Micro-architectural Attacks and Countermeasures

By

Lu Shiting

Supervisor: Prof. Zeng Xiaoyang Dr. Han Jun

Examiner: Prof. Axel Jantsch

Thesis Period: Nov. 2009 – Sep. 2010

Department of Microelectronics and Information Technology (IMIT),
School of Information and Communication Technology (ICT),
Royal Institute of Technology (KTH),
Stockholm, Sweden

M.Sc Diploma Work in System-on-Chip Design Area

Abstract

Micro-architectural analysis (MA) is a fast evolving area of side-channel cryptanalysis. This new area focuses on the effects of common processor components and their functionalities on the security of software cryptosystems. The main characteristic of micro-architectural attacks, which sets them aside from classical side-channel attacks, is the simple fact that they exploit the micro-architectural behavior of modern computer systems. Attackers could get running information through malicious software, then get some sensitive information through off-line analysis. This kind of attack has the following features: 1.) side channel information are acquired through software measurement on target machine with no need to use sophisticated devices. 2.) non-privilege process could get the running information of the privilege process. 3.) people can mount both a remote attack and local attack.

This thesis mainly focuses one kinds of these attacks, data cache based timing attacks(CBTA). First, the main principle of CBTA is introduced, and several kinds of CBTA technique are discussed. Moreover, theoretical model is given under some attacks. Second, various countermeasures are described and their advantages and disadvantages are pointed out. Based on these discussions, the author proposes two anti-attack measures using hardware modification. Aiming at access-driven attacks, a XOR address remapping technique is proposed, which could obfuscate the mapping relationship between cache line and memory block. Aiming at timing-driven attacks, the IPMG mechanism is proposed innovatively. This mechanism could generate cache miss dynamically through observing the historic miss rate. These two mechanisms are realized on the MIPS processor and their effectiveness is verified on the FPGA board. At last, performance penalty and hardware cost are evaluated. The result shows that the proposed solution is effective with very low performance penalty and area cost.

Key Words: Cache attack, AES, countermeasures, side channel, security

Classification code: TN949.197

摘要

随着人们对私密信息的安全性要求的不断提高，各种密码算法正在进入生活的每一个角落。今年来，基于微处理器体结构分析的旁道攻击在信息安全领域迅速发展，此类攻击主要利用了处理器在体系结构的安全缺陷，攻击者通过编写恶意的软件可获得程序运行的相关信息，经过线下分析可获取敏感信息。该种攻击主要具有以下特点：1. 旁道信息可通过软件测量获得而无需特别的设备。2. 非特权进程可通过其获得特权进程的信息。3. 攻击者既可以实施本地攻击也可以实施远程攻击。

本文主要针对其中一类针对数据 cache 的时间攻击进行了深入的研究。首先本文介绍 CBTA 攻击的主要原理，并且分别介绍了几种不同种类的 CBTA 攻击技术并给出了理论分析模型。其次，本文全面介绍了各种 CBTA 攻击的防御技术并分析各自的优缺点，在此基础上本文提出了两种针对两类主要攻击的硬件防御方案。针对基于访问驱动的攻击，提出一种基于异或的地址映射重排技术，该技术通过打乱 cache 行与主存储块的映射关系使得攻击不再可行。针对基于时间驱动的攻击，本文创造性地提出了 IPMG 策略。该策略通过测量历史 cache 缺失率，动态的产生 cache 缺失以调节缺失率维持在一个恒定的水平。该技术可以有效防止攻击者利用 cache 缺失的波动获取敏感信息。

此后，本文在 MIPS 处理器上将上述两种抗攻击措施融合实现，并通过 FPGA 验证了其抵抗攻击的有效性。最后对加入防御机制后所带来的硬件代价和性能损失做了综合的评估。结果表明，提出的防御方案在极低的性能损失和面积代价下能有效抵御相应得攻击。

关键词: AES，旁道攻击，信息安全，cache 攻击，微体系结构分析

中图分类号: TN949.197

Content

Figure Index

Chapter 1 Introduction

1.1 Information Security and Cryptography

Information is an asset that, like other important business assets, is essential to an organization's business and consequently needs to be suitably protected. This is especially important in the increasingly interconnected business environment. As a result of this increasing interconnectivity, information is now exposed to a growing number and a wider variety of threats and vulnerabilities. Especially, with the technology advancing, the so called internet of things (IoT) and cloud computing allows information to flow more fluently. At the same time, security issues, particularly surrounding unauthorized access to and unintended disclosure of data are becoming more prevalent.

Information security is the protection of information from a wide range of threats in order to ensure business continuity, minimize business risk, and maximize return on investments and business opportunities. Information security is achieved by implementing a suitable set of controls, including policies, processes, procedures, organizational structures and software and hardware functions. To protect information includes the following aspects[1]:

1. Data confidentiality: Confidentiality is the protection of transmitted data over shared communication links. It requires that an attacker not be able to observe the content, source and destination, frequency, length, or other characteristics of the traffic on a communication facility.
2. Authentication: The authentication service is concerned with assuring that a communication is authentic.
3. Data integrity: It assures that messages are received as sent, with no duplication, insertion, modification, reordering, destruction or replays.
4. Access control: Access control is the ability to limit and control the access to host systems and applications via communications links. To achieve this,

5. each entity trying to gain access must first be identified, or authenticated, so that access rights can be tailored to the individual.
6. Non-repudiation: Non-repudiation prevents either sender or receiver from denying a transmitted message. Thus, when a message is sent, the receiver can prove that the alleged sender in fact sent the message. Similarly, when a message is received, the sender can prove that the alleged receiver in fact received the message.
7. Availability: An availability service is one that protects system to ensure its availability. This service addresses the security concerns raised by denial-of-service attacks.

1.2 Side Channel Attacks

Side-channel attacks have been demonstrated experimentally against a variety of cryptographic systems. A side channel attack is any attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms. It utilize the fact that in reality, a cipher is not a pure mathematical function $\mathbf{EK}[\mathbf{P}]\rightarrow\mathbf{C}$, but a function $\mathbf{EK}[\mathbf{P}]\rightarrow(\mathbf{C}, t)$, where t is any additional information produced by the actual encryption operation. Often the additional data t leaks enough useful information for an attacker to fully recover the key. There are several ways to acquire the side channel information: timing information, power consumption, electro-magnetic radiation leaks, etc. Along with technical knowledge of the internal operation of the system, one can analyze these side channel information to break cryptosystem.

(1) Timing Analysis Attack

The first timing analysis attack is proposed by Paul C. Kocher[3] in 1996. It mainly targets at public cryptographic algorithm. In some cases, the time needed to complete an encryption or decryption is dependent on the length of key used by some public cryptographic algorithm, such as RSA. If the attacker can choose the data which is the input of cryptographic hardware, the key information can be deduced

through observing the computation time according to corresponded input data. There are many countermeasures are proposed to thwart timing analysis by improving the algorithm. The improved algorithm will present the same computation time using different keys.

(2) Power Analysis Attack

The power consumption of cryptographic devices, such as smart cards, depends on the manipulated data and of the executed instructions. It can be monitored on an oscilloscope by inserting a resistor in series with the ground or power supply pin in some noiseless systems. The so-obtained measurement is called a power trace. Simple power analysis (SPA) [2] is a technique that involves direct interpretation of a power trace. The attack can get part of the key bits through analyze the power information in one cryptography calculation. Due to its directness, this kind of attack is relatively easy to thwart. Differential power analysis (DPA) [2] is a sophisticated power analysis technique that makes use of statistical methods on a collection of power traces. To thwart the DPA attack, some custom design techniques are proposed to get rid of power variation of CMOS devices. However, this will prolong the design cycle and can not meet the time-to-market requirement in some cases.

Alternatively, the masking method can introduce random noises to hide the sensitive power information from being observed by attackers, and it shall raise the effort of analyze for a successful attack.

(3) Fault Analysis Attack

Differential fault analysis is a type of side channel attack in the field of cryptography, specifically cryptanalysis. The principle is to induce faults—unexpected environmental conditions—into cryptographic implementations, to reveal their internal states. For example, a smartcard containing an embedded processor might be subjected to high temperature, unsupported supply voltage or current, excessively high overclocking, strong electric or magnetic fields, or even ionizing radiation to influence the operation of the processor. The processor may begin to output incorrect results due to physical data corruption, which may help a cryptanalyst deduce the instructions that the processor is running, or what its internal

data state is. For DES and Triple DES, about 200 single-flipped bits are necessary to obtain a secret key.

(4) Electro-magnetic Radiation Attack

Electro-magnetic radiation analysis is similar to power analysis, and it can guess out the key by measuring the magnitude of the EM radiation which highly depends on the input data and key material.

1.3 Micro-architectural Analysis Attack

The so-called micro-architectural attack[25, 31] is fast evolving area of side-channel cryptanalysis. The effects of common processor components and their functionalities on the security of software cryptosystems are the focuses of this area. Modern computers are designed to be parallel or quasi-parallel to support multi-task application, and some hardware components are shared among many tasks. The basic idea of micro-architectural analysis is that the shared components can leak information from one process to another, especially from privileged process to unprivileged one.

The MA attack has been proved to be practical and potential harm would be more severe considering popularity of cloud computing in the near future. Cloud Security Alliance (CSA) has identified the following threats in its document [4,5]:

(1) Abuse and nefarious use of Cloud Computing.

(2) Insecure Application Programming Interfaces.

(3) Malicious Insiders.

(4) Shared Technology Vulnerabilities.

(5) Data Loss/Leakage.

(6) Account, Service and Traffic Hijacking.

(7) Unknown risk profile.

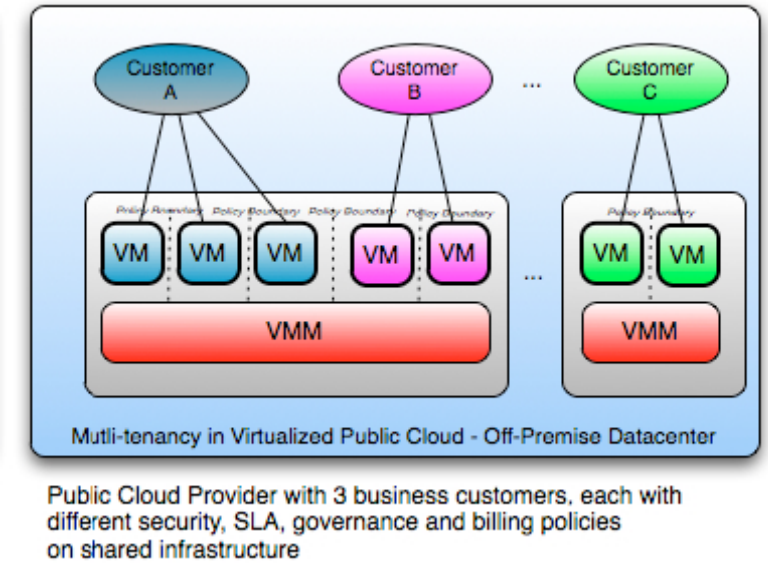

Figure 1. 1 Virtualization Technology in Cloud Computing

As pointed out in the document relating to the 4th threat, the underlying components that make up the cloud computing infrastructure (e.g. CPU caches, GPUs, etc.) were not designed to offer strong isolation properties for a multi-tenant architecture. The virtualization technology has been proved to be an isolation method among different tenant, and a customer can own several VMs[43] which are mananged by virtual machine monitor (VMM) as shown Figure 1. 1[4]. However, the underlying hardware resources are shared by different software agents. Disk partitions, CPU caches, GPUs, and other shared elements were never designed for strong compartmentalization. So customers would have opportunity to have access to any other tenant's actual or residual data, data work traffic, etc. As a result attackers focus on how to impact the operations of other cloud customers, and how to gain unauthorized access to data.

According to the hardware components used to obtain the side channel information, micro-architectural attack can be categorized into four groups.

(1) Data Cache Timing Attack

This attack targetting data cache is the most mature attack in MA attack and many attack methods have been proposed by researchers. Data cache attacks try to reveal the data access patterns of cryptosystems, unlike the following ones which

expose the execution flow of the ciphers. This thesis focuses on protection measure against this kind of attack.

(2) Instruction Cache Analysis Attack

Instruction cache was identified as another micro-architectural analysis source. Unlike data cache which leaks information of data access, instruction cache reveal s fetching information of a program and so the execution flow. The malicious one can leverage the execution flow information to break system down in the system in which the conditional branch is highly dependant on key bits. O. Aciicmez proposed this kind of attack targeting at RSA algorithm[6]. The consequences of cache conflicts are exploited by creating intentional conflicts between the instructions of an RSA process and a spy code and forcing the processor to evict the RSA instructions out of I-cache. By doing so, the attack can obtain the knowledge of the program flow by cache timing.

(3) Branch Predication Unit Analysis Attack

Deep microprocessor pipelines coupled with the ability to fetch and issue multiple instructions at every machine cycle led to the development of superscalar processors. Due to branch cost, superscalar processor rely on branch prediction mechanisms to execute instructions speculatively to overcome control hazards, and branch prediction units (BPU) are used to full-fill this task. The BPU consists of mainly two “logical” parts, the BTB and the predictor. The BTB is the buffer where the CPU stores the target addresses of the previously executed branches. Since this buffer is limited in size, the CPU can store only a number of such target addresses, and previously stored addresses may be evicted from the buffer if a new address needs to be stored instead. Reference [30] demonstrated the first simple branch prediction analysis (SBPA) attack on the S&M algorithm implemented in OpenSSl-0.9.7. It could reveal the execution flow of the RSA process by observing the BTB state transitions during a single RSA operation. So the key can be got through analyzing the execution flow information.

(3) Shared Function Unit Attack

Simultaneous Multi-Threaded (SMT) processors run many processes

concurrently, sharing almost all processor resources in a very tightly coupled way. In particular, the concurrent threads share a pool of functional units (FUs) that are allocated dynamically to each process every cycle. By contending for the shared FUs, one process can interfere with another, leading to side channels. Though in principle this is a typical timing channel, it is orders of magnitude faster than traditional side channels. Consider a system which contains two processes that are not allowed to communicate at all with each other. The insider (sender) process can modulate the use of functional units, e.g. the multipliers, to send information to the receiver process. For example[7], the spy process continuously executes a number of dummy multiplication instructions and measures their execution time. Whenever the other process performs some multiplications by executing its own multiplication instructions, the time measured by the spy will be longer. This is because the spy and the cipher instructions race to occupy the shared multiplier. When the multiplier executes a multiplication instruction from the cipher process, the spy multiplication instructions have to wait their turn until the multiplier finishes its current task, which eventually cause longer execution time for the spy instructions and can easily be detected by the spy.

1.4 Layout of This Thesis

This work mainly focuses on cache based timing attacks and proposes countermeasures against specific attack. The realization of hardware protection strategy is implemented on MIPS processor. The structure of this thesis is as follows.

In chapter 2, some fundamental knowledge of AES and cache is introduced. Also, the basic principal of cache based timing attacks is presented in detail. Three kinds of attack are listed to give the status quo of attack methods and the attack model is given at last.

The chapter 3 gives two protection strategies to thwart access driven and time driven attacks. At the beginning, related work about anti-attack method against CBTA is presented.

In chapter 4, implementation details are given to show how our strategies are realized in real system.

In chapter 5, the environment for effectiveness test and performance evaluation is constructed. Then test results are shown and the performance comparison is given.

The chapter 6 concludes this thesis and makes some proposals about future work.

Chapter 2 Cache Based Timing Attack and Mathematical Modeling

2.1 Fundamentals of CBTA

2.1.1 Memory Hierarchy and Cache Structure

Advances in performance of processors have made the memory hierarchy more and more important. For example, in 1980 microprocessors were often designed without caches, while in 2001 many come with two levels of caches on the chip. Micro-processor performance improved 55% per year since 1987, and 35% per year until 1986. Figure 2. 1[1] plots CPU performance projections against the historical performance improvement in time to access main memory. Clearly, the performance gap between processor and memory is becoming bigger and computer architects must try to close it.

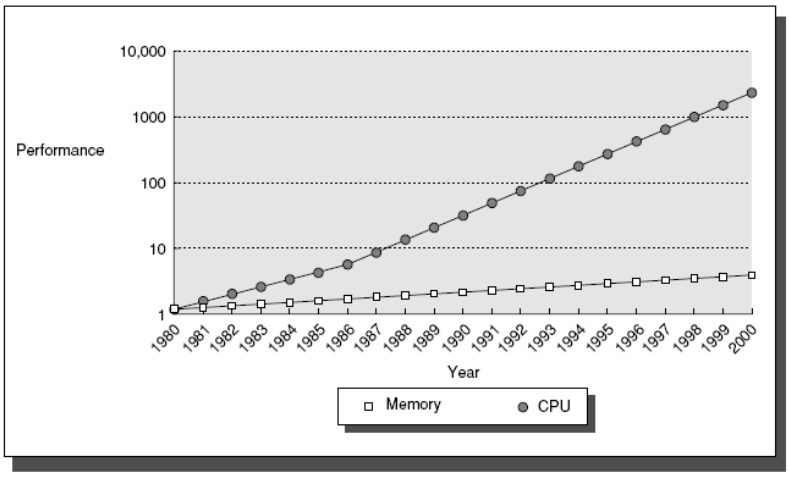

Figure 2. 1 Performance gap between memory and CPU .

A common solution to tackle with this problem is using memory hierarchy to boost the computation performance. Normally, registers within processor core have fastest speed but it also cost large silicon area. The main memory is where the running program and data are stored, and its size is now of several Giga bytes in current PCs and more in servers. The cache memory is used as a bridge to weaken the impact of

the performance mismatch between processor core and main memory. The principle of cache is that the program has the time and space locality, due to which storing some data items in cache will bring some extra benefit. More about cache will be introduced latter. The magnetic disks are of much more larger size but accessing data in it is much slower. The lowest level in memory hierarchy is some offline storage media, like tape or a remote storage server. Figure 2. 2 shows comparison of performance and size among these storage components in a memory hierarchy.

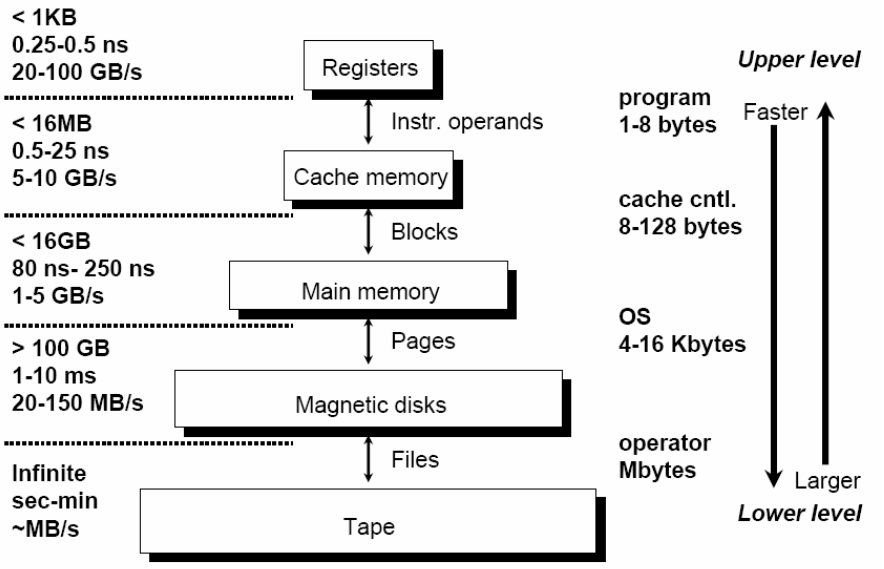

Figure 2. 2 Memory hierarchy of a typical computer.

As pointed out above, cache memory is an essential component of modern computers and can effectively bridge the speed gap between CPUs and main memories. As speed is constrained by capacity, to achieve high speed, cache always has very limited size. So there is a trade off between speed and capacity. Normally, different cache levels are used: L1 cache has smallest size (e.g. 64K), the fastest speed and is closest to processor. L2 cache usually is of size from 512k to several mega bytes. L3 will present in some high performance system and it certainly has larger size. For simplicity, only L1 cache is considered in this thesis, and the principle is the same as for the L2 and L3 cache.

The structure of cache has different types according to the placement strategy from main memory block to cache line. Here are two concepts: memory block and

cache line. Usually, the smallest storage size in main memory is one byte, but we need larger unit as the basic element transferred from memory to cache to leverage the space locality of program. So, the memory block is referred to the smallest unit which is fetched by cache and it always align to a $2^{\wedge n}$ bound. Correspondingly, a cache line is employed to store a memory block. When we discuss the data in memory, the memory block is used and the cache line is used to refer to the data in cache. There are three types of memory block placement into cache lines: directed mapped, fully associative and set associative. If each block has only one place it can appear in the cache, the cache is said to be direct mapped. The mapping is usually block address mod number of blocks in cache. If a block can be placed anywhere in the cache, the cache is said to be fully associative. If a block can be placed in a restricted set of places in the cache, the cache is said to be set associative. The memory block can be placed any cache lines within a set. The set is usually chosen through a modular computation like block address mod number of sets in cache. The range of cache from direct mapped to fully associative is really a continuum of levels of set associativity: direct mapping is simply one-way set associative and a fully associative cache with with m blocks cound be called m-way set associative. In Figure 2. 3, there are 8 sets in cache, so the memory blocks at location 0, 8, 16 and etc. could be place in set 0.

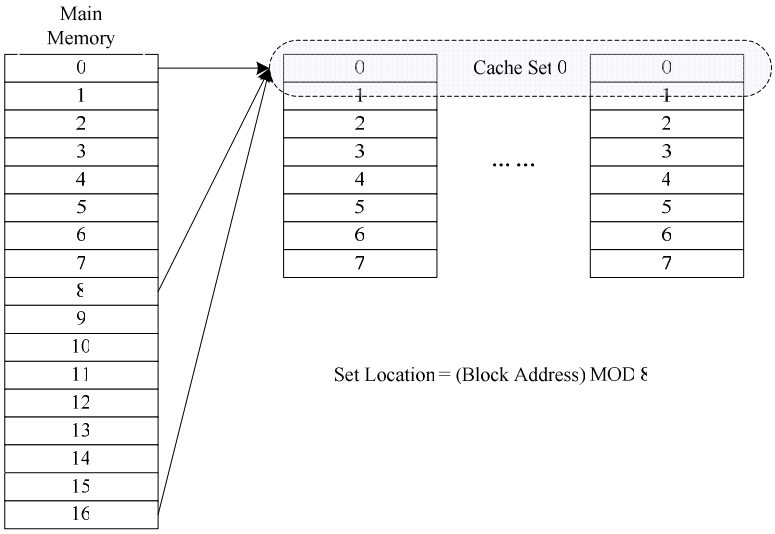

Figure 2. 3 Memory block placement in set associative cache

Since the main memory is much larger than cache memory, many memory blocks are contending for limited cache lines in one set. Consequently, different accesses contending for the same cache entry induce data conflict and replacement occurs. Actually, cache-miss costs much more time than cache-hit because of fetching the requested data from main memory to cache, which makes the modern processors vulnerable to CBTA. Rather than exploring in mathematical domain, CBTA try to attack the cipher by discovering the side-channel information leaked by key-dependent memory accesses if these accesses will be speeded by the cache-hit or penalized by cache-miss.

2.1.2 AES Algorithm

DES algorithm is the most widely used cryptographic algorithm in the world wide. However, with the sharp increasing of performance of microprocessors, the crack of 64-bit DES algorithm is becoming feasible, so the security of information is potentially challenged.

Since considering the vulnerability of DES[9], triple DES (3DES) algorithm became popular. Although the counter-attack resistance of 3DES is strengthened greatly, the fatal shortage of both DES and 3DES is that the length of block is 64 bit. In terms of efficiency and security, longer block size need to satisfy the requirement of symmetric algorithms in new age.

In October 2000, Rijndael algorithm wined out among many competitors and was selected as new standard, Advanced Encryption Standard (AES)[10], by NIST due to its advantages in security, performance, efficiency and flexibility of implantation.

AES is block encryption algorithm--that is the basic processing unit is a data block with fixed length. NIST has designated the length of AES block is 128 bit, and the key which is used in algorithm is 128 bit, 192 bit or 256 bit. In this thesis, 128 bit key length is employed and the other two cases are theoretically the same except the number of rounds in the crypto-procedure. In recent years, it becomes a popular and commonly used symmetric block cipher.

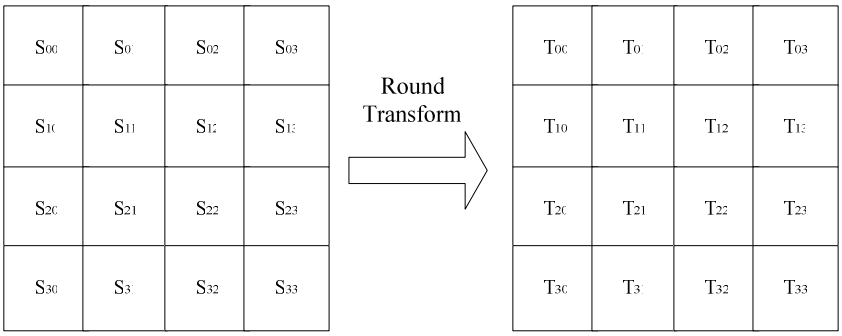

Figure 2. 4 State matrix in AES algorithm

A data block in AES is arranged in a byte matrix which is called state matrix as shown in Figure 2. 4. The basic transform of AES is round transform which comes in three types: byte by byte, column by column and row by row.

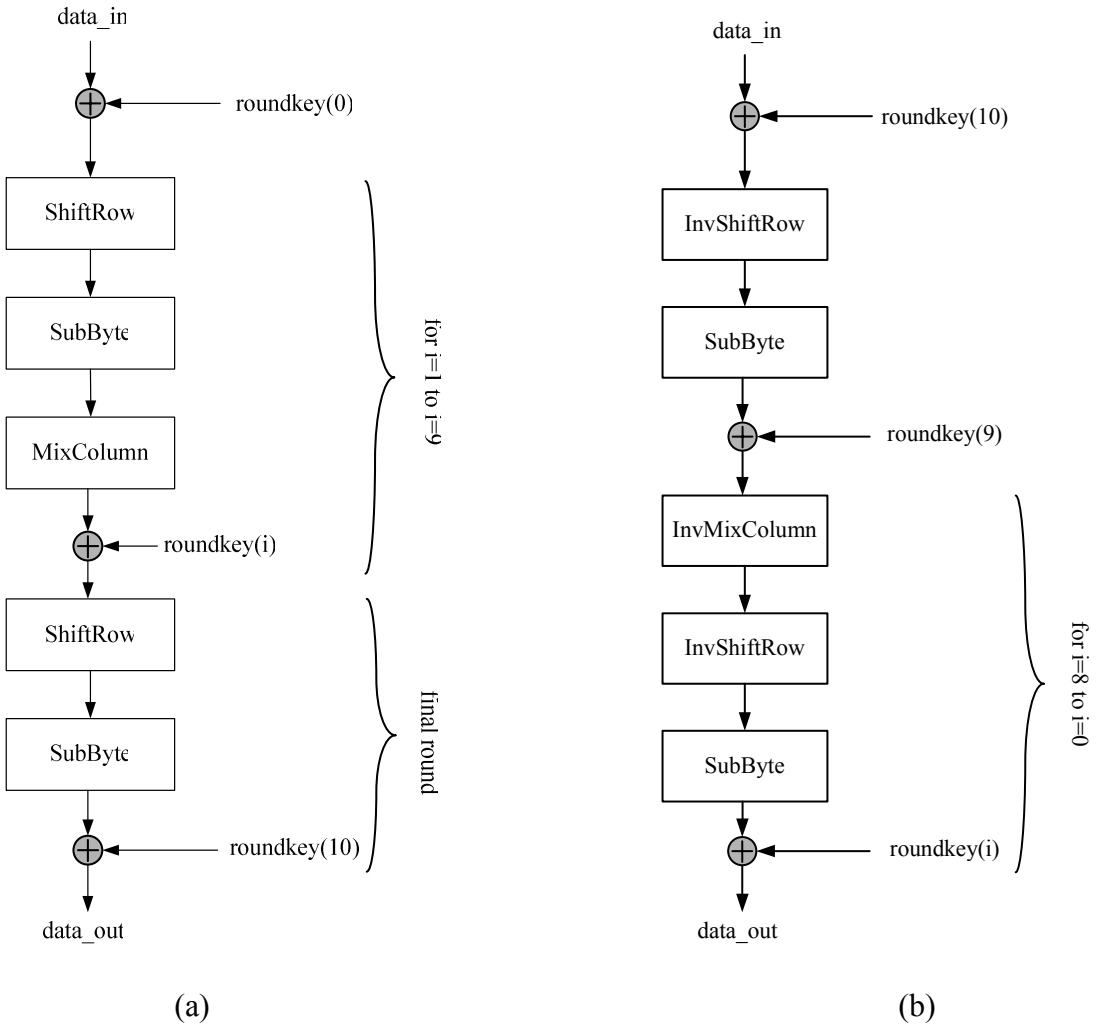

Figure 2. 5 Encryption and decryption procedure flow of AES algorithm

Figure 2. 5 shows the encryption procedure of AES algorithm, including SubByte, ShiftRow, MixColumn and KeyExpansion. For 128 bit key length, these operations should iterate for 10 times. The decryption algorithm is the inverse calculation of encryption and the basic operation includes InvSubByte, InvShiftRow, InvMixColumn. Figure 2.5 (a) is the computation flow of AES encryption and (b) shows the decryption flow of AES.

2.2 Software Implementation of AES

Since the elementary operations in AES are done in Galois Field. For a general purpose processor, it would be quite timing consuming to do calculation in finite field, and the performance will be sometimes unacceptable. So, in software implementation the lookup tables are used to accelerate the Galois field calculation. Here, the lookup table is a set of pre-calculated results of SubByte, ShiftRow and MixColumn, and we could use the input as the index of tables to get the output.

For instance, 4 lookup tables are introduced in openssl 0.9.8j [11]. Secret key of i^{th} round could be expressed as $K_i=(k_{0,i}, k_{1,i}, k_{2,i}\ldots\ldots k_{15,i})$, which will be calculated in advance during key expansion. Let $PT = (pt_0, pt_1, pt_2\ldots\ldots pt_{15})$ and $CT = (ct_0, ct_1, ct_2\ldots\ldots ct_{15})$ denote plaintext and ciphertext respectively. When the intermediate result of i^{th} round is defined as $X_i=\{x_{0,i},x_{1,i}, x_{2,i}\ldots\ldots x_{15,i}\}$, operation could be expressed as in equation (2-1), where X_0 represents XOR result of plaintext and raw key. In encryption stage, there are 160 table lookups in total and 40 accesses to each data table, which are actually utilized by adversary in cracking AES.

$$
\begin{aligned}
(x_{0,(r+1)},x_{1,(r+1)},x_{2,(r+1)},x_{3,(r+1)}) &\leftarrow T_0\left[x_{0,r}\right]\oplus T_1\left[x_{5,r}\right]\oplus T_2\left[x_{10,r}\right]\oplus T_3\left[x_{15,r}\right]\oplus\{k_{0,r+1},k_{1,r+1},k_{2,r+1},k_{3,r+1}\}\\
(x_{4,(r+1)},x_{5,(r+1)},x_{6,(r+1)},x_{7,(r+1)}) &\leftarrow T_0\left[x_{4,r}\right]\oplus T_1\left[x_{9,r}\right]\oplus T_2\left[x_{14,r}\right]\oplus T_3\left[x_{3,r}\right]\oplus\{k_{4,r+1},k_{5,r+1},k_{6,r+1},k_{7,r+1}\}\\
(x_{8,(r+1)},x_{9,(r+1)},x_{10,(r+1)},x_{11,(r+1)}) &\leftarrow T_0\left[x_{8,r}\right]\oplus T_1\left[x_{13,r}\right]\oplus T_2\left[x_{2,r}\right]\oplus T_3\left[x_{7,r}\right]\oplus\{k_{8,r+1},k_{9,r+1},k_{10,r+1},k_{11,r+1}\}\\
(x_{12,(r+1)},x_{13,(r+1)},x_{14,(r+1)},x_{15,(r+1)}) &\leftarrow T_0\left[x_{12,r}\right]\oplus T_1\left[x_{1,r}\right]\oplus T_2\left[x_{6,r}\right]\oplus T_3\left[x_{11,r}\right]\oplus\{k_{12,r+1},k_{13,r+1},k_{14,r+1},k_{15,r+1}\}
\end{aligned}
\quad (2\text{-}1)
$$

Since there is no MixColumn operation in final round, another lookup table is constructed to tackle with this special round. So the round transform is shown in the

following formula (2-2). As we can see in the following sections, this table can be a very vulnerable feature.

$$\begin{aligned} C = \{ & T4[x_0^{10}] \oplus k_0^{10}, T4[x_5^{10}] \oplus k_1^{10}, T4[x_{10}^{10}] \oplus k_2^{10}, T4[x_{15}^{10}] \oplus k_3^{10}, \\ & T4[x_4^{10}] \oplus k_4^{10}, T4[x_9^{10}] \oplus k_5^{10}, T4[x_{14}^{10}] \oplus k_6^{10}, T4[x_3^{10}] \oplus k_7^{10}, \\ & T4[x_8^{10}] \oplus k_8^{10}, T4[x_{13}^{10}] \oplus k_9^{10}, T4[x_2^{10}] \oplus k_{10}^{10}, T4[x_7^{10}] \oplus k_{11}^{10}, \\ & T4[x_{12}^{10}] \oplus k_{12}^{10}, T4[x_1^{10}] \oplus k_{13}^{10}, T4[x_6^{10}] \oplus k_{14}^{10}, T4[x_{11}^{10}] \oplus k_{15}^{10} \} \end{aligned} \tag{2-2}$$

The Figure 2. 6 gives a code snippet in OpenSSL0.9.8 and the main operations are table lookup and exclusive-or.

```
 /* round 1: */
t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[ 4];
t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[ 5];
t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[ 6];
t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[ 7];

/* round 2: */
s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3 & 0xff] ^ rk[ 8];
s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0 & 0xff] ^ rk[ 9];
s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1 & 0xff] ^ rk[10];
s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[11];
```

Figure 2. 6 Code fragments in openssl 0.9.8j

2.3 Access Driven CBTA

In access driven attack[18,19], we get the access information of every cache line or memory block. Since the key and a plaintext are combined to decide the access pattern of cache line, we can use access information to get the key. Here, we assume that there are no AES lookup table data items before encryption or decryption. So, the access pattern got is caused purely by cared cryptographic operation. As a result, always access-driven attacks require cache-eviction by loading a hostile cache-sized table before cryptographic operation, and by reading the data table again afterwards adversary could be aware of which part of cache has been evicted.

During one AES encryption/decryption of random plaintext/ciphertext, entries of table lookups having been accessed could be showed as in equation (2-3). Here, the elements in access set $S_{p/c}$ denote the indexes of accessed items in lookup table used by AES. Access-driven attacks work under the assumption that there is no any

common item in $S_{p/c}$ given completely random plaintext/ciphertexts. However, if certain byte is fixed in plaintext/ciphertext, there will be one and only one common item among the access sets.

$$S_{p/c} = \{I_{T_0,1}, I_{T_0,2}...I_{T_1,1}, I_{T_1,2}...I_{T_2,1}, I_{T_2,2}...I_{T_3,1}, I_{T_3,2}...\} \quad (2\text{-}3)$$

Under this assumption, adversary could make AES determinedly access certain cache line and then discover the corresponding table entry by pinpointing the location where cache-miss always occurs during rereading his own data table. For an obvious speed gap between cache and lower memory architecture, we could expect to pick out cache-misses from cache-hits by observing a single timing sample.

However, in embedded system, it is not the case at all, so adversary should seek help from statistical differential timing analysis to confirm table entry. With the confirmed table entry, it is easy to recovery the secret key, because, as given in equation (2-4), the first round lookup indexes are determined by XOR of corresponding plaintext and symmetric key bytes. Here, λ represents the cache line size in words. Since adversary is not capable to distinguish accesses in one cache line, given fixed pt_i and the confirmed entry $I_{Ti,x}$, adversary could only extract the upper part of key byte k_i. Seeking help from second round attacks, he could fully recovery the key material.

$$I_{T_i,x} = \left\lfloor pt_{i+4j} \oplus k_{i+4j} / \lambda \right\rfloor,\ j = 0,\ 1,\ 2,\ 3 \quad (2\text{-}4)$$

If we use statistical differential timing analysis in access-driven attack, the probabilities of AES operations accessing certain cache lines must be studied here. For fully random plaintext, there are 50 accesses to each lookup table in all but 10 accesses are identical due to fixed round-key expansion. Every unvisited lookup table entry is assumed being accessed with the equal probability as given in equation (4) where l is the cache lines one table occupies, p represents probability accessed and q for not.

$$p = 1 - q\ ,\ q = (1 - \frac{1}{l})^{40} \quad (2\text{-}5)$$

For plaintext with one byte fixed, one cache line will always be accessed in the first round and so in the whole encryption process. Refer to the constantly accessed

cache line as CACL. The other entries in the same table are accessed with the following probability.

$$p_1 = 1 - q_1 \ , \ q_1 = (1 - \frac{1}{l})^{39} \tag{2-6}$$

Similarly, for plaintexts with that byte unequal to the fixed value, cache line corresponding to CACL is accessed with zero probability in the first round. After a whole encryption, the access probability is also equal to p_1. And other table entries in the same table are accessed with probability as:

$$p_2 = 1 - q_2 \ , \ q_2 = (1 - \frac{1}{l-1})(1 - \frac{1}{l})^{39} \tag{2-7}$$

To give an intuitive view of these trivial things, we take a typical configuration for an example. With cache line of 4 words and lookup tables of 256 words each, the following numerical results can easily obtained: p=0.4674, q=0.5326; p_1=0.4589, q_1=0.5411; p_2=0.4675, q_2=0.5325. Statistically, we can distinguish the particular line from other ones and this makes access-driven attacks possible and reliable.

In a real attack, we classify samples into two groups based on whether $<pt_0>$ equals 0. Calculate the differences of average access times of these two groups cache line by cache line. And there is a peak as observed in Figure 2. 7.

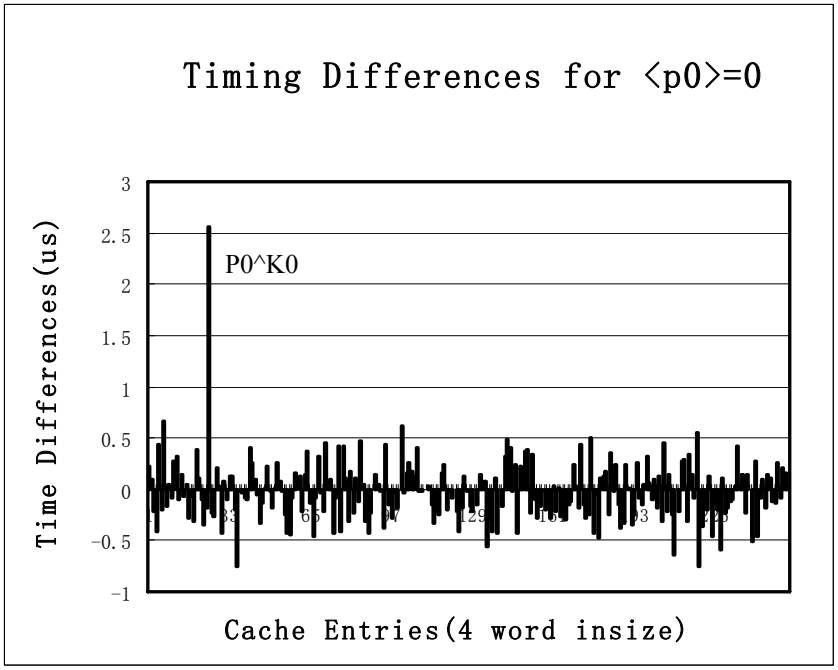

Figure 2. 7 Attack demo of access driven CBTA

2.4 Trace Driven CBTA

The adversary is assumed to be able to capture the profile of the cache activity during an encryption in trace-driven attacks[28]. This profile includes the outcomes of every memory access the cipher issues in terms of cache hits and misses. Therefore, the adversary has the ability to observe if a particular access to a lookup table yields a hit and can infer information about the lookup indices, which are key dependent. This ability gives an adversary the opportunity to make inferences about the secret key.

In trace-driven cache attacks, the adversary obtains the traces of cache hits and misses for a sample of encryptions and recovers the secret key of a cryptosystem using this data. We define a trace as a sequence of cache hits and misses. For example,

MHMM, HMHM, MMHM, HHMH, MMMM, HHHH

are examples of a trace of length 4. Here H and M represents a cache hit and miss respectively. The first one in the first example is a cache miss, the second one is a hit, and so on. If an adversary captures such traces, she can determine whether a particular access during an encryption is a hit or miss. And then using some dedicated method she can recover the key material through these traces.

2.5 Timing Driven CBTA

The methods of the above two attack are somewhat complicated because the access of every cache line is timed. However, the timing driven CBTA [16,29]is quite easy and it only needs to measure the total encryption or decryption time. Although the total number of access is constant, the cache collision is not necessarily the same. So more cache hit will result in less execution time, and we could get the key information according to known plaintext or ciphertext.

Unlike access-driven attacks, time-driven attacks utilize the correlation between total number of cache-misses and execution time. This assumption rests on the approximation that the individual table lookups in the sequence are approximately independent for random plaintexts, which seems to hold in practice. There are 160 table lookups in an AES encryption, and take 16byte cache line for example. We

randomly generate 2,000,000 single block plaintext for encryption input. We can get the miss rate distribution through software simulation as shown in Figure 2. 8. Here, we make the assumption that the cache is big enough to hold all tables and there are no inner-collisions in AES encryption process. From the figure, we can observe that the miss rate exhibit a normal distribution rather than keep at a constant value.

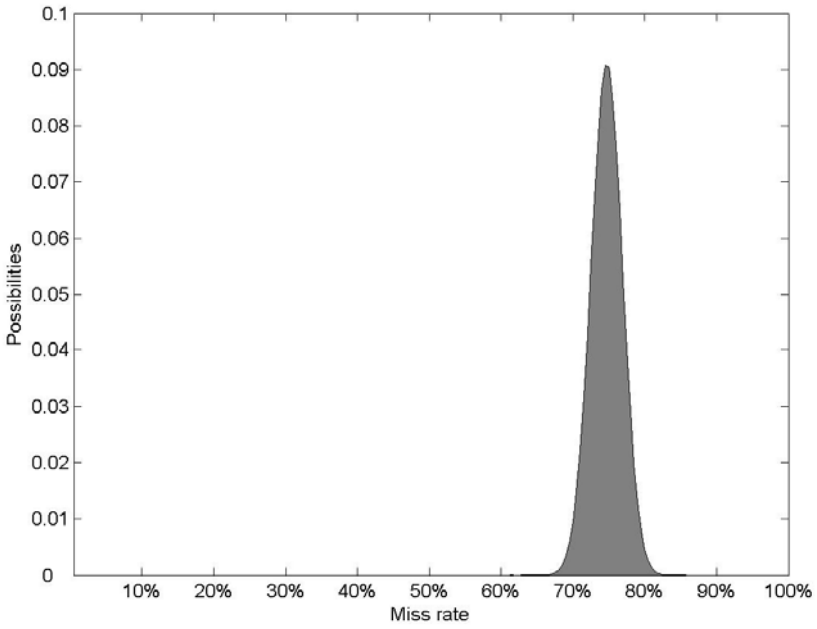

Figure 2. 8 Miss rate distribution of AES encryption from software simulation.

Make use of either more misses leading longer execution time [15] or shorter time due to cache collisions [16], and the latter one receives prevalent hail for its simple implementation, we can get miss rate information to measure encryption or decryption time.

2.5.1 Differential Analysis

There are several analyzing techniques to deduce the AES key byte by byte after getting the timing information. One of cache-collision attacks working unfailingly relies on the assumption that the average execution time of cryptographic operations with two separate accesses constantly visiting a same cache line will be less than average time spent in encrypting/decrypting absolutely random plaintexts/ciphertexts.

So，we can make key values guesses and the correct guess will present a pulse among other incorrect ones. We call this kind of technique differential analysis.

Taking size of cache line into consideration, this assumption could be expressed as cryptographic operations having two constant accesses to a same cache line will take less time than those not having, and the entries of lookup table corresponding to this cache line must satisfy equation (2-8) . Let <X> denotes X ignoring the lower log2 λ bits.

$$\langle I_i \rangle = \langle I_j \rangle \tag{2-8}$$

So we can suppose cache accesses are independent for random plaintexts/ciphertexts. As mentioned in [20], the probability $P_{w,l}(k)$ that k lines of a lookup table occupying l=(256/ λ) cache lines are accessed after w mutually independent accesses is given by the first equation (2-9).

$$P_{w,l}(k) = \frac{C_l^k \left(\sum_{i=1}^{k} (-1)^{k-i} C_k^i i^w \right)}{l^w} \tag{2-9}$$

Based on equation (2-9), it's convenient to calculate the expected value and the variance of the number of cache-misses after w accesses as showed in equation (2-10). Noting that the probability of a single cache line not being loaded into the cache after w accesses is $(1-1/l)^w$, we can derive a relatively simple expression. The average number of cache lines among clean ones not being polluted is given by $l(1-1/l)^w$, and the expected number of lines that are loaded into clean line is $l-l(1-1/l)w$.

$$\begin{aligned} \mu_{Tc}(w,l) &= \sum_{i=1}^{l} i * P_{w,l}(i) = l - l\left(\frac{l-1}{l}\right)^w \\ \delta^2_{Tc}(w,l) &= \sum_{i=1}^{l} i^2 * P_{w,l,s}(i) - \mu^2_{Tc}(w,l) \end{aligned} \tag{2-10}$$

Here, we suppose that the cache is large enough to hold the whole table so that there are only compulsory misses (first access to the data block). Therefore, the total cache misses are equal to lines polluted by encryption/decryption. Thus, $\mu_{Tc}(w, l)$ can be considered as the expected number of cache misses during one operation. If we fix the difference of two plaintext bytes to make intentional collision, the average cache miss number should be equal to $\mu_{Tc}(w, l)$ which is statistically lower than completely

random ones.

For time-driven attacks, the adversary guesses the difference of certain two bytes of cipher key, and then divides the samples into two groups according to guess value. G_M is the sample set with elements whose corresponding byte difference of plaintexts or ciphertexts is equal to guessed key byte difference, but elements of G_N do not have such characteristic. If we have made a correct guess, the time spent by samples of G_M in one encryption/decryption can be denoted as S_4 and for G_N it is denoted as S_5. Under most circumstances, wrong guess is made and the time values are referred to as S_6 and S_7 for G_M and G_N respectively. Thus, we construct two statistics U_{MIN} and U_T in (2-11) and by distinguishing these two variables the correct key guess can be picked out from wrong ones.

$$U_{MIN} = \frac{\sum_{(i \in G_M)} S_4}{m} - \frac{\sum_{(i \in G_N)} S_5}{n}$$
$$U_T = \frac{\sum_{(i \in G_M)} S_6}{m} - \frac{\sum_{(i \in G_N)} S_7}{n} \tag{2-11}$$

The expected values and variances of the two variables of equation (2-11) are enumerated in equation (2-12). The results obtained above are used in the motivation. The following is a brief explanation of the first expression of (3-4). The terms *(a1-a2)(μ(w-1,l)+3μ(w,l))* and *4(a1-a2)μ(w,l)* represent the miss penalty and $4wa_2$ is the minimum time needed by each access.

$$E(U_{MIN}) = \frac{m[(a_1-a_2)(\mu(w-1,l)+3\mu(w,l))+4wa_2]}{m} - \frac{n[4(a_1-a_2)\mu(w,l)+4wa_2]}{n}$$
$$= -(a_1-a_2)(1-1/l)^{w-1}$$
$$D(U_{MIN}) = (a_1-a_2)^2[\frac{\delta_{Tc}^2(w-1,l)+3\delta_{Tc}^2(w,l)}{m} + \frac{4\delta_{Tc}^2(w,l)}{n}] + \delta_b^2\left(\frac{1}{m}+\frac{1}{n}\right) \tag{2-12}$$
$$E(U_T) = 0$$
$$D(U_T) = \left(4(a_1-a_2)^2\delta_{Tc}^2(w,l)+\delta_b^2\right)\left(\frac{1}{m}+\frac{1}{n}\right)$$

The statistic $R_T = U_{MIN}$-U_T follows normal distribution, and we could deduce the sample number needed in time-driven attacks. The total sample number N_{RT} is shown

in equation (2-13).

$$n_{R_T} = \frac{Z_\alpha^2}{(1-1/l)^{2(w-1)}}\left\{8\delta_{Tc}^2(w,l)(1+\beta)+\left(\delta_{Tc}^2(w-1,l)-\delta_{Tc}^2(w,l)\right)\beta+\frac{2(1+\beta)\delta_b^2}{(a_1-a_2)^2}\right\} \quad (2\text{-}13)$$

$$N_{R_T} = (1+1/\beta)\,n_{R_T}$$

Similarly, by defining SNR_{RT} as given in equation (2-14), N_{RT} could be expressed as in equation (2-15). In equation (2-15), the expression in numerator is the intrinsic fluctuation due to random input and the denominator is caused by system noise.

$$SNR_{RT} = \frac{(a_1-a_2)^2[8\delta_{Tc}^2(w,l)(1+\beta)+\left(\delta_{Tc}^2(w-1,l)-\delta_{Tc}^2(w,l)\right)\beta]}{2\delta_b^2(1+\beta)} \quad (2\text{-}14)$$

$$N_{R_T} = (1+1/\beta)\frac{Z_\alpha^2\left(1+1/SNR_{R_T}\right)\left[8\delta_{Tc}^2(w,l)(1+\beta)+\left(\delta_{Tc}^2(w-1,l)-\delta_{Tc}^2(w,l)\right)\beta\right]}{(1-1/l)^{2(w-1)}} \quad (2\text{-}15)$$

Through anatomizing the equation (2-15), we could learn that the miss penalty a_1-a_2 also appears in the SNR_{RT} expression, which means that SNR_{RT} will be much lower on embedded system than PC. So, for time-driven attack, the difficulty to crack the embedded system down is extremely huge. In the verification section, we will introduce another way to the timing information different from existing ones. The low noise property of the new method leads to timing attack against embedded system much easier.

2.5.2 Correlation Analysis

Another effective method is to make correlation between measurements and estimations, and the correct key guess will present a correlation pulse among other guesses. Normally, the correlation attacks aim to the final round of AES algorithm because a separate table T4 is used and there are 16 accesses to this table. The collision probability will be much larger compared with the other four tables in first round. So the final round reveals much more effectiveness.

We could define the measurements as a set $\{m_1, m_2, m_3, m_4, \ldots, m_n\}$. For

simplicity, we assume that there are several key bytes which have been broken down. Here, the i^{th} key byte has been successfully guessed out. We have 256 guesses for the j^{th} key byte. For every guess, we could calculate the final round indices used to lookup the T_4 table. As a result, we could make an inference whether a cache collision would happen. For the measurements above, we could obtain a set of estimation and make correlation of the two sets using the following formula.

$$\rho = \frac{E(E_{K_{secret}} \cdot M) - E(E_{K_{secret}}) \cdot E(M)}{\sqrt{E(E_{K_{secret}}^2) - E(E_{K_{secret}})^2}\sqrt{E(M^2) - E(M)^2}} \tag{2-16}$$

For other guesses, the correlation coefficient can be got using similar procedure. There should be a pulse at the correct guess since the estimations of cache collision are as predicted as the real case.

An analytical model was constructed by Kris Tiri *et al*. [20] to predict the required number of measurements for a successful table index estimation attack. The following equation is used to estimate the resistance of system.

$$N = 3 + 8 \times (\frac{Z_\alpha}{\ln(\frac{1+\rho}{1-\rho})})^2 \approx \frac{2 \times Z_\alpha^2}{\rho^2} \tag{2-17}$$

2.6 Typical Attack Flow

A typical attack flow is shown in Figure 2. 9. Usually, an attack includes two phases: sampling phase and analyzing phase.

1. Sampling phase: The timing information is collected in this phase. In synchronous attacks, the plaintext or ciphertext is known and the attack can operate synchronously with the encryption on the same processor, by using some interface that triggers encryption under an unknown key. For example, a Virtual Private Network (VPN) may allow an unprivileged user to send data packets through a secure channel. This lets the user trigger encryption of plaintexts that are mostly known, and our attack would thus enable any such user to discover the key used by the VPN to protect all users' packets. According to the machine the spy process run on, there are two kinds of attack which are local attack and remote attack. The local attack samples data on

local machine, but the remote attack samples the timing data sent by another machine on the network. In the later case, the network delay is included in the timing machine and the uncertainty of the delay may introduce the sampling noise.

The sampling technique is very important in CBTA since an effective and low noise sampling could make the later analyzing much more easier. The noise source may includes following aspects: operating system actions like memory management and process switches, other process impact due to multi-process property of modern system, the resolution of time stamp function used in measurement, network delays, etc. Although some can be cancelled out, some would make the attack impossible. For example, if the resolution of time stamp function can not distinguish a cache hit and a cache miss, the access driven and trace driven attacks cannot be mounted.

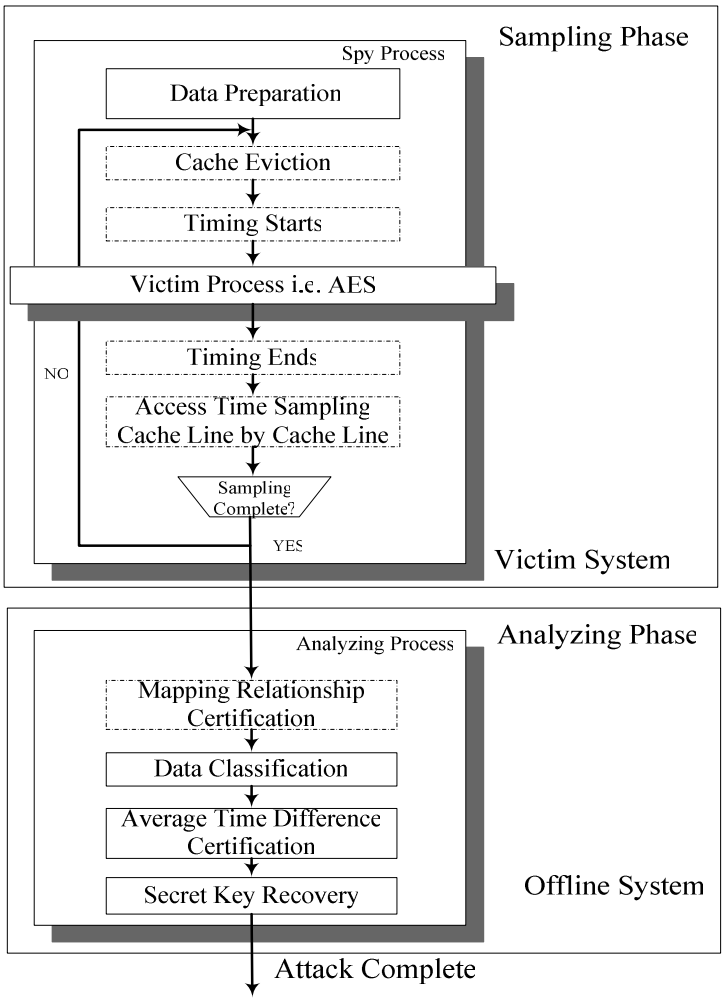

Figure 2. 9 Typical Attack Flow

2. Analyzing Phase: the key bits are got through analyzing the data obtained from the sampling phase. If an attacker knows the plaintexts, the first round properties is analyzed and this is called first round attack. Normally, the cache line are composed of several lookup table items, and one can only get the most significant bits in every single byte. The second round analyzing should be carried out to get the least significant bits.

If an attacker knows the ciphertexts, the final round is analyzed and it can obtain all bits unlike the first round attack. However, the inverse S-box calculation should be done in final round attack, so the computation effort in final round attack would be larger.

There are some differences between access driven attack and time driven attack. As shown Figure 2.8, the dashed blocks are specific to access driven attack. In one encryption/decryption, there is only one data for time driven attack and there is a set of data for access time driven attack.

2.7 Summary

In this chapter, the principal of cache based timing attack (CBTA) are introduced. Three types of CBTA is given and their mathematical models are developed to analyze the main factors of corresponding attack. At last, the typical attack flow is given

Chapter 3 Anti-attack Measures to Thwart CBTA

In this thesis, we focus on time-driven and access-driven attacks. It is not difficult to uncover the fact that the time difference for a cache hit and cache miss is the source of side channel leakage utilized by these two kinds of attacks since it is indeed the measurable parameter used to reveal the key material. Therefore, how to hide this difference becomes a key point to devise a countermeasure. But according to different types of attacks, the anti-attack methods are rather different and we will review some countermeasures in Section 3.1. In order to defeat the two main types of cache-based timing attacks at low cost and small performance penalty, we propose a novel secure cache architecture and related techniques. The work in this chapter is the main contribution of this dissertation, particularly the IPMG mechanism proposed in section 3.3 can be used to thwart the timing-driven attacks.

3.1 Related Work about Thwarting CBTA

Various defend techniques have been proposed to thwart cache timing attacks[22, 23, 26], including both software mitigation and hardware modification. Aiming at different types of attacks, the measures employed to defend system are also quite distinct. These methods are introduced in following sub-sections.

3.1.1 Countermeasures against Access Driven Attack

To defend the access driven attack, several obfuscating methods have been proposed by reference [33] and [24]. In reference [33], the random permutation of the lookup tables hides mapping relationship between the cache line index and table item index. The main drawback of this method is that the permutation is not dynamical and could be cracked down by sophisticated strategy [33]. From the perspective of hardware, Wang et al. [24] proposed a cache architecture which permutes the cache lines dynamically and can achieve some performance gain at the same time. But

hardware cost of the scheme is not trivial.

3.1.2 Countermeasures against Time Driven Attack

To defend the time-driven attack is somewhat challenging and quite state of the art since the presence of cache makes the memory access no longer time-constant. Moreover, it would always bring some performance deterioration to diminish time fluctuation. In fact, software methods mainly concentrate on avoiding cache using. For example, Kong et al. [27] proposes three methods to defend the time driven attack, but the basic idea under all these methods is preloading all critical data before they are really wanted by the processor. By doing so, the time variation which is correlated with the input/output material will diminish, and statistical analysis of the time measurements is infeasible. However, the performance penalty is also quite taxing.

3.1.3 Pros and Cons for Using Small Lookup Table

To defend both access-driven attacks and time-driven attacks, one proposed approach is to construct small lookup tables which only occupy one or two cache line. The first advantage of small table is to suppress access-driven attacks because the measuring skills cannot differentiate different access in one single cache line and so the access pattern of lookup table is invisible to attackers. Furthermore, because the small table would be entirely fetched into cache by one or a few accesses due to its small size, the time variation between different encryption executions will decrease largely, which invalidates the method of time-driven attacks. Nevertheless, small lookup table implies additional logical calculation and longer execution time. Indeed, an extreme situation is avoidance of lookup tables, which means the software must implement the cryptographic performing complex Galois Field operations that induces unbearable performance penalty.

There is a hardware alternative of small lookup table, which is large cache line configuration. Small lookup table will make one cache line hold more data items, and

large cache line can also hold more without making the items small. Large cache line will exert significant impact on the sample number needed by a successful attack using both methods. For example, for a cache with l words (four bytes) in one cache line, one can not deduce the lower log2l bits of the key bytes using first round attack in AES case [18]. So, with large cache line configuration, the sample number would increase and so would the counter resistance of the system.

Both of the above countermeasures utilize the fact that more data items of the lookup table could hide more information from the attacker. The drawback of the first is that performing Galois Field operations is quite time consuming. This is unbearable for applications demanding high throughput encryption and decryption, for example, Virtual Private Networks (VPNs). In fact, software implementation efficiency is a big concern when evaluating the goodness of a crypto algorithm. Using table lookups rather than Galois Field operations could significantly boost the performance of Rijndael, and it made a big reason why it can win out other competitors as AES. As for the large cache line, it is quite attractive due to its no need to modify the software and can also achieve high performance. However, a typical cache line size of a modern processor is 64 or 128 bytes, but the lookup table used by AES may be of size 1KB or 4KB. So, the confidential information can not be fully hidden and the attack is also feasible.

To avoid table lookup, some dedicated function units and newly defined instruction set extension could be added on the general purpose core which is originally used to augment the performance of cryptographic algorithms[32]. By doing so, the side-channel information leaked by table lookup is no longer present, so both types of attack are no longer valid. However, this approach requires considerable hardware cost and software modification and will be lack of flexibility in protecting varied cryptographic algorithms.

This thesis proposes an anti-attack scheme that is mainly hardware oriented with minor software modification. This scheme employs two main mechanisms to establish the secure cache architecture. First, we use the randomization mechanism of cache line permutation and virtual address random mapping, which minimize the hardware

cost to defend access-driven attacks. Second, the induced pseudo-miss generation (IPMG) mechanism is used to resist the time-driven attacks rather than roughly preload the whole table or shut down the cache, which can highly reduce the performance penalty at low hardware cost. Moreover, the cache is designed to work under two modes: normal mode and secure mode. Some function in secure mode is utilized to boost the performance in normal mode.

3.2 Countermeasure Aiming at Access-Driven Attacks

3.2.1 Threat Model

To define our threat model of the access-driven attacks, we make the following assumptions. First, the adversary could run one or multiple spy processes which are parallel to the crypto process. And the spy processes will continuously load a data table of the size of the cache, and some part will be evicted by the crypto process because of the cache sharing. By measuring the access time of its own data stored in or evicted from cache lines, the spy processes could reveal which cache lines are accessed by the crypto process. Generally, the spy processes must take advantage of the multi-threading capacity of certain processors, which allows multiple processes running quasi parallel on the same processor. However, reference [29] indicates that not only the hardware-assisted multi-threaded processor but also the single-threaded processor could fulfill this requirement. In fact, although the threads/processes are run serially, the OS manages to execute several programs in a quasi parallel way at a coarser resolution. Second, the cache access pattern of the particular operations in the crypto process must be key-dependent, for example, the table lookups required by the “substitute bytes” operation of AES. Therefore, the mapping relationship between cache line and table entry will leak enough information for the adversary to disclose the secret key. Third, the adversary can trigger the crypto process to input chosen plaintexts (or ciphertext) and get the corresponding ciphertexts(or plaintexts).

Considering the threat model, a practical way to defeat this kind of attacks is

obfuscating the mapping relationship between cache line and table entry, and a simple virtual address transform can be used to realize this perturbation. Originally, these schemes are employed to reduce cache conflict by making the cache access uniform [33], [34]. Intensive research effort has been devoted into searching an ideal randomization algorithm which could achieve performance improvement under different applications. The randomization method proposed by refernce [36] could achieve high defense effect, but the hardware cost is also quite high. As predicted in reference [24], the overhead would be 5.8% in the case the author described. However, for other configuration, this overhead could be much higher than one can bear, so the scalability is not that good.

In fact, various methods could effectively hide the mapping relationship between cache lines and table index. So we choose a simple randomization algorithm, and then present a dynamical index remapping mechanism, by which hardware overhead can be controlled at a low level.

3.2.2 The Permutation Logic

Generally, a lookup to the table of AES implementation is bound to access cache, and the accessible candidates are W cache lines in one cache set (if the cache is organized into S sets) according to the index, a part of virtual address of the table entry. The access pattern of cache lines are employed to deduce to the key byte involved lookup operation. Let the original index be named i, a particular lookup will access the W cache lines in the ith cache set. To change this mapping relationship, the index could be XORed with a random number rnd0, and the $(i \oplus rnd0)^{th}$ cache set will be accessed. This is called XOR-based remapping, which could make the attacker's job more difficult because S possible mapping ways should be exhaustively searched. However, if the r0 is fixed for the encryption process triggered by the attacker, this countermeasure is not sufficient because of the deterministic association of each cache set to its corresponding lookup table entry. So we provide a dynamical remapping method which repeatedly changes the random number by each interval of a program.

Because of the noises that come from both the encryption algorithm itself and

computer environment, practically, the attacker should let the encryption program be running more and more times so as to collect enough information. For original index or static remapping, a program that manipulates an identical entry of the lookup table should always access a fixed cache set despite how many times the program has been run. However, for dynamical remapping, a new random number will be used to XOR the index if the program interval is executed again, so that it could randomize the access pattern of the cache sets. Assuming that the interval of a program is executed by n times (n is big enough), the probability of accessing the j_{th} cache set can be calculated as follows.

$$P\{access\ j_{th}\ set\ |index = i\} = \begin{cases} 1, j = i \\ 0, j \neq i \end{cases}$$

$$P\{access\ j_{th}\ set\ |index = i \oplus rnd_0\} = \begin{cases} 1, j = i \oplus rnd_0 \\ 0, j \neq i \oplus rnd_0 \end{cases} \quad (3\text{-}1)$$

$$P\{access\ j_{th}\ set\ |index = i \oplus rnd_k\} = \frac{1}{S}, 0 \le k \le n$$

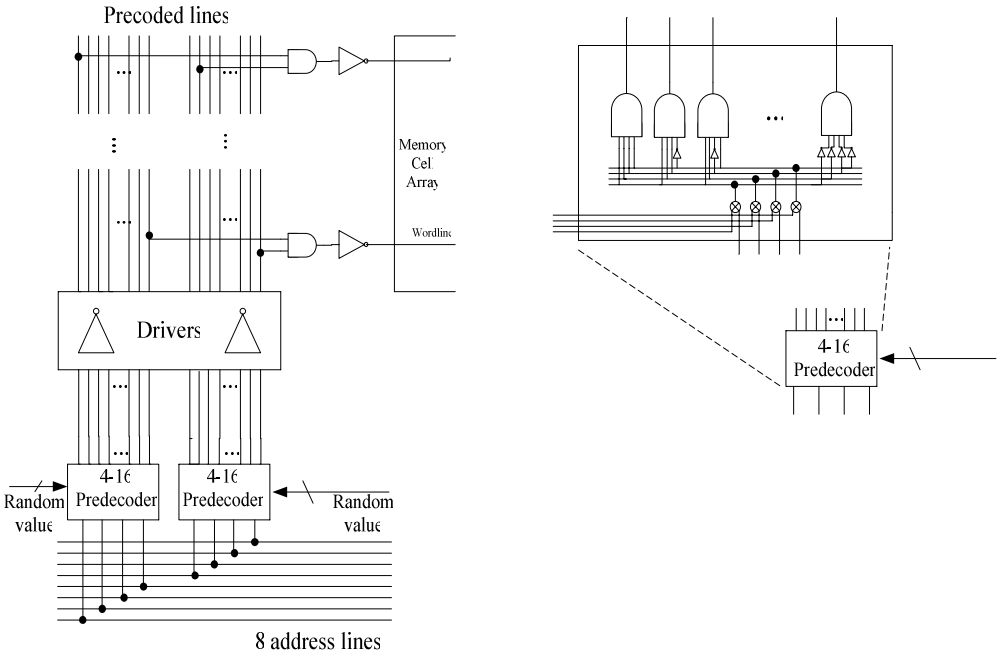

Figure 3. 1 Address line XOR remapping

As shown by (3-1), for a constant index i, each cache set will be accessed with probability 1/S under dynamical remapping if rnd_k has a uniform distribution. Since

each cache line has almost equal probability to be accessed when the data at same virtual address is loaded during a quantity of executions of a program interval, the access pattern of cache lines couldn't provide any significant information for access driven attacks.

3.2.3 The Task Oriented Random Permutation

To fulfill the requirement of dynamical index remapping, the target interval in a program to be assigned a temporal random number should be clearly defined. And it needs some ISA and software support. Program interval is a code segment that starts from an instruction *NewRandom* and ends at an instruction *RmvRandom*. And it should be noticed that the instructions in this interval could call a subroutine but could not induce an unreturnable jump that might evade executing *RmvRandom* instruction. And, as specialized instruction, *NewRandom* could adopt a new random number to remap the original index, but *RmvRandom* will release the cache from remapping mechanism. In fact, by using these two instructions, the length of a program interval could range from several instructions to a huge nesting function.

For AES implementation, the program interval could be the whole encryption function or several sub-rounds of the encryption. An architecturally visible register in cache is needed to store the random number, which is connected to a random source and could be update by using the *NewRandom* instruction. If the interruption or process switch occurs, the value of this register should be preserved in stack. Certainly, the random source of good quality is very important for the security of this dynamical remapping method as well as other randomization methods such as [24]. So, we assume that a true or pseudo random number generator (TRNG or PRNG) on chip could be available, but the detail design of these devices is out-of-scope for this paper.

3.3 Countermeasure Aiming at Timing-driven Attacks--IPMG

Induced Pseudo-Miss Generation (IPMG) is a dynamic miss rate regulation

mechanism. According to historic miss rate level, IPMG module could generate additional miss artificially. Therefore, the side channel information could be hidden in this additional misses and the vulnerability of system will be lowed.

3.3.1 Threat Model and Mathematical Analysis

To define our threat model of time-driven attacks, we make the following assumptions. First, the crypto implementation, such as AES, relies on table lookup operations which use key and inputs/outputs to determine the indices to lookup tables. Second, different data inputs may cause different numbers of cache misses, thus different execution times should be observed by the adversary. Ideally, the smaller number of misses is encountered by the table lookup operations, the shorter execution time are required by the AES, which could be formulated by (3-2). Third, a cache collision is defined as the situation involving two different memory accesses attempting to access the same memory location or different but very close memory locations that are stored in the same cache line. And the effect of cache collision is that although first access causes a cache miss, the second access inevitably induces a cache hit. So, a higher number of cache collisions ideally bring a smaller number of cache misses.

$$Access\ Time = (hit\ time + miss\ rate * miss\ penalty) * totale\ number\ of\ memory\ access \quad (3\text{-}2)$$

In time-driven attacks, the most effective way is to utilize correlation between the cache-collision in final round and the total execution time. Tiri *et al*. presents an analytical model to predict the sample number needed to successfully recover the AES key material. From equation (3-3)[20], one can see that $\mu D(k_T, l_T)$ are negatively relevant to total samples. Here, $\mu H(k_T, l_T)$ means the expected number of cache misses with k accesses to l cache lines when a cache hit is correctly estimated in final round table look up, and $\mu M(k_T, l_T)$ represents just the expected number of cache misses in a normal situation. Intuitively, with larger time difference between them, the more quantity of time information leakage could be acquired by attackers and simultaneously fewer samples are needed to recover the key.

$$N = \frac{2 \cdot Z_\alpha^2}{\frac{\mu_E^2}{\sigma_E^2} \cdot \frac{\mu_D^2(k_T, l_T)}{\sum_{t=1}^{T} \sigma_M^2(k_t, l_t)}} \qquad (3\text{-}3)$$

$$where\ \mu_D(k_T, l_T) = \mu_H(k_T, l_T) - \mu_M(k_T, l_T)$$

A coarse way to get rid of time distinction mentioned above is simply to disable cache when entering the security critical region or just pre-loading all data items into cache. The previous one can totally rule out the inner fluctuation caused by different inputs, and so the time difference $\mu D(k_T, l_T)$ is zero and the sample number needed is infinite large theoretically. However, the total execution time will rise up. The second method does not guarantee attack proof because the malicious process can be switched in during the execution of cryptographic algorithm [41, 42]. Usually, other operations or other processes in the system (not necessary attack process) are also responsible for eviction of pre-loaded data, so side channel information can still be present. Kong et al. [27] gives a refined way to ensure that the data could be re-loaded in case of unwanted replacements. The drawback is obvious: 1) each loading is whole scale (loading all data items), and it would occur unpredictably and might be triggered many times. 2) The bus would be jammed due to bunch of data pre-loading.

The non-zero $\mu D(k_T, l_T)$ means that different numbers of cache misses will be induced by the different input data selected from collision case and non-collision case respectively. So the best countermeasure is to balance the number of misses corresponding to each input data. According to (3-2), with fixed total access number, it requires to keep constant miss rate for each execution fed with different data. Therefore, we propose a new scheme to minimize the $\mu D(k_T, l_T)$ with low performance penalty, called Induced Pseudo-Miss Generation (IPMG). The basic idea behind IPMG is that the execution time variation is hidden by man-made penalty if the low miss rates of the corresponding executions of collision case are increased to a little higher but constant level. (3-2) is a simple equation giving the total memory access time for fetching data. So through controlling the miss rate in (3-2), one can adjust the access time of an application and so impact the execution time fluctuation indirectly.

3.3.2 Induced Pseudo-Miss Generation (IPMG)

As mentioned above, the miss rate for a single crypto process is not constant for a given key value. Figure 3.2 shows the distribution of miss rate got from software simulation. The plaintexts here are all of size of one single block that is 128 bit because normally attacker uses as short plaintext as possible to get rid of interference among different blocks. One can see that the miss rate concentrates on a small region near mean value. So according to historical miss rate, we can tune the miss rate to a relatively constant value without sacrificing much performance.

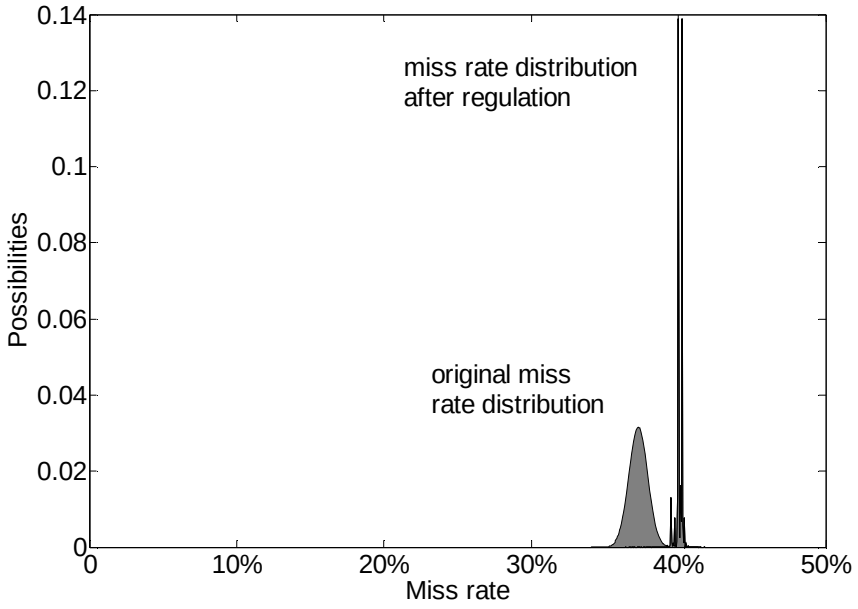

Figure 3. 2 Miss rate distribution comparison from software simulation.

An algorithm is developed to estimate the expected value of miss rate dynamically, which is shown in (3-4). The algorithm is bulk based, that means n encryptions are encapsulated in a bulk and the average miss rate of last bulk is used as threshold value to adjust the miss rate of cryptographic operation of current bulk. Thus, the miss rates $m_{in}+k$ (k=1,2,⋯,n) in bulk i is regulated by the threshold value $MR_{th}^{(i-1)}$, and for bulk i+1 the threshold value is $MR_{th}^{(i)}$, so on and so forth.

$$
\begin{aligned}
BULK\ i-1:\ MR_{th}^{(i-1)} &= \frac{1}{n}(m_{(i-1)n+1} + m_{(i-1)n+2} + \ldots + m_{(i-1)n+n-1} + m_{(i-1)n+n}) \\
BULK\ i:\quad MR_{th}^{(i)} &= \frac{1}{n}(m_{in+1} + m_{in+2} + \ldots + m_{in+n-1} + m_{in+n}) \\
BULK\ i+1:\ MR_{th}^{(i+1)} &= \frac{1}{n}(m_{(i+1)n+1} + m_{(i+1)n+2} + \ldots + m_{(i+1)n+n-1} + m_{(i+1)n+n})
\end{aligned}
\tag{3-4}
$$

By doing so, the time difference between concerned encryptions and ordinary ones would vanish because the induced pseudo misses hide these time distinctions. The software simulation is done using the method above. As Figure 3.x shows the miss rate distribution is concentrated at a small region that is the high bound of original distribution. This rather narrow rather indicate that the execution time is nearly constant, so the time difference is small enough to invalidate existing attack methods to detect the internal time difference caused by different plaintext inputs.

But in reality, the situation is not so optimistic due to other memory accesses except table lookups. These accesses are normally stack operations interleaved with cared table lookups, and this part of accesses has lower miss rate due to their space locality. And at the beginning of one encryption/decryption, the miss rate will oscillate with large magnitude due to small denominator in calculating the miss rate. This situation will be alleviated during later period of calculation when the total access number is larger. If at the beginning of the encryption there is an initial value of access number and miss number, this ostensible oscillation could be effectively avoided. In Figure 3. 3, the dashed line shows the miss rate trace with no zero initial value of miss number and access number and the solid line shows that with non-zero initial value.

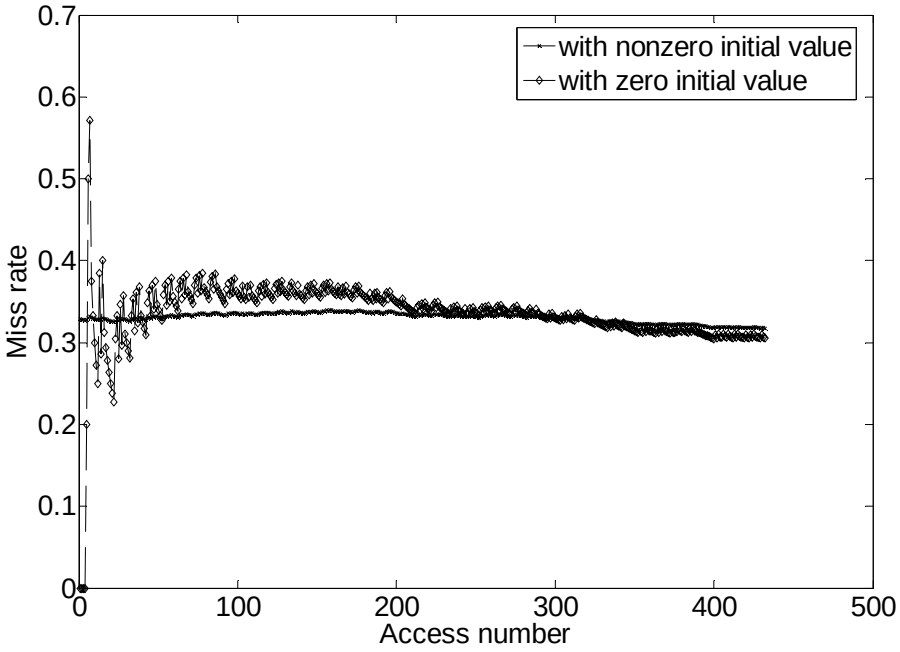

Figure 3. 3 Miss rate trace through one AES encryption.

Another problem is that this miss rate regulation is monotonically increasing, so the threshold of the miss rate will ultimately become very big compared with the miss rate without this regulation mechanism. In turn, this will impact the performance significantly. So, some kind of threshold limit should be set to get rid of ever increasing threshold. From the Figure 3. 2, one can intuitively get that the minimum stable threshold value should larger than the up-bound of the original miss rate distribution. Because only with this threshold value can the time difference information be hidden from attacks. One method is to record the maximum miss rate of the first p encryptions. After getting this value, it is used as a threshold limit and once the threshold reaches this value the updating of threshold is needless. Here, p should be negligible with respect to the total number needed for a successful attack.

At the same time, it should large enough to expect the up boundary with high precision. Theoretically, we could calculate the expected value of maximum miss rate among p encryptions. Firstly, we should denote that

$$MR_{\max} = \max(m_1, m_2, m_3, \ldots, m_p) \tag{3-5}$$

From (3-6), one could get the probability that MR_{max} is less than x, so the distribution function of MR_{max} is given by (3-7) and the corresponding probability

density function is shown in (3-8).

$$P(MR_{\max} < x) = P(m_1 < x, m_2 < x, ..., m_p < x) = P(m_1 < x)P(m_2 < x)...P(m_p < x) \quad (3\text{-}6)$$

$$F_{\max}(x) = [F(x)]^p \quad (3\text{-}7)$$

$$f_{\max}(x) = p[F(x)]^{p-1} f(x) \quad (3\text{-}8)$$

Using the software simulation result in Figure 3.2, the expected value of MR_{max} is given in Table 3-1 with various p values. One could see that several hundred encryptions could obtain the up bound of miss rate and this value is acceptably small considering that it is tens or hundreds of thousand for a successful attack normally.

Table 3-1 Expected Value of MR_{max} with Different p

P	**32**	**64**	**128**	**256**	**512**
E(MRmax)	0.7780	0.7823	0.7863	0.7899	0.7934

The bulk size will exert some influence on the converge speed of the threshold. The reason is quite apparent: with larger bulk size, the regulation period would be longer and the threshold update will be less frequent. But large bulk size is more reliable to estimate the expected value of the miss rate and less vulnerable to the impact of the singularity which has unusually high miss rate due to some kind of reason. Figure 3. 4 gives the change of miss rate threshold under different bulk size configuration. This is MATLAB simulation result of the proposed algorithm. There is two groups of random numbers: one with miss possibility of 0.33, and the other is 0.01. These data are interleaved with each other.

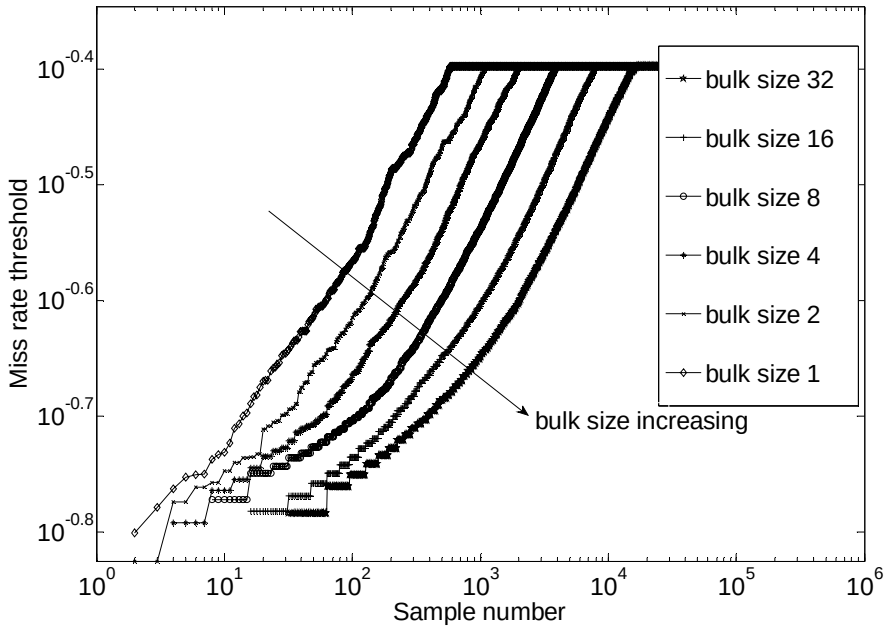

Figure 3. 4 Miss rate threshold steps upwards and is limited by a constant value of 0.4.

3.3.3 ISA Extension and Accurate Miss Rate Recording

The IPMG mechanism will present some performance penalty, but it is not necessary under normal situation. So a cache security mode is added to distinguish security critical circumstances from usual ones. To support this, some special instructions are added such that the cache could switch between these two modes. Moreover, in order to precisely monitor the miss rate of one single cryptographic operation, the critical region should be delimited using these instructions. So following instructions are designed to fulfill these requirement.

CacheProEnter: enter cache protection mode. This instruction indicates clearing the miss rate monitor and a new start to make real time miss rate recording. Usually, this instruction just precedes the security critical region and the cache is switched in secure mode when entering this region.

CacheProLeave: leave cache protection mode. This instruction is used to notify cache that it should make mode change when in secure mode. Normally, *CacheProEnter* and *CacheProLeave* are present in pair: they embrace a code segment

to mark that the cache should work in security mode when executing the instructions in that region. At the same time, for each protection mode entering, a new miss rate monitoring is started.

CacheConfig: configure some registers in cache. To enhance flexibility, some parameters are designed to be capable to be programmed by users.

3.4 Prefetch Mechanism

Adopting the IPMG scheme, we could diminish the variance of the execution time due to some artificial miss penalty. However, the performance will be impacted and the processor is under idle state when a pseudo miss is triggered. So, to take advantage of time cost on waiting, the dedicated prefetching mechanism is added that could get some performance gain.

The prefetch is triggered by the induced miss and the fetched data is placed in a buffer which is indexed in parallel with the main cache. This buffer is directly associative. The size of this buffer is four cache lines so that the hardware cost is small and the address comparing would be fast. The prefetch size is a normal cache line, and the location of prefetched data is just the next cache line after the line where induced miss is triggered. The performance gain of the prefetch is obtained by virtual of two factors: first, the prefetch operation needs no additional time consumption since it take advantage of CPU core and cache idle state; second, the prefetched line may not be in the main cache, and the miss penalty will be significantly reduced once this cache line is required by processor in the future.

In fact, another more important benefit of pre-fetch is that the induced miss penalty time could be self-aligned with the real penalty time. Using the method proposed in previous section, the penalty time should be controlled precisely to make it exactly like a real penalty. With prefetch operation, we can rely on the memory access operation to judge when the intentional idle state can be exit. That is to say, until the first data from memory arrive can the cache done signal be forwarded to processor. Consequently, from processor's point of view, it is exactly like a cache

miss. Therefore, the execution time variation will be flattened by IPMG much better through prefetching.

Since the security module proposed above is idle in normal mode, we could reuse part of the hardware to make some performance boosting. Intensive research work has been devoted in how to reduce the compulsary miss using hardware prefetch mechanism [38], [39], [40]. So, the prefetch buffer could be shared in normal mode to augment the performance of the system. Since pretch in normal mode is out of the range of discussion, the details are not discussed here.

3.5 AES Specific Instruction Set Extension

Normally, the instruction set extension is used to accelerate some complex computation by adding some dedicated function unit. For AES, the finite field operation can be realized using ISE, so the lookup table in software implementation is avoided. Since the root of information leakage is from the table lookups, the foundation of vulnerability no longer exists.

At the same time, this kind of realization of AES algorithm is still software based --the main control flow is done by software. We can use the flexibility of software to get rid some potential attack through adding some nop instructions. By doing so, some noises can be imported as mask information to hide some sensitive information from being stolen. As for this is not our focus in this thesis, the detailed design of function unit is not covered here and the integration of AES function unit is given in the next chapter.

Chapter 4 Implementation of Counter-measures on MIPS Processor

4.1 Structure of MIPS processor

The proposed countermeasure is implemented on MIPS processor[44], and the architecture of MIPS is introduced in this section. The MIPS CPU is one of the RISC CPUs, born out of a particularly fertile period of academic research and development. In this thesis, the MIPS 4KEc core is employed to add the security feature on it. Figure 4. 1 shows the block diagram of the processor. The processor core contains fundamental units of MIPS like execution unit and MDU, and other supporting units like MMU, cache and coprocessors as shown in the block diagram of Figure 4. 1. In our implementation, the dedicated crypto function units include an ECC encryption unit and an AES encryption unit which are supported through coprocessors.

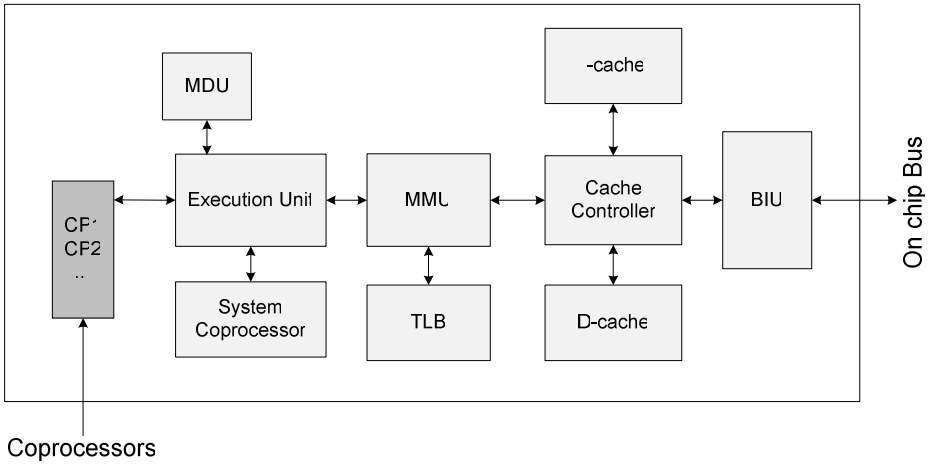

Figure 4. 1 Block Diagram of MIPS Processor

Table 4- 1 Component Features of MIPS processor

Parameters	Descriptions
I-Cache/D-Cache	2KB and 2-way set
MDU	Multiply-Divide Unit
MMU	Memory Management Unit
CP0	System Control Coprocessor
Execution unit	Including ALU and RegisterFile
Pipeline	5-stage

The processor pipeline structure is shown in Figure 4. 2. There are five stages in MIPS pipeline.

I stage: instruction fetch stage. In this stage, the PC address is translated and instruction cache is looked up. If both TLB and instruction cache are hit, the instruction could be fetched from cache and is ready for being decoded and executed in following stages.

E stage: instruction decode and execution stage. In the first half of clock cycle, the instruction is decoded and the corresponding control signals are generated. Also, the registers are accessed to get the operand ready for use. In the second half of clock cycle, some functions are executed and may extend to the following M stage.

M stage: memory access stage. For memory access instructions, the address is translated by TLB and the data cache is accessed. For other instructions, it performs idle operation.

A stage: align stage. The data obtained at M stage is post-processed in this stage. For example, the byte read instruction should extract the required byte from the word and then make zero or one extension according to the type of data.

W stage: write back stage. In this stage, the intermediate data is put into register file for the future use.

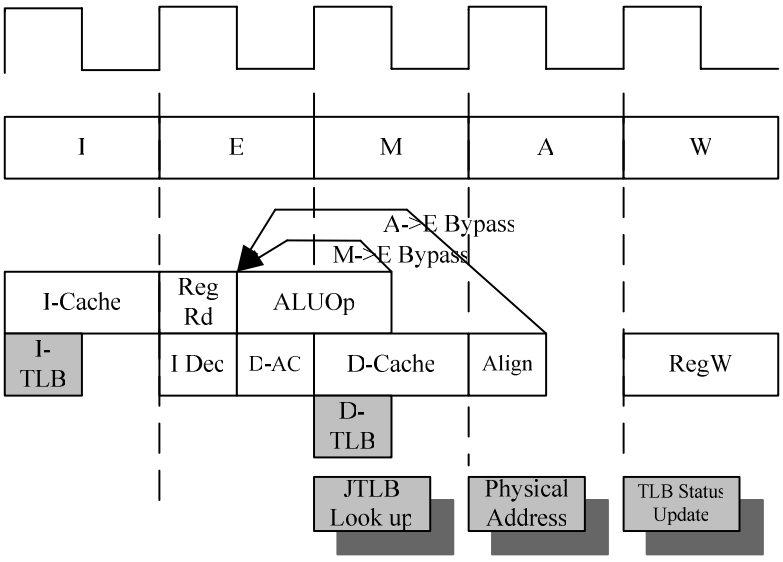

Figure 4. 2 Pipeline structure of MIPS Processor

4.2 Security Modules

4.2.1 Remapping Module

The major problem caused by the dynamical index remapping is the coherence between the internal and external of the program interval. The following part of an ended program interval will probably fetch the wrong data whose address has same tag but different index because of the mapping relationship being changed. To solve this problem, a modified cache design is presented as shown by Figure 4. 3.

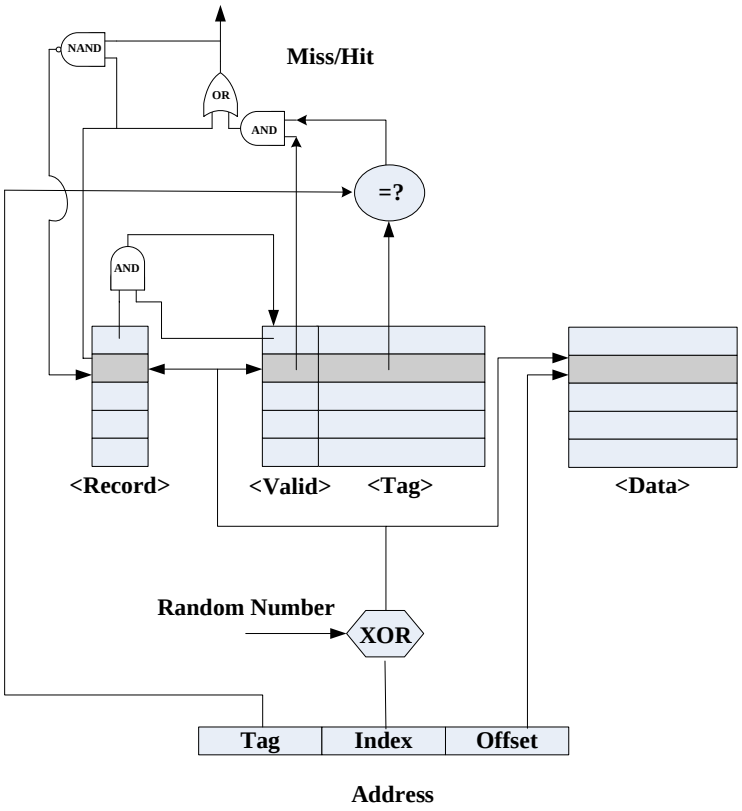

Figure 4. 3 Modified cache architecture for dynamical remapping

An *S*-bit register named Record is set in the cache, which could record the usage of a cache line during the program interval by one corresponding bit. When a program interval is entered at instruction *NewRandom*, all bits of Record register will be set to 1. Because of the OR gate, this operation can induce compulsive cache miss if the program interval accesses the memory address just at first time. In fact, the compulsive cache miss make the program avoid improperly using data already

deposited in cache. And, after the first-time memory access occurrence, the Record bit could be change to 0 by the NAND gate so as to memorize the corresponding cache line "polluted" by current program interval. When the program interval is ended at instruction *RmvRandom*, each bit in Record will be ANDed with the corresponding Valid bit to generate a result which could immediately replace the previous value of Valid bit. So, all "polluted" cache lines could be invalid, which prevents remained program from fetching wrong data in cache.

Since the proposed cache design is base on traditional architecture, the additional implementation cost could be limited at very low level. Assuming that the Address, Index, and Tag are d, n, t bits respectively, the extra combinational logics are n XOR gates, 2n AND gates, 1 NAND gate, 1 OR gate, 1 MUX and 1 DEMUX. The extra storage cost is the Record register of 2n bits, which could be laid out beside the memory cells of cache lines and implemented with the same memory cells. In fact, for a 2n lines cache array, one extra memory cell is needed by each cache line which has M memory cells, so the relative storage overhead is 1/M. As defined in [24], M is the total number of data, tag, flags and ECC bits, and take a 64KB cache with 64-bit address and 64-byte cache line size for example, $M \approx 64 \times 8 + 50 = 562$. So the relative storage overhead will be $1/562 = 0.17\%$. Contrarily, the overhead reported by [24] is 2.9%, and it will be doubled to 5.8% in the cache implementation the tag array and data array may be separated.

Two kinds of performance penalty might be induced by the dynamical index remapping. First, adding logic gates in cache will increase the path delay of circuit; second, if the program interval shares some data with the program outside the interval, dynamical index remapping could induce a number of additional cache misses. However, in fact, the delay time of the simple gates (XOR, OR) is rather small. And the miss rate of AES program using dynamical index remapping could be evaluated as follows. Figure 4. 4 shows the average miss rate of the AES lookup table operations based on different program interval, and these results are obtained from thousands of continued AES encryptions implemented by software simulation.

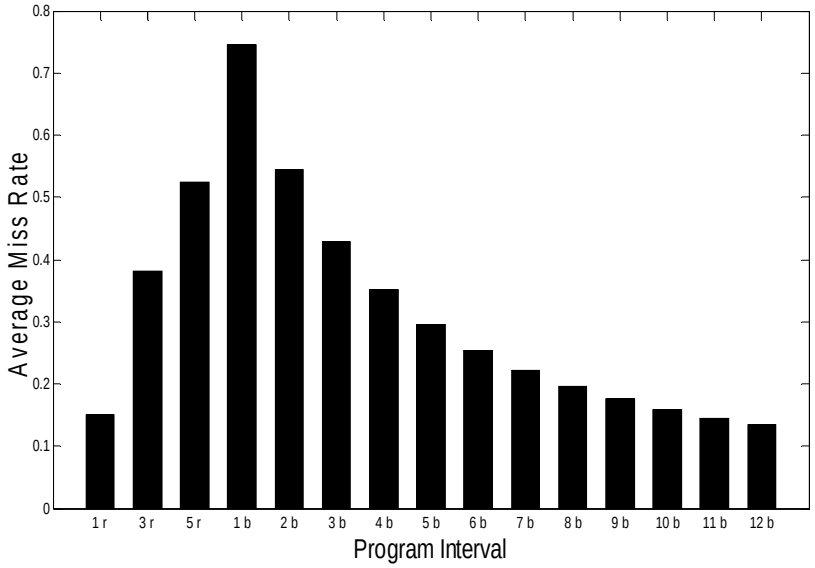

Figure 4. 4 Miss rate evaluation for dynamical index remapping

Let a program interval to encrypt 1 block of 128 bits plaintext be named 1b, and the program interval to execute 1 round of the 1 block encryption be named 1r. As shown in Figure 4.5, the average miss rate will increase when the program interval expands from 1r, 3r, 5r to 1b, but rapidly decrease to a very low level that is very close to the case without any remapping if the interval size exceeds the 10 block encryptions. So there are two recommended strategies to implement the dynamical index remapping. First one is to set the program interval just on the first or last round of the 1 block AES encryption, which could accurately protect the critical round targeted by varied attacks but induce neglectable performance penalty. Second one is to set a program interval including 10~20 block encryptions, which also minimizes the overhead but preserve the security level because no reported attacks could success with 10~20 times running of AES encryption.

4.2.2 IPMG module

Figure 4. 5 shows security and performance enhanced cache architecture. The only modification of original cache is the address decoder which has been described in Section 3, and other part can remain unchanged. The prefetch module is designed

as a parallel small fully associative cache which would compete for the bus interface port with main cache. So an arbiter should be added to tackle with the competition problem. The main advantage of this structure is that it needs little modification to original cache architecture and the added logic is very small.

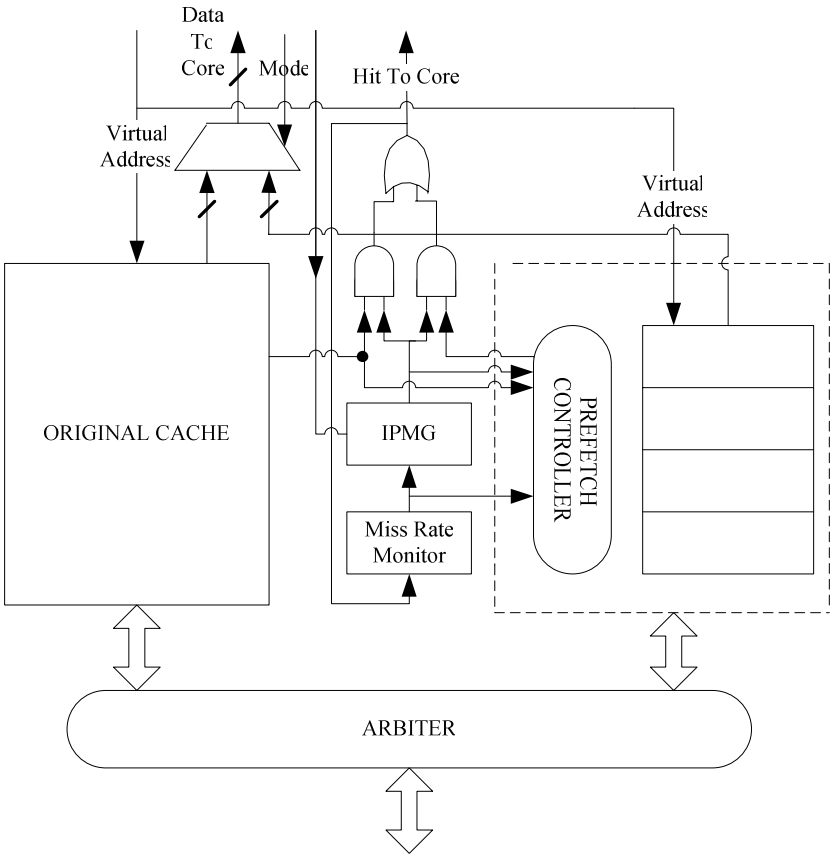

Figure 4. 5 Security Enhanced Cache Structure

The IPMG module is the core implementation of the proposed security mechanism. In real implementation, the miss rate is not directly generated and used. As shown in formula 4-1, division is a critical operation to judge whether an induced miss should be generated. Moreover, this signal should be generated in real time, so the pipelined or iterative logic could not be employed here. It would introduce long critical path to realize judgment logic (division and comparison arithmetic operation) and so exert some ill impact on the system performance.

$$threshold\ level > \frac{miss\ number}{memory\ access\ number} \tag{4-1}$$

To avoid this bottleneck, formula 4-1 can be reformulated into 4-2 equivalently.

In 4-2 the division operation is transformed into multiplication, and at the mean time the miss number term is moved to left hand. The expression in left hand side of 4-2 is just a simple multiply-subtract operation which could be realized by a multiplier-adder. Compared with division-comparison, multiplication-add is relatively easy for hardware implementation and its combinational path would be much shorter.

$$threshold\ level \times memory\ access\ number - miss\ number > 0 \tag{4-2}$$

The divider in Figure 4. 6 is a low cost iterative divider which is used to generate the miss rate at the end of security critical region. Since the miss rate should be memorized as mentioned in Section 4.1 to produce threshold value, the divider is unavoidable. Considering that it is not so urgent as the real-time induced miss generation logic, one can calculate this value in multi-cycles, like pipelined or iterative logic. Here the iterative logic is employed to reduce hardware cost. However, hazard will occur if the next *CacheProEnter* is executed before the completion of division. It is should guaranteed by software to get rid of this situation. For example, several instructions should be inserted between the *CacheProLeave* and *CacheProEnter*.

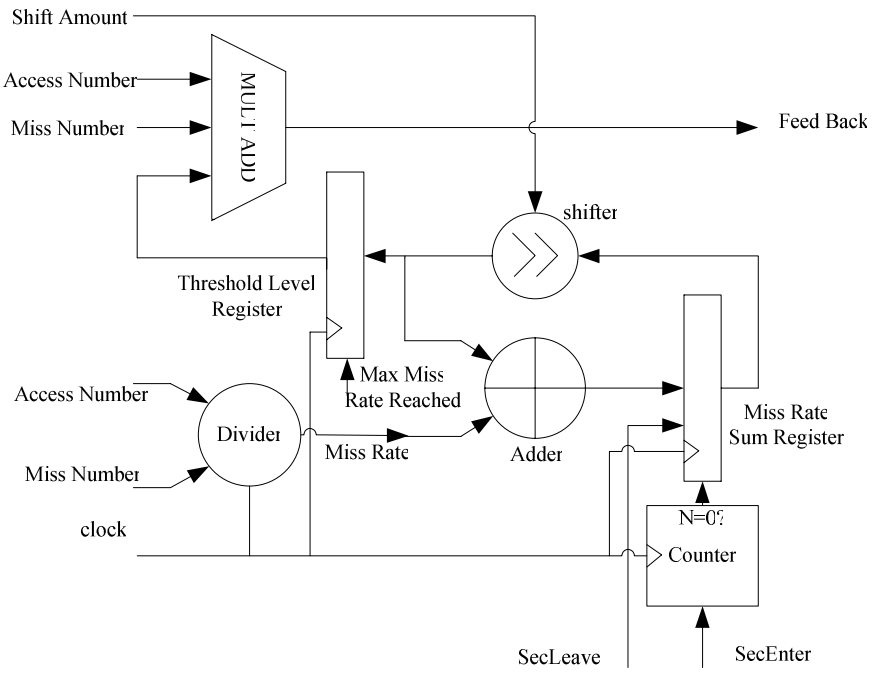

Figure 4. 6 Structure of IPMG Module

To calculate average value of miss rate, the result of sum register in Figure 4. 6 is divided by bulk size value. Normally, this bulk size is a power of 2, so simple shift logic could be used to get the mean value instead of complex division operation. To support flexibility, the bulk size is designed to be programmable and shift amount signal is connected to configure register.

According the above structure proposed, one could give the cost the IPMG with the help of synthesis tool. In our case, the miss rate has a precision of 16-bit and the iterative structure is used in realizing the divider. As a result, we get the total equivalent gates of IPMG module is 4k NAND gates. This cost is low enough is a normal cache design and it will not increasing with the cache size.

4.3.3 AES Extension on MIPS

The AES Encryption Unit performs the AES operations for 128bits key encryption or decryption. The 128 bits operation can be divided into four 32 bits operations and dedicated finite field arithmetic circuits can be used to realize the GF(2^8) inversion operation. However, as shown in Figure 4. 7, our AES unit employs 64 bits width data-path in the 2-stage pipeline to satisfy the timing constraints of a 5-stage MIPS pipeline using SIMD technology. It can accomplish each round operation of AES encryption or decryption in 3 cycles including twice 64 bits AES computation.

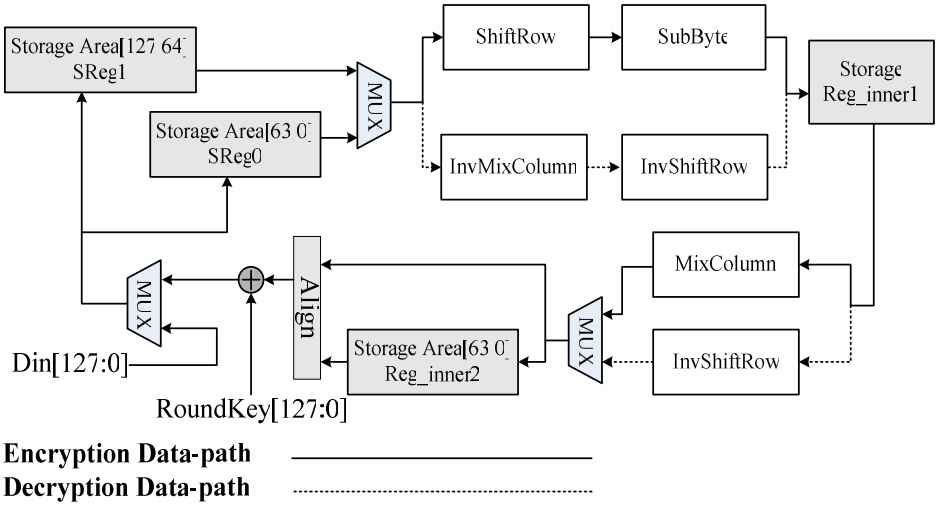

Figure 4. 7 The structure of the proposed AES unit

4.3 Summary

The detailed implementation issues are given in this chapter. The architecture of MIPS processor and pipeline structure are introduced. The following two sections presents the hardware structure of proposed security mechanisms in Chapter 3. At last, AES specific instruction extension is given.

Chapter 5 Verification and Evaluation

5.1 Test Environment

The verification platform is a small SoC and the architecture is given by Figure 5. 1. The AHB bus is used to connect processor and memories, and UART is used as the communication port to download the test software from PC and transfer test result to PC. The processor here is the MIPS core mentioned in the chapter 4 and the on-chip rom is used as the boot rom which can retrieve the test software from PC through UART.

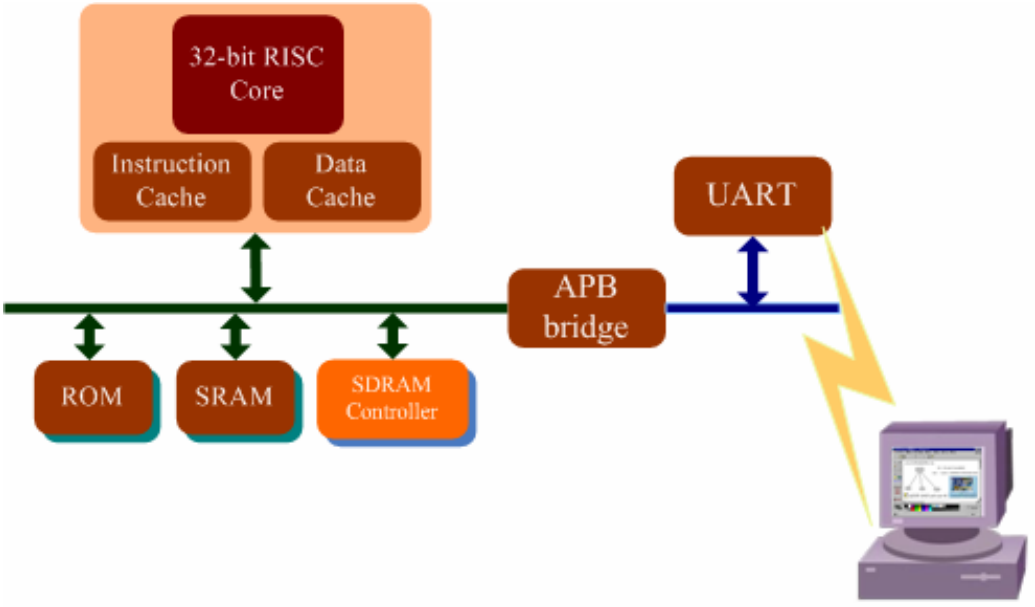

Figure 5. 1 Platform for test

To show the effectiveness of the proposed countermeasure against time-driven attacks, the above four attacks are mounted on both systems: one with cache protection mechanism and the other one not. The design is prototyped on a FPGA development board which is Strix II: EP2S180F1020C3. The OpenSSL implementation of AES is extracted and modified to a standalone program. The compiler used to generate binary file is sde-6.06 shipped by MIPS Technologies. The sample number is set to 2,000,000 for all three kinds of attacks.

Figure 5. 2 FPGA and PC test environment

5.2 Design of test software

The whole test flow is designed as shown in Figure 5. 3. On system reset, the MIPS processor executes instruction from boot ROM which configures the hardware components like UART initialization and SDRAM configuration. And then download test software binary file from PC to SDRAM at predetermined location. Finally it branches to the main test entry.

The main test body is a loop depending on the test sample number needed. As shown on the right of Figure 5. 3, the plaintext is got randomly with the help of a pseudo-random generator in MIPS processor. Before encryption and decryption, the time stamp is got to record the start of cared section, and then the encryption and decryption are performed. For protected mode, *CacheProLeave* and *CacheProEnter* are executed before and after the encryption/decryption which is shown in dashed boxes. The test results are transferred to PC through UART which including

plaintext, cyphertext and measured time data.

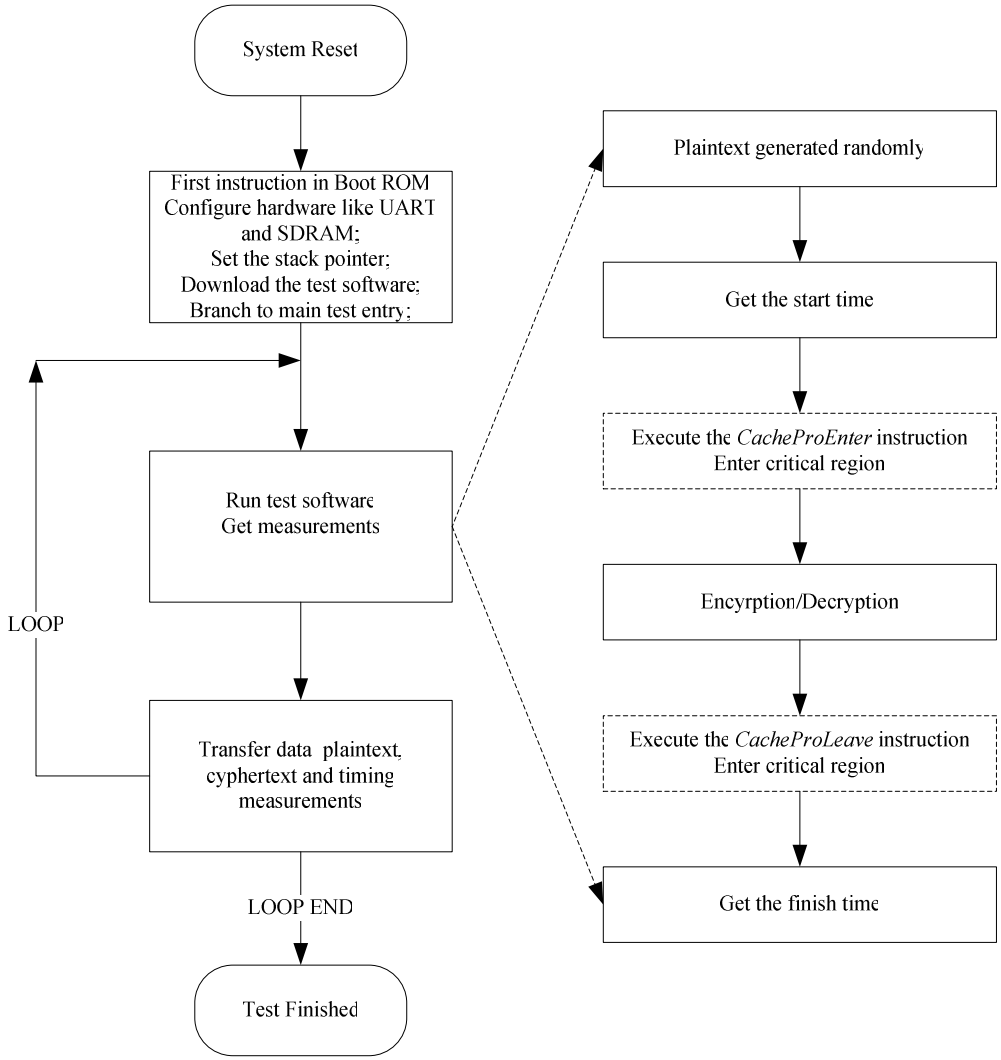

Figure 5. 3 Test software execution flow

Pseudo code fragment in Figure 5. 4 shows how to get the AES encryption execution time and how the cache security mechanism is controlled by *CacheProLeave/CacheProEnter* instruction pair. The embedded assembler codes are the extended instructions upon MIPS instruction set: the machine code of *CacheProLeave* and *CacheProEnter* is 0x78000000 and 0x7c000000 respectively. Sample test programs are given here.

```
1       exec_time = timestamp()
2       __asm__("  worc  0x78000000")
3       AES(plaintext  ciphetext  &KeyExpand  AESSymKey)
4       __asm__("  worc 0x7c000000")
5       exec_time = exec_time - timestamp()
```

Figure 5. 4 Pseudo-code for security critical region delimitation

5.3 Effectiveness of the Proposed Solutions

The effectiveness should be verified under following four cases. Figures will be given to illustrate the effectiveness. There are several time-driven cache attack methods proposed so far. In this section, some of these attacks are used to show the effectiveness of our countermeasures. Three methods presented in [17] mainly use cache collision of the first round and final round table look-ups. All these three attacks employ the fact that the average execution time of random samples is longer than that of samples with cache collision at one cache line. Tiri *et al.* [20] employs correlation analysis between the measured execution time and estimated cache collision, and an analytical model is also given to predict the sample number needed for a successful attack. All these attack methods need to sample timing data from target machine, but they employ different analyzing techniques.

As illustrated in Figure 5. 5 and Figure 5. 6, the execution time versus cache collision in the final round is plotted for both systems: original cache and security enhanced cache. In general, cache collision can avoid memory access to lower memory system and thus could get some performance benefit. The more cache collisions occur in the final round, the less time cost by encryption or decryption operation. In fact, it is the foundation of time driven cache attack. Figure 5. 5 shows the linear relationship between the number of cache collisions and encryption time. However, this is not the case anymore in the security enhanced cache architecture. The adaptive modulation of cache misses makes the execution time independent to how many cache collisions occurring since the induced misses influence the total execution time. Figure 5. 6 gives what induced misses impact on the relation between

collision number and execution time. With the proposed secure mechanism, more collisions can not result in shorter execution time, which cracks the foundation of time drive cache attacks. In other words, statistically, the linear relationship between execution time and collision number is undermined by the protection mechanism.

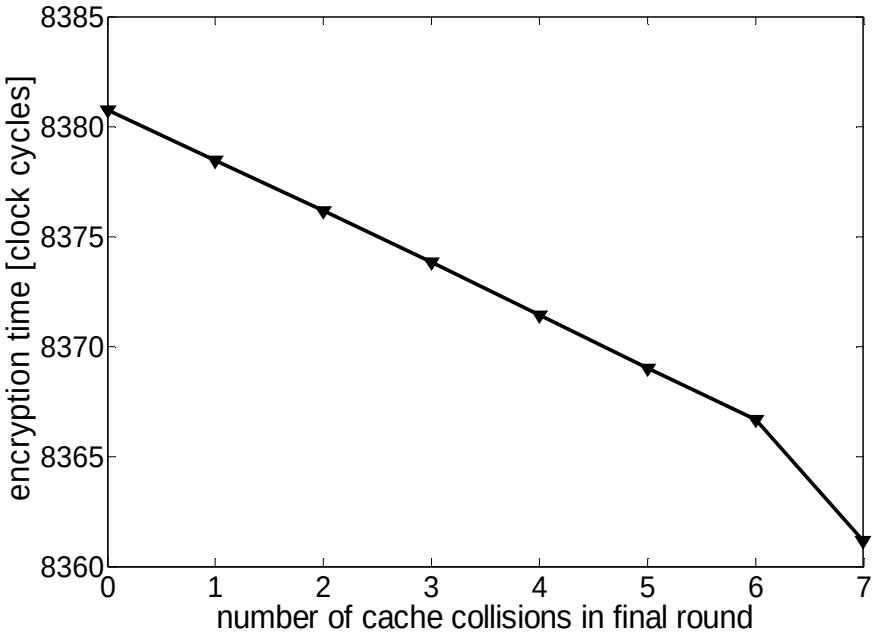

Figure 5. 5 Linear relationship without cache protection

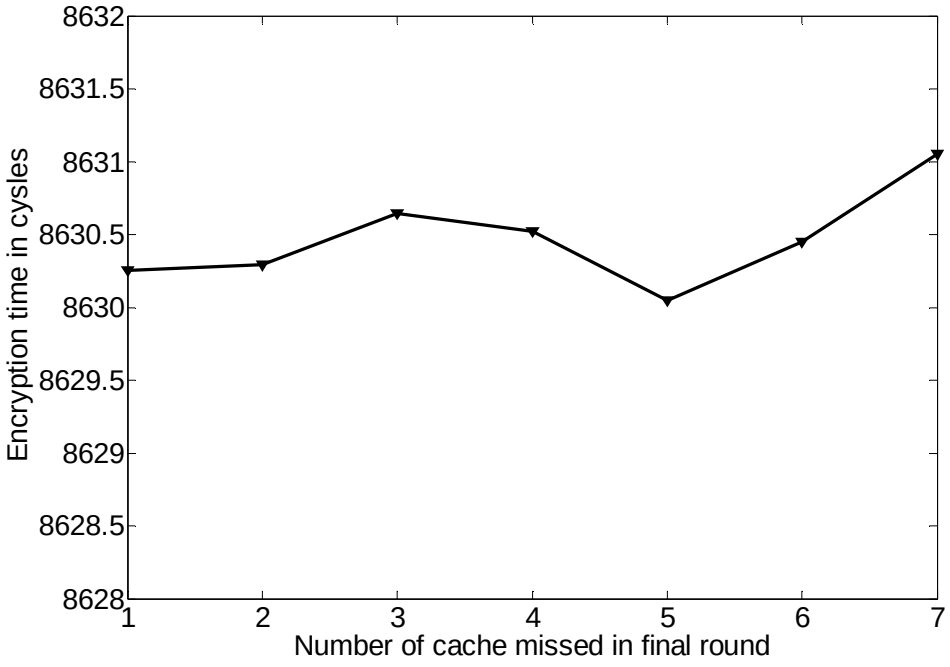

Figure 5. 6 Non-linear relationship with cache protection

5.3.1 Access driven attack

The random permutation of the virtual address has fully shade the foundation of the access driven attack since the mapping relationship between cache lines and table entries are obfuscated. Software obfuscating has been pointed to be unreliable because this mechanism is fixed obfuscation and the adversary can use some dedicated method to find out this mapping relationship. Our hardware obfuscating is similar to the random permutation of [14] with relatively low hardware complexity. [14] realizes a fully dynamic random permutation which means that every block in the main memory can be mapped into arbitrary cache line independently. In fact, our dynamical index remapping mechanism will also make different memory blocks have an equal probability to be mapped into each cache line, but the hardware complexity can be diminished and so is the cost.

5.3.2 First round attack

Using first round attack, one can get the difference between two key bytes since samples with $\langle p_i \oplus k_i \rangle = \langle p_j \oplus k_j \rangle$, or after rearranging, $\langle p_i \oplus p_j \rangle = \langle k_i \oplus k_j \rangle$ will exhibit lower execution time. Since the cache collision happens if plaintexts satisfying the equation $\langle p_i \oplus p_j \rangle = \langle k_i \oplus k_j \rangle$, the average execution time will be lower compared with fully random plaintexts. From Figure 5. 7, one could find a negative pulse at the location 34, which means the corresponding key byte difference with lower two bits omitted is 34. To get lower bits of key bytes, further analysis should be applied, such as second round attack.

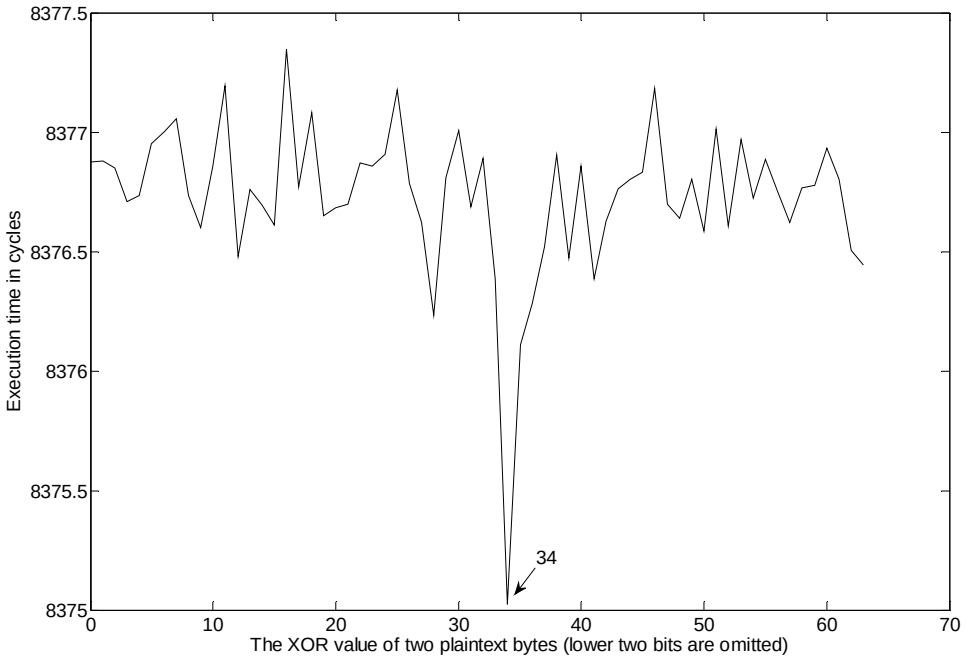

Figure 5. 7 First round attack result without cache protection

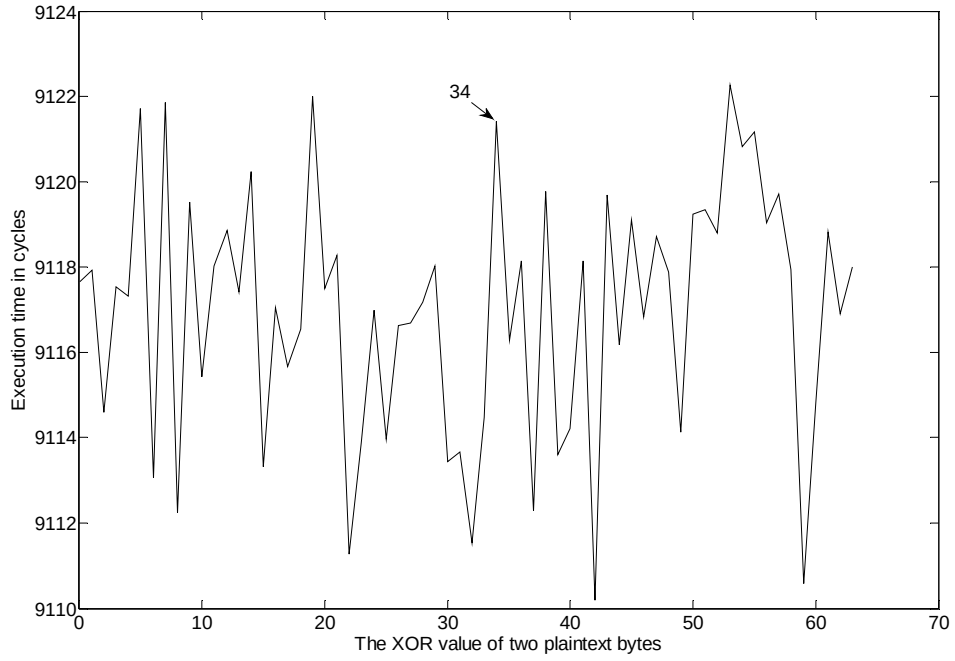

Figure 5. 8 First round attack result with cache protection

With the protection mechanism proposed in this thesis, the system could effectively be protected from leaking this information. From Figure 5. 8, we could see that the peaks in Figure 5. 7 are hidden in the random noises, and so the attacker can

not obtain the key information easily.

5.3.3 Final round attack

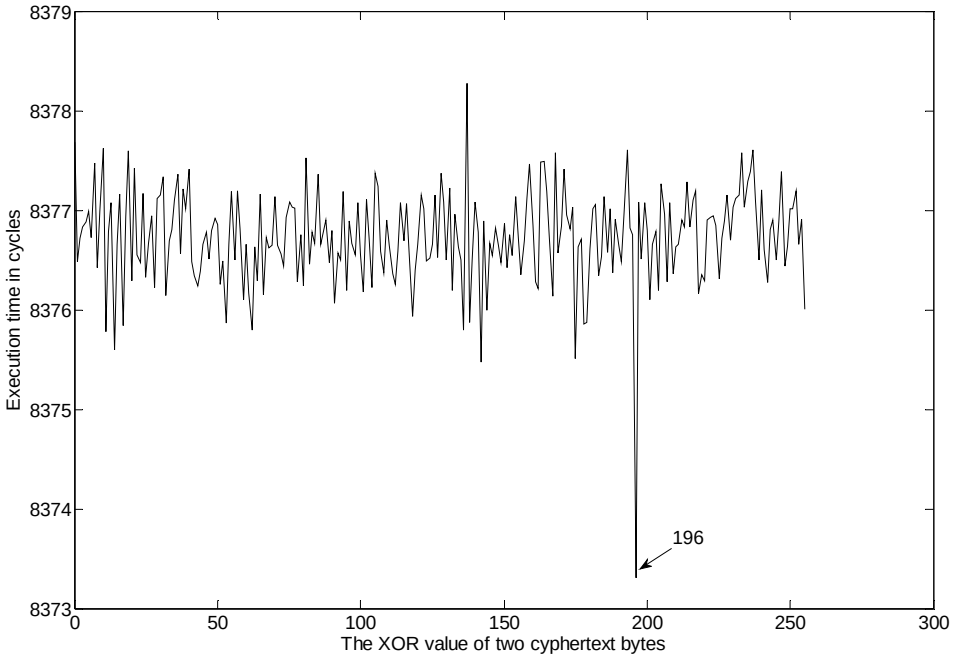

Figure 5. 9 Final round attack result without cache protection

The final round attacks are popularized because of their high efficiency and easiness to implement when the adversary could access ciphertext. Figure 5. 9 and Figure 5. 10 show the attack result when the IPMG is not implemented on data cache using two types of final round attack. First is the normal final round attack. By classifying the collected samples into 256 different sets which correspond to the results of two particular ciphertext bytes XOR each other, the mean execution time of more than 7 thousand samples corresponding to each set has been illustrated in Figure 5. 9. Since $c_i \oplus c_j = k_i \oplus k_j$ (c_i, c_j are ciphertext bytes, and k_i, k_j are key bytes), the negative pulse indicates that the difference between the key bytes can be successfully guessed out through this attack.

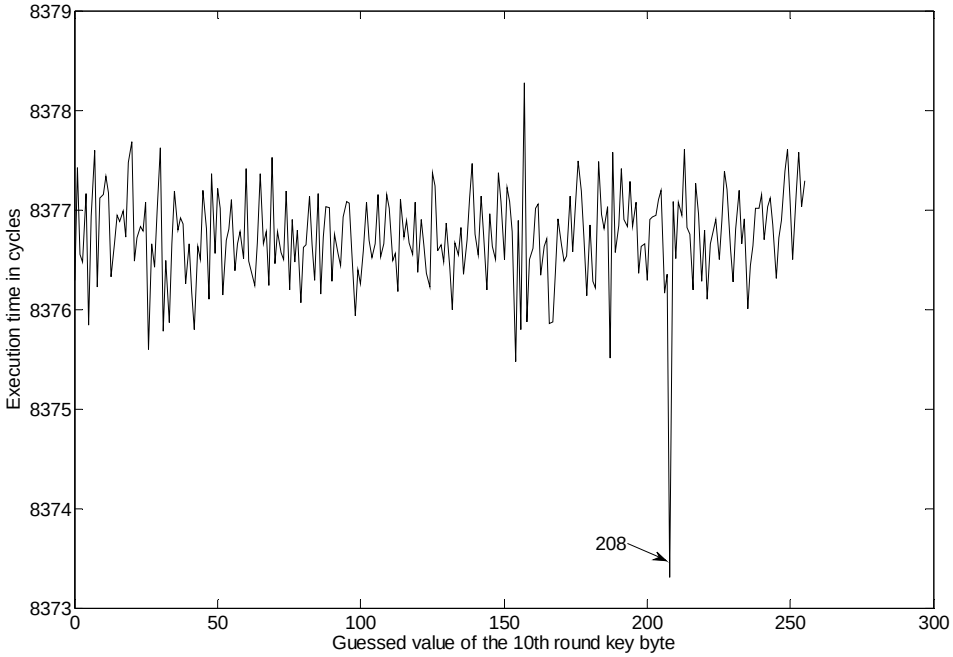

Figure 5. 10 Expanded final round attack result without cache protection

Second is the expanded final round attack, which, in fact, is more effective to predict the correct difference of key bytes with the respect to the samples needed for a successful attack. Here, for simplicity, one key byte is assumed to be already discovered, thus, we could directly obtain each of the remained key bytes since the mean execution time of the samples set corresponding to the right difference between the attacked key byte and discovered one indeed demonstrates a negative pulse, as shown in Figure 5. 10.

For comparison, the Figure 5. 11 and Figure 5. 12 show the attack results when the IPMG mechanism is activated in the system. Under the same sample number, the peaks in Figure 5. 9 and Figure 5. 10 are no longer present. One can see the location where peaks exhibit are now have normal values which can not be differentiated from other values for the attackers.

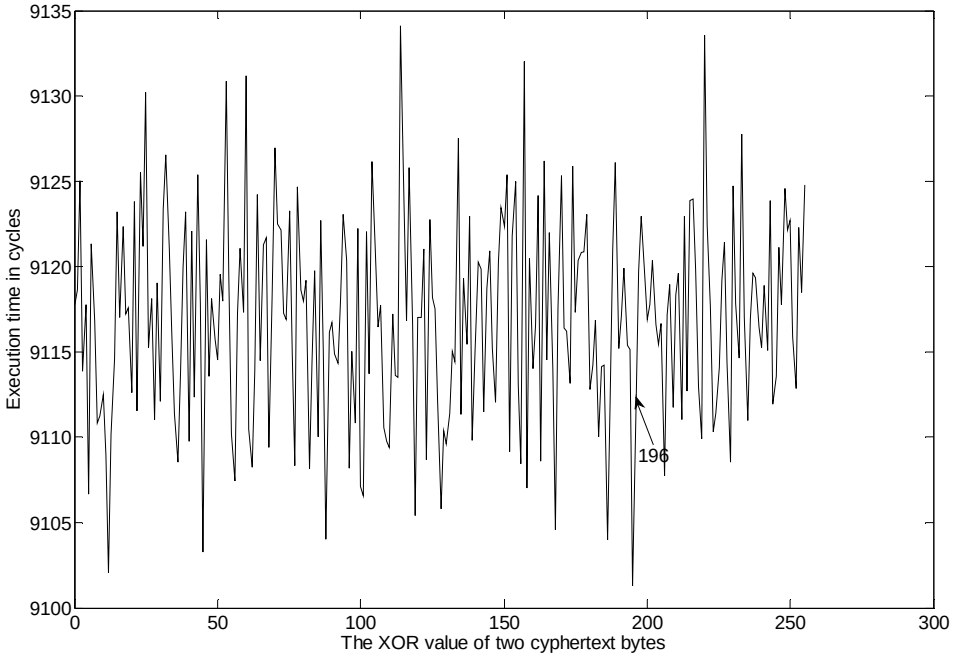

Figure 5. 11 Final round attack result without cache protection

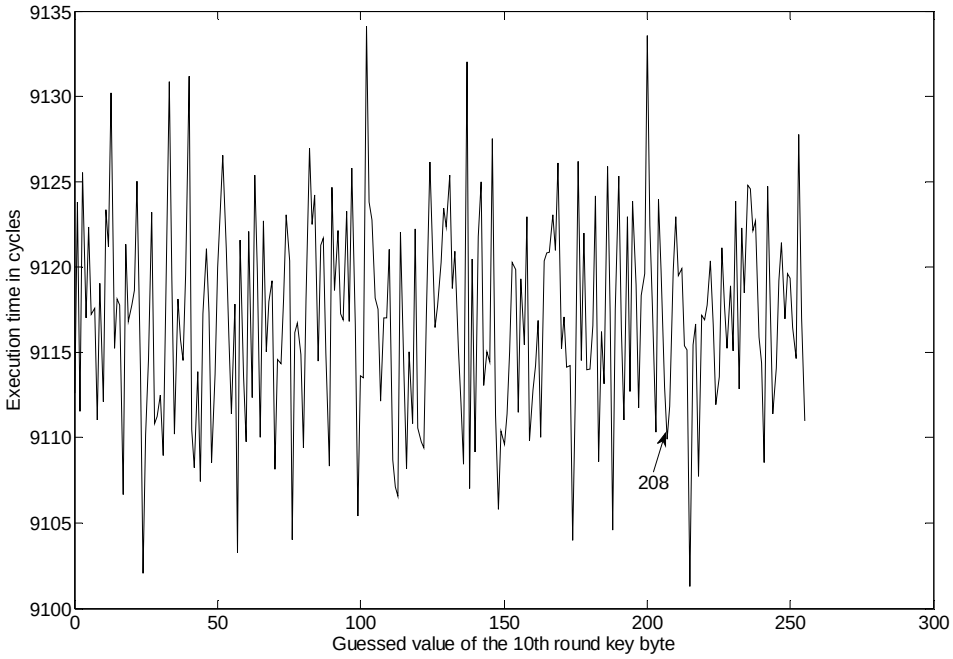

Figure 5. 12 Expanded final round attack result without cache protection

5.3.4 Correlation analysis

The correlation attack for final round can also successfully get the correct key byte through correlation analysis between actual measurements and estimations.

Given the ciphertext and assumed key bytes, we could give an estimation of a cache collision or not, which is marked 0 or 1. And the correlation coefficient between the estimations and the measurements of execution time can be calculated as [20] suggested. Obviously, the correctly guessed key byte that results in a high absolute value of correlation coefficient could be pointed out in Figure 5. 13. Like above cases, the undistinguishable random noises are observed in Figure 5. 14 instead of the peak in Figure 5. 13, thus the key information could not be leaked.

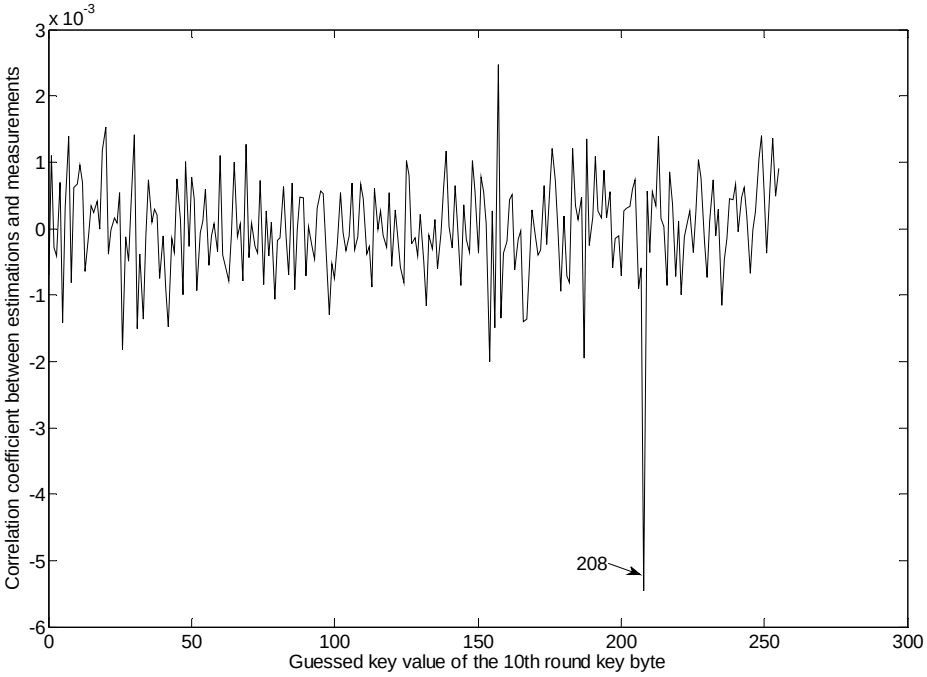

Figure 5. 13 Correlation attack result without cache protection

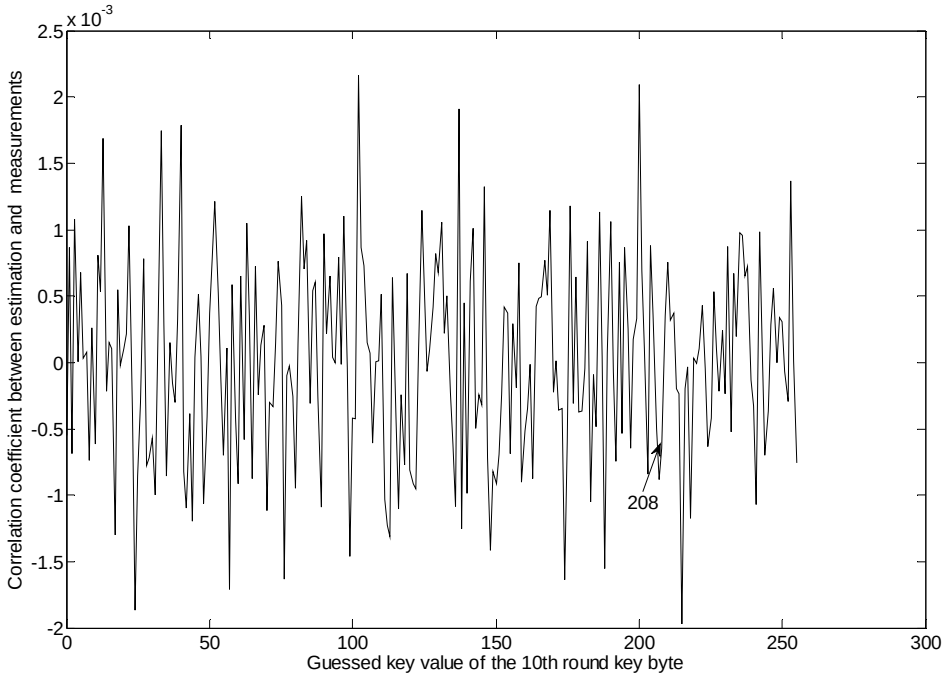

Figure 5. 14 Correlation attack result with cache protection

5.4 Evaluation of Performance Loss and Area Cost

To show the advantage of countermeasures proposed in this dissertation, comparison of performance is given between our solution and other's in reference. We could see that the performance penalty of our measures is quite low.

Since the induced miss will prolong the memory access time which is intended to be a cache hit, the total execution time will be larger due to this effect. Figure 5. 15 gives the performance under different cache configuration. From the figure, we could see that the performance loss is about 3%-4% with cache size of 8k or 16k and 8%-9% with cache size of 32k which is quite low compared with software mitigation.

Moreover, the hardware cost of additional security module is also very small. As for the access-driven attack, the cost area is about 0.17% as analyzed in previous chapter. The hardware cost for IPMG is also very small and it is not scaled with the cache size. From the synthesis result, it costs about 4K logic gates. This is quite small with respect to today's scaled of integrated circuits.

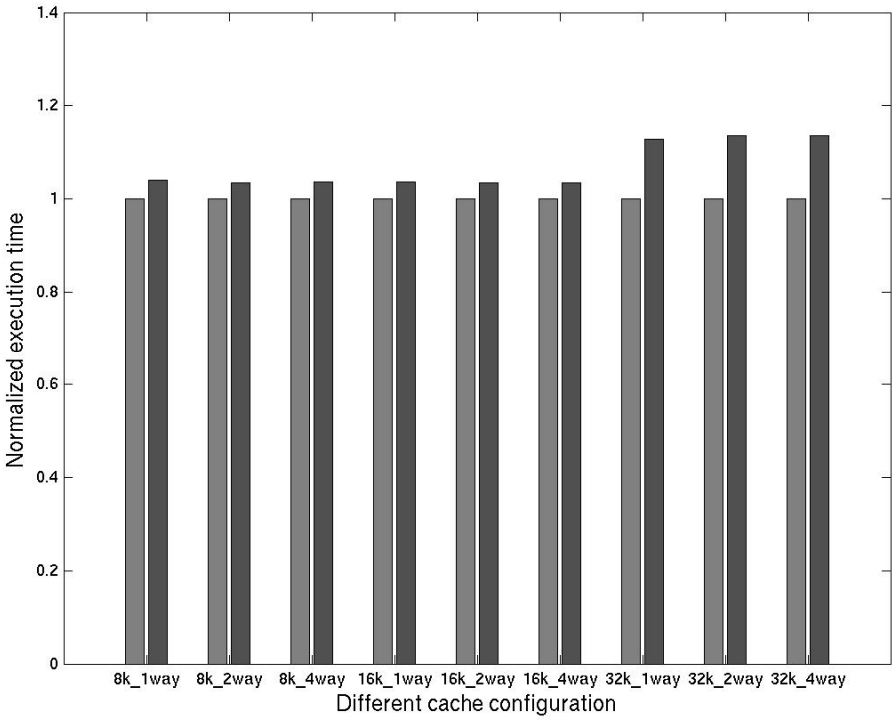

Figure 5. 15 Performance comparison under different cache configuration

5.5 ISE Tape out Result

To evaluate the proposed security processor with ISE extension, we implement it based on SMIC 0.13um standard CMOS Technology. Figure 5. 16 is the die photo of our design, and the features of our chip are described in Table 5. 1. The comparisons with the previous work are presented in Table 4. In Table 4, it is shown that the proposed security processor only needs 381 clock cycles to complete one AES-128 encryption, which is much faster than the software algorithm presented by [45]. On the other hand, it also saves considerable area cost for AES enhancement, compared with [46] that accelerates AES by on-chip lookup tables and achieves same performance level in a 32-bit processor as ours.

Table 5. 1 Features of the security processor chip

Features	Descriptions
Chip Size	7.56 mm^2 (2.75mm x 2.75mm)
CPU Core Size(Logic)	1.90 mm^2 (1.01mm x 1.88mm approximately)
PAD Number	80
Clock Frequency	125M (Worst Condition)
Power consumption	89mW @100MHz,1.2V voltage supply

Table 5. 2 Test Result of MIPS Processor

Ref	Technology	Area	Cycles
[45]	Efficient software implement based on ARM9	-	1384
[46]	32bit RISC processor + on-chip lookup tables	73K gates	315
Ours	ISE based on MIPS 4KE Family processor	15K gates	381

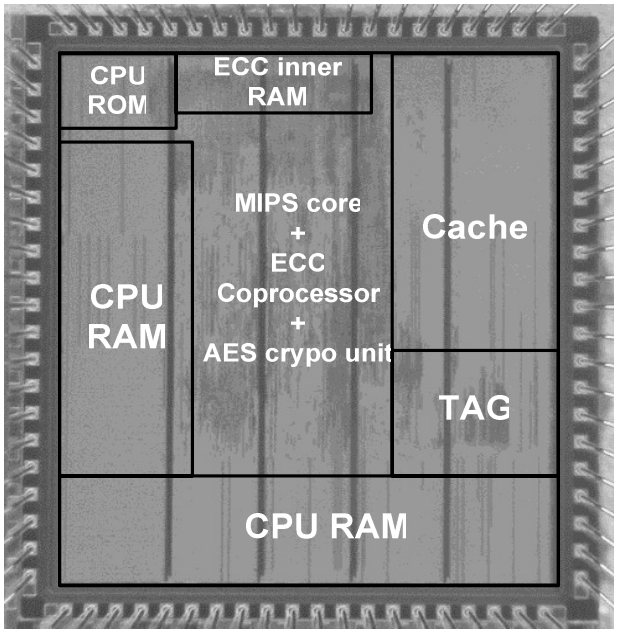

Figure 5. 16 Die photo of MIPS processor

5.6 Summary

In this chapter, the test environment is introduced at first which includes the system architecture and test tools. Then the test software design flow is given. The comparison between protected system and unprotected system are made which shows the effectiveness of our proposed hardware modification strategy. Also, the performance and cost are evaluated for our mechanism. At last, the tape out result is given for AES instruction extension MIPS processor.

Chapter 6 Conclusion

With the popularity of cloud computing, the virtualization technology which is the core in CC is based on the hardware sharing among different clients. Micro-architectural attacks using the underlying hardware sharing as the side channel to get sensitive information are proved to be practical. The cache based timing attacks have been attracting so much research attention recent years. So many papers have proposed various techniques to mount a successful attack, and miscellaneous anti-attack methods are provided to protect our system. But, most of them are software mitigation based, but the scheme described in this paper is mainly hardware oriented with little software support.

The cache design in past research is mainly focusing on how to increase performance, lower the hardware cost and reduce the power consumption in mobile application. The security is not given enough consideration although it is desperately tackled with. The security dimension of cache design can be roughly categorized into two domains: one is how to get rid of side channel information leakage including both timing and power information, and another is how to seal the system by sophisticated design of the cache which usually serves a security border. Moreover, with the increasingly intensive research on grid computing and multi-core architecture, the security problem will certainly remain a scratching one.

This thesis is concentrating on data cache, and some hardware modification has been proposed to protect the system from CBTA. First, the XOR address remapping is used to thwart the access driven cache attack. Second, the IPMG mechanism can effectively counter time-driven attack which is the first hardware strategy. Also, AES specific instruction extension can also avoid CBTA due to its absence of lookup table operations. At the mean time, the performance can be boosted for AES encryption and decryption.

As for other micro-architectural attacks, the instruction cache has similar problem though the attack technique is quite different. How to effectively counter these kinds

of attack will be part of our future work. Also, the similar problem in other processor components like branch predication, shared function units is present. The future work would be focused on what kind of attack would be proposed and how to thwart the attack through hardware improvements. More aggressively, some new architectures should be proposed to be applied in shared environment computing to guarantee security.

References

[1] William Stallings. Cryptography and Network Security. Principles and Practices，4th Edition. Prentice Hall, 2005. ISBN: 0131873164

[2] Paul Kocher, Joshua Jae, Benjamin Jun. “Introduction to Differential Power Analysis and Related Attacks”, 1998. This paper is available at http://www.cryptography.com/dpa/technical/index.html.

[3] P. Kocher. Timing Attacks on Implementation of Diffie-Hellman, RSA, DSS and Other Systems, Advances on Cryptology: Proceeding of CRYPTO’96, Springer-Verlag, August 1996, pp. 104-113.

[4] Top Threats to Cloud Computing V1.0, prepared by the Cloud Security Alliance, March 2010. http://www.cloudsecurityallicance.org/guidance/csaguide.v.1.0.pdf

[5] Security Guidance for Critical Areas of Focus in Cloud Computing V2.1, prepared by the Cloud Security Alliance , December 2009 http://www.cloudsecurityallicance.org/guidance/csaguide.v2.1.pdf

[6] O. Aciicmez. Yet another micro-architectural attack: Exploiting I-cache. *Proceedings of the 2007 ACM Workshop on Computer Security Architecture*, pp. 11-18, ACM Press, 2007.

[7] O. Aciicmez and J.-P. Seifert. Cheap hardware parallelism implies cheap security. *4th Workshop on Fault Diagnosis and Tolerance in Cryptography*—FDTC 2007, pp. 80-91, IEEE Computer Society, 2007.

[8] John Hennessy and David A. Patterson. Computer Architecture: A Quantitative Approach, 2nd Edition. Morgan Kaufmann Pub., 1995. ISBN: 1-55860-329-8.

[9] National Bureau of Standards, Data Encryption Standard (DES), U.S. Dept. of Commerce, FIPS pub.46, January 1977.

[10] Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[11] Openssl: the open-source toolkit for ssl / tls. Available online at http://www.openssl.org.

[12] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104-113. Springer, Heidelberg (1996).

[13] Page, D.: Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel. Technical Report CSTR-02-003, Department of Computer Science, University of Bristol, June 2002.

[14] Y. Tsunoo, E. Tsujihara, K. Minematsu, and H. Miyauchi. Cryptanalysis of Block Ciphers Implemented on Computers with Cache. ISITA 2002,2002.

[15] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. *Cryptographic Hardware and Embedded System* – CHES 2003, C.D. Walter, C. K. Koc, and C. Paar, editors, pages 62-76, Springer-Verlag, LNCS Nr. 2779, 2003.

[16] O. Acricmez, W. Schindler, and C. K. Koc. Cache Based Remote Timing Attack on the AES. Topics in Cryptology – CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007.

[17] J. Bonneau, I. Mironov. Cache-Collision Timing Attacks against AES. *Cryptographic Hardware and Embedded System* – CHES 2006, Springer, LNCS, Berlin 2006.

[18] D. A. Osvic, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. Topics in Cryptology – CT-RSA 2006, T*he Cryptographers' Track at RSA Conference 2006*, D. Pointcheval, editor, page 1-20, Springer-Verlag, LNCS Nr. 3860, 2006.

[19] M. Neve, J.-P. Seifert. Advances on Access-driven Cache Attacks on AES. Selected Areas of Cryptography – SAC'06.

[20] K. Tiri, O. Aciicmez. M. Neve, and F. Andersen. An Analytical Model for Time-Driven Cache Attacks. *14th International Workshop on Fast Software Encryption workshop (FSE)* 2007, A. Biryukov, editor, pages 399-13, Springer, LNCS 4593, 2007.

[21] X. Zhang, T. Zhang, and S. Pande, "HIDE: an infrastructure for efficiently protecting information leakage on the address bus," *ACM 11thInternational*

Conference on Architecture Support for Programming Language and Operating Systems, 2004

[22] D. Page. Partitioned Cache Architecture as a Side-Channel Defense Mechanism. Cryptology ePrint Archive, Report 2005/280, 2005. Available at: http://eprint.iacr.org/2005/280.

[23] Z. Wang and R. B. Lee. New Cache Designs for Thwarting Software Cache-based Side Channel Attacks, *the 34th International Symposium on Computer Architecture*(ISCA 2007), June 9-13, 2007

[24] Z. Wang and R. B. Lee. A Novel Cache Architecture with Enhanced Performance and Security. *41st IEEE/ACM International Symposium on Microarchitecuture* (MICRO-41), pages: 83-93, 2008.

[25] Z. Wang and R. B. Lee. Covert and Side Channels due to Processor Architecture. *22nd Annual Computer Security Application Conference, 2006* (ACSAC'06), pages: 473-482, Dec. 2006.

[26] J. Kong, O. Aciicmez and H. Zhou. Deconstructing New Cache Designs for Thwarting Software Cache-based Side Channel Attacks, *Workshop on Computer Security Architecture*(CSAW), 2008

[27] J. Kong, O. Aciicmez, J. Seifert and H. Zhou. Hardware-Software Integrated Approaches to Defend Against Software Cache-based Side Channel Attacks. I*EEE 15th International Symposium on High Performance Computer Architecture,2009* (HPCA 2009), pages: 393-404, Feb. 2009

[28] O. Aciicmez and C. K. Koc. Trace-Driven Cache Attacks on AES.

[29] D. Bernstein. Cache-timing attacks on AES. Preprint, 2005, http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[30] O. Aciicmez, J.-P. Seifert, and C. K. Koc. Predicting Secret Keys via Branch Prediction. *The Cryptographers' Track at the RSA Conference (CT-RSA)*, 2007

[31] O. Aciicmez and C. K. Koc. Chapter 18 Microarchitechtural Attacks and Countermeasures. Chapter in "*Cryptographic Engineering*" by C. K. Koc, Springer, ISBN 0387718168.

[32] AJ Elbirt. Fast and Efficient Implementation of AES Via Instruction Set

Extensions.

[33] J. Blomer and V. Krummel. Analysis of countermeasures against access driven cache attacks on AES. *Workshop on Selected Areas in Cryptography (SAC),* 2007

[34] E. Brickell, G. Graunke, M. Neve, and J.-P. Seifert. Software mitigations to hedge AES against cache-based software side channel vulnerabilities. Cryptology ePrint Archive, Report 2006/052, 2006. http://eprint.iacr.org/.

[35] A. Seznec, "A case for Two-Way Skewed Associative Caches." *Proc. Int'l Symp. On Computer Architecture*, pp. 169-173, 1993

[36] Nigel Topham, Antonio Gonzalez. Randomized Cache Placement for Eliminating Conflicts. *IEEE Trans on computers*, vol. 48, no. 2, pp. 185-192, February 1999.

[37] Office of State Commercial Administration of China. "SMS4 cipher for WLAN products (in Chinese)," 2006. Available: http://www.oscca.gov.cn/UpFile/200621016423197990.pdf.

[38] X. Zhuang and H. S. Lee. Reducing Cache Pollution via Dynamic Data Prefetch Filtering. *IEEE Trans on computer*, vol. 56, NO. 1, pp. 18-31, January 2007.

[39] K. J. Nesbit and J. E. Smith. Data Cache Prefetching Using a Global History Buffer.

[40] R. L. Oliver and P. J. Teller. Dynamic and Adaptive Cache Prefetch Policies.

[41] A. Silberschatz, G. Gagne, P.B. Galvin. Operating system concepts, 7th edn. John Wiley and Sons, Inc. USA(2005)

[42] W.-M. Hu. Lattice scheduling and covert channels. Proceedings of the IEEE Symposium on Security and Privacy, vol. 25, pp. 52-61 (1992)

[43] R. Uhlig, G. Neiger, D. Rodgers, et al. Intel Virtualization Technology.

[44] MIPS Technologies Inc. MIPS32 4K™ Processor Core Family Software User's Manual. 2001

[45] G.. Bertoni et al, "Efficient Software Implementation of AES on 32-Bit Platforms", Lecture Notes In Computer Science, vol. 2523, pp.159-171, 2003.

[46] Fiskiran A. M., Lee R. B., "On-Chip Lookup Tables for Fast Symmetric-Key Encryption", ASAP 2005. 16th IEEE International Conference, pp.356-363, July. 2005.

Publications

[1] Lu Shi-Ting, Zhang Suiyu, Zhang Yulong, Han Jun, Zeng Xiaoyang. Architectural Integration of RSA Accelerator into MIPS Processor, ASICON'2009.

[2] 卢仕听，尤凯迪，韩军，曾晓洋。“MIPS 内存管理单元的设计与实现”，计算机工程，2010 年 22 期。（已录用）

[3] 卢仕听，王帅，韩军，曾晓洋。“AES 算法 SIMD 指令扩展方法与实现研究”，计算机工程，2011 年。（已录用）

[4] 韩军，卢仕听，张随欲，曾晓洋。一种抗时间驱动缓存攻击的硬件改进方案。专利

[5] 韩军，王帅，曾晓洋，卢仕听。一种使用于 MIPS 处理器的 AES 加密单元。专利申请号：200910198314.6

[6] 张随欲, 韩军, 卢仕听, 曾晓洋. 针对 SoC 系统的 Cache 攻击方法及建模分析. 计算机研究与发展。（已录用）

Acknowledgements

清晰地记得第一次来到张江这片静土，懵懵懂懂的跨入了三年的科研生涯。上海的纷繁也许带给我更多的是浮躁与不安，但是回首这成长与磨练的三年，没有斐然的成绩但是足迹清晰。

很感谢能师从曾晓洋老师门下，曾老师青年才俊的魅力对我是一种潜移默化的影响。很感谢曾老师对我生活的关切，很多的感动是信心和动力的来源。许多次交谈您给我的指点和宝贵的意见，教我如何规划自己的未来，细细体会小心珍藏这笔财富，对我的未来道路也有莫大的指引作用。

很感谢韩军老师在科研上的指导和生活上的关心，能在研究生阶段得到韩老师的帮助无疑是一笔最宝贵的财富。揣着知遇之心感谢韩老师对我的赏识和信任，我想您是我一生的良师和益友。

感谢师弟张随欲，尤凯迪，黄伟在科研上对我的帮助，跟你们走过的是一段愉快的赛龙舟时光， 默契的合作和快速的前进。

感谢我的诸位同门，和你们的共处的三年是一段热烈的弗拉明戈舞，洋溢着热情和快乐。

感谢已经毕业的师兄师姐陆荣华，赵佳，李庆，曹丹等，感谢你们的帮助。

感谢很乖的王帅师弟，很感激能帮我审阅多篇论文。

感谢肖瑞瑾对我的鼓励和对诸多话题的讨论和分享。

感谢刘俊宝，李毅，张玉立，李辉凯，黄贝，英彦，石泽文，周薇娜众师兄弟姐妹的欢声笑语，特别感谢和我一样喜欢《Fringe》的黄贝与我分享剧集的精彩。

感谢我的姐姐和姐夫，是你们的鼓励和支持使我能坚毅地走完这三年；感谢我的爸爸妈妈，是你们的疼爱和养育让我感受幸福，感恩生命。希望我的小外甥快乐健康的成长。

2010 年 5 月 18 日晨