Countermeasures in CPU for Timing and Power Side Channel Attack

Xue-song YAN\textsuperscript{1,2,*}, Xi-ge ZHANG\textsuperscript{1,2}, Hai-feng ZHANG\textsuperscript{1,2} and Liang LIU\textsuperscript{1,2}

\textsuperscript{1}State Grid Key Laboratory of Power Industrial Chip Design and Analysis Technology, Beijing Smart-Chip Microelectronics Technology Co., Ltd. Beijing 100192, China

\textsuperscript{2}Beijing Engineering Research Center of High-reliability IC with Power Industrial Grade, Beijing Smart-Chip Microelectronics Technology Co., Ltd. Beijing 100192, China

*Corresponding author

Keywords: Side channel attack, Security CPU.

Abstract. Security chip is the critical component for information security system. Cryptographic modules for various algorithms and random number generation are usually provided inside the security chip. They are essential elements for achieving the confidentiality, integrity, and availability of the entire system. However, sensitive data in security chips are often assigned to other components through CPU operations, and transfers, so CPU security is also an essential part of the security chip. In order to improve the anti-side channel attack capability of CPU, this paper puts forward several countermeasures, including: unified instruction execution time, random switch of idle component, random switch of data bus, and randomization of program execution time. Some of these countermeasures are anti timing attack, some of which are anti power attack. By adopting multiple protection measures in CPU, it not only reduces the information leakage problem when sensitive information passes through the CPU, but also provides a powerful supplement to enhance the security of the other cryptographic components in the chip.

Introduction

Security chips often contain a number of security related modules, such as algorithmic accelerators, voltage/temperature/frequency detectors, random number generator, etc. The security hardware and related software work together to form a complete security countermeasure solution. The side channel attack is a kind of attack that the defense plan should focus on. At present, a lot of research have been made on the side channel attack and protection technology of various cryptographic algorithms (symmetric, asymmetric, Hash Algorithm, etc.), however, the research on CPU protection is still a relatively new topic, especially embedded CPU.

The CPU is the main control unit of the whole chip. When running the security-related programs, sensitive data are often used to perform operations in the CPU or to be moved between different addresses by the CPU. These processes generate a large amount of side channel information that, if not protected, can easily be used by the attacker for analysis.

In addition, a large number of protection schemes are focused on the special processing of cryptographic algorithms and the design of corresponding hardware accelerators \cite{1,2}. Whether these algorithms are secure enough, and whether they can achieve the expected protection effect, it is usually only after the actual chip is available that an evaluation can be carried out, which leads to significant time and cost risks for the actual application of the chip.

In order to solve the above problems, this paper proposes a protection scheme of security CPU, which provides several hardware protection mechanisms for side channel attacks. These mechanisms are configurable, easy to adjust by software, and can be used as an effective supplement to other protection schemes on the security chip.

Section 2 of this paper introduces the basic concept of the timing and power attack. Section 3 briefly recalls the history of CPU architecture evolution and introduces the CPU pipeline used in this paper. Section 4 describes the countermeasure adopted in the CPU for timing attack. Section 5 describes the countermeasure for power attack, and section 6 summarizes the whole paper.
Basic Concept of Timing and Power Side Channel Attack

A side-channel attack is any attack based on non-functional information gained during the work of an electrical system. Timing information, power consumption, electromagnetic leaks or even sound can be source of side-channel information, which can be exploited. The protection scheme in this paper is proposed for timing and power attacks, which are two of the most typical attack techniques.

Timing attack takes advantage of the characteristics of differences in execution time for different input variables when executing secure related operations [3]. The differences in execution time are induced partly by different code branches for different input value, and partly by the micro-architectures of CPU. The former requires programmer to strictly follow the secure programming manual, and the latter requires improvements at the CPU micro-architecture.

Power attack is another typical side channel attack technique. By recording and analyzing a lot of power curve of different data grouping, the key in the secure device is restored. Power attack takes advantage of the characteristic that the power consumption of a cryptographic device relies on the intermediate value of the cryptographic algorithm that the device performs [4]. The goal of the countermeasures is to eliminate this dependency. There are two basic strategies of countermeasures: concealment and concealment.

Basic Concept of CPU Pipeline

Since the invention of the computer in the 1940s, great progress has been made in the architecture evolution of the processor field, during which several far-reaching innovations have taken place. On the one hand, the processor continues to evolve from early single-cycle implementations to deep pipeline and superscalar architecture. On the other hand, professor David A Patterson invented the RISC instruction set in the 1980s, overcoming bottlenecks in the CISC processor architecture. In recent decades, with Moore's law reaching saturation, it has become more and more difficult to improve the performance by increasing the frequency of the processor, and multi-core architecture has become the mainstream of high-performance processor. Up to today, the processor has become the backbone technology of the electronic industry, with extremely rich types facing very different application fields such as personal computer, wearable devices, IoT, cell phone and high performance computing.

The security CPU discussed in this paper is a three-stage pipeline, embedded CPU with RISC instruction set. It is mainly used in smart card, IoT and industrial control. The instruction set for this CPU contains about 150 instructions, including arithmetic logic instructions, data transfer instructions, branch instructions, privileged instructions, etc. Fig. 1 is the structure diagram for this CPU:

![Figure 1. Three stage pipeline block diagram.](image-url)
This CPU contains a three-stage pipeline, which is the fetch stage (FA), the execution stage (EX) and the commit stage (CA). At the fetch stage, the instruction address is generated and the instruction is fetched from the memory. The execution stage is responsible for decoding the instruction and doing the corresponding operation according to the instruction type. The commit stage is responsible for writing the results back to the register file and updating the processor state. Ideally, three instructions per clock cycle are in different stages of the pipeline, and the instruction stream will flow through the pipeline endlessly. However, due to the inherent characteristics of the pipeline, in some cases there will be a stall in the pipeline, the cause of which can be attributed to data hazard, control hazard and structural hazard[5] . The pipeline stall caused by branch instruction is a typical control hazard.

Countermeasure for Timing Attack

At present, a lot of research on CPU timing attack is aimed at cache of CPU [6, 7]. Since there is no cache component in our CPU, we focus on the micro-architecture of instruction execution time.

For RISC CPU, just like we used in this paper, the cycle consumption of the instruction is usually constant, but the execution time may vary due to the dependency with preceding instructions.

In the case of the branch instruction of the CPU, the execution time may vary for not taken and taken. The taken/not-taken decision cannot be made until the end of EX stage, but the FA stage is fetching new instruction before the decision, which inducing the control hazard between FA and EX stages. When a branch instruction intends to jump (taken), which is determined at the EX stage, the target instruction will be fetched to the FA stage at next clock cycle, and the instruction in current FA stage is useless. So the FA stage produces a bubble, the bubble then travels to the EX and CA stages, consuming an extra clock cycle in the branching command where the jump occurs. When the branch instruction does not jump, the command is executed in sequence, the next instruction has been fetched to the FA stage, as long as it continues to run, the pipeline does not produce bubbles, and the branching command consumes only one clock cycle.

Fig. 2 below shows the pipeline flow when branch is not taken:

![Figure 2. Pipeline flow when branch is not taken.](image)

Fig. 3 below shows the pipeline flow when branch is taken, which consumes an extra cycle:

![Figure 3. Pipeline flow when branch is taken.](image)

Based on the above analysis, whether the branch instruction jumps determines the number of clocks that the instruction consumes. The variables that determine whether to jump come from the environment input of the program. For example, if the program compares each character of the key, and the comparison result is used as a branching condition, so that by analyzing the execution time of
the branch instructions you can deduce whether each comparison is passed, and retrieve each character of the key.

In response to this characteristic of branch instructions, we add the pipeline blocking mechanism, which force the branch instruction blocks a clock cycle even it is not jumping. So that the branch instruction consumes the same cycles whether taken or not taken. The CPU simulation confirms this behavior, as illustrated in Fig. 4 below:

![Figure 4. Unified timing characteristic for branch instruction.](image)

Time window 1 and time window 2 are the same branch instructions with taken and not taken respectively. If not taken, the branch instruction consumes one cycle, but in the next cycle, the pipeline is blocked an extra cycle. So, whether taken or not, the branch instruction and the next instruction consume three clock cycles.

In addition to the branch instructions, we also fix the timing characteristics of other instructions with similar method.

### Countermeasure for Power Attack

Based on the analysis of sections 2 and 3, the objectives of the countermeasures for power attacks are to hide or mask the power characteristics of the security chip. We employ several different countermeasures to achieve this purpose, which can be used separately, or in combination.

#### Randomization of Power Consumption

It is obvious that each instruction does not use all the hardware resources in the CPU during execution. For example, data moving instructions don’t usually use ALU, and addition instructions don’t use multiplier. So, the pipeline knows which hardware resources are idle according to instruction decoding. Obviously the power consumption of the current instruction can be disturbed by making these idle hardware resources to switch randomly.

And further, the data used by instruction also has a lot of side channel information leakage when it is moved through CPU. Data path scrambling is used to randomize the power information related to data pattern.

Power randomization increases the overall power consumption of the CPU and should therefore be used only when running security-related programs. At the same time, the frequency of random operation injection and the components used for the random operation can be configured on demand by software.

Fig. 5 below shows a piece of dynamic power curve before enabling the power randomization. Fig. 6 is the same piece of program while the power randomization is enabled.

![Figure 5. A Piece of dynamic power curve before enabling power randomization.](image)
It can be seen that the power consumption is significantly randomized in Fig. 6 compared with Fig. 5.

**Randomization of Program Execution Time**

The randomization of program execution time refers to the random insertion of nop (none of operation) instructions in the program, which makes the whole execution time of the program random. It leads to the program execution time and the power consumption become difficult to analyze. Literature [8] adopts the similar idea, while different approach is proposed in the paper. The nop instruction inserted in this scheme only adds a very small amount of hardware area to the instruction decoding unit, the result of the instruction is discarded directly, so the execution of the nop instruction does not affect the state of the CPU. At the same time, in order to further disturb the power consumption during executing nop instructions, the power randomization measures proposed in section 5.1 are used to make the power consumption of nop instruction completely independent of the instruction type.

Of course, random insertion of nop instructions increases the execution time, which results in performance degradation, so it is only necessary when running security-related programs. Further, the frequency of nop instructions to be inserted can be configured by software on demand.

Fig. 7 is a simulation result of the program that adds random nop instructions. The rectangular-marked clock cycle is executing nop instructions, and the execution of a nop instruction does not result in a wrongly increase in the Program Counter. The program is still running in the same order.

**Summary**

At present, most of the secure CPU research on the anti-side channel attack technique is only for one type of the attack. This paper proposes a hybrid protection scheme, which resists both timing attack and attack. The countermeasures should be used in combination to avoid the problem of introducing new power or timing characteristics to a particular countermeasure.

The countermeasures proposed in this scheme can be configured flexibly according to the need of secure plan, and can be used in conjunction with other algorithmic hardware accelerators to increase security, or can be closed at any time when running non-security-related programs.
Acknowledgement

This research was financially supported by the science and technology project of State Grid Corporation of China. The project name is, Key Technology and Application Research for Security MCU in Smart Endpoint Device (546816180009).

References

[1] B. Gerard, V. Grosso, M. Naya-Plasencia, F. Standaert. Block Ciphers That Are Easier to Mask: How Far Can We Go? J. CHES 2013, Springer Berlin Heidelberg, 8086(2013)383-399.

[2] Messerges T S, Dabbish E A, Sloan R H. Power Analysis Attacks of Modular Exponentiation in Smartcards, J. Cryptographic Hardware & Embedded Systems, 1717(1999) 144-157.

[3] Paul Kocher, Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems, C. International Cryptology Conference on Advances in Cryptology, Springer-Verlag, (1996)104-113.

[4] Paul Kocher, Joshua Jaffe, Benjamin Jun, Differential Power Analysis, J. Proc Crypto, 1666(1999)388-397.

[5] David A Patterson, John L Hennessy. Computer Organization and Design: The Hardware/Software Interface, fifth edition, Chinese edition, China Machine Press, Beijing, 2015.

[6] Percival Colin, Cache missing for fun and profit, J. Proc of Bsdcan, 2005

[7] Eran Tromer, Dag Arne Osvik, Adi Shamir, Efficient Cache Attacks on AES, and Countermeasures, J. Journal of Cryptology, 23(1)(2010)37-71.

[8] He Zhangqing, Ao Tianyong, Liu Kai, Dai Kui. Hardware countermeasure against side-channel attacks based on randomized instruction injection, J. Huazhong Univ. of Sci. & Tech. (Natural Science Edition), 5(2014)128-132.