195 bits are all 0. It means in the $i^{th}$ times, input plaintext 000...0001000...0 (64 bits) where the $i^{th}$ bit is 1 and others are all 0. So the location, which is 1 in the bit stream, represents the $i^{th}$ Most Significant Bit (MSB) of Input Register.

All locations of Input Register DFFs are determined in the scan chain.

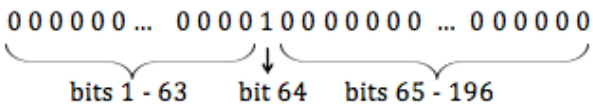


Figure 3.3: Determine LSB of Input Register.

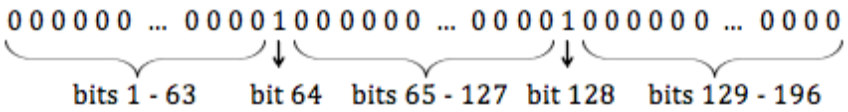


Figure 3.4: Determine LSB of Right Register.

**Left and Right Register** Left and Right Registers are all include 32 DFFs respectively, which represents the left half block (32 bits) and the right half block (32 bits) in DES Round operations. In this simulation, the left and right half block is loaded into Left and Right Registers before IP. So the Left and Right Registers can be loaded in cycle 2.

1. Select normal mode, input plaintext 0000000000000001 (in hexadecimal) and run 2 cycles. Switch to test mode, scan out the bit stream. There should be two bit, which are 1 and other 195 bits are all 0. One of the bits, which are 1, is determined as the DFF of Input Register. So the left one is the location of LSB of Right Register. For example, in Figure 3.4, the 64 bit in the scan chain is the location of LSB of Input Register, so the 128 bits represents the location of LSB of Right Register.

2. Repeat step 1 31 times with changing right half block of plaintext one bit different from previous plaintext. The range is from 0000000000000002 (in hexadecimal) to 000000008000000 (in hexadecimal). Determine the locations of Right Register in step used to determine Input Register.

3. Repeat step 1 and 2 32 times with changing left half block plaintext one bit different. The range is from 0000000100000000 (in hexadecimal) to 8000000000000000 (in hexadecimal). Determine the locations of Left Register.

All locations of Left and Right Register DFFs are determined in the scan chain. Therefore, there are already 128-bit locations are determined, and 68 bits are to be determined. In these 68 bits, 64 bits are from Output Register and 4 bits are from Counter Register.

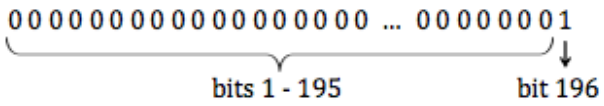


Figure 3.5: Determine LSB of Counter Register.

**Counter Register** Counter Register includes four bits, which is the MSB, Second MSB, Third MSB and LSB. After recover Input Register, Left Register and Right Register, there are Counter Register (4 bits) and Output Register (64 bits) not determined. However, before FP is operated, the final ciphertext will not be scanned into Output Register. So the DFFs of Output Register will still store 0.

1. Select normal mode, input plaintext 0000000000000000 (in hexadecimal) and run 4 cycles (Round 1 in DES encryption). Switch to test mode, scan out the bit stream. In the remaining bit locations (68 bits) which are not determined, there should be only one bit, which is 1 and other 67 bits are all 0.

2. Repeat step 1 three times with changing the clock cycle to cycle 5 (Round 2), cycle 7 (Round 4) and cycle 11 (Round 8). In the remaining bit locations which are not determined, there should be also only one bit, which is 1 and other 67 bits are all 0. The location of bit 1 using 2 cycles represents Third MSB. The location of bit 1 using 4 cycles represents Second MSB. The location of bit 1 using 8 cycles represents MSB.

In this method, all locations of Counter Register DFFs are determined in the scan chain.

**Output Register** Up to now, only the locations of Output Register are not determined. So the remaining 64 bits are locations of Output Register.

### 3.2.2 Recover Round Keys

After attackers know the locations of all the registers, they are easy to determine the bits of Left and Right Register from scan chain, which means they can know the intermediate data: $L_i$ and $R_i$ of plaintext. From equations in Section 3.1, the variables $l$, $r$ and $e$ are easy to determine from scan chain. The purpose of this section is to use the known $l$, $r$, $e$ to recover $RK_i$.

From Equations 3.1 to 3.7, the analysis is listed as follows:

$$d = e \oplus l \tag{3.8}$$

$$c = ReversePermutation(d) \tag{3.9}$$

$$a = Expand(l) \tag{3.10}$$

$$b = ReverseSBox(c) \tag{3.11}$$

$$RK_i = a \oplus b \tag{3.12}$$

Variables $e$ and $l$ are known from scan chain, d is calculated from Equation 3.8 as the operation $\oplus$ is a one-to-one correspondent. The *ReversePermutation* function in Equation 3.9 permutes a 32-bit input into a 32-bit output. Only the table of $RP$ is different from *Permutation* ($P$) in DES algorithm, which is shown in Table 3.1.

Table 3.1: *ReversePermutation*.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8 | 16 | 22 | 30 | 12 | 27 | 1 | 17 |
| 23 | 15 | 29 | 5 | 25 | 19 | 9 | 0 |
| 7 | 13 | 24 | 2 | 3 | 28 | 10 | 18 |
| 31 | 11 | 21 | 6 | 4 | 26 | 14 | 20 |

In Equation 3.10, variable $a$ can be calculated from table *Expand* in DES algorithm [17]. The next analysis in Equation 3.11 is to calculate variable $b$ from variable $c$ via *ReverseSBox*. *SBox* is a function, which has a 6-bit input and 4-bit output. So the input amount is $2^6$ and the output amount is $2^4$, which means a 4-bit output could have four possible 6-bit input hypotheses. In *SBox* tables, if $b$ is a 6-bit block, let the number which is represented by the first and last bits in base 2 be i, let the number which is represented by the middle 4 bits in base 2 be j. The number in $i^{th}$ row, $j^{th}$ column, is the output number, which ranges from 0 to15. This output number is represented by a 4-bit block in base 2. For example, in *SBox* $S3$, which is shown Table 3.2, if the output is 5(0101), the input can be the number in row 0, column 11, or row 1, column 1, or row 2, column 15, or row 3, column 7. So the four possible 6-bit block inputs are 22(010110), 3(000011), 62(111110) and 47(101111). In this method, all four input hypotheses can be calculated when given a fixed output by *ReverseSBox* ($RS$) 1 to 16, which is listed in Table 3.3. In this table, all the numbers

represent their corresponding number in base 2. Therefore, there are four possible $b$ from a fixed $c$ from Equation 3.11 and four possible $RK_i$ from Equation 3.12. The steps of using three known scan-in bit stream to recover $RK_i$ are described as follows:

1. Scan 0000000000000000 (in hexadecimal) as a 64-bit block into L and R Registers before loading $RK_i$. From Figure 3.1, $l$ and $r$ are all 32 bits of 0. Scan out the bit stream after one round and variable $e$ is known From Equations 3.8 to 3.12, $a$ is a 48-bit stream of 0. As $e$ is determined from scan-out bits and variable $l$ is 0, variable $d$ is equal to $e$. variable $c$ can be calculated from $d$. So four hypotheses $b$ can be analyzed from $c$ and Table 3.3. As $a$ is 0, variable $b$ can be written as $K_{48}K_{47}K_{46}K_{45}K_{44}K_{43}...K_1$ where $K_j$ is the j$^{\text{th}}$ bit of $RK_i$. Therefore, four possible $RK_i$s are analyzed in this method.

2. Repeat step 1 and scan in 0000000044444444 (in hexadecimal), another four possible $RK_i$s are analyzed. Here variable $b$ is $K_{48}K_{47}\overline{K}_{46}K_{45}K_{44}K_{43}...K_1$ where there is an over line every 6 bits.

3. Repeat step 1 with scanning in 0000000088888888 and get four possible $RK_i$s. Here variable $b$ is $K_{48}\overline{K}_{47}K_{46}K_{45}K_{44}K_{43}...\overline{K}_1$ where there are two over lines every 6 bits.

Table 3.2: $SBox$ 3.

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

Combined analyses of step 1 and step 2, the amount of hypothesis $RK_i$ are narrowed to two. Combined with step 3, there is only one hypothesis left, which is $RK_i$. For example, supposing the first 6 bits of $RK_i$ are 44 (101100), the analysis using steps 1 to 3 are listed as follows:

Table 3.3: $ReverseSBox$.

| | Output | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $RS1$ | Input 1 | 28 | 6 | 8 | 16 | 2 | 24 | 20 | 30 | 14 | 26 | 18 | 12 | 22 | 4 | 0 | 10 |
| | Input 2 | 1 | 15 | 11 | 29 | 7 | 27 | 19 | 5 | 31 | 25 | 17 | 23 | 21 | 13 | 9 | 3 |
| | Input 3 | 62 | 34 | 44 | 56 | 32 | 60 | 42 | 54 | 38 | 52 | 58 | 46 | 50 | 60 | 36 | 48 |
| | Input 4 | 59 | 45 | 39 | 53 | 41 | 49 | 61 | 47 | 37 | 43 | 57 | 51 | 35 | 63 | 55 | 33 |
| $RS2$ | Input 1 | 26 | 2 | 20 | 12 | 14 | 28 | 8 | 18 | 4 | 16 | 30 | 10 | 24 | 22 | 6 | 0 |
| | Input 2 | 19 | 21 | 11 | 1 | 5 | 31 | 25 | 7 | 13 | 27 | 23 | 29 | 17 | 3 | 15 | 9 |
| | Input 3 | 32 | 46 | 60 | 58 | 42 | 48 | 54 | 36 | 50 | 56 | 40 | 38 | 52 | 44 | 34 | 62 |
| | Input 4 | 57 | 39 | 47 | 41 | 45 | 59 | 51 | 53 | 35 | 63 | 37 | 49 | 55 | 33 | 61 | 43 |
| $RS3$ | Input 1 | 2 | 16 | 28 | 10 | 26 | 14 | 8 | 22 | 30 | 4 | 0 | 24 | 20 | 18 | 6 | 12 |
| | Input 2 | 5 | 31 | 17 | 9 | 11 | 21 | 13 | 3 | 19 | 7 | 15 | 27 | 25 | 1 | 23 | 29 |
| | Input 3 | 46 | 50 | 52 | 44 | 36 | 56 | 34 | 62 | 40 | 38 | 58 | 48 | 54 | 32 | 60 | 42 |
| | Input 4 | 39 | 33 | 61 | 55 | 49 | 59 | 41 | 47 | 45 | 43 | 35 | 57 | 63 | 37 | 53 | 51 |
| $RS4$ | Input 1 | 8 | 16 | 18 | 6 | 28 | 22 | 10 | 0 | 20 | 12 | 14 | 24 | 26 | 3 | 4 | 30 |
| | Input 2 | 18 | 25 | 21 | 15 | 17 | 7 | 9 | 19 | 3 | 31 | 27 | 5 | 23 | 2 | 29 | 11 |
| | Input 3 | 38 | 50 | 58 | 52 | 62 | 56 | 34 | 44 | 60 | 36 | 32 | 42 | 40 | 46 | 54 | 48 |
| | Input 4 | 37 | 43 | 61 | 33 | 51 | 53 | 39 | 59 | 47 | 49 | 41 | 55 | 57 | 45 | 63 | 35 |
| $RS5$ | Input 1 | 26 | 6 | 0 | 20 | 4 | 18 | 14 | 8 | 16 | 30 | 10 | 12 | 2 | 24 | 28 | 22 |
| | Input 2 | 19 | 15 | 5 | 15 | 9 | 17 | 31 | 11 | 29 | 27 | 23 | 3 | 7 | 13 | 1 | 21 |
| | Input 3 | 60 | 36 | 34 | 58 | 32 | 54 | 56 | 44 | 46 | 50 | 40 | 38 | 52 | 42 | 62 | 48 |
| | Input 4 | 53 | 41 | 45 | 63 | 59 | 61 | 49 | 39 | 35 | 55 | 57 | 33 | 37 | 47 | 43 | 51 |
| $RS6$ | Input 1 | 16 | 2 | 10 | 20 | 22 | 28 | 12 | 26 | 14 | 8 | 4 | 30 | 0 | 18 | 24 | 6 |
| | Input 2 | 25 | 19 | 7 | 29 | 5 | 15 | 17 | 9 | 31 | 13 | 1 | 27 | 11 | 21 | 23 | 3 |
| | Input 3 | 50 | 56 | 40 | 46 | 52 | 38 | 62 | 48 | 42 | 32 | 54 | 60 | 44 | 58 | 34 | 36 |
| | Input 4 | 59 | 53 | 37 | 35 | 33 | 43 | 57 | 55 | 61 | 41 | 47 | 49 | 39 | 63 | 51 | 45 |
| $RS7$ | Input 1 | 10 | 30 | 4 | 16 | 0 | 24 | 28 | 22 | 12 | 20 | 26 | 2 | 18 | 14 | 6 | 8 |
| | Input 2 | 3 | 13 | 25 | 19 | 9 | 21 | 31 | 7 | 29 | 11 | 15 | 5 | 23 | 1 | 17 | 27 |
| | Input 3 | 56 | 32 | 62 | 42 | 34 | 58 | 52 | 44 | 54 | 60 | 48 | 36 | 40 | 38 | 46 | 50 |
| | Input 4 | 53 | 41 | 59 | 61 | 43 | 51 | 33 | 47 | 39 | 49 | 45 | 35 | 63 | 37 | 57 | 55 |
| $RS8$ | Input 1 | 25 | 14 | 2 | 20 | 6 | 24 | 8 | 30 | 4 | 18 | 16 | 12 | 28 | 0 | 22 | 10 |
| | Input 2 | 26 | 1 | 31 | 11 | 15 | 19 | 21 | 13 | 7 | 29 | 9 | 23 | 17 | 5 | 27 | 3 |
| | Input 3 | 48 | 38 | 46 | 58 | 36 | 60 | 50 | 32 | 62 | 40 | 52 | 34 | 42 | 54 | 44 | 56 |
| | Input 4 | 55 | 35 | 33 | 57 | 41 | 59 | 61 | 39 | 45 | 53 | 43 | 63 | 51 | 47 | 37 | 49 |

1. When scan-in data is 0000000000000000 (in hexadecimal), the first 4 bits of variable $c$ are 2 (0010), which are calculated from l and e. From variable $c$ and Table 3.3, the first 6 bits of $b$ can be 8 (001000), 11 (001011), 44 (101100) and 39 (100111). As the first 6 bits of a are 0 (000000), the first 6 bits of possible $RK_i$ is equal to $b$.

2. When scan-in data is 0000000044444444 (in hexadecimal), the first 4 bits of variable $c$ are 14 (1110), which are calculated from l and e. From variable $c$, the first 6 bits of $b$ can be 0 (000000), 9 (001001), 36 (100100) and 55 (110111). As the first 6 bits of a are 8 (001000), the first 6 bits of $RK_i$ can be 8 (001000), 1 (000001), 44 (101100) and 63 (111111). Combined with previous step, the hypotheses of the first 6 bits of $RK_i$ are narrowed to 8 (001000) and 44 (101100).

3. When scan-in data is 0000000088888888 (in hexadecimal), the first 4 bits of variable $c$ are 6 (0110), which are calculated from l and e. From variable $c$, the first 6 bits of $b$ can be 20 (010100), 19 (010011), 42 (101010) and 61 (111101). As the first 6 bits of a are 16 (010001), the first 6 bits of $RK_i$ can be 5 (000101), 2 (000010), 59 (111011) and 44 (101100). Combined with previous step, the hypothesis is narrowed to 44 (101100).

Using analyses of the above steps, all Round Keys can be recovered using scan chain.

### 3.2.3 Recover User Key

This is the last phase to attack DES IC. After analyzing $RK_i$s in DES algorithm, attackers need to recover User Key to complete a successful attack. The Round Key generation is shown in Figure 3.6. In this figure, $RK_i$s of DES are all generated from User Key using two operations: Permutation and Left Shift. The details of Permutation Choice 1, Permutation Choice 2 and Left Shift, are described in [17].

From the generation of Round Keys, $RK_1$ and $RK_2$ can be regarded as a contracted permutation of User Key. The Permutation table of the first two Round Keys are listed in Table 3.4. In this table, each $RK_i$ is 48 bit, and each bit in the table is the corresponding location in User Key. For example, the $12^{\text{th}}$ bit of $RK_2$ is 29, which means the $29^{\text{th}}$ bit of User Key. Therefore, $RK_1$ only not covers bit 5, 6, 9, 10, 37, 40, 43 and 45. Combined with $RK_2$, all the bits in User Key are covered, which

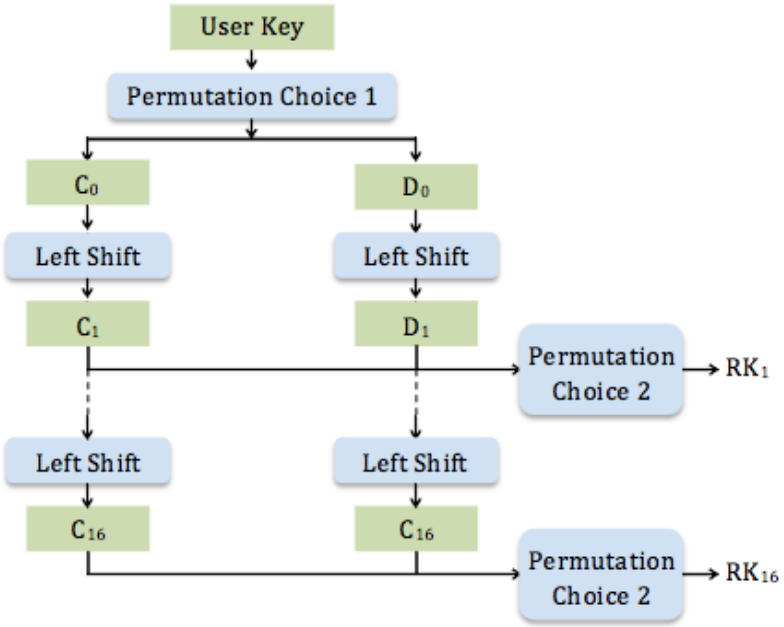


Figure 3.6: Round Key Generation.

means User Key can be determined from $RK_1$ and $RK_2$. Then a valid and complete attack has been successfully made.

In Table 3.4, number '8' in the first line means the first position of $RK_1$ is the $8(+1)^{\text{th}}$ position of User Key. So from this table, the $0(+1)^{\text{th}}$ bit of User Key is the $20^{\text{th}}$ position of $RK_1$. Therefore, User Key can be determined from $RK_1$ and $RK_2$ as these two Round Keys have covered all positions of User Key.

## 3.3 A Specific Attacking Case on DES

This section is to describe a specific attacking case on DES in details, which has already been simulated in this project. Although [6] has given an algorithm that how to recover the User Key using scan test technique, this project improves this algorithm in details and gives the $ReverseSBox$ of this attack. The attack can be easily mounted using the above tables shown. A specific attacking case is given to analyze that how a successful attack is developed.

Table 3.4: Round Keys Permutation.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $RK_1$ | 8 | 44 | 29 | 52 | 42 | 14 | 28 | 49 |
| | 1 | 7 | 16 | 36 | 2 | 30 | 22 | 21 |
| | 38 | 50 | 51 | 0 | 31 | 23 | 15 | 35 |
| | 19 | 24 | 34 | 47 | 32 | 3 | 41 | 26 |
| | 4 | 46 | 20 | 25 | 53 | 18 | 33 | 55 |
| | 13 | 17 | 39 | 12 | 11 | 54 | 48 | 27 |
| $RK_2$ | 1 | 37 | 22 | 45 | 35 | 7 | 27 | 42 |
| | 51 | 0 | 9 | 29 | 52 | 23 | 15 | 14 |
| | 31 | 43 | 44 | 50 | 49 | 16 | 8 | 28 |
| | 12 | 17 | 27 | 40 | 25 | 55 | 34 | 19 |
| | 24 | 39 | 13 | 18 | 46 | 11 | 26 | 48 |
| | 6 | 10 | 32 | 5 | 4 | 47 | 41 | 20 |

This project has simulated the DES IC in software based on the implementation described in Section 3.1. The simulation of DES IC is written as software using C language. It is based on DES open resource code by Christopher R. Hertel [1]. This simulation can perform scan test technique by injecting into the encryption flow. For example, attackers can stop at any clock cycle and then scan data into or out from scan chain. To simplify the attacking program, the DES IC simulation program has three input parameters (performing like interface parameters in hardware): scan in data, start cycle and clock cycles. In the following example of input parameters:

$$000000AA000000AA \quad 5 \quad 8$$

The first parameter represents the scanned-in bit stream (in hexadecimal), which is the plaintext or intermediate ciphertext (64 bits) in DES encryption. These 64 bits will load into Left and Right Register. To simplify the calculation, scanning data into the scan chain is modified to only scanning Left and Right Register because other registers are useless using scanned-in data. The second parameter is starting cycle of DES IC, which means the first parameter, scanned-in bit stream, will be loaded into its corresponding register at the beginning of Cycle 5 (before Round 2). The third parameter represents the clock cycles the IC will operate after running at start cycle. So the IC will operate 8 cycles after scanning data at Cycle 5, and it will stop at Cycle 13 (before Round 10). The output of the DES IC simulation will

be loaded out after Cycle 13. The above description is the simplification of scan test technique program on DES IC.

The method that how to determine the location of scan chain has already been shown in Section 3.2.1 in details. A specific attacking case on DES is to focus on the recovering $RK$s and User Key.

This DES IC has a fixed User Key embedded inside, which is 2960375 (in ASCII). Although the key size of DES is 64-bit, actually, only 56 bits are useful as positions 8, 16, 24, 32, 40, 48, 56, 64 are used for checking parity. The following steps are to describe how to recover this key. The variables in the steps are all corresponding to the variables noted in Figure 3.1.

**Recover** $RK_1$

1. Scan 0000000000000000 (in hexadecimal) starting at Cycle 0 and run 3 cycles. Then scan out the bit stream and divide it into Left Register bits and Right Register bits. After calculation, $l$ is 32-bit:

   00000000000000000000000000000000

   and $r$ is also 32-bit :

   00000000000000000000000000000000

2. Scan 0000000000000000 (in hexadecimal) starting at Cycle 0 and run 4 cycles. Scan out the bit stream and analyze it. The location of Right Register represents variable $e$. So after calculation, $e$ is:

   11000100011110111111110011010100

3. Calcalate variable $a$ based on $r$ using Equation 3.10. Then $a$ is 48-bit:

   000000000000000000000000000000000000000000000000

4. Calculate variable $d$ based on $l$ and $e$ using Equation 3.8. Then $d$ is:

   11000100011110111111110011010100

5. Calculate variable $c$ using $ReversePermutation$ table in Table 3.1. After permutation, c is:

   01001111011111110010001101100011

6. Variable $c$ can be divided into 8 groups and each group is represented by a decimal number. So it will be 4, 15, 7, 15, 2, 3, 6, 3. Each number it the output of its corresponding $SBox$.

7. Use the table of ReverseSBox in Table 3.3 and calculate variable $b$ from $c$. As described in Section 3.2.2, there are four hypotheses of variable $b$ because $SBox$ has a 6-bit input and 4-bit output. XOR $b$ with $a$ to get $RK_1$. So there are also four hypotheses of $RK_1$. Table 3.5 shows the four hypotheses of $RK_1$ after $ReverseSBox$. The number, which represents the 6-bit input of a $SBox$, will be shown in decimal format in following table.

Table 3.5: Hypotheses of $RK_1$ with first plaintext.

| Hypotheses | $RS1(4) \oplus a$ | $RS2(15) \oplus a$ | $RS3(7) \oplus a$ | $RS4(15) \oplus a$ |
|---|---|---|---|---|
| 1 | 2 | 0 | 22 | 30 |
| 2 | 7 | 9 | 3 | 11 |
| 3 | 32 | 62 | 62 | 48 |
| 4 | 41 | 43 | 47 | 35 |
| Hypotheses | $RS5(2) \oplus a$ | $RS6(3) \oplus a$ | $RS7(6) \oplus a$ | $RS8(3) \oplus a$ |
| 1 | 0 | 20 | 28 | 20 |
| 2 | 5 | 29 | 31 | 11 |
| 3 | 34 | 46 | 52 | 58 |
| 4 | 45 | 35 | 33 | 57 |

8. Repeat steps 1 to 6 only with changing scan in data to 0000AA000000AA00 (in hexadecimal). Because there is a $InitialPermutation$ before the first DES round, this step needs one more operation. Data 0000AA000000AA00 will be changed to 0000000044444444 (in hexadecimal).

9. Get variables $l$, $r$ and $e$ from scanned-out bit stream. Variable $l$ is still:

   00000000000000000000000000000000

   $r$ is:

   01000100010001000100010001000100

and $e$ is:

$$11001011100001101010101000111011$$

10. Calculate variables from $l$, $r$ and $e$. Get $d$ is:

$$11001011100001101010101000111011$$

$a$ is:

$$00100000100000100000100000100000100000100000100000$$

and $c$ is:

$$11110110000000001110001011001111$$

11. Divide $c$ into 8 groups and write each 4-bit group as a decimal format number: 15, 6, 0, 2, 12, 5, 9, 15.

12. Use the table of ReverseSBox in Table 3.3 and calculate variable $b$ from $c$. This time there are still four hypotheses of $RK_1$ from XORing $b$ with $a$. But some of them will overlap with the hypotheses in Table 3.6 and these number will be greyed.

Table 3.6: Hypotheses of $RK_1$ with second plaintext.

| Hypotheses | $RS1(15) \oplus a$ | $RS2(6) \oplus a$ | $RS3(0) \oplus a$ | $RS4(1) \oplus a$ |
|---|---|---|---|---|
| 1 | 2 | 0 | 10 | 24 |
| 2 | 11 | 17 | 13 | 17 |
| 3 | 56 | 62 | 38 | 58 |
| 4 | 41 | 59 | 47 | 35 |
| Hypotheses | $RS5(12) \oplus a$ | $RS6(5) \oplus a$ | $RS7(9) \oplus a$ | $RS8(16) \oplus a$ |
| 1 | 10 | 20 | 28 | 2 |
| 2 | 15 | 7 | 3 | 11 |
| 3 | 60 | 46 | 52 | 48 |
| 4 | 45 | 35 | 57 | 57 |

13. Repeat steps 7 to 11 only with changing scan in data to 000000AA000000AA (in hexadecimal). The variables calculated from scan chain are listed as follows. Variable $l$ is still:

$$00000000000000000000000000000000$$

$r$ is:

10001000100010001000100010001000

and $e$ is:

10001110000110011000101000011101

14. The variables analyzed from $l$, $r$ and $e$ are listed as follows. Variable $a$ is:

    01000101000101000101000101000101000101000101000101

    $d$ is:

    10001110000110011000101000011101

    and $c$ is:

    01101100011100010000010011011001

15. The hypotheses of $RK_1$ are still four, but only one will overlap with the greyed number in Table 3.6. These greyed numbers are the bits of $RK_1$. They are shown in Table 3.7.

Table 3.7: Hypotheses of $RK_1$ with third plaintext.

| Hypotheses | $RS1(4) \oplus a$ | $RS2(15) \oplus a$ | $RS3(7) \oplus a$ | $RS4(15) \oplus a$ |
|---|---|---|---|---|
| 1 | 5 | 9 | 7 | 1 |
| 2 | 2 | 0 | 18 | 8 |
| 3 | 59 | 37 | 47 | 35 |
| 4 | 44 | 38 | 62 | 58 |
| Hypotheses | $RS5(2) \oplus a$ | $RS6(3) \oplus a$ | $RS7(6) \oplus a$ | $RS8(3) \oplus a$ |
| 1 | 11 | 7 | 31 | 3 |
| 2 | 2 | 20 | 16 | 12 |
| 3 | 45 | 37 | 55 | 57 |
| 4 | 36 | 48 | 52 | 36 |

16. Write these greyed numbers in binary (each number is written in 6 bits) and connect them:

    000010000000101111100011101101010100110100111001

    This is $RK_1$.

**Recover $RK_2$** Recovering $RK_2$ uses the same steps with recovering $RK_1$ only with changing start cycle and clock cycle period. The start cycle is before Round 2 of DES encryption and clock period is 1 cycle. Using the detailed method in recovering $RK_1$, finally $RK_2$ is calculated as

$$001110111000001001010000101001110101111011000100$$

**Recover User Key** After recovering the first two Round Keys successfully, it is easy to determine the User Key using the method described in Section 3.2.3. From the Table 3.4, the User Key can be analyzed from $RK_1$ and $RK_2$, which is a 56-bit stream:

$$00110010001110010011011000110000001100110011011100110101$$

It is written in hexadecimal: 32393630333735. After looking up in ASCII table, these bit stream represents 2960375 in ASCII, and this is the User Key.

# Chapter 4

# Novel Attack on PRESENT

After simulating a scan based side-channel attack on DES, a novel attack on PRESENT is proposed in this section. This section is divided into five subsections, which describe how this novel attack is developed based the cryptographic algorithm and hardware implementation using scan chain technique.

## 4.1 PRESENT Algorithm

PRESENT is a ultra-lightweight block cipher, which is designed to have a moderate security level and constraint space environment for hardware device [18]. Therefore, it is necessary to do a research on scan based side-channel attack of this cipher.

As a block cipher, the block size is 64-bit and the key size is either 80-bit or 128-bit. This project just focuses on the version with 80-bit key. This 80-bit key will derive 32 64-bit Round Keys (RK) in encryption. PRESENT has 32 round operations during an encryption. The first 31 rounds contain three operations: $addRoundKey$, $sBoxLayer$ and $pLayer$. The last round only contains $addRoundKey$.

An overall description of PRESENT is shown in Figure 4.1. The structure of PRESENT algorithm is similar to DES, but the operations are simpler because PRESENT needs to have a fast performance in hardware while it saves more space for device.

The operation $addRoundKey$ is to XOR the 64-bit block current ciphertext with 64-bit $RK$. The operation $sBoxLayer$ is to divide current 64-bit ciphertext into 16

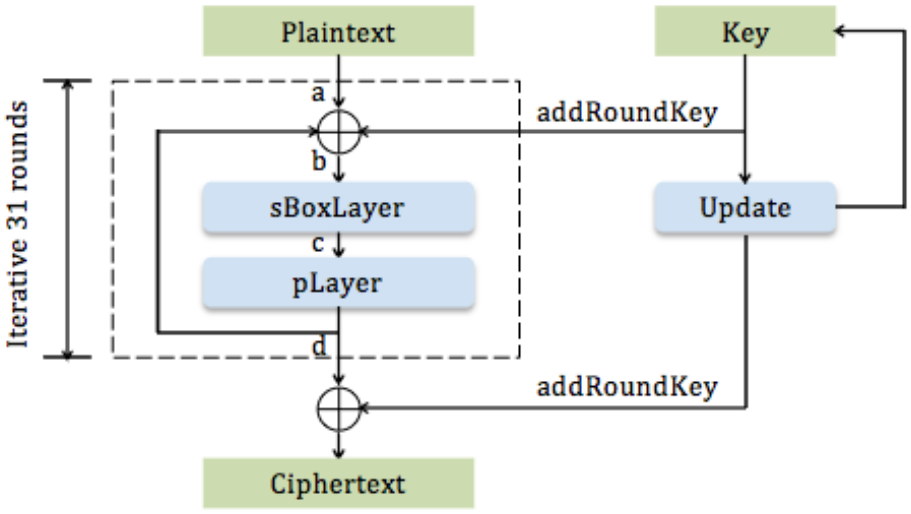


Figure 4.1: PRESENT Algorithm.

groups, and each group has 4 bits. The $SBox$es used in PRESENT are all 4-bit to 4-bit. So the output of $sBoxLayer$ is still 64-bit. The $SBox$ in hexadecimal notation is given in following Table 4.1. For example, if the input of $sBoxLayer$ is D571A0862153E43A (in hexadecimal), the output will be 70D5FC3A650B19BF.

Table 4.1: $SBox$ in PRESENT.

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $SBox[x]$ | C | 5 | 6 | B | 9 | 0 | A | D | 3 | E | F | 8 | 4 | 7 | 1 | 2 |

The operation $pLayer$ is a 64-bit permutation of the input. Similar to the permutation in DES, pLayer permutes each bit to a new location. It is shown in Table 4.2. For example, if the input of $pLayer$ is 70D5FC3A650B19BF (in hexadecimal), the output will be 2D17BCC18B93BA5F.

The RK generation of PRESENT is quite different from DES RK generation. The User Key is 80-bit, but $RK$s are all 64-bit. The generation is listed as follows:

1. Input 80-bit User Key into a Key Register ($K$).

Table 4.2: *pLayer*.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 48 | 52 | 56 | 60 |
| 1 | 5 | 9 | 13 | 17 | 21 | 25 | 29 | 33 | 37 | 41 | 45 | 49 | 53 | 57 | 61 |
| 2 | 6 | 10 | 14 | 18 | 22 | 26 | 30 | 34 | 38 | 42 | 46 | 50 | 54 | 58 | 62 |
| 3 | 7 | 11 | 15 | 19 | 23 | 27 | 31 | 35 | 39 | 43 | 47 | 51 | 55 | 59 | 63 |

2. Extract the 64 leftmost bits from Key Register and make them be $RK_1$, which is $K_{79}K_{78}...K_{16}$.

3. Update the $K$ is three steps:

   (a) Rotate the Key Register 61 bits to left: $K_{79}K_{78}...K_1K_0 = K_{18}K_{17}...K_{20}K_{19}$.

   (b) Pass first 4 bits to $SBox$: $K_{79}K_{78}K_{77}K_{76} = SBox[K_{79}K_{78}K_{77}K_{76}]$.

   (c) There are 32 rounds in total, so Round Counter is 5-bit. XOR Round Counter variable with internal five bits: $K_{19}K_{18}K_{17}K_{16}K_{15} = K_{19}K_{18}K_{17}K_{16}K_{15} \oplus RoundCounter$.

4. Repeat step 2 and 3 until 32 $RK$s are generated.

## 4.2 Implementation of PRESENT IC

PRESENT is a block cipher, which has 32 rounds, so PRESENT IC has an iterative structure. From the DES IC Implementation in Figure 3.2, the implementation of PRESENT IC also has two parts: registers and operations. After fixed operations in an encryption, data is loaded into registers and repeat another round.

This PRESENT IC has an 80-bit key stored securely inside the hardware device, and it can encrypt one block (64 bits) of plaintext to one block of 64-bit ciphertext using PRESENT algorithm.

The implementation of PRESENT IC is shown in Figure 4.2. The scan chain includes two registers: Counter Register (5 bits) and Data Register (64 bits). There is a Round Key Register, but it is not included in the scan chain. There are three operations in encryption flow: $AddRoundKey$, $SBoxLayer$ and $PLayer$. Tester can

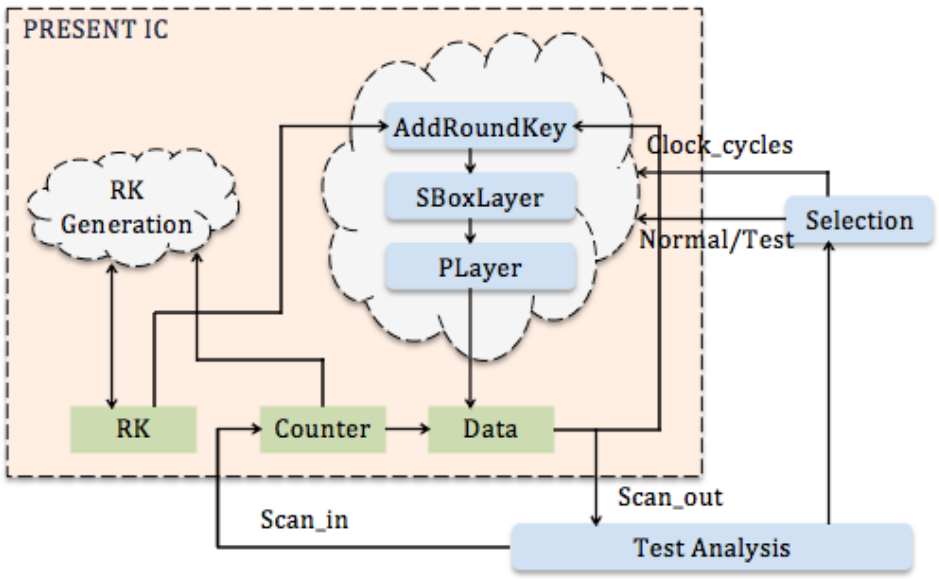


Figure 4.2: PRESENT IC Implementation.

select the clock cycles to control scan-in bit stream and observe scan-out bit stream. They can also switch IC between normal mode and test mode. The operations between RK and registers in a PRESENT round are described in following equations. The variables in these equations are illustrated in Figure 4.1.

$$b = a \oplus RK_i \tag{4.1}$$

$$c = SBoxLayer(b) \tag{4.2}$$

$$d = PLayer(c) \tag{4.3}$$

An entire encryption of PRESENT IC takes 33 cycles. They are listed as follows:

1. Cycle 1: Load plaintext into Data Register via scan chain.
2. Cycle 2 - 32: Load data into Data Register after Round 1- 31 (three operations in internal logic). The Counter Register is updated each clock cycle.
3. Cycle 33: Load ciphertext into Data Register after Round 32 (only one operation).

For example, testers switch PRESENT IC to normal mode and run 4 clock cycles. Then, the IC will stop after Round 3 in PRESENT encryption. Testers switch PRESENT IC to test mode. They can load the bit stream into or out from Counter Register and Data Register via scan chain. This is the method how test engineers use and attackers abuse.

## 4.3 Attack Algorithm on PRESENT

This section is to propose a new, valid scan based side-channel attack on PRESENT IC, which is the most novel and important part of the whole project. All the attack phases are newly developed in this project. There are three phases to mount an attack on PRESENT, which is similar to the attack on DES: determine the location of scan chain, recover round keys and recover user key.

Before a valid attack, the attackers need to think about what they know and what they are going to analyze. Therefore, this section is to give attack hypotheses based on the algorithm and hardware environment of PRESENT.

The hypotheses about what the attackers know are listed as follows:

1. PRESENT algorithm is known as it is public.
2. Attackers know the clock cycles of PRESENT IC (the corresponding relationship between cycles and rounds of PRESENT) provided by hardware manufacturers.
3. Attackers can infer the structure of PRESENT IC as an iterative implementation because this is a general structure for all cryptographic IC.

The hypotheses about what the attackers do not know are listed as follows:

1. Attackers do not know the exact number of registers in scan chain.
2. The RK register is not included in the scan chain, otherwise it is easy to determine RK via scan chain. For security, hardware manufacturers do not connect RK in scan chain for testing.

3. Although attackers can scan out bit stream from scan chain, they do not know the exact location of each DFF in scan chain. For example, if the scan chain has 85 bits in total, attackers do not know which 80 bits belong to data register and which 5 bits belong to counter register.

Given all the hypotheses, the attack is developed using three phases like the attack on DES. They are described in this following sections in details.

### 4.3.1 Determine location of Scan Chain

PRESENT is simpler in storing data. There are only Data Register, which stores 64 bits of intermediate data of an encryption flow, and Counter Register, which stores 5 bits of counter (from 0 to 31) as there are 32 rounds. After attackers scan out 69 bits from any clock cycle, they need to determine the location of Data Register and Counter Register in this bit stream.

**Data Register** Data Register contains 64 DFFs. They are the main part of PRESENT IC. They store all data bits which are processed in an encryption. As shown in Figure 4.2, the intermediate ciphertext will be stored in Data Register after $addRoundKey$, $sBoxLayer$ and $pLayer$ in the internal logic. How to determine Data Register of scan chain is shown in following steps:

1. Select normal mode, input plaintext 0000000000000001 (in hexadecimal) and run 1 cycle. Because the Counter Register just stores the current round in PRESENT encryption, the five DFFs in Counter Register are all 0. Therefore, in a 69-bit stream, only one location, which is 1 corresponding to a DFF, represents the LSB of Data Register.

2. Repeat step 1 63 times with changing the one block one bit different from previous plaintext. The range is from 0000000000000002 (in hexadecimal) to 8000000000000000 (in hexadecimal). Because this range has covered all bits of plaintext, which means all DFFs of Data Register are determined once, the locations of the DFFs in Data Register can be determined.

**Counter Register** Counter Register contains 5 DFFs. They are used to count the rounds of PRESENT operations. As PRESENT has 32 rounds in total, and the first 31 rounds include the same operations, the Counter Register can be designed at most 5 bits, which can count 32 times. Therefore, 5 DFFs are enough for Counter

Register in PRESENT IC. In addition, Counter Register performs as an important part in RK generation; it is necessary for attackers to determine the locations of Counter Register in the scan chain.

In the simulation of PRESENT IC in this project, the scan out data is only a 69-bit stream. So after determining 64 bits which are belonged to Data Register, the remaining 5 bits are belong to Counter Register. So the attackers only need to focus on these 5 bit locations. The steps that how to determine the exact locations of DFFs in Counter Register are shown as follows:

1. Select normal mode, input plaintext 0000000000000001 (in hexadecimal) and run 2 cycles (Round 1). In these five bits, only one bit should be 1 and others are all 0 because the counter is 00001 at present. So this bit should be LSB of Counter Register.

2. Repeat step 1 31 times and change the clock cycles from 3 cycles (Round 2), 5 cycles(Round 4), 9 cycles (Round 8) and 17 cycles (Round 16). The attackers can use the scan out bits to determine the remaining four locations of DFFs in Counter Register one by one. For example, after attackers scan out bit stream after 5 cycles, it means the PRESENT algorithm has already performed 4 rounds, so the counter must be 00100. Attackers can determine the third MSB in Counter Register according to the location of 1 in scanned out bit stream.

### 4.3.2 Recover Round Keys

Similarly to the attack on DES, the second phase of the scan based attack is to recover RKs in PRESENT. As described in Section 4.1, the encryption algorithm of PRESENT is simple and clear. There are 32 rounds in encryption and the first 31 rounds only have three operations: $addRoundKey$, $sBoxLayer$ and $pLayer$.

Attackers can scan out intermediate data from Data Register, whose location has been already determined in Section 4.3.1 using scan chain test technique. So in the Figure 4.1, the intermediate data $a$ and $d$ in the PRESENT algorithm can be determined from scan chain. From Equations from 4.1 to 4.3, the attackers know variable $a$ and $d$ from scan chain. Their goal is to recover $RK_i$. So they need to reverse $PLayer$ to get c, then reverse $SBoxLayer$ to get b. The last step is to calculate $RK_i$ by XORing variables $a$ and $b$. The analysis of the attack is listed as follows:

$$c = ReversePLayer(d) \tag{4.4}$$

$$b = ReverseSBoxLayer(c) \tag{4.5}$$

$$RK_i = a \oplus b \tag{4.6}$$

The *ReversePLayer* is similar to the *ReversePermutation* in the attack on DES IC. The reverse function is shown in Table 4.3. For example, if there is a bit stream, which is to input in this *ReversePLayer* table, the $6^{\text{th}}$ bit of the output is the $33^{\text{th}}$ bit of the input (the first location is regarded as $0^{\text{th}}$ bit).

Table 4.3: *ReversePLayer*.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 16 | 32 | 48 | 1 | 17 | 33 | 49 | 2 | 18 | 34 | 50 | 3 | 19 | 35 | 51 |
| 4 | 20 | 36 | 52 | 5 | 21 | 37 | 53 | 6 | 22 | 38 | 54 | 7 | 23 | 39 | 55 |
| 8 | 24 | 40 | 56 | 9 | 25 | 41 | 57 | 10 | 26 | 42 | 58 | 11 | 27 | 43 | 59 |
| 12 | 28 | 44 | 60 | 13 | 29 | 45 | 61 | 14 | 30 | 46 | 62 | 15 | 31 | 47 | 63 |

The *ReverseSBoxLayer* is much simpler than *ReverseSBox* in the attack on DES IC because the *SBoxLayer* in the PRESENT algorithm has the same length of input and output. In the *SBoxLayer* of PRESENT, the 64-bit input is divided into 16 groups, and each 4-bit group is input into a *SBox* and get a 4-bit output. Therefore, the *ReverseSBoxLayer* is only reversing the *SBox*, but the operation is still the same as *SBoxLayer*. The *ReverseSBoxLayer* is shown in following Table 4.4.

Table 4.4: *ReverseSBox*.

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $SBox[x]$ | 5 | E | F | 8 | C | 1 | 2 | D | B | 4 | 6 | 3 | 0 | 7 | 9 | A |

After the calculations are described, the attackers need to choose proper scan-in data and analyze scan-out data. The steps of recovering $RK$s are described as follows:

1. Load 69 bits (64 bits are for Data Register and remaining 5 bits are for Counter Register) of all 0 into scan chain in Cycle 1. Scan out the bit stream in the scan chain after Cycle 2. Then the scanned-in data includes variable $a$ and variable $d$ can be determined in scanned-out data.
2. Calculate variable $c$ from $d$ using Equation 4.4.
3. Calculate variable $b$ from $d$ using Equation 4.5.
4. Recover $RK_1$ from $a$ and $b$ using Equation 4.6.
5. Repeat the above four steps by changing clock cycles. Then all $RK_i$ can be recovered.

### 4.3.3 Recover User Key

As the key generation of PRESENT is very different from that of DES, the steps to recover User Key from Round Keys are more complex. This section is to give a detailed description that how to recover User Key by analyzing the relationship between User Key and Round Keys.

A detailed generation of $RK_1$ and $RK_2$ is listed as follows:

1. The User Key (80 bits) is stored in Key Register, which is written as: $K_{79}K_{78}K_{77}K_{76}...K_1K_0$.
2. After extracting the 64 leftmost bits from User Key, $RK_1$ (64 bits) is: $K_{79}K_{78}K_{77}K_{76}...K_{17}K_{16}$.
3. Update Key Register by following steps:
   (a) Rotate Key Register 61 bits to left, then the data in Key Register will be: $K_{18}K_{17}...K_1K_0K_{79}K_{78}...K_{20}K_{19}$.
   (b) Pass first 4 bits to $SBox$, then the data in Key Register will be updated as: $SBox[K_{18}K_{17}K_{16}K_{15}]K_{14}...K_0K_{79}K_{78}...K_{20}K_{19}$.
   (c) XOR Round Counter variable with internal five bits: $K'_{19}K'_{18}K'_{17}K'_{16}K'_{15} = K_{19}K_{18}K_{17}K_{16}K_{15} = K_{19}K_{18}K_{17}K_{16}K_{15} \oplus RoundCounter$. After previous step, these internal five bits are: $K_{38}K_{37}K_{36}K_{35}K_{34}$. So the Key Register will be updated as: $SBox[K_{18}K_{17}K_{16}K_{15}]K_{14}...K_0K_{79}K_{78}...K'_{38}K'_{37}K'_{36}K'_{35}K'_{34}...K_{20}K_{19}$.

4. Extract 64 leftmost bits from the Key Register to be $RK_2$: $SBox[K_{18}K_{17}K_{16}K_{15}]K_{14}...K_0K_{79}K_{78}...K'_{38}K'_{37}K'_{36}K'_{35}$.

From the description above, it is shown that $RK_1$ only does not contain rightmost 16 bits: $K_{15}K_{14}...K_1K_0$, but they are all included in $RK_2$. Therefore, only recovering the first two $RK$s is enough to recover User Key.

## 4.4 A Specific Attacking Case on PRESENT

After developing the algorithm that how to recover User Key in PRESENT IC, this project has also simulated the attack in the software program. All the algorithm and simulation in this section are firstly proposed in this project.

The simulation of PRESENT IC is written as software using C language. The implementation of PRESENT IC is based on PRESENT source code by Bo Zhu and Zheng Gong [2]. This simulation can perform scan test technique by injecting into the encryption flow like the attacking simulation on DES in Section 3.3. The attackers can decide when (clock cycle) they need to scan in and scan out intermediate data in the scan chain. They can control the bits they want to scan in and observe the bit stream from scan chain.

The simulation of attack on PRESENT is drawn ideas from attack on DES because they are similar attacking steps. Therefore, to make it simple and convenient for attacking program to access the software simulation of PRESENT IC using scan chain test technique, the implementation of PRESENT IC also gives out three parameters (perform like interface parameters in hardware): scanned-in data, start cycle and clock cycles. Besides, given these three parameters, the simulation of PRESENT IC must stop at certain clock cycles, and it will give out an output, which is the scanned-out bit stream in the scan chain. In this method, this simulation can still perform all functions of scan test technique. The simulation of PRESENT IC is more accurate than that of DES IC. The scanned-in data here is a 69-bit stream, which contains a 64-bit Data Register and 5-bit Counter Register. The attackers need to analyze all input bit stream especially Counter Register because it plays a very important role in generating Round Keys. If any intermediate bit scanned in by attackers is incorrect, the whole encryption will be mixed by one single wrong bit.

For example, there is a requirement that attackers need to scan in intermediate ciphertext A4519E846A83746B (in hexadecimal) before Cycle 5. After they determine the locations of scan chain, they should calculate the scanned-in bit stream of 69 bits. Despite the 64 bits of intermediate ciphertext, attackers need to know that Cycle 5 performs as Round 4. So the bits in Counter Register are 00100. Then the connection of the 64-bit data and 5-bit counter is an integrated scan chain.

A specific attacking case on PRESENT is shown in this section. There is a fixed User Key embedded inside this PRESENT IC, which is a software simulation of hardware device. The size of User Key in PRESENT is 80-bit, which is 012345678998765-43210 (in hexadecimal). The steps of the attack simulation are described in details. The variables in the following steps are all corresponding to the variables noted in Figure 4.1. Because the details that how to determine the location of scan chain has been described in Section 4.3.1, the attacking case will not analyze it again. This attacking phase is still simulated in this project.

1. Load data 000000000000000000 (in hexadecimal) into scan chain at the $0^{\text{th}}$ cycle. Although the data is 72-bit, actually the leftmost 3 bits are useless. Writing the 69-bit scan chain in hexadecimal is only convenient for programming. In the scanned-in 69-bit data, the leftmost 5 bits are for Counter Register and the remaining 64 bits are for Data Register. Run PRESENT IC for 2 cycles and get the output from scan chain, which is:

   000011001101101101101111000010110100000110010111101000101100110011001

   From the encryption cycles in Section 4.2, the data scanned in is before Round 1 in encryption flow of PRESENT algorithm, the right most 64 bits represent intermediate ciphertext after Round 1, which is variable $d$:

   1001101101101101111000010110100000110010111101000101100110011001

2. From Equation 4.4 and Table 4.3, variable $c$ is calculated as:

   1100010101101011100100001010110100111110111000111101101000001001

3. From Equation 4.5 and Table 4.4, variable $b$ is calculated as:

   000000010010001101000101011001111000100110011000011101100101010 0

4. As scanned-in bit stream to Data Register is all 0, variable $a$ is:

   0000000000000000000000000000000000000000000000000000000000000000000000

   $RK_1$ is:

   0000000100100011010001010110011111000100110011000011101100101010 0

5. Repeat step 1 to 4 only with changing the starting cycle to Cycle 2. Finally, $RK_2$ is recovered as:

   0011011001000010000000000010010001101000101011001111000100110011

6. This step is to use $RK_1$ and $RK_2$ to recover User Key. As shown in Section 4.3.3, $RK_1$ represents the $K_{79}K_{78}...K_{16}$ in User Key. So the leftmost 64 bits of User Key is:

   0000000100100011010001010110011111000100110011000011101100101010 0.

7. The remaining $K_{15}K_{14}...K_0$ can be inferred from $RK_2$. Send the leftmost 4 bits 0011 of $RK_2$ to $ReverseSBox$ shown in Table 4.4. Then $K_{18}K_{17}K_{16}K_{15}$ is recovered as: 1000. Then $K_{15}$ is 0 and $K_{14}K_{13}...K_{12}$ can be determined from $RK_2$ directly, which is 011001000010000.

8. Connect the recovered bits of User Key and get:

   00000001001000110100010101100111100010011001100001110110010101000011001000010000

   Write this 80-bit stream in hexadecimal: 01234567899876543210 and this is the User Key.

## 4.5 Attack Assessment

The scan based side-channel attack on PRESENT is firstly proposed in this project. The algorithm that how to recover User Key is based on the simulation of attack on DES in [6]. This attack does not use any special or professional tools or probes. All the attacks use a cryptographic hardware device, which has a fixed User Key inside, test chain technique and some attacking programs.

Based on preparing all scanned-in and scanned-out data, it only takes 3 seconds at most to recover the User Key in the attacking simulation of this project. In addition, this attack only requires scan test technique, so it is convenient for attackers to mount an attack in practical. Finally, it is less costly that attackers do not need to modify any hardware device because this attack is only based on a normal test method.

According to the analysis of the attacking method in this section, it shows that scan based side-channel attack is a great threat to PRESENT ICs.

# Chapter 5

# Countermeasures against Attack

It is vital to propose countermeasures against such attack after developing a novel scan based side-channel attack on PRESENT. Four countermeasures have been proposed in this project. The ideas of these countermeasures are all from some existing papers, but some places are modified to fit for PRESENT IC. In [11], the countermeasures are divided into two kinds: test mode protections and system mode protections.

The idea of test mode protections is to protect the scan chain technique from being abused by attackers. The basic method is that when a test engineer does not have a valid authentication, the data in the scan chain will be reset to 0 after switching IC from normal mode to test mode. So the intermediate data, which is processed in encryption, cannot be accessed by attackers. This kind of countermeasures transfers security risks to other aspects of cryptographic IC.

Despite attackers use scan test technique, they can still use some probes to access scan chain directly and recover the User Key by activating in system mode. So system mode protections have avoided such security risks by considering this problem. These countermeasures always modify scan chain and prevent the data from being accessed by some probes. Even some data is scanned out using probes, the bit stream has already been mixed by special methods.

This project has simulated four countermeasures. Two of them are test mode protections: Mirror Key Register (MRK) and Lock&Key Technique. The remaining two are system mode protections: Inverter Chain and XOR Chain. This section is to describe these countermeasures in details.

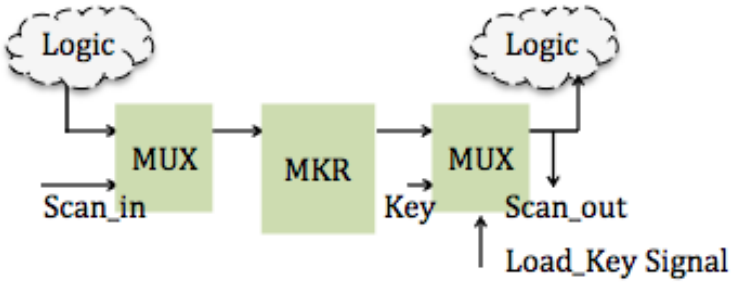


Figure 5.1: MRK Structure.

## 5.1 Protections in Test Mode

Although these two countermeasures have modified the architecture of IC board a lot, the testability is still reasonable for test engineers.

### 5.1.1 Mirror Key Register

[9] proposed MKR prevention scheme in 2005, together with the scan based attack on AES. In this paper, the authors proposed that there are two modes in this DFT architecture; one is insecure mode and the other is secure mode. In insecure mode, the cryptographic IC can work as usual in normal mode and test mode. However, in secure mode, the IC can only work in normal mode. Switching from secure mode to insecure mode would have a POWER-OFF reset. This prevents attackers to scan out the bit stream in secure mode.

This countermeasure is to use a Key Register in the scan chain, which is performed like a mirror. If the Load_Key signal is on, the key will be scanned into internal logic as normal. If the signal is off, the Mirror Key stored in MKR will be scanned into internal logic as a mix for encryption via a multiplexer. So this technique very depends on a single signal, and it needs to modify the IC board a lot. The algorithm of MRK is shown in Figure 5.1.

This project has simulated this prevention scheme for PRESENT IC. There is a MKR embedded in this hardware device, which stores a 69-bit fixed bit stream. These bits are randomly chosen by hardware manufacturers. When the test engineers fail authentication by scan test technique using some methods, the Load_Key signal will be switched off. The multiplexer in Figure 5.1 will choose MKR to scan into

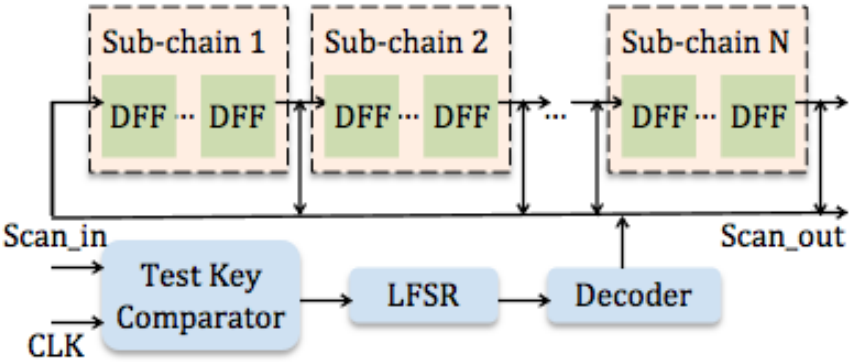


Figure 5.2: Lock&Key Structure.

internal logic. In this case, the bit stream scanned out from scan chain will be mixed by MKR.

### 5.1.2 Lock&Key Technique

This countermeasure is proposed in [12]. Lock&Key technique is to divide scan chain into smaller subchains. Test engineering need to scan in a valid key to the Test Key Comparator (TKC). After TKC confirms this key, it will be used as a seed for the next operation, Linear Feedback Shift Register (LFSR) [19]. The LFSR is operated as a pseudorandom selection of subchains. If the key is correct, the LFSR will use this key (seed) to generate some fixed sequence of numbers and send these numbers to Decode. Then Decode analyzes the sequence, gives the right order of subchains and connect them together as a whole chain. This technique is clearly shown in Figure 5.2.

This project has simulated Lock&Key Technique on PRESENT using software environment. The scan chain, which is 69-bit, is divided into 23 subchains and each chain includes 3 DFFs. The subchains can be correctly connected as a whole scan chain by the decode in a fixed order. The order determined by the decoder is from the LFSR operation. This simulation uses a pseudorandom function in software to perform as a LFSR and set a fixed sequence of subchains. If the key (seed) is correct, LFSR can connect the 23 subchains in a valid order to form a scan chain, and the test technique is used as normal.

There is fixed key and fixed sequence of subchains embedded in PRESENT IC in the simulation in this project. If attackers do not know the correct key, they will scan out a disordered sequence of subchains and fail to mount a valid attack. Otherwise, the PRESENT IC is still not secure if attackers know the correct key to access scan chain.

In 2011, [20] proposed a secure scan test technique, which is using Secure Scan Test Key Randomization (SSTKR). This prevention scheme is similar to Lock&Key technique, but less impact on area and test time. Both of these two countermeasures all use the property of pseudorandom generators: the produced numbers are following definite sequence with a settled seed.

## 5.2 Protections in System Mode

Protections in system mode mean that even attackers can use probe or special technique to inject into the scan chain, no information is leaked. The basic idea of these countermeasures is to mix the bit stream in the scan chain. Although bit stream is mixed by some modification of scan chain; test engineers can still test the IC board as usual.

### 5.2.1 Inverter Chain

Inverter Chain is to insert inverters between DFFs of registers, which is firstly proposed in [14]. The structure of this prevention scheme is shown in Figure 5.3. The inverters are inserted randomly by hardware manufacturer, so the data in the scan chain will be mixed after loading in or loading out data. The bits inside the registers and scan-in bits and scan-out bits are one-to-one corresponding. This project analyzes this technique in details and simulates it for PRESENT IC.

Because there are inverters inserted among DFFs in the scan chain, the scanned-in and scanned-out bits are different from that in the scan chain. Test engineering should recover the relationship between these bit stream. Without the Inverter Chain, the scanned-in or scanned-out bit streams are denoted as a vector space $\{X\}$. With the Inverter Chain embedded in the PRESENT IC, the above bit streams are denoted as a vector space $\{Y\}$. The logic of the testing steps is shown as follows:

1. Testers transform $\{X\}$ to $\{Y\}$.

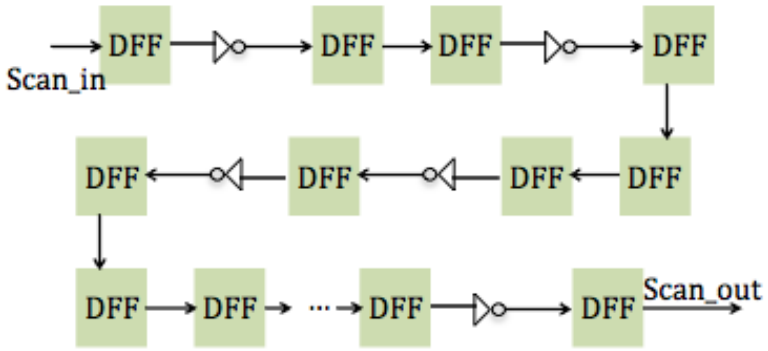


Figure 5.3: Inverter Chain Structure.

2. Scan bit stream $\{Y\}$ into scan chain.
3. After many inverters, the scanned-in bit stream is transformed to $\{X\}$ again.
4. After several operations in internal logic, the bit steam in scan chain is transformed to $\{X'\}$.
5. Scan bit stream $\{X'\}$ out from scan chain.
6. After many inverters, the scanned-out bit stream is transformed to $\{Y'\}$.
7. Testers transform $\{Y'\}$ to $\{X'\}$ again.

In the above description, Step 1 and Step 7 represent the additional logic in PRESENT IC: $ScanInReverse$ and $ScanOutReverse$ of this countermeasure respectively.

For example, there is a 7-bit scan chain, which has 7 DFFs. These DFFs are numbered in order: 0, 1, 2, 3, 4, 5, 6. There are 4 inverters inserted between some DFFs and the number of these numbers are: 0, 2, 5, 6. When test engineers want to scan in a bit stream, 1011001, to this scan chain. Besides, it is assumed that all DFFs store bit 0 at first. After scanning data in these DFFs, the bit stream stored in the scan chain is changed to 0111011 after 4 inverters.

Table 5.1 shows the process of DFFs' change and one-to-one correspondent relationship between scanned-in bits and the bits stored in DFFs. $In_i$ represents the i$^{th}$ bit of the input vector bit stream. $D_i{}^{j}$ represents the i$^{th}$ bit of DFF vector bit stream in j$^{th}$ clock cycle. $Inverter_i$ denotes whether there is an inverter before i$^{th}$ DFF: $Inverter_i = 1$ means there is an inverter, $Inverter_i = 0$ means there is no inverter. So the one-to-one correspondence is described in Equation 5.1 and 5.2. There are n DFFs in total.

$$D_1{}^{j} = In_{n-j} \oplus D_1{}^{j-1} * Inverter_1 \tag{5.1}$$

$$D_i{}^{j} = D_{i-1}{}^{j-1} \oplus D_i{}^{j-1} * Inverter_i \tag{5.2}$$

Therefore, the DFF No. in the first line denotes the No. of this DFF. The *Inverter* in the second line in this table means whether there is an inverter before this number of DFF. There is no inverter before when the number is 0. The Clock lines mean that the bit stored in its corresponding DFF after these clocks. Therefore, in this example, the scanned-in bits are: 1011001, the initial bits in the scan chain are: 0000000 and the bits after scanning in all 7 times are: 0111011.

Table 5.1: Inverter Chain - Scan In.

| DFF No. | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| *Inverter* | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| Initial | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Clock 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| Clock 2 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| Clock 3 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| Clock 4 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| Clock 5 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| Clock 6 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| Clock 7 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |

Scanning data out from Inverter Chain has the same analysis method like scanning in data. The one-to-one correspondence between the bits in the scan chain and output bits scanned out from scan chain is described in Equation 5.3 and 5.4. $Out_i$ represents the i$^{th}$ bit of the output vector bit stream. There are n DFFs in total.

$$D_i^{\text{j}} = D_{i-1}^{\text{j-1}} \oplus D_i^{\text{j-1}} * Inverter_i \tag{5.3}$$

$$Out_j = D_{n-1}^{\text{j-1}} \oplus D_n^{\text{j-1}} * Inverter_n \tag{5.4}$$

There is an example of corresponding relationship shown in Table 5.2. In this example, the initial bits stored in DFFs are: 0110100 and the bits after scanning out all 7 times are: 1010110 (the bits scanned from from No. 6 of DFFs).

Table 5.2: Inverter Chain - Scan Out.

| DFF No. | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| *Inverter* | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| Initial | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| Clock 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| Clock 2 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| Clock 3 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| Clock 4 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| Clock 5 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| Clock 6 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

However, all the analyses on Inverter Chain are based on knowing the exact positions of inverters. So except the testers, who have already known the positions, others are hardly to know the bits stored the DFFs when they scan out the bit stream.

The simulation of Inverter Chain prevention scheme on PRESENT uses the above analysis. Test engineers can determine the corresponding scanned-in or scanned-out data and data stored in DFFs. If attackers can know the positions of inverters, the attack is still successful. However, it is hard for attackers to mount a valid attack only by accessing scan chain.

In the simulation of Inverter Chain of PRESENT IC, there are 15 inverters inserted in the original scan chain. There is one inverter before these No. of DFFs: 2, 5, 8, 15, 16, 25, 30, 33, 39, 47, 51, 55, 56, 66 and 68. If input bit stream is:

000000011100001010110111111101000001010101101000001001000001110110011

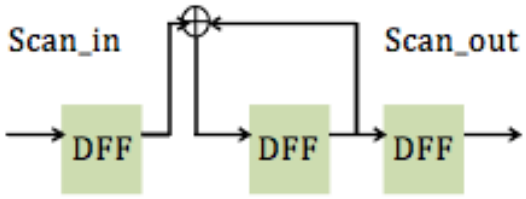


Figure 5.4: XOR Chain Structure.

the bits stored in scan chain will be:

0011100100111100010010000111011110010100100101100011101011100010010 10

If the bits stored in the scan chain is:

0101110001011001011010011110010001010111001000010000100111101001 1101

the output bit stream will be:

1001101101011000011010011001100000101001001000001110100011101001 1110

according to above analysis.

In this method, the additional functions $ScanInReverse$ and $ScanOutReverse$ are simulated and thus countermeasure of Inverter Chain is simulated in this project.

### 5.2.2 XOR Chain

XOR Chain is very similar to Inverter Chain, but only with changing inverters to XOR gates. This countermeasure was firstly proposed by [13]. This project analyzes it in details and simulates it for PRESENT IC. A novel calculation of correspondent relationship between bit stream scanned in or scanned out and bit stream in the scan chain is firstly described in this project. The correspondent operations $ScanInReverse$ and $ScanOutReverse$ are calculated as a chain. The structure of XOR gate part is shown in Figure 5.4.

Similarly to Inverter Chain, XOR Chain also needs additional logic $ScanInReverse$ and $ScanOutReverse$. The steps for test engineers to test PRESENT IC are the same as that described in Section 5.2.1.

The one-to-one correspondence between input bits and scan chain is described in Equation 5.5 and 5.6. The one-to-one correspondence between scan chain and output bits is described in Equation 5.7 and 5.8. Value $XOR_i$ in these equations denotes that whether there is a XOR gate before the i$^{th}$ DFF. Other variables are similar to the variables in Inverter Chain.

$$D_1{}^{\text{j}} = In_{n-j} \oplus D_1{}^{\text{j-1}} * XOR_1 \tag{5.5}$$

$$D_i{}^{\text{j}} = D_{i-1}{}^{\text{j-1}} \oplus D_i{}^{\text{j-1}} * XOR_i \tag{5.6}$$

$$D_i{}^{\text{j}} = D_{i-1}{}^{\text{j-1}} \oplus D_i{}^{\text{j-1}} * XOR_i \tag{5.7}$$

$$Out_j = D_{n-1}{}^{\text{j-1}} \oplus D_n{}^{\text{j-1}} * XOR_n \tag{5.8}$$

Table 5.3 and Table 5.4 are examples of a 6-DFF scan chain. The one-to-one correspondence can be clearly shown in these two tables. In Table 5.3, the bits in the scan chain is: 101111 after XOR gates. In Table 5.3, the bits scanned out from scan chain is: 100110.

Table 5.3: XOR Chain - Scan In.

| DFF No. | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $XOR$ | 0 | 1 | 0 | 1 | 1 | 0 |
| Initial | 0 | 0 | 0 | 0 | 0 | 0 |
| Clock 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| Clock 2 | 0 | 1 | 0 | 0 | 0 | 0 |
| Clock 3 | 0 | 1 | 1 | 0 | 0 | 0 |
| Clock 4 | 0 | 1 | 1 | 1 | 0 | 0 |
| Clock 5 | 1 | 1 | 1 | 0 | 1 | 0 |
| Clock 6 | 1 | 0 | 1 | 1 | 1 | 1 |

This project has simulated XOR Chain successfully. There are 15 XOR gates inserted in the original scan chain. There is one XOR gate before these No. of DFFs: 2, 5, 8, 15, 16, 25, 30, 33, 39, 47, 51, 55, 56, 66 and 68. If input bit stream is:

000000111000010101101111111010000010101011010000010010000011101100110

Table 5.4: XOR Chain - Scan Out.

| DFF No. | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $XOR$ | 0 | 1 | 0 | 1 | 1 | 0 |
| Initial | 0 | 1 | 1 | 0 | 1 | 0 |
| Clock 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| Clock 2 | 0 | 1 | 1 | 0 | 0 | 1 |
| Clock 3 | 0 | 1 | 1 | 1 | 0 | 0 |
| Clock 4 | 0 | 1 | 1 | 0 | 1 | 0 |
| Clock 5 | 0 | 1 | 1 | 1 | 1 | 1 |

the bits stored in the scan chain will be:

000001111000110000100100100010010101110110111001100001111011110000111

If the bits stored in scan chain is:

010111000101100101101001111001000101011100100001000010011111010011101

the output bit stream will be:

101101000010101100011101111111111001000110101101011111111011000101100

according to above analysis.

Then the additional functions $ScanInReverse$ and $ScanOutReverse$ are simulated and thus countermeasure of XOR Chain is simulated in this project.

# Chapter 6

# Further Attack Mechanism and Comparison

Recently, some existing countermeasures against scan based side-channel attacks are proposed. However, none of these prevention schemes are widely accepted by hardware manufacturers. Although they can make the hardware device more secure to a certain degree, they may be not practical to have a mass production. This section is to mount a reset attack against Inverter Chain prevention technique first. Some comparisons among these countermeasures against attacks on PRESENT ICs will be drawn from both efficiency and security views.

## 6.1 Reset Attack against Inverter Chain

Although [14] proposed the Inverter Chain countermeasure, which is a very important milestone in giving a system protection scheme against scan based side-channel attacks, [13] has given some basic analysis that there are some drawbacks in this countermeasure. In [13], the authors described that there is a reset attack against such Interter Chain. This project has analyzed and simulated this attack, which is shown in Figure 6.1. The detailed attacking phases are proposed in this section.

A reset attack means that attackers can determine the locations of inverters in the scan chain by resetting all DFFs to 0. The algorithm is described as follows:

1. Reset all DFFs to 0. which means each DFF in scan chain stored bit 0.

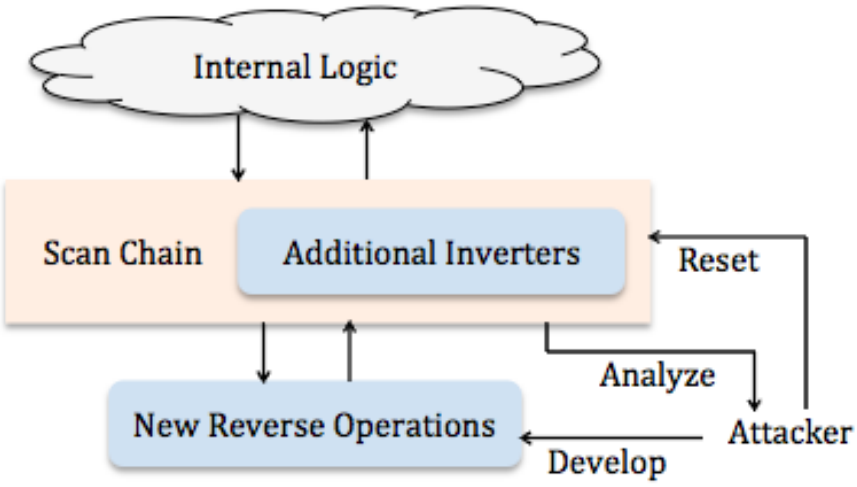


Figure 6.1: Reset Attack on Inverter Chain in PRESENT IC.

2. Scan out these bits from scan chain. The scanned-out bit stream should be changed to a certain bit vectors because the fixed inverters inserted before some DFFs.

3. The one-to-one correspondence between bits in scan chain and scanned-out bits are shown in Equation 5.3 and 5.4. After resetting all DFFs, the initial bit of variables $D_i^0$ are all set to 0.

4. The following Equation 6.1 is drawn from above analysis:

$$Out_j = \sum_{i=j+1}^{n} Inverter_i \tag{6.1}$$

   For example, in a 69-DFF register (the DFFs are numbered from 0 to 68), the $34^{\text{th}}$ bit in the scanned-out bit stream is: $Out_{34} = Inverter_{35} \oplus Inverter_{36} \oplus Inverter_{37} \oplus ... \oplus Inverter_{68}$.

5. By analyzing the equations, the $Inverter_i$ is calculated in Equation 6.2.

$$Inverter_i = Out_{i-1} \oplus Out_i \tag{6.2}$$

This project has simulated a successful reset attack against PRESENT IC with Inverter Chain in the software environment. The DFFs store all 0s after resetting all the registers in the scan chain. Then the 69 bits in the scan chain are loaded out after some inverters:

110001110000000100000000011111000111111000000001111000010000000000110

After calculating using Equation 6.2, the Inverter Chain can be written as:

001001001000000110000000010000100100000100000001000100011000000000101

So the locations of inverters can be obtained from this scan, which is: 2, 5, 8, 15, 16, 25, 30, 33, 39, 47, 51, 55, 56, 66 and 68.

After determining the locations of inverters. The attackers can simulate the one-to-one correspondence described in Section 5.2.1. Then further scan based side-channel attack on PRESENT can be developed using the attacking phases described in Chapter 4.

## 6.2 Efficiency

All these four countermeasures have modified the IC boards a lot, so the efficiency must be influenced to some degrees. However, the cost of efficiency is increased; the testability should be ensured.

The PRESENT IC with MKR can be tested by activating Load_Key signal. The Lock&Key technique will connect subchains in a correct order when test engineers have a valid key (seed for LFSR). Inverter Chain and Chain prevention schemes can load in or out the original bit streams by using some additional reverse programs.

From the detailed analysis of countermeasures in Chapter 5, all these prevention schemes add some additional gates and modify the IC board. Although all these countermeasures modify a lot of IC boards, protection in test mode and protection in system mode modify different parts of IC. So scan chain test technique is influenced by different aspects.

The following Table 6.1 shows that the additional signal/logic, gates and registers that each countermeasure uses. It is shown in this table that all countermeasures develop additional signal/logic, which means all of them are timing overhead.

However, the additional signal and logic of MRK need to modify TAP board a lot while Lock&Key technique adds some new logic operations. [13] also argues that MRK prevention scheme is difficult to design into existing encryption flow of stream ciphers. The MRK will cost more because the modification of TAP is not practical. Lock&Key technique may cost additional space to perform. Compared these two protections in test mode, Lock&Key technique is more practical for hardware manufacturers to modify PRESENT board.

The operations $ScanInReverse$ and $ScanOutReverse$ in Inverter Chain and XOR Chain can be performed in the software environment. It is no need to cost more space for hardware device because these two operations are only designed for test engineers. However, although Inverter Chain and XOR Chain do not need additional registers, the scan chain is modified too much. Besides, it is costly to design the locations of inverter or XOR gates for different cryptographic ICs because hardware manufacturers should need more mathematic theory to prove the rationality of the fixed locations.

Considered time overhead and space overhead, only Lock&Key technique is practical for a mass production. Inverter Chain and XOR Chain are easy to produced but costly to design.

Table 6.1: Additional implementation.

| Countermeasure | Signal/Logic | Gates | Registers |
|---|---|---|---|
| MRK | Decoder Logic<br>Load_Key Signal | | Mirror Key Register |
| Lock&Key | Comparator<br>Decoder Logic | | Linear Feedback<br>Shift Register |
| Inverter Chain | $ScanInReverse$<br>$ScanOutReverse$ | Inverters | |
| XOR Chain | $ScanInReverse$<br>$ScanOutReverse$ | XOR Gates | |

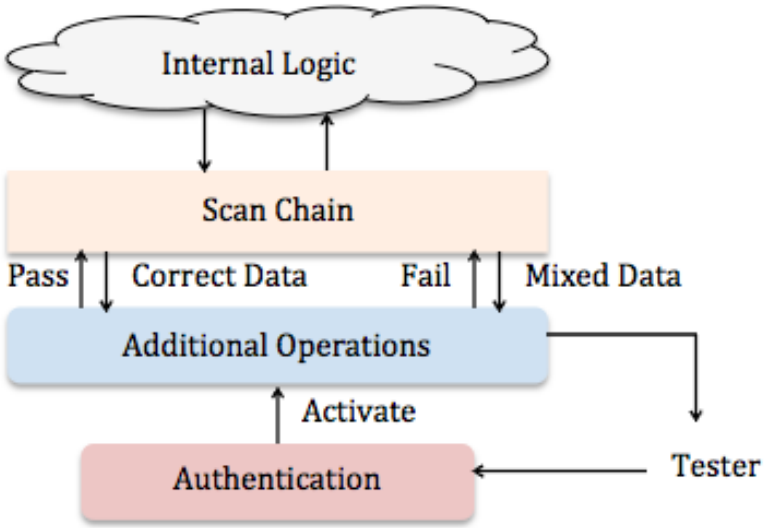


Figure 6.2: Secure Structure of Test Mode Protection.

## 6.3 Security

To improve the security of PRESENT IC, four countermeasures are simulated in this project. MKR and Lock&Key technique are belonged to protection in test mode. The modification is shown in Figure 6.2. It can be seen that these countermeasures transfer security risks to other authentication program. Inverter Chain and XOR Chain are belonged to protection in system mode. The structure is shown in Figure 6.3. The system protection countermeasures mix the data inside scan chain. So attackers must know additional reverse operations to recover the mixed bit stream and then mount an attack.

The security of MKR is low because this countermeasure just depends on a single signal. In addition, the attackers can still access scan chain using some special probes by breaking the whole PRESENT IC board. The design is basic to prevent the data in sensitive registers being accessed by attackers. Therefore, if the MKR is not as long as scan chain, it only protects parts of the scan chain. In [16], the authors use partial scan technique, which not inverts sensitive DFFs into the scan chain, but the User Key can still be retrieved using attacking tool on RSA algorithm. Therefore, even some sensitive data is mixed using MKR; it still has risks to recover User Key.

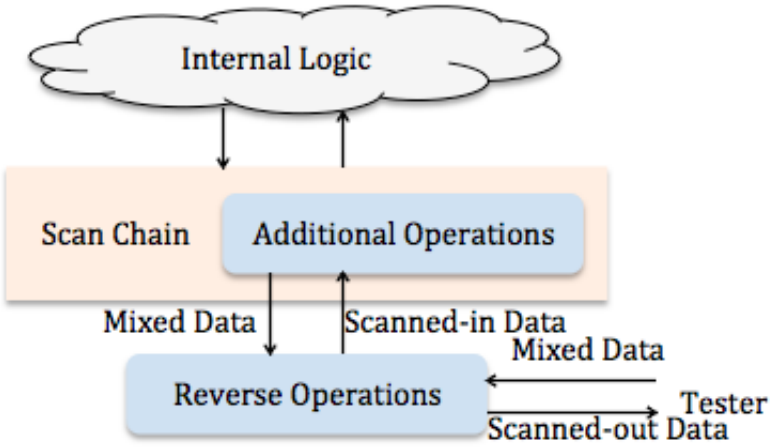


Figure 6.3: Secure Structure of System Mode Protection.

Compared with MKR, Lock&Key technique performs better in the aspect of security. Even [21] has proposed a semi-invasive attack against this countermeasure, it is secure from the view of non-invasive method. The scan chain mixed by connecting subchains using a pseudorandom function. Each PRESENT IC is secure with different, valid key (seed for LFSR). Even attackers break one PRESENT IC, they cannot recover other ICs with known this valid key. This means each PRESENT IC security is independent. However, this countermeasure still transfer security risk to Key Comparator operation. If attackers activate an attack on Key Comparator program, the scan based side-channel attack on PRESENT is still valid.

Inverter Chain is not secure as there has been already a reset attack on this countermeasure of PRESENT IC. Attackers can determine the locations of inverters and develop reverse operations to recover mixed bit stream into or from scan chain as shown in Figure 6.3. Attackers can perform like a legal test engineers to scan data in and out. This further attack on Inverter Chain is based on some mathematical analysis.

XOR Chain has not been attacked using mathematical analysis, but the locations of XOR gates are fixed in PRESENT IC. Therefore, there should be some relationship between additional operations and reverse operations. A successful attack can still be developed if attackers guess the locations of XOR gates. In addition, the locations of XOR gates should be different among many PRESENT ICs. Otherwise, if attackers

recover one reverse operation program, they can use it as a general attacking tool to recover all PRESENT ICs. So far, XOR Chain is still secured as a system mode protection.

According to the above analysis, Lock&Key technique is more secure as a countermeasure in test mode protection while XOR Chain performs more secure in system mode protection. Hardware manufacturers should take both efficiency and security into account from a balanced view. Therefore, there is still not a perfect countermeasure against attack on PRESENT after evaluating many prevention schemes. Lock&Key technique and XOR Chain are relatively more efficient than others. There should be still some drawbacks of these two countermeasures, and these two methods are protecting cryptographic ICs from different aspects. So this project gives more views for hardware manufacturers to consider the countermeasure they can take.

# Chapter 7

# Conclusion

## 7.1 Evaluation

In this project, the simulation of scan based side-channel attack on DES IC has been simulated successfully. This project has proposed a $ReverseSBox$, which can be used to recover Round Keys more rapidly. A specific attacking case on DES is described and simulated to show that how a successful attack is developed by analyzing some reverse operations using bit stream in the scan chain. These works are described in Chapter 3.

Since [6] proposed scan chain side-channel attack on DES in 2004, many researchers have realized this robust attacking method. Attacks on cryptographic ICs with different algorithms have been developed, such as AES, Trivium and RSA. A novel scan based side-channel attacking algorithm on PRESENT is firstly proposed in this paper. Some hypotheses about PRESENT IC are proposed before developing an attack. This attack determines the locations of registers in the first phase. Then it only uses two scanned-in bit streams to recover $RK_1$ and $RK_2$ as the second phase. The third attacking phase is to use the first two Round Keys to retrieve User Key embedded inside PRESENT IC. After giving the attacking algorithm, this project also simulates it and verifies the correctness of this attack. The PRESENT hardware device and the attack are all simulated in the software program using C language. The algorithms and implementations are analyzed in Chapter 4.

Then four countermeasures against attack on PRESENT are simulated based on some existing algorithm. In these simulations, some calculation methods are improved to fit for PRESENT IC, which are described in Chapter 5.

Finally, a reset attack on Inverter Chain is simulated, and some comparisons are drawn of the four countermeasures. It shows that there is a trade-off among efficiency, cost and secure. Hardware manufacturers should choose a relatively appropriate countermeasure to be against scan based side-channel attacks. These parts are analyzed in Chapter 6.

However, this project still not gives the most efficient countermeasure because there are limited time and not many professional tools. All the countermeasures are simulated in software, actually they need to be implemented in hardware, so it is difficult to compare the efficiency, cost, etc.

As PRESENT is a very useful cipher for cryptographic ICs, the designers of such PRESENT ICs should pay more attention to the robust attack proposed in this paper. The attack will give warnings for the PRESENT IC designers that they will have some drawbacks in the productions from the view of scan chain test. In addition, the countermeasures simulated and compared in this project can give useful ideas for hardware manufacturers to study.

So this project has its particular value in hardware security, especially the cipher fit for hardware environment. This project will be meaningful and helpful to draw more attention on scan based side-channel attacks. It can be used as practical guidance to design cryptographic ICs.

## 7.2 Future Work

This paper gives a detailed description about the attacking phases on PRESENT. More attacks can be developed on other cryptographic ICs with different algorithms using scan test technique according to the methods proposed in this paper. Additionally, some further work can give a more effective countermeasure in PRESENT ICs while implementing the prevention schemes in hardware device. This is the further work for the theory researchers.

This project is a mixed type with novel algorithm and simulation of hardware. Therefore, it is a challenge to apply theory to practice. This is the future work for hardware manufacturers.

# Bibliography


[1] C. R. Hertel. Des.c. `https://dev.mobileread.com/trac/iliados/browser/upstream/nbtquery/src/Auth/DES.c`, 2003.

[2] B. Zhu and Z. Gong. Present.c. `http://cis.sjtu.edu.cn/index.php/Software_Implementation_of_Block_Cipher_PRESENT_for_8-Bit_Platforms`, 2009.

[3] F. Standaert. Introduction to side-channel attacks. In *Secure Integrated Circuits and Systems Integrated Circuits and Systems*, pages 27–42, 2010.

[4] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: a reality today, a challenge tomorrow. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, pages 191–206, 2010.

[5] Data breach investigations report. `http://www.verizonbusiness.com/resources/reports/rp_data-breach-investigations-report-2012_en_xg.pdf`, 2012.

[6] B. Yang, K. Wu, and R. Karri. Scan-based side-channel attack on dedicated hardware implementations on data encryption standard. In *Proceedings of the International Test Conference on International Test Conference*, pages 339–344, Washington, DC, USA, 2004.

[7] K. P. Parker. *The Boundary-Scan Handbook*. Kluwer Academic, 101 Philip Drive, Assinippi Park, Norwell Massachusetts, 1992.

[8] Security requirements for cryptographic modules. Federal Information Processing Standards Publication FIPS PUB 140-2, 2002.

[9] B. Yang, K. Wu, and R. Karri. Secure scan: A design-for-test architecture for crypto chips. In *Proceedings of the 42nd annual Design Automation Conference*, pages 135–140, New York, USA, 2005.

[10] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side channel cryptanalysis of product ciphers. *Journal of Computer Security*, 8(2-3):141–158, 2000.

[11] D. Hely, M. L. Flottes F. Bancel, and B. Rouzeyre. Secure scan techniques: a comparison. In *Proceedings of the 12th IEEE International Symposium on On-Line Testing*, pages 119–124, Washington, DC, USA, 2006.

[12] J. Lee, M. Tehranipoo, C. Patel, and J. Plusquellic. Securing designs against scan-based side-channel attacks. *IEEE Transactions on Dependable and Secure Computing*, 4(4):325–336, October 2007.

[13] M. Agrawal, S. Karmakar, D. Saha, and D. Mukhopadhyay. Scan based side channel attacks on stream ciphers and their counter-measures. In *Proceedings of the 9th International Conference on Cryptology in India: Progress in Cryptology*, pages 226–238, Berlin, Heidelberg, 2008.

[14] G. Sengar, D. Mukhopadhyay, and D. R. Chowdhury. Secured flipped scan-chain model for crypto-architecture. *Computer-Aided Design of Integrated Circuits and Systems*, 26(11):2080–2084, 2007.

[15] M. Inoue, T. Yoneda, M. Hasegawa, and H. Fujiwara. Partial scan approach for secret information protection. In *Test Symposium, 2009 14th IEEE European*, pages 143–148, May 2009.

[16] J. D. Rolt, A. Das, G. D. Natale, M. Flottes, B. Rouzeyre, and I. Verbauwhede. A new scan attack on RSA in presence of industrial countermeasures. In *Constructive Side-Channel Analysis and Secure Design*, pages 119–124, Darmstadt, Germany, May 2012.

[17] Data encryption standard. Federal Information Processing Standards Publication FIPS PUB 46, January 1977.

[18] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In *Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, pages 450–466, Berlin, Heidelberg, 2008.

[19] W. A. Wijesinghe, M. Jayananda, and D. U. Sonnadara. Hardware implementation of random number generators. In *Proceedings of the Technical Sessions*, pages 28–38, 2006.

[20] M. A. Razzaq, V. Singh, and Adit Singh. Sstkr: Secure and testable scan design through test key randomization. In *Test Symposium (ATS), 20th Asian*, pages 60–65, 2011.

[21] S. P. Skorobogatov. *Semi-Invasive AttacksA New Approach to Hardware Security Analysis*. Phd dissertation, Cambridge, April 2005.

# Source Code

## Attack on DES

```
/*
 * Function: Reverse SBox function.
 *
 * Input: a string of output variable of SBox layer.
 * Output: a array of four hypothesis keys.
 *
 * Description: Give all the hypothesis of the keys from output of
 * s-boxes. For each s-box, there are four possible inputs (hypothesis
 * key). Actually, only when a is all 0, the hypothesis are the RK.
 * */
void ReverseSBox(char *c, int K_g[8][4])
{
  char *c_4bits;                    /* 4 bits of output of one s-box. */
  char s_out[8][4] = {{0},{0},{0},{0},{0},{0},{0},{0}};
  int i, j;
  for(i = 0; i < 8; i++)
  {
    c_4bits = c + i*4;
    strncpy (s_out[i],c_4bits,4);   /* First 4 bits of c1 of s-box1. */
    int c_s_out = binToInt(s_out[i], 4);
    for(j = 0; j < 4; j++)/* Four hypotheses of RK1 from 8 s-boxes. */
      K_g[i][j] = S[i][c_s_out][j];
  }
}



/*
 * Function: Recover Round Keys.
 *
 * Input: three arrays of three groups hypothesis keys
```

```
 * Output: a string of real Round Key.
 *
 * Description: Given three groups of hypothesis keys, this function
 * will give real RK.
 * */
void RecoverRK(char *K, int K_g1[8][4], int K_g2[8][4], int K_g3[8][4])
{
  int K_g[8] = {0};/* Hypotheses of RK, after analyzing plaintexts. */
  char k_bit[6] = {0};
  int i, i1, i2, i3;
  for(i = 0; i < 8; i++)
  {
    int stop_flag = 0;
    for(i1 = 0; i1 < 4; i1++)
    {
      int number2 = KXOR2[K_g1[i][i1]];
      for(i2 = 0; i2 < 4; i2++)
      {
        if(K_g2[i][i2] == number2)     /* Narrow key on plaintext 2. */
        {
          int number3 = KXOR3[K_g1[i][i1]];
          for(i3 = 0; i3 < 4; i3++)
          {
            if(K_g3[i][i3] == number3)/* Narrow key on plaintext 3. */
            {
              K_g[i] = K_g1[i][i1];
              stop_flag = 1;
              break;
            }
            if(stop_flag == 1)
              break;
          }
          if(stop_flag == 1)
            break;
        }
        if(stop_flag == 1)
          break;
      }
      if(stop_flag == 1)
        break;
    }
  }
}
```

# Attack on PRESENT

```
/*
 * Variable: This array simulates a scan chain.
 *
 * Description:
 * 0-63: data  register
 * 64-69: counter register
 * */
int register_position[69];



/*
 * Function (Simulation): Determine scan chain.
 *
 * Description: determine positions of data register in
 * scan chain.
 * */
void DetermineData()
{
  char result[69] = {0};
  char scan_in[40] = {0};
  int i;
  char bin[64][65];
  char hex[64][16];
  for(i = 0; i < 64; i++)
  {
    int j;
    for(j = 0; j < 64; j++)
      bin[i][j] = '0';
    bin[i][63-i] = '1';
    bin[i][64] = '\0';
    bitsToHex(hex[i], bin[i], 16);
  }
  for(i = 0; i < 64; i++)
  {
    strcpy(scan_in, "./PRESENT_oracle 00");
    strncat(scan_in, hex[i], 16);
    strncat(scan_in, " 0 1", 4);
    PRESENTOracle(result, scan_in);
    int j;
    for(j = 0; j < 69; j++)
    {
      if(result[j] == '1')
```

```
        register_position[63-i] = j;        // Determine Input register
    }
  }
}



/*
 * Function (Simulation): Reverse SBox function.
 *
 * Description: Given a string output of SBox layer this function will
 * give the input string corresponding to the output.
 * */
static void ReverseSBox( uint8_t *dst, const uint8_t *src )
{
  int i;
  for( i = 0; i < 8; i++ )
    dst[i] = 0;
  for( i = 0; i < 16; i++ )
  {
    int j;
    int Snum;
    int bitnum;
    /* Extract the 4-bit integer from the source.
     * This will be the lookup key within the ReverseSBox[i] array.
     */
    for( Snum = j = 0, bitnum = (i * 4); j < 4; j++, bitnum++ )
    {
      Snum <<= 1;
      Snum  |= GETBIT( src, bitnum );
    }
    if( 0 == (i%2) )
      dst[i/2] |= ((ReverseSbox[Snum]) << 4);
    else
      dst[i/2] |= ReverseSbox[Snum];
  }
}



/*
 * Function (Simulation): Recover round key.
 *
 * Description: Given a string of variable a and d, which are described
 * in Chapter 4, this function will give the Round Key string.
 * */
```

```
void RecoverRKey(uint8_t *key, const uint8_t *a, const uint8_t *d   )
{
  uint8_t b[8];
  uint8_t c[8];
  Permute( c, d, ReversePermutation, 8 ); /* Reverse permutation. */
  ReversesBox( b, c );
  XOR(key, a, b, 8);
}



/*
 * Function (Simulation): Attacking function.
 *
 * Description: This is the attacking function, which is the simulation
 * of attacking phases in software. It can recover User Key
 * successfully using scan based side-channel attack.
 * */
int main(void) {
  /*
   * Phase 1: Determine positions in scan chains
   *  0-63: data register
   * */
  int i;
  for(i = 0; i < 64; i++)
    register_position[i] = -1;
  DetermineData();
  /* Phase 2: Recover round key (RK1, RK2). */
  char scan_in[40] = {0};
  char a_hex[16] = {0};
  char d_bin[65] = {0};
  uint8_t a[8];        /* a = scan in data */
  uint8_t d[8];        /* d = Permute(c) */
  uint8_t RK1[8];      /* b = a + RK1 */
  uint8_t RK2[8];      /* b = a + RK2 */
  char K[21] = {0};
  char RK1_bin[64] = {0};
  char RK2_bin[64] = {0};
  char Key_bin[81] = {0};
  /* Recover RK1. */
  strcpy(scan_in, "./PRESENT_oracle 00000000000000000 0 2");
  RoundGetd(scan_in, d_bin);
  strcpy(a_hex, "0000000000000000");
  HexBytesToChar(a, a_hex, 8);
  bitsToChar(d, d_bin, 8);
  RecoverRKey(RK1, a, d);
  bytesToBits(RK1_bin, RK1, 8);
```

```
  printf("RK1: %s\n", RK1_bin);
  /* Recover RK2. */
  strcpy(scan_in, "./PRESENT_oracle 000000000000000000 2 1");
  RoundGetd(scan_in, d_bin);
  strcpy(a_hex, "0000000000000000");
  HexBytesToChar(a, a_hex, 8);
  bitsToChar(d, d_bin, 8);
  RecoverRKey(RK2, a, d);
  bytesToBits(RK2_bin, RK2, 8);
  printf("RK2: %s\n", RK2_bin);
  /* Phase 3: Recover key. */
  for(i = 0; i < 64; i++)
    Key_bin[i] = RK1_bin[i];
  char k_s[4] = {0};  /* Key in/out sBox. */
  for(i = 0; i < 4; i++)
    k_s[i] = RK2_bin[i];
  int k_s_i = bitsToInt(k_s, 4);
  k_s_i =  ReverseSbox[k_s_i];
  intToBits(k_s, k_s_i, 4);
  Key_bin[64] = k_s[3];
  for(i = 0; i < 15; i++)
    Key_bin[65+i] = RK2_bin[4+i];
  bitsToHex(K, Key_bin, 20);
  printf("Key(Hex): %s\n", K);
  return 0;
}
```