Detecting Trapdoors in Smart Cards Using Timing and Power Analysis *

Jung Youp Lee[1], Seok Won Jung[2], and Jongin Lim[1]

[1] Graduate School of Information Security,
Korea University, Anam Dong, Sungbuk Gu, Seoul, Korea
[2] Department of Information Security,
Mokpo National University, ChonNam, Korea

Abstract. For economic reasons, in spite of security problems, the commands of re-initializing the card and writing patch code are widely used in smart cards. The current software tester has difficulty in detecting these trapdoor commands by reason that trapdoors are not published and programmed sophisticatedly. Up to now the effective way to detect them is to completely reveal and analyze the entire code of the COS with applications such as the ITSEC. It is, however, very time-consuming and expensive processes. We propose a new approach of detecting trapdoors in smart cards using timing and power analysis. By experiments, this paper shows that this approach is a more practical method than the current methods.

Keywords: Smart Card, Trapdoor, Timing Analysis, Power Analysis

1 Background

The smart card has a high level of security, since it could safely store secret keys and execute cryptographic algorithms. In addition, smart cards are so small and easy to handle that they are replacing magnetic-stripe cards as bank cards and credit cards in electronic payment systems.

Since enormous amounts of money flow in a widely-distributed system, the service provider of an electronic payment system must have a high degree of confidence in the IC chip manufacturer, the producer of the Chip Operating System (COS) with applications, and the smart card issuer. The service provider must be able to be certain that the software in the COS performs the required financial transactions without any errors and that the software is free of security leaks, not to mention trapdoors deliberately introduced into the software.

Evaluating and testing smart cards could provide the service provider with confidence. Evaluations are generally applied to the description documents or

* This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment).

the program code with static procedures. In contrast to static methods, tests are applied to the real smart card in operation with dynamic procedures.

The Trusted Computer System Evaluation Criteria (TCSEC) [1], the Information Technique System Evaluation Criteria (ITSEC) [2], and the Common Criteria (CC) [3] are representative evaluation methods. The TCSEC were created in order to establish a catalog of criteria for evaluating the trustworthiness of software products by the American Department of Defense (DoD) in 1985. The ITSEC published in 1990 were European criteria based on TCSEC. The CC were made in order to provide a uniform standard for evaluating the correctness of a software in 1996. The CC have also been published as international standard ISO 15408. The basic procedure for evaluating a system is to rate the mechanisms that it uses to maintain security with regard to the pre-defined basic threats. For example, the ITSEC have six quality levels from E1 to E6. If the software satisfies minimum requirements such as informal descriptions of functions, it has the lowest level E1. For the highest level E6, a complete evaluation is required. For instance, full source code and object code testing are necessary for level E6.

As well as the software, the hardware could be evaluated by various criteria. The VISA corporation requires the Chip Hardware Architecture Review for their smart cards [4]. This is an evaluation of the basic chip, without a COS or an application. This evaluation identifies features that the COS and applications must enable properly in order to achieve the security desired.

To test the software of a smart card, the service provider or its agency examines the input and output data with regard to their relationship to each other, as defined in the specifications. If the examiner knows the internal data structures and processes of the COS with applications, the number of possible input values could be reduced. Besides the functional tests of the COS with applications, the VISA corporation performs risk testing [4]. This testing verifies that the security features provided by the IC chip are appropriately implemented by the COS, and evaluates the protection that a card provides against various documented and well-known attacks.

The hardware of a smart card is tested by the IC chip manufacturer and by the card body manufacturer. The card body is verified by mechanical, chemical, and thermal test regulations of the ISO/IEC 10373. In every fabrication processes of a smart card, the ATR test and the EEPROM test are performed to check whether the IC chip has been damaged by being packaged into the module or by being heated during the embedding process.

In the real smart card fields, the commands of re-initializing the card and writing patch code are widely used. For economic reasons, the re-initializing command is intentionally inserted into the COS to reuse incorrectly issued cards which should be cut into pieces using a pair of scissors. This command clears the EEPROM of a smart card. Nowadays the patch command is almost always used at initial stage that a new COS with applications is introduced. This command is used to correct the errors of the COS or to adapt the minor changes of the specifications. This command writes patch code into the EEPROM. In many cases, these commands are not published. Such unpublished commands could be trapdoors. The malicious developer of the COS could easily introduce trapdoors in the form of patch code that reads all of the EEPROM memory including the key information. This is a serious risk, especially to the electronic payment system that uses only the symmetric key. The exposure of the key means a crash of the system.

By means of the software test of a smart card, the service provider could have the confidence that the smart card of its system complies with the specifications and that it operates without errors or security weakness including trapdoors. However, it is sometimes incorrectly assumed that these tests can discover all Trojan horses in the software. Although the unsophisticated trapdoors could be detected, an experienced programmer can easily create trapdoors that are not detected by current tests. Up to now the effective way to detect them is to completely reveal and analyze the entire code of the COS with applications. However, it is very time-consuming and expensive processes. The ITSEC level E4 which is the lowest level of the source code testing can cost around 300,000 euro. In order to be certified of the level E6, it takes several years and costs several million euros [5].

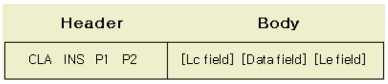
We propose a new approach of detecting trapdoors in smart cards using timing analysis and power analysis. The basic idea is as follows: if a terminal transmits the commands which are not in the specifications into a smart card, their response time and power consumption are same. However, the trapdoor command which is also not in the specifications has the different response time and the different power consumption as compared with the other commands. This idea provides a fast and inexpensive method for detecting trapdoors compared to known test methods.

The following section classifies the types of trapdoors in smart cards. Section 3 illustrates the current test methods to detect trapdoors. Section 4 introduces the timing analysis and the power analysis. Section 5 presents the methods to detect the defined trapdoors using timing and power analysis, and shows the practice.

2 Types of Trapdoors in Smart Cards

The smart card has a interface so-called Application Protocol Data Unit (APDU) that consists of a command APDU and a response APDU to communicate with the terminal [6]. This tells us that a trapdoor in a smart card also has a APDU format and that it could be detected by the analysis of the APDU format.

A command APDU, which is sent by the terminal to the card, consists of a mandatory header and an optional body in Figure 1. The header is composed

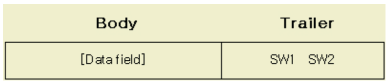


Header	Body
CLA INS P1 P2	[Lc field] [Data field] [Le field]

Fig. 1. Structure of a command APDU

of four elements: the class byte (CLA), the instruction byte (INS), and two parameter bytes (P1 and P2). The body is composed of the length of input data (Lc field), the input data (data field), and the expected length of output data (Le field).

A response APDU, which is sent by the card in reply to a command APDU, consists of an optional body and a mandatory trailer in Figure 2. The body is the output data and the trailer is a status word.



Body	Trailer
[Data field]	SW1 SW2

Fig. 2. Structure of a response APDU

Trapdoors could be inserted in a command APDU that is in the published documents such as the specifications or a manual. This command operates the defined function as well as the hidden function and it returns the response with the hidden information. Trapdoors could be a new command APDU that is not in the published documents. Also, a series of commands could be a trapdoor. If the commands of this trapdoor executes in order, the last command performs a hidden function. Otherwise, it performs a normal function. We classify these trapdoors into three types: steganographic commands, trapdoor commands, and trapdoor sequences. In the following subsections, we will define the types of trapdoors in detail.

2.1 Steganographic Commands

Definition 1. *A steganographic command is a command defined in the published documents and implemented in the smart card, which has one or more hidden functions than defined in the published documents.*

Almost all smart card operating systems contain the GET CHALLENGE command for generating and issuing random numbers [7]. This command could be modified to a steganographic command. If a smart card generates a 16-byte random number, The first 8-byte number of the random number is actually generated by the pseudo-random number generator. The remaining 8-byte number would then consist of an 8-byte value taken from the EEPROM and XORed with the first 8-byte random number. An external program could then be used to read out the entire memory contents, including all of the keys. Incidentally, this is a good example of a steganographic trapdoor.

2.2 Trapdoor Commands

Definition 2. *A trapdoor command is a command that is not defined in the published documents and is implemented in the smart card.*

To make a trapdoor in smart cards, the developer of the COS and applications usually defines a new command rather than modifies an existing command. The reason is that he could handle the trapdoor as a normal command.

A command APDU consists of CLA, INS, P1-P2, Lc, Data, and Le. Therefore a trapdoor command could use one or more elements of a command APDU as the trapdoor awareness data. According to that data, trapdoor commands could be defined by CLA trapdoor commands, INS trapdoor commands, P1-P2 trapdoor commands, and so on.

If an experienced programmer inserts a trapdoor command into the COS, he would use one or more commands before the trapdoor command to block easy detecting. The successful authentication command, for example, may be requested before the patch command is applied. Also he would intentionally use the error status word for the successful trapdoor command to pretend that the smart card does not support that command.

2.3 Trapdoor Sequences

Definition 3. *A trapdoor sequence is a sequence of commands defined in the published documents and implemented in the smart card, of which the last command operates as a trapdoor if the commands are executed in predefined order.*

Suppose a trapdoor sequence is a sequence of the GET RESPONSE command, the PUT DATA command, and the GET CHALLENGE command. The GET CHALLENGE command, which returns a random number in normal state, could respond to the key information if the above sequence of commands are sent into a smart card according to the defined order.

Trapdoor sequences are usually defined to avoid the collision with the possible command sequences of transactions in the specifications.

3 Current Test Methods to Detect Trapdoors

A functional test is commonly used to discover trapdoors in a real smart card in operation. Figure 3 illustrates the method used to determine the unpublished commands. A CLA in the APDU is sent into the smart card, being changed from '00' to 'FF'. As soon as a return code other than 'invalid class' is received, the first valid class byte has been determined. Then all possible INSs are sent with the determined CLA. The unsupported INS returns the status word 'unknown instruction', and the supported INS does return the other status word. In a similar manner, the possible parameters of a command could be determined. If suitable software is available in the terminal, this method can be used determine which commands are supported by a smart card in a few minutes.

The reason that this simple search algorithm for CLA, INS, and P1-P2 is possible is that practically most of command interpreters in smart card operating systems evaluate received commands by starting with the CLA byte and working through the following bytes.

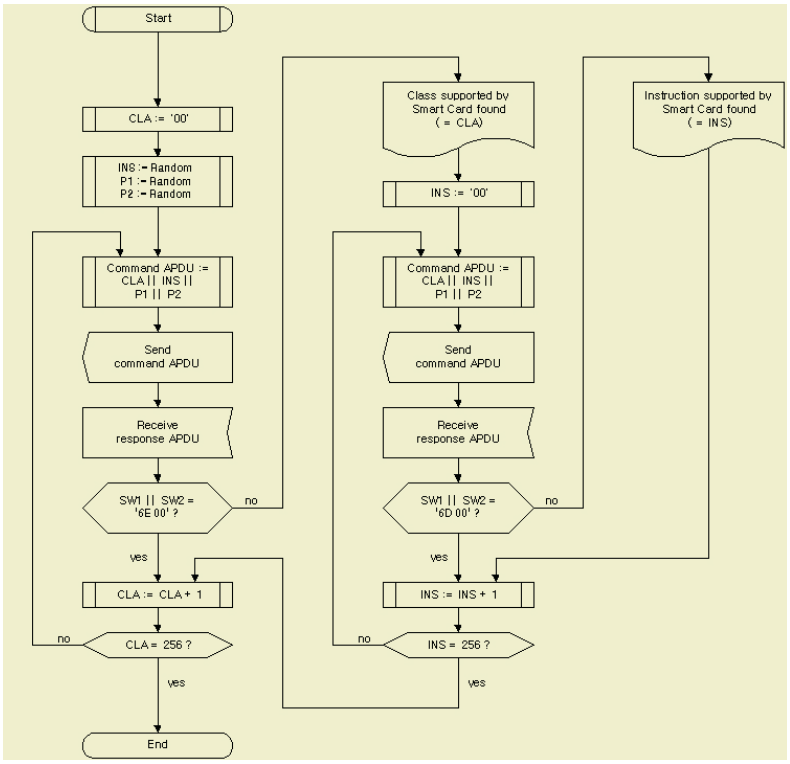


Fig. 3. Basic procedure for determining the commands set of a smart card

However, an experienced programmer can easily create trapdoors that are not detected by this test as mentioned in Subsection 2.2. If he uses the error status word 'unknown instruction' for the successful trapdoor command, there is no way to detect them by this test. What is worse, this test could not practically detect a steganographic command or a trapdoor sequence.

4 Timing Analysis and Power Analysis

In the previous section, we show that the current test methods are not sufficient to detect trapdoors in smart cards. It is well known fact that a timing analysis and a power analysis are powerful methods to attack the smart card. This paper explains how these methods could be applied to detect trapdoors effectively in the next section. We introduce the concept of a timing analysis and a power analysis in the following subsections.

We suggest that the side channel attacks which are focus to on the cryptographic algorithms in the smart card could be used for detecting trapdoors. It would be powerful methods as much for the cryptographic algorithms.

4.1 Timing Analysis

This subsection briefly describes the timing analysis of [10]. A timing attack can be mounted if the execution time of the cipher depends on the value of the key. For example, consider the square-and-multiply algorithm for modular exponentiation, which is the basis of many public-key cryptosystems. If no special precautions are taken, the total execution time of the cipher will vary depending on the key. Hence, it is possible to deduce the key by comparing the cipher execution times for different keys.

4.2 Power Analysis

This subsection briefly describes the power analysis of [11] and how it can be used to attack encryption algorithms. Simple Power Analysis (SPA) involves directly interpreting the power consumption measurement of a device like a smart card. SPA can yield information about a device's operation as well as key material. For example, when an attacker can find out which branch of a jump instruction is taken in the DES operation, it becomes possible to use such information to draw conclusions about the secret key because a conditional branch is commonly used to compute the DES key scheduling.

Differential Power Analysis (DPA) is a statistical approach, where many traces are collected, and are examined for correlations. A partial guess of a key could determine whether the value of a particular bit in the outputs is 0 or 1. The value divides the traces into two sets. Then, the averages of the traces for the two sets are compared. If the guess is incorrect, there will be no correlation between the two sets. However if the guess is correct, the first set will have a different bias than the second one. When the averages of the two sets are subtracted, there will be a spike in the difference. In such manner, the entire key can be derived.

5 Detecting Trapdoors in a Smart Card

Most smart cards support the protocol T=0, half-duplex transmission of asynchronous characters [6]. In the protocol T=0, the COS has to parse the received INS in order to distinguish the commands that are incoming data transfers to the smart card and the commands that are outgoing data transfers to the terminal. Therefore, the COS commonly has the command processing steps as the following Algorithm 1.

Algorithm 1. APDU command processing steps

```
Step 1. Receive a command APDU header.
Step 2. Parse the command according to INS.
Step 3. Receive a command APDU body if it is the incoming data transfer
   command.
Step 4. Process the secure messaging.
```

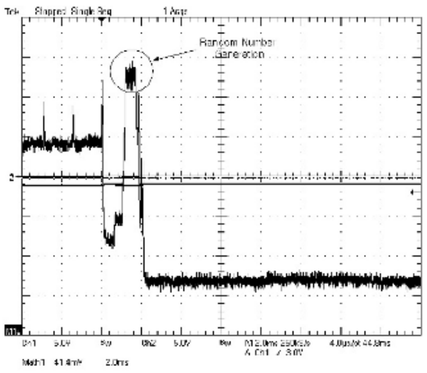


Fig. 4. Power consumption of a general GET CHALLENGE command

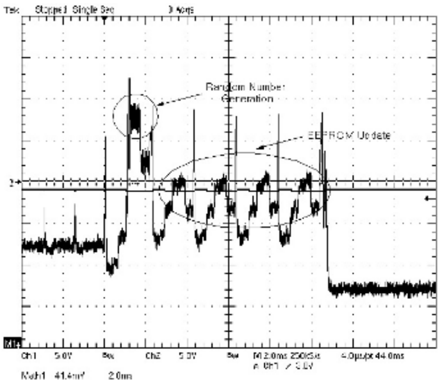


Fig. 5. Power consumption of a trapdoor GET CHALLENGE command

Step 5. Check the input command according to the specifications.
Step 6. Performs the command.
Step 7. Send a response APDU.

If the smart card supports only block transmission protocols such as T=1, type A, type B [6, 8, 9], the steps may differ from the above. For example, Step 3 could be merged to step 1. Step 2 and 4 could be swapped. Since most of smart cards supporting block transmission protocols are implemented T=0 also, without loss of generality, we assume that the smart cards have T=0 protocol.

For experiments, the sample COS is developed on the Samsung OPENice i500 smart card development tools. The power consumption graph is the difference of the voltages over 50 ohms between Vcc of the smart card and Vcc of the terminal measured by the Tektronix TDS 5052 Digital Phosphor Oscilloscope.

Detecting Steganographic Commands
This trapdoor executes an ordinary function and a trapdoor function at the same time. It should be implemented in Step 6 of Algorithm 1, and it has additional operations compared with an original command.

Let us use an example where a trapdoor GET CHALLENGE command generates a random number and operates XOR with one byte of the EEPROM memory. After that, it updates the EEPROM area for the next address to read.

Most smart card ICs have a hardware random number generator, so the power consumption graph of the GET CHALLENGE command is generally simple like Figure 4. On the other hand, because the trapdoor GET CHALLNGE command has additional operations, the power consumption graph differs compared with the general implementation of the GET CHALLENGE command if the same IC is used. Figure 5 tells us that it has additional operations that are not known, so we suspect this as a trapdoor.

Detecting Trapdoor Commands
This trapdoor has one or more new CLA, INS, P1-P2, Lc, Data, or Le. We illustrate a INS trapdoor command first, which is the representative trapdoor. In Step 2, the INS of an input command APDU is compared within the switch clause or the if-else clause. If it is matched, the COS calls the function and

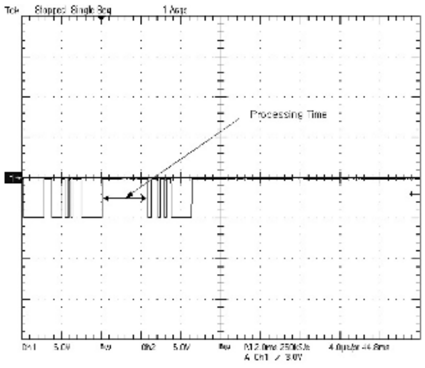


Fig. 6. Processing time of commands that are not in a command parser

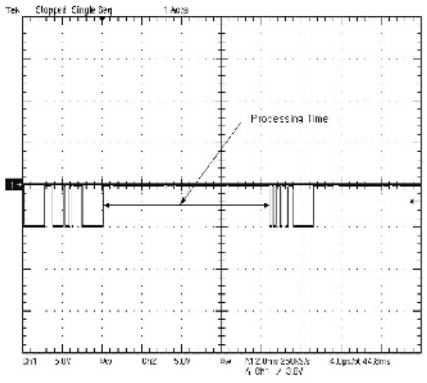


Fig. 7. Processing time of an INS trapdoor command

returns an ordinary status word. Otherwise, the COS returns the status word '6D00' (unknown instruction).

The INS trapdoor command can be implemented in Step 2 of Algorithm 1. Although this trapdoor returns the status word '6D00' as Example 1, we could easily guess that this command of which the processing time is different from those of other commands would be a trapdoor. In a similar manner like Figure 3, INSs from '00' to 'FF' are sent into the smart card. Figure 6 shows that the processing time of INSs that are not in the command parser and Figure 7 shows an INS trapdoor command.

Because the INS value is between 0 and 255, the full search time of INS trapdoor commands is only about 8 seconds assuming that the processing time of one command is 30ms as [5].

Example 1. Pseudo-code for command parser and trapdoor function

```
command_parser ()
{
    switch (apdu.ins)
    {
        case TRAPDOOR:
            trapdoor();
        break;
        case READ_BINARY:
            read_binary();
        break;
        case WRITE_BINARY:
            if (apdu.lc > 0)
                receive_arr(apdu.body, apdu.lc);
            write_binary();
        break;
        // *** Abbr. ***//
        case GET_CHALLENGE:
            get_challenge(&tApdu);
        break;
        // *** Abbr. ***//
        default:
            send_sw(0x6D00);
        return;
    }
}

trapdoor()
{
    format_file_system();
    send_sw(0x6D00);
}
```

Although an experienced programmer can enforce a delay that the command parser matches the processing time with a trapdoor command. Since instructions are different, the power consumption of a trapdoor command would be different from those of other commands. Figure 8 shows the power consumption of commands which return the status word '6D00' with a enforced delay, and Figure 9 shows the power consumption of a trapdoor.

In Step 5 of Algorithm 1, P1 and P2 are checked whether they comply with the specifications or not, respectively. Generally P1 is checked first and P2 second. For example, that the P1-P2 trapdoor command operates as a trapdoor only when P1 = 0x37 and P2 = 0xBF. Matching P1 causes a small difference of the processing time because matching P1 enables P2 to be compared. To search the matching P1, the same method for detecting the INS trapdoor command is applied. After the full search of P1, P2 could be searched by the same way.

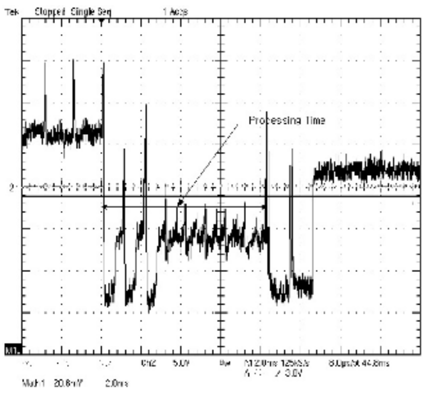


Fig. 8. Power consumption of commands which return the status word '6D00' with a enforced delay

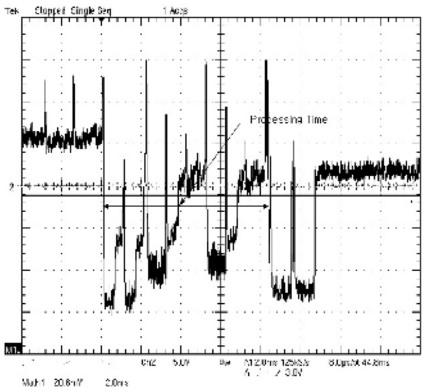


Fig. 9. Power consumption of a trapdoor command

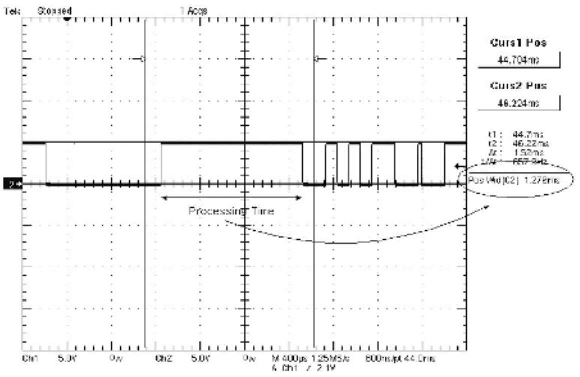


Fig. 10. Processing time of a P1-P2 trapdoor command when P1 != 0x37

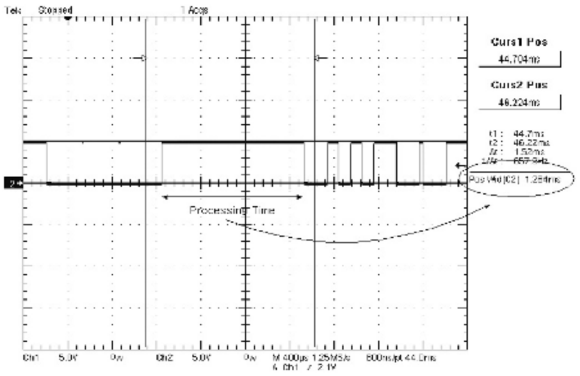


Fig. 11. Processing time of a P1-P2 trapdoor command when P1 = 0x37

Figures 10 and 11 show the processing time of the P1-P2 trapdoor command with matching P1 or not.

Detecting other trapdoor commands are the extension of detecting INS trapdoor commands or P1-P2 trapdoor commands.

Detecting Trapdoor Sequences

To implement a trapdoor sequence, the commands in a sequence have to save the current state for the next command to execute. Suppose that a trapdoor sequence is a sequence of the GET RESPONSE command, the PUT DATA command, and the GET CHALLENGE command. The variable CUR_STATE in RAM of a smart card is for the current state. If the GET RESPONSE command is executed, the CUR_STATE will be set the GET RESPONSE state as defined by S1. If the PUT DATA command is executed and the CUR_STATE is S1, the CUR_STATE will be set the PUT DATA state as defined by S2. If the GET CHALLENGE command is executed and the CUR_STATE is S2, a trapdoor

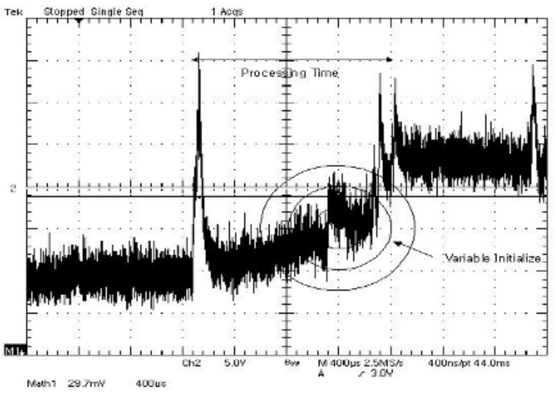


Fig. 12. Power consumption of the PUT DATA command if the previous command is not the GET RESPONSE command

will be executed and the CUR_STATE will be initialized to S0. Any other cases make the CUR_STATE to S0.

With respect to the PUT DATA command, the CUR_STATE will be set the next state if the previous command is the GET RESPONSE command, or it will be initialized. This means that the power consumption difference exists. Figure 12 and 13 show the difference of the power consumption.

The number of commands in most electronic payment systems is 30, more or less. The number of states is about 10 in one command. In other words, one command may return one of about 10 status words. To find one chain, a minimum of two command executions is needed. The full search time of a trapdoor sequence is estimated only about 90 minutes assuming that the processing time of one command is 30ms.

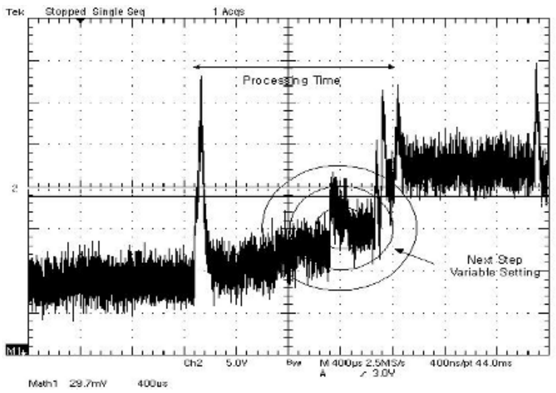


Fig. 13. Power consumption of the PUT DATA command if the previous command is the GET RESPONSE command

6 Conclusion

In the real field, even though the COS developers often insert trapdoors for economic reasons, the current analyzing methods for source code of the COS or the functional test software for smart cards could not detect trapdoors efficiently.

In this paper, trapdoors are classified into steganographic commands, trapdoor commands, and trapdoor sequences, based on the idea that the number of the smart card trapdoors are finite apart from general software because they should follow the APDU format. They are implemented with various ways in the special area of source code according to the basic COS structure.

Side channel attacks which are used to expose secret values of the crypto algorithms in the smart card could be applied to detect trapdoors in the smart card. The timing analysis is useful to detect trapdoor commands which are representative trapdoors in the real smart cards. The power analysis could be used to detect steganographic commands and trapdoor sequences which are more difficult to detect than trapdoor commands. We explain that this idea is very useful to detect trapdoors compared with the current methods.

Furthermore, they also provide the practical and inexpensive methods in the real smart card world. We want this paper to be the start of detecting trapdoors for real smart cards using a timing analysis and a power analysis.

References

1. Trusted Computer Systems Evaluation Criteria, US DoD 5200.28-STD, Dec. 1985.
2. Information Technology Security Evaluation Criteria, Version 1.2, Office for Official Publications of the European Communities, June 1991.
3. Common Criteria for Information Technology Security Criteria, Version 2.1, Aug. 1999.
4. VISA Corporation, Chip Card: Testing and Approval Requirements Version 7.0, Industry Services, Dec. 2002.
5. W. Rankl and W. Effing, "Smart Card Handbook," Third Edition, John Wiley & Sons, Ltd, 2003, pp.244, pp.544-546, pp.579, pp.589.
6. ISO/IEC 7816-3:1997, Identification cards - Integrated circuit(s) cards with contacts - Part 3: Electronic signals and transmission protocols.
7. ISO/IEC 7816-4:1995, Identification cards - Integrated circuit(s) cards with contacts - Part 4: Interindustry commands for interchange.
8. ISO/IEC 14443-3:2001, Identification cards. Contactless integrated circuit(s) cards. Proximity cards. Part 3: Initialization and anticollision.
9. ISO/IEC 14443-4:2001, Identification cards. Contactless integrated circuit(s) cards. Proximity cards. Part 4: Transmission protocol.
10. P. Kocher, "Timing Attacks on Implementation of Diffie-Hellman, RSA, DSS, and Other Systems," CRYPTO 1996, LNCS 1109, Springer-Verlag, 1996, pp.104-113.
11. P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," CRYPTO 1999, LNCS 1666, Springer-Verlag, 1999, pp.388-397.