Scan-based Attacks against Cryptography LSIs and their Countermeasure

暗号LSIに対するスキャンベース攻撃とその対策技術に関する研究

2011年2月

早稲田大学大学院 基幹理工学研究科

情報理工学専攻 情報システム設計研究

Ryuta Nara

奈良 竜太

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Individual authentication increases in importance as network technology advances. IC passports, SIM cards, and ID cards used in entering and leaving management systems are dependent on their embedded LSI chips for keeping their security. These LSI chips achieve secure communication and reject counterfeit cards. The LSI chip usually includes cryptography circuits and encrypt/decrypt important data such as ID numbers and electronic money information.

There are symmetric-key cryptography and public-key cryptography as cryptography circuits inside LSI chips. Symmetric-key cryptography such as DES [4] and AES [5] are very popular and widely used. They make use of the same secret key in encryption and decryption. However, it may be difficult to securely share the same secret key, such as in communicating on the Internet. Public-key cryptography, on the other hand, makes use of different keys to encrypt and decrypt data so that it solves the key sharing problem of symmetric-key cryptography. One of the most popular public-key cryptography algorithms is RSA [6], which is used by many secure technologies such as secure key agreement and digital signature. An elliptic curve cryptography (ECC) [7, 8] is well known as a public-key cryptography with low cost and high throughput. Public-key cryptography is used by many security applications to achieve secure communication. However, there is a

threat that a secret key may be retrieved in cryptography LSI circuits.

A scan path is one of the most important testing techniques. In a scan path, registers inside an LSI are connected serially so that they can be controlled and observed directly from outside the LSI. Then test efficiency can be increased significantly. On the other hand, one can obtain register data easily by using a scan path, which implies that one can retrieve a secret key in a cryptography LSI. This is called a "scan-based attack", The scan-based attack is a method to retrieve a secret key from the scanned data obtained from the scan path in the cryptography LSI. The scan-based attack has attracted attention over the years as new side channel attack.

One of the difficulties in the scan-based attack is how to retrieve a secret key from scanned data obtained from a cryptography LSI. In a scan path, registers inside a circuit have to be connected so that its interconnection length will be shortened to satisfy timing constraints. This means that no one but a scan-path designer knows correspondence between registers and the scanned data. In order to succeed the scan-based attack against the cryptography LSI, an attacker needs to retrieve a secret key from the scanned data almost "randomly" connected. Yang et al. first showed a scan-based attack against DES in 2004 and retrieved a secret key in DES [9]. They also presented the scan-based attack against AES in 2006 [10]. Yang et al. proposes the method to make use of the hamming weight of the scanned data storing intermediate values during encryption/decryption. This is because this method only needs the correspondence between the scanned data and intermediate values, but does not need the bit-to-bit correspondence between them. To calculate the hamming weight of the scanned data for analysis, Yang's method needs to specify the position of registers storing the intermediate values so that Yang et al. needs to observe the change of the intermediate values.

However, Yang's method assumes the following two points below for the observation.

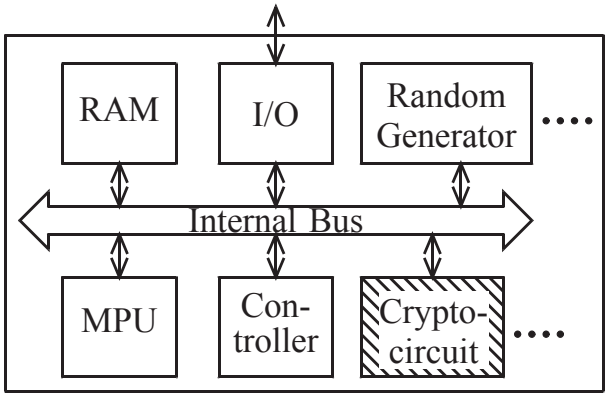

Figure 1.1: Architecture of a security LSI.

Assumption 1: A scan path does not include random elements caused by memories, processors, I/Os, and control circuits other than registers of cryptography circuits storing the intermediate values.

Assumption 2: An attacker knows when the registers store the intermediate values necessary for analysis.

An cryptography circuit is not implemented onto an LSI by itself. It is implemented onto an LSI with its memories, processors, I/Os, and control circuits (Figure 1.1). It is quite possible that a scan path includes random elements caused by memories, processors, I/Os, and control circuits other than registers of cryptography circuits storing the intermediate values. In that case, an attacker cannot know when the registers store the intermediate values necessary for analysis. If he/she do not have both Assumption 1 and Assumption 2 above, it is very hard to retrieve a secret key by using Yang's method.

In this dissertation, we propose new scan-based attacks against AES, RSA, and ECC. Our proposed new scan-based attack method against AES has an advantage that we do not need the correspondence between the scanned data and the registers of cryptography circuits storing the intermediate values, and further, we do not have to know when the registers store the intermediate values necessary for analysis. In addition to these advantages, we successfully reduce considerably the number of

input to retrieve a secret key. Therefore, our proposed scan-based attack against AES is more practical and powerful than Yang's method is. The algorithm of our scan-based attack can retrieve a secret key of RSA and ECC circuits as well as AES. Scan-based attacks against public-key cryptography have not been presented yet. We propose the world's first scan-based attacks against public-key cryptography circuits. RSA circuit is more complicated than AES one, and an ECC algorithm is much more complicated than the RSA one. However, our scan-based attacks are almost independent of a scan-path structure, so that we successfully retrieve secret keys inside them by using only 30 through 40 of input.

The purpose of our proposed attacking method is, not to make secure scan architecture ineffective but to retrieve a secret key using scanned data in an cryptography circuit with as few limitations as possible. In fact, our scan-based attack method without any modification might not work against cryptography LSIs using some secure scan architecture. However, some secure scan architecture cannot prevent from our proposed scan-based attacks. Sengar's secure scan architecture [2] inserts inverters between registers randomly in order to modify scanned data. Testers can do scan path test because they know positions of inserted inverters, but attackers do not know them and cannot turn modified scanned data back to normal. However, inserted inverters whether invert or not the scanned data, whose modified pattern of a particular register is only two. Because our proposed attacking method checks the same value corresponding the secret key whether exists or not in columns of the scanned data, Sengar's method is not effective against our proposed scan-based attacks. In order to prevent our proposed scan-based attacks from retrieving a secret key, we propose a new secure scan architecture named state-dependent configurable secure scan path. We insert some state-dependent scan flip-flops (SDSFFs) between registers. The SDSFF dynamically changes inverted positions so that our secure scan architecture prevents a secret key from scan-based attacks effectively.

This dissertation is organized as follows:

Chapter 2 [Scan-based Attack against AES] describes our proposed scan-based attack against AES. We explain the algorithm and the architecture of AES at first. AES is one of the most popular symmetric-key cryptography algorithm and it is used by many security applications. We introduce Yang's scan-based attack against AES as a conventional method. We propose the new algorithm of a scan-based attack against AES in this chapter. We experiment with scanned data obtained from AES C simulator. **Chapter 3 [Scan-based Attack against RSA]** describes our proposed scan-based attack against RSA. We explain the algorithm and the architecture of RSA. RSA is the first public-key algorithm in practical use, which is used by many secure technologies such as secure key agreement and digital signature. Second, we explain the correspondence between a secret key and intermediate values during decryption, and then we propose the world's first scan-based attack against RSA. We experiment with scanned data obtained from RSA C simulator. **Chapter 4 [Scan-based Attack against ECC]** describes our proposed scan-based attack against elliptic curve cryptography (ECC). We explain the algorithm and the architecture of ECC at first. ECC is one of the public-key algorithm, whose merit is superior cryptography strength. 160-bit key size of ECC is as the same security level as 1,024-bit key size of RSA. Second, we explain the algorithm to retrieve a secret-key from intermediate values during a scalar multiplication of ECC, and then we propose the world's first scan-based attack against ECC. We experiment with scanned data obtained from ECC gate-level simulator. **Chapter 5 [State-dependent Secure Scan Architecture]** describes our proposed secure scan architecture. Secure scan architecture is important technique in order to prevent scan-based attacks. We explain our proposed state-dependent scan flip-flops (SDSFFs) and the architecture of secure scan path using them. We experiment with state-dependent secure scan path in order to validate the effectiveness and area overhead. **Chapter 6 [Conclusion]** summarizes our research and indicates future works.

Chapter 2

Scan-based Attack againt AES

In this chapter, we propose a scan-based attack method against AES which retrieves a secret key, even if its scan path includes random elements other than the 128-bit register storing the round function output in AES. Our proposed method checks whether data specific to the secret key exists or not in the scanned data. Unlike Yang's method, our method is not influenced by other registers since we only focus on 1-bit register value. Then our method can attack an AES cryptography LSI, even if its scan path contains registers other then AES circuits.

If a secret key on an AES cryptography LSI is retrieved by using our proposed method, attackers may make a counterfeit smart card and steal money by using it. They also may have an unauthorized access to the Internet and do an expensive shopping. It is worth pointing out that there is vulnerability in the scan path of an AES-based cryptography LSI.

2.1 AES encryption algorithm

The Advanced Encryption Standard (AES) is a symmetric-key encryption standard announced in 2001 as FIPS PUB 197 [5] by National Institute of Standards and Technology (NIST). The AES is expected to completely replace the Data En-

cryption Standard (DES) [4], and it has been already used by many cryptography applications.

The AES algorithm encrypts and decrypts 128-bit data using a 128-bit, 192-bit, or 256-bit secret key. The AES algorithm is shown in Figure 2.1. See [5] in the detailed algorithm. After an input data is XORed to the initial round key (which is called the *pre-round function*), the AES algorithm encrypts it by applying the *round function* to it 10, 12, or 14 times, depending on the key bit length. The round function uses the round keys generated using the key expansion process as shown in Figure 2.2.

Assume that a secret key has a length of 128 bits. Hereafter, all the variables and round keys in the AES algorithm have a length of 128 bits. Each of the variables and round keys is shown by the 4×4 matrix, each of whose elements is 8-bit data. For example, let a be a 128-bit plaintext. Then it is composed of $a_{0,0}, a_{0,1}, \cdots, a_{3,3}$, each of which is an element of a which is a 8-bit data (or a byte data) as in Figure 2.1. In this chapter, similar notations will be applied to other variables and round keys.

By applying the key expansion process to the 128-bit secret key, we have the 11 round keys, $RK0, \cdots, RK10$, whereby the first one is the initial round key and each of the others are used for each of the 10 round functions. Each round function other than the final one is composed of SubBytes, ShiftRows, MixColumns, and AddRoundKeys and the final one is composed of SubByes, ShiftRows, and AddRoundKeys.

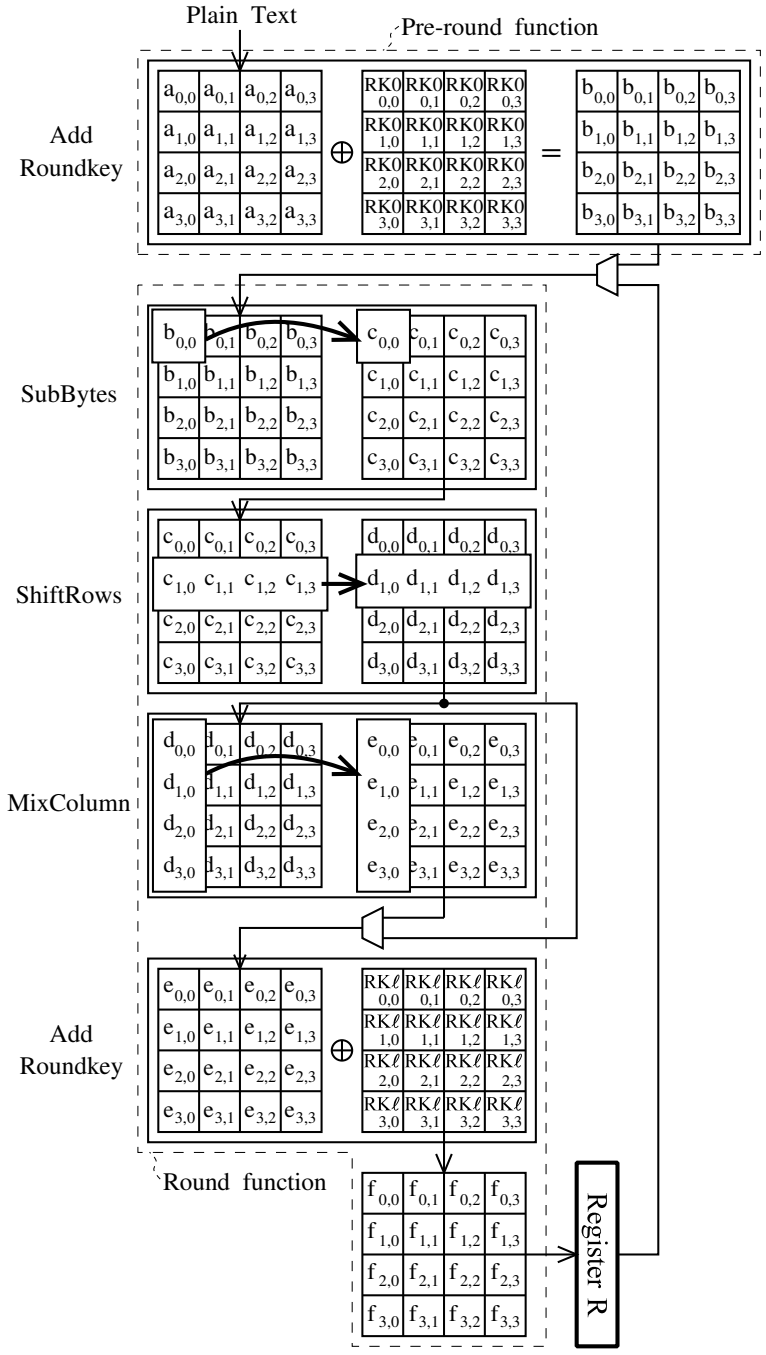

Figure 2.1: AES algorithm.

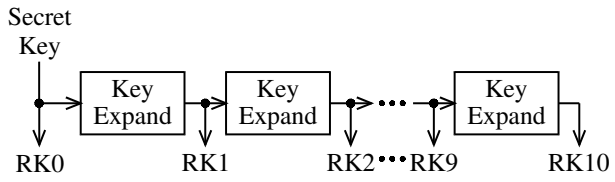

Figure 2.2: Key expansion process.

2.1.1 SubBytes: $c = SubBytes(b)$

The SubBytes function is a non-linear byte-wise substitution using the S-box which is expressed as follows:

1. Compute the multiplicative inverse in the finite field GF(2^8), whose irreducible polynomial is $m(x) = x^8 + x^4 + x^3 + x + 1$.

2. Apply the affine transformation over GF(2).

2.1.2 ShiftRows: $d = ShiftRows(c)$

In the ShiftRows function, each row in the input are cyclically shifted as follows:

1. **The first row** is not shifted,

2. **The second row** is shifted to the left by one byte,

3. **The third row** is shifted to the left by two bytes, and,

4. **The fourth row** is shifted to the left by three bytes.

2.1.3 MixColumns: $e = MixColumns(d)$

The MixColumns function is a column-by-column linear function whereby we consider each column in the input as a four-term polynomial over GF(2^8). It can be written as a matrix multiplication as in Eqn (2.1).

$$\begin{bmatrix} e_{0,k} \\ e_{1,k} \\ e_{2,k} \\ e_{3,k} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} d_{0,k} \\ d_{1,k} \\ d_{2,k} \\ d_{3,k} \end{bmatrix} \tag{2.1}$$

where $0 \leq k \leq 3$.

2.1.4 AddRoundKey: $f = AddRoundKey(e, RK\ell)$

In the AddRoundKey function, the input is XORed to the round key bit by bit.

2.2 Scan-based attack

The purpose of a scan-based attack is to retrieve the secret key from scanned data in a cryptography circuit. Throughout this chapter, we focus on AES as a cryptography. AES encryption repeats the round function 10 times when the size of a secret key is 128 bits. The round function uses the 11 round keys $RK0, RK1, \cdots, RK10$ which are generated from a secret key using the key expansion process as in Figure 2.2. If one of the 11 round keys is known, its secret key can be easily computed. A scan-based attack against AES retrieves one of the 11 round keys from scanned data and then computes the secret key using it.

As an AES cryptography, an architecture as in [11] can be used. Assume that 128-bit plaintext a is given. As in Figure 2.1, the result b is obtained by XORing a with the round key $RK0$ (pre-round function). The result goes into the 1st round function. The output of the 1st round function is stored into the 128-bit register R and it goes into the 2nd round function. When the secret key bit length is 128, the loop counts are 10.

In the AES cryptography LSI [11], the following three points are assumed:

1. Attackers can input any plaintext to the AES cryptography LSI.

2. Attackers can obtain scanned data from the scan path.

3. Connection sequence in the scan path is random and unknown.

In [12], two more points are assumed in addition to the points above, which is described in Section 2.3. The main contribution of this chapter is to propose a new scan-based attack which just assumes the above three points.

2.2.1 Scan path test

A scan path connects registers in an circuit serially so that a tester can observe the register values inside the circuit easily. The scan path is widely used in recent circuit implementations due to its testability and easiness.

Scan path test needs to replace standard flip-flops(FFs) with scan flip-flops(SFFs). An SFF usually consists of an FF and a multiplexer. The multiplexer output pin is connected to the FF input pin. It selects one from its two inputs. When the select line of the multiplexer is 0, it outputs the combinational circuits output. When the select line of the multiplexer is 1, it outputs the SFF output. A scan path model is shown in Figure 2.3. Control pin is used to choose between the system mode or the test mode. While Control pin is 0, normal operation is performed in the system mode as shown in Figure 2.4. While Control pin is 1, SFFs are connected serially and we obtain scanned data stored in each FF from the scan out as shown in Figure 2.5.

We call three operations at test mode as "Scan In", "Capture", and "Scan Out" and we can control and observe internal states of cryptography circuits through the scan path.

Scan In: Input test patterns to scan FFs inside circuits at test mode.

Capture: Operate normally at system mode.

Scan Out: Observe values of FF inside circuits at test mode.

Testers prepare test patterns for input and anticipated output patterns corresponds to input patterns. They input test patterns in "Scan In" at test mode, and operate circuits at one or some cycles in "Capture" at system mode, and obtain scanned data in "Scan Out" at test mode. If scanned data corresponds with anticipated output patterns, they find the circuits operates accurately.

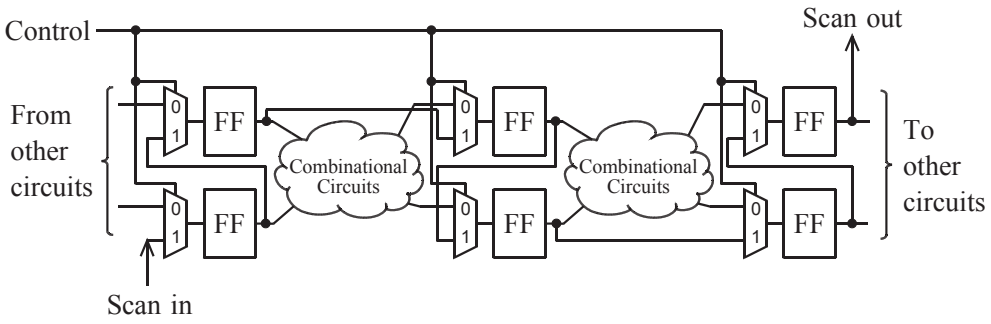

Figure 2.3: Scan path model.

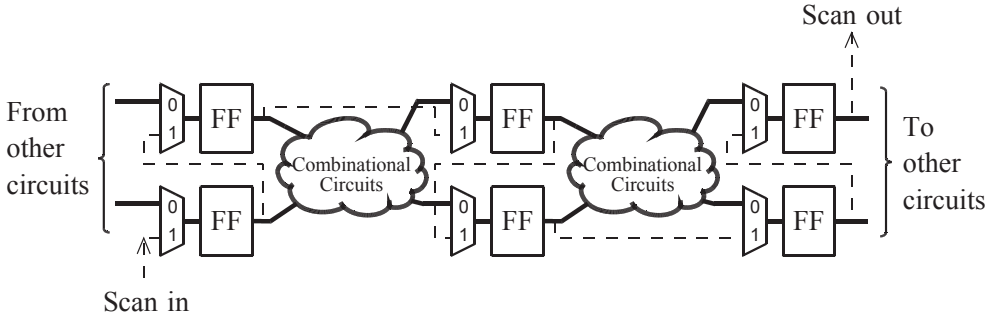

Figure 2.4: System mode (Control = 0).

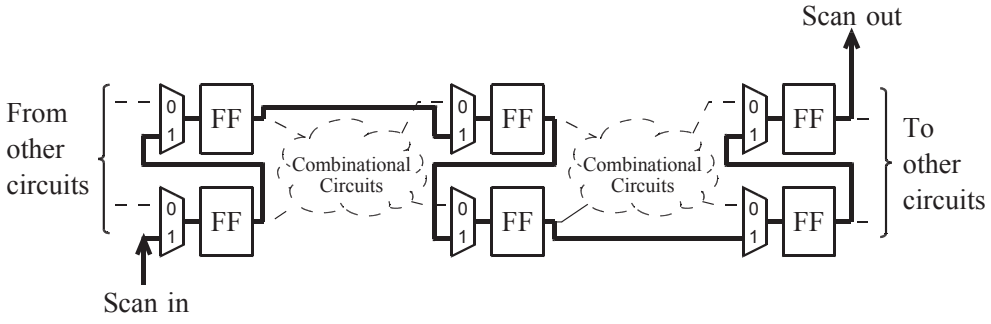

Figure 2.5: Test mode (Control = 1).

2.2.2 Problems to retrieve the round keys

Let a be a plaintext, $RK\ell$ be the round keys, $Round$ be the round function for 1st loop to 10th loop, and $Round\_final$ be the last round function, where ℓ in $RK\ell$ shows the loop count in the Figure 2.1. Then the round function output f_ℓ at Round ℓ is expressed as in Eqn. (2.2).

$$\begin{aligned} f_1 &= Round(RK1, a \oplus RK0) \\ f_\ell &= Round(RK\ell, f_{\ell-1}) \qquad (2 \leq \ell \leq 9) \\ f_{10} &= Round\_final(RK10, f_9) \end{aligned} \tag{2.2}$$

In Round 1, $a \oplus RK0$ shows the pre-round function. At Round 1, the round function output f_1 is given by a function of the plain text a, the round key $RK0$, and the round key $RK1$. At Round 2 or later ($2 \leq \ell \leq 10$), the round function output f_ℓ is given by a function of $f_{\ell-1}$ and the round key $RK\ell$.

As indicated above, f_ℓ is influenced by $RK0$, $RK1$, $\cdots$, $RK\ell$. This means that it is best for us to analyze the round function output f_1 at Round 1 because f_1 is given by $Round(RK1, a \oplus RK0)$, which only includes a, $RK0$, and $RK1$. Since the round function output f_1 is stored in the register R at several clocks after the plaintext is inputted, it may be obtained from the scanned data.

However, there are the following three problems to solve in order to retrieve the round key $RK0$ from the scanned data:

Problem 1: The round function output f_1 includes the two round keys $RK0$ and $RK1$.

Problem 2: The round key $RK0$ is a 128-bit data. The number of possible values of $RK0$ becomes 2^{128} and it is impractical to perform exhaustive search for all of them.

Problem 3: The bit-to-bit correspondence between the scanned data and the register R is unknown.

2.2.3 Eliminating the round key $RK1$ rrom the round function output f_1

In order to solve the Problem 1 in Section 2.2.2, the round key $RK1$ have to be eliminated from the round function output f_1. The AddRoundKey is the last sub-function of the 1st round function. In the AddRoundKey, its input e is XORed to the round key $RK1$ and then we have the round function output. Consider that, if the two identical values are XORed, we will have zero.

Let a^1 and a^2 be two plaintexts, e^1 and e^2 be their MixColumns outputs, f_1^1 and f_1^2 be the round function outputs, respectively. As shown in Eqn. (2.3), the result of $f_1^1 \oplus f_1^2$ is independent of the round key $RK1$. $e^1 \oplus e^2$ can be computed with only the two plaintexts a^1 and a^2, and the round key $RK0$, not using $RK1$.

$$\begin{aligned} f_1^1 \oplus f_1^2 &= (e^1 \oplus RK1) \oplus (e^2 \oplus RK1) \\ &= e^1 \oplus RK1 \oplus e^2 \oplus RK1 \\ &= e^1 \oplus e^2 \oplus RK1 \oplus RK1 \\ &= e^1 \oplus e^2 \end{aligned} \tag{2.3}$$

By using $e^1 \oplus e^2$, we can eliminate $RK1$ from the round function output.

2.2.4 Obtaining the data dependent on the single element in $RK0$

Problem 2 can be solved as follows: Let us consider to retrieve the round key $RK0$ from two plaintexts a^1 and a^2, and their XORed MixColumns outputs $e^1 \oplus e^2$.

First assume that it is possible to find anyhow the round function outputs f_1^1 and f_1^2 for the plaintexts a_1 and a_2, respectively, from the scanned data. Then, the simplest retrieving method is to perform exhaustive search for all possible values for $RK0$. However, the number of possible values of $RK0$ becomes 2^{128} and it is impractical to perform exhaustive search for all possible values for $RK0$.

This problem is solved by generating the data which only depends on a single element of $RK0$. Let a^1 and a^2 be two plaintexts which has the same values except for their first element $a^1_{0,0}$ and $a^2_{0,0}$ as in Eqn. (2.4):

$$a^1 = \begin{bmatrix} a^1_{0,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \; a^2 = \begin{bmatrix} a^2_{0,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.4}$$

If a_1 and a_2 defined above are given to the round function, we have the two round function outputs f^1_1 and f^2_1 at Round 1. As in Figure 2.6, only the first column of $f^1_1 \oplus f^2_1$ depends on $a^1_{0,0}, a^2_{0,0}$, and $RK0_{0,0}$. The other columns of $f^1_1 \oplus f^2_1$ becomes zero. The number of possible values of $RK0_{0,0}$ becomes at most $2^8 = 256$ and then it is very practical to perform exhaustive search for all possible values for $RK0_{0,0}$. All other elements of the round key $RK0$ can be retrieved in the same way.

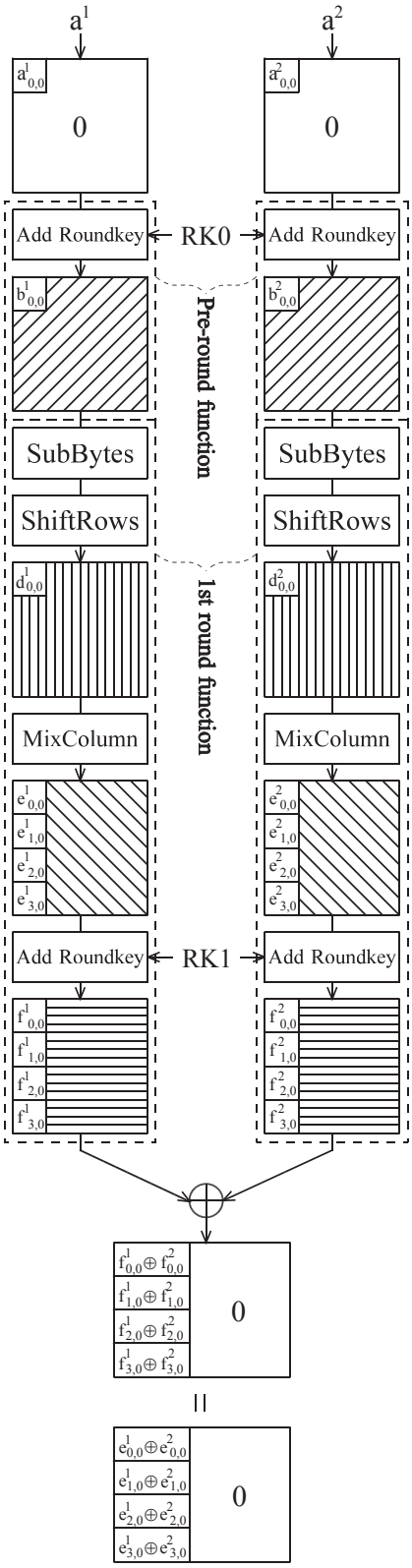

Figure 2.6: Data dependent on $a^1_{0,0}$, $a^2_{0,0}$, and $RK0_{0,0}$.

2.2.5 Correspondence between the round function output and the scanned data

In fact, Problem 3 is essential in the scan-based attack. In a scan chain, registers inside a circuit have to be connected so that its interconnection length will be short for satisfying constraints. This means that no one but a scan-chain designer knows correspondence between registers and scanned data. Attackers cannot compare the expected data with the scanned data and cannot retrieve the round key $RK0$.

Yang et al. proposed the solution for this problem as in Section 2.2.6

2.2.6 Yang's method

In order to solve Problem 3, Yang et al. proposes the method to make use of the hamming weight of the first column of $f_1^1 \oplus f_1^2$ to retrieve the round key $RK0$ [12]. This is because this method only needs the correspondence between the first column of f_1^1 or f_1^2 and the scanned data, but does not need the bit-to-bit correspondence between them.

Retrieving $b_{0,0}$ with the hamming weight

The output b of the pre-round function is obtained by XORing a plaintext a and the round key $RK0$. On the contrary, $RK0$ is calculated by $a \oplus b$. From the viewpoint of the first element $(0, 0)$, the following equation can be obtained:

$$\begin{aligned} a_{0,0} \oplus b_{0,0} &= a_{0,0} \oplus (a_{0,0} \oplus RK0_{0,0}) \\ &= (a_{0,0} \oplus a_{0,0}) \oplus RK0_{0,0} \\ &= RK0_{0,0} \end{aligned} \tag{2.5}$$

This means that, retrieving $RK0_{0,0}$ is equivalent to retrieving $b_{0,0}$. Now consider to retrieve $b_{0,0}$ from the round function output f_1 at Round 1.

First, consider the relation between b and the round function output f_1 at Round 1. Assume that b^1 and b^2 which has the same values except for their first

Table 2.1: Relation of the hamming weight of the first column of $f_1^1 \oplus f_1^2$ and $(b_{0,0}^1, b_{0,0}^2)$.

Hamming weight	$(b_{0,0}^1, b_{0,0}^2)$
9	226, 227
12	242, 243
23	122, 123
24	130, 131

elements $b_{0,0}^1$ and $b_{0,0}^2$ are obtained. Let us further assume that $b_{0,0}^1$ and $b_{0,0}^2$ satisfy Eqn. (2.6) below:

$$\begin{pmatrix} b_{0,0}^1 \\ b_{0,0}^2 \end{pmatrix} = \begin{pmatrix} 2m \\ 2m+1 \end{pmatrix} \text{ or } \begin{pmatrix} b_{0,0}^1 \\ b_{0,0}^2 \end{pmatrix} = \begin{pmatrix} 2m+1 \\ 2m \end{pmatrix} \tag{2.6}$$

where $0 \leq m \leq 127$. The round function outputs f_1^1 for b^1 and f_1^2 for b^2 can be calculated and then $f_1^1 \oplus f_1^2$ is obtained. As shown in Figure 2.6, the first column of $f_1^1 \oplus f_1^2$ depends on $b_{0,0}^1$ and $b_{0,0}^2$ and the other elements becomes zero.

Since the number of possible values of $(b_{0,0}^1, b_{0,0}^2)$ satisfying Eqn. (2.6) becomes merely $2^7 = 128$, all possible values of $(b_{0,0}^1, b_{0,0}^2)$ can be evaluated and the hamming weight of the first column of $f_1^1 \oplus f_1^2$ for each of them can be calculated. If and only if $(b_{0,0}^1, b_{0,0}^2)$ is $(226, 227)$, the hamming weight of the first column of $f_1^1 \oplus f_1^2$ becomes 9. Then, if the hamming weight of the first column of $f_1^1 \oplus f_1^2$ for two plaintexts a^1 and a^2 becomes 9, $(b_{0,0}^1, b_{0,0}^2)$ can be $(226, 227)$. There are four such cases which are summarized in Table 2.1.

Hence, there are still two problems remaining to be solved:

Problem (i): The correspondence between the first column of the round function output and the scanned data is unknown.

Problem (ii): Two plaintexts a^1 and a^2 are required such that, if the pre-round function is applied to them, $b_{0,0}^1$ and $b_{0,0}^2$ satisfy Eqn. (2.6).

Specifying the rirst column of the round function in the scanned data

In order to solve Problem (i), the 32 1-bit registers storing the first column of the round function output at Round 1 must be specified in the scanned data.

By applying the method described in the Section 2.2.4, the first column of the round function output depends on the plaintext element $a_{0,0}$. Several plaintexts which have the same values except for the element $a_{0,0}$ are inputted into the AES circuit and, after Round 1 is over, their scanned data sets are picked up and compared one another. If one of the scanned data is different from another one, $a_{0,0}$ causes the differences. The positions in which the bit data is different give the 32 1-bit registers storing the first column of the round function output at Round 1 in the scanned data, because their positions are the same even if the input is different.

Selection of the plaintext elements $a^1_{0,0}$ and $a^2_{0,0}$

Consider to solve Problem (ii). Let a^1 and a^2 be two plaintexts which have the same value except for their first elements $a^1_{0,0}$ and $a^2_{0,0}$ defined as in Eqn. (2.7) below.

$$\begin{pmatrix} a^1_{0,0} \\ a^2_{0,0} \end{pmatrix} = \begin{pmatrix} 2t \\ 2t+1 \end{pmatrix} \text{ or } \begin{pmatrix} a^1_{0,0} \\ a^2_{0,0} \end{pmatrix} = \begin{pmatrix} 2t+1 \\ 2t \end{pmatrix} \tag{2.7}$$

where $0 \leq t \leq 127$. If $a^1_{0,0}$ is XORed to $RK0_{0,0}$ and also $a^2_{0,0}$ is XORed to $RK0_{0,0}$, we can have the Eqn. (2.8) below regardless of the value in $RK0$:

$$\begin{aligned} \begin{pmatrix} a^1_{0,0} \oplus RK0_{0,0} \\ a^2_{0,0} \oplus RK0_{0,0} \end{pmatrix} &= \begin{pmatrix} 2m \\ 2m+1 \end{pmatrix} \text{ or } \begin{pmatrix} 2m+1 \\ 2m \end{pmatrix} \\ &= \begin{pmatrix} b^1_{0,0} \\ b^2_{0,0} \end{pmatrix} \end{aligned} \tag{2.8}$$

where $0 \leq m \leq 127$. Then the two plaintexts a^1 and a^2 are prepared which have the same value except for their first elements satisfying Eqn. (2.7).

Number of plaintexts to retrieve $RK0$ with Yang's method

Yang et al. shows that six plaintexts on average and 15 plaintexts in the worst case are required for specifying the 32 1-bit registers in the scanned data in Section 2.2.6. Since $RK0$ is composed of four columns, then $4 \times 6 = 24$ plaintexts on average and $4 \times 15 = 60$ plaintexts in the worst case are required. Let us consider how many plaintexts are required to retrieve $RK0$. The number of possible patterns for a^1 and a^2 which satisfies the discussion in Section 2.2.6 is 128. We have four hamming weight values of 9, 12, 23, and 24, which lead to retrieving $RK0$. Thus, the number of required plaintexts to retrieve one of the elements in $RK0$ becomes $128/4 = 32$ on average and $128 - 4 = 124$ in the worst case. Since the number of the elements in $RK0$ is 16, the number of plaintexts to retrieve all the elements in $RK0$ becomes $32 \times 16 = 512$ on average and $128 \times 16 = 1,984$ in the worst case. Overall, $24 + 512 = 536$ plaintexts on average and $60 + 1,984 = 2,044$ plaintexts in the worst case are required to retrieve $RK0$ by using Yang's method.

2.3 Proposed method

In addition to the assumptions described in the Section 2.2.1, Yang's method assumes the following two more points below when specifying the first column of the round function in the scanned data.

Assumption 1: A scan path does not include random elements caused by memories, processors, I/Os, and control circuits other than the register R storing the round function output.

Assumption 2: We know when the round function output at Round 1 is stored in the register R.

An AES circuit is not implemented onto an LSI by itself. It is implemented onto an LSI with its memories, processors, I/Os, and control circuits (Figure 2.7). It is quite possible that a scan path includes random elements caused by memories, processors, I/Os, and control circuits other than the register R storing the round function output.

Further, we cannot know when the round function output at Round 1 is stored into the register R. After a plaintext is inputted, we may require one clock cycle to obtain the round function output at Round 1 but may require several clock cycles to obtain it depending on the AES architecture. If we do not have both Assumption 1 and Assumption 2 above, it is very hard to retrieve a secret key by using Yang's method.

In this section, we propose a new scan-based attack method whereby we do not need the correspondence between the scanned data sd and the register R storing the round function output, and further, we do not have to know when the round function output at Round 1 is stored in the register R. In that sense, it is more practical and powerful attack than Yang's method.

Our method focuses on the round key element $RK0_{0,0}$ which only has 2^8 possible values. First, we assume one value k for $RK0_{0,0}$ and calculate specific informa-

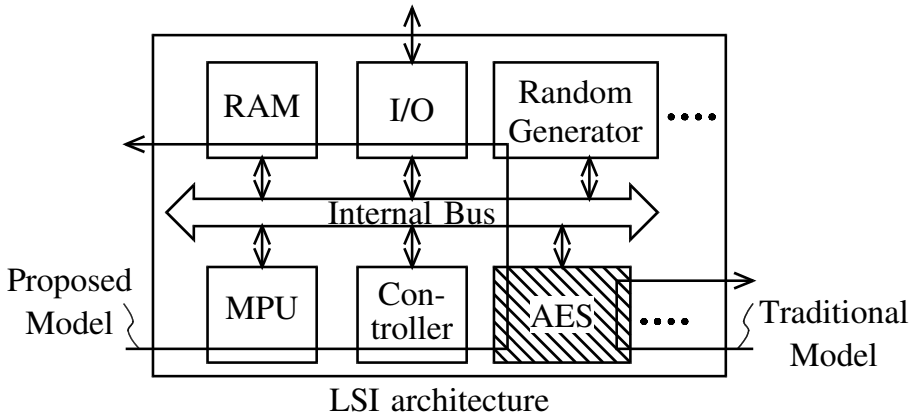

Figure 2.7: Scan chain model.

tion to k, which is called a *discriminator* to $RK0_{0,0}$. Second, we retrieve several scanned data $sd^0, \cdots, sd^n$ and check if they include the discriminator or not. If $sd^0, \cdots, sd^n$ include the discriminator, k is equal to $RK0_{0,0}$. If they do not include the discriminator, k is not equal to $RK0_{0,0}$. We can ignore registers other than the register R storing the round function output, since the discriminator is unique to $RK0_{0,0}$.

Even if we do not know when the round function output at Round 1 is stored into the register R, we can retrieve the scanned data several times and connect them as a single one. We can retrieve $RK0_{0,0}$ when the register R storing the round function output at Round 1 is included anywhere in the scanned data.

2.3.1 Discriminator of $RK0_{0,0}$

Assume that we have two plaintexts a^1 and a^2 whose first element $(0, 0)$ has a different value but other elements have the same values. Let f_1^1 and f_1^2 be the round function outputs at Round 1 for a^1 and a^2, respectively. By calculating $f_1^1 \oplus f_2^2$ as in Figure 2.6, the four elements of 32-bit data in the first column only depend on $a_{0,0}^1$, $a_{0,0}^2$, and $RK0_{0,0}$ and the other elements become zero. The four elements of 32-bit data in the first column of $f_1^1 \oplus f_2^2$ can be shown as in

Eqns. (2.9)–(2.11):

$$\begin{aligned}\begin{bmatrix} f^1_{0,0} \oplus f^2_{0,0} \\ f^1_{1,0} \oplus f^2_{1,0} \\ f^1_{2,0} \oplus f^2_{2,0} \\ f^1_{3,0} \oplus f^2_{3,0} \end{bmatrix} &= \begin{bmatrix} e^1_{0,0} \oplus e^2_{0,0} \\ e^1_{1,0} \oplus e^2_{1,0} \\ e^1_{2,0} \oplus e^2_{2,0} \\ e^1_{3,0} \oplus e^2_{3,0} \end{bmatrix} \\ &= \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} d^1_{0,0} \oplus d^2_{0,0} \\ d^1_{1,0} \oplus d^2_{1,0} \\ d^1_{2,0} \oplus d^2_{2,0} \\ d^1_{3,0} \oplus d^2_{3,0} \end{bmatrix} \end{aligned} \tag{2.9}$$

Since the two plaintexts a^1 and a^2 have the same value except for their first element, we have $d^1_{0,0} \neq d^2_{0,0}, d^1_{1,0} = d^2_{1,0}, d^1_{2,0} = d^2_{2,0}$, and $d^1_{3,0} = d^2_{3,0}$ as in Figure 2.6. Then we have:

$$= \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} d^1_{0,0} \oplus d^2_{0,0} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.10}$$

At ShiftRows, we have $c_{0,0} = d_{0,0}, c_{1,0} = d_{1,0}, c_{2,0} = d_{2,0}$, and $c_{3,0} = d_{3,0}$. Then we have:

$$= \begin{bmatrix} 02(c^1_{0,0} \oplus c^2_{0,0}) \\ c^1_{0,0} \oplus c^2_{0,0} \\ c^1_{0,0} \oplus c^2_{0,0} \\ 03(c^1_{0,0} \oplus c^2_{0,0}) \end{bmatrix} \tag{2.11}$$

Eqns. (2.9)–(2.11) show that the four elements of 32-bit data in $f^1_1 \oplus f^2_1$ can be calculated by XORing the SubByte outputs $c^1_{0,0}$ and $c^2_{0,0}$, where $c^1_{0,0}$ is derived from $a^1_{0,0} \oplus RK0_{0,0}$ and $c^2_{0,0}$ is derived from $a^2_{0,0} \oplus RK0_{0,0}$. The 8-bit value of $c^1_{0,0} \oplus c^2_{0,0}$ depends on only $a^1_{0,0}$, $a^2_{0,0}$ and $RK0_{0,0}$ and, if we can give $a^1_{0,0}$ and $a^2_{0,0}$ systematically, $c^1_{0,0} \oplus c^2_{0,0}$ can be used as a candidate of discriminators to retrieve $RK0_{0,0}$.

Now we propose our discriminator to retrieve $RK0_{0,0}$.

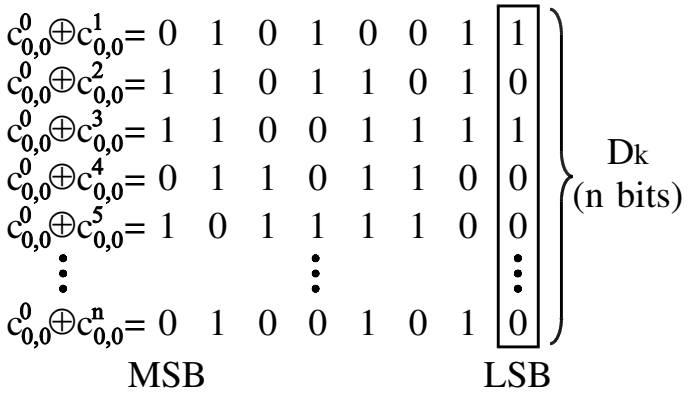

Figure 2.8: A discriminator for retrieving $RK0_{0,0}$.

D1: First we assume one value k for $RK0_{0,0}$. Then let $a^0_{0,0}$ be zero and $a^q_{0,0}$ $(q = 1, \cdots, n)$ be a 8-bit arbitrary non-zero value such that $a^p_{0,0} \neq a^q_{0,0}$ $(p \neq q)$.

D2: We calculate $c^0_{0,0} \oplus c^q_{0,0}$ using $a^0_{0,0}$, $a^q_{0,0}$ $(q = 1, \cdots, n)$, and the assumed value k for $RK0_{0,0}$, by applying the pre-round function and SubBytes to them.

D3: We focus on each bit of $c^0_{0,0} \oplus c^q_{0,0}$ $(q = 1, \cdots, n)$. As shown in Figure 2.8, the LSB of $c^0_{0,0} \oplus c^q_{0,0}$ $(q = 1, \cdots, n)$ will give us n-bit data which is dependent on $a^0_{0,0}$, $a^1_{0,0} \cdots a^n_{0,0}$, and k. We employ this n-bit data as the discriminator to retrieve $RK0_{0,0}$ and it is denoted by D_k.

2.3.2 Retrieving $RK0_{0,0}$ using our discriminator

Using the n-bit discriminator proposed in Section 2.3.1, we retrieve the round key element $RK0_{0,0}$ as follows: Let $a^q_{0,0}$ $(1, \cdots, n)$ be the 8-bit data which is determined at Step D1 in Section 5.1 to calculate the discriminator D_k. Then we can have the $(n + 1)$ plaintexts as shown in Eqn. (2.12).

$$a^0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad a^q = \begin{bmatrix} a^q_{0,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.12}$$

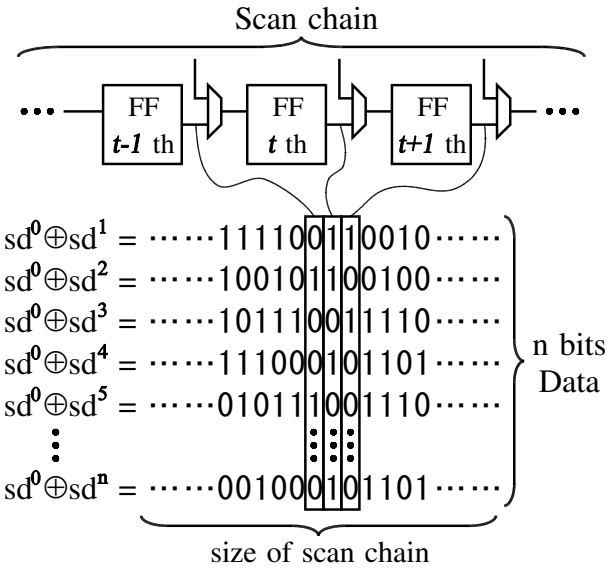

Figure 2.9: n-bit data based on $sd^0 \oplus sd^q$ $(q = 1, \cdots, n)$.

where $q = 1, \cdots, n$. Assume that we have the scanned data sd^0, $sd^1, \cdots, sd^n$ which include the round function output at Round 1 anywhere for a^0, $a^1, \cdots, a^n$, respectively. sd^0, $sd^1, \cdots, sd^n$ can include any register values, such as those in other circuits. Then we focus on each bit of $sd^0 \oplus sd^q$ $(q = 1, \cdots, n)$. Since each bit of sd^0, $sd^1, \cdots, sd^n$ always has the particular register value, we can have a set of n-bit data which corresponds to the particular register as in Figure 2.9.

At that time, if there exists the discriminator D_k in a set of the n-bit data, the assumed value k for $RK0_{0,0}$ is correct since D_k appears in the n-bit data only when the calculation of $c^0_{0,0} \oplus c^q_{0,0}$ for $q = 1, \cdots, n$ is correct. This means that we succeed to retrieve $RK0_{0,0}$. Otherwise, the assumed value k for $RK0_{0,0}$ is incorrect and we try the next value for $RK0_{0,0}$.

2.3.3 Retrieving $RK0$ using our method

Overall process to retrieve $RK0$ with our method is shown as follows:

P1: Calculation of discriminators.

- Let $a_{0,0}$ be zero and $a^q_{0,0}$ $(q = 1, \cdots, n)$ be a 8-bit arbitrary non-zero value such that $a^p_{0,0} \neq a^q_{0,0}$ $(p \neq q)$.
- Calculate the discriminator D_k for each k $(k = 0, \cdots, 255)$ of all 256 patterns of $RK0_{0,0}$.

P2: Obtaining scanned data.

- Let a^0 and a^q $(q = 1, \cdots, n)$ be 128-bit plaintexts whose $(0, 0)$ element is $a^0_{0,0}$ and $a^q_{0,0}$ $(q = 1, \cdots, n)$ determined at Step P1, respectively, and the other elements are all zero.
- Obtain scanned data sd^0 and sd^q $(q = 1, \cdots, n)$ including the round function output at Round 1 using $a^0_{0,0}$ and $a^q_{0,0}$ $(q = 1, \cdots, n)$.
- Calculate $sd^0 \oplus sd^q$ $(q = 1, \cdots, n)$ from the scanned data sd^0 and sd^q $(q = 1, \cdots, n)$.

P3: Retrieving $RK0_{0,0}$.

- Check if the discriminator D_k $(k = 0, \cdots, 255)$ exists in a set of n-bit data obtained from $sd^0 \oplus sd^q$ $(q = 1, \cdots, n)$. If D_k exisits, then $RK0_{0,0}$ is retrieved as k.

P4: Retrieving $RK0$.

- Retrieve the other elements in $RK0$ by repeating the above steps.

Our method retrieves $RK0$ by checking whether the discriminator D_k exists in each bit of $sd^0 \oplus sd^q$ $(q = 1, \cdots, n)$ or not. When the number of plaintexts prepared in Steps P1 and P2 above is large enough, our method successfully retrieve the round key $RK0$ if the register R storing the round function output at Round 1 is included anywhere in the scanned data. The scanned data may include data in other circuits, the round function outputs at Round 2 and later, and/or round function output before Round 1 (that will be zero or anything).

The number of required plaintexts to retrieve is calculated as follows: First, our method requires $(n+1)$ plaintexts to retrieve one element of $RK0$. However, since the plaintext a^0 only consists of zero as in Eqn. (2.12), this plaintext will be commonly used to retrieve the other elements in $RK0$. Then we can retrieve each element other than the first one in $RK0$ by using n plaintexts. In summary, retrieving the round key $RK0$ requires $n + 1 + 15n = 16n + 1$ plaintexts in total by using our proposed method.

As in Section 2.4, the number n of required plaintexts to correctly retrieve a single element in $RK0$ will be 16 to 20.

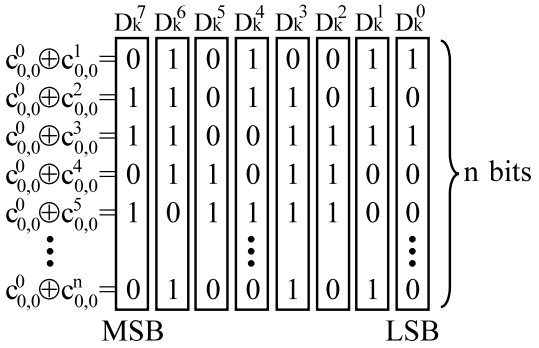

Figure 2.10: Discriminators $D_k^0, D_k^1, \cdots, D_k^7$ for retrieving $RK0_{0,0}$.

2.3.4 Reduction of plaintexts to retrieve $RK0$

Now, we consider to reduce plaintexts required by our proposed method to retrieve the round key $RK0$.

As described in Section 2.3.1, our discriminator D_k is a n-bit data composed of the LSB of $c_{0,0}^0 \oplus c_{0,0}^q$ $(q = 1, \cdots, n)$. However, since $c_{0,0}^0 \oplus c_{0,0}^1$ is a 8-bit data, we can also have 8 n-bit data, each of which is composed of each bit of $c_{0,0}^0 \oplus c_{0,0}^q$ $(q = 1, \cdots, n)$ (See Figure 2.10). These 8 n-bit data can also be used as a set of discriminators $\{D_k^0, \cdots, D_k^7\}$ for the value k.

If they all can be found in the scanned data, we can conclude that $RK0_{0,0}$ is k. But if one of them cannot be found in the scanned data, $RK0_{0,0}$ is not k. Then we can reduce the number of required plaintexts smaller than our original one proposed in Section 2.3.3.

As in Section 2.4, the number n of required plaintexts to correctly retrieve a single element in $RK0$ will be reduced to 14 to 18.

2.4 Performance analysis

In this section, we simulate our proposed method and analyze the number of plaintexts required to retrieve the round key $RK0$. We have conducted three types of experiments:

1. The AES architecture model is the one in [11] and the size of the secret key is 128 bits, which is the same condition as Yang's method [12]. The scan path includes only the register R storing the round function output.

2. The scan path includes the register R storing the round function output at Round 1 and also registers other than R.

3. The scan path includes the round function output at Round 1 and those at Round 2 through Round 10.

In each experiment, we generate randomly 10,000 secret keys and retrieve each of them using our method (Section 2.3.3) and our improved method (Section 2.3.4). Then we calculate the number of plaintexts required to correctly retrieve each of them.

Experiment 1 In Experiment 1, our method can retrieve the round key element $RK0_{0,0}$ by using 16 plaintexts on average and 20 plaintexts in the worst case. When retrieving all the elements in the round key $RK0$, we need $16 \times 16 + 1 = 257$ plaintexts on average and $20 \times 16 + 1 = 321$ plaintexts in the worst case. Our improved method described in Section 2.3.4 can retrieve the round key element $RK0_{0,0}$ by using 14 plaintexts on average and 18 plaintexts in the worst case. When retrieving all the elements in the round key $RK0$, we need $14 \times 16 + 1 = 225$ plaintexts on average and $16 \times 16 + 1 = 257$ plaintexts in the worst case.

As described in Section 2.2.6, Yang's method requires 536 plaintexts on average. Our method achieves 48% reduction of the number of the required plaintexts experimentally.

Figure 2.11 summarizes the number of required plaintexts on average to retrieve $RK0$ comparing Yang's method and our methods.

Experiment 2 In Experiment 2, we first obtain the scanned data composed of the 128-bit register R only and then add random bits to it. Total bit length

of scanned data is 128 (no extra data is added) to 4,096 (3,968-bit extra data is added). We calculate the number of required plaintexts by using our improved method in Section 2.3.4 to correctly retrieve the round key $RK0$. Figure 2.12 shows the experimental results.

Generally saying, if several plaintexts which have the same value except for their first element are given to the AES cryptography LSI, they affect not only the register R storing the round function output but also the registers in other circuit elements such as memories and peripheral circuits. Since the scanned data will include several "extra" data in them, the extra data can be randomly changed by inputting these plaintexts.

Yang's method assumes in Section 2.2.6 that several plaintexts which have the same value except for their first element affect the only round function output, but which is not the case in a practical AES cryptography LSI.

On the other hand, the experimental results show that our method can successfully retrieve the round key even if the scanned data includes the random data other than the register R storing the round function output. Even if the scanned data includes the 3,968-bit random data other than the round function output, our improved method is capable to retrieve the round key $RK0$ only requiring $20 \times 16 + 1 = 321$ plaintexts on average and $23 \times 16 + 1 = 369$ plaintexts in the worst case.

Experiment 3 Figure 2.13 shows the number of required plaintexts to retrieve the scanned data including the round function output at Round 1 only, and the number of required plaintexts to retrieve the scanned data including the round function output at Round 1 through Round 10.

As in Experiment 2, Yang's method assumes in Section 2.2.6 that several plaintexts which have the same value except for their first element affect the only round function output at Round 1. If the scanned data includes not only the round function output at Round 1 but also round function outputs at Round 2 through 10,

these input plaintexts affect all of these round function outputs. This means that we cannot find out the positions storing the first column of the round function output Round 1 unlike the discussion in Section 2.2.6.

On the other hand, even if the scanned data includes the round function output other than Round 1, the experimental results show that our method can successfully retrieve the round key. Even if the scanned data includes all of the round function output, our improved method is capable to retrieve the round key $RK0$ only requiring $19 \times 16 + 1 = 305$ plaintexts on average and $22 \times 16 + 1 = 353$ plaintexts in the worst case.

2.5 Concluding remarks

The scan-based attack proposed by Yang et al. is an effective attack but has disadvantages that they assume too ideal AES cryptography LSI. In contrast, our proposed method can perform scan-based attack without such assumptions. Our method does not have to know when the round function output is stored in the register R. Our method can retrieve the round key $RK0$ even if its scan path includes random elements caused by memories, processors, I/Os, and control circuits other than the register R storing the round function output. Furthermore, our method can reduce the number of plaintexts by 48% compared with Yang's method. Our method is more practical and powerful than Yang's method.

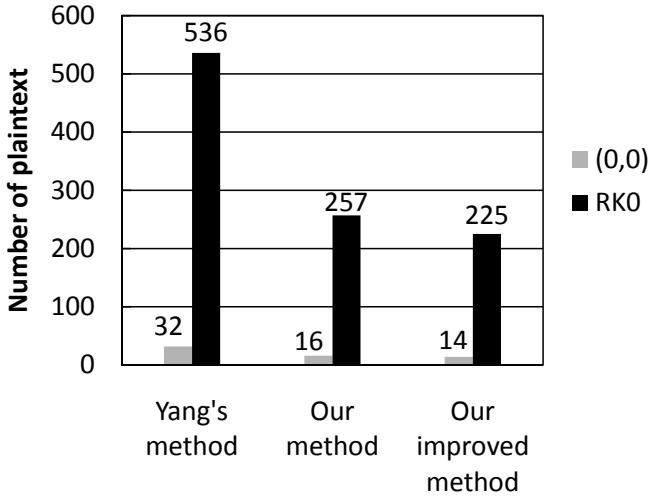

Figure 2.11: Number of plaintexts required to retrieve $RK0$ or $RK0_{0,0}$ on average.

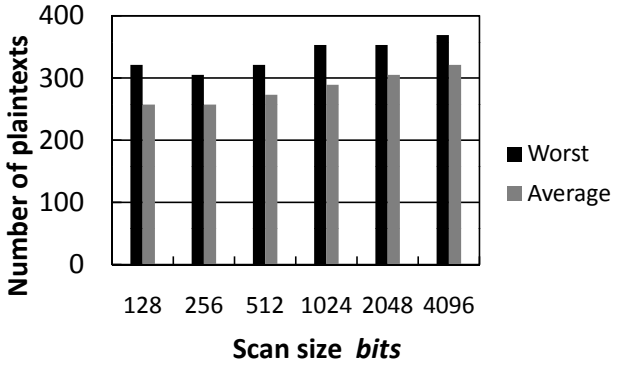

Figure 2.12: The experimental results for Experiment 2.

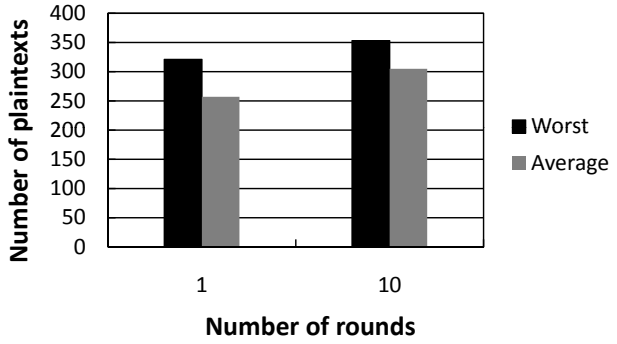

Figure 2.13: The experimental results for Experiment 3.

Chapter 3

Scan-based Attack againt RSA

In this chapter, we propose a scan-based attack against an RSA circuit, which is almost independent of a scan-path structure. The proposed method is based on detecting intermediate values calculated in an RSA circuit. We focus on a 1-bit time-sequence which is specific to some intermediate value. We call it a *scan signature* because its value shows their existence in the scanned data obtained from an RSA circuit. By checking whether a scan signature is included in the scanned data or not, we can retrieve a secret key in the target RSA circuit even if we do not know a scan path structure, as long as a scan path is implemented on an RSA circuit and it includes at least 1-bit of each intermediate value.

3.1 RSA Algorithm

RSA cryptography [6] was made public in 1978 by Ronald Linn Rivest, Adi Shamir, Leonard Max Adleman. The RSA is known as the first algorithm which makes public-key cryptography practicable. It is commonly used to acheive not only encryption/decription but also a digital signature and a digital authentication, so that most cryptography LSIs in the market implement and calculate the RSA cryptography.

The security of an RSA cryptography depends on the difficulty of factoring large numbers. To decrypt a ciphertext of an RSA cryptography will be almost impossible on the assumption that no efficient algorithm exists for solving it.

3.1.1 Encryption and decryption

An RSA algorithm encrypts a plaintext with a public key (n, e) and decrypts a ciphertext with a secret key (n, d). Let us select two distinct prime numbers p and q. We calculate n by multiplying p by q, which is used as the modulus for both a public key and a secret key. To determine exponents of them, we calculate $\varphi(pq)$[1] for multiplying $(p-1)$ by $(q-1)$.

Let us select an integer e satisfying the conditions that $1 < e < \varphi(pq)$ and, e and $\varphi(pq)$ is coprime, where e is an exponent of a public key. Let us determine an integer d satisfying the congruence relation $de \equiv 1 \mod \varphi(pq)$. That is to say, the public key consists of the modulus n and the exponent e. The private key consists of the modulus n and the exponent d.

Let us consider that Alice secretly sends a message m to Bob. First, Alice receives his public key (n, e). Second, she calculates the ciphertext c with Eqn. (3.1).

$$c = m^e \mod n \tag{3.1}$$

Then Alice transmits c to Bob. Bob decrypts c by using his private key and receive her message m. Eqn. (3.2) represents a decryption computation.

$$m \equiv c^d \mod n \tag{3.2}$$

3.1.2 Binary method

The bit length of an RSA key must be more than 1,024 bits because its security depends on its key length. It is currently recommended that n be at least 2,048

[1] $\varphi()$ is Euler's totient function.

Algorithm 1 Binary method (MSB to LSB).

Input: c, d, and n.

Output: $c^d \bmod n$.

```
i = L − 1.
m = 1.
while i ≥ 0 do
    m = m^2 mod n.
    if d_i = 1 then
        m = m × c mod n.
    end if
    i = i − 1.
end while
return m.
```

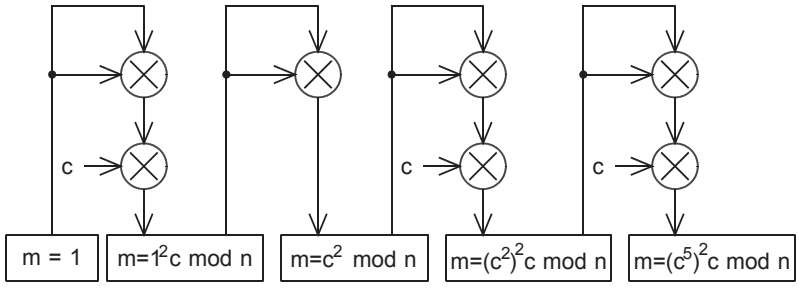

Figure 3.1: Binary method example ($d = 1011_2$).

bits long [13]. This means that the exponent d in Eqn. (3.2) is at least 1,024 bits long. When we decrypt a cyphertext, its computation amount becomes quite large without modification. Since modulo exponentiation dominates the execution time of decrypting a cyphertext, efficient algorithms have been proposed. The binary method [14], as shown in Algorithm 1, is one of the most typical exponent algorithms. In Algorithm 1, the exponent d is represented by $d = d_{L-1}2^{L-1} + d_{L-2}2^{L-2} + \cdots + d_1 2 + d_0$, where L shows the maximum key bit length. Figure 3.1 shows an example of the binary method in case of $d = 1011_2$.

3.2 Scan-based attack against RSA

A scan path connects registers in an circuit serially and makes us access to them directly so that a tester can observe register values inside the circuit easily. We explain a scan path test in detail at Section 2.2.1.

The purpose of a scan-based attack against RSA is to retrieve a secret exponent d from scanned data in an RSA circuit. Scan-based attack here requires several assumptions as in the previous researches in [9, 10, 15, 16], which are summarized as shown below:

1. Attackers can encrypt/decrypt arbitrary data using the secret key on a target RSA circuit.
2. Attackers can obtain scanned data from a target RSA circuit.
3. Scanned data is not modified with compactors aimed at test efficiency.
4. Attackers know that the binary method in Algorithm 1 is used in a target RSA circuit.
5. Attackers also know the modulus n used in a target RSA circuit.[2]

In addition to these, they need to be able to predict the intermediate values of the binary method using an off-line simulation.

In this section, we explain the scan-based attack against an RSA circuit (Section 3.2.1) and its problems in a practical case (Section 3.2.2).

3.2.1 Retrieving a secret exponent using intermediate values [1]

In order to retrieve a secret exponent d, we have to solve the integer factorization in RSA. If the bit length of a secret exponent d is more than 1,024 bits or more

[2]Note that, since the public key consists of the modulus n and the public exponent e, attackers can easily know the modulus n.

than 2,048 bits, it is impossible to solve this problem within a realistic time. However, if we know all the "intermediate values" during the binary method shown in Algorithm 1, we can retrieve a secret exponent d in a polynomial time [1].

Let $d = d_{L-1}2^{L-1} + d_{L-2}2^{L-2} + \cdots + d_1 2 + d_0$, where L is the maximum key bit length of d. Assume that all the intermediate values in Algorithm 1 are obtained. Let $m(i)$ be the intermediate value of m at the end of loop i in Algorithm 1. Assume also that $d_{L-1}, d_{L-2}, \cdots, d_{i+1}$ are already retrieved. An attacker tries to reveal the next bit d_i. In this case, $m(i)$ is equal to Eqn. (3.3) below, if and only if $d_i = 0$:

$$c^{\sum_{j=i+1}^{L-1} d_j 2^{j-i}} \mod n. \tag{3.3}$$

Similarly, $m(i)$ is equal to Eqn. (3.4) below, if and only if $d_i = 1$:

$$c^{\sum_{j=i+1}^{L-1} d_j 2^{j-i}+1} \mod n. \tag{3.4}$$

Based on the above discussion, we employ $SF(i)$ defined by Eqn. (4.4) as a *selective function* for RSA:

$$SF(i) = c^{\sum_{j=i+1}^{\ell-1} d_j 2^{j-i}+1} \mod n. \tag{3.5}$$

ℓ represents a significant key length, or key length in left-align representation, i.e., the secret exponent can be represented by

$$\begin{aligned} d &= d_{L-1}2^{L-1} + \cdots + d_1 2 + d_0\big|_{d_{L-1}=0,\ldots,d_\ell=0} \\ &= d_{\ell-1}2^{\ell-1} + \cdots + d_1 2 + d_0. \end{aligned} \tag{3.6}$$

When using the selective function for RSA above, we have to know in advance $d_{\ell-1}, d_{\ell-2}, \cdots, d_{i+1}$.

$SF(i) \neq SF(j)$ always holds true for $i \neq j$ for $0 \leq i, j \leq \ell - 1$. Given a message c and bit values of secret component $d_{\ell-1}, d_{\ell-2}, \cdots, d_{i+1}$, we assume that $d_i = 1$ and check whether $SF(i)$ appears somewhere in intermediate values. If it appears in them, we really determine d_i as one. If not, we determine d_i as zero.

Example 1 *Let us consider that the public key* $(n, e) = (101111001, 1011)$ *and the secret key* $(n, d) = (101111001, 10111)$. *The maximum key length* L *is 8 bits and the secret exponent* $d = 10111$, *i.e,* $d_7 = 0$, $d_6 = 0$, $d_5 = 0$, $d_4 = 1$, $d_3 = 0$, $d_2 = 1$, $d_1 = 1$, $d_0 = 1$. *We assume that we do not know* d *and a significant key length* ℓ. *The intermediate values in Algorithm 1 are summarized in Table 3.2 when we use a message* $c = 10011100$, *whose parameters are shown in Table 3.1.*

Now we try to retrieve the 8-bit secret exponent d *using intermediate values.*

First we try to retrieve the first bit $d_{\ell-1}$ $(i = \ell-1)$. *We find* $d_{\ell-1} = 1$ *by the definition of a significant key length* ℓ. *Then* $SF(\ell-1)$ *is calculated as* $SF(\ell-1) = c = 10011100$. *Since* 10011100 *appears in Table 3.2, we confirm that* $d_{\ell-1}$ *is retrieved as one. Now we assume that the secret exponent* $d = \underline{\mathbf{1}}$. *We compare* $m(\ell-1) = (c^1 \mod n) = 10011100$ *with the binary method result* 10001111. *Since they are not equal,* $d \neq 1$.

Next, we try to retrieve the second bit $d_{\ell-2}$ $(i = \ell-2)$. *We have already known that* $d_{\ell-1} = 1$. *We assume here that* $d_{\ell-2} = 1$. *In this case,* $SF(\ell-2)$ *is calculated as* $SF(\ell-2) = 11010$. *Since* 11010 *does not appear in Table 3.2, then* $d_{\ell-2}$ *is retrieved not as one but as zero, i.e.,* $d_{\ell-2} = 0$. *Now we assume that* $d = 1\underline{\mathbf{0}}$. *We compare* $m(\ell-2) = (c^{10} \mod n) = (m(\ell-1)^2 \mod n) = 11010000$ *with the binary method result* 10001111. *Since they are not equal,* $d \neq 10$.

Next, we try to retrieve the third bit $d_{\ell-3}$ $(i = \ell-3)$. *We have already known that* $d_{\ell-1} = 1$ *and* $d_{\ell-2} = 0$. *We assume here that* $d_{\ell-3} = 1$. *In this case,* $SF(\ell-3)$ *is calculated as* $SF(\ell-3) = 10000010$. *Since* 10000010 *appears in Table 3.2, then* $d_{\ell-3}$ *is retrieved as one, i.e.,* $d_{\ell-3} = 1$. *Now we assume that* $d = 10\underline{\mathbf{1}}$. *We compare* $m(\ell-3) = (c^{101} \mod n) = SF(\ell-3) = 10000010$ *with the binary method result* 10001111. *Since they are not equal,* $d \neq 101$.

Next, we try to retrieve the fourth bit $d_{\ell-4}$ $(i = \ell-4)$. *We have already known that* $d_{\ell-1} = 1$, $d_{\ell-2} = 0$ *and* $d_{\ell-3} = 1$. *We assume here that* $d_{\ell-4} = 1$. *In this case,* $SF(\ell-4)$ *is calculated as* $SF(\ell-4) = 100111$. *Since* 100111 *appears in Table 3.2, then* $d_{\ell-1}$ *is retrieved as one, i.e.,* $d_{\ell-4} = 1$. *Now we assume that* $d = 101\underline{\mathbf{1}}$. *We compare* $m(\ell-4) = (c^{1011} \mod n) = SF(\ell-4) = 100111$ *with the binary method result* 10001111. *Since they are not equal,* $d \neq 1011$.

We have already known that $d_{\ell-1} = 1$, $d_{\ell-2} = 0$, $d_{\ell-3} = 1$ *and* $d_{\ell-4=1}$. *We assume here that* $d_{\ell-5} = 1$. $SF(\ell - 5)$ *is calculated as* $SF(\ell - 5) = 10001111$ *(*$i = \ell - 5$*). Since* 10001111 *appears in Table 3.2, then* $d_{\ell-5}$ *is retrieved as one, i.e.,* $d_{\ell-5} = 1$. *Now we assume that* $d = 1011\underline{\mathbf{1}}$. *We compare* $m(\ell-5) = (c^{10111} \mod n) = SF(\ell-5) = 10001111$ *with the binary method result* 10001111. *Since they are equal to each other, we find that the secret exponent* d *is* 10111 *and a significant bit* ℓ *is five.*

3.2.2 Problems to retrieve a secret key using scan path

If we retrieve an L-bit secret exponent d using an exhaustive search, we have to try 2^L possible values to do it. On the other hand, the method explained in Section 3.2.1 retrieves a secret exponent one-bit by one-bit from MSB to LSB. It tries at most $2L$ possible values to retrieve an L-bit secret exponent. Further, the method just checks whether $SF(i)$ exists in the intermediate value $m(i)$ in Algorithm 1.

In order to apply this method to a scan-based attack, we have to know which registers store intermediate values, i.e., we have to know correspondence between scanned data and $SF(i)$.

However, scan paths are usually designed automatically by EDA tools so that nearby registers are connected together to shorten the scan path length. Only designers can know the correspondence between scanned data and registers and thus retrieved scanned data can be considered to be "random" for attackers. Therefore, it is very difficult to find out the values of $SF(i)$ in scanned data for attackers.

Messerges [1] only shows the correspondence between intermediate values and a bit of a secret exponent. It does not indicate the method how to discover the intermediate value from scanned data. For that reason, its analysis method cannot directly apply to scan-based attacks against an RSA LSI.

We have to find out only $SF(i)$ somehow in the scanned data to retrieve a secret exponent d using the method in Section 3.2.1.

Table 3.1: Example parameters in Algorithm 1.

Maximum key length L	8 bits
Modulus n	101111001
Public exponent e	1011
Secret exponent d	10111

Table 3.2: Intermediate values at the end of i-th loop of Algorithm 1 (message $c = 10011100_2$).

i	d_i	m^2	m
7	0	1	1
6	0	1	1
5	0	1	1
4	1	1	10011100
3	0	11010000	11010000
2	1	100011110	10000010
1	1	100111000	100111
0	1	1101	10001111

3.3 Analysis scanned data

In order to solve the problem that attackers do not know the correspondence between registers of the scanned data and ones storing intermediate values during the binary method, we focus on the general property on scan paths: *a bit position of a particular register r in a scanned data when giving one input data is exactly the same as that when giving another input data.* This is clearly true, since a scan path is fixed in an LSI chip and the order of connected registers in its scan path is unchanged.

If we execute the binary method for each of N messages on an RSA circuit, a bit pattern of a *particular* bit position in scanned data for these N messages gives

N-bit data. Based on the above property, this N-bit data may give a bit pattern of a particular bit in an intermediate value when we give each of these N messages to the RSA circuit.

We can calculate $SF(i)$ from the same N messages and $d_{\ell-1}$ down to d_0 of the secret exponent d by using an off-line simulation. By picking up a particular bit (LSB, for example) in each of $SF(i)$ values for N messages, we also have an N-bit data (see Figure 3.2). If N is large enough, this N-bit data gives information completely unique to $SF(i)$. We can use this N-bit data as a *scan signature* SS_i to $SF(i)$ in scanned data.

Our main idea in this section is that we find out a scan signature SS_i to $SF(i)$ in scanned data (see Figure 3.3) to retrieve the secret exponent d from $d_{\ell-1}$ down to d_0. If an N-bit scan signature SS_i appears in the scanned data for N messages, d_i is determined as one. If not, it is determined as zero.

In the rest of this section, we firstly propose a scan signature SS_i to $SF(i)$. Secondly we propose an overall method to retrieve a secret exponent d using scan signatures. Thirdly we analyze the probabilities of successfully retrieving a secret exponent by using our method.

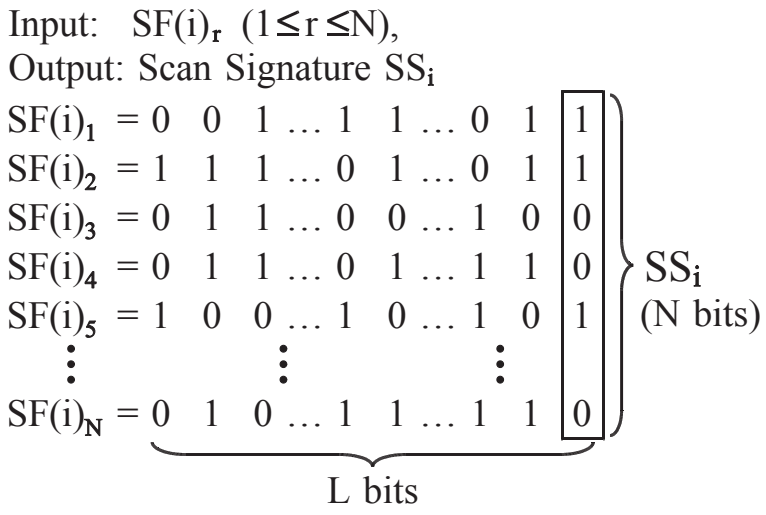

Figure 3.2: Scan signature SS_i.

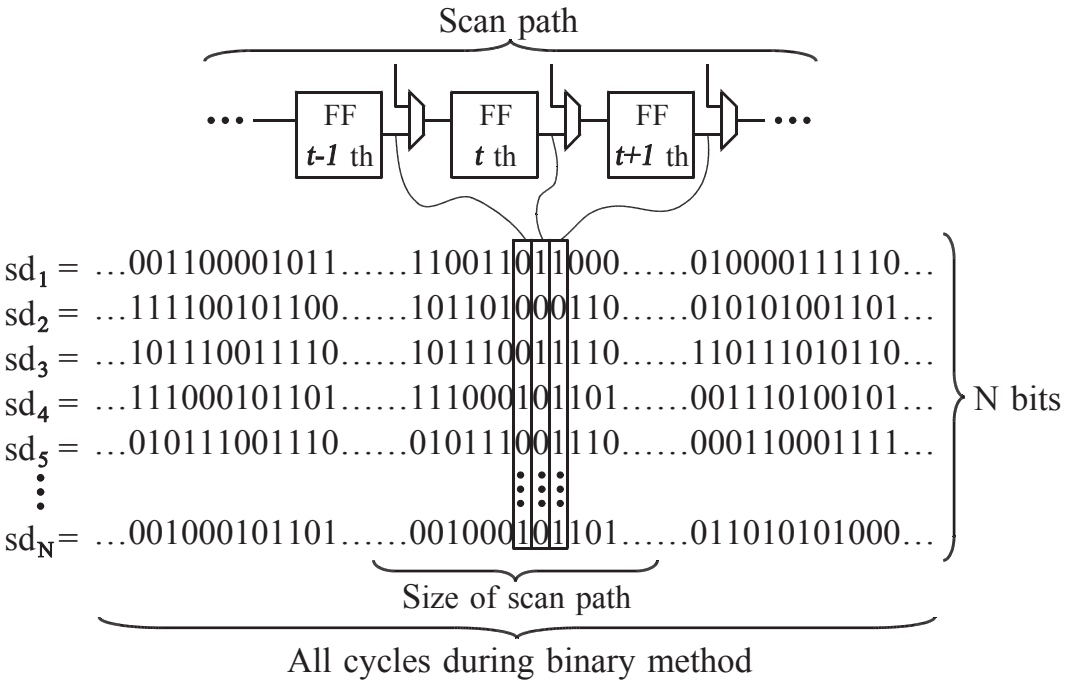

Figure 3.3: Scanned data.

3.3.1 Calculating a scan signature to $SF(i)$

Assume that N messages $c_1, \cdots, c_N$ are given. Also assume that we have already known $d_{\ell-1}, \cdots, d_{i+1}$ for a secret exponent d. Let $SF(i)_r$ be the selective function for RSA when giving the message c_r for $1 \leq r \leq N$. Assuming that $d_i = 1$, we can calculate $SF(i)_r$ for $1 \leq r \leq N$.

Let us focus on a particular bit of $SF(i)_r$. If N is large enough, a set of these bits for $SF(i)_r$ $(1 \leq r \leq N)$ gives information unique to $SF(i)_r$. By using it, we can check whether $SF(i)_r$ are calculated or not in the target. As Figure 3.2 shows, we define a *scan signature* SS_i to be a set of $SF(i)_r$ LSBs for the sake of convenience.

If SS_i appears in scanned data, d_i is determined as one. If not, d_i is determined as zero. After d_i is correctly determined, we can continue to determine the next bit of the secret exponent d in the same way.

Our proposed method has an advantage compared to conventional scan-based attacks [9, 10]. Our method is effective in the case of partial scan architecture. As long as a scan path includes at least 1-bit of each intermediate value, we can check whether the scan signature exists or not in the scanned data.

3.3.2 Scanned data analysis method

First we prepare N messages $c_1, \cdots, c_N$ and give them to an RSA circuit. For each of these messages, we obtain all the scanned data from the scan out of the RSA circuit until it outputs the binary method result. As Figure 3.3 shows, the size of scanned data for each of these messages is ("scan path length" $\times$ "number of binary method cycles.")

Now we check whether a scan signature SS_i to $SF(i)$ appears in the obtained scanned data under the assumption that we do not know a secret exponent d in the RSA circuit as follows:

Step 1: Prepare N messages $c_1, c_2, \cdots, c_N$, where $c_r \neq c_s$ for $1 \leq r, s \leq N$ and

$r \neq s$.

Step 2: Input c_r $(1 \leq r \leq N)$ into the target RSA circuit and obtain scanned data every one cycle while the binary method works, until the RSA circuit outputs the result. Let sd_r denote the obtained scanned data for the message c_r $(1 \leq r \leq N)$.

Step 3: From the definition, we have $d_{\ell-1} = 1$. Compare $m(\ell-1) = (c_1 \mod n)$ with its binary method result. If they are equal, then we find that the secret exponent d is one and stop. If not, go to the next step.

Step 4: Calculate $SF(\ell-2)_r$ assuming $d_{\ell-2} = 1$ for each c_r $(1 \leq r \leq N)$ and obtain the scan signature $SS_{\ell-2}$.

Step 5: Check whether the scan signature $SS_{\ell-2}$ exists in the scanned data $sd_1, \cdots, sd_N$, which includes the scanned data in all the cycles while the binary method runs. If it exists, then we can find out that $d_{\ell-2}$ is equal to 1, and if it does not exist, then we can find out that $d_{\ell-2}$ is equal to 0.

Step 6: Calculate $m(\ell-2) = ((c_1)^{d_{\ell-1}\times 2+d_{\ell-2}} \mod n)$ and compare it with its binary method result. If they are equal, then we find that the secret exponent d is retrieved and terminate the analysis flow.

Step 7: We determine $d_{\ell-3}, d_{\ell-4}, \cdots$ in the same way as Step 4–Step 6 until the analysis flow is terminated at Step 6.

We show the example below to explain how the method above works.

Example 2 *As in Example 1, let us consider that the public key* $(n, e) = (101111001, 11)$ *and the secret key* $(n, d) = (101111001, 10111)$. *The maximum key length* L *is 8 bits and the secret exponent* $d = 10111_{10} = 10111_2$, *i.e,* $d_7 = 0$, $d_6 = 0$, $d_5 = 0$, $d_4 = 1$, $d_3 = 0$, $d_2 = 1$, $d_1 = 1$, $d_0 = 1$. *We assume that we do not know* d *and a significant key length* ℓ. *The parameters are shown in Table 3.1. Assume that the cycle counts of binary method are 16 and the size of the scan path is 128 in the target RSA circuit.*

*(***Step 1***) First we prepare* 8 *messages* $c_1, c_2, \cdots, c_8$*, where* $c_r \neq c_s$ *for* $1 \leq r, s \leq 8$ *and* $r \neq s$*. The target RSA circuit executes the binary method as in Table 3.2.*

*(***Step 2***) We input* c_r *(*$1 \leq r \leq 8$*) into the target RSA circuit and obtain scanned data every one cycle while the binary method works, until the RSA circuit outputs the result. Let* sd_r *denote the obtained scanned data for the messages* c_r $(1 \leq r \leq 8)$*. The total size of scanned data is* $16 \times 128 = 2,048$ *(see Figure 3.4).*

*(***Step 3***) Let us start to determine* $d_{\ell-1}$*. We find* $d_{\ell-1} = 1$ *by the definition of* ℓ*. It is not necessary to check whether* $d_{\ell-1} = 1$ *or not, but we can check it as follows: we calculate* $SF(\ell-1)_r = c_r$ *for each* c_r *(*$1 \leq r \leq 8$*) and obtain the scan signature* $SS_{\ell-1}$ *(see Figure 3.5). As Figure 3.5 (a) shows, the scan signature* $SS_{\ell-1}$ *becomes "11101001". Since we find out that the scan signature* $SS_{\ell-1}$ *exists in bit patterns of scanned data* sd_r $(1 \leq r \leq 8)$ *in Figure 3.4, we confirm that* $d_{\ell-1}$ *is retrieved as one, i.e.,* $d_{\ell-1} = 1$*. Now we assume that* $d = \underline{\mathbf{1}}$*. We compare* $m(\ell - 1) = ((c_1)^1 \mod n)$ *with its binary method result. In case they are not equal,* $d \neq 1$*.*

*(***Step 4***,* **Step 5***,* **Step 6***, and* **Step 7***) Next let us determine* $d_{\ell-2}$*. We calculate* $SF(\ell - 2)_r$ *assuming* $d_{\ell-2} = 1$ *for each* c_r *(*$1 \leq r \leq 8$*) and obtain the scan signature* $SS_{\ell-2}$ *(see Figure 3.5 (b)). As Figure 3.5 (b) shows, the scan signature* $SS_{\ell-2}$ *becomes "01111100". Since we find out that the scan signature* $SS_{\ell-2}$ *does not exist in bit patterns of scanned data* sd_r $(1 \leq r \leq 8)$ *in Figure 3.4, we can determine that* $d_{\ell-2}$ *is equal to zero, i.e.,* $d_{\ell-2} = 0$*. Now we assume that* $d = 1\underline{\mathbf{0}}$*. We compare* $m(\ell - 2) = (m(\ell - 1)^2 \mod n)$ *with its binary method result. In case they are not equal,* $d \neq 10$*.*

*(***Step 4***,* **Step 5***,* **Step 6***, and* **Step 7***) Next let us determine* $d_{\ell-3}$*. We calculate* $SF(\ell - 3)_r$ *assuming* $d_{\ell-3} = 1$ *for each* c_r *(*$1 \leq r \leq 8$*) and obtain the scan signature* $SS_{\ell-3}$ *(see Figure 3.5 (c)). As Figure 3.5 (c) shows, the scan signature* $SS_{\ell-3}$ *becomes "00010110". Since we find out that the scan signature* $SS_{\ell-3}$ *exists in bit patterns of scanned data* sd_r $(1 \leq r \leq 8)$ *in Figure 3.4, we can determine that* $d_{\ell-3}$ *is equal to one, i.e.,* $d_{\ell-3} = 1$*. Now we assume that* $d = 10\underline{\mathbf{1}}$*. We compare* $m(\ell - 3) = (m(\ell - 2)^2 \times c_1 \mod n) = SF(\ell - 3)_1$ *with its binary method result. In case they are not equal,* $d \neq 101$*.*

*(***Step 4***,* **Step 5***,* **Step 6***, and* **Step 7***) Next let us determine* $d_{\ell-4}$*. We calculate* $SF(\ell - 4)_r$ *assuming* $d_{\ell-4} = 1$ *for each* c_r *(*$1 \leq r \leq 8$*) and obtain the scan signature*

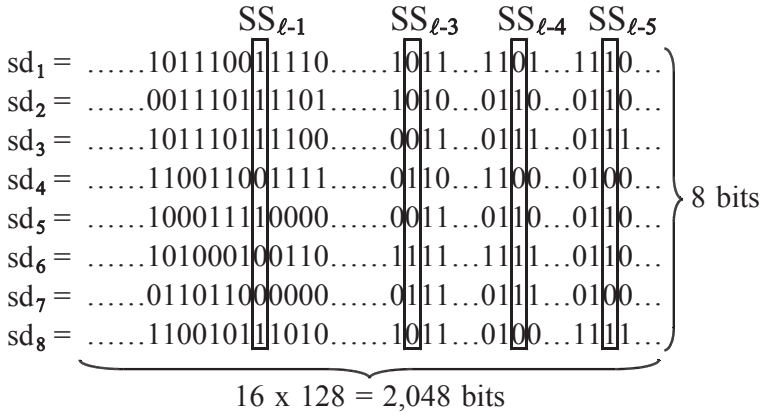

Figure 3.4: Scanned data example.

$SS_{\ell-4}$ (see Figure 3.5 (d)). As Figure 3.5 (d) shows, the scan signature $SS_{\ell-4}$ becomes "01101110". Since we find out that the scan signature $SS_{\ell-4}$ exists in bit patterns of scanned data sd_r $(1 \leq r \leq 8)$, we can determine that $d_{\ell-4}$ is equal to one, i.e., $d_{\ell-4} = 1$. Now we assume that $d = 101\underline{\mathbf{1}}$. We compare $m(\ell-4) = (m(\ell-3)^2 \times c_1 \mod n) = SF(\ell-4)_1$ with its binary method result. In case they are not equal, $d \neq 1011$.

*(**Step 4**, **Step 5**, **Step 6**, and **Step 7**) Finally let us determine $d_{\ell-5}$. We calculate $SF(\ell-5)_r$ assuming $d_{\ell-5} = 1$ for each c_r $(1 \leq r \leq 8)$ and obtain the scan signature $SS_{\ell-5}$ (see Figure 3.5 (e)). As Figure 3.5 (e) shows, the scan signature $SS_{\ell-5}$ becomes "11101101". Since we find out that the scan signature $SS_{\ell-5}$ exists in bit patterns of scanned data sd_r $(1 \leq r \leq 8)$, we can determine that $d_{\ell-5}$ is equal to one, i.e., $d_{\ell-5} = 1$. Now we assume that $d = 1011\underline{\mathbf{1}}$. We compare $m(\ell-5) = (m(\ell-4)^2 \times c_1 \mod n) = SF(\ell-5)_1$ with its binary method result. In case they are equal to each other, we find that the secret exponent d is 10111 and a significant bit ℓ is five.*

Input: $SF(\ell\text{-}1)_r$ $(1 \le r \le 8)$
Output: Scan signature $SS_{\ell\text{-}1}$

$SF(\ell\text{-}1)_1$ =	1	0	1	1	0	0	1	1
$SF(\ell\text{-}1)_2$ =	1	0	0	1	1	0	0	1
$SF(\ell\text{-}1)_3$ =	0	0	1	0	1	1	0	1
$SF(\ell\text{-}1)_4$ =	0	1	0	1	0	0	1	0
$SF(\ell\text{-}1)_5$ =	1	0	1	0	1	1	0	1
$SF(\ell\text{-}1)_6$ =	0	0	0	1	0	0	1	0
$SF(\ell\text{-}1)_7$ =	1	1	1	1	1	0	0	0
$SF(\ell\text{-}1)_8$ =	0	0	0	1	1	0	0	1

$\} SS_{\ell\text{-}1}$

(a) Scan signature $SS_{\ell\text{-}1}$.

Input: $SF(\ell\text{-}2)_r$ $(1 \le r \le 8)$
Output: Scan signature $SS_{\ell\text{-}2}$

$SF(\ell\text{-}2)_1$ =	0	0	0	0	1	0	1	0
$SF(\ell\text{-}2)_2$ =	1	1	1	1	0	1	0	1
$SF(\ell\text{-}2)_3$ =	0	1	1	0	0	1	1	1
$SF(\ell\text{-}2)_4$ =	1	1	0	0	1	1	0	1
$SF(\ell\text{-}2)_5$ =	1	1	0	0	1	0	1	1
$SF(\ell\text{-}2)_6$ =	0	0	1	1	1	0	1	1
$SF(\ell\text{-}2)_7$ =	0	1	1	0	0	1	1	0
$SF(\ell\text{-}2)_8$ =	0	1	1	1	0	1	0	0

$\} SS_{\ell\text{-}2}$

(b) Scan signature $SS_{\ell\text{-}2}$.

Input: $SF(\ell\text{-}3)_r$ $(1 \le r \le 8)$
Output: Scan signature $SS_{\ell\text{-}3}$

$SF(\ell\text{-}3)_1$ =	0	1	1	0	0	1	1	0
$SF(\ell\text{-}3)_2$ =	1	1	1	1	1	0	0	0
$SF(\ell\text{-}3)_3$ =	0	1	1	0	0	1	0	0
$SF(\ell\text{-}3)_4$ =	1	0	0	1	0	0	1	1
$SF(\ell\text{-}3)_5$ =	0	0	0	0	1	1	0	0
$SF(\ell\text{-}3)_6$ =	0	0	0	1	0	0	1	1
$SF(\ell\text{-}3)_7$ =	0	1	1	1	0	0	0	1
$SF(\ell\text{-}3)_8$ =	0	0	0	1	1	0	0	0

$\} SS_{\ell\text{-}3}$

(c) Scan signature $SS_{\ell\text{-}3}$.

Input: $SF(\ell\text{-}4)_r$ $(1 \le r \le 8)$
Output: Scan signature $SS_{\ell\text{-}4}$

$SF(\ell\text{-}4)_1$ =	1	1	0	0	1	0	1	0
$SF(\ell\text{-}4)_2$ =	0	1	0	1	1	1	0	1
$SF(\ell\text{-}4)_3$ =	1	1	1	0	0	1	1	1
$SF(\ell\text{-}4)_4$ =	0	1	1	0	0	1	0	0
$SF(\ell\text{-}4)_5$ =	1	1	0	1	1	0	1	1
$SF(\ell\text{-}4)_6$ =	1	0	1	1	0	0	1	1
$SF(\ell\text{-}4)_7$ =	0	1	1	0	0	1	1	1
$SF(\ell\text{-}4)_8$ =	1	0	1	1	0	0	0	0

$\} SS_{\ell\text{-}4}$

(d) Scan signature $SS_{\ell\text{-}4}$.

Input: $SF(\ell\text{-}5)_r$ $(1 \le r \le 8)$
Output: Scan signature $SS_{\ell\text{-}5}$

$SF(\ell\text{-}5)_1$ =	0	0	1	0	1	1	1	1
$SF(\ell\text{-}5)_2$ =	0	0	0	1	1	0	0	1
$SF(\ell\text{-}5)_3$ =	0	0	1	0	0	1	0	1
$SF(\ell\text{-}5)_4$ =	0	1	1	1	0	1	0	0
$SF(\ell\text{-}5)_5$ =	0	1	1	1	1	0	1	1
$SF(\ell\text{-}5)_6$ =	1	0	1	0	0	0	0	1
$SF(\ell\text{-}5)_7$ =	0	1	1	0	1	1	1	0
$SF(\ell\text{-}5)_8$ =	1	0	1	1	1	0	1	1

$\} SS_{\ell\text{-}5}$

(e) Scan signature $SS_{\ell\text{-}5}$.

Figure 3.5: Example of scan signatures.

3.3.3 Possibility of successfully retrieving a secret key

Given that the scan size is α bits and the cycle counts to obtain the binary method result is T. Assume that scanned data are completely random data.

Even though $SF(i)_r$ for $1 \leq r \leq N$ is not calculated in the target RSA circuit, its scan signature may exist in scanned data. When $\alpha T < 2^N$, the probability that the scan signature SS_i to $SF(i)_r$ exists in somewhere in bit patterns of scanned data sd_r $(1 \leq r \leq N)$ is $\alpha T/2^N$ despite we do not calculate $SF(i)_r$.

Sufficiently large N can decrease the probability that we mistakenly find out the scan signature SS_i in scanned data. For instance, if α is 3,072, T is 1,024, and N is 30[3], then the probability that we mistakenly find out the scan signature SS_i in scanned data is $3,072 \times 1,024/2^{30} \simeq 2.93 \times 10^{-3}$. If α is 6,144, T is 2,048, and N is 35, then the probability that we mistakenly find out the scan signature SS_i in scanned data is $6,144 \times 2,048/2^{35} \simeq 3.66 \times 10^{-4}$.

3.4 Experiments and analysis

We have implemented our analysis method proposed in Section 3.3 in the C language on Red Hat Enterprise Linux 5.5, AMD Opteron 2360SE 2.5GHz, and 16GB memories and performed the following experiments:

1. First, we have generated secret exponents randomly. Thousand of them have a bit length of 1,024 and 2,048, respectively. The other hundred of them have a bit length of 4,096.

2. Next, we have given each of the secret exponents into the target RSA circuit based on Algorithm 1 and obtained scanned data. The target RSA circuit obtains binary method results in 1,024 cycles for a 1,024-bit secret exponent, in 2,048 cycles for a 2,048-bit secret exponent, and in 4,096 cycles for a 4,096-bit secret exponent. Scan path length for a 1,024-bit secret exponent is 3,072

[3]These values are derived from the experiments in Section 3.4.

Table 3.3: Secret exponent example.

	4-th secret exponent d
1,024 bits	0x3AD29CF2FC6CB6B0C010B17DF98C5081
	4E4585225AC42E8ECB7BB1847498D62F
	BA696CDD226EE9195F4E58A89321721F
	021C4511E6C994301363706058FF3765
	E29EEBA03E370A201BA5B60A356682A5
	1D05EE10DF8CB75D7B4578B3D29A515E
	2F86DEC487AB6BCD88C7351908D71851
	6C11B2419BD8C05739214E6CF44D12F

bits, that for a 2,048-bit secret exponent is 6,144 bits, and that for a 4,096-bit secret exponent is 12,192 bits. Then total size of the obtained scanned data for 1,024-bit secret exponent is $3,072 \times 1,024 = 3,145,728$ bits, that for 2,048-bit secret exponent is $6,144 \times 2,048 = 12,582,912$ bits, and that for 4,096-bit secret exponent is $12,192 \times 4,096 = 49,938,432$ bits

3. Finally, we have retrieved each of the secret exponents by our proposed analysis method using the obtained scanned data.

Figure 3.6 and Table 3.4 show the results. Figure 3.6 shows the number N of required messages to retrieve each secret exponent when giving each of the secret exponents. For example, the 4th 1,024-bit secret exponent is shown in Table 3.3. In order to retrieve this secret exponent, we need 29 messages, i.e., $n = 29$. In this case, we can successfully retrieve the 4th secret exponent using 29 messages but fail to retrieve it using 28 messages or less.

Throughout this experiment, the required number of messages is approximately 29.5 on average for 1,024-bit secret exponents and is approximately 32 for 2,048-bit secret exponents and is approximately 37 for 4,096-bit secret exponents.

Table 3.4: Experimental results.

Key bit length *bit*	1,024	2,048	4,096
# of retrieving secret exponents	1,000	1,000	100
# of required messages (average)	29.5	32	37

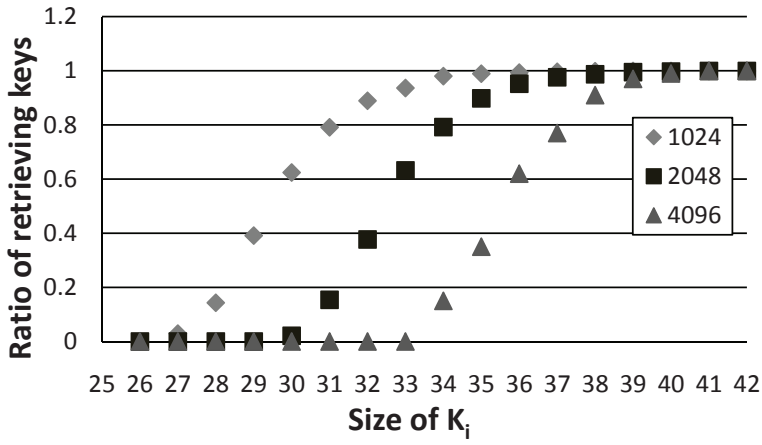

Figure 3.6: Number of required messages to retrieve secret exponents.

3.5 Discussions

We consider secure scan architecture proposed so far against our proposed scan-based attack.

Firstly, the secure scan architecture proposed in [2] cannot protect our proposed method from retrieving a secret key. [2] inserts some inverters into a scan path to invert scanned data. However, since inverted positions of scanned data are always fixed, the value of a 1-bit register sequence is only changed to its inverted value. By checking whether SS_i or inverted SS_i exist in the scanned data, our proposed method can easily make it ineffective.

Inoue's secure scan architecture [3] adds unrelated data to scanned data to confuse attackers. A sequence of scanned data to which unrelated data are added is fixed and it is not always true that they confuse all the bits to protect the scanned data in order to reduce area overhead. If the register storing scan signature SS_i is

not confused, our proposed method can easily make it ineffective, too.

Secondly, [10, 17, 18, 19, 20, 21, 22, 23, 24] require authentication to transfer between system mode and test mode, and their security depends on authentication methods. If authentication would be broken-through and attackers could obtain scanned data, a secret key in an RSA circuit could be retrieved by using our proposed method. We consider that authentication strength is a different issue from the purpose of this chapter.

Finally, [25, 26, 27] use a compactor so as not to output scanned data corresponding to registers directly. [28] proposes AES-based BIST, whereby there is no need for scan path test. However, applying these methods effectively to an RSA circuit is quite unclear because these methods are implemented only on an AES circuit or just on a sample circuit not for cryptography.

3.6 Concluding remarks

Our proposed scan-based attack can effectively retrieve a secret key in an RSA circuit, since we just focus on the variation of 1-bit of intermediate values named a scan signature. By monitoring it in the scan path, we can find out the register position specific to intermediate values. The experimental results demonstrate that a 1,024-bit secret key can be retrieved by using 29.5 messages, a 2,048-bit secret key by using 32 input, and a 4,096-bit secret key can be retrieved by using 37 messages.

Chapter 4

Scan-based Attack againt ECC

In this chapter, we propose a scan-based attack against elliptic curve cryptography (ECC) which is almost independent of a scan-path structure[1].

An elliptic curve cryptography (ECC) [7, 8] is well known as a public-key cryptography with low cost and high throughput. Finite field arithmetic is used in ECC where field multiplication requires most of the time in decryption and encryption and thus many research have been done in field multiplication [29, 30, 31, 32, 33]. Also many research on an ECC circuit implementation are reported as in [31, 32, 33, 34, 35, 36, 37, 38, 39]. For instance, architectures including memories storing all ECC parameters and field multipliers which can execute the arbitrary polynomial reduction are proposed in [31, 33] for high-throughput ECC applications. On the contrary, architectures including minimal memories storing fixed polynomial reduction and a field-dedicated multiplier are proposed in [32, 38] for low-area and low-cost ECC applications.

Retrieving a secret key in a security LSI chip by using a scan path, we have to find out positions of registers storing the secret key in the scan path. There are, however, many architectures and implementations as above in ECC and then there can be many scan-path structures as well. This means that it is very difficult to

[1]The preliminary version of this chapter appeared in [16].

find out positions of registers storing a secret key in a scan path in the ECC circuit. In other words, it is very difficult to retrieve a secret key from the scanned data. For that reason, scan-based attacks against symmetric-key cryptography succeed as reported in [9, 10, 15], but a scan-based attack against public-key cryptography such as ECC has not been proposed yet.

The proposed method is based on detecting intermediate values calculated in an ECC circuit. We focus on a 1-bit sequence which is specific to some intermediate values. Then we check whether data dependent on this intermediate value is included in the scanned data. As long as a scan path is implemented on the ECC circuit and it includes at least 1-bit of each intermediate value, we can retrieve a secret key in the target ECC circuit even if we do not know a scan path structure. The proposed method reveals the vulnerability of a scan path in the ECC circuit.

4.1 Elliptic curve cryptography

An elliptic curve cryptography makes use of the difficulty in solving the discrete logarithm problem defined in the elliptic curve additive group. This problem is called the *elliptic curve discrete logarithm problem* (ECDLP). The 160-bit key in ECC provides the equivalent security level as the 1024-bit key in RSA [6]. An ECC circuit can have higher throughput and smaller area than an RSA circuit. This section briefly explains ECC [7, 8].

4.1.1 Elliptic curve arithmetic

An elliptic curve E with non-supersingular over a field $\mathbb{F}_{2^m}$ is defined by Eqn. (4.1).

$$E : y^2 + xy = x^3 + ax^2 + b. \tag{4.1}$$

Let $E(\mathbb{F}_{2^m})$ be a group of points on the elliptic curve E. $E(\mathbb{F}_{2^m})$ has the four properties shown below and forms a group.

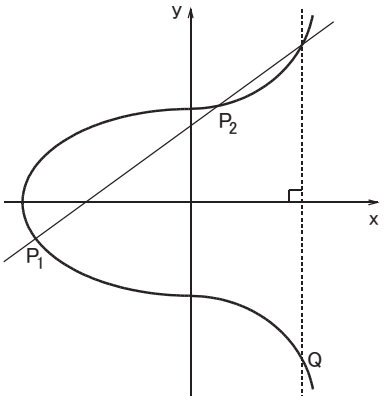

Figure 4.1: Point Addition $P_1 + P_2 = Q$.

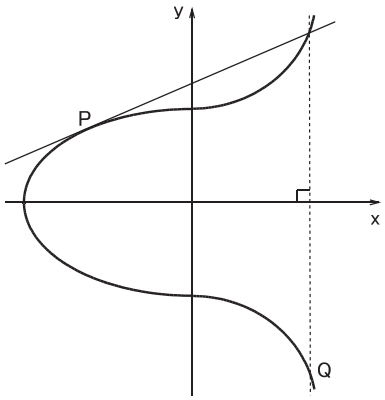

Figure 4.2: Point Doubling $2P = Q$.

1. *Identity.* $\infty \in E(\mathbb{F}_{2^m})$ is called the *identity* and it satisfies $P + \infty = \infty + P = P$ for all $P \in E(\mathbb{F}_{2^m})$.

2. *Negatives.* If $P = (x, y) \in E(\mathbb{F}_{2^m})$, then $(x, y) + (x, x + y) = \infty$. The point $(x, x + y)$ is denoted by $-P$ and is called the *negative* of P.

3. *Point addition.* Let $P_1 = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ and $P_2 = (x_2, y_2) \in E(\mathbb{F}_{2^m})$, where $P_1 \neq \pm P_2$. Then $P_1 + P_2 = (x_3, y_3) = Q \in E(\mathbb{F}_{2^m})$, where

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

   with $\lambda = (y_1 + y_2) / (x_1 + x_2)$. Figure 4.1 shows the point addition.

4. *Point doubling.* Let $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$, where $P \neq -P$. Then $2P = (x_3, y_3) = Q \in E(\mathbb{F}_{2^m})$ where

$$x_3 = \lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2}$$
$$y_3 = x_1^2 + \lambda x_3 + x_3$$

   with $\lambda = x_1 + y_1/x_1$. Figure 4.2 shows the point doubling.

4.1.2 Point multiplication

Let k be an m-bit integer and denoted as $k = k_{m-1}2^{m-1} + k_{m-2}2^{m-2} + \cdots + k_1 2 + k_0$. A *point multiplication* is defined by computing kP with k and $P \in E(\mathbb{F}_{2^m})$. The point multiplication is calculated in polynomial time by using point addition and point doubling. Given P, Q, where Q is a result of the point multiplication with k and P. To determine an integer k satisfying the equation $[kP \equiv Q \mod f(z)]$[2] is an *elliptic curve discrete logarithm problem* (ECDLP). Solving the elliptic curve discrete logarithm problem requires exponential time. If the integer k is large enough, the point multiplication $Q = kP$ can be calculated easily. However determining k from the point P and $Q \in E(\mathbb{F}_{2^m})$ requires very long time. Q can be used as a public key and k can be used as a secret key in ECC.

The point multiplication kP dominates the execution time of ECC so that several efficient algorithms have been proposed. *Montgomery method* [40] is one of point multiplication algorithms. This algorithm has two advantages. One is that it does not require any extra storage with a low calculation time. The other is that the same operations are performed in every iteration of the main loop, therefore it has a resistance against power analysis attacks [41].

The Montgomery method is first proposed in [40] and shown in Algorithm 2. It converts affine coordinate (x, y) into projective coordinates (X, Y, Z) to reduce total calculation amount. Algorithms in [42, 43, 44, 45] are also based on the original Montgomery method. In this algorithm, the secret key k is written as $2^{m-1} + k_{m-2}2^{m-2} + \cdots + k_1 2 + k_0$. k_{m-1} will be always one to achieve the same number of iterations in the main loop.

[2] $f(z)$ is an irreducible polynomial.

Algorithm 2 Montgomery method

Input: $k = (1, k_{m-2}, \ldots, k_1, k_0)_2, P \in E(\mathbb{F}_{2^m})$

Output: $Q_0 = kP$

1: $Q_0 \Leftarrow P$

2: $Q_1 \Leftarrow 2P$

3: **for** $i = m - 2$ to 0 **do**

4: $\quad Q_{1-k_i} \Leftarrow Q_0 + Q_1$

5: $\quad Q_{k_i} \Leftarrow 2Q_{k_i}$

6: **end for**

7: **return** Q_0

4.2 Attack against elliptic curve cryptography

A scan path connects registers in an circuit serially so that a tester can observe the register values inside the circuit easily. The scan path is widely used in recent circuit implementations due to its testability and easiness. We explain a scan path test in detail at Section 2.2.1.

The purpose of a scan-based attack is to retrieve a secret key from scanned data in an ECC circuit. Scan-based attack here requires several assumptions as in the previous research in [9, 10, 15] which are summarized as shown below:

1. Attackers can input an arbitrary point $P = (x, y) \in E(\mathbb{F}_{2^m})$ into a target ECC circuit.

2. Attackers can obtain scanned data from the target circuit.

In this section, we explain the scan-based attack against ECC.

4.2.1 Retrieving a secret key using intermediate values during the point multiplication

In order to retrieve a secret key k, we have to solve the discrete logarithm problem in the elliptic curve additive group. If the bit length of secret key k is more than 160, it is impossible to solve this problem within realistic time. However, if we know all the "intermediate values" during the point multiplication in Algorithm 2, we can retrieve a secret key k in a polynomial time [46].

Let $k = k_{m-1}2^{m-1} + k_{m-2}2^{m-2} + \cdots + k_1 2 + k_0$. Assume that all the intermediate values in Algorithm 2 are obtained. Let $Q_0(i)$ and $Q_1(i)$ be the intermediate values of Q_0 and Q_1 at the end of loop i in Algorithm 2, respectively.

Assume also that $k_{m-1}, k_{m-2}, \cdots, k_{i+1}$ are already retrieved. An attacker tries to reveal the next bit k_i. In this case, if and only if $k_i = 0$, either $Q_0(i-1)$ or $Q_1(i-1)$ is equal to Eqn. (4.2) below:

$$\left(\sum_{j=i}^{m-1} k_j 2^{j-i+1} + 1 \right) P. \tag{4.2}$$

Similarly, if and only if $k_i = 1$, either $Q_0(i-1)$ or $Q_1(i-1)$ is equal to Eqn. (4.3) below:

$$\left(\sum_{j=i}^{m-1} k_j 2^{j-i+1} + 3 \right) P. \tag{4.3}$$

In [46], differential power analysis attack is proposed based on the above ECC properties. Notice that, $Q_0(i-1) \neq Q_1(i-1)$ for any $1 \leq i \leq m-1$ and that $Q_0(i-1) \neq Q_0(j-1)$ and $Q_1(i-1) \neq Q_1(j-1)$ for $1 \leq i, j \leq m-1$ and $i \neq j$.

Based on the above discussion, we employ $V(i)$ defined by Eqn. (4.4) as a *selective function*:

$$V(i) = \left(\sum_{j=i}^{m-1} k_j 2^{j-i+1} + 1 \right). \tag{4.4}$$

When using the selective function above, we have to know $k_{m-1}, k_{m-2}, \cdots, k_{i+1}$. In addition to that, we assume that $k_i = 0$. $V(i) \neq V(j)$ always holds true for

Table 4.1: Intermediate values at the end of i-th loop of Algorithm 2 with input P and $k = 10_{10}$.

i	Q_0	Q_1
3	P	$2P$
2	$2P$	$3P$
1	$5P$	$6P$
0	$10P^{*1}$	$11P$

*1: The result of the point multiplication.

$i \neq j$ for $1 \leq i, j \leq m-1$. Given a point P over the elliptic curve E and $k_{m-1}, k_{m-2}, \cdots, k_{i+1}$, we assume that $k_i = 0$ and check whether $V(i)P$ appears somewhere in intermediate values. If it appears in them, we determine k_i as zero. If not, we determine k_i as one.

Finally, the LSB of a secret key k is determined by using the final point multiplication result. Since a point multiplication result $Q = kP$ is a public key itself, it must be obtained easily.

Example 3 *Let us consider that the 4-bit secret key $k = 10_{10} = 1010_2$, i.e, $k_3 = 1$, $k_2 = 0$, $k_1 = 1$, $k_0 = 0$, and $m = 4$ but assume that we do not know k except for its bit length. The intermediate values $Q_0(i)$ and $Q_1(i)$ in Algorithm 2 are summarized in 4.1.*

Now we try to retrieve the 4-bit secret key k using intermediate values. Since we know that k has four bits, k can be written as $k = xxxx$, where x shows the unknown bit. In Algorithm 2, MSB of k is defined by one. Then $k = \underline{\mathbf{1}}xxx$.

Next we try to retrieve the second bit k_2 ($i = 2$) of k. The MSB of k is one by definition ($k_3 = 1$). We assume here that $k_2 = 0$. Then $V(1)$ is calculated as $V(1) = 5$. Since $5P$ appears in 4.1, then k_2 is retrieved as zero, i.e., $k = 1\underline{\mathbf{0}}xx$.

After that we try to retrieve the third bit k_1 ($i = 1$) of k. We have already known that $k_3 = 1$ and $k_2 = 0$. We assume here that $k_1 = 0$. $V(0)$ is calculated as $V(0) = 9$. Since $9P$ does not appear in 4.1, then k_1 is retrieved as one, i.e., $k = 10\underline{\mathbf{1}}x$.

Finally, we can have the point multiplication result $10P$ as in Table 4.1. If $k = 101\underline{\mathbf{0}}$,

then $kP = 10P$. *If* $k = 101\underline{\mathbf{1}}$, *then* $kP = 11P$. *Since the result is* $10P$, *then we can have* $k = 101\underline{\mathbf{0}}$.

4.2.2 Problems to retrieve a secret key using a scan path

If we retrieve an m-bit secret key using an exhaustive search, we have to try 2^m possible values to do it. On the other hand, the method explained in Section 4.2.1 retrieves a secret key one-bit by one-bit from MSB to LSB. It tries at most $2m$ possible values to retrieve an m-bit secret key. Further, the method just checks whether $V(i)P$ is in intermediate values of Algorithm 2.

In order to apply this method to a scan-based attack, we have to know which registers store intermediate values, i.e., we have to know correspondence between scanned data and (Q_0, Q_1).

However, a scan path is usually designed automatically by CAD tools so that nearby registers are connected together to shorten the scan path length. Only designers can know the correspondence between scanned data and registers and thus retrieved scanned data can be considered to be "random" for attackers. Therefore, it is very difficult to find out the values of $V(i)P$ in scanned data for attackers. As indicated before, an ECC circuit have very complicated architecture, its scan path can include too many registers other than those storing intermediate values.

We have to find out only $V(i)P$ somehow in the scanned data to retrieve a secret key k using the method in Section 4.2.1.

4.3 Analysis scanned data obtained from an ECC circuit

In order to solve the problem that attackers do not know the correspondence between registers of the scanned data and ones storing intermediate values during point multiplication, we focus on the general property on a scan path below:

Property 1 *A bit position of a particular register r in a scanned data when giving one input data is exactly the same as that when giving another input data.*

This property is clearly true, since a scan path is fixed in an LSI chip and the order of connected registers in its scan path is unchanged.

If we execute point multiplication for each of n points on an ECC circuit, a bit pattern of a *particular* bit position in scanned data for these n points gives n-bit data. Based on the above property, this n-bit data also may give a bit pattern of a particular bit in some intermediate values when we give each of these n points to the ECC circuit.

By using the same n points we can calculate $V(i)P$ from k_{m-2} down to k_1 of the secret key k. By picking up a particular bit (LSB, for example) in each of $V(i)P$ values for n points, we also have an n-bit data. If n is large enough, this n-bit data gives information completely unique to $V(i)P$. We can use this n-bit data as a *discriminator* D_i to $V(i)P$ in scanned data.

Our main idea in this section is that we find out a discriminator D_i to $V(i)P$ in scanned data to retrieve the secret key k from k_{m-2} down to k_1. If an n-bit discriminator D_i appears in the scanned data for n points, k_i is determined as zero. If not, it is determined as one.

In the rest of this section, we firstly propose a discriminator D_i to $V(i)P$. Secondly we propose an overall method to retrieve a secret key k using discriminators. Thirdly we analyze the probabilities of successfully retrieving a secret key by using our method.

4.3.1 Calculating a discriminator to $V(i)P$

Assume that n points $P_1, \cdots, P_n$ over the elliptic curve E are given. Also assume that we have already known $k_{m-2}, \cdots, k_{i+1}$ for a secret key k. Assuming that $k_i = 0$, we can calculate $V(i)P_r$ for $1 \leq r \leq n$. As Figure 4.3 shows, we define a

discriminator D_i to be a set of LSBs of $V(i)P_r$ [3]. If n is large enough, the discriminator D_i must give information unique to $V(i)P_r$ for $1 \leq r \leq n$. Consequently, if D_i appears in scanned data, k_i is determined as zero. If not, k_i is determined as one. After k_i is determined, we can continue to determine next bit of the secret key k in the same way.

Our proposed method has two advantages compared to conventional scan based attacks [9, 10]. One is that our method is effective in the case of partial scan architecture. As long as a scan path includes at least 1-bit of each intermediate value, we can check whether the discriminator whether exists or not in the scanned data.

The other is that our method can crack the secure scan technique by [2], which inserts inverters into the internal scan path to complicate the scan structure. It protects Yang's method [9, 10] with low area cost. However, the value of a 1-bit register sequence is only changed to its inverted value. The variation of scanned data obtained by [2] is not enough to prevent our proposed method from retrieving a secret key. The detailed discussion will be described in Section 4.4.4.

[3]Since $V(i)P_r$ shows the point in XZ-plane, it has its X-coordinate and Z-coordinate. In our method, we just pick up LSB of its X-coordinate as in Figure 4.3.

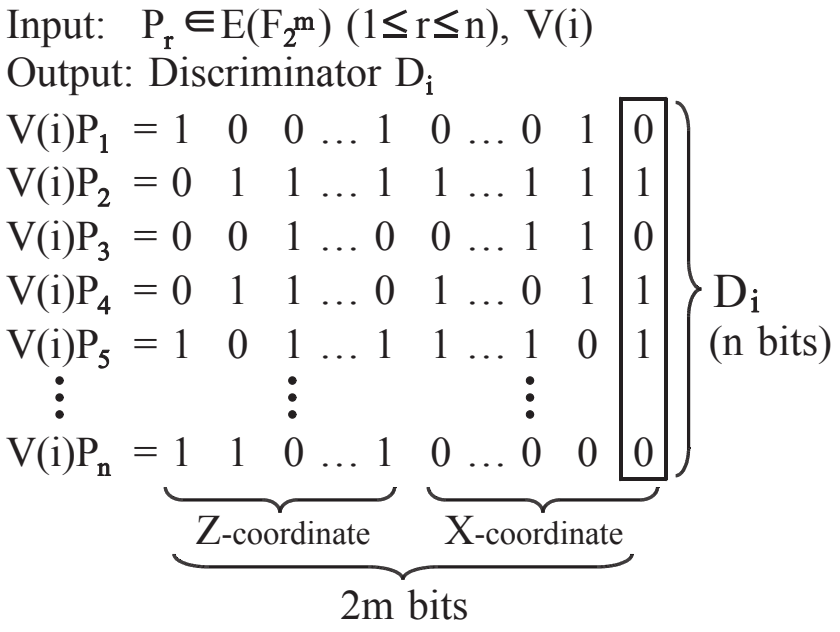

Figure 4.3: Discriminator D_i.

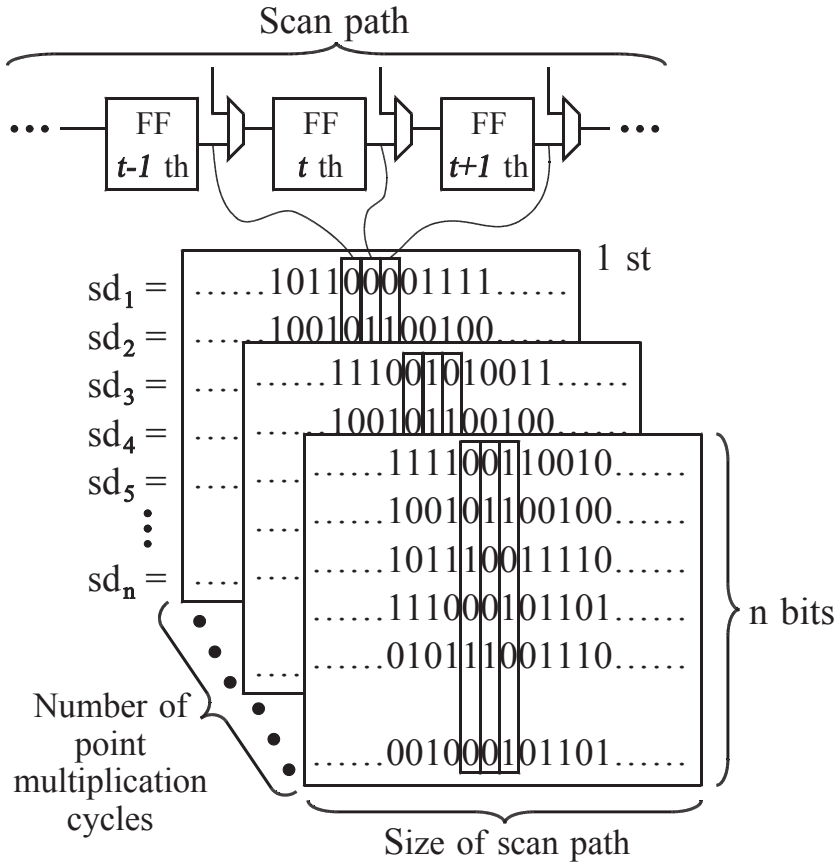

Figure 4.4: Scanned data.

4.3.2 Scanned data analysis method

First we prepare n points $P_1, \cdots, P_n$ over the elliptic curve E and give them to an ECC circuit. For each of these points, we obtain all the scanned data from the scan out of the ECC circuit until the ECC circuit outputs the point multiplication result. As Figure 4.4 shows, the size of scanned data for each of these points is ("scan path length" $\times$ "number of point multiplication cycles.")

Now we check whether a discriminator D_i to $V(i)P$ appears in the obtained scanned data under the assumption that we do not know a secret key k in the ECC circuit as follows:

1. Prepare n points $P_1, P_2, \cdots, P_n \in E(\mathbb{F}_{2^m})$, where $P_r \neq P_s$ for $1 \leq r, s \leq n$ and $r \neq s$.

2. Input P_r $(1 \leq r \leq n)$ into the target ECC circuit and obtain scanned data every one cycle during point multiplication until the ECC circuit outputs the result. Let sd_r denote the obtained scanned data for the point P_r $(1 \leq r \leq n)$.

3. Calculate $V(m-2)P_r$ assuming $k_{m-2} = 0$ for each P_r $(1 \leq r \leq n)$ and obtain the discriminator D_{m-2} to $V(m-2)P_r$.

4. Check whether the discriminator D_{m-2} exists in the scanned data $sd_1, \cdots, sd_n$. If it exists, then we can find out that k_{m-2} is equal to 0, and if it does not exist, then we can find out that k_{m-2} is equal to 1.

5. We can determine $k_{m-3}, k_{m-4}, \cdots, k_1$ in the same way as Step 4.

6. k_0 (LSB of a secret key k) is determined by comparing the expected kP value with the point multiplication result outputted by the ECC circuit.

We show the example below to explain how the method above works.

Example 4 *As in Example 1, let us consider that the 4-bit secret key* $k = 10_{10} = 1010_2$, *i.e,* $k_3 = 1$, $k_2 = 0$, $k_1 = 1$, $k_0 = 0$, *and* $m = 4$ *but assume that we do not know* k *except*

for its bit length and $k_3 = 1$. *k can be written as* $k = \underline{\mathbf{1}}xxx$, *where x shows an unknown bit. Assume that the cycle counts of point multiplication are 4 and the size of the scan path is 62 in the target ECC circuit.*

First we prepare 8 *points* $P_1, P_2, \cdots, P_8 \in E(\mathbb{F}_{2^4})$, *where* $P_r \neq P_s$ *for* $1 \leq r, s \leq 8$ *and* $r \neq s$. *The target ECC circuit executes the point multiplication as in Table 4.1.*

We input P_r *(*$1 \leq r \leq 8$*) into the target ECC circuit and obtain scanned data every one cycle during point multiplication until the ECC circuit outputs the result. Let* sd_r *denote the obtained scanned data for the point* P_r $(1 \leq r \leq 8)$. *The total size of scanned data is* $4 \times 62 = 248$ *(see Figure 4.5).*

The MSB of k is one by definition ($k_3 = 1$*). Let us start to determine* k_2. *We calculate* $V(2)P_r = 5P_r$ *assuming* $k_2 = 0$ *for each* P_r *(*$1 \leq r \leq 8$*) and obtain the discriminator* D_2 *to* $5P_r$ *(see Figure 4.6). As Figure 4.6 shows, the discriminator* D_2 *becomes "10011011". Since we find out that the discriminator* D_2 *exists in bit patterns of scanned data* $sd_r (1 \leq r \leq 8)$ *in Figure 4.5, we can determine that* k_2 *is equal to zero, i.e.,* $k = 1\underline{\mathbf{0}}xx$.

Next let us determine k_1. *We calculate* $V(1)P_r = 9P_r$ *assuming* $k_1 = 0$ *for each* P_r *(*$1 \leq r \leq 8$*) and obtain the discriminator* D_1 *to* $9P_r$ *(see Figure 4.7). As Figure 4.7 shows, the discriminator* D_1 *becomes "01010100". Since we find out that the discriminator* D_1 *does not exist in bit patterns of scanned data* $sd_r (1 \leq r \leq 8)$ *in Figure 4.5, we can determine that* k_1 *is equal to one, i.e.,* $k = 10\underline{\mathbf{1}}x$.

Finally let us determine k_0. *If* $k = 101\underline{\mathbf{0}}$, *then* $kP = 10P$. *If* $k = 101\underline{\mathbf{1}}$, *then* $kP = 11P$. *We calculate* $10P_1$ *and* $11P_1$ *and compare each of them with the point multiplication result* kP_1. *The point multiplication result obtained by the ECC circuit is* $10P_1$ *and we can determine that* k_0 *is equal to zero, i.e.,* $k = 101\underline{\mathbf{0}}$. *Therefore we can retrieve the secret key* $k = 10_{10} = 1010_2$.

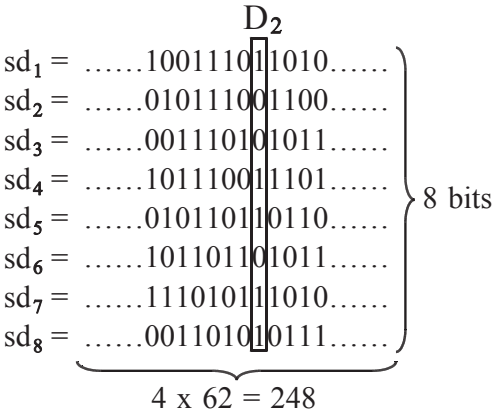

Figure 4.5: Scanned data example.

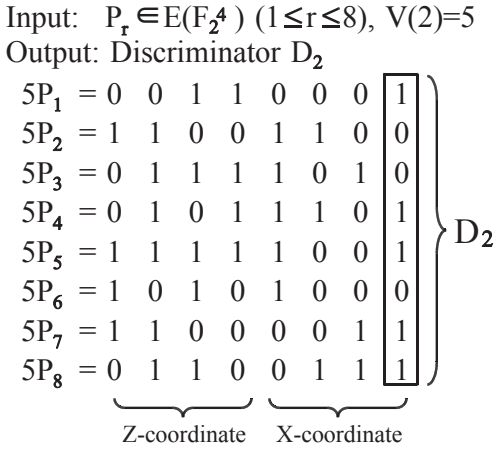

Figure 4.6: Discriminator D_2.

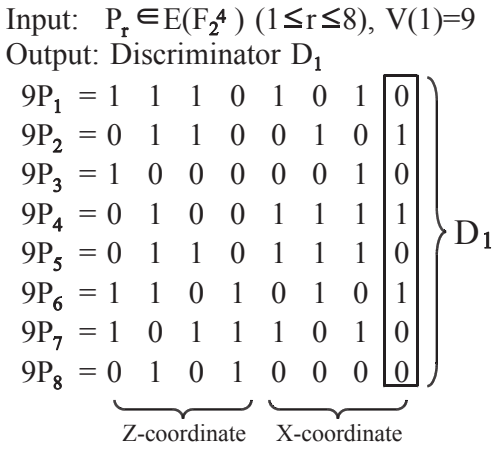

Figure 4.7: Discriminator D_1.

4.3.3 Possibility of successfully retrieving a secret key

Given that the scan size is α bits and the cycle counts to obtain point multiplication is T. Assume that scanned data are completely random data.

Even though $V(i)P_r$ for $1 \leq r \leq n$ is not calculated in the target ECC architecture, its discriminator may exist in scanned data. When $\alpha T < 2^n$, the probability that the discriminator D_i to $V(i)P_r$ exists in somewhere in bit patterns of scanned data sd_r $(1 \leq r \leq n)$ is $\alpha T/2^n$ despite $V(i)P_r$ does not calculate.

Sufficiently large n can decrease the probability that we mistakenly find out the discriminator D_i in scanned data. For instance, If α is 2,520, T is 15,137, and n is 32 [4], then the probability that we mistakenly find out the discriminator D_i in scanned data is $2,520 \times 15,137/2^{32} \simeq 8.88 \times 10^{-3}$, which is low enough. If α is 25,200, T is 15,137, and n is 36, then the probability that we mistakenly find out the discriminator D_i in scanned data is $25,200 \times 15,137/2^{36} \simeq 5.55 \times 10^{-3}$, which is also low enough.

4.4 Experiments and performance analysis

Let us analyze the number of points n required to retrieve a secret key k by using our proposed method. n must be large enough to be unique to $V(i)P_r$ $(1 \leq r \leq n)$. But it must be small enough to make retrieving time as short as possible.

In this section, we retrieve some secret keys in the practical ECC architecture to determine the appropriate number of points n by using our method. We generate randomly 1,000 secret keys and retrieve each of them. Then we calculate the number of points required to correctly retrieve the secret keys.

[4] These values are derived from the experiments in Section 4.3.

4.4.1 Architecture of an elliptic curve cryptography circuit

Block diagram of the target ECC architecture for our scan-based attack is shown as in Figure 4.8 and Figure 4.9. Its architecture is based on [34, 47] and it executes point multiplication using the López's method [43], an improved version of the Montgomery method. The method requires only one inversion and reduces the number of the multiplications compared with other point multiplication algorithms. The ECC architecture has an adder, a multiplier and a square unit over $\mathbb{F}_{2^m}$. These computing units can operate in parallel so that they can improve throughput effectively. Registers are used for input data, temporary data, and parameters for ECC. The ECC architecture also has registers for a secret key k and attackers cannot access these registers directly. In this ECC architecture, its secret key k can be set to be an arbitrary value beforehand.

We have designed the ECC architecture in Verilog HDL and synthesized it using Synopsys Design Compiler A-2007.12-SP3 with STARC 90nm process library. A scan path has been implemented automatically using Synopsys DFT Compiler. We have obtained scanned data from the gate-level ECC circuit using HDL simulator Synopsys VCS-MX B-2008.12[5].

The implementation result indicates that the delay time is 1.66 ns, the area is 32.5k gates and the total number of registers is 2,520 bits. Using this ECC architecture, the point multiplication requires 15,137 cycles.

[5]This work is supported by VLSI Design and Education Center(VDEC), the University of Tokyo in the collaboration with Synopsys Corporation and with STARC.

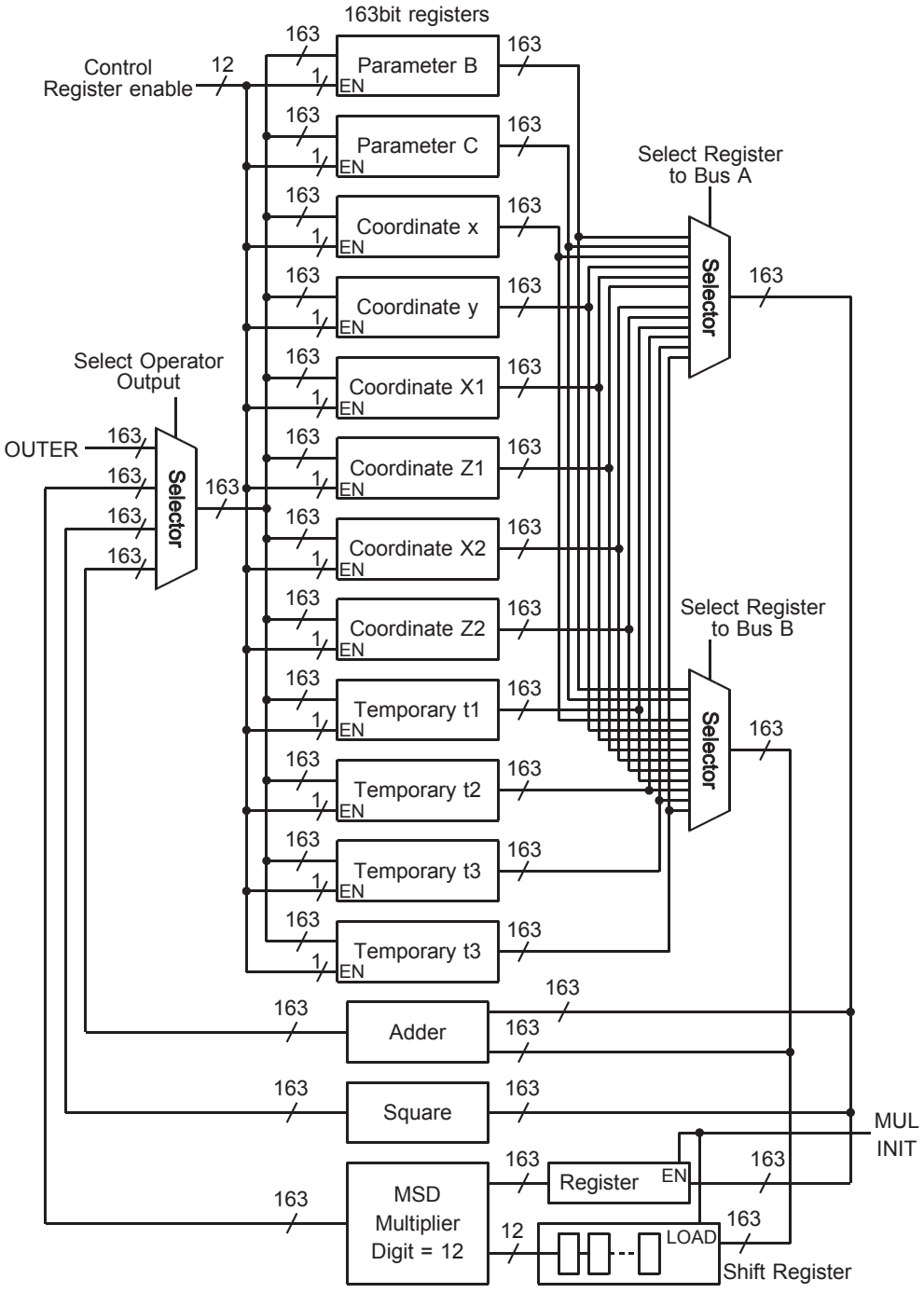

Figure 4.8: Block diagram of the elliptic curve cryptography (Data path).

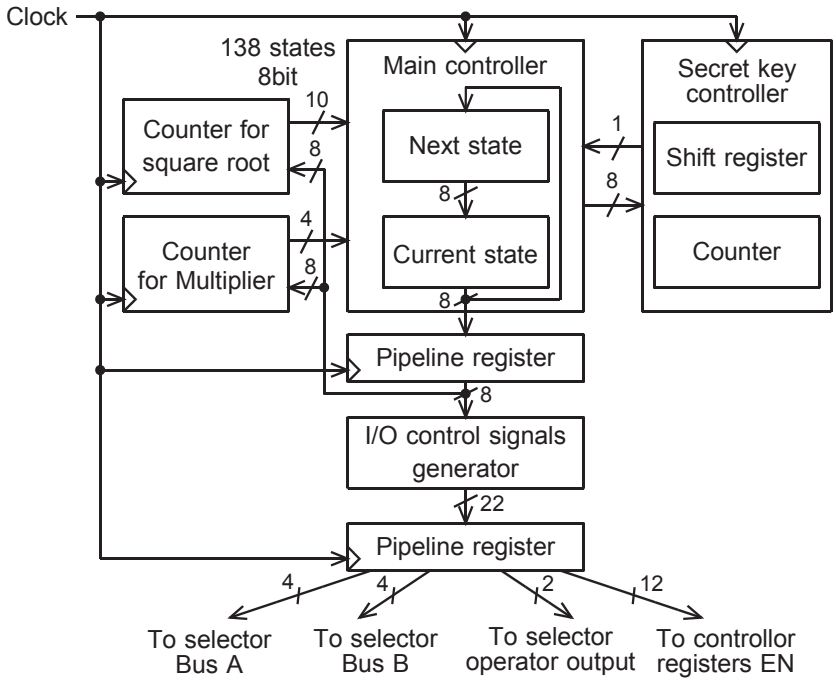

Figure 4.9: Block diagram of the elliptic curve cryptography (Controller).

4.4.2 Target scan path architecture

For simplicity, the scan path used by our experiment just includes all the registers in the target ECC architecture. This means that it also includes the shift registers storing the secret key and registers for the controller in our experiment. However, we assume that attackers just attack scanned data in the data path in the ECC circuit. This is because of the following reasons:

A controller architecture depends on implementation approach and is essentially unrelated to cryptography algorithm. For example, our ECC circuit uses a state machine as a controller but the ECC architecture in [31] uses a user-configurable circuit as a controller. Unlike cryptography algorithm, the controller architecture does not have to be open, and it is very hard for attackers to know what kind of controllers are used in a cryptography circuit.

On the other hand, a modern cryptography algorithm has to be open to check its security and we need to know it to realize a secure communication. Attackers can easily know cryptography algorithm used by a target cryptography LSI.

Our proposed attacking method is based on an ECC algorithm and attackers know its algorithm using a target ECC LSI much easier than its controller architecture. We can say that scan-based attacks analyzing a data path is more practical than those analyzing a controller.

4.4.3 Results

We have implemented the analysis method proposed in Section 4.3 in C on the SuSE Linux 9, Intel Xeon 3.4GHz, and 4GB memories and performed the following experiments.

First, we have generated 1,000 secret keys randomly. Each of the generated secret keys has a bit length of 163. Next, we have given each of the 1,000 secret keys into the target ECC circuit and obtained scanned data. Total size of the obtained scanned data for each secret key is $2,520 \times 15,137 = 38,145,240$ bits.

Table 4.2: The experimental results

Key bit length *bit*	163
Number of retrieving keys	1,000
Number of required points (Average)	29
Number of required points (Worst)	36
Retrieving time *second*	$\leq$40

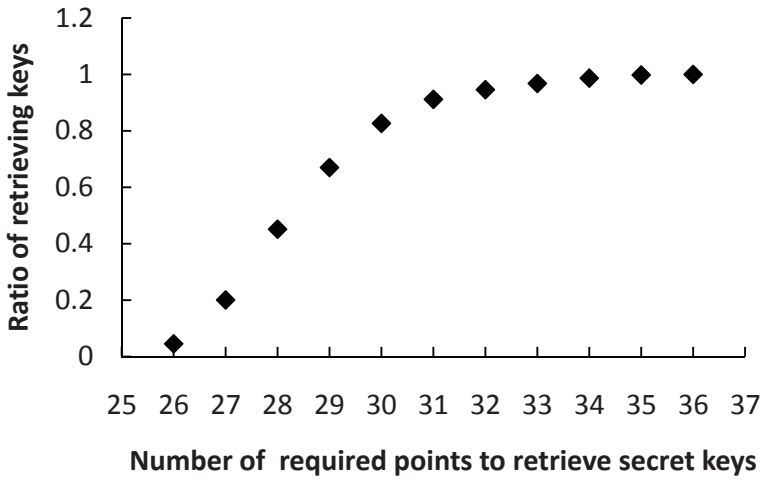

Figure 4.10: Number of required points to retrieve secret keys.

Using these scanned data, we have retrieved each of the secret keys by using our proposed analysis method. Figure 4.10 and 4.2 show the retrieving results. Figure 4.10 shows a histogram which demonstrates the number n of required points to retrieve each secret key versus its frequency. For example, the 572th secret key is 0x7e5f91be081095bf9eb1bc5d1e46f0001cb1d7b32. In order to retrieve this secret key, we need 28 points, i.e., n is 28. In this case, we can successfully retrieve the 572nd secret key using 28 points but fail to retrieve it using 27 points or less. Throughout this experiment, the required number of points is 29 on average and 36 in the worst case. A retrieving time is at most 40 seconds when analyzing one secret key.

4.4.4 Discussions

Some secure scan architecture without consideration of a 1-bit sequence which is specific to some intermediate values cannot protect against our method. Here, we consider secure scan architecture against our proposed scan-based attack proposed so far.

Firstly, the most straightforward method against our proposed scan-based attack is to keep scan path open after testing the chip. However, scan path can be reconnected and be accessed by cracking the package [48].

Secondly, the secure scan architecture proposed in [2] cannot protect against our proposed method from retrieving a secret key. [2] inserts some inverters into a scan path to invert scanned data as shown in Figure 4.11. However, since the value of a 1-bit register sequence is only changed to its inverted value, the variation of scanned data is not enough to prevent attackers from checking whether the discriminator exists or not. For instance, assume that the discriminator D_i is $10100\ldots1$ and we check whether the discriminator D_i exists or not in the scanned data $sd_1, sd_2, \cdots, sd_n$ modified by [2] as shown in Figure 4.11. If the discriminator D_i exists in the modified scanned data, we can successfully find out that k_i is zero. If not, we check whether the inverted discriminator $D_{i-inv} = 01011\ldots1$ exists or not. If the inverted discriminator D_{i-inv} exists in the modified scanned data, we can find out that k_i is zero. If the inverted discriminator D_{i-inv} does not exist, we can find out that k_i is one.

[3] adds unrelated data to scanned data to confuse attackers as shown in Figure 4.12. However, a sequence of scanned data to which unrelated data are added is fixed in each LSI chip and it just confuses only a part of scanned data to achieve lower area overhead. In other words, unmodified bits exist in the scanned data $sd_1, sd_2, \cdots, sd_n$ modified by [3]. In this case, if the discriminator D_i exists in the modified scanned data, we can successfully find out that k_i is zero. If not, we check whether the disicriminator D_i^1 calculated when k_i is one exists or not

in the modified scanned data because a discriminator is defined as not only when k_i is zero but also when k_i is one. If the discriminator D_i^1 exists in the modified scanned data, we can successfully find out that k_i is one. Even if these discriminators do not exist in the modified scanned data, we can use other discriminators like $D_{i1}, D_{i2}, \cdots$ as shown in Figure 4.12, which are defined as a set of other bits of $V(i)Pr$ for $1 \leq r \leq n$. If one of these other discriminators exists in the modified scanned data, we can find out that k_i is zero. Consequently, [3] cannot completely protect against our method.

Thirdly, [17, 18, 19, 20, 21, 22, 23, 24] require authentication to transfer between system mode and test mode, and their security depends on authentication methods. If authentication would be broken-through and attackers could obtain scanned data, a secret key in an ECC LSI could be retrieved by using our proposed method. We consider that authentication strength is a different issue from the purpose of this chapter.

Yang's method [10] limits transition between test mode and system mode to prevent attackers from obtaining scanned data during encryption/decryption using the secret key in their cryptography circuit. However, it could not support in-field testing required for high reliable LSI.

Finally,[25, 26, 27] use a compactor so as not to output scanned data corresponding to registers directly. [28] proposes AES-based BIST, whereby there is no need for scan path test. However, applying these methods effectively to an ECC LSI is quite unclear because these methods implement only an AES circuit or just a sample circuit not for cryptography.

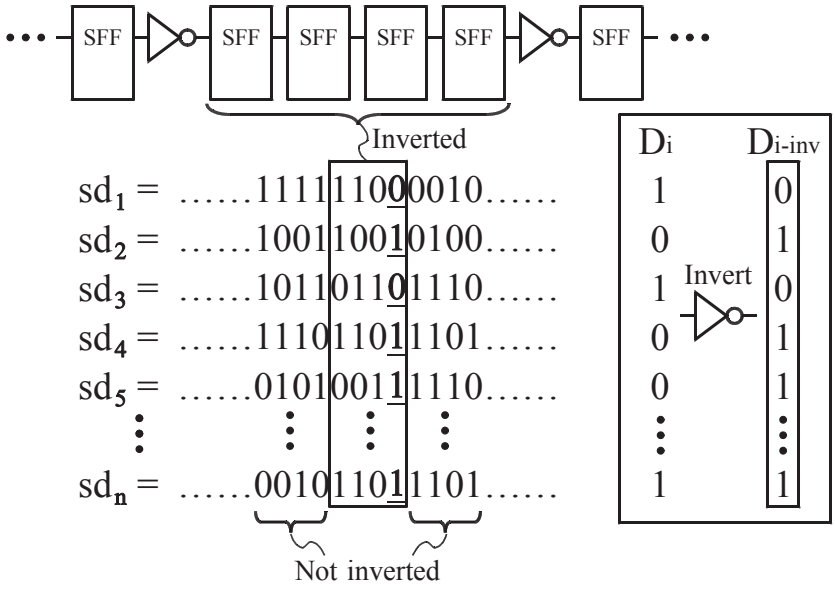

Figure 4.11: Scanned data modified by [2].

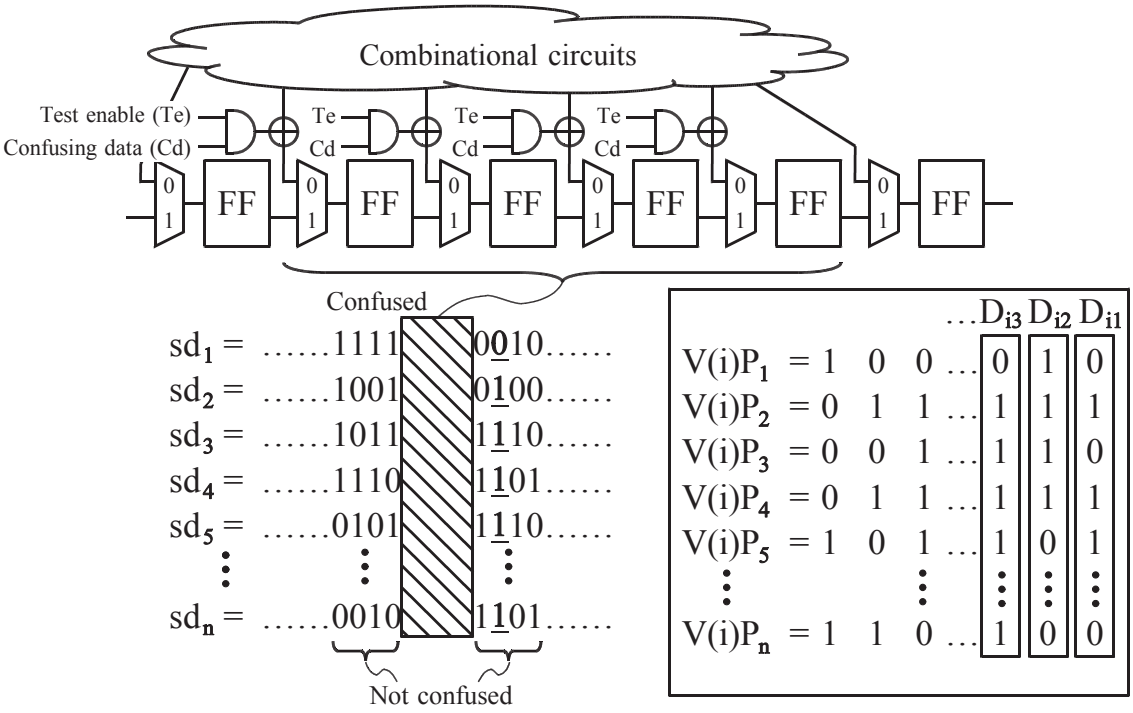

Figure 4.12: Scanned data modified by [3].

4.5 Concluding remarks

We have focused on a scan-based attack against an ECC circuit. Three scan-based attacks against symmetric-key cryptography are reported [9, 10, 15] but those against public-key cryptography are not reported yet. Since public-key cryptography are more complex than symmetric-key cryptography, scan-based attacks against symmetric-key cryptography cannot directly applied to retrieve a secret key in public-key cryptography circuit.

Our proposed scan-based attack can effectively retrieve a secret key k in an ECC circuit, since we just focus on the variation of 1-bit of intermediate values. By monitoring it in the scan path, we can find out the register position specific to intermediate values. The experimental results demonstrate that a secret key in a practical ECC circuit architecture can be retrieved by using 29 points over the elliptic curve E within 40 seconds. We can say that the proposed method reveals the vulnerability of a scan path in an ECC circuit.

In this chapter, we deal with an elliptic curve cryptography over $GF(2^m)$. But even if we deal with an elliptic curve cryptography over $GF(p)$, where p is prime, the intermediate values during the point multiplication are determined by its inputs and a secret key, and consequently, our proposed method can retrieve a secret key in the similar way.

Chapter 5

State-dependent secure scan architecture

In this chapter, we propose a new secure scan architecture having tolerability against [15] by changing structure of a scan path dynamically even after it is designed.

A test for manufactured chips individually is very important for offering high quality LSI chips. Recently, circuit size dramatically increases because process technology makes remarkable progress and CAD tools become widespread. We have to consider a test for LSI when we design circuits because it is more and more difficult to test a whole circuit completely.

Scan test is a powerful and popular test technique because it achieves high fault coverage and is implemented easily. Scan test architecture is designed by connecting scan FFs(flip-flops) inside circuit, which are registers for a scan test. in series, which is called a *scan path*. It has input and output pins outsides the chip to control and observe the internal states of the circuit.

Since FF is accessible from outside circuit in a scan path, there is a threat for obtaining confidential information such as a secret key which is used for cryptography circuits. In fact, scan based attacks are already proposed [9, 12, 15], which

retrieves the secret key by analyzing scanned data obtained from a cryptography circuit. It is non-straightforward to analyze, because the connection of scan FFs is almost random by each layout. However, scan based attacks solve this connection problem using the characteristic of a scan path.

Secure scan architecture to defend scan based attacks is divided into 2 patterns. One is a restriction that no one can obtain original scanned data without permission. Only testers can obtain them for test. This method requires a circuit and a controller for restriction but security settings can be reasonably flexible [12, 19, 23]. However, there are 3 demerits as follows: the circuit and the controller needs to be re-designed for each cryptography circuit, area overhead for the circuit and the controller is too large, and if attackers access a scan path with permission, it is easy to retrieve a secret key by using scan-based attacks.

The other method is making secret information unretrieveable for attackers even if they can access a scan path. [2] changes inputs and outputs of a scan path in inside circuit. Even if attackers obtain scanned data, they do not understand the internal states without modification, and consequently, they cannot use scan-based attacks. This method does not require a controller and only requires simple circuits to change data. It also builds a secure scan path automatically by using CAD tools, and furthermore, it can be adapted to any intellectual property easier. This method has more advantages compared to the first method. However, there is 1 demerit as follows: [2] cannot defend a certain scan-based attack. The scan-based attack proposed by [15] can retrieve a secret key even by using scanned data changed by [2].

To defend the scan-based attack [15], our proposed method use a State-dependent Scan FF(SDSFF) we propose changes an scan FF output using a latch memorizing a past value of the scan FF. Our proposed method can change a security level flexibly by considering objective circuits, and it also does not need any controller, and then, area overhead is small.

5.1 Scan-based attacks

A scan-based attack is one of side channel attacks based on information gained from the physical implementation of a cryptography. A scan path comprises scan FFs connected one another in serial. We explain a scan path test in detail at Section 2.2.1.

By using "Scan In", "Capture", and "Scan Out" through a scan path, which is same procedures as test, attackers obtain scanned data to know the internal states of a cryptography circuits. [9, 12] proposed the method to make use of the hamming weight of scanned data to find out the internal states. This method only needs the correspondence between the first column of AES and the scanned data, but does not need the bit-to-bit correspondence between them.

On the other hand, [15] proposes a scan-based attack which is almost independent of a scan-path structure. This method checks whether a 1-bit sequence which is specific to some intermediate values is included or not in scanned data. As long as a scan path is implemented on an AES circuit and it includes at least 1-bit of each intermediate value, [15] can retrieve a secret key even if the scan path structure is unknown.

5.2 Secure scan architecture

Secure scan architecture for scan based attacks can be divided into 2 types.

Method1: Make scan path unusable for attacker by private controller limitation.

Method2: Make scan path usable for anyone, but make secret information undecodable.

Generally, a security level will be threatened when mode jump occurs such as "system mode" to "test mode" or "test mode" to "system mode". For this reason,

method 1 protects circuits from scan based attacks by restricting mode jumps using a test controller [12, 19, 23].

[12] proposes mirror key register (MKR) architecture to protect secret information from attacker. A secret key, which is stored with ROM in a cryptography chip, only loaded at system mode in secure mode. Testers make a cryptography chip jump to insecure mode from secure mode for test and that occurs only a power off reset. Therefore, attackers cannot obtain scanned data during operation. However, the same number of mirror key registers as the number of scan FFs which hold target data are required, so this method has large area overhead.

[19] can automatically detect whether or not a scan path comes to a test mode using Spy Scan FF (SpySFF) inside a scan path. All mode jumps are restricted so that secret information is safe, but this condition makes primal required test unexecutable. For this reason, mode jump by enable signal tree is permitted. This exception is a technique that mode jump is permitted only when output of some scan FF inside scan path is equal to designated patterns. However, re-designing of test controller is required for each circuit, also area overhead is large.

In [23], Scan Out is permitted only when M keys (a length is N bits) input to specific N FFs as test vector to test start. N FFs which are used as keys are randomly chosen, and is decided at system design timing. As long as N-bit key does not input to FFs in specific order for M times, no one can obtain inside information. Probability that attackers find N-bit key is $1/2^N$, with M times input, makes final probability of $1/2^{MN}$. Also, M and N can be chosen freely, and then a security level can be chosen flexibly. On the other hand, area overhead will be larger because we need to design a controller for checking input keys M times.

[2] protects secret information by making scanned data change undecodable. This can be done by inserting a number of inverters randomly into a scan path. There are 2 patterns of which inverter is inserted or is not inserted for each scan FF. If the number of all scan FF is m, the number of structures that a scan path can take will be 2^m. Therefore, it is substantially safe because attackers only obtain

structure of the scan path in probability of $1/2^n$ and required impractical time for decoding. It does not need test controller, and consequently area overhead is very small, and it is easy to implement by using CAD tools automatically. However, it has a problem that output is fixed for each bit of scanned data. Because places of inverters is fixed after chip layout has designed, [2] cannot defend attacking method [15] which just focuses on 1-bit of register inside a scan path.

5.3 Proposed method

We propose new secure scan architecture whose path dynamically changes to have tolerability against the scan-based attack [15]. The attack method uses a characteristic that the structure of scan path is fixed once implemented. To protect a secret key against the attack method, we change the output value of a scan FF by XOR'ing with the past output value of it. To store the past output value of the scan FF, we propose a State-dependent Scan FF (SDSFF). The SDSFF changes the output value of a scan FF with internal states of a circuit.

As shown in Figure 5.1, the SDSFF stores the output value of the scan FF (SFF1) in a latch when a load signal is enable. The output value S for the next scan FF (SFF2) is calculated by XOR'ing the output value A of the latch with the output value B of scan FF1. The value A dynamically changes after the circuits implemented at the timing when the load signal is enable, and the value S changed at the same time accordingly. The values among A, B, and the output value S for scan FF2 is shown Figure 5.1.

If testers use our proposed method, they replace some normal scan FFs with SDSFFs. Scanned data are changed by SDSFFs, but testers can generate test patterns and corresponded output patterns because testers know which scan FFs are replaced with SDSFFs. Attackers, however, do not know which scan FFs are replaced in a cryptographic circuit, and then, they do not know how scanned data they obtained are changed. For this reason, our proposed method can defend the

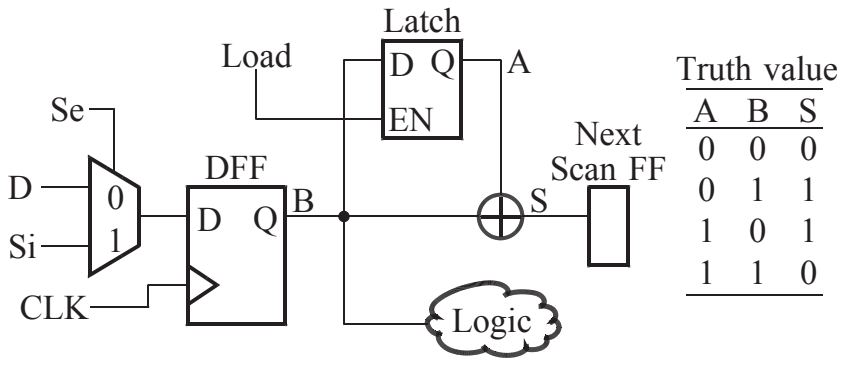

Figure 5.1: State-dependent Scan FF(SDSFF).

scan-based attack [15] and other attack methods [9, 12], too.

We discuss a test algorithm using our proposed method. The output of a scan FF changes dynamically by updating the value A with Load signal at the timing when moving from a test mode to a system mode. We show the timing chart between a clock signal, Se signal, and Load signal is shown in Figure 5.2. We defined that an cryptography LSI is operated in a system mode when Se signal is 0 and it is operated in a test mode when Se signal is 1.

While Se signal is 1, scan data shift one by one with every clock cycle, We input test patterns to scan FFs of scan path. After inputting them, we change Se signal to 0, which represents system mode. At the same time, Load signal changes to 1, and a latch of a SDSFF stores output value A of a scan FF1. Load signal changes to 0 again before the next clock rising occurs. After the clock rises in the system mode, and as executed results are stored in scan FF, we make Se signal 1 to observe the internal state of the cryptographic circuit. It is important that the output value A of the latch is updated at every timing to move from the test mode to the system mode. Thereby, structure of scan path is not fixed.

Testers can decode scanned data as they know which scan FFs are replaced with SDSFFs and the output value A of the latch. Attackers do not know where SDSFFs insert of a scan path and then they also do not know which bits of scanned data inverted, thereby they cannot decode and analyze scanned data to retrieve a

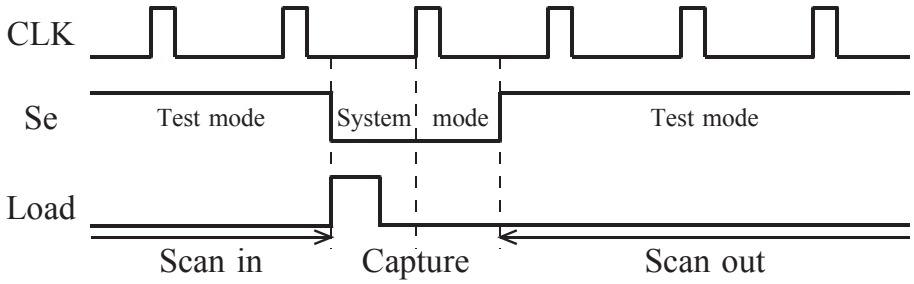

Figure 5.2: Timing chart.

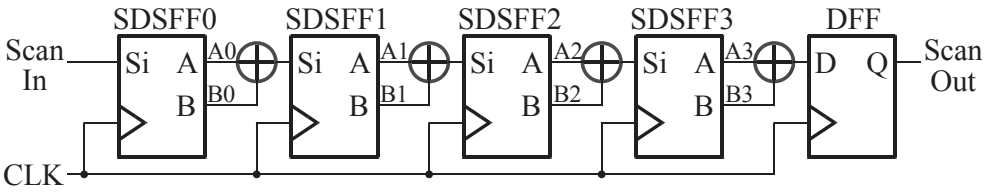

Figure 5.3: Model of proposed scan architecture.

secret key. We show the example below to explain how scanned data decode.

Example 5 *Let us consider that a scan path model which consists of four SDSFFs and one DFF for I/O as shown in Figure 5.3. We assume that four scan FFs stores* $D_1D_2D_3D_4$ *and four latches of them stores 0101, which means that the data through SDSFF A1 and A3 are inverted. We input* 1001 *as a test pattern to the scan path and show change in scanned data in Table 5.1.*

At the rising of the 1st clock cycle, four scan FFs store $1D_3\overline{D_2}D1$*. A1 inverts* D_2 *to* $\overline{D_2}$ *when* D_2 *passes through the SDSFF1. We obtain* $\overline{D_0}$*, which A3 inverts* D_0 *to* $\overline{D_0}$ *when* D_0 *pass through the SDSFF3. We input* 0 *to the scan path.*

At the rising of the 2nd clock cycle, four scan FFs store $01\overline{D_3D_2}$*. A1 inverts* D_3 *to* $\overline{D_3}$ *when* D_3 *passes through the SDSFF1. We obtain* $\overline{D_1}$*, which A3 inverts* D_1 *to* $\overline{D_1}$ *when* D_1 *pass through the SDSFF3. We input* 0 *to the scan path.*

At the rising of the 3rd clock cycle, four scan FFs store $000\overline{D_3}$*. A1 inverts* 1 *to* 0 *when it is passed through the SDSFF1. We obtain* D_2*, which A3 inverts* $\overline{D_2}$ *to* D_2 *when* $\overline{D_2}$ *pass through the SDSFF3. We input* 1 *to the scan path.*

At the rising of the 4th clock cycle, four scan FFs store 1010*. A1 inverts* 0 *to* 1 *when*

it is passed through the SDSFF1. We obtain D_3*, which A3 inverts* $\overline{D_3}$ *to* D_3 *when* $\overline{D_3}$ *pass through the SDSFF3.*

Before the 5th clock cycle is rising, we change Se signal to 0 and move to a system mode. At the same time, four latches store 1010*, and thereby positions inverting bit is changed. At the rising of the 5th clock cycle, four scan FFs store executing results* $D'_3D'_2D'_1D'_0$*. We input* 1001 *as the same test pattern as the first time.*

At the rising of the 6th clock cycle, four scan FFs store $1\overline{D'_3}D'_2\overline{D'_1}$*. A0 inverts* D'_3 *to* $\overline{D'_3}$ *when* D'_3 *passes through the SDSFF0 and A2 inverts* D'_1 *to* $\overline{D'_1}$ *when* D'_1 *passes through the SDSFF2. We obtain* D'_0 *and input* 0 *to the scan path.*

At the rising of the 7th clock cycle, four scan FFs store $01\overline{D'_3D'_2}$*. A0 inverts* 0 *to* 1 *when it is passed through the SDSFF0 and A2 inverts* D'_2 *to* $\overline{D'_2}$ *when* D'_2 *passes through the SDSFF2. We obtain* $\overline{D'_1}$ *and input* 0 *to the scan path.*

A0 inverts 0 *to* 1 *when it is passed through the SDSFF0 and At the rising of the 8th clock cycle, four scan FFs store* $000D'_3$*. A2 inverts* $\overline{D'_3}$ *to* D'_3 *when* D'_3 *passes through the SDSFF2. We obtain* D'_2 *and input* 1 *to the scan path.*

At the rising of the 9th clock cycle, four scan FFs store 1111*. A0 inverts* 0 *to* 1 *when it is passed through the SDSFF0 and A2 inverts* 0 *to* 1 *when it is passed through the SDSFF2. We obtain* D'_3*.*

5.4 Implementation and Results

We have implemented our proposed method to an AES cryptography circuit in Verilog-HDL and have synthesized it using Synopsys Design Compiler Z-2007.03-SP4 with 90nm process library[1]. The AES cryptography circuit has 716 registers in total and the length of scan path is 716, too. We have replaced n normal scan FFs in a scan path with n proposed SDSFFs and have showed area after synthesize to compared with other scan path methods.

[1]This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in the collaboration with Synopsys Corporation and with STARC.

Table 5.1: Values in Example 1.

Clock Cycle		A0	A1	A2	A3	Scan In	B0	B1	B2	B3	Scan Out
						1	D_3	D_2	D_1	D_0	–
1st						0	1	D_3	$\overline{D_2}$	D_1	$\overline{D_0}$
2nd	Scan shift	0	1	0	1	0	0	1	$\overline{D_3}$	$\overline{D_2}$	$\overline{D_1}$
3rd						1	0	0	0	$\overline{D_3}$	D_2
4th						–	1	0	1	0	D_3
	Load	1	0	1	0	–	1	0	1	0	–
5th	Capture	1	0	1	0	–	D'_3	D'_2	D'_1	D'_0	–
						1	D'_3	D'_2	D'_1	D'_0	–
6th						0	1	$\overline{D'_3}$	D'_2	$\overline{D'_1}$	D'_0
7th	Scan shift	1	0	1	0	0	0	0	$\overline{D'_3}$	$\overline{D'_2}$	$\overline{D'_1}$
8th						1	0	1	0	D'_3	$\overline{D'_2}$
9th						–	1	1	1	1	D'_3
	Load	1	1	1	1	–	1	1	1	1	–
10th	Capture	1	1	1	1	–	D''_3	D''_2	D''_1	D''_0	–

The circuit area is 19,030 gates when scan path is not implemented. Implementing scan path, the area increases 19,594 gates. Inserting 358 inverters among scan path to implement [2], we have 19,863 gates. The increased amount compared with normal scan path is 269 gates.

To implement our proposed method, we replace normal scan FFs with 45, 60, 90, 179, 358, 716 SDSFFs, and increased amounts compared with normal scan path are 203, 270, 405, 806, 1,611, 3,218, respectively. A SDSFF uses a latch and an XOR gate, and it is bigger than [2] using an inverter accordingly. In our method, when we replace 60 scan FFs with SDSFFs, area is as same as [2], over 60 scan FFs makes larger area than [2].

Table 5.2: Implementation results for an AES cryptography circuit.

	Modified Scan FFs	Area *gates*	Area Overhead *gates*
No Scan	–	19030	—
Normal Scan	–	19594	0
[2]	358	19863	269(1.4%)
Proposed method	716	22812	3218(16%)
	358	21205	1611(8.2%)
	179	20400	806(4.1%)
	90	19999	405(2.1%)
	60	19864	270(1.4%)
	45	19797	203(1.0%)

Table 5.3: Security comparison.

	[2]	Proposed method
Possible patterns against [12] *patterns*	2^n	2^n
Possible patterns against [15] *patterns*	2	2^n

If attackers obtain scanned data to retrieve a secret key using the scan-based attack, they have to find out the position of SDSFFs in scan path of a cryptography circuit. The more the number of SDSFFs is, the more difficult to find out the position of SDSFFs. Suppose the number of SDSFFs to replace is n, the combination number of possible scan path architecture will be 2^n, which means, the probability of specifying structure of scan path by attacker will be $1/2^n$. We can adjust the security level against attack methods and by increasing replaced SDSFFs.

5.5 Concluding Remarks

In this chapter, we have proposed secure scan architecture using SDSFF. We focused on the point that scan based attack cannot attack when scan path structure changes dynamically. By using this point, we used a latch for saving past state to change scan path structure dynamically. We have evaluated the security level and validity by implemented to an AES cryptography circuit.

Chapter 6

Conclusion

In this dissertation, I proposed new scan-based attacks against AES, RSA and ECC and a new secure scan architecture.

Scan-based attacks whereby we do not need the correspondence between the scanned data and the registers of cryptography ciruits storing the intermediate values, and further, we do not have to know when the registers store the intermediate values necessary for analysis.

Conventinal scan-based attacks such as Yang's method [12] has following problems so that it is much difficult to retrieve a secret key in practical security LSIs.

1. A security LSI consists of many circuit other than the cryptography circuit, such as a microprocessor, memory and a controller. Scan path in the security LSI generally includes not only registers of the cryptography circuit but also many registers of other circuits.

2. Attackers cannot know the timing when intermediate values corresponding to the secret key are stored in registers.

In order to solve these problems, we focus on the general property on scan paths below:

Property 2 *A bit position of a particular register r in a scanned data when giving one input data is exactly the same as that when giving another input data.*

This property is clearly true, since a scan path is fixed in an LSI chip and the order of connected registers in its scan path is unchanged.

A bit pattern of a *particular* bit position in scanned data for n intermediate values gives n-bit data. Based on the above property, this n-bit data also may give a bit pattern of a particular bit in these intermediate values when we give each of these n inputs to the cryptography circuit.

By using the same n inputs, we can calculate intermediate values corresponding to the secret key. By picking up a particular bit (LSB, for example) in each of intermediate values for n inputs, we also have an n-bit value. If n is large enough, this n-bit value gives information completely unique to intermediates values corresponding to the secret key. We can use this n-bit value as a *discriminator* D_i to intermediates values corresponding to the secret key in scanned data. Methodology to retrieve the secret key is following steps.

1. Prepare n inputs.

2. Calculate the cryptography algorithm with one of the n inputs by using the target cryptography circuit and obtain scanned data every one cycle until the crypgtography circuit outputs the result.

3. Calculate a discriminator from intermediate values corresponding to the secret key.

4. Check whether the discriminator exists in the scanned data. If it exists or not, then we can determine the particular bit of the secret key.

5. We can determine other bits of the secret key in the same way as Step 3-4.

Our proposed scan-based attack against AES retrieves the secret key in the AES circuit by using only 225 inputs. We success to reduce the number of inputs

by as much as half compared to Yang's AES scan-based attack. In addition, we experiment with scanned data adding random bits to it. Total bit length of scanned data is 128 (no extra data is added) to 4096 (3968-bit extra data is added). Even if the scanned data includes the 3,968-bit random data other than the round function output, our improved method is capable to decipher the secret key 321 inputs on average and 369 inputs in the worst case.

Our proposed scan-based attack against RSA succeeds in retrieving the secret key in the RSA circuit. We experiment three pattern of key bit length, 1,024 bits, 2,048 bits and 4,096 bits. Requiring the average number of inputs is 29.5, 32 and 37, respectively.

Our proposed scan-based attack against ECC succeeds in retrieving the secret key in the ECC circuit. We experiment 1,000 secret keys generated randomly and we retrieve all of the secret keys. Key size is 163 bits and the scan path length is 2,520 bits and the point multiplication requires 15,137 cycles. Requiring the number of inputs is 29 on average and 36 in the worst case.

State-dependent configurable secure scan architecture achieves effecitive countermeasure against scan-based attacks. By subsititing State-dependent scan flip-flop(SDSFF) for some scan flip-flops, scanned data is changed depend on its state. Increased area by using SDSFF to prevent scan-based attacks is only 5 through 10 % against cryptography circuit. If attackers obtain scanned data to retrieve a secret key using the scan-based attack, they have to find out the position of SDSFFs in scan path of a cryptography circuit. Suppose the number of SDSFFs to replace is n, the combination number of possible scan path architecture will be 2^n, which means, the probability of specifying structure of scan path by attacker will be $1/2^n$.

6.1 Future works

My future works are summarized as follows:

1. Proposing a theory about the number of input required for retrieving a secret key.

2. New scan-based attack against compressed scanned data.

3. Experimental evaluation using our proposed method against scanned data modified by countermeasures.

4. New scan-based attack against other crytography algorithm.

We reveal the number of input required for retrieving secret keys against some cryptography circuits. However, these numbers are obtained by experimantal results, so we have not understood the reason why our scan-based attack against ECC requires the number of inputs is 29 on average and 36 in the worst case for retrieving the secret key in the ECC circuit. By developing a theory about the number of input required for retrieving a secret key, our proposed scan-based attacks must be more effective methods.

All of scan-based attacks are based on the assumption that scanned data is directly output from registers. In practice, compressing scanned data is sometimes used in order to reduce the time of scan test. If a scan-based attack by using compressed scanned data is possible, it must be more practical attacking method.

Some countermeasure against scan-based attack are reported, but all of them do not experiment with practical circuits. Experiment of scan-based attacks using modified scanned data by countermeasure is important.

Scan-based attacks are powerful and threat against cryptography circuits. However, effective attacking method against some cryptogaphy algorithm are not found out. Camellia [49], for instance, which is one of symmetric-key cryptography algorithm, cannot not be attacked because side-channel attacks have not been reported

yet. Algorithm of Camellia considers countermeasure against side-channel attacks so that a scan-based attack against Camellia is difficult, too. Scan-based attacks against Camellia is worth researching.

Acknowledgment

First and foremost, I would like to give heartful thanks to Professor Nozomu Togawa at the department of computer science and Engineering of Waseda University who gives me constant encouragement, guidance, and support during my research. He advised me enthusiastically about not only research but also my attitude to it. Working on my research with him must be the most irreplaceable assets in my life.

I also express my appreciation to Professor Tatsuo Ohtsuki at the department of computer science and Engineering of Waseda University and Professor Masao Yanagisawa at the department of Electronic and Photonic Systems of Waseda University for continuous encouragement, support, direct guidance, and insightful comments throughout my work. They gave me a lot of helpful suggestions for my research life.

I also thank Professor Naohisa Komatsu at the department of computer science and Engineering of Waseda University whose gives me constructive comments and warm encouragement.

I also thank Professor Satoshi Goto and Professor Takeshi Ikenaga of Waseda University for giving me a lot of helpful advices and continuous encouragement during my research at Kita-kyushu.

I also thank Dr. Youhua Shi for giving me a lot of helpful advices and support my research. He kindly introduce research in a scan-based attack to me. I received generous support from Ryo Tamura for my English. I also thank Hiroshi Atobe and Kei Satoh for researching in scan-based attack. I have greatly benefited from

their research for my dissertation.

I also thank Dr. Kazunori shimizu for giving me a lot of helpful advices and support not only my research but also my life. He is one of the most respect people for me.

I also thank all of the students in Professor Togawa's laboratory, Professor Ohtsuki's laboratory, and Yangisawa's laboratory for their cooperations.

Finally, I thank my families for their kind support over the years.

References

[1] T. S. Messerges, E. A. Dbbish, and R. H. Sloan, "Power analysis attacks of modular exponentiation in smartcards," in *Proc. Workshop on Cryptographic Hardware and Embedded Systems*, ser. Lecture Notes in Computer Science, vol. 1717, pp. 44–157, 1999.

[2] G. Sengar, D. Mukhopadhyay, and D. R. Chowdhury, "Secured flipped scan-chain model for crypto-architecture," *IEEE Trans. VLSI Syst.*, vol. 26, no. 11, pp. 2080–2084, 2007.

[3] M. Inoue, T. Yoneda, M. Hasegawa, and H. Fujiwara, "Partial scan approach for secret information protection," in *Proc. European Test Symposium*, pp. 143–148, 2009.

[4] "Data encryption standard (DES)," Federal Information Processing Standards Publication 46-3 (FIPS 46-3), National institute of standards and technology (NIST), Tech. Rep., 1999.

[5] "Advanced encryption standard (AES)," Federal Information Processing Standards Publication 197 (FIPS 197), National institute of standards and technology (NIST), Tech. Rep., 2001.

[6] R. L. Rivest, A. Shamir, and L. Adelman, "A method for obtaining digital signature and public-key cryptsystems," vol. 21, pp. 120–126, 1978.

[7] V. Miller, "Uses of elliptic curves in cryptography," in *Proc. the Advances in Cryptology*, H. Williams, Ed., pp. 417–426, 1986.

[8] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203–209, 1987.

[9] B. Yang, K. Wu, and R. Karri, "Scan based side channel attack on dedicated hardware implementations of data encryption standard," in *Proc. International Test Conference 2004*, pp. 339–344, 2004.

[10] ——, "Secure scan: a design-for-test architecture for crypto chips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 10, pp. 2287–2293, 2006.

[11] S. Mangard, M. Aigner, and S. Dominikus, "A hignly regular and scalable AES hardware architecture," *IEEE Trans. Comput.*, vol. 52, no. 1, pp. 483–491, 2004.

[12] B. Yang, K. Wu, and R. Karri, "Secure scan: a design-for-test architecture for crypto chips," in *Proc. the 42nd Design automation Conference*, pp. 135–140, 2005.

[13] R. D. Silverman, "Has the RSA algorithm been compromised as a result of Bernstein's Paper?" RSA Laboratories, Tech. Rep., Apr. 2002. [Online]. Available: http://www.rsa.com/rsalabs/node.asp?id=2007

[14] J. Stein, "Computational problems associated with Racah algebra," *J. Computational Physics*, vol. 1, pp. 397–405, 1967.

[15] R. Nara, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "A scan-based attack based on discriminators for AES cryptosystems," *IEICE Trans. Fundamentals*, vol. E92–A, no. 12, pp. 3229–3237, 2009.

[16] ——, "Scan-based attack against elliptic curve cryptosystems," in *Proc. IEEE Asia South Pacific Design Auto Conference*, pp. 407–412, 2010.

[17] D. Hely, F. Bancel, M. L. Flottes, and B. Rouzeyre, "Test control for secure scan designs," in *Proc. European Test Symposium*, pp. 190–195, 2005.

[18] M. Gomułkiewicz, W. Nikodem, and T. Tomczak, "Low-cost and universal secure scan: a design-architecture for crypto chips," in *Proc. International Conference on Dependability of Computer Systems*, pp. 282–288, 2006.

[19] D. Hely, F. Bancel, M. L. Flottes, and B. Rouzeyre, "Secure scan techniques: a comparison," in *Proc. of 12th IEEE International On-Line Testing Symposium*, pp. 119–124, 2006.

[20] J. Lee, M. Tehranipoor, and J. Plusquellic, "A low-cost solution for protecting IPs against scan-based side-channel attacks," in *Proc. 24th IEEE VLSI Test Symposium*, pp. 94–99, 2006.

[21] D. Hely, F. Bancel, M. L. Flottes, and B. Rouzeyre, "Securing scan control in crypto chips," *J. Electron Test*, pp. 457–464, 2007.

[22] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, "Securing designs against scan-based side-channel attacks," pp. 325–336, 2007.

[23] S. Paul, R. S. Chakraborty, and S. Bhunia, "VIm-scan: a low overhead scan design approach for protection of secret key in scan-based secure chips," in *Proc. the 25th IEEE VLSI Test Symmposium*, pp. 455–460, 2007.

[24] U. Chandran and D. Zhao, "SS-KTC: a high-testability low-overhead scan architecture with multi-level security intergration," in *Proc. 27th IEEE VLSI Test Symposium*, pp. 321–326, 2009.

[25] D. Mukhopadhyay, S. Banerjee, D. RoyChowdhury, and B. B. Bhattacharya, "CryptoScan: a secured scan chain architecture," in *Proc. 14th Asian Test Symposium*, pp. 348–358, 2005.

[26] G. Sengar, D. Mukhopadhyay, and D. RoyChowdhury, "An efficient approach to develop secure scan tree for crypto-hardware," in *Proc. 15th International Conference of Advanced Computing and Communications*, pp. 21–26, 2007.

[27] Y. Shi, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "A secure test technique for pipelined advanced encryption standard," *IEICE Trans. Information and Systems*, vol. E91–D, no. 3, pp. 776–780, 2008.

[28] M. Doulcier, M. L. Flottes, and B. Rouzeyre, "AES-based BIST: Self-test, test pattern generation and signature analysis," in *Proc. 25th IEEE VLSI Test Symposium*, pp. 94–99, 2007.

[29] T. Beth and D. Gollmann, "Algorithm engineering for public key algorithms," *IEEE J. Sel. Areas Commun.*, vol. 7, pp. 458–465, 1989.

[30] L. Song and K. K. Parhi, "Low energy digit-serial/parallel finite field multipliers," *Journal of VLSI Signal Processing*, vol. 19, no. 2, pp. 149–166, 1998.

[31] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 449–460, 2003.

[32] S. Kumar, T. Wollinger, and C. Paar, "Optimum digit serial GF(2^m) multipliers for curve-based cryptography," *IEEE Trans. Comput.*, vol. 55, pp. 1306–1311, 2006.

[33] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "Multicore curve-based cryptoprocessor with reconfigurable modular arithmetic logic units over GF(2^n)," *IEEE Trans. Comput.*, vol. 56, no. 9, pp. 1269–1282, 2007.

[34] G. Orlando and C. Paar, "A high-performance reconfigurable elliptic curve processor for GF(2^m)," in *Proc. Workshop on Cryptographic Hardware and Embedded Systems*, ser. Lecture Notes in Computer Science, vol. 1965, pp. 41–56, 2000.

[35] N. Gura, S. C. Shantz, H. Eberle, S. Gupta, V. Gupta, D. Finchelstein, E. Goupy, and D. Stebila, "An end-to-end systems approach to elliptic curve cryptography," in *Proc. Workshop on Cryptographic Hardware and Embedded Systems*, ser. Lecture Notes in Computer Science, vol. 2523, pp. 349–365, 2002.

[36] H. Eberle, N. Gura, and S. Chang-Shantz, "A cryptographic processor for arbitrary elliptic curves over GF(2^m)," pp. 444–454, 2003.

[37] A. E. Cohen and K. K. Parhi, "Implementation of scalable elliptic curve cryptosystem crypto-accelerators for GF(2^m)," pp. 471–477, 2004.

[38] S. Moon, "A 193-bit encryption processor for elliptic curve cryptosystem using fast VLSI algorithms in finite fields," pp. 611–613, 2005.

[39] W. N. Chelton and M. Benaissa, "Fast elliptic curve cryptography on FPGA," *IEEE Trans. VLSI Syst.*, vol. 16, no. 2, pp. 198–205, 2008.

[40] P. L. Montgomery, "Speeding the pollard and elliptic curve methods for factorizations," *Mathematics of Computation*, vol. 48, pp. 243–264, 1987.

[41] C. J. S, "Resistance against differential power analysis for elliptic curve cryptosystems," in *Proc. Workshop on Cryptographic Hardware and Embedded Systems*, ser. Lecture Notes in Computer Science, vol. 1717, pp. 292–302, 1999.

[42] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, "An implementation of elliptic curve cryptosystems over $\mathbb{F}_{2^{155}}$," *IEEE J. Sel. Areas Commun.*, vol. 11, no. 5, pp. 804–813, 1993.

[43] J. López and R. Dahab, "Fast multiplication on elliptic curves over GF(2^m) without precomputation," in *Proc. Workshop on Cryptographic Hardware and Embedded Systems*, ser. Lecture Notes in Computer Science, vol. 1717, pp. 316–327, 1999.

[44] K. Okeya, H. Kurumatani, and K. Sakurai, "Elliptic curve with the montgomery form and their cryptographic applications," in *Proc. the International Conference on Theory and Practice of Public-Key Cryptography*, ser. Lecture Notes in Computer Science, vol. 1751, pp. 238–257, 2000.

[45] K. Okeya and K. Sakurai, "Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a montgomery-form elliptic curve," in *Proc. Workshop on Cryptographic Hardware and Embedded Systems*, ser. Lecture Notes in Computer Science, vol. 2162, pp. 126–124, 2001.

[46] L. Goubin, "A refined power-analysis attack on elliptic curve cryptosystems," in *Proc. the International Conference on Theory and Practice of Public-Key Cryptography*, ser. Lecture Notes in Computer Science, vol. 2567, pp. 199–211, 2003.

[47] "Digital signature standard (DSS)," Federal Information Processing Standards Publication 186-2 (FIPS 186-2), National institute of standards and technology (NIST), Tech. Rep., 2000.

[48] O. Kömmerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," in *Proc. the USENIX Workshop on Smartcard Technology (Smartcard '99)*, 1999.

[49] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "The 128-bit block cipher camellia," *IEICE Trans. Fundamentals*, vol. E85–A, no. 1, pp. 11–24, 2001.

List of Publications

論文（学術誌原著論文）

1. ◯ **R. Nara**, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Scan vulnerability in elliptic curve cryptosystems", IPSJ Trans. SLDM, vol. 4, February Issue, pp. 47–59, Feb. 2011.

2. ◯ **R. Nara**, K. Satoh, M. Yanagisawa, T. Ohtsuki, and N. Togawa, "Scan-based side-channel attack against RSA cryptosystems using scan signatures," IEICE Trans. Fundamentals, vol. E93–A , no. 12, pp. 2481–2489, Dec. 2010.

3. K. Tanimura, **R. Nara**, S. Kohara, Y. Shi, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Unified dual-radix architecture for scalable montgomery multiplications in GF(P) and GF(2^n)," IEICE Trans. Fundamentals, vol. E92–A, no. 9, pp. 2304–2317, Sep. 2009.

4. ◯ **R. Nara**, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "A scan-based attack based on discriminators for AES cryptosystems," IEICE Trans. Fundamentals, vol. E92–A, no. 12, pp. 3229–3237, Dec. 2009.

論文（国際会議）

1. ◯ **R. Nara**, H. Atobe, Y. Shi, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "State-dependent changeable scan architecture against scan-based side channel attacks," in *Proc. IEEE ISCAS 2010,* pp. 1867–1870, May 2010.

2. ◯ **R. Nara**, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Scan-based attack against elliptic curve cryptosystems," in *Proc. IEEE ASP-DAC 2010*, pp. 407–412, Jan. 2010.

3. ◯ **R. Nara**, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Scan-based attack against elliptic curve cryptosystems," in *Proc. IEEE ASP-DAC 2010, Poster session*, Jan. 2010.

4. K. Tanimura, **R. Nara**, S. Kohara, K Shimizu, Y. Shi, N. Togawa, M. Yanagisawa, T. Ohtsuki, "Scalable unified dual-radix architecture for montgomery multiplication in GF(P) and GF(2^n)," in *IEEE ASP-DAC 2008*, pp. 697–702, Jan. 2008.

5. ◯ **R. Nara**, K. Shimizu, S. Kohara, N. Togawa, M. Yanagisawa, T. Ohtsuki, "An area-efficient GF(2^m) MSD multiplier based on an MSB multiplier for elliptic curve LSI," in *Proc. IEICE ITC-CSCC 2007,* vol. 1, pp. 37–38, Jul. 2007.

国際会議（査読付）

1. ◯ **奈良竜太**, 佐藤　圭, 戸川　望, 柳澤政生, 大附辰夫, "RSA 暗号に対するスキャンベース攻撃, " 信学 回路とシステム軽井沢ワークショップ, pp. 197–202, Apr. 2010.

2. ◯ **奈良竜太**, 戸川　望, 柳澤政生, 大附辰夫, "楕円曲線暗号に対するスキャンベース攻撃," 情処学 DA シンポジウム 2009, vol. 2009, no. 7, pp. 109–114, Aug. 2009.

3. ◯ **奈良竜太**, 小原俊逸, 清水一範, 戸川　望, 柳澤政生, 大附辰夫, "GF(2^m) 上の MSB 乗算器をベースにした楕円曲線暗号 LSI 向け MSD 乗算器の実装," 信学 回路とシステム軽井沢ワークショップ, pp. 355–360, Apr. 2007.

4. ○ **奈良竜太**，清水一範，小原俊逸，戸川　望，柳澤政生，大附辰夫，“GF(2^m)上のSIMD型MSD乗算器を用いた楕円曲線暗号回路の実装,” 情処学 DA シンポジウム 2007，vol. 2007，no. 7，pp. 221–226，Aug. 2007.

5. ○ **奈良竜太**，清水一範，小原俊逸，戸川　望，柳澤政生，大附辰夫，“楕円曲線暗号用SIMD型MSD乗算器の設計，” 情処学 組込みシステムシンポジウム 2007，vol. 2007，pp. 90–99，Oct. 2007.

研究会

1. ○ **奈良竜太**，小寺博和，戸川　望，柳澤政生，大附辰夫，“SASEBO-GIIを使用したAESに対するスキャンベース攻撃の実装実験，” 信学 暗号と情報セキュリティシンポジウム (SCIS2011)，1D1-2，Jan. 2011.

2. ○ **奈良竜太**，戸川　望，柳澤政生，大附辰夫，“RSA暗号に対するスキャンベース攻撃の評価実験，” 信学 2010 ソサイエティ大会，A-3-6，p. 68，Sep. 2010.

3. ○ **奈良竜太**，柳澤政生，大附辰夫，戸川　望, “スキャンチェインの構造に依存しないAESスキャンベース攻撃,” 信学 暗号と情報セキュリティシンポジウム (SCIS2009)，3A4-3，p. 277，Jan. 2009.

4. 荒幡　明，**奈良竜太**，戸川　望，柳澤政生，大附辰夫, “歩行者向けデフォルメ地図生成のための並列処理ハードウェアエンジンの設計,” 信学 VLD 研究会，VLD2008-67，pp. 43–48，Nov. 2008.

5. ○ **奈良竜太**，戸川　望，柳澤政生，大附辰夫, “周辺回路を含むAES-LSIへのスキャンベース攻撃,” 信学 VLD 研究会，VLD2008-68，pp. 49–53，Nov. 2008.

6. 跡部浩士，**奈良竜太**，史　又華，戸川　望，柳澤政生，大附辰夫, “暗号回路における動的に構造変化するセキュアスキャンアーキテクチャ,” 信学 VLD 研究会，VLD2008-69，pp. 55–59，Nov. 2008.

7. 川畑伸幸，**奈良竜太**，戸川　望，柳澤政生，大附辰夫, “クロスバスイッチを用いたS-Box切替によるAES暗号処理回路のパワーマスキング手法,” 信学VLD研究会，VLD2008-70，pp. 61–66，Nov. 2008.

8. ◯ **奈良竜太**，清水一範，小原俊逸，戸川　望，柳澤政生，大附辰夫, “楕円曲線暗号に適したGF(2^m)上のSIMD型MSD乗算器の設計,” 信学，VLD研究会，VLD2007-11，pp. 25–29，May 2007.

9. 谷村和幸，**奈良竜太**，小原俊逸，史　又華，小原俊逸，戸川　望，柳澤政生，大附辰夫, “GF(2^n)及びGF(p)におけるスケーラブル双基数ユニファイド型モンゴメリ乗算器,” 信学，VLD研究会，VLD2007-42，pp. 43–45，Jun. 2007.

10. ◯ **奈良竜太**，小原俊逸，清水一範，戸川　望，池永　剛，柳澤政生，後藤敏，大附辰夫, “楕円曲線暗号向けGF(2^m)上のDigit-Serial乗算器の設計,” 信学 VLD研究会，VLD2006-89，pp. 25–30，Jan. 2007.

11. 内田純平，**奈良竜太**，宮岡祐一郎，戸川　望，柳澤政生，大附辰夫, “ワードベースモンゴメリ乗算器を搭載した高速楕円曲線暗号LSI,” 信学，VLD研究会，VLD2004-125，pp. 5–10，Mar. 2005.

招待講演

1. 2007年9月 IEEE SSCS Japan Chapter VDECデザイナーフォーラム2007（若手の会）Ph.D企画セッション パネリスト

業績賞等

1. 2011年3月 電気通信普及財団賞第26回テレコムシステム技術賞
2. 2011年2月 第23回回路とシステム軽井沢ワークショップ奨励賞
3. 2010年3月 2009年度大川功記念賞
4. 2009年8月 情報処理学会優秀発表学生賞

5. 2007 年 8 月 情報処理学会優秀発表学生賞

6. 2007 年 2007 年度大川功情報通信学術奨学金