A Processor Architecture Protecting Secret Data from Hostile Software

Mori, Tatsuya
Department of Computer Science and Communication Engineering Graduate School of Information Science and Electrical Engineering Kyushu University

Yasuura, Hiroto
Department of Computer Science and Communication Engineering Graduate School of Information Science and Electrical Engineering Kyushu University

Inoue, Koji
PRESTO, Japan Science and Technology Agency | Department of Computer Science and Communication Engineering Graduate School of Information Science and Electrical Engineering Kyushu University

A Processor Architecture Protecting Secret Data from Hostile Software

Tatsuya MORI†, Hiroto YASUURA†, Koji INOUE†‡

†Department of Computer Science and Communication Engineering
Graduate School of Information Science and Electrical Engineering
Kyushu University
6-1 Kasuga-Koen, Kasuga-City, Fukuoka 816-8580, Japan

‡PRESTO, Japan Science and Technology Agency
4-1-8 Honcho Kawaguchi, Saitama 332-0012 Japan

E-mail: {t-mori,yasuura,inoue}@c.cscs.kyushu-u.ac.jp

Abstract

Recently, security on programs and data is strongly required in multitask open computer systems such as personal computers, mobile phones and IC cards. Several kinds of secure microprocessor architectures have been proposed to offer solutions to the problem. But these architectures are much inclined to general software protection. We focus on only secret data protection including its control code in a program. In this paper, we propose a processor architecture, which enables programmer to use special instruction set for control cryptography data and protect secret data from hostile software. It is called Cryptographic DataPath Processor(CDPP). This architecture lets not memorize secret data while decrypted even if in internal registers. Secret data is decrypted just in front of an ALU on datapath and encrypted just behind it. Programmers can control the cryptograph by special instructions activating symmetric cipher circuits. CDPP enables secure data control implementation by programmers under hostile software environments.

1 Introduction

As the semiconductor technology advances remarkably, information devices that installs LSI with various functions is widespread. With the enhancement of the function, the information terminal treating the confidential information that doesn't want to be known to others increases rapidly. When programmer mount two or more applications on one terminal, sharing processor by some application programs installed in terminal is general technique for giving generality to mounting terminal. As a typical example, in IC card system that provides the personal authentication and the digital cash service, several services exists together in one card, and the mechanism that the confidential information on individual service is not exposed to other services is needed.

However, the problem that the confidential information on the program and data is violated has happened on the multitasking OS of the computer in recent years. A variety of secure processor architectures are proposed to solve this problem, but these architectures tends to protect the entire software excessively[1][2][3][4]. These approaches require extra processing for security and additional hardware and extra energy are needed. Another problem is ability of anti-tampering. If the decrypted secret codes or data stored some memory devices in the system, there is a possibility to be stolen by illegal methods. We appropriate the focus to the confidential information alone that doesn't want to be exposed truly as a protection target.

In this paper, we assume a special instruction which enable programmer to control cryptograph, and propose the processor architecture that defends the confidential information from other software. In this architecture, even if it is an internal register of the processor, it is prohibited to memorize it with the confidential information decrypted. The secret data is decrypted immediately before Arithmetic and Logic Unit(ALU) on the data passing, and encrypted immediately after ALU. The programmer can control this cryptography processing by special instructions that activates the symmetric-key encryption circuit. CDPP enables programmer to mount the operation of secret data in the software environment under the multitasking OS.

Originally, the confidential information that wants to be protected is a part of program, and it is a programmer that knows which information should be protected. We think that programmer should be able to specify the secret data, and when it is unnecessary of protection, it is thought that the processor should be able to execute the instruction at high speed. The feature of CDPP is to be able to achieve the data protection management by programmer. In CDPP, the throughput of the instruction execution by the cryptography processing is decreased according to the programming, programmer comes to be able to handle the

trade-off between execution speed of program and stubbornness of secret data protection.

The remainder of this paper is organized as follows. Section 2 arranges the demand condition based on the analysis of the existing multitask system. Section 3 explains details of the proposal architecture and the security model. Section 4 consider CDPP from both sides of strength and the execution performance of safety. Section 5 concludes this paper with summary.

2 Basic Assumptions and Condisions

A present multitask system is made aiming to multiplex the limited resource like the processor and the memory, etc. between two or more processes, and to share efficiently. One of our targets is a proposal of the secret management method that can coexist with a basic function of processor and OS to multiplex the resource efficiently. In this section, the system model whom the secret protection that we think about targets is defined, and demand condition of CDPP is arranged from restriction of the process management of OS and processor architecture of the multitask system to the model.

2.1 Assumption

We assume a computer system which is composed of hardware and software. hardware consists of a processor, a memory, input devices and output units. software consists of programs and data stored in the memory.

If a certain programmer thinks about the protection of secret data on his program, the system user can read and modify software in the memory. So, all other software including OS can become a means of attack. The attack means the act that exposes secret data on a certain software. In this paper, the object that can become a means of attack is expressed, "Untrustworthy". Moreover, the modification of hardware which needed an exclusive knowledge, a technology, and a huge cost, and achievement is more difficult than the modifications of software in general. Therefore, hardware won't become a means of attack, it is expressed, "Trustworthy". By the way, the reliability of hardware is based on the reliability of the hardware designer.

2.2 Restriction of Processor and OS

Figure 1 shows the structure of an existing and each element of a process that is during the execution of software. The component of the process can be classified into the program and data by the treatment on processor.

When a process is executed, program is read from an external memory through Memory Management Unit(MMU), it stores in instruction register, it interprets, and it is executed with ALU. The change of the value stored in register and writing to an external memory that reflects the execution of instruction are done through MMU.

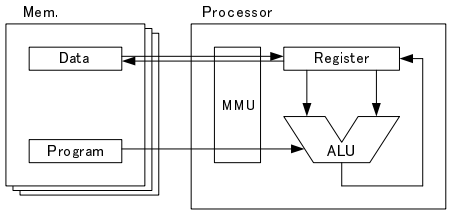

Figure 1. Operation model in existing processor

When the execution process is switched, the content of register on a process is saved in an external memory by OS. MMU converts the virtual address seen from process and the physical address in external memory, and gives memory area isolated from other processes to each process.

OS maintains the management data called Process Control Block(PCB) in process correspondence, and memory map in the process etc. are stored there. The memory management function of OS manages virtual address area seen in each process by rewriting the page table of MMU. This memory management operation is penetration from a process.

When the execution of a process starts once, it is a processor that loads program into instruction register, executs instructions, and accesses data. OS doesn't lie between this process. The data referred frequently is put on register, and was executed immediately before exists together, too. However, it is restricted by MMU, and the data of other processes is able not to be referred to unrestrictedly for the range of address where the process can be accessed to memory.

By the cooperated operation of such processor and OS, a special memory and a processor core are given respectively and program is executed. A virtual execution environment is offered for processes like being able to execute the I/O of data.

It is our purpose to construct architecture that offers secret protection safe for some processes with assumption that a register and an external memory cannot be trusted, and only the mechanism in processor can be trusted. It is necessary to protect process information to be protected even when put on either the processor internal and external. And, it is necessary to have the correspondence with memory management operation with OS from the viewpoint of correspondence with an existing multitasking OS by secret protection, and to penetrate from an application the memory management operation.

3 CDPP Architecture

We propose Cryptographic DataPath Processor(CDPP) architecture in this chapter. CDPP is composed of a special instruction set that achieves the cryptography protection of data and the microprocessor architecture that achieves the instruction set. It is an explain that the instruction set and the microprocessor architecture of CDPP, and management model of data

protection which uses both parties as follows.

3.1 Instruction set

In this subsection, it explains the instruction set of proposal architecture CDPP together with data protection model based on the cryptograph. Figure 2 shows data protection model of CDPP.

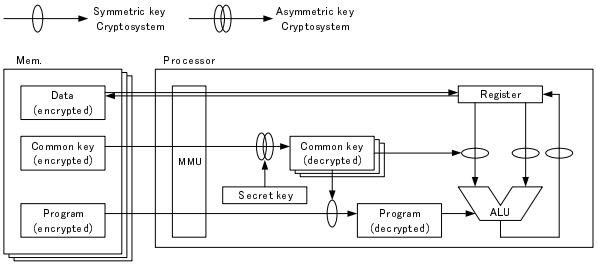

Figure 2. Data protection model in CDPP

3.1.1 Single process

The secret data of process is stored in memory while encrypted with a common key to individual process based on the conventional encryption system. The secret data is read to register while encrypted, and it is decrypted for the first time when it is referred with ALU as an argument of operation. Moreover, if the operation result of ALU is secret data and it is encrypted and stored in register. ALU consists of one-input-one-output function and two-inputs-one-output function. Depending on combining the plain data and the encrypted data, the one-input-one-output function has combination by 2^2 kinds of and the two-inputs-one-output function has the combination by 2^3 kinds of (Table 1). In CDPP, these enhanced operations are mounted as a protective instruction.

Table 1. I/O of ALU

X,Y:input			Z:output	
X	Y	Z	X	Z
p	p	p	p	p
p	p	e	p	e
p	e	p	e	p
p	e	e	e	e
e	p	p		
e	p	e		
e	e	p	p:plain data	
e	e	e	e:encrypted data	

At this time, a common key necessary for encrypting and decrypting secret data is stored in memory while encrypted with the public key. The key pairs with the secret key to processor based on the public key cryptosystem. When the process issues the Key Loading instruction, the common key is decrypted and stored in core. The above is summarized in the Table 2 as a Cryptograph Operating Instruction.

Table 2. Cryptograph Operating Instruction

instruction	operation
Protective Operation	Operation treating secret data
Key Load	Loading common key into processor

Moreover, the instruction can be protected in CDPP, and this encrypted instruction is called a Protected Code. The calculation process of program can be concealed by making a Cryptograph Operating Instruction the Protected Code. The distinction between protection code and non-protection code is achieved by adding the code judgment bit.

3.1.2 Multi-process

In CDPP, secret data is protected independently in each process. The process information on the data protection manages by OS, and is maintained in PCB.

In multiprocessing environment, OS manages parallel execution in a process. OS is issued the Key Operating Instruction shown in the Table 3, registered, switched, and liberated the common key to each process. The decrypted common key is made a capsule in each process, and OS cannot operate directly. Moreover, it is necessary to interpret the Key Operating Instruction independently of Cryptograph Operating Instruction shown in the Table 2. This is achieved by adding the code judgment bit added to the Cryptograph Operating Instruction to the Key Operating Instruction, and making it interpret as Non-protected Code.

Table 3. Key Operating Instruction

instruction	operation
Key Entry	Registering a common key in an individual PCB
Key Switch	Switching the common key to each PCB
Key Release	Liberating the common key of PCB

3.2 Microprocessor architecture

In CDPP, secret data is decrypted just in front of an ALU in datapath and encrypted just behind it, so as not to memorize secret data while decrypted even if in internal registers. Figure 3 shows the structure of datapath around ALU.

The plain data or the encrypted data is selected by instruction as for each I/O, and controller controls this selection by interpreting instruction. If it is not secret data, passing Symmetric Key Cryptosystem(SKC) is not used. In case of treating secret data for the input, SKC decrypts data. In case of the output is secret data, SKC encrypts data. Therefore, secret data is not memorized while encrypted.

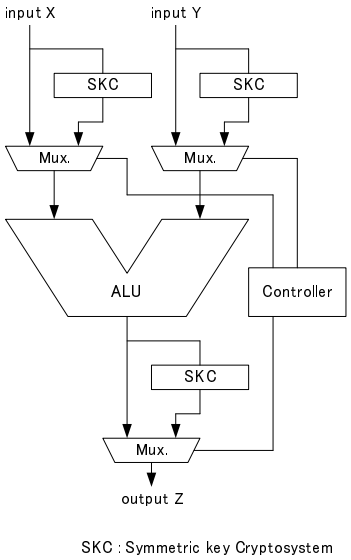

Figure 3. Datapath structure around ALU

3.3 Operation model

3.3.1 Software distribution

Figure 4 shows the operation model of software distribution in CDPP.

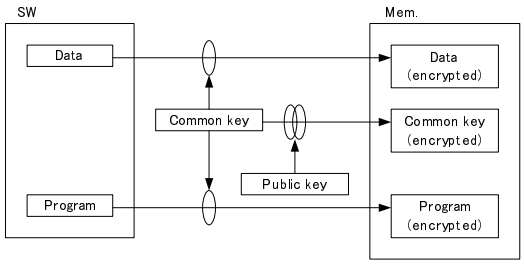

Figure 4. Software distribution model

First of all, a common key that encrypts own software is made according to the conventional encryption system of CDPP, and secret data is encrypted. Next, necessary instructions are made Protected Code while considering the stubbornness of program. Finally, the common key is encrypted with the public key, and distributed together with software.

3.3.2 Software execution

When PCB is generated, the storage table of the common key is secured in controller of processor that OS specified. The common key is decoded with Processor's secret key when the key loading instruction is issued, and it is stored in storage table.

The program is one by one interpreted by instruction fetch when the execution of program starts, and it is executed by ALU. The program specifies the Protected Code or Non-protection Code, is decrypted if it is a Protected Code, and interpreted as plain if it is a Non-protected Code. And, if it is a protection instruction, data is decrypted in former steps of ALU, encrypted by latter part or both parties are executed. The protected data is made a block cipher with the system bus size as a unit.

When the execution of PCB is interrupted, the PCB management function of OS saves contents from register in an external memory. The PCB management function of OS returns contents to register when execution restarts. At this time, because common key is protected by specification of CDPP, OS cannot expose secret data.

The execution environment of each individual process secretly protected is offered with these mechanisms. And, the secret protection can coexist with the resource management mechanism of OS like memory management and schedule, etc.

4 Consideration

4.1 Security

In this section, it is described that programming is possible with safety of secret data kept when Table 2 and 3 instruction is mounted.

The function of the symmetric-key encryption used with CDPP is assumed to be $C(x)$, and programming object that treats protection data $C(x)$ is assumed the function $H(x)$. Moreover, H is assumed that it has one direction, and x is not obtained from the operation result by H^{-1}. When H is done in programming, it is achieved by combining two-inputs-one-output function and one-input-one-output function, and it becomes $H(x) = h_n h_{n-1} \cdots h_2 h_1(x)$. There is danger from which x is exposed during the calculation of this $H(x)$, and the middle variable of these $h_{n-1} \cdots h_2 h_1(x)$, $\cdots$, $h_2 h_1(x)$, $h_1(x)$ becomes the secret data, too. The operation done by one instruction is shown by " $\Rightarrow$", and the calculation process is shown as follows.

$$C(x) \Rightarrow C(h_1(x)) \Rightarrow C(h_2 h_1(x)) \Rightarrow$$

$$\cdots \Rightarrow C(h_{n-1} \cdots h_1(x)) \Rightarrow h_n \cdots h_1(x) = H(x)$$

Thus, even if these middle variables leak, protecting x becomes possible according to strength of $C(x)$ because it makes all the middle variables to which x can be presumed secret data.

Moreover, if the program of calculation process leaks, the middle variable is exposed by replacing Protective Operation with Non-protective Operation, $h_n h_{n-1} \cdots h_2 h_1(x)$ is traced oppositely and x is exposed. However, if the Protective Operation is encrypted, the effective replacement's of the attacker of instruction becoming difficult, and concealing the calculation process of program become possible.

Even if attacker rewrite memory while the execution of PCB stopped, a key is registered after PCB begins and the counterfeit in memory is not significant. The secret data cannot be effectively operated without knowing a common key, it doesn't become an effective attack that only inserts random instruction even if the program is replaced.

4.2 Performance

We consider how much instruction execution throughput decreases when CDPP is mounted on an existing processor. Figure 5 shows the model of program written aiming at the data protection on CDPP.

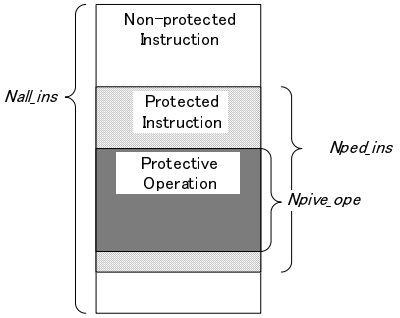

Figure 5. Program model on CDPP

However, to pay attention to one PCB, the Key Load Instruction and Key Operating Instruction are disregarded here.

We define the number of all instructions is N_{all_ins}, the number of Protected Instructions is N_{ped_ins}, the number of Protective Instructions is N_{pive_ope}, the proportion of Protected Instructions in all instructions is $P_{ped\_ins} = N_{ped\_ins}/N_{all\_ins}$, the proportion of Protive Instructions in Protected Instructions is $P_{pive\_ope} = N_{pive\_ope}/N_{ped\_ins}$.

The average time when the processor before CDPP is applied hangs to one instruction execution is assumed to be T_N, and the operation time of the symmetric-key encryption used with CDPP is assumed to be T_{SKC}. The increase rate at the operation time by the symmetric-key encryption of one time becomes $P_{SKC} = T_{SKC}/T_N$, and the average time to execute Protected Instruction of Non-protective Instruction becomes $T_{n\_inst} = T_N + T_{SKC}$.

Moreover, when Protective Instructions of Protected Instructions assumes the ratio that uses the symmetric-key encryption by both before and behind ALU to be extremely a lot of, $T_{pive\_ope} = T_N + 3 * T_{SKC}$. By the way, it is required that Protective Instructions be the Protected Instructions of all here.

Time T that hangs to all instruction execution on CDPP is requested above.

$$
\begin{aligned}
T &= (N_{all\_ins} - N_{ped\_ins}) * T_N \\
&\quad + (N_{ped\_ins} - N_{pive\_ope}) * T_{n\_inst} + N_{pive\_ope} * T_{pive\_ope} \\
&= (N_{all\_ins} - N_{ped\_ins}) * T_N \\
&\quad + (N_{ped\_ins} - N_{pive\_ope}) * (T_N + T_{SKC}) \\
&\quad + N_{pive\_ope} * (T_N + 3 * T_{SKC}) \\
&= N_{all\_ins} * T_N + N_{ped\_ins} * T_{SKC} + 2 * N_{pive\_ope} * T_{SKC}
\end{aligned}
$$

Increase rate P is requested compared with all instruction execution time $N_{all\_ins} * T_N$ before CDPP is applied.

$$
\begin{aligned}
P &= \{(N_{all\_ins} * T_N + N_{ped\_ins} * T_{SKC} \\
&\quad + 2 * N_{pive\_ope} * T_{SKC}) - N_{all\_ins} * T_N\} / N_{all\_ins} * T_N \\
&= (1 + 2 * N_{pive\_ope}/N_{ped\_ins}) \\
&\quad * (N_{ped\_ins} * T_{SKC})/(N_{all\_ins} * T_N) \\
&= P_{SKC} * P_{ped\_ins} * (1 + 2 * P_{pive\_ope})
\end{aligned}
$$

Therefore, the throughput after CDPP is mounted compared before mounted is Expression (1).

$$
(1+P)^{-1} = (1 + P_{SKC} * P_{ped\_ins} * (1 + 2 * P_{pive\_ope}))^{-1} \quad (1)
$$

Figure 6 and 7 show Expression (1). Figure 6 assumes $P_{pive\_ope} = 0.8$, $P_{SKC} = \{0.1, 0.2, 0.3, 0.4\}$, and takes P_{ped_ins} in a horizontal axis. This is an aspect from processor design side, how the overhead of processor by symmetric-key encryption processing changes by mounted application is estimated.

Figure 7 assumes $P_{SKC} = 0.2$, $P_{pive\_ope} = \{0.4, 0.6, 0.8, 1.0\}$, and takes P_{ped_ins} in a horizontal axis. This is an aspect from software design side, how overhead of the processor by symmetric-key encryption processing changes by the mounted program is estimated.

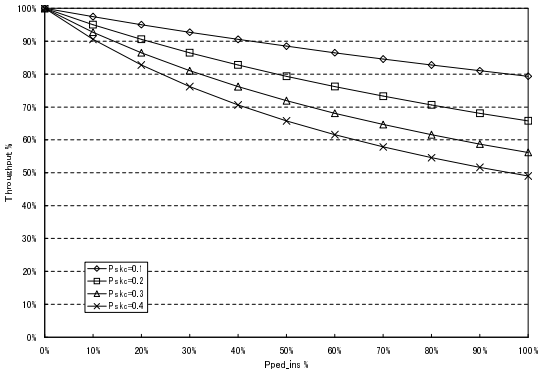

Figure 6. Throughput decrease after CDPP is mounted($\mathbf{P_{pive\_ope} = 0.8}$)

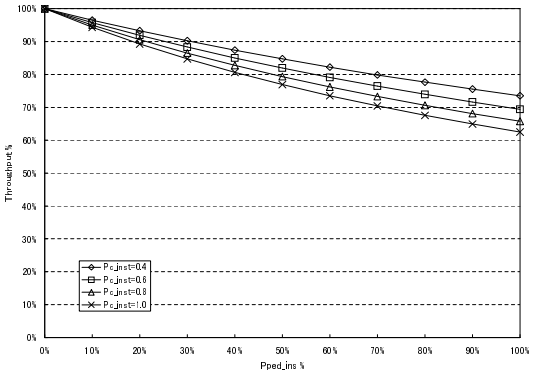

Figure 7. Throughput decrease after CDPP is mounted($\mathbf{P_{SKC} = 0.2}$)

Even if it is the maximum, the lower performance is 20〜40%. This is not a big penalty when thinking the improvement of safety.

5 Conclusions

We proposed CDPP architecture that offered a secret safeguard of process that was able to coexist with the multitasking OS. CDPP installs the next mechanism, and offers the means that program protects secret data of a process leading.

- Cryptgraph control with software
- All the memory units are objects of cryptography protection
- Behavior of process is concealed by the cryptograph

If the processor that can be trust it with CDPP architecture in the terminal is mounted, own secret data can be defended from all software except me including OS of the terminal, and safe data management be achieved. Moreover, CDPP enables program to mount the secret protection mechanism as an instruction, and to protect own secret data specifying it.

Future work is estimating the increase of circuit area and power consumption by CDPP mounting and an examination of the Protective Instruction and cryptograph circuit considering with these overheads.

Acknowledgment

We thank Mr. M. Muroyama, Mr. U. Mesbah, Mr. K. Tarumi, Mr. S. Yamaguchi and Mr. M. Tokunaga for their help of preparation of this article. Thanks are also due to all of members System LSI Research Center of Kyushu University.

This work has been supported by the Grant-in-Aid for Creative Scientific Research No.14GS0218 of the Ministry of Education, Science, Sports and Culture(MEXT) from 2002 to 2006. We are grateful for their support.

References

[1] Markus Kuhn,“ The TrustNo 1 Cryptoprocessor Concept”,April 30, 1997.

[2] T Gilmont, J legat, J Quisquater“ Enhancing Security in the Memory Management Unit”

[3] Lie, Thekkath, Mitchell, Horowitz,“ Architectural Support for Copy and Tamper Resistant Software”, ASPLOS2000, pp.168-177, 2000.

[4] M Hashimoto, H Haruki,“ Multi-vendor Secure Processor under a Hostile Operating System”, IPS of Japan Thesis magazine, Mar, 2004.