Information Leakage Due to Cache and Processor Architectures

Zhenghong Wang

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
ELECTRICAL ENGINEERING
Advisor: Prof. Ruby B. Lee

November 2012

Abstract

When users share resources, interference between users often reflects their activities and thus leaks out information of a user to others. Microprocessors, and their associated cache memories, are typically one of the most shared resources in a computer system. Compared with traditional software-based and system-level information leakage channels, the ones in microprocessors are often much faster and more reliable – and hence more dangerous. They can also bypass existing software-based protection and isolation mechanisms, and can nullify any confidentiality or integrity protections provided by strong cryptography. Because of the ubiquitous deployment of microprocessors and the fact that the attacks are effective on essentially all modern processors, such microprocessor-level information leakage exists in almost all computing systems and has become a serious security threat to a wide spectrum of platforms and users.

Motivated by the increasing importance of the processor and cache information leakage problem, this dissertation aims to investigate the information leakage problem in microprocessors in a more generalized manner. The goal is to first understand the fundamental, rather than attack-specific, mechanisms that enable information leakage, and then propose countermeasures that attack the root causes and thus are generally effective. The dissertation also attempts to develop a theoretical model of information leakage channels, which can help analyze existing channels, identify new channels, evaluate their severity, and avoid such channels in future designs.

The dissertation starts with concrete practical issues that are of high importance. It first analyzes the recent cache-based software side-channel attacks, revealing their common root cause, then proposing novel cache designs that can effectively defend against all attacks in this category without compromising performance, power efficiency and cost. The proposed Newcache design can even improve performance over traditional cache architectures. The dissertation also analyzes existing processor architectures, identifies several new covert channels that are much faster than traditional channels, and discusses alternative countermeasures. The dissertation then generalizes the problem of covert channels with abstract modeling and analysis, which clarify the ambiguity in traditional classifications of covert storage versus timing channels, help identify new channels and reveal limitations of existing covert channel identification methods. The dissertation also recognizes that asynchronism is an inherent characteristic of covert channels that should be properly captured in channel capacity estimation. Quantitative results are presented.

Acknowledgements

It would not have been possible to write this doctoral dissertation without the help and support from many people. There are no words adequate to express my deep gratitude to all of them.

First of all, I would like to thank my thesis advisor, Professor Ruby Lee, for the patient guidance, constant encouragement and insightful advice she has provided throughout my time as her student. During the hard times in my thesis writing, in particular after I had started a fast-paced job in Silicon Valley and fathered two children, it was her patience and persistence that drove me to get through all the difficulties and keep making progress. I truly enjoyed working with her and feel lucky for being one of her students.

I am grateful to Professor Andrew Appel and Professor Mung Chiang for taking time to read the thesis as well as providing invaluable feedback and suggestions that help clarify the work. I also thank the faculty and staff of the Princeton University Department of Electrical Engineering for all the help and advice they gave me during my study in Princeton.

I thank the members of the PALMS group for their direct and indirect help in improving the work. I enjoyed the fun and inspiring discussions with them. I also thank all my friends, who helped make my life and study in Princeton more enjoyable.

Finally, I would like to thank my family. My wife Yan has provided endless support in all aspects of my life. My lovely son Eddie and daughter Eileen came to this world just in time to make my thesis writing more challenging. They however bring great happiness and helped me truly understand the meaning of the word "cherish". My parents and sisters have always cared for me, loved me, and supported me, even when they are thousands of miles away. To them I dedicate this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The term “information security” is often interpreted as the protection of information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction. The basis of information security consists of three components: *confidentiality*, *integrity* and *availability*. Confidentiality is the concealment of information, ensuring that information is accessible only to authorized parties. Integrity means guarding improper or unauthorized information modification or destruction. Integrity includes data integrity – the content of the data, and also origin integrity – the source of the data. Availability is the access to, and use of, information and resources in a timely and reliable manner.

Information leakage is a direct threat to confidentiality. It leads to the disclosure of information to someone who should not learn the information. In many situations, the leaked information may not be the ultimate target of an attack. It may help compromise integrity or availability as well and facilitate further steps of the attack, e.g., gaining higher privilege or even taking over the whole system. In practice, despite huge efforts in securing computer systems, the information leakage problem exists, more or less, in almost all practical systems.

1.1 Information Leakage: Practical and Theoretical Issues

Many factors contribute to the wide existence of the information leakage problem – some are due to practical reasons whereas others lie in the fundamental theory upon which the systems are built.

Among the practical issues, design flaws and implementation errors are common causes of information leakage problems. Design flaws may be introduced at various levels in the development phase. At the system level, incorrect assumptions and/or an insufficient threat model often lead to bad design decisions and flawed system architecture. A good example is the first generation XBOX gaming console. Among many mistakes that Microsoft made [1] (from the hackers’ perspective), transmitting the secret key in plaintext over the HyperTransport bus between the CPU chip and the Southbridge chip[1], which is directly visible to the hackers, is the worst one. The rationale behind this design decision was probably the fact that the HyperTransport bus was so fast that no logic analyzer at that time could sniff the bus, and it is a cheaper design. But it turned out to be an underestimation of the hackers' resources and expertise. Custom hardware was quickly developed and the attack succeeded [1].

When getting into low level implementations, the chances of developers making mistakes become higher due to the increasing amount of details involved. In complicated system design that involves developers with various skill levels producing millions of lines of code, having errors in implementation is essentially inevitable.

The common use of less secure programming languages, e.g., C, has also contributed to many of the implementation bugs. Unlike other high-level programming languages like Java, they are more vulnerable to security issues like the buffer overrun problem [2-3], which has been recognized as a major threat to software/system security. In reality, not all these errors can lead to information leakage, but many of them do cause issues, e.g., the well known "kernel memory disclosure" problem, which exposes the content of kernel memory to unprivileged users [4].

In addition to the mistakes made in the design and development stage, information leakage problems can also be introduced in the later stages of the system life cycle, e.g., after the system is placed online, or even after the life of the system has ended. Security problems in live systems are often due to incorrect configurations and/or bad practices in system management and maintenance, e.g., inappropriate security policies for users and data objects or simply bad passwords. There can also be problems after a system or some of its components are dead. Sensitive information should be securely shredded or carefully scrubbed, or it has to be properly protected or migrated to other live systems. Dead components, particularly storage devices, should be carefully processed before they leave the trusted sites. It is not difficult for an attacker to recover data from a dead hard drive. Advanced technology can even recover data from a hard drive that has been overwritten with 0's [5].

Theoretically, all these practical issues can be avoided, given sufficient time, effort and resources. However, there are some types of information leakage, which lie in the fundamental theoretic model on which systems are built, that are unavoidable even with perfect design, implementation, configuration and maintenance.

One theoretical issue originates from the process of abstraction, in which the physical system is abstracted into a logical model, which preserves only the properties of interest and discards the rest. In reality this is a very common practice as it allows most of the development efforts to be independent of physical devices. However, any protection mechanisms developed upon the logical system model would be effective only for the aspects that are modeled. Information may still leak out via the physical aspects that are not modeled in the system, e.g., through acoustic or electro-magnetic emission, power variation, or even thermal activities. Attacks based on these issues are often referred to as *side channel attacks*. One may argue that if the system model includes all aspects of the physical system, the design may be immune to side channel attacks. However, that would only be possible for very simple devices, assuming it is possible to enumerate all physical

[1] The Southbridge is one of the two chips in the chipset on a PC motherboard. It typically handles the I/O devices such as USB, audio, ISA bus, PCI, IDE channels etc.

properties. For most practical systems, it is infeasible to consider all physical aspects of the system in the design and implementation.

Another example of a theoretical issue relates to covert channels, which are often referred to as channels that are neither designed nor intended to transfer information [6]. One type of covert channels, the covert *storage* channels, make use of entities not normally viewed as data objects to transfer information from one process to another. Another type of covert channels, the covert *timing* channels, exploit the temporal characteristics of events to transfer information. As an example of a covert storage channel, the file lock attribute can be used to transfer information. One process can lock a file to encode a bit '1' and unlock the file to encode a bit '0', and another process can retrieve the bit by checking if the file is locked. As an example of a covert timing channel, information can be leaked out by modulating the use of CPU time, e.g., a process can try to use as much CPU time as possible to indicate a bit '1', and try not to use any CPU time to indicate a bit '0'. Other processes can extract the information by observing the system response time.

Due to the essentially unlimited number of ways in exploiting various objects and resources in a system, covert channels widely exist in practically all computer systems. Despite extensive work on covert channel analysis, systematic identification and elimination of covert channels remains a very difficult problem.

1.2 New Issues due to Cache and Processor Architectural Features

In the literature, information leakage problems were mostly studied in the area of covert channels and side channels which focused on either specific hardware/software targets such as cryptographic devices and software ciphers, or system and software level covert channel issues. Information leakage due to hardware processors (including their tightly coupled caches) did not receive as much attention in the past.

Compared with information leakage at other levels of a system, information leakage at the processor level is unique in several aspects. First, as the central processing unit of a system, processors are typically the most shared resources in the system – often among all users who may belong to different security domains, and therefore is an ideal place for inferring information from interference between users. Second, microprocessors are fast, and covert or side channels based on processor level interferences can often be orders of magnitude faster than those at the software level. Third, the processor level sharing often breaks software level isolation mechanisms like virtual machines (VMs). For example, two logically isolated VMs can still be running on the same physical microprocessor and share caches. It has been demonstrated in several recent work that sensitive information like cryptographic keys can be leaked out through shared caches. Fourth, in a processor the clocks are usually derived from common oscillating sources. This makes synchronization easier, e.g., between sender and receiver in covert channels, and therefore leads to faster covert channels.

The situation is getting even worse over the last few decades. The advances in process technology have enabled billions of transistors to be integrated into a single chip, allowing more on-chip resources to be allocated for new architectural and micro-architectural features that enhance performance, power efficiency, etc. Such new

```
...
/* loop counter is dependent on program input */
int cnt;
cnt = input();
for i = 0 to cnt  do begin
      …; /* some work done here */
end;
…
```

Figure 1-1. An input-dependent loop

additions however were often designed without being carefully examined for security. Furthermore, the increased system complexity increases the difficulty of identifying and mitigating of information leakage channels.

To understand the information leakage problem in processors, we first need to know what information can be leaked out, by what means. A program running on a processor may leak information due to various reasons, some dependent on processor features while others not. Information leakage due to a program's inherent characteristics, e.g., its algorithmic computation complexity, is often independent of the type of processor on which the program is running. Figure 1-1 is a simple example of this. Since the number of iterations is determined by the program input, the execution time of this code segment would reveal the value of the input (whether it is small or large), no matter what type of processor it runs on.

On the other hand, other types of information leakage by a program may depend on what processor it is running on. For example, a program doing table lookups may have constant execution time if the table lookups take constant time. This may be true if the program runs on a processor without caches and the memory accesses take constant time. However, if the underlying processor has a cache, constant table lookup time may not be possible. A table lookup that hits in the cache would take shorter time than one that misses in the cache. In this dissertation, we focus on such information leakage that is specifically caused by cache and processor architectural features.

1.2.1 Information Sources in Processors

Despite various forms of information leakage and the resulting attacks, in this dissertation they are categorized into two types, as we will explain below. Most modern processors (if not all) are based on the stored-program architecture, which consists of a processing unit, that performs operations to manipulate data, and memory storage, that keeps instructions as well as data. The instruction memory and the data memory can be unified, e.g., in Von Neumann Architecture, or separated, e.g., in Harvard Architecture. Programs running on such architectures involve two types of operations: fetching and storing the instructions or data, and processing the instructions or data. The first type of operations deal with the object location information, and the second type of operations deal with the values of the objects. These two types of information – the object location and the object value – are the main sources of information leakage in processors.

According to the type of operations that a processing unit is involved in, most components in real processors can be categorized into two classes. Components such as the instruction decoder and functional units such as ALUs are units that manipulate or transform object values. Components such as caches, TLBs, and branch predictors perform or facilitate object fetching and/or storing operations and process information about the object locations. During the operations of such components, information about the operations being performed can often be leaked out, which is the basis of real information leakage attacks. In the next section, basic mechanisms of information leakage will be summarized and examples given.

1.2.2 Basic Leakage Mechanisms

Information leakage mechanisms in processors mostly fall into two categories. The first type of leakage occurs during the operation of the information-leaking component. For example, the branching unit may exhibit different power consumption characteristics, indicating a taken branch or a non-taken branch. Another example is the memory system. When performing a memory access operation, the memory system may exhibit input-dependent delay – due to cache hit or miss, and can leak information about the input, i.e., the address of the access. Since such information leakage is due to the use of on-chip resources, we refer to this type of leakage as *leakage by resource use*.

Another type of information leakage is due to reporting mechanisms that exist in many modern processors. For the purpose of debugging and performance analysis and tuning, modern processors often implement event monitors and counters accessible to system software as well as application software. Such reporting mechanisms allow a program to learn information about other programs that it may not be able to observe via the first type of information leakage. We refer to this type of information leakage as *leakage by event reporting*. Note that leakage by event reporting does not require direct use of resources.

Leakage by resource use

Based on what type of information is leaked, leakage by resource use can be value-dependent leakage, address-dependent leakage, or hybrid leakage.

Value-dependent leakage: If a processing unit exhibits input-dependent behavior, it may leak out information about the value of the object being processed, causing value-dependent leakage. For example, functional units may exhibit input-dependent power consumption and leak out information about the input value, e.g., its hamming weight. Some simple CISC processors may implement complex instructions with micro programs, which could lead to variable execution cycles and leak information about the value of the instruction operand.

Address-dependent leakage: If a processor component operates on an object's location information, i.e., its address, it may cause address-dependent leakage. The branching unit and memory system including caches, Translation Lookaside Buffers (TLBs), etc., are good examples of this. A prefetching unit may also exhibit different behaviors for different access patterns, leaking out information about the memory access history.

Hybrid leakage: The leakage of object values and location information can sometimes be twisted together. For example, value prediction [7-8] and speculative execution [9] may speculate on operand values and lead to different execution paths for correct and incorrect prediction or speculation, respectively. The leakage of object values in such mechanisms, however, is through the variations of the objects' addresses.

Leakage by event reporting
Unlike leakage by resource use, leakage by event reporting usually does not involve operations that generate information-leaking variations of object values or locations. Instead, it exposes events that are already generated. For example, the performance counters in Intel processors can record information about a wide range of events including the number of cache misses, or retired branches, or TLB references, or micro-operations (uops) of various types, etc. In addition to the dedicated reporting mechanisms, other processor architectural features may also expose events that are originally invisible to a program. For example, control speculation in IA-64 allows a program to see the occurrence of an individual event such as a cache miss with a long delay, a page fault, an access right violation, etc.

Due to the different nature of the two types of information leakage mechanisms, the leaked information has different characteristics. Leakage by resource use usually exposes primitive information of individual operations, e.g., values or locations of certain objects, whereas leakage by event reporting usually exposes composite "high" level events that represent the overall program behavior. Leakage by resource use therefore is commonly exploited in attacks that require accurate knowledge of certain internal objects of a program, e.g., side channel attacks, whereas leakage by event reporting is more suitable for constructing covert channels – it provides a large set of mechanisms that allow the receiver of the channel to observe various aspects of the behavior of the sender, which is ideal for covert channels.

1.3 Dissertation Overview

Despite the extensive research in the areas of covert channels and side channel attacks, the information leakage problem due to microprocessors did not receive as much attention in the past. Existing work include the information leakage through the memory bus, covert timing channels based on CPU or bus contention, and cache based covert channels and side channel attacks, etc. We review all this in detail in chapter 2. All these work however addressed only specific attacks and the coverage is very limited. To the best knowledge of the author, a comprehensive and systematic examination of information leakage at the processor architectural level is not available.

This work is motivated by the fact that there are rich and unique mechanisms in microprocessors which allow unintended and undesired information leakage, and the lack of a thorough investigation of the problem in the past. It aims to analyze and understand information leakage at processor architectural and micro-architectural level, and research suitable countermeasures before irreparable damage is done. The work first investigates real attacks, especially on processor caches, that are of high significance, analyzes concrete problems and proposes novel and effective solutions. The work then generalizes the problem with abstract modeling and classification, based on which theoretical analyses are performed. The generalized discussion helps clarify past misconceptions and allows better modeling and clearer classification of information leakage channels. The better understanding of the nature of the problem also helps identify new information leakage channels.

The rest of the dissertation is organized as follows. In chapter 2 we review related past work, including covert channel analysis, side channel attacks, steganography as well as other information hiding techniques. In chapters 3, 4, and 5, concrete information leakage problems of significant importance are first addressed. Chapter 3 [10] analyzes the recently reported cache based side channel attacks and reveals the common root cause of these attacks. Novel cache architectures, including the PLcache and RPcache, are then proposed as universal countermeasures to such classes of attacks. In chapter 4 [11], we present Newcache, another novel cache architecture that improves performance even as it improves security. The proposed cache architecture inherits the short access time and high power efficiency from the direct mapped cache architecture, and at the same time, enjoys low miss rates comparable to a highly associative cache. It also can prevent information leakage in caches and hence is immune to the cache-based side-channel attacks. In addition, it can bring several other benefits such as fault tolerance, power and thermal optimizations. In chapter 5 [12], we present new fast covert channels we identified in processors, and propose corresponding countermeasures. The theoretical aspect of the information leakage problem is addressed in chapter 6 [13-14]. As the basis of further theoretical analysis, the information leakage mechanisms are first generalized and an abstract channel model is constructed. The ambiguity in traditional classifications of covert storage and timing channels is explained and we propose a new classification that resolves this ambiguity. This chapter also presents new results on channel capacity estimation, pointing out that asynchronism is an inherent characteristic of covert channels that should be properly captured in channel capacity estimation. Finally, Chapter 7 summarizes the contributions of the work and discusses possible directions of future work.

Chapter 2

Related Work

In the most general sense, information leakage can be defined as any unwanted information distribution or transfer. It may occur anywhere in a system and the information can be leaked out in vastly different ways. In practice, information leakage is often a result of access control failure, which leads to direct exposure of information to unauthorized parties. Information can also be leaked out indirectly, even when the system has properly designed and implemented access control. In a computer system, the operations that process data can cause interference observable to others, from which certain information of the data being processed can be inferred. Many covert channels and side channel attacks are based on such indirect information leakage. Information leakage may occur at various levels of the system. For example, application software may fail to properly clear cryptographic keys after use, system software may contain bugs that allow exposure of kernel memory to unprivileged processes, and hardware circuits may exhibit data dependent behaviors such as various power consumption or operation timing. Information leakage can be unintended leakage, e.g., due to buggy software that accidentally expose one's private data to others. Information can also be leaked out intentionally, e.g., by a Trojan horse that deliberately sends out information. Due to the abundance of information leakage mechanisms, information leakage was studied in a wide range of research areas in the literature, each of which investigates certain aspects of the problem in a particular context. In this section, the related work including covert channels, side channel attacks, information hiding and miscellaneous unintended data exposure are first reviewed. The relationship between the main focus of this work and the past work is then summarized, and the scope of this dissertation clarified.

2.1 Covert Channels

2.1.1 Definitions of Covert Channels

In the literature, the term "covert channel" was used to refer to a variety of unconventional communication mechanisms. This section gives an overview of covert channels, reviews the existing definitions, and clarifies the subject of discussion.

The notion of covert channels was first introduced by Lampson [6]. He examined nontraditional means of information transfer, which he referred to as “covert channels”, in the context of program confinement. In such a context, there are normally two parties (e.g., processes in a computer system) involved in a covert channel: the sender S and the receiver R, who are disallowed by the system security policy to communicate in one or both directions between them. S and R therefore have to retort to nontraditional ways to exchange information. Lampson’s definition of covert channels was [6]:

Definition 1: channels that are neither designed nor intended to transfer information.

This definition points out the nature of covert channels but does not provide information on how covert channels can be constructed. Definitions 2 and 3 define covert channels from an implementation perspective.

Definition 2: channels that use entities not normally viewed as data objects to transfer information from one subject to another [15].

Definition 3: channels that are a result of resource allocation policies and resource management implementation [16].

Unlike in normal communication channels, information transferred in covert channels is usually encoded into objects not used for data storage (e.g., control objects rather than files or messages), or modulated over the use of shared resources, causing interference among processes from which information can be indirectly inferred. Below are two examples of such covert channels:

Example 1: the file lock channel [17]
In systems that provide file locking capability, the status of the file lock can be exploited to transfer information between processes. The sending process S can lock a file to indicate a 1 and unlock it to indicate a 0. The receiving process R can then extract the bit by checking the status of the lock.

The file lock channel is a representative example of covert channels under Definition 2. There are also other means to construct covert channels, among which exploiting shared resources, e.g., the CPU time, is the most common one.

Example 2: the CPU scheduling channel
In multi-tasking systems, the CPU is shared among multiple processes, each of which is given a certain amount of CPU time. A process can modulate information over its own use of CPU time and interfere with other processes. For example, the sender S can use as much CPU time as possible to indicate a 1 and use little CPU time to indicate a 0. The receiver R can recover the information by comparing its own progress with a timer. A slow down of R’s execution would indicate a bit 1 sent by S.

Definitions 2 and 3 however can hardly cover all covert channels due to the unlimited number of ways that covert channels can be constructed with. For example, Definition 2 did not consider the covert channels based on the timing of events. Furthermore, although all the previous definitions are intuitively clear, they are informal and ambiguous, and thus can hardly be used in systematic covert channel analysis with automatic tools. In [17-18], security policy was introduced into the definition of covert channels, which removes much of the ambiguity.

Definition 4: channels that allows a process to transfer information in a manner that violates the system's security policy [18].

Definition 5: given a nondiscretionary (e.g., mandatory) security policy model M and its interpretation I(M) in an operating system, any potential communication between two subjects $I(S_h)$ and $I(S_i)$ of I(M) is covert if and only if any communication between the corresponding subjects S_h and S_i of the model M is illegal in M [17, 19].

In particular, Definition 5 pointed out the irrelevance of covert channels with discretionary security policies [17]. Implementations of discretionary policy models within operating systems cannot determine whether a program may release information in a legitimate manner [20], hence any user can make use of the legitimate communication channels rather than covert channels to leak out information. Compared with exploiting mechanisms not intended for communications as in covert channels, leaking information through legitimate channels is much faster, more convenient and harder to detect, in particular with the help of steganography as well as other data hiding techniques.

Definitions 4 and 5 are particularly interesting in the context of Multi-Level Security (MLS) systems where mandatory security policies are widely used and formally defined. DoD's Trusted Computer System Evaluation Criteria (TCSEC) [18] has adopted such a definition. In this dissertation, a combination of Definitions 1 and 5 is adopted as the definition of covert channels. This is not to create a new definition, but to clarify the three characteristics of covert channels. More specifically, by "covert channel" we refer to a channel that:

1. exploits mechanisms that are not designed for communications,
2. violates the system's security policy M, where M is a non-discretionary policy,
3. involves an insider that intentionally sends out information.

Note that the third characteristic was not explicitly stated in definitions 1 to 5, but was indeed assumed in the context of discussion, e.g., insider attacks in MLS systems. This distinguishes covert channels from other unintentional information leakage problems.

In addition to the covert channels defined above, there are also other uses of the term "covert channels" in the literature which however refer to different areas of work. For example, steganography is often regarded as a form of covert channel [21-22]. Definition 6 is a definition used in such areas.

Definition 6: A covert channel is a parasitic communications channel that draws bandwidth from another channel in order to transmit information without the authorization or knowledge of the latter channel's designer, owner, or operator [22].

Unlike covert channels that exploit unintended mechanisms to transfer information, steganography hides information in legitimate text such as ordinary files and network packets. Since the main interest of this area of work is the *covertness* (i.e., the *secrecy*) of the *communications* (not just the information being transferred), and because the term "information hiding" was not yet invented at that time [23], the term "covert channels" was also used in this area. In this dissertation, according to our definition of covert channels, steganography will not be discussed in the scope of covert channels but will be discussed as a form of information hiding technique.

The definition of covert channels in this dissertation also distinguishes side channels from covert channels. Although information leakage in covert channels and side channels may be based on the same physical mechanisms such as operation timings, side channel attacks assume no insiders and thus are unintentional information leakage. The targets of side channel attacks are mostly crypto ciphers which by no means would intentionally leak out information.

2.1.2 Covert Channel Classification

In the literature, covert channels are often categorized into two types: covert *storage* channels and covert *timing* channels [18, 24-26]. Covert storage channels usually make use of objects not intended for data storage whereas covert timing channels exploit the temporal characteristics of events to transfer information. For example, TCSEC [18] adopted the following definitions:

> *Covert storage channels*: covert channels that "would allow direct or indirect writing of a storage location by one process and the direct or indirect reading of it by another".
> *Covert timing channels*: covert channels that "would allow one process to signal information to another process by modulating its own use of system resources in such a way that the change in response time observed by the second process would provide information".

The file lock channel described in section 2.2 is a typical covert storage channel, which makes use of the file lock object to carry the information. The sender is able to "write" a bit to the file lock, though indirectly, and the receiver can "read" the file lock value. Many other mechanisms in operating systems can be exploited as well. For example, by allowing a process to detect whether a directory exists or not even though the process does not have enough security clearance to access the directory, an insider can send bits out by creating and removing a directory known by the receiver. Resource-exhaustion channels are another common class of storage channels. Based on the information bit to be sent, the insider can choose to use up, or not, the shared resources. The receiver then makes an allocation of the exploited resource and observes if the allocation fails, or not, to infer the bit. The CPU scheduling channel is a representative covert timing channel as the information is modulated over the observer's response time. Other shared resources can also be exploited to construct covert timing channels in a similar way. Indeed, almost all mechanisms that allow a process to impact the system performance can be exploited as timing channels [21].

In addition to those in standalone systems, covert channels exist in networks as well. Examples of covert storage channels in networks include those based on the pattern of the network packets' destination address, unused header bits and certain packet fields, and examples of covert timing channels include those based on packet rate, packet timing, network protocols such as the Medium Access Control (MAC) protocol of wireless networks, etc. [27-34]. A survey of network covert channels can be found in [35].

The classification of covert channels as storage channels vs. timing channels is clear and helpful in covert channel analysis most of the time, and thus was widely accepted and used in the literature. However, researchers admitted that the difference between storage channels and timing channels is actually unclear [17, 23, 26] and there are covert channels that are hard to categorize. Below is an example:

Example 3: the disk arm channel
When servicing a sequence of disk access requests, the operating system often re-orders the requests to avoid unnecessary seek operations, i.e., the radial movements of the disk arm, which are very expensive in terms of time. Due to the similarity between the problem of scheduling a disk arm and that of scheduling an elevator in a tall building, the *elevator algorithm* is commonly used for disk access optimization, i.e., the disk arm keeps moving in the same direction until there are no more outstanding requests in that direction and then switches the direction. This allows a covert channel as described below. Assume that the inner-most cylinder is numbered 0 and the outer-most is numbered N. The receiving process R first initializes position of the disk arm to cylinder N/2 by requesting accesses to that cylinder. R relinquishes CPU after this access is completed. The sending process S can then encode a bit of information by accessing either cylinder 1 or N-1 to send a bit ‘0’ or ‘1’, respectively. To receive the bit sent by S, R issues two requests – one to cylinder 0 and the other to cylinder N, and observes the order of completion of the two requests. If the access to cylinder 0 completes first, it means that the disk arm’s movement is N/2 → 1 → 0 → N, indicating that S accessed cylinder 1 and thus a bit ‘0’ is sent. If R’s access to cylinder N completes first, the disk arm’s movement is N/2 → N-1 → N → 0, meaning that S accessed cylinder N-1 and thus a bit ‘1’ is sent.

This channel was categorized as a storage channel by Karger [36]. His argument was that the disk arm is a storage object whose value is the position of the disk arm and there is no timing measurement involved in this channel. However, Wray, the second author of the paper, regarded the disk arm channel as a timing channel. He argued that the disk itself is a timer and observing the ordering of events is a kind of timing measurement [26]. In fact, the ambiguity of the difference between storage channels and timing channels is due to the non-rigorous definitions of the two types of channels. For example, the rigor in the definition of timing channels relies on the rigor in the definition of time, which however is ambiguous as illustrated in [26]. To the best knowledge of the author, this ambiguity is still unresolved.

2.1.3 Covert Channel Analysis

Dealing with the covert channel problem usually involves three steps: covert channel identification, capacity (or bandwidth) estimation, and covert channel handling. Covert channel identification attempts to find all, or as many as possible, covert channels in the system. Once identified, covert channels must be measured for their severity. A common metric is the channel capacity, or channel bandwidth, which measures how fast information can be sent over the channel. Depending on the needs, covert channels can be audited, mitigated, or eliminated.

2.1.3.1 Identification

Covert channel identification requires the analysis of all or a subset of system documents, including high level system specifications, system reference manuals, implementation source code and hardware manuals. A common approach of covert channel analysis is flow analysis. Information flow can be derived from the documents by attaching information flow semantics to the statements of the specification or programming language. In a program, data dependency and control dependency lead to information flow. For example, both assignment statement "$x{:=}y$" and if-else statement "if ($y{==}c$) $x{:=}a$ else $x{:=}b$;" causes information flow from variable y to variable x. Examples of more information flows in programming language statements can be found in [20, 37-38]. The derived flows are then checked with the flow policy which is a representation of the system security policy. Such a procedure can be automated for analysis over formal specifications and source code, which has been adopted in several tools, including the SRI Hierarchical Development Methodology (HDM) and Enhanced HDM (EHDM) tools [39-40], the Ida Flo tool [41] and the Gypsy tools [42-44]. Information flow analysis can be further augmented with more semantic components. In [45] Tsai presented a method based on the analysis of programming language semantics, kernel code and data structures, and the resolution of aliasing of kernel variables. Together with the information flow analysis, direct and indirect visibility and alterability of kernel variables are examined and potential covert storage channels are identified.

Information flow analysis methods shown above however are not suitable for specifications written with informal languages. In [15] Kemmerer proposed the Shared Resource Matrix (SRM) method which can be applied to both formal and informal specifications. The SRM approach requires the construction of a shared resource matrix from the specifications being analyzed. The matrix consists of visible/alterable shared resources and their attributes as columns and user-visible primitives as rows. Each entry of the matrix can be marked as either R or M if the corresponding primitive can reference (read) or modify (alter) the corresponding attribute. To identify indirect references to resource attributes, a transitive closure needs to be performed on the entries of the matrix. To detect potential covert channels, each column containing row entries with either an R or an M is analyzed since the resource attributes of these columns may be exploited for covert channels. A process that can alter an attribute can send information to a process that can read the same attribute, which may form a potential covert channel. Further analysis of the identified potential covert channels is then performed to determine if they are indeed exploitable. Some potential covert channels identified by the SRM method may be in parallel with an overt channel, or have the same process as both the sender and the receiver, or can only pass information that is already known by the receiver. Such channels are not real covert channels.

Based on similar information and procedures used in the SRM method, Porras and Kemmerer proposed the Covert Flow Tree (CFT) method [46] that allows the search for covert communication scenarios with a graphical tool. The dependency information is first analyzed, which identifies the resource attributes that should be further analyzed. The trees are then constructed for such attributes. The left branch of the tree is the series of operations caused by the sender to alter the attribute and the right branch is the operations that enable the receiver to perceive the modification. Potential channels are then analyzed as in the SRM method to determine if real covert channels exist.

Noninterference analysis is another popular covert channel analysis method, first introduced by Goguen and Meseguer [47]. It is based on the concept that "one user should not be aware of any activity of another user that he does not dominate" [48] and does not examine flows directly. By modeling the system as a state machine, a precise definition of noninterference can be expressed with the effects of the system's input history on a user's view of the system state and output. Loosely speaking, if user X does not interfere with user Y, deletion (or purge) of any or all X's inputs from the system's input history should not change Y's output. Formal definitions of noninterference can be found in [47, 49-50]. In practice, analyzing the entire history of the system inputs is infeasible, and the "Unwinding Theorem" solves this problem [51]. The "Unwinding Theorem" allows noninterference to be checked by examining the properties of the machine' state transition function, avoiding analysis of history traces. Noninterference analysis is advantageous for avoiding the discovery of false illegal flows. Its main drawback is that it requires the construction of the state machine and the selection of users' "view" functions that captures the system state visible to the user, which are nontrivial. Noninterference analysis was popular and has been applied to several systems including the SAT abstract model [52].

In summary, covert channel identification is a difficult problem which has not been completely solved despite all of the above work. On the one hand, the lack of formal specifications in real system designs disallows the use of the formal security-provable techniques, and at the same time analysis of informal specifications can not ensure security. Furthermore, it is really hard, if not impossible, to enumerate all possible information transfer mechanisms and include them into the system model. Under the "incomplete" system model, the ignored mechanisms can lead to covert channels even if the security of the system is theoretically proved.

2.1.3.2 Capacity Estimation

The task of capacity estimation of a covert channel is to estimate the maximum attainable bit rate of information transfer over the channel. Millen [53] first connected information flow models to Shannon's communication theory [54] and introduced the notion of covert channel capacity as a measurement of the covert channel information rate.

Unlike in most traditional communication channels, the times required for sending different bit values in many covert channels are different and depend on the history of bit transmission. Such channel properties are better captured by a state machine model as proposed by Millen in [55]. Millen assumed that the channels are noiseless, without interruption from processes other than the sender and the receiver, and the time required for the synchronization between the sender and the receiver is negligible. Such assumptions are valid in the context of estimating the maximum information rate. With the help of information theory, capacity of the state machine channels can be calculated once the transition overheads are determined. Information theory based capacity analysis was also applied to various other types of channels. The related work include Moskowitz's work on the capacity of certain noisy timing channels [56], the simple timing channels [57], the Timed-Z channels [58], Kang's work on the pump [59-60], Gray's work on the bus-contention channel [61], Giles's work on channels based on packet timing [62], and Venkatraman's work on network channel capacities [63], etc.

The information rate of a covert channel can also be estimated with informal methods. In this dissertation, such estimation is referred to as bandwidth estimation rather than capacity estimation although some past work used these two terms in an exchangeable manner. Capacity estimation gives the theoretical upper bound (which is tight) of the information rate whereas bandwidth estimation gives an approximation of the bound. Bandwidth estimation usually adopts simpler equations as an approximation for the rate calculation. Examples of such work can be found in [17, 64].

2.1.3.3 Covert Channel Handling

The purpose of covert channel handling is to minimize the damage of covert channels to system security. The ultimately secure way to deal with covert channels is to eliminate them. Some covert channels can be closed by redesigning the system. For example, the file lock channel can be blocked by disallowing a user to check the lock status of a file that he is not entitled to access. However, handling all covert channels in a system via channel elimination is practically impossible. For example, closing covert channels based on shared resources would essentially require strict resource partitioning or no resource sharing at all. Such an approach may be applicable to small system modules without sacrificing system usability and performance, but applying it to the whole system would normally impose too many restrictions on the system design and eventually lead to unusable systems.

Another covert channel handling strategy is through deterrence. The main deterrence method is channel auditing. If a known covert channel involves operations that are normally infrequently used and has low capacity, handling it via auditing can be a good choice. It is easy for the auditing system to detect the use of such a channel with reasonable expenses on recording, and before the channel use is detected only a tolerable amount of information can be leaked out. Channel auditing however has several fundamental problems. Not all covert channels are suitable for auditing, and some of them are simply undetectable. The use of a covert channel is usually detected afterwards, sometime long after the occurrence and the damage has been done. The analysis of audit data is nontrivial and could be very time consuming. More details about the problems of channel auditing can be found in [65].

The third method of covert channel handling is based on capacity reduction. Such a method does not eliminate the covert channel completely, but makes it much slower. One class of the work aimed to reduce capacities of timing channels by playing tricks with time. Popek and Kline proposed to restrict each process to see only virtual time, which depends solely on its own activity [66]. By making the time of each process less correlated to each other, the capacities of timing channels between these processes are reduced. Hu proposed to use "fuzzy time" [25] in the system, which makes *all* timing sources visible to the processes noisy. Giles et al. proposed the timing jammer to mitigate the packet timing channels [62, 67]. Noisy timing measurement would generally lead to lower capacity of most timing channels. Another approach for reducing covert channel capacity is to slow things down such that given the same amount of time, less information can be transferred. The most straight forward implementation is to slow everyone down, e.g., by adding delay to every system call in a system [23]. In [68] Hu proposed the lattice scheduling technique which makes use of the secrecy class attributes of processes to make decisions. The lattice scheduler reduces the frequency of transitions between processes that could be the two ends of the covert channel and slows down the transmission procedure. Resource partitioning can also be used to reduce covert channel capacity. Gray proposed probabilistic partitioning to mitigate the bus-contention channel

[61]. Reducing covert channel capacity is also an interesting topic in the design of legitimate communication channels between security domains, e.g., from the low security level to the high security level. To ensure reliable communications, feedback mechanisms are usually needed to send back reception acknowledgements to the sender who is at the low security level, which leads to covert channels. To reduce the capacity of such covert channels, the pump and its variants were proposed [59-60, 69-70]. By placing a buffer between Low and High, the acknowledgements from High to Low are decoupled into two separate ACKs, one from the buffer to Low and the other from High to the buffer. By properly controlling the ACK times, the capacity of the covert timing channel can be reduced without compromising the reliability and performance of the legitimate communication channels.

The covert channel problem in general is a hard problem. It is believed impossible to make any realistic systems free of covert channels. There is always a trade-off between the level of security and the performance, usability, cost, etc.

2.2 Side Channel Attacks

2.2.1 Overview

Side channel attacks are a special class of attacks that are based on indirect information leakage due to a systems' physical implementation. During the operation of a system, the variation of the system's power supply current, operation timing, electromagnetic radiation and/or acoustic emission can all carry information – often referred to as side channel information – which reveals the system's internal states and the data being processed. Although theoretically side channel information can be exploited to attack all kinds of systems, in practice the targets of the attacks are mostly implementations of cryptosystems, particularly simple cryptographic devices such as smart cards.

Side-channel cryptanalysis is different from classical cryptanalysis although they both aim to break cryptosystems. Classical cryptanalysis views the target cryptosystem as an abstract mathematical object and attacks the weakness of the mathematical composition itself, whereas side-channel cryptanalysis views the target as a particular physical realization and attacks the system physically rather than mathematically. Although side-channel cryptanalysis is much less general than classical cryptanalysis due to its implementation-specific nature, it is often much more powerful. Classical cryptanalysis typically requires a huge amount of computation and is only able to shrink the search space for the secrets. If the cipher is mathematically strong, classical cryptanalysis may be practically ineffective. In contrast, side-channel cryptanalysis often can recover more secret information – sometimes the full crypto key – in a much shorter time. It can be very effective even in attacking mathematically strong ciphers.

2.2.2 Classification of Side Channel Attacks

In the literature, side channel attacks are usually classified in two different ways.

Passive vs. *active*: In passive attacks, the attacker does not interfere with the target system and only observes the system's normal behavior. In active attacks, the attacker tries to tamper with the system's normal operations, e.g., by introducing faults into the computation, and deduces useful information from the system's responses.

Invasive vs. *non-invasive*: Invasive attacks usually involve physical deconstruction of some parts of the target system, e.g., depackaging a chip, to gain access to some internal components such as an internal data bus. Non-invasive attacks only exploit externally available information such as power supply current, operation timing etc. In [71], Skorobogatov et al. introduced a new type of attacks – the semi-invasive attacks. These attacks require depackaging the chip to gain access to the chip surface, but do not tamper with the chip further to probe internal components.

Note that the above two classification methods are orthogonal. For example, a passive attack may require depackaging a chip to gain access to the necessary information sources, and an active attack does not always imply an invasive attack – faults can be introduced with nondestructive methods, e.g., by heating the chip.

Although invasive attacks often allow the attacker to gain access to more information which makes the attack more powerful, in practice non-invasive attacks seem to receive more research and engineering effort due to economic reasons. Invasive attacks often require special equipment such as a scanning electron microscope or a probing station which are expensive and usually not available for individuals. Non-invasive attacks do not have this requirement and can be adopted more widely.

2.2.3 Representative Attacks

Side channel attacks are extremely implementation-specific and this review does not attempt to list all of them. Below we briefly introduce representative attacks in each category and give references to other related work.

2.2.3.1 Power analysis attacks

The power consumption of a cryptographic device often carries much information about the operation taking place and the data being processed in the device. In [72] Kocher et al. first presented attacks based on power analysis – referred to as simple power analysis (SPA) and differential power analysis (DPA).

The power measurements involved in power analysis are usually referred to as traces. A trace consists of a set of measurement samples, e.g., the power supply current, during a cryptographic operation. Due to the physical characteristics of the device, the device's internal states and the operations taking place in the device are transformed into the variation of the power consumption, which is recorded in the traces and can be recovered directly or statistically.

Simple Power Analysis (SPA): SPA refers to the technique that analyzes variations in the power traces that are directly distinguishable. Such large-scale variations are usually due to the execution of different operations. In implementations with software running on processors, different instruction sequences exhibit different power consumption profiles. In pure hardware implementations, different function modules consume different amount of power. In real attacks, SPA can identify operations with unique properties, e.g., round operations in ciphers like DES and AES – they cause repetitive patterns in power traces.

A more significant application of SPA is to identify conditional branches, which exhibit different SPA characteristics when executing different paths. In some RSA implementations, the conditional branches in exponentiation operations are controlled by the secret key bits. Finding out the branching direction would directly lead to the recovery of the secret key bit. Identifying conditional branches was also adopted in SPA to recover information from the DES key schedule computation, DES permutations, and various string or memory comparison operations [72].

Differential Power Analysis (DPA): DPA aims to recover information from subtle variations in the power traces that SPA can not exploit and thus is more powerful. In particular, the power variations correlated to a device's internal states are usually very small and overshadowed by the large-scale signals and measurement noise. Statistical techniques are employed in DPA to amplify the weak signal of interest and remove the effects of noise as well as other unrelated signals. Although the exact form of DPA is attack-specific, the underlying ideas are similar. In a typical DPA, the attacker first selects an internal state to exploit – often an intermediate result of the cryptographic computation that depends only on the plaintext or ciphertext, and a small part of the secret key (meaning a small search space). The effects of the selected state on power consumption therefore carry information about the secret key. Multiple power traces are normally needed to provide enough samples for statistical analysis. The key recovery procedure typically involves tests of a small number of hypotheses on the secret key values in the search space using the power measurement samples. Below is an example.

Let $S(C,"b",K_s)$ denote the function – often referred to as the *selection function* – that describes the relation between the selected state b (e.g., a binary bit), the system input or output C (e.g., the plaintext or ciphertext), and the secret K_s (e.g., a byte of the secret key). The outcome of the selection function $S(C,"b",K_s)$ is the value of the state b given the values of C and K_s. In DPA, the attacker knows the C value for each trace T. He guesses possible values of K_s in the search space and tests if the hypotheses are true. To test a hypothesis, each power trace is classified as a 1-trace or a 0-trace based on the outcome the selection function given the trace's C value and the guessed value of K_s. A 1-trace has a corresponding b=1 and a 0-trace has a corresponding b=0. The hypothesis can be tested by computing the difference between the averages of 1-traces and 0-traces:

$$\Delta T = \frac{1}{M1}\sum_{i=1}^{M1} T_i\Big|_{b=1} - \frac{1}{M0}\sum_{j=1}^{M0} T_j\Big|_{b=0} \tag{2.1}$$

where M1 and M0 are the numbers of 1-traces and 0-traces, respectively. For a correct guess of the K_s value, the classification of 1-traces and 0-traces is correct and the mean computations in (2.1) average out the noise in 1-traces and 0-traces while maintaining the bias caused by b. The subtraction then removes the common signals that is unrelated to b. ΔT therefore measures the power variation due to different values of b. If the hypothesis is wrong, the traces are incorrectly classified and the computed ΔT is roughly zero. The attacker tests hypotheses on all possible key values in the search space. The key value with a corresponding ΔT that is sufficiently different from zero is likely the correct guess.

More advanced power analysis techniques were also available. An important improvement of DPA is high-order DPA [73-75]. Instead of assuming that at some point of the computation the intermediate state value is correlated to the power consumption as in classical DPA, high-order DPA considers effects of the state at multiple points in the power consumption curve. The resulting differential function – the equation (2.1) in the case of classical DPA – has a high-order form. High-order DPA is able to break countermeasures that defeat classical DPA [76]. Another important improvement on power analysis is the template attack [77-78]. Template attacks rely on precise modeling of noise, with which the attacker is able to fully extract information from a *single* sample. Template attacks however require the attacker to possess an experimental device identical to the target system on which he can do experiments of his choice.

Power analysis has been demonstrated very effective in attacking various implementations of almost all major cryptosystems. Numerous reports were published on attacks on software as well as hardware implementations of DES [72, 79-83], AES [84-91], RSA [92-95], Elliptic Curve Cryptosystems (ECC) [96-100], RC4 [77], IDEA, RC6 and HMAC [101]. A comprehensive review of power analysis can be found in [102].

2.2.3.2 Timing analysis attacks

Operation timing is another important source of side channel information. The variations in execution time can be the result of multiple factors: the operation's inherent computation complexity, software implementation issues such as branches and conditional execution of operations, and hardware dependent effects such as cache misses and variable branching penalty.

The idea of timing analysis of cryptosystems was first introduced by Kocher in [103] where he analyzed implementations of Diffie-Hellman, RSA and RSS. A timing attack was then practically implemented against an RSA implementation with Montgomery algorithm [104]. The attack was further improved in [105-106] and was able to recover a 512-bit secret key with 5000~10000 timing measurements. Timing analysis was also applied to RSA implementations that use Chinese Reminder Theorem (CRT). The resulting attack was very powerful – a 1024-bit secret key can be recovered using about 370 time measurements [107]. In addition to attacks targeted at local machines, timing analysis even allows attacks over networks. In [108], a remote attack against the OpenSSL implementation of RSA was demonstrated practical, despite the high noise level in real world networks. This work was further improved in [109]. Timing attacks were also applied to block ciphers such as AES [106].

Most of the early work on timing analysis were based on a *black-box* model, i.e., only externally available signals (normally the running time of the entire cryptographic operation) are accessible to the attacker. This is particularly true in attacks on embedded devices such as smart cards where the timings of the internal sub-operations are normally unavailable. Timing analysis attacks under such a model therefore rely on statistical analysis and require a considerable number of time measurements. The recent cache attacks, however, have demonstrated that information leakage in processor caches enables richer forms of attacks based on both *black-box* analysis and *white-box* analysis. The attacks are generally effective – almost all processors with caches are vulnerable, and can be used to attack embedded devices as well as general purpose systems. Cache attacks have received significant interest due to their wide impact.

In the literature, Page first described theoretical attacks based on information leakage in caches and categorized the attacks into two types: trace-driven attacks vs. time-driven attacks [110]. The attacker in trace-driven attacks is able to detect the outcome of each victim's memory reference in terms of hit or miss whereas in time-driven attacks, the attacker can only see an aggregated profile, e.g., the total number of hits or misses.

Although Page's work is not specifically in the context of timing analysis, most existing cache attacks employ timing analysis due to the convenient timing measurements with the high accuracy timers widely available in modern processors. Also, some processor features such as Simultaneous Multi-Threading (SMT) allow an attacker to run in parallel with the victim on the same chip and observe the victim's memory references in real time which exposes information about the internal sub-operations of the target cryptosystem. In addition to the trace-driven and time-driven types of attacks, a cache attack can also be classified as an access-driven attack [111]. The attacker in an access-driven attack is able to learn not only the cache's hit/miss behavior but also which cache lines/sets are touched, individually or in an aggregated manner.

The first practical cache attack was implemented by T sunoo et al. [112-113] in 2002 and 2003. The attack exploits the cache collisions in DES code and was a time-driven attack. In 2005, Bernstein showed the vulnerability of software AES implementations due to evictions of AES table entries in the cache and presented a time-driven attack that was able to recover a large portion of the key (the exact number of key bits that are recoverable depends on the cache configuration of the target system) [114]. This attack was further improved in [115] by exploiting the second round operation of AES – in addition to the first round operation, and was able to recover the full key. Bonneau et al. [116] proposed another time-driven attack that requires less timing measurements than in Bernstein's attack. This attack relies on cache collisions – cache hits due to accesses to the same AES table entry – rather than cache evictions of AES table entries. In [117], Aciiçmez et al. pointed out the infeasibility of the existing cache attacks as remote attacks and proposed a real remote attack. A significant amount of timing measurements however are required. In addition to time-driven attacks, Osvik et al. also described several variants of cache attacks against AES in [118] and [119], in which the attacker is able to detect individual internal operations such as AES table lookups by making use of Simultaneous Multi-Threading (SMT). The attacker runs simultaneously with the victim process on the same chip where the cache is shared. Due to the cache line evictions caused by the victim, the attacker is able to learn which cache lines/sets are touched by individual memory references of the victim. The resulting attacks are very powerful. In [120], Neve et al. improved the attacks by making use of the final round operation of AES and making the use of SMT not necessary. Similar techniques were also adopted by Percival in his attack against RSA [121]. The full 1024-bit secret key can be recovered in a single encryption.

In addition to cache based attacks, other processor architectural features can also be exploited in timing attacks. Aciiçmez et al. have demonstrated successful attacks against RSA by exploiting branch prediction units in modern processors [122-124].

2.2.3.3 Electromagnetic analysis attacks

Electrical currents produce electromagnetic fields. Electromagnetic analysis makes use of the information carried in the electromagnetic (EM) field of the target device during its operation and extracts the information of interest. Since the 1950's, the US government has been aware of the information leakage via electromagnetic emanations, which leads to the standard called TEMPEST [125].

According to Agrawal et al. [126], electromagnetic emanations can be *direct* or *unintentional*. Direct emanations are caused by *intentional* current flows, which often consist of short bursts of current due to sharp rising edges and can be observed in a wide frequency band. Unintentional emanations are normally due to couplings. Harmonic-rich signals such as “square-wave” clocks and communication related signals contribute most to emanation via coupling and generate carrier signals for modulations. Data signals can be modulated over the carrier signal via Amplitude Modulation (AM) or Angle Modulation (FM or phase modulation). Observing direct emanation usually involves measurements of near-field signals, which may require chip depackaging. Carrier signals of unintentional emanation can have much better propagation and can be exploited more easily and effectively. Due to the close correlation between the currents flowing through the target device and the associated EM field, the measured EM traces often carry similar information as power traces. The power analysis techniques therefore are often also applicable to electromagnetic analysis. Electromagnetic emanations however contain even more information than power variations and therefore may enable more powerful attacks. Unlike in power analysis where the measured current is the overall current of all components in the device, the electromagnetic field of the device indeed contains multiple channels, which may enable the isolation of effects from different components.

The first published attack based on electromagnetic analysis (EMA) was introduced by Quisquater et al. in [127] and further improved in [128-129]. Quisquater et al. showed that with a simple flat coil, an attacker is able to measure the electromagnetic emanations produced by a smart card. Similar techniques as in power analysis were used in their attacks, referred to as Simple EMA (SEMA) and Differential EMA (DEMA). In [130], Mangard showed his near-field EM attack with a simple handmade coil and also demonstrated that far-field EM measurements of the power supply unit enabled the recovery of the secret key. Carlier et al. presented an EM attack on an FPGA implementation of AES and described a new way of retrieving some secret information [131]. In [132], Agrawal et al. proposed multi-channel attacks, which combine multiple side channels of the same or different kinds, including EM channels, power channels, etc. In [133], Quisquater et al. combined electromagnetic analysis and power analysis and were able to identify instructions executed by a processor based on a dictionary of instructions and their power/electromagnetic traces. Electromagnetic emanations were also exploited in retrieving information from computer displays including CRT as well as flat-panel displays [134-135]. It is worth noting that although most existing work on EMA employed similar techniques as power analysis, in the future the rich information contained in electromagnetic field can be further explored and enable new attacks.

2.2.3.4 Other attacks

Other side channel information can also facilitate side channel cryptanalysis. In 2004, Agrawal et al. demonstrated attacks based on acoustic emanations of computer keyboards, telephone and ATM keypads [136]. The key being pressed can be recognized by differentiating the sound produced by different keys. Acoustic emanations were also employed in analyzing noise generated by computers and allowed attackers to learn CPU behavior [137]. In [134], Kuhn presented techniques that retrieve information from diffuse visible light of CRT displays.

Fault analysis attacks sometimes are also discussed in the context of side channel attacks although they do not rely on the leakage due to physical side channel information. The basic idea of fault attacks is to induce faults into the target device during its operation and observe the erroneous output. Depending on the implementation, faults can be introduced transiently or permanently by manipulating power supply voltage, clock and device temperature, applying radiation or light to the device, or exploiting eddy current [5, 138-144]. Fault attacks have been successfully applied to DES [145-147], AES [148-152], stream ciphers [153-154], RSA-CRT [155-157], ECC [158-159], and modular exponentiation-based cryptosystems [160-161].

Information leakage via address bus sometimes is also regarded as a form of side channel. Knowing the address trace of a program, e.g., the one exposed on a computer's address bus, allows an attacker to learn the internal state of the program, enabling attacks on copyright-protected software and ciphers [162-166].

2.2.4 Countermeasures

2.2.4.1 Countermeasures of power analysis and electromagnetic analysis

We review countermeasures of power analysis and EM analysis together since electromagnetic emanation is essentially a product of current flows. The countermeasures of power analysis thus are usually also effective on EM analysis.

In the literature, most of the countermeasures are based on the following ideas: removing variations, hiding dependence, randomization, and blinding or masking.

Software countermeasures: As suggested in [167], making the execution flow of a crypto implementation as constant as possible can help mitigate SPA – it reduces the major portion of the power variation [155, 168]. In the case of RSA, inserting dummy multiply operations in the square-and-multiply implementation and the balanced Montgomery powering ladder [169] are examples of such. Block cipher implementations tend to have fewer or no branches and their execution path can be made constant more easily. SPA can also be mitigated by hiding dependency, which makes it harder for the attacker to reconstruct the secret even when knowing the operations performed. The use of sliding window techniques [170] and m-ary RSA [171-172] were suggested for this purpose. Other exponent recoding schemes were suggested by Walter [173-174]. Randomization can help mitigate both SPA and DPA. The randomized algorithms can make it harder to identify target patterns in a single trace – mitigating SPA, or make the target power characteristics random among different traces – mitigating DPA. Randomized implementations of RSA as well as ECC can be found in [174-184]. Techniques that randomize variations among traces however do not stop attacks that can recover secrets in a single trace [77, 185]. Masking the internal states of the computation, i.e., preventing the attacker from predicting such states, which is the basis of DPA, can solve this problem. In [186] and [76], techniques that divide each bit of the original computation into two statistically independent shares were proposed. DPA relying on the prediction of the original internal bit therefore will not succeed. These methods however were proven vulnerable to high-order DPA [73-75]. An alternative approach is blinding or masking by combining input with random numbers. The techniques [187-188] for blinding signatures are good examples of this [103]. Other masking techniques can be found in [189-194].

Hardware countermeasures: The idea of randomization can be easily applied at the processor architectural or micro-architectural level. Random register renaming [195] and random code injection [196] was proposed to randomize the power variations. Masking is often adopted in logic or cell level circuit design. A theoretical work on gate level masking was presented in [197]. Various implementation work such as multiplexor-based circuit, correction-term-based circuit as well as other circuit styles can be found in [198-202]. Removing power variations was mostly done at cell level by making the power consumption of the cell data-independent. The major logic styles proposed include asynchronous circuit [203-204], Dual-Rail Precharge (DRP) circuits [205-208], and Current Mode Logic (CML) circuits [209-211]. There are also other hardware countermeasures in addition to the above methods. Power measurement noise can be increased by using a random number generator [212]. Signal suppressing techniques were proposed in [213]. The detachable power supply technique was presented in [214]. To defend against EM analysis, the use of metal shield layer and random number generator was suggested to reduce the EM field and make it noisy [128].

2.2.4.2 Countermeasures of timing analysis

The concepts of removing variations, randomization, and blinding/masking are also applicable in defending against timing analysis. To remove timing variations, in addition to the work on the defense of power analysis that make the execution path constant [169], the timing variations due to the reduction operations in Montgomery algorithm [104] was also considered. Dhem [215] proposed improved multiplication schemes that allow chaining of several modular multiplications with only one extra reduction, thus removing most timing variation. Similar work were also presented by Walter [216-217] and Hachez et al. [218]. The issues of implementing constant time block ciphers were discussed in [106]. Randomization techniques for power analysis protection [174-184] are also helpful in preventing timing attacks. Randomized algorithms randomized power variations as well as timing variations. Blinding techniques are particularly effective in defending against timing attacks. Kocher in his first paper on timing attacks [103] had suggested the use of blind signatures [189-194] as an effective countermeasure. The countermeasures of cache-based attacks are reviewed separately in the next section due to the uniqueness of cache attacks.

2.2.4.3 Countermeasures of cache-based attacks

The area of cache-based attacks is young and still rapidly evolving. Although various intuitive ideas have been suggested, the application of them is not trivial, and many of them are still not carefully investigated.

Software countermeasures: Page [110], Bernstein [114] and Osvik et al. [118-119] suggested several conceptual countermeasures to mitigate cache attacks. The first approach is to avoid memory accesses so that the security of the cipher is irrelevant to caches. Specific techniques include replacing table lookups with logical and arithmetic operations, putting tables in registers (if the architecture has a sufficient amount registers), or using implementations such as the bitslice scheme [219]. Making the memory accesses data-oblivious [162-163] can also mitigate the attacks. Observing a statically or statistically fixed memory access pattern would reveal no useful information to the attacker. Software masking may also be helpful since it would prevent the attacker from knowing the internal states of the cipher. Pre-loading tables, dynamically moving tables around, and hiding timing were also suggested, but with comments on their obvious limitations and nontrivial applications. Brickell et al. [220-221] proposed several implementations of AES based on three mitigation strategies: (1) compact S-box tables; (2) table randomization; and (3) pre-loading of relevant cache lines. These strategies together with selective round protection can enable various combinations for different security-performance trade-offs. Software mitigations for RSA implementations were proposed in [221]. Two methods for the binary implementations were presented, including conditional branches elimination and the replacement of all squares with multiplies. The attacks against the fixed window and sliding window implementation were mitigated by storing the pre-computed multiplier table in an interleaved manner. The bytes of each table entry are distributed to all cache lines in the table. The access to any multiplier therefore leads to accesses to all cache lines, resulting in fixed access pattern.

Hardware countermeasures: Conceptual countermeasures were suggested by Page [110], Bernstein [114] and Osvik et al. [118-119], including: (1) disabling cache sharing; (2) static or disabled cache; (3) larger cache lines; (4) non-deterministic cache placement; and (5) hardware masking. Percival [121] suggested not allowing one process to evict cache lines of another. These ideas however were not investigated in detail. As the first dedicated work, the Partitioned cache [222] was proposed by Page to mitigate cache based attacks. With instruction set architecture (ISA) extensions, private cache partitions can be formed for the protected processes or software modules, with the ability of reconfiguring cache line sizes as well as address masks. The author however admitted that the overhead of the architecture – in terms of both performance and hardware cost – could be high, and it might not be suitable for high clock rate processor design. Trade-offs have to be made among performance, cost and security.

2.3 Information Hiding

In the most general sense, covert channels, steganography, anonymity, and watermarking are all forms of information hiding [223-225]. This section reviews steganography and watermarking techniques, which hide information in ordinary messages such as media files or network packets. In the context of information leakage, information hiding allows undetectable information transfer covered by legitimate communications.

2.3.1 Steganography and Watermarking Basics

2.3.1.1 Steganography

Steganography in Greek means "cover writing". It is the art and science of hiding information by embedding messages in other seemingly harmless messages [226]. Unlike cryptography which aims to hide the *content* of a message, the goal of steganography is to hide the *presence* of the message. In other words, steganography allows secret transfer of information while no one else can detect the very existence of the communication.

According to the terminology used in [223], the original object in which the message is embedded is referred to as the *cover-object*, e.g., cover-text, cover-image, cover-music, etc. The message to be embedded is referred to as the *stego-message*. The object with embedded message is called the *stego-object*. In some cases, the sender/receiver needs a secret to embed/retrieve the message. The secret is referred to as the *stego-key*. Figure 2-1 illustrates the generic embedding/retrieving process. Note that stego-keys k_1 and k_2 may or may not be the same key, depending on whether a symmetric-key or public-key system is used. Also note that the cover-object c at the decoding end is optional, indicated by a dashed line. Some applications do not require the original object to recover the message.

Three properties are usually discussed in steganographic systems: transparency, capacity and robustness. Transparency describes the similarity between the cover-object and the stego-object, or the imperceptibility of the embedded information. Capacity measures the amount of the information that can be embedded in the cover-object without compromising transparency. The robustness of the embedding scheme in the context of steganography is often related to the security of the steganographic system. Depending on the application, it is relevant to the detectability of the presence of the stego-message, the retrievability of the stego-message, and the resistance of overwriting, removing or disabling the stego-message in the stego-object.

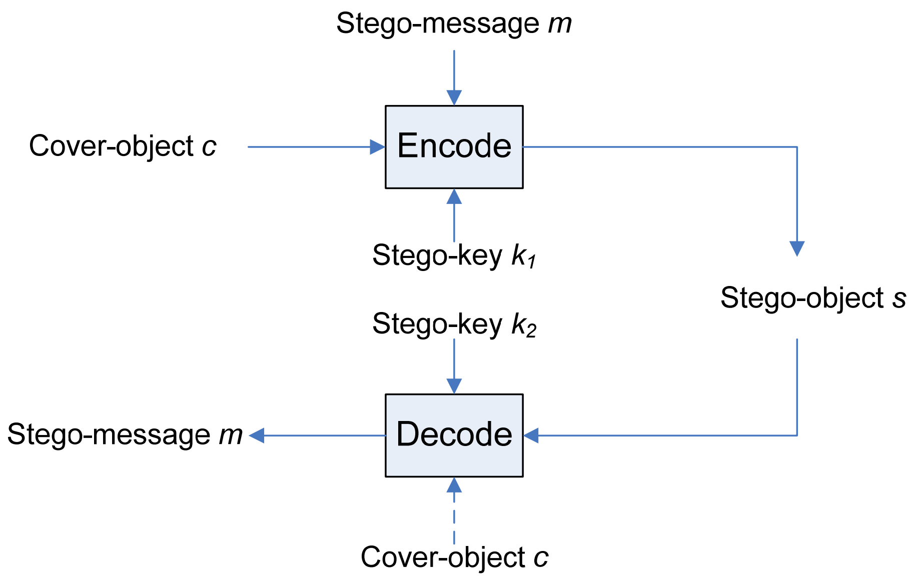


Figure 2-1. A generic process of message embedding and retrieving

2.3.1.2 Watermarking

Watermarking is closely related to steganography but has slightly different goals. Unlike steganography, the embedded message is usually related to the cover object [227], e.g., the mark indicating the creator of an art work. In practice, watermarking is more relevant to the protection of mark tampering or removal rather than the protection of message detection.

Watermarks can be *perceptible* or *imperceptible*, *public* or *private*, and *robust* or *fragile*, as explained in the following applications [228-229]:

- Copyright watermarks: By embedding information about the creator or owner of the object, watermarking provides a way for securing the ownership rights or proving the ownership. Copyright watermarks should be robust, meaning that they are hard to remove, and should be still detectable even when the stego-object has been modified considerably. Copyright watermarks can be either perceptible or imperceptible.
- Fingerprint watermarks: Fingerprint watermarks can be used to track and trace copies of an original work, such as copyright protected images and movies. For

example, unique fingerprints such as serial numbers can be embedded into each copy when distributing it to the customer. The fingerprint has to be imperceptible, robust and private. Private means that only a select group – the distributor in the above example – can detect or extract the watermark.

- Broadcast watermarks: Broadcast watermarks can be used as a copy control mechanism and allow copyright protection to be built into software and hardware devices. For example, copy control can be achieved by detecting a watermark and invoking proper software or hardware operations such as enabling or disabling the record module. Broadcast watermarks should be imperceptible, public and robust.
- Annotation watermarks: Metadata can be embedded into the object itself using watermarking techniques. For example, date, location, author's information, and search keywords etc. can be embedded into the image itself. Such watermarks should be imperceptible, public and robust.
- Integrity watermarks: Watermarks can also be used to ensure integrity of the cover-object. Such watermarks should be fragile, i.e., tiny modification of the original object would lead to damage of the embedded watermark. Integrity watermarks can be either perceptible or imperceptible, public or private.

2.3.2 Data Hiding Techniques

Despite the fundamental philosophical differences between steganography and watermarking, these two fields share many of their underlying technical approaches, which are briefly reviewed below.

Substitution techniques: Many cover-objects, e.g., images and audio files, contain considerable redundancy. Substitution techniques usually involve replacing some of the redundant parts, e.g., the least significant bit (LSB) of the original object [230], with the secret message. Substitution techniques can produce imperceptible watermarks and have high capacity. The embedded watermarks however are not robust.

Transform domain techniques: The transform domain techniques embed information in a transform space, such as DCT [231-233], Wavelet [234-236], and DFT [237-239], and can be naturally integrated with popular compression techniques such as JPEG and JPEG 2000. These techniques overcome the robustness problem of substitution techniques while still produce imperceptible watermarks.

Spread spectrum techniques: Spread spectrum techniques used in communication systems can also be applied to embedding schemes [240-241]. By distributing the information in a much wider spectrum, the resulting watermark is much more resistant to attacking techniques such as filtering and lossy compression, leading to better robustness.

Statistical methods: Statistical methods embed information by changing the statistical properties of the cover-object and use hypothesis testing to retrieve the embedded information [242-244]. In the binary case, a single bit of information can be embedded into the cover-image by changing statistical distribution of luminance values in the set of pseudo-randomly selected pairs of image pixels.

Distortion techniques: Information can be also embedded by distorting the cover-object and measuring the deviation from the original object for secret extraction. Distortion techniques are commonly used in hiding information in text files. For example, the line-shift coding displaces a whole line of text by a small amount, e.g., 1/300 inch, to indicate a ‘1’ bit. Similarly, word-shift coding and character coding can also be used to encode information [245].

Cover generation methods: Cover generation methods generate an object based on the secret only for the purpose of being a cover-object. Techniques such as the Mimic functions [246] can be used to hide the presence of the message by making the statistical characteristics of the generated cover-object match those of an innocent looking text.

2.4 Miscellaneous Unintended Data Exposure

Unintended data exposure in this dissertation refers to the accidental exposure of information, which allows unauthorized users to gain direct access to the protected data. It is often due to various implementation bugs and design flaws and is one of the most common types of information leakage problems in computer systems. Due to the huge number of vulnerabilities, this review illustrates the problem with only representative examples. More complete lists of known vulnerabilities can be found in the US-CERT data base [247] as well as various security online bulletins [248-249].

At the application level, implementation bugs, bad architecture design and default program settings and behaviors can all lead to information leakage. An example of the implementation bugs is the JavaScript bug in Mozilla and Firefox web browsers. It allows an arbitrary amount of heap data to be leaked out to a malicious website [250]. Another example is a bug in Microsoft Word that can cause a document to contain hidden data that are from another completely unrelated document [251]. If two documents were open using the buggy version of Word, saving one of the documents would lead to the inclusion of text from the other. In addition to implementation bugs, programmers’ lack of understanding of security requirements lead to bad design and introduce inadvertent information leakage as well. For example, crypto building blocks such as AES and RSA are often misused [252], leading to leakage of critical information even though the ciphers themselves are strong. Applications’ default settings and behaviors may also lead to information leakage. For example, Word documents usually contain hidden data that most ordinary users are unaware of, including names and usernames of the documents’ creators, pathnames of the documents, text that were already deleted, etc. Such hidden data have enabled an attacker to obtain a significant amount of sensitive information by simply examining published documents [251].

Unintended data exposure at the system level is usually due to bugs in system software, and can appear in various forms. For example, information can be leaked out through memory: bugs in kernel software [253] can expose content of kernel memory to user space applications. Error reporting and logging systems can also be exploited. Core dump files may contain sensitive data (e.g., user’s password), and can be accessed by unprivileged users or even remote users [254-257]. Logs and session files have similar problems [258]. File systems can also be problematic. A few versions of the ext2 file system leak kernel memory data to disk when creating new directories [259]. Paging mechanisms may swap out memory pages containing critical data to disks (though not due to bugs) [260], which can be examined by an attacker at a later time.

At the hardware level, information can be leaked out through storage devices such as hard drives and memories. An attacker can collect surplus hard drives which still contain a large amount of information and recover sensitive data from them. Even if the storage devices have been cleared before they are released to untrusted parties, e.g., by filling the disk with zeros, information can still be recovered [5]. In addition to non-volatile storage devices, volatile memory can also cause information leakage. Contrary to most people's belief, solid-state Random-Accessed Memory (RAM) can retain its data even after being powered off. Peter Gutmann [5, 141] examined the "burn-in" effects that occur in both static RAM (SRAM) and dynamic RAM (DRAM). When a memory cell stores the same value for a relative long time, the physical attributes of the semiconductor devices may change and leave trace of the stored value. Information may remain in memory even if the data are only momentarily stored. Recent work on Cold Boot Attacks [261] showed that data can remain in memory for seconds to minutes after being powered off, and this time can be extended to hours by cooling the memory modules.

The mitigation of unintended data exposure due to software bugs is mostly *ad hoc*: for each particular bug, a patch is issued and the information leakage channel is blocked. Chow et al. studied the data leakage problem in a more general manner, by examining the data lifetime in a system [258]. They also proposed secure deallocation mechanisms to reduce data lifetime and thus the risk of unintended data exposure. To avoid information leakage through paging system, Provos [260] proposed to encrypt virtual memory such that all pages swapped out to disk are encrypted, thus meaningless to unauthorized users. Secure processor architectures [262-267] that support encrypted memory also help mitigate various attacks that rely on information leakage via memory.

2.5 Scope of this Dissertation

The information leakage problem can be discussed in a space spanned over the three aforementioned dimensions: whether the leakage is via direct information exposure or indirect interference, whether the leakage is intentional or unintentional, and at which level the leakage occurs. Covert channels are intentional information leakage and mostly due to indirect interference rather than direct exposure. Side channels are similar to covert channels except that they are unintentional leakage and mostly due to physical leakage at the hardware level. Information hiding techniques such as steganography can also be exploited for intentional information leakage. These techniques however hide information in legitimate messages rather than exploit indirect interference as in covert channels. Unintended data exposure due to system or software vulnerabilities belongs to unintentional leakage, and the data leaked out are directly exposed to the attacker.

The focus of this work is on the mechanisms that allow information-leaking interference in microprocessors. It is therefore more relevant to covert channels and side channels. Unlike past work in these two areas, we particularly focus on architectural or micro-architectural level information leakage rather than information leakage at the software level or physical circuit level. Such mechanisms allow covert channels that are much faster than traditional ones, and enable side channel attacks on embedded devices as well as general purpose systems. We consider countermeasures to the architectural-level attacks. We also propose a new model for covert channels, and new results on covert channel capacity estimation.

Chapter 3

Cache-based Side Channel Attacks: Analysis and Countermeasures

3.1 Introduction

Protecting sensitive information within computers and over networks is a major concern of users of computer systems. To achieve this goal, cryptographic methods are widely deployed in platforms ranging from simple embedded devices to complex server systems. The cryptographic primitives are designed to be mathematically strong such that even if the adversary gets hold of the encrypted data, it is computationally infeasible to infer the original data or the secret crypto key by brute-force trials, or even by differential cryptanalysis [145] and linear cryptanalysis [268]. However, side-channel attacks make use of auxiliary side channel information rather than mathematical analysis to deduce key bits, and can easily break even mathematically strong ciphers.

In the past, side channel attacks were mostly used in attacking simple devices such as smart cards rather than more complicated general purpose systems, due to the noisy nature of the side channel information, the difficulty in collecting such information and the need for physical access or proximity. The recent cache-based attacks however can impact a much wider spectrum of systems and users. This is because caches exist in almost all modern processors, and the attacks are effective on various platforms [114, 118, 121] and can be pure software attacks which are very easy to perform. This makes cache-based side channel attacks extremely attractive as a new weapon in the attacker's arsenal.

Existing mitigations of cache attacks are mostly software approaches, which typically involve rewriting the software implementations such that they are not vulnerable to known attacks. These software countermeasures however are cipher-specific and only effective for known attacks. Due to more restrictive designs, significant performance degradations are commonly observed in such "secured" software implementations. Moreover, some software mitigations are based on empirical ideas and do not provide sufficient security. Hardware countermeasures are also discussed in the past. Despite various conceptual ideas, applying them in practice is not trivial and few of these ideas were fully investigated.

In this chapter, we first identify the root causes of cache attacks and clarify the following questions that have not been answered in the past: can the problem of cache-based attacks be solved by software or hardware alone, or what is the proper partition of work for software and hardware in mitigating cache based attacks? Based on these results, we then propose effective solutions by attacking the root causes of cache-based side-channel attacks. We also strive to achieve security without compromising performance, power efficiency or other cache design goals.

3.2 Attack Analysis

3.2.1 Information Leakage in Caches and Cache-based Attacks

Cache hits and misses leak information. For example, a program's memory reference traces may exhibit different cache hit and miss behavior, causing variations in the program's execution time or power dissipation. Such variations allow an attacker to infer information about the program's internal states. In addition to the interference in a single program, cache accesses from different processes using a shared cache may interfere with each other and allow one process to infer information about another. Such cache interference leaks information and makes caches susceptible to side channel attacks.

Traditionally, cache-based side channel attacks were categorized into trace-driven attacks, time-driven attacks [110] and recently access-driven attacks [111]. The difference between these types of attacks is the attacker's ability in observing the victim's memory references. In trace-driven attacks, the attacker is able to detect the outcome of each memory reference of the victim in terms of hit or miss. In time-driven attacks, the attacker can only see an aggregated profile, e.g., the total number of hits or misses. In access driven attacks, the attacker is able to know which cache sets have been touched, individually or in an aggregated manner.

Although in past work the difference in timing between cache hits and misses has been recognized as the source of information leakage [114, 118, 121], in this work we further distinguish the type of information leakage achieved via cache hits versus that achieved via cache misses, since they have different impact on cache design. The difference between these two information leakage mechanisms is that cache misses involve interference between references to two different memory blocks – one replacing the other in the cache, whereas cache hits only involve the same block – a former access to a block can interfere with subsequent accesses to the same block, by causing them to hit in the cache. Below we categorize existing attacks in terms of how interference due to cache misses and cache hits is exploited and analyze a representative attack in each category. We discuss in turn:

- Internal interference due to cache misses
- External interference due to cache misses
- Internal interference due to cache hits
- External interference due to cache hits.

The implications on cache designs and possible countermeasures are then discussed.

```
For key K:
For s = 1 to N do begin
    Generate a random 128-bit Plaintext block, $P_s$;
    $T_s$ = time taken for AES encryption of $P_s$ using K;
end;
For i = 0 to 15 do begin
    For j = 0 to 255 do begin
        count = 0;
        For s = 1 to N do begin
            If $p_i = j$ then
                $TSUM_i(j) = TSUM_i(j) + T_s$;
                count = count+1;
        end;
        $t_{avg}^i(j,K) = TSUM_i(j)/count$;
    end;
end;
```

```
For i = 0 to 15 do begin
  For j = 0 to 255 do begin
    $Corr[j] = \sum_{m=0}^{255} \left[ t_{avg}^i(m,K) \bullet t_{avg}^i(m \oplus j, K') \right]$
  end;
  $k_i$'= findMax(Corr);
end;

Note: Function findMax() searches for the
maximum value in the input array and returns its
index.
```

Figure 3-1. (a) Timing characteristic generation (b) Key-byte searching algorithm

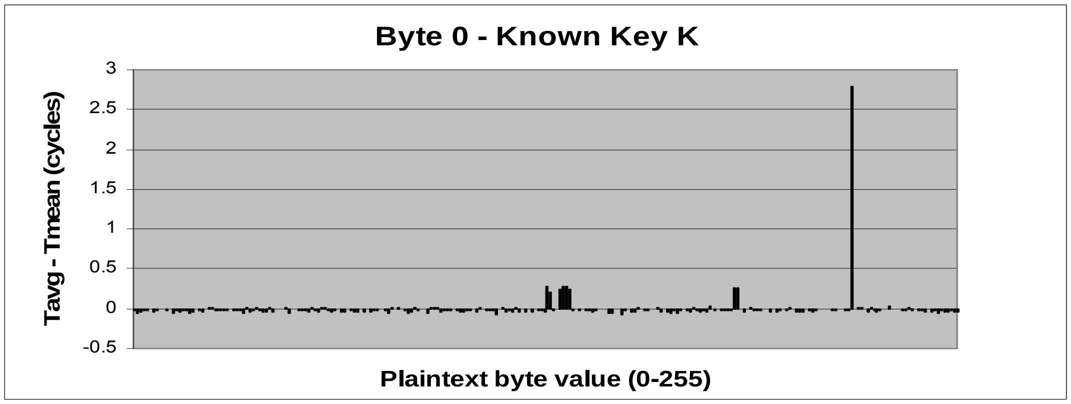
3.2.1.1 Internal interference due to cache misses: Bernstein's Attack

The first class of attacks we discuss is due to cache interference that comes from the victim's code itself. Furthermore, the interference is caused by cache misses rather than cache hits. Hence, we call this *internal interference due to cache misses*. Bernstein's attack is representative of this class of attacks.

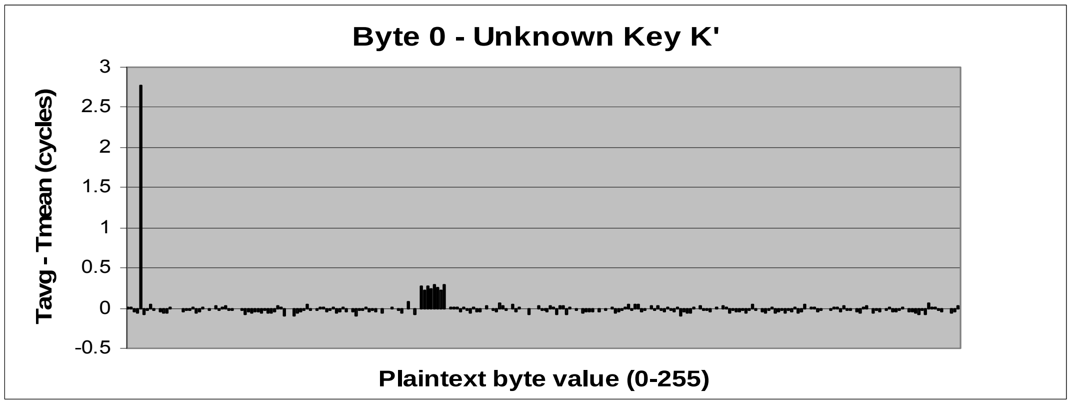
Attack description: Bernstein's attack is a time-based attack. The victim of the attack is a software module that can perform AES encryption. The module is a "black box" to the attacker. The attacker is able to choose the input to the victim and measure how long it takes to complete the encryption. The attacker may be a process in the same machine with the victim, or a remote user requesting encryption service. Empirical studies show that for most software AES implementations running on modern microprocessors, the execution time of an encryption is input-dependent and can be exploited to recover the secret encryption key. The attack consists of three steps:

1. *Learning phase*: Let the victim use a known key ***K***. The attacker generates a large number, *N*, of random plaintexts ***P***. He sends the plaintexts to the cipher program and records the encryption time for each plaintext. He uses the algorithm shown in Figure 3-1 to obtain the timing characteristics for ***K***, shown in Figure 3-2(a).
2. *Attacking phase*: Repeat step 1 except that an unknown key ***K'*** is used. The timing characteristics for ***K'*** is shown in Figure 3-2(b). Note that the input set is randomly generated and not necessarily the same as that used in step1.
3. *Key recovery*: Given the two sets of timing characteristics, use the correlation algorithm shown in Figure 3-1(b) to recover the unknown key ***K'***. As explained below, the timing characteristic charts for different keys, e.g., Figure 3-2(a) and Figure 3-2(b), should be the same except that the locations of the bars in the charts are permuted. The correlation algorithm simply tries all 256 possible permutations (each of which corresponds to a value of *j*) and finds the one that would permute Figure 3-2(b) into Figure 3-2(a).

In Figure 3-2, the height of the bar at position *j* is $t_{avg}^i(j,\boldsymbol{K})$, which is the average of the execution time of the AES encryptions when the value of the *i*-th byte of plaintext ***P*** is *j*,



(a) Average encryption time for byte 0 with known key K



(b) Average encryption time for byte 0 with unknown key K'

Figure 3-2. Timing characteristic charts for byte 0 (obtained on a Pentium-M machine)

using key $\boldsymbol{K}$ (for visual clarity, a constant, denoted as T_{mean} which is the mean value of $t_{avg}^{i}(j,\boldsymbol{K})$ for all j, is subtracted from $t_{avg}^{i}(j,\boldsymbol{K})$ when plotting the figure). In the AES algorithm, each plaintext $\boldsymbol{P}$ is an M-byte block, e.g., M=16, therefore M pairs of such timing characteristic charts are generated. Figure 3-2 only shows one such pair, corresponding to byte 0 in $\boldsymbol{P}$. Experiments show that $t_{avg}^{i}(j,\boldsymbol{K})$ is pretty much fixed for a given system configuration. Furthermore, it is found that when a different key $\boldsymbol{K'}$ is used, the timing charts roughly remain the same except that the locations of the bars in the charts are permuted, as shown in Figure 3-2. More specifically, equation (3.1) holds:

$$t_{avg}^{i}(p_i, \boldsymbol{K}) = t_{avg}^{i}(p'_i, \boldsymbol{K'}) \text{ if } p'_i \oplus k'_i = p_i \oplus k_i \qquad (3.1)$$

where $\oplus$ is the bit-wise XOR operation, and k_i and k'_i are the i-th byte of $\boldsymbol{K}$ and $\boldsymbol{K'}$ respectively.

Attack analysis: Bernstein's attack itself does not show what actually causes the information leaking timing characteristics. This is actually due to the cache miss behavior of the memory references corresponding to the table lookups that are used in various software AES implementations. The following analysis assumes the OpenSSL v0.9.7a implementation and can be applied to other implementations as well. The software AES cipher in OpenSSL v0.9.7a uses five tables, four for the first 9 rounds of operations and one for the last round, which is irrelevant to the attack. During the encryption, for each byte p_i of the plaintext, one of the four tables is accessed using the index ($p_i \oplus k_i$) where k_i is the *i*-th byte of the encryption key. Ideally, these table lookups will hit in the cache since normally the cache is large enough to accommodate all these tables. However, in reality it is found that there are always other memory accesses that regularly contend for cache lines at some fixed locations and cause evictions of corresponding table entries. Therefore, given an index ($p_i \oplus k_i$), if the corresponding table entry is mapped into one of these "hot" cache locations, the table lookup will have a higher probability to experience a cache miss due to the evictions caused by the contending memory accesses. This will lead to larger $t_{avg}^{i}(p_i, \boldsymbol{K})$, i.e., a high bar in Figure 3-2(a). When a different encryption key $\boldsymbol{K'}$ is used, the same analysis applies and the resulting timing characteristics charts should be the same except that the locations of the bars are permuted. This is because given an arbitrary value p_i for the key-byte k_i, there is always such a value p'_i for key-byte k'_i that generates the same index, i.e., $p'_i \oplus k'_i = p_i \oplus k_i$. The table lookup with the same index would share the same timing characteristics. Therefore, a bar at location p_i in Figure3-2(a) will also appear in Figure 3-2(b), but at location $p'_i = p_i \oplus k_i \oplus k'_i$.

The evictions of AES table entries can be the result of memory references of other processes as well as the victim AES process itself. To be useful, such evictions have to be regular and also consistent during the learning phase and the attacking phase of the attack. Since in many cases other processes are unrelated to the victim process, they do not generate regular evictions at fixed locations (relative to the location of the AES tables) or do not produce consistent cache eviction characteristics during the learning phase and the attacking phase. In such cases, cache interference from other processes contributes little to Bernstein's attack. In contrast, memory references from the same process – possibly from code segments other than the AES cipher code – can cause more robust cache interference during both learning and attacking phases. In our experiments, the observed common sources of interfering memory references include the wrapper code of the core AES encryption engine and the stack adjustment instructions of the user function that contains the AES cipher. Such internal interference allows the attack to succeed even if the cipher runs alone without being interfered by any other process. For this reason, we consider Bernstein's attack as a representative attack that is based on *internal interference due to cache misses*.

3.2.1.2 External interference due to cache misses: Percival's Attack

Another class of information-leakage attacks is due to cache interference from other processes. For example, processors supporting Simultaneous Multi-Threading (SMT) allow multiple processes to run simultaneously on the same chip, sharing the cache system. A process (e.g., the victim process) therefore can evict cache lines holding data of another (e.g., the attacker process), causing it to miss on these cache lines. This gives the attacker the ability to observe the victim's cache access behavior (i.e., which cache lines/sets are touched) and obtain a relatively accurate access trace. Percival's attack is a representative attack for this class of attacks.

Attack description: Percival's attack was demonstrated on an Intel processor with HyperThreading (HT) technology. In Percival's attack, the attacker manages to launch a process running simultaneously with the victim process, i.e., the process that performs RSA encryption. His goal is to discover the private encryption key used by the victim. The attacker sequentially and repeatedly accesses an array, thus loading in his own data to occupy all cache lines. During accessing the array, he also measures the delay for each access to detect cache misses, e.g., using the `rdtsc` instructions to read a timer in Intel x86 processors. The victim's memory accesses will cause evictions of the attacker's data, and when the attacker accesses the evicted data in his next round of array access, he will miss on these cache lines and observe longer delays. In this way, the attacker is able to obtain a figure that accurately shows the evolution of the victim's footprint in the cache, which allows him to extract information about the internal states of the RSA encryption.

Attack analysis: The core operation used in RSA is modulo exponentiation. It is often implemented with a series of squarings and multiplications. The secret encryption key is divided into segments of multiple bits, each of which is associated with a number of squarings followed by a multiplication. For each multiplication, a multiplier is selected from a set of pre-computed constants – stored in a table, and the key segment is used as the index of the corresponding table lookup. With the ability to observe the victim's footprint in the cache, the attacker can obtain the following information: (1) the attacker is able to identify every squaring and multiplication due to the different footprints of these two sub-operations; (2) for each identified multiplication, the associated table lookup can be identified, i.e., the attacker is able to know which cache line is accessed, thus knowing which table entry is accessed. Obtaining the squaring and multiplication chain of an RSA encryption in some implementations (e.g., the sliding window implementation used in OpenSSL) allows the attacker to know some information (e.g., the Hamming weight) about the secret key segments. Even if safer RSA ciphers such as the fixed window implementation are used such that no information can be extracted from the squaring and multiplication chain, knowing which table entry is accessed during a multiplication directly allows the attacker to learn the index used in the table lookup – the index being the secret key segment.

Clearly, Percival's attack is based on the cache misses caused by other processes: the victim's memory accesses cause the attacker process to miss in the corresponding cache lines, thus allowing the attacker to observe the victim's footprint in cache. Hence, it is a case of external interference due to cache misses.

3.2.1.3 Internal interference due to cache hits: the cache-collision timing attacks

Unlike in Bernstein's attack and Percival's attack where a cache miss indicates the occurrence of cache interference, in the cache-collision attacks, cache interference leads to cache hits instead. The cache collision attacks are similar to Bernstein's attack in the way that the attacks are launched. During the attack, the victim cipher (AES in this case) is also considered a black box and the measurements of the encryption times of a large number of random messages are the only thing that the attacker is able to do. With the knowledge of the time of each encryption and either the plaintext or the ciphertext involved in the encryption, the attacker is able to recover the secret encryption key. Note that the collision attacks do not require a learning phase, i.e., the attacker does not need to possess a system identical to the target system.

Attack analysis: Unlike Bernstein's attack which is based on the empirical observation of timing variations without exploiting any information about how such timing variations are caused, the cache collision attacks make full use of the cipher structure to design the attack. In particular, the cache collision attacks exploit the cache interference caused by the AES encryption code itself. To illustrate the attack, we use the OpenSSL v0.9.7a as an example. The cache interference exploited in the attacks is due to table lookups in the first round or the last round of the AES encryption. Assuming a 128-bit (16-byte) wide cipher, each round operation requires 16 table lookups, as shown in equation (3.2) except for the last round.

$$\begin{aligned}
\mathbf{x}_0^{(r)} &= T_0\left[x_0^{(r-1)}\right] \oplus T_1\left[x_5^{(r-1)}\right] \oplus T_2\left[x_{10}^{(r-1)}\right] \oplus T_3\left[x_{15}^{(r-1)}\right] \oplus \mathbf{K}_0^{(r)} \\
\mathbf{x}_1^{(r)} &= T_0\left[x_4^{(r-1)}\right] \oplus T_1\left[x_9^{(r-1)}\right] \oplus T_2\left[x_{14}^{(r-1)}\right] \oplus T_3\left[x_3^{(r-1)}\right] \oplus \mathbf{K}_1^{(r)} \\
\mathbf{x}_2^{(r)} &= T_0\left[x_8^{(r-1)}\right] \oplus T_1\left[x_{13}^{(r-1)}\right] \oplus T_2\left[x_2^{(r-1)}\right] \oplus T_3\left[x_7^{(r-1)}\right] \oplus \mathbf{K}_2^{(r)} \\
\mathbf{x}_3^{(r)} &= T_0\left[x_{12}^{(r-1)}\right] \oplus T_1\left[x_1^{(r-1)}\right] \oplus T_2\left[x_6^{(r-1)}\right] \oplus T_3\left[x_{11}^{(r-1)}\right] \oplus \mathbf{K}_3^{(r)}
\end{aligned} \tag{3.2}$$

where $\mathbf{x}_i^{(r)} = \left(x_{4i}^{(r)}, x_{4i+1}^{(r)}, x_{4i+2}^{(r)}, x_{4i+3}^{(r)}\right)$, i = 0, 1, 2, 3, is the i-th state word of round r (a word contains 4 bytes). T_0 through T_3 are four AES lookup tables with 1 byte input and 1 word output. $\mathbf{K}_i^{(r)} = \left(K_{4i}^{(r)}, K_{4i+1}^{(r)}, K_{4i+2}^{(r)}, K_{4i+3}^{(r)}\right)$ is the i-th word of the r-th round key. In the first round, the state byte $x_i^{(0)}$ is indeed $p_i \oplus k_i$, where p_i is the plaintext byte and k_i the original key byte, i = 0, … , 15. As shown in (3.2), each table is shared by four state bytes for their table lookups. The cache collision attacks are based on the interference between table lookups accessing the same table. To clarify, a collision occurs if two table lookups access the same table entry. If the AES table initially is not present in cache, the first table lookup would make the second table lookup hit in cache. Colliding table lookups therefore should have shorter execution time than non-colliding table lookups. Since the index used to access a table is the state byte, a collision occurs if

$$p_i \oplus k_i = p_j \oplus k_j \quad or \quad p_i \oplus p_j = k_i \oplus k_j \tag{3.3}$$

for all i and j values where table lookups for the i-th and j-th bytes of the state look up the same AES table. To exploit such equations, the average execution times t(i,j,Δ) are computed for all qualifying i and j over all plaintexts that satisfy $p_i \oplus p_j = \Delta$, Δ = 0, …, 255. According to (3.3), the Δ value that leads to the minimum average execution time is the difference between the key bytes k_i and k_j. Since each AES table is shared by four state bytes, the difference between each pair of the four key bytes can be recovered. This essentially means the discovery of 3 bytes of the key. For example, bytes 0, 4, 8 and 12 share table T_0, and the attack can discover $k_0 \oplus k_4, k_0 \oplus k_8$ and $k_0 \oplus k_{12}$. The attacker can then do brute-force search to find k_0 and then obtain the absolute values of the other three key bytes. In the end, the attacker is able to discover twelve key differences and needs to do brute-force search to find the remaining 4x8=32 bits of the key. This attack can be further improved by exploiting the final round operation instead of the first round. Table lookups in the final round share one single table, allowing the recovery of the full key (with some more optimization techniques).

In summary, the cache collision attacks are based on internal cache interference – the interference caused solely by the AES encryption code itself. Furthermore, the cache interference is among memory references accessing the shared objects, one reference causing others to hit in the cache.

3.2.1.4 External interference due to cache hits

Although in theory, side channel attacks based on external interference due to cache hits are possible, to the best knowledge of the author, none of the existing attacks fall into this category. This is mainly due to the fact that the attacker and the victim rarely share objects. For security reasons, the victim is normally isolated from other processes, e.g., via address space isolation, sandboxing or virtual machines. Therefore, the situation where both the attacker and the victim access the same cache line, one access causing the other to hit in cache, would not occur.

Despite the low possibility of attacks based on external interference due to cache hits, caution is still necessary in practice. For example, due to the popularity of shared dynamic-linked libraries, an attacker process may share some library code with the victim. If the shared code operates on sensitive information (for example the two processes share the crypto library), an attack is still possible. The victim's instruction fetch can cause the attacker to hit in the I-cache, enabling the attacker to learn the victim's execution paths. In such situations, a simple yet effective solution is to disallow sharing of library code. In this dissertation, we assume the attacker and the victim are fully isolated unless otherwise specified.

3.2.1.5 Remarks

Although both miss-based interference and hit-based interference can be exploited in cache attacks, existing attacks all exploit either one of them, not both. To our understanding, this is because opposite assumptions are required for attacks that exploit these two different types of interference. Miss-based attacks (e.g., Bernstein's attack and Percival's attack) assume the data are initially in cache and expect hits when accessing them. Interference is expressed with "abnormal" evictions that cause cache misses. In contrast, hit-based attacks (e.g., the cache-collision attacks) assume that the data initially is not present in the cache, and the "abnormal" hits carry the information of interest. However, it is still worthy of further research to see whether smarter techniques exist that can combine both miss-based and hit-based interference.

3.2.2 Countermeasures and Implications on Cache Designs

From the perspective of a cache designer, miss-based information leakage is easier to mitigate than hit-based leakage. In miss-based interference, references to two memory blocks interfere with each other if they contend for the same cache line and evict each other. Since the mapping between memory blocks and the cache lines is determined by the cache indexing scheme, miss-based interference may be mitigated by manipulating the memory-to-cache mapping to reduce cache miss contentions. Indeed, many cache indexing schemes have been proposed to reduce conflict misses – though the purpose was for performance rather than for security. In contrast, the interference due to cache hits, i.e., where a former access to a block interferes with subsequent accesses to the same block – making them hit in the cache – is hard to mitigate, because this is the *desired* behavior and the basis of the performance benefit brought by caches. Removing such interference is equivalent to no caching, which makes the use of caching meaningless. This dilemma makes the mitigation of hit-based information leaks inherently hard.

Fortunately, even though hit-based interference is hard to remove without losing performance, there are still ways to circumvent the problem, particularly through software techniques. The most straightforward software countermeasure to eliminate information leakage in caches is to simply avoid using memory accessing operations (e.g., table lookups). However, the performance overhead is very high and the method is not generally applicable. When memory accesses cannot be avoided, some software countermeasures help mitigate hit-based information leakage by preloading objects into the cache before any use of them so that all subsequent accesses hit in cache, thus leaking no information. This approach however is not really secure since the preloaded objects could be evicted by other memory references at a later time, which indeed often occurs. Other software techniques try to avoid interference due to hits by not sharing objects. For example, if in the AES cipher, table lookups for different bytes do not share tables, accesses to these tables will not interfere with each other, and the attacks that rely on this interference, e.g., the cache-collision attacks [116] described in section 3.2.1.3, would not succeed. However, this does not stop attacks based on miss-based information leakage, e.g., Bernstein's attack. In general, we observe that just as it is difficult for hardware to mitigate hit-based interference, it is difficult for software to mitigate miss-based interference. The developer of one program cannot control undesirable evictions of his program's cache lines by another program, since he has little control on how other programs are designed and behave.

Fortunately, hardware mechanisms can help prevent the above problems that software can not handle. For example, cache partitioning [222] can help to prevent undesirable cache evictions if the objects are put into a private partition. This essentially prevents interference due to misses. The problem of such hardware solutions is that they degrade cache utilization and hence, cache performance. Another issue is that from an architecture point of view, cache partitioning is non-trivial in design and often has strict restrictions on certain aspects such as the size of a partition [269]. In addition to cache partitioning, randomization can also help mitigate miss-based information leakage, e.g., by manipulating cache addressing schemes such that interference is randomized rather than eliminated. This may incur fewer restrictions in cache design and have lower performance impact – and is the approach we propose.

The above discussion indeed reveals an important new insight. While software can easily handle hit-based information leak but has little control on miss-based information leak, hardware can easily mitigate miss-based information leak. Therefore a natural choice to build a secure system is that, *the hardware provides the mechanisms that prevent interfering misses while software developers focus only on avoiding interfering hits of their own code* without worrying about how other programs are designed and behave. This simplifies the jobs at both sides.

In this dissertation, we focus on the hardware side, showing novel cache architectures that provide security without compromising performance as well as other design goals. The proposed architectures attack the root cause of cache attacks and follow the two general approaches mentioned above: eliminating interference or randomizing interference. The resulting design therefore is generally effective.

3.3 New Cache Designs for Mitigating Software Cache Attacks

This section presents two novel cache designs, the Partition-Locked cache (PLcache) and the Random Permutation cache (RPcache), that realize cache interference elimination and randomization with little hardware cost and performance impact.

3.3.1 Partition-Locked Cache (PLcache)

The concept of cache partitioning is not new, as described in section 3.2. However, in previous designs, the partitions are mostly static. We refer to such a cache as a statically partitioned cache, or a *partitioned cache* in short. Static partitioning prevents sharing, often leading to large performance degradation. A process may use very few cache lines in its partition, but unused lines are not available to other processes which may need more cache lines than they have in their partitions. Simple static partitioning does not provide sufficient security. For example, even if an AES cipher is running in a private cache partition, attacks based on internal cache interference, e.g., Bernstein's attack, can still succeed. This is because simple partitioning does not eliminate all cache interference.

Instead, we propose the Partition-Locked cache (PLcache) that essentially achieves the effect of cache partitioning, but much more flexibly with less performance degradation and better security. In PLcache, the cache lines of interest are locked in cache, creating a flexible "private partition". These cache lines can not be evicted by other cache accesses not belonging to this private partition. When properly employed, all critical cache accesses will always hit in cache, meaning that the timing variations due to hits or misses are completed eliminated, thus preventing both internal and external interference.

3.3.1.1 Architecture description

The PLcache consists of two parts: the hardware addition to the cache and the system interface for controlling which cache lines should be locked.

A. Hardware addition:
Figure 3-3 shows the hardware addition to the cache, consisting of two new tags, L and ID, per cache line. The 1-bit L flag indicates whether this cache line is locked or not. The ID field indicates the owner of the cache line. Not shown in Figure 3-3, is an optional LL bit per TLB entry, page-table entry or segment descriptor (if the architecture supports segmentation) which indicates if an access to a page or a segment should cause the corresponding cache line to be locked in cache.

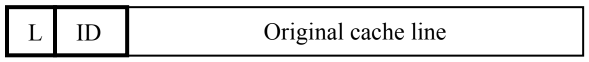


Figure 3-3. A cache line of the PLcache

B. Control interface:
There are two mechanisms that allow the programmer, compiler and OS to control what to lock in the cache. Either mechanism can be implemented:

ISA extension: a new set of load/store instructions with a lock/unlock sub-op can be added to the base ISA (Instruction Set Architecture). This provides the fine-grain control on what data to lock. Table 3-1 describes the new load/store instructions.
Segment/Page-based protection: Regions of memory, e.g., those containing AES and RSA tables, can be marked as LOCKED. Accesses to such regions of memory should cause the corresponding cache line to be locked. This uses the LL bit described above, added to the segment descriptor and the TLB entry. This interface gives the operating system an opportunity to control what data should be locked in the cache. Table 3-2

Table 3-1: Optional ISA extension for PLcache

Name	Description
ld.lock/ ld.unlock	Identical to a normal load instruction with the additional action: If the memory access hits in the cache or causes a cache line to be fetched into the cache, the L bit of the cache line is set/cleared.
st.lock/ st.unlock	Identical to a normal store instruction with the additional action: If the memory access hits in the cache or causes a cache line to be fetched into the cache, the L bit of the cache line is set/cleared.

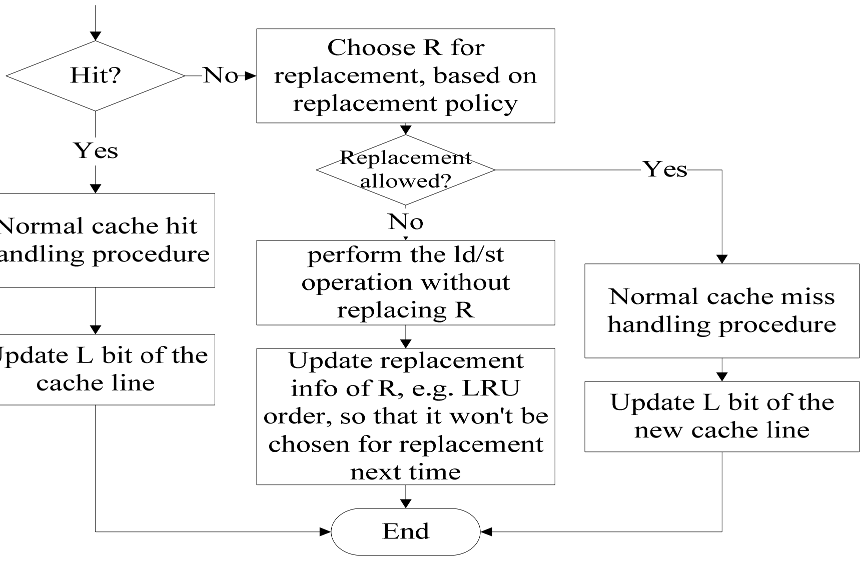
Table 3-2: Potential API calls for PLcache

Declaration
int lock_mem_region(unsigned long start_addr, unsigned long length);
int unlock_mem_region(int region_id);

shows API calls that can be exposed to programmers to make use of this mechanism. To lock a memory region, the function lock_mem_region() can be called which returns a region id. The LL bit of the corresponding segment is set. To unlock a region, the function unlock_mem_region() can be called with the id of the region to be unlocked as the input argument. The LL bit of the corresponding segment is cleared, and the locked cache lines invalidated.

C. Cache access handling:

Figure 3-4 shows the flow chart of an access to a PLcache. Note that the sequential steps shown in the flow chart do not necessarily execute sequentially in the hardware. The *cache hit* handling procedure is the same as in traditional caches except that the L bit of



Hit?
No
Yes
Choose R for replacement, based on replacement policy
Replacement allowed?
Yes
No
Normal cache hit handling procedure
perform the ld/st operation without replacing R
Normal cache miss handling procedure
Update L bit of the cache line
Update replacement info of R, e.g. LRU order, so that it won't be chosen for replacement next time
Update L bit of the new cache line
End

Figure 3-4. Cache access handling procedure for PLcache

the cache line accessed needs to be updated if necessary. If the access is a load/store instruction with lock/unlock sub-op, the instruction itself determines if the L bit should be set or cleared. This information is available early in the pipeline (after the instruction decoding stage) and hence does not impact cache access time. If the LL bit in segment descriptors is implemented, its checking can be done together with the checking of existing protection bits, and no extra delay is added. Similarly, if the LL bit in the TLB entry is implemented, the check can be done together with that for existing protection bits during the TLB access.

During a *cache miss*, the replacement algorithm differs from a traditional cache because of the Locked cache lines. Let R denote the line chosen to be evicted by the normal cache replacement algorithm (e.g., LRU) and D denote the new data block that is being fetched into the cache. The following cases need to be considered:

Case	Description
1	If D does not need to be locked and R is also not locked, D replaces R like in a normal cache miss.
2	If D does not need to be locked but R is a locked line, D can not replace R. In this case, for a load instruction, one can simply return D to the processor execution core. For a store instruction, the data is written back to the next level of memory, without replacing R. The LRU list should be updated so that R becomes the most recently used line and will not be chosen for eviction next time. This can avoid repeatedly missing on this cache set due to the locked line.
3	If D needs to be locked in the cache, it is allowed to replace any line that is not locked or any locked line that belongs to the same process. We do not allow the new line to evict a locked line of another process. Such a miss can be handled as described in case 2.

D. Updating the L bit of a cache line:
If the ISA extension is implemented, the instructions with locking/unlocking capability can set or clear the bits whereas normal load and store instructions can not. If the segment/page based protection is implemented, in each memory access the address is checked and the L bit is set or cleared accordingly. If both mechanisms are implemented, locking/unlocking instructions always set/clear the L bit, and a normal load/store instruction can also set the L bit if the address is in a locked memory region.

3.3.1.2 Discussion

A. ISA extension vs. segment/page-based protection:
The ISA extension gives the software developer the flexibility to prevent cache interference for any portion of its memory. Legacy code however can not benefit without modification. The segment/page based protection provides a rather coarse-grain control mechanism – but both future code and legacy code can benefit from it. For example, the programmer can exploit the API calls to specify a memory region to be protected, and the OS can mark memory regions such as AES or RSA tables used by crypto libraries during load time.

B. Controlling the use of locking mechanisms:
The proper use of PLcache will not allow any program to lock cache lines without OS oversight. Otherwise, a process may, maliciously or naively, lock excessive amounts of data in the cache, causing a security or fairness problem, respectively. An adversary can

also selectively lock certain lines to interfere with other processes. In PLcache, the hardware only provides the locking mechanisms, and the software should ensure their proper use.

In one usage model, the programmer and compiler can specify and optimize what to lock, and request the lock through the API or system call interface. The OS then determines if the lock is allowed based on the resource usage as well as security policy etc. This might impose an upper bound on the number of cache lines that a process can lock and might allow only trusted processes to lock cache lines. For our segment/page-based PLcache mechanism, the OS can make this decision during the API call for locking a memory region, denying this service when necessary. If the call is successful, the OS sets the LL bit of the page/segment, and may optionally access the corresponding lines on behalf of the caller such that the data are locked in the cache when the call completes.

For our ISA-based PLcache mechanism, one simple usage model is to allow only trusted programs to issue the lock and unlock instructions. Another usage model is to have the application first request a memory region of proper size via API or system calls that is "lockable" by the caller. The user-level application then uses lock and unlock instructions to access that region to lock/unlock lines in cache without making further system calls. Note that the user-level lock/unlock instructions will be treated as normal memory instructions if the accesses are outside the "lockable" region. To implement such a system, in addition to the ISA extension, a page/segment level mechanism similar to the LL bit should also be implemented, which would allow the OS to mark a region of memory as "lockable".

In summary, the two basic locking mechanisms can be implemented in various ways to meet different needs. The system designer and the software developer should understand the security implications of the design and make sure the locking mechanisms in PLcache are properly employed.

C. Cache line ID management:
Any hardware implemented field has a limit on the number of items that it can represent. Hence, an n-bit ID field of a cache line limits the maximum number of processes that can own lines in the cache *at any one time* to 2^n. This does not limit the total number of concurrent software processes that the OS can support. For example, processes that do not need to be isolated for side channel attack protection can share the same ID value. In most systems, the majority of the processes are normal processes that do not possess critical information. They do not need to be protected against each other and can share the same ID value, e.g., '0'. Other OS concepts and techniques that manage limited system resources may also be applicable here. For example, processes that will be blocked for a long time, e.g., waiting for disk services, can be temporarily swapped out and the cache line ID freed and allocated to other processes.

3.3.2 Random Permutation Cache (RPcache)

We propose a Random Permutation Cache (RPcache) for the randomization-based approach. In contrast to the PLcache, this approach allows cache sharing, but randomizes the resulting interference, so that no useful information about which cache line was evicted can be inferred.

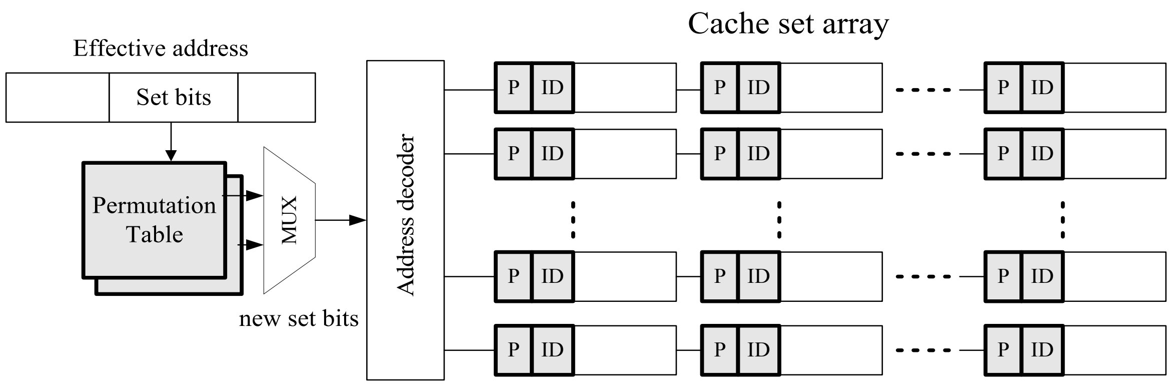


Figure 3-5. A logical view of the RPcache

An attacker can observe another process's cache access only if that process changes the attacker's cache usage, i.e., evicts the attacker's cache lines. If the process evicts its own cache lines, the attacker has no way to know that. By knowing which cache lines have been accessed by the victim process, the attacker can infer critical information about the victim process. In RPcache, each time such cache interference occurs, we randomize it such that the interference carries no useful information.

3.3.2.1 Architecture description

We assume a generic set-associative cache where M bits of the effective address, the *set* bits, are used to index the cache set array. The number of cache sets in the array is 2^M and each cache set contains N cache lines for an N-way set-associative cache, including direct-mapped caches where N=1.

A. Permutation of memory-to-cache mapping

A key operation the RPcache performs is the permutation of the memory-to-cache mapping. Conceptually, this is done by using a level of indirection in indexing the cache. In RPcache, the memory-to-cache mapping for a process is stored in a *permutation table* (PT), as shown in Figure 3-5. The table has the same number of entries as the number of cache sets, and each entry contains a different M-bit number which indicates the new set. For each cache access, the PT is indexed with the M set bits of the effective address to obtain the new set bits, which are then used to index the cache set array. A complete randomization of the memory-to-cache mapping can be achieved by a random permutation of the contents of the table entries. This can be decomposed into a series of swap operations, each of which exchanges the contents of two entries. Swapping the *k*-th and the *i*-th table entries means changing the memory-to-cache mapping, $k \rightarrow$ S and $i \rightarrow$ S', to the new mapping $k \rightarrow$ S' and $i \rightarrow$ S. This indirect indexing scheme is the logical explanation and is not necessary in real hardware, as we will show later.

In the RPcache, a number of permutation tables are added and each table can be used by one or more processes to access the cache. For example, an encrypting process can use one table and all other non-critical processes use another. The number of such tables implemented depends on needs and cost. In PC systems where only occasionally a process needs to be protected, one table should be enough. All other processes can use the original mapping that does not need a remapping table. The memory-to-cache mapping needs to be updated from time to time, during the execution of the process, as described later. Similar to PLcache, a P bit and ID field are also added to each cache line.

B. Randomization of cache interference

We first define terms we will use in our discussion.

Name	Description
R , S	R is the cache line being replaced in cache set S.
R' , S'	R' is the cache line being replaced in another cache set S' which is randomly selected.
D	The memory block being fetched into the cache.
P_X	The P-bit of cache line X, e.g., of R, R' or D.

In the case of cache interference between the victim and attacker processes (external interference), the interference occurs only when the victim evicts a line of the attacker. In RPcache, rather then replacing line R, another cache set S' is randomly selected with equal probability. The new line D that is to be put into the cache then replaces a line R' in S' instead of R in S. The memory-to-cache mappings of S and S' are swapped such that next time when the victim process wishes to access D, he will access the correct cache line. From the attacker's point of view, when he detects a cache miss, the cache miss can be caused by the victim's access to any cache set, with equal probability. Hence he can learn nothing about the address that the victim accessed. Note that after swapping the memory-to-cache mapping of S and S', if the process wishes to access another cache line originally in set S, it will now access set S'. It will miss and bring another copy of the line into set S' although set S still has it. To avoid this undesirable aliasing, the cache lines in S and S' that belong to the current process should also be swapped. However, for efficiency we invalidate all such lines in S and S' and write them back if they are dirty. Future accesses to them will get them correctly from the next level of the memory hierarchy. Since the selection of S' is independent of S, R and D, it can be pre-computed and the write-backs can be performed in the background to hide the associated overhead.

In the case of cache interference from other code in the victim's own process (internal interference), a similar idea can be applied. To distinguish the memory region to be protected from such internal interference, two fields, a P bit and ID field are added to each cache line (shown in Figure 3-5), similar to the L bit and ID field in the PLcache. An internal cache interference occurs if the new line D is not-protected while the old line R is protected, or if D is protected and R is not-protected. As the attacker can not directly observe internal cache interference (since the evicted lines belong to the victim himself), the attacker can only observe the overall effect like the encryption time in Bernstein's attack. If such internal interference is rather fixed, or repeatable, like the eviction of AES table entries at fixed locations, the attacker can learn the fixed interference by performing a large number of trials, observing the cipher's execution time for each trial, and using statistical analysis of these times. Therefore by randomizing every internal cache interference there will not be any repeatable interference (which carries information) that can be observed by the attacker. To randomize internal cache interference, each time when the new line D and the old line R have different P-bit values, R is not replaced. D is

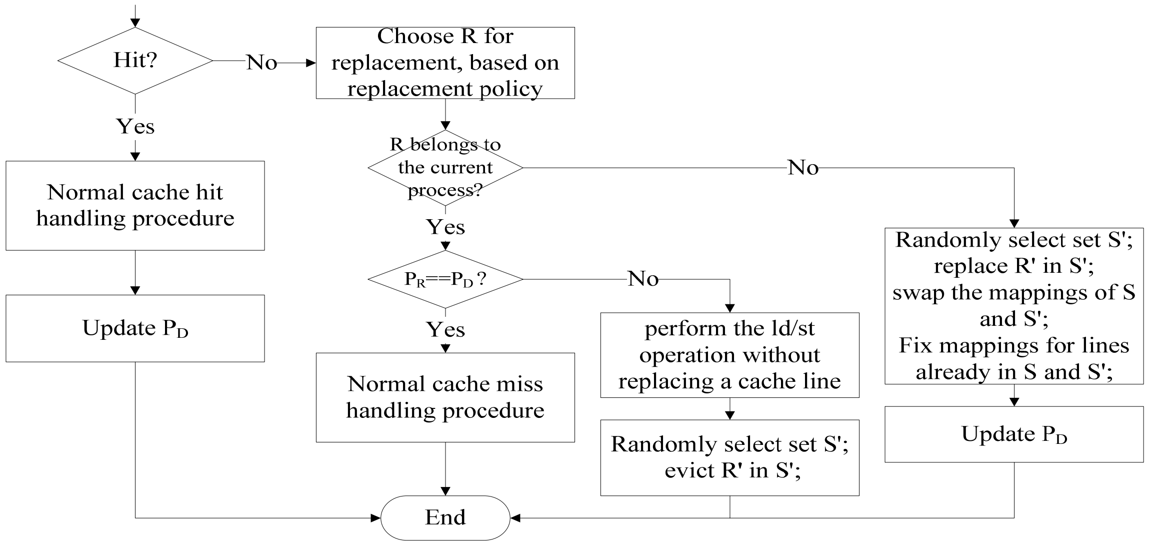

Hit?
No
Yes
Choose R for replacement, based on replacement policy
Normal cache hit handling procedure
Update P_D
R belongs to the current process?
No
Yes
P_R==P_D?
No
Yes
Normal cache miss handling procedure
perform the ld/st operation without replacing a cache line
Randomly select set S'; evict R' in S';
Randomly select set S'; replace R' in S'; swap the mappings of S and S'; Fix mappings for lines already in S and S';
Update P_D
End

Figure 3-6. Cache access handling procedure for RPcache

returned to the execution core if it is a load, or written to the next level of the memory hierarchy if it is a store, without replacing any line in the cache. At the same time, a cache set S’ is randomly selected, and a line R’ in S’ is evicted. Then the original cache interference on R is now on R’ which is purely random.

The mechanisms for controlling which cache lines should be protected are similar to those used in the PLcache except that no new instructions are needed. In addition to the P bit and ID field in each cache line, a PP bit is also added to segment descriptors or the TLB entries. By using the segment/page based protection mechanism described for the PLcache, the OS and programmer can specify the memory region to be protected. In addition, if a section of code is marked as protected, i.e., the code segment descriptor or the ITLB entry has its PP bit set, any cache accesses issued by the protected code will set the P bit of the touched cache lines. This gives a convenient way for the OS to protect critical modules, e.g., the crypto libraries. The OS only needs to set the PP bit of the code pages of such modules.

C. Cache access handling

Figure 3-6 shows the flow chart of the cache access handling procedure. A cache hit in the RPcache is the same as a normal cache hit except that the P-bit of the cache line needs to be updated, based on the value of the PP-bit. During a cache miss, a line R in set S is chosen using the normal cache replacement policy. If R belongs to another process, a random set S’ is selected. The new line D then replaces R’ in S’ and the memory-to-cache mapping for S’ and S is swapped. Note that in the last column of Figure 3-6, “Fix mappings for lines already in S and S’ ” means invalidating (and flushing if dirty) all lines in S and S’ that belong to the current process, except for the new line D, to avoid accessing those lines with old mapping after the mapping of S and S’ is swapped. This is also explained in section 3.3.2.1 (B).

If R belongs to the same process, two cases need to be considered, as shown below.

Case	Description
1	If P_D == P_R, R is replaced by the new line like in a normal cache miss.
2	If P_D != P_R, R can not be replaced and the access is performed without replacing R. R's replacement information is updated so that it will not be selected for eviction next time. This avoids repeated misses in set S. At the same time S' is randomly selected with equal probability among all cache sets, and R' in S' is evicted, based on the normal cache replacement policy for blocks in a set.

3.3.2.2 Low-overhead RPcache implementation

Using an extra level of indirection in cache indexing can introduce extra delay into the cache access time. For an L2 or L3 cache, a straightforward table lookup implementation may be good enough since one extra cycle in L2 or L3 cache loads will not cause much performance loss. However, for an L1 cache, which is often the most delay-sensitive module in a processor, an extra cycle on a cache hit may be unacceptable. We now show that indirect indexing for our RPcache can be implemented, without requiring an extra cycle, nor extending the cycle time latency.

Figure 3-7 shows the modified decoder circuitry for the RPcache based on the common implementation with the 3-to-8 NAND pre-decoder and the second stage NOR gates. Rather than having a fixed connection for each input of the NOR gate with one output of a 3-to-8 NAND pre-decoder, each input line of the NOR gate is connected via switches to all of the 8 output lines of the pre-decoder. The switches are controlled by a register called the *permutation register(PR)*, and at any time only one switch is on. Each permutation register is one entry of the permutation table in Figure 3-5. Note that we omit the MUX in Figure 3-5 for clarity. Compared with the original decoder, the only extra delay in the critical path is caused by the switch transistor. The path from the PR to the output of the NOR gate is not the critical path since the PR can be read out early in the pipeline instead of at the beginning of the cache access cycle: once the instruction is known as a memory-accessing instruction and to which process it belongs, the PRs can be read out and properly selected by the MUX. The delay caused by the switches is mainly due to the drain capacitance of the switch transistors which increase the load

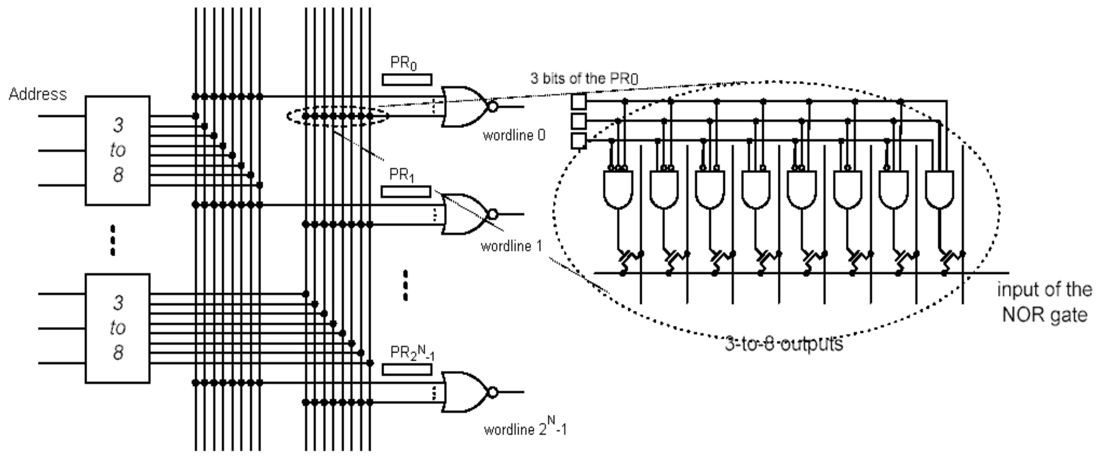


Figure 3-7. Address decoder circuitry of the RPcache

capacitance of the 3-to-8 NAND pre-decoders. To overcome this, we implement multiple copies of the pre-decoders, and let each of them drive a portion of the vertical lines such that the load of each NAND gate does not increase much. We also manually adjust the transistor sizes along the critical path, including the address bit drivers, the NAND gates, and the switches. We also insert a buffer between the address bit driver and the pre-decoders. We model this using cacti-3.2 tool [270], assuming a 0.18um technology. Table 3-3 shows the simulated results, where we first optimized the access time to less than 5% increase, then optimized the power to less than 10% increase. The increase in percent is relative to the unmodified cache modeled in cacti-3.2. Our results show that we can achieve approximately the same cache access time (within 3%) with less than 10% increase in power consumption. This is a straight forward implementation and further circuit optimization can certainly lead to even better designs.

Table 3-3. Timing and Power Estimation of RPcache

RPcache	16K 2way	32K 2way	16K 4way	32K 4way
Access time(ns)	1.225 (+2.1%)	1.331 (+1.7%)	1.293 (+1.1%)	1.344 (+3.3%)
Power (nj)	1.205 (+8.6%)	1.282 (+1.3%)	1.792 (+6.1%)	1.906 (+2.1%)

3.3.3 Evaluation

3.3.3.1 Security analysis

A. Security analysis of the PLcache

In a PL cache, the critical cache lines of the victim are locked in the cache. This leads to two consequences: 1) the victim's accesses to these lines will always hit in the cache without causing any evictions of the attacker's cache lines; 2) accesses to non-critical lines will not evict locked lines. Consequence 1 eliminates external cache interference due to cache misses, thus defeats the Percival-type attacks. Consequence 2 stops internal cache interference due to cache misses and defeats Bernstein-type attacks. Furthermore, if the software preloads critical lines before any use, all accesses to them will hit in cache, thus avoiding hit-based cache interference. This defeats attacks based on cache hits such as the cache collision attacks.

B. Security analysis of the RPcache

To analyze the security of RPcache's randomization technique, we model the information leak channel via cache misses as a communication channel and use information theory to prove that the channel capacity is zero. As the information leak can be exploited in various ways, e.g., time-driven, trace-driven or access driven attacks, our analysis has to be valid in general. Our approach is to prove that in the best case – when the attacker has the greatest power to observe the information leak, the channel capacity is zero. When the attacker has less power, he cannot do better – thus still getting zero information. Note that the power of the attacker indeed determines the type of the attack. For example, if the attacker is able to observe the outcome of each memory reference instead of just the total

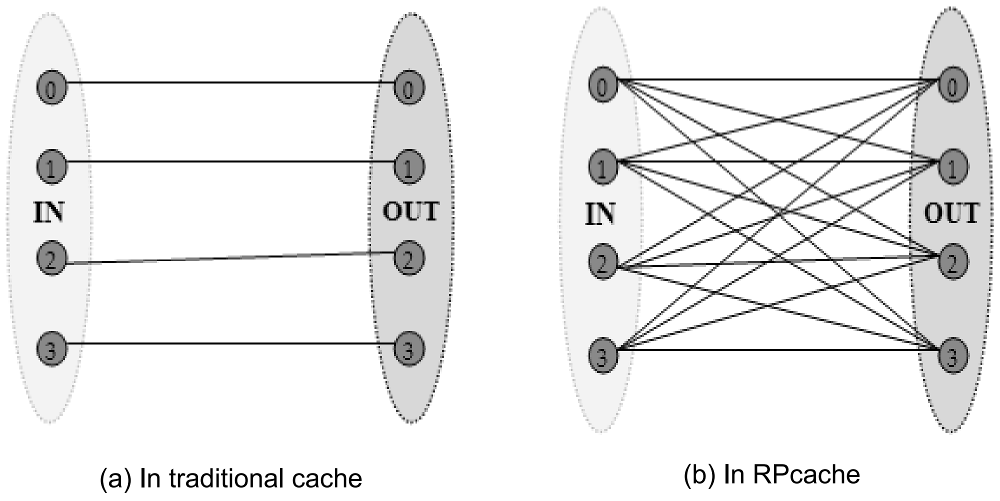


Figure 3-8. A channel model of the cache-address-based side channel

number of cache misses, choosing to launch a time-driven attack rather than a trace-driven attack would waste his observation power without bringing him any extra advantages. Therefore our analysis is valid for different types of attacks. Also note that this analysis is independent of the techniques used to recover the information, e.g., the statistical methods used in time-driven attacks. Such techniques only affect how much information can be recovered from the information that has been leaked. If the information leak itself is zero, then statistical techniques cannot recover any information.

In cache-based side channel attacks, in the best case for the attacker, the attacker is able to observe every eviction caused by the victim without error and know exactly which cache line is evicted. This can be modeled as a classic discrete time synchronous channel, as shown in Figure 3-8. The input symbol of the channel is the line number of the cache line accessed by the victim that would cause an eviction and the output symbol is the line number of cache line for which the attacker observes an eviction. Note that the same physical cache line may have different line numbers from the victim and attacker's points of view (due to different permutation tables they use). In traditional caches, an eviction at a given line number caused by the victim is normally observed at the same line number by the attacker, due to the common memory-to-cache mapping. This leads to the channel model shown in Figure 3-8(a). In contrast, due to random permutations in RPcache, an eviction caused by the victim can be observed at any line number by the attacker with equal probability, as shown in section 3.3.2.1 and in Figure 3-8(b) —which is the channel model for the RPcache. In other words, given an input symbol *i*, the probability that it is observed as an output symbol *j* is equal for any *j*. We then have the following theorem.

Theorem 1: In an RPcache, the capacity of the side channel based on cache line addresses is zero.

Proof:

Let *Pr*(*j*|*i*) denote the conditional probability that given the input symbol *i*, the output symbol *j* is observed:

$Pr(j|i)$ = Prob(output = j | input = i)

The set of such conditional probabilities is called the channel matrix, which determines the channel capacity. For RPcache the following relation holds:

$Pr(j|i) = Pr(j'|i)$ for any i,j and j'

In information theory, it is straightforward to prove that a channel with such a channel matrix has a zero capacity [271]. □

In the above proof we do not consider the real timing of the input symbol arrival rate, i.e., the physical time between successive evictions. Indeed, in communication systems, information can be modulated over the time interval between two successive symbol transmissions. However, in cache-based side channel attacks, little useful information is leaked out in this way and none of the current software cache-based side channel attacks rely on this type of leakage.

C. Remarks

PLcache is a design in favor of minimal hardware complexity, and software has the full control on how the provided basic hardware mechanisms are used. Like any other security mechanisms, a naïve use of PLcache may still be insecure [272]. PLcache relies on software making proper use of the hardware mechanisms to achieve security. For example, to prevent information leakage on the first accesses to an AES table, the whole table should first be loaded (and locked) by the AES program. Secure design with an unmodified PLcache as well as PLcache-based variants have been presented [273].

In contrast to PLcache, RPcache is more complex in hardware but needs less software involvement. However, RPcache may not be able to defeat hit-based attacks (also called cache collision attacks) [272], if used without any help from software. As we have pointed out in section 3.2.2, while information leakage due to cache misses can easily be mitigated by hardware, information leakage due to hits is more suitable to be mitigated by software. Therefore, RPcache is designed to be efficient and effective in mitigating miss-based attacks such as Percival's attack [121], Bernstein's attack [114] and Osivk's attack [118] against AES. Software-based mitigation methods that were previously vulnerable due to miss-based interference, object preloading and no object sharing, can then become more secure when running on RPcache and thus can mitigate a wider range of attacks.

3.3.3.2 Performance evaluation

We implemented the PLcache and the RPcache on M-Sim v2.0 [274] which is a multi-threaded microarchitectural simulation environment based on simplescalar3.0d. AES is used to evaluate the performance impact of the new cache architectures on code being protected. The SPEC2000 benchmark suite is used for evaluating the performance impact on general purpose workloads. In SPEC2000 benchmark simulation, the appropriate number of instructions are fast forwarded, ranging from 100 million to 2.1 billion instructions. Cycle-accurate simulations are then performed for 100 million instructions. Table 3-4 shows the simulation parameters used.

Table 3-4. Simulation parameters

Simulation Parameters	Value
Decode/Issue width	4/4
Integer ALUs	4+1 multi/div unit
Floating-point ALUs	4+1 multi/div unit
ROB size	96
Physical RF size	96 each for Int/FP
Fetch Policy for SMT	Icount
L1 instruction cache	64KB,2-way, 32B lines
L2 unified cache	512KB, 8-way, 64B lines
Cache access time	2 cycles L1, 12 cycles L2
Memory access latency	200 first chunk, 4 inter
L1 data cache ports	2
LSQ entries	48

A. Performance impact on the protected code

Figure 3-9 shows the performance of the OpenSSL 0.9.7a implementation of AES on a processor with a traditional cache with no protection for side-channel attacks (Baseline), an L1 PLcache and an L1 RPcache. A total of 5 Kbytes of data need to be protected in this AES implementation. The simulated program performs the generation of 1 KByte packets and the encryption of the packets, and runs alone on the processor. To examine the effects of the cache capacity and the configuration on performance, we vary the cache size from 4K to 32K and simulated the direct-mapped, 2-way and 4-way set-associative configurations for each size.

Our results show that PLcache is sensitive to the cache size and configuration. When the size of the protected memory (5KB) is larger than the cache capacity (4KB cache), the performance is always bad because all cache lines are locked. Implementing the PLcache as a direct-mapped cache is also not a good idea since once a line is locked, it generates a lot of conflict misses. For cache sizes larger than the protected data, with set-associativity at least 2, the PLcache can achieve comparable performance to the traditional cache.

In contrast, the RPcache consistently achieves almost the same performance as the traditional cache, regardless of the cache capacity and configuration. The performance impact caused by the random cache evictions in RPcache is negligible: worst case 1.7% (on 4K directed-mapped cache) and 0.3% on average.

We also simulate the L2 PLcache and L2 RPcache. As the L2 cache is large enough to hold the working set, no performance degradation is observed.

B. Performance impact on the whole system due to the protected code

The PLcache and RPcache may impact the performance of the system during the execution of the protected code, e.g., the performance of other general purpose workloads running concurrently while encryption is being done for a file. In the simulation, we assume that the protected code (AES) is running concurrently with another thread. We use an 8Kbyte direct-mapped L1 D-cache and a 32Kbyte 4-way L1 D-cache to bound the cache impact. The 6 bars per SPEC2000fp or SPEC2000int benchmark in Figure 3-10

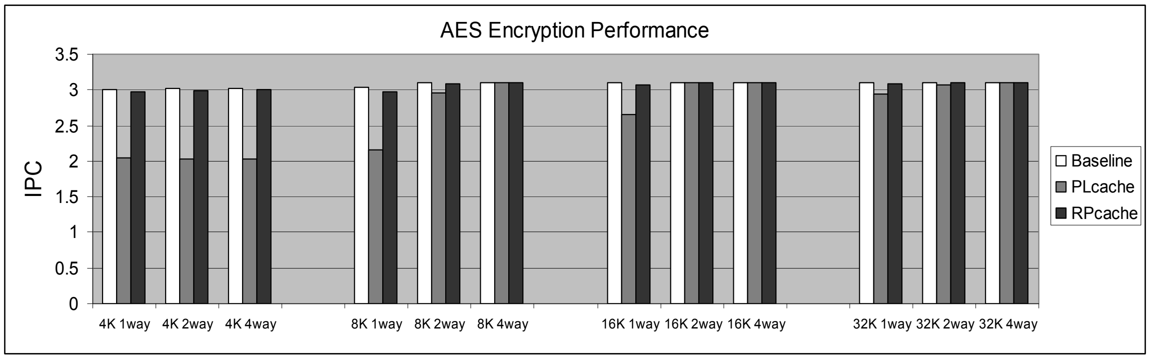
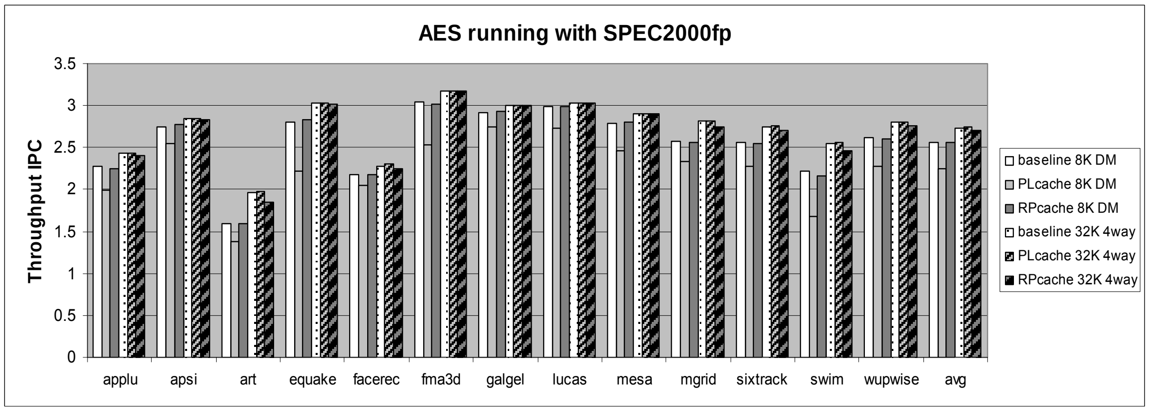
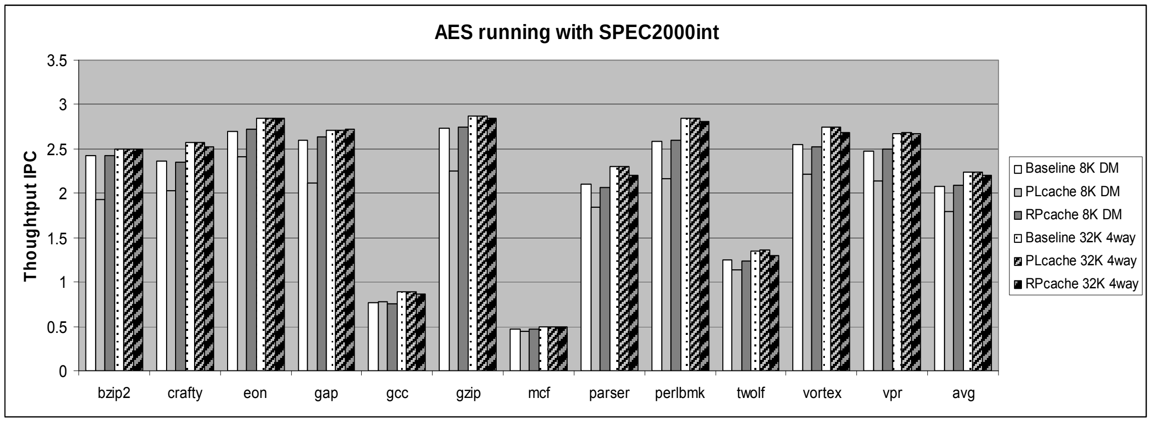


Figure 3-9. Performance comparison of AES code



(a) Overall throughput with SPEC2000fp benchmarks



(b) Overall throughput with SPEC2000int benchmarks

Figure 3-10. Performance impact on overall throughput

show the simulations of the baseline, PLcache and RPcache for 8K 1-way L1 D-cache, then for 32K 4-way D-cache.

For an 8Kbyte direct-mapped cache, PLcache causes an average performance degradation of 12% and 14% on floating point benchmarks and integer benchmarks, respectively. The RPcache causes 0.3% degradation on floating point benchmarks and 0.07% *improvement* on integer benchmarks. The improvement is a result of the swap operations of the RPcache which avoid many conflict misses. On a 32Kbyte 4-way cache, the PLcache achieves a 0.2% performance *improvement* on both integer and floating-point benchmark sets. This is because the 32Kbyte cache is large enough to hold the working sets for both threads and the protected code benefits from the locked cache lines that avoid misses on these lines. The performance degradation for the RPcache is 0.3% on FP suite and 1.2% on INT suite, respectively. The increase in performance degradation is due to the higher overhead associated with the swap operations for a set-associative cache. However, the absolute degradation is still very small. We also examined the effect of implementing the L2 cache as a PLcache or RPcache. The effect is again insignificant.

Although we only use AES as the protected code in our simulations, our conclusions are not specific to AES. The sensitivity of PLcache's performance to the cache configuration and capacity (relative to the size of the protected memory region) is due to the locking behavior and is not a result of any AES-specific factor. The robustness of the RPcache's performance is due to the fact that we allow sharing – and our design intentionally minimizes the restrictions on sharing.

3.3.3.3 Comparison with prior-art

Table 3-5 summarizes the advantages of our PLcache and RPcache solutions compared with the prior-art partitioned cache solution, in terms of both security and performance.

Table 3-5. Comparing with prior-art Partitioned Cache

Security & Performance	Partitioned Cache	Our PLcache	Our RPcache
Prevents external Interference?	Yes	Yes	Yes
Prevents Internal Interference?	No	Yes	Yes
Relative Performance	Low	Medium	High

Security: All three approaches can prevent information leakage via external cache interference. Partitioned cache and PLcache provide private partitions to a process which are not accessible by other processes. RPcache randomizes the interference so that it carries no useful information. The partitioned cache can not, however, defend against attacks based on internal interference; a private partition still allows code within a process to contend for cache lines and cause interference, as in Bernstein's statistical attack. PLcache does not have this problem, because it explicitly locks the desired lines in cache, and other parts of the same process cannot interfere with these cache lines. RPcache randomizes the interference – hence it carries no useful information.

Performance: A partitioned cache does not allow a process which uses very few cache lines to make its unused cache lines available to other processes which may need more cache lines than they have in their partitions. Hence, it has the lowest performance among the three approaches. PLcache can achieve better performance because it has a locking mechanism that allows it to minimize the size of flexible private partitions, leading to better cache utilization. RPcache allows different processes to share cache slots and therefore has the smallest performance degradation. In addition, the performance of the partitioned cache and PLcache depend on software to specify proper partitioning of the cache, while the performance of the RPcache is very robust, with little dependence on the software and the underlying hardware cache architecture.

3.4 Summary

The PLcache and RPcache are the realizations of two leakage-blocking approaches. The PLcache achieves flexible cache partitioning through cache line locking mechanism and mitigates cache based attacks via interference elimination. The RPcache allows cache interference but randomizes it such that it carries no useful information. The PLcache requires minimal hardware cost, but more software interventions. Its performance as well as security relies on the software to make proper use of it. In contrast, the RPcache needs a little more hardware but provides much more robust security as well as performance, without needing inputs from the programmer. As shown in our evaluation, both cache architectures can provide desired security from information leakage, with little impact on performance.

Chapter 4

Improving Cache Performance while Improving Cache Security

4.1 Overview

Due to the restrictions imposed by security requirements, design for security and design for performance are usually at odds. In chapter 3 we have shown that security can be achieved with little impact on performance with the proposed PLcache and RPcache. In this chapter, we show that designing for security, using the randomizing approach of RPcache, can even *improve* performance and bring more benefits. We present a novel cache architecture, Newcache, that randomized dynamic memory-to-cache mapping with three other architectural features that enhance performance and power-efficiency. The proposed architecture can achieve the same level of security as RPcache, facilitate efficient implementation of cache partitioning/locking, and at the same time achieve even higher performance than traditional caches. The proposed cache architecture is also power efficient -- it consumes as little power as a traditional direct mapped cache. Furthermore, the proposed architecture can bring additional benefits including fault tolerance, hot-spot mitigation and further optimization for low power.

4.2 The Proposed Cache Architecture

The proposed cache architecture, Newcache, features four architectural characteristics to achieve performance, power efficiency as well as security. To enable fast cache access time and high power efficiency, Newcache adopts the *direct-mapped* architecture. *Dynamic memory-to-cache mapping* and a *longer cache index* are introduced to achieve low miss rates. To improve security, our cache enhances the randomization approach, which is achieved by *dynamic memory-to-cache mapping* and a new *security-aware cache replacement algorithm* (SecRAND). The performance-enabling features also allow the cache partitioning/locking mechanisms to be implemented efficiently without incurring the performance problems as in traditional caches.

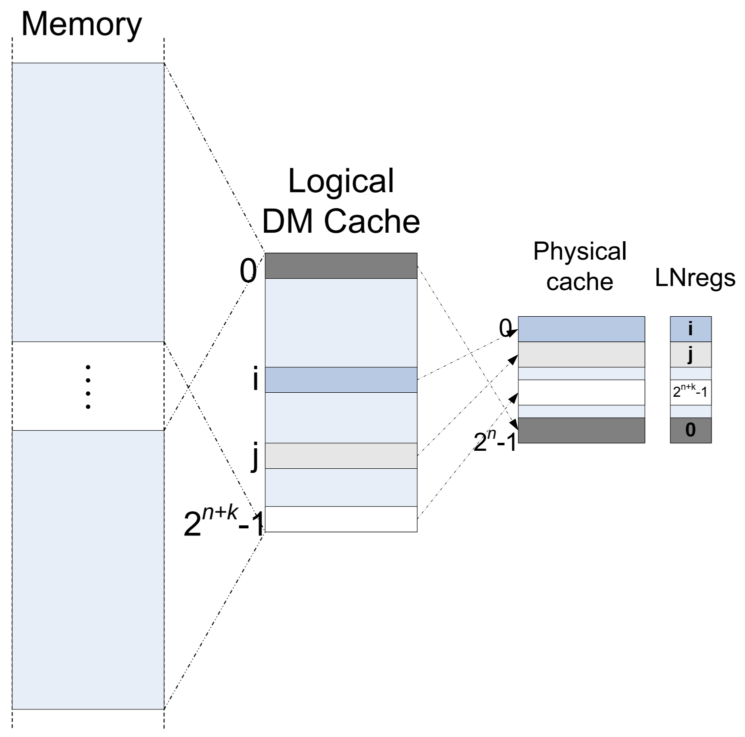


Figure 4-1. Mapping memory space to the physical cache

4.2.1 Dynamic-Remapping and Logical Direct Mapping

The proposed cache implements *dynamic memory-to-cache remapping*, meaning that a memory block can be mapped to any desired cache line at run time. Logically, this can be achieved by using a level of indirection. The index bits of the address are first used to lookup a ReMapping Table (RMT), which returns the index of the real cache set that the address is mapped to. By changing the contents of a RMT entry, an address can be mapped to an arbitrary cache line. The RMTs are updated seamlessly by the cache replacement algorithm – whenever a cache line replacement occurs, the corresponding RMT entry is updated. The indirection overhead to realize dynamic re-mapping can be avoided by clever circuit implementation (as we will show in section 4.2.4).

The proposed cache also adopts the direct-mapped architecture to inherit its fast access time and power efficiency. To avoid excessive conflict misses, a *longer cache index* is introduced. Unlike in traditional direct-mapped caches where using more index bits exponentially increases the cache size, *the proposed cache exploits the dynamic memory-to-cache mapping to achieve low conflict misses without increasing its physical size*. This is illustrated in Figure 4-1. Assuming that the cache contains 2^n physical cache lines, it uses $n+k$ index bits rather than n as in a traditional direct-mapped cache. This is conceptually equivalent to mapping the memory space to a large *logical* direct-mapped cache with 2^{n+k} lines, referred to as the *LDM cache* in the rest of the paper. Note that the LDM cache does not physically exist and is introduced only to facilitate the analysis and discussion of the proposed cache architecture. The dynamic mapping mechanism enables the proposed cache to adapt to store the most useful 2^n lines at run time, rather than

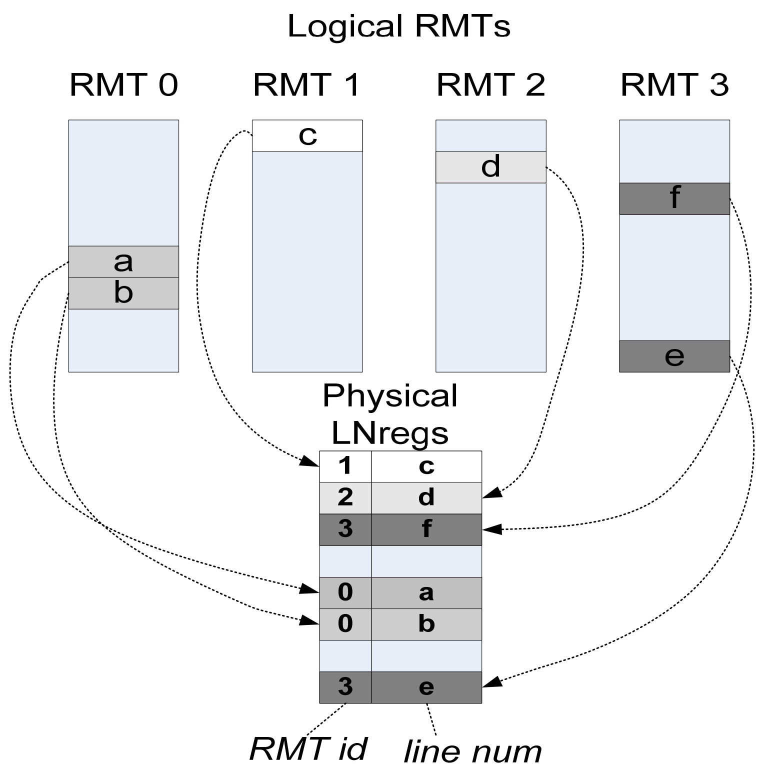


Figure 4-2. Supporting multiple logical RMTs

holding a fixed set of cache lines and missing on others. To remember which lines in the LDM cache are stored in the real cache, each physical cache line is associated with a Line Number register (LNreg), which stores the (n+k)-bit line number of the corresponding logical cache line in the LDM cache. An LNreg physically implements an entry of the RMT (ReMapping Table), and changing the line numbers stored in an LNreg maps another logical cache line to the physical cache line. Although we assume 2^n cache lines in the above discussion, the number of cache lines s in the proposed cache can be any number – not necessarily a power of two – as long as $s < 2^{n+k}$.

A RMT stores a memory-to-cache mapping. For security as well as performance reasons, it is desirable to have multiple mappings, each of which may be used by one or more processes. Note that although logically multiple RMTs are required, they are physically implemented with one set of LNregs. This is because at any time, for each physical cache line storing a logical cache line, only the entry of the RMT associated to the logical cache line needs to be stored in the LNreg. The corresponding entries in all other RMTs are invalid since no logical cache lines of these RMTs are mapped to the physical cache line. Figure 4-2 shows how a single set of LNregs implement multiple logical RMTs. To distinguish which RMT the entry in an LNreg belongs to, an *RMT_ID* field is included in each LNreg in addition to the *line_num* field.

4.2.2 A Summary of the Proposed Cache Architecture

The proposed cache architecture (Figure 4-3) is very similar to the traditional direct-mapped cache architecture, with some significant differences summarized below:

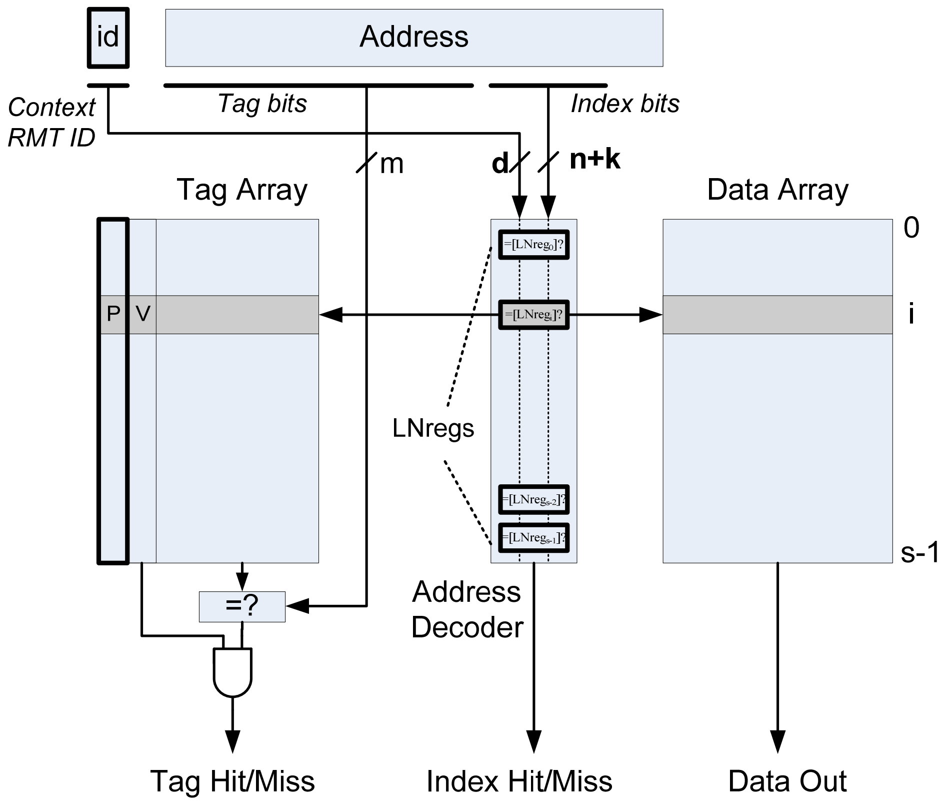


Figure 4-3. The proposed cache architecture

- The address decoder of the proposed cache is modified to implement dynamic memory-to-cache mapping. The LNregs are integrated into the address decoder.
- More address bits, $n+k$, are used as index to access a cache of size $s < 2^{n+k}$. A memory address is mapped into a Logical Direct Mapped (LDM) cache of size 2^{n+k}, then dynamically re-mapped into the real cache of size *s*.
- The number of cache lines is not necessarily a power of two; it can be any $s < 2^{n+k}$.
- Each process is attached to a context RMT ID which specifies the Re-Mapping Tables (RMT) it will use. Different processes therefore can have different memory-to-cache mappings if they are attached to different context RMT IDs.
- Each LNreg contains a *RMT_ID* field of *d* bits and a *line_num* field of $n+k$ bits.
- Each cache line also has a *P* flag bit, indicating protected cache lines. Each Page Table Entry (and/or segment descriptor, if implemented) also has a PP flag bit, indicating a Protected Page. This memory marking mechanism is similar to the RPcache.
- A replacement algorithm is needed on cache misses.

Context RMT_ID: This identifies a hardware context, specifying which RMT is used by a process. A process that needs to be protected against information leak from other processes should use a different RMT. The OS is in charge of associating a process with a RMT_ID when the process is assigned a hardware context for execution.

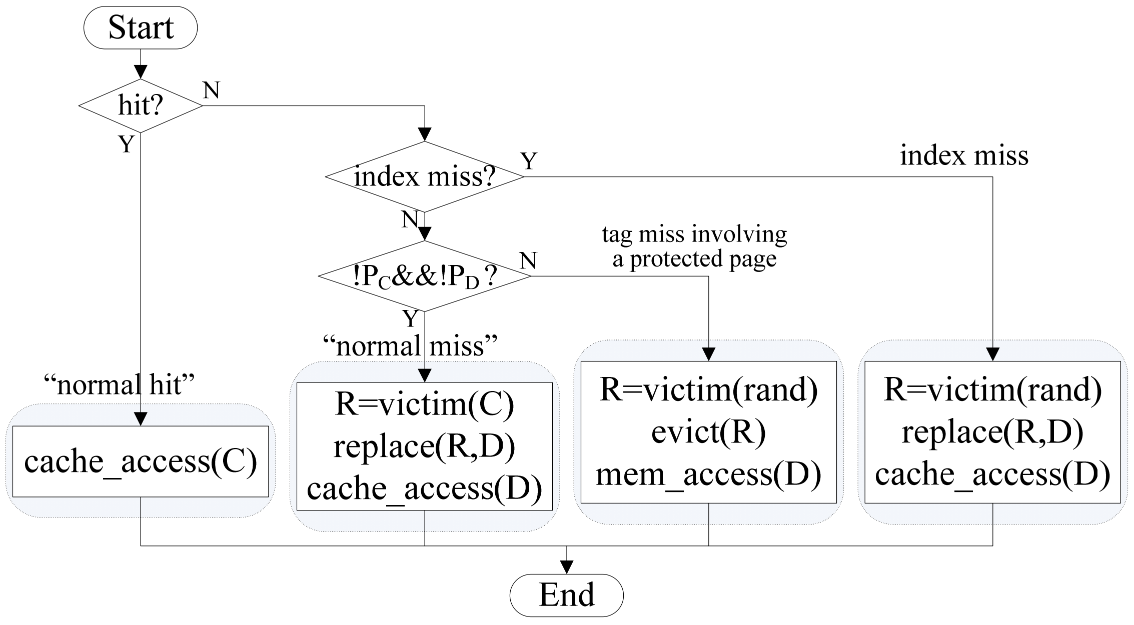


Figure 4-4. New security-aware random cache replacement algorithm

Address decoder and LNregs: In a traditional cache, the address decoder in essence tests a set of conditions (*index* == 0?), (*index* == 1?), … (*index* == 2^n-1?) that compare the index with a series of constants (0 through 2^n-1) and selects one cache line based on the outcome of these comparisons. In the proposed cache, the address decoder tests a similar set of conditions, except that the condition is a variable, viz., the contents of the i^{th} LNreg, [$LNreg_i$], for i = 0, 1, …, s-1. The address decoder activates a cache line if the *RMT_ID field* in $LNreg_i$ matches the d-bit Context RMT_ID *and* if the *line_num* field in $LNreg_i$ matches the $n+k$ index bits. The LNregs are updated when cache line replacements occur. The new line's context RMT_ID and index bits are written to the *RMT_ID* field and *line_num* field respectively.

4.2.3 SecRAND: the Security-Aware Random Replacement Algorithm

Unlike in traditional direct mapped caches, a cache replacement algorithm is necessary in the proposed cache due to the dynamic remapping. During a cache miss, the replacement algorithm determines which physical cache line should be selected for holding the new logical cache line. Since replacing the logical cache line that the physical cache line holds normally means mapping a new memory address to the physical cache line, the LNreg (i.e., the physical realization of the logical RMT entry, which stores the corresponding memory-to-cache mapping) of the selected physical cache line needs to be updated accordingly. There are two types of misses, *index misses* and *tag misses*, in the proposed cache. An *index miss* occurs if none of the LNregs matches the given RMT_ID and index. None of the cache lines is selected in an index miss. A *tag miss* occurs if the index hits in one LNreg, but the tag of the selected cache line does not match the address tag. A tag miss essentially is the same as an ordinary miss in a traditional direct-mapped cache, whereas the index miss is a unique type of miss in our proposed cache. Since an index hit means the match of the RMT ID, tag misses only occur within the same process or among processes using the same RMT. Index misses occur early in the hardware pipeline during address decoding, before the tag is read out and compared, and this early miss signal could be used by the pipeline control logic to improve performance.

The replacement policies for the two types of misses are different as we show in Figure 4-4. The tag misses are conflict misses in the LDM cache since the addresses of the incoming line and the line in cache have the same index (as well as the same RMT ID) but different tags. Because in a direct-mapped cache at most one cache line can be selected at any time, no two LNregs can contain the same index (and the same RMT_ID). Therefore either the original line in the cache is replaced with the incoming line or the incoming line is not cached. For index misses, the new memory block can replace any cache line. While various replacement policies can be used to choose the desired victim line to be replaced, we propose a new modified random replacement policy, which we call SecRAND, for the proposed cache, which provides improved security as well as excellent performance. Figure 4-4 shows the SecRAND replacement algorithm. The cache lines involved and the procedures used in the replacement algorithm are described in Table 4-1.

Table 4-1. Definitions and Notations

Notation	Description
C	The cache line selected by the address decoder (during a cache hit or a tag miss).
D	The memory block that is being accessed.
R	The cache line that is selected for replacement (victim).
P_x	The protection bit of X. If X is in a cache line, it is the P bit of the cache line. Otherwise it is determined by the PP bit of the page/segment that X belongs to.
cache_access(C)	Access line C as in a traditional Direct Mapped cache.
Victim(C)	Select C as the victim line to be replaced.
victim(rand)	Randomly select any one out of all possible cache lines with equal probability.
replace(R,D)	Replace line R with line D, update LNreg.
evict(R)	Write back R if it is dirty. Invalidate R.
mem_access(D)	Access to line D without caching it in the current level of cache.

Cache hits (1st column in the flow chart) are handled as in a traditional cache. When a cache miss occurs, if the LNreg of a cache line C matches the Context RMT_ID and index of the memory block D, then this is a tag miss. As a tag miss always indicates a matching RMT_ID, lines C and D must use the same RMT, which usually means that they belong to the same process. We call this interference *internal* to a process or processes in the same security group. If neither the incoming line (D) nor the selected line (C) is protected (2nd column), meaning that the interference is harmless, the miss is handled normally like in a traditional cache. If either C or D are protected (3rd column),

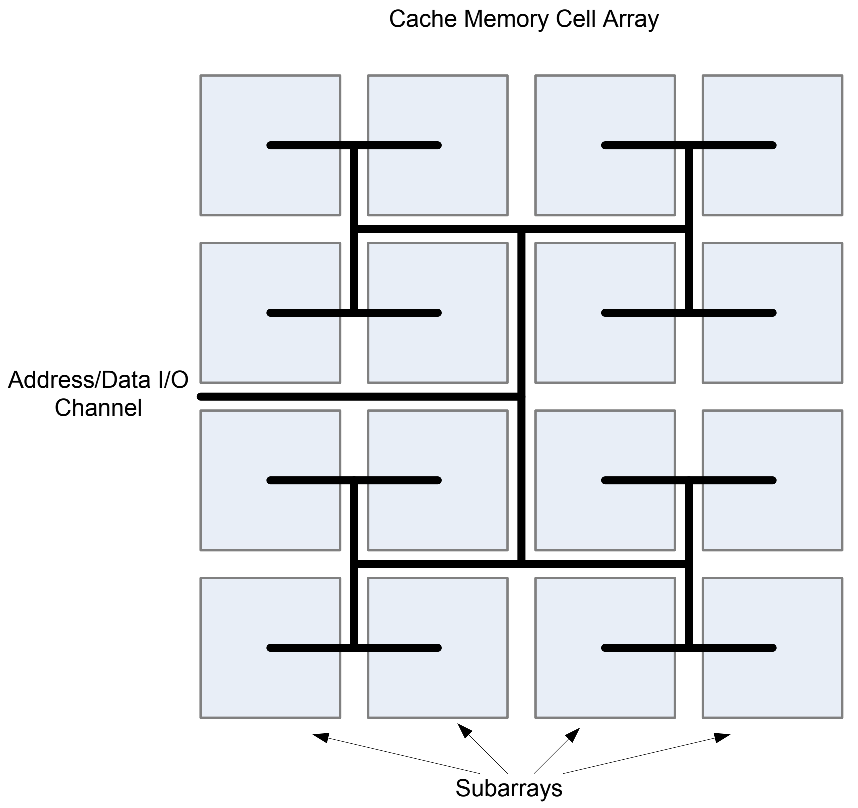


Figure 4-5. A generic cache organization

meaning that the interference may leak out critical information, the replacement algorithm randomizes the cache interference due to the conflict between C and D. To avoid information-leaking interference, D does not replace C, and since in a tag miss D can not replace cache lines other than C, D is sent directly to the CPU core without being put in the cache. On the other hand, since a miss should normally cause an eviction, a random line is evicted which "substitutes" for the eviction of C as well as randomizes the interference. Otherwise the old cache lines tend to stay in cache and new cache lines will not get cached. If the miss is not a tag miss, it is an index miss (4th column) – none of the LNregs match the RMT_ID and index of D. In this case, C and D may or may not belong to the same process. Since for an index miss the new memory block D can replace any cache line, a cache line is randomly selected (with equal probability as in the normal RAND) and evicted. The interference caused by an index miss therefore is always randomized. Detailed security analysis of the SecRAND algorithm will be given in section 4.3.4.

4.2.4 Hardware Implementations

Because of the similarity between the proposed cache and a traditional direct-mapped cache, they share most logic and organization in common, which can be implemented in the same way as in the traditional caches. The distinct part in the new architecture includes the new address decoder and the new SecRAND replacement algorithm. Below we focus on these two aspects.

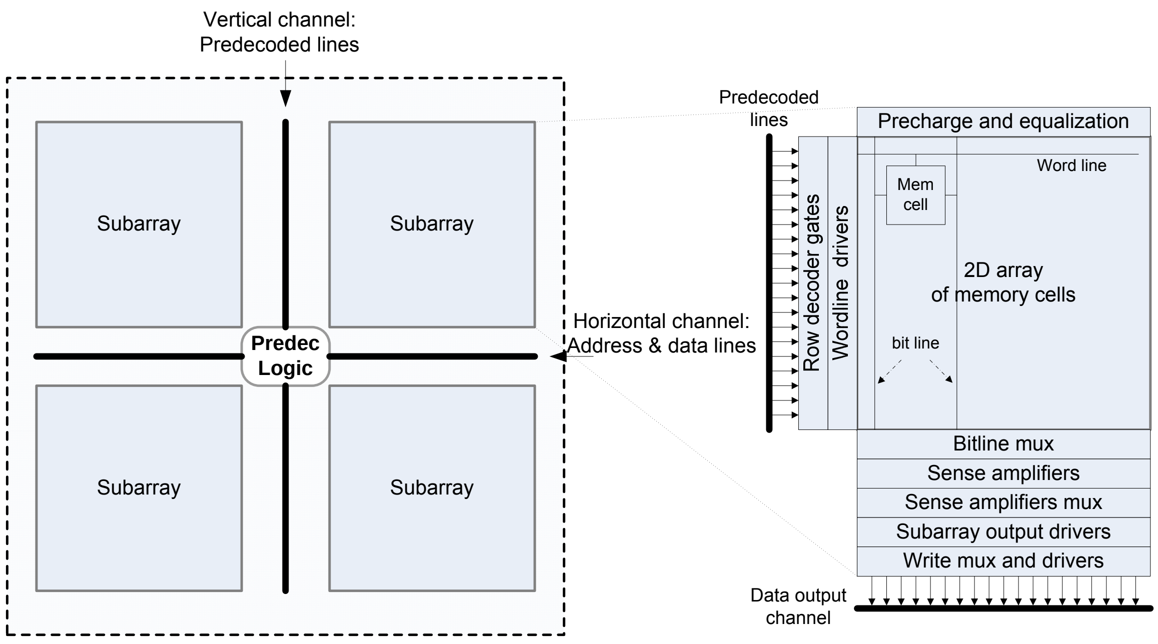


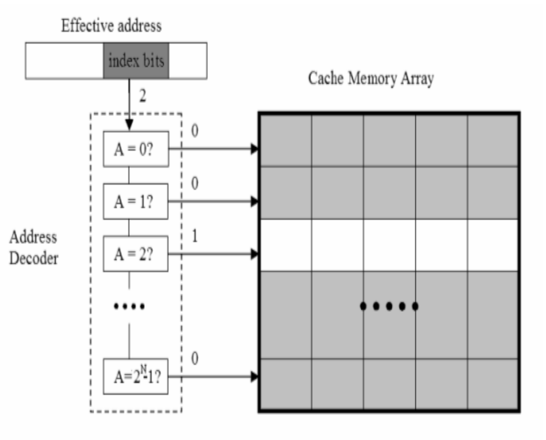
Figure 4-6. Address decoder and subarray structures

A. The new address decoder design

In modern cache implementations, instead of using a single huge array, memory cells are typically partitioned into sub-arrays to achieve fast timing and low power dissipation. Figure 4-5 shows a generic example of such a cache organization. Clocks, address lines, and data I/O lines are often routed through an H-tree network to the sub-arrays. The address decoder lies in the center of each sub-array, decoding the address into word line selection signals, each of which selects a row of memory cells (e.g., a cache line) to access. Figure 4-6 illustrates more details of the sub-array structure, including the pre-decoder logic, row decoder, word line and bit line structures.

As shown in Figure 4-3, the main difference between a traditional DM cache and the Newcache lies in the logic that determines which cache line to select given the input address. In a traditional DM cache, the mapping between an input address to a cache line is fixed, whereas in the Newcache, the mapping is controlled by the contents in the RMT table entries (i.e., the LNregs as shown in Figure 4-3) – given an input address along with its RMT id, a cache line will be selected only if the address and RMT id pair match the content in the LNreg associated to the cache line. In the physical implementation, this logic exists between the pre-decode logic and the row decoder gate in Figure 4-6, where the input address is decoded and the cache line select signal is generated. The rest of the design is essentially the same, i.e., both caches should have similar sub-array organization and routing, and therefore similar wire delay, gate delay, and eventually the total access time. Below we describe the physical implementations of the address decoders in detail.

Logically, as depicted in Figure 4-7(a), the traditional address decoder in essence tests a set of conditions (*index* == 0?), (*index* == 1?), … (*index* == 2^n-1?) that compare the index with a series of constants (0 through 2^n-1) and selects one cache line based on the outcome of these comparisons. In physical implementations, as shown in Figure 4-7(b)



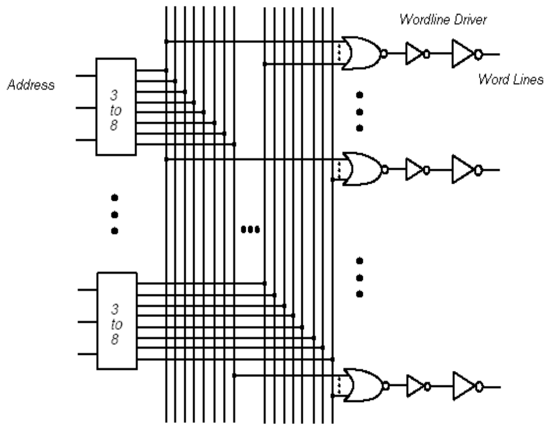


Figure 4-7. (a) logical view of the address decoder (b) physical implementation

which is the most commonly used design, this is reflected in how the 3-to-8 predecoded address lines (the long vertical lines in the figure) are connected to each word line driver (the horizontal NOR gate in the figure). In other words, each NOR gate is connected to a different combination of the predecoded address lines as its inputs, testing one of the conditions (*index* == k?). In the traditional address decoder, the address-to-cacheline mapping is fixed, and therefore the physical connections between the predecoded address lines and the NOR gate inputs are fixed. In RPcache, the address-to-cacheline mapping is dynamic, meaning in the condition (*index* == k?) k is a variable instead of a constant. Physically, this is achieved by using a switch array to control which predecoded address line is connected to the input of the NOR gate, as shown in Figure 4-8 (the circuitry in the middle). The Newcache address decoder is similar to the RPcache address decoder in the sense that the address-to-cacheline mapping is dynamic. Below we discuss three implementation alternatives: the Content-Addressable-Memory (CAM) based design [275-276] (since the Newcache address decoder logically performs a search in the LNregs, looking for a match of the index bits of the address and the RMT_ID), the switched-based design in the RPcache which can be directly applied to the Newcache address decoder design, and a new improved design.

A traditional way to implement associative search is through Content-Addressable Memory (CAM), i.e., the LNregs are implemented as a CAM array. CAM search however is slow and/or power consuming. As shown in [276], the word length of each CAM entry is limited to 6 bits to achieve a delay comparable to that of a traditional decoder. Furthermore, if implemented as a separate array, the CAM approach would require routing the output of the CAM array to the main cache array, which could impact cache access time even more since routing delay has become a dominant factor of the overall access latency.

The switch-based design in the RPcache and the proposed new design both integrate the comparison logic into the address decoder. By making use of the existing decoding logic distributed along with the cache memory cell array, slow CAM search and unnecessary routing are avoided. Figure 4-8 shows a comparison of the traditional address decoder, the RPcache decoder and the new improved decoder based on the

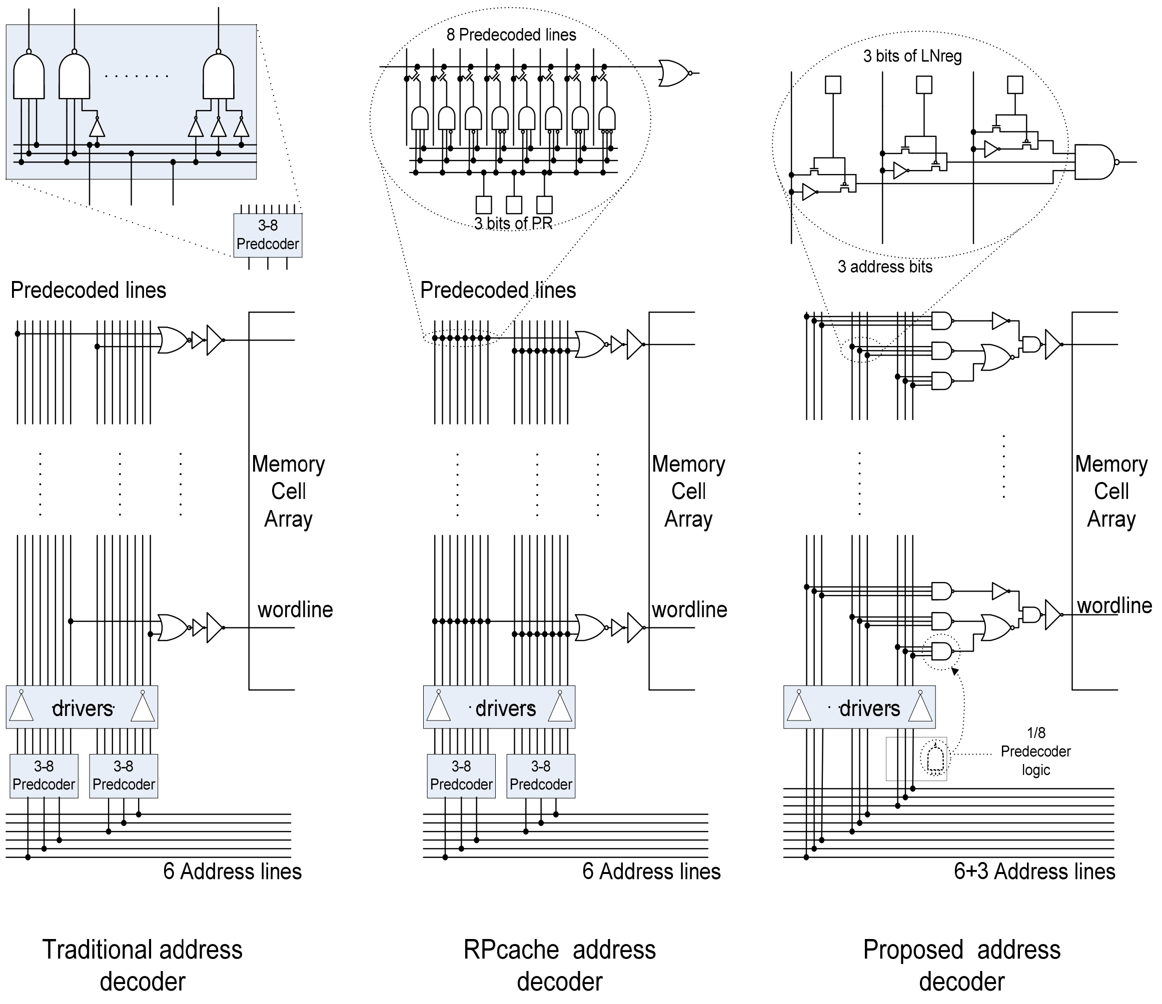


Figure 4-8. Overall structures of the address decoders

commonly used NAND-NOR topology. Note that in real design the exact circuitry may vary, but the same principle still applies.

As we mentioned earlier, the RPcache decoder design is indeed identical to a traditional address decoder with one exception. In both decoders, address lines are first pre-decoded with 3-8 decoders, i.e., for each 3 address bits, a group of 8 pre-decoded lines are generated and sent along the edge of the memory cell array. The row decoder of each row then takes one of the 8 lines from each group as its input and generates the word line select signal. The connections between the pre-decoded lines and the inputs of the row decoders determine on which address value a row is "selected", i.e., the connections determine the memory-to-cache mapping. In traditional caches, the memory-to-cache mapping is fixed, and therefore the connections between the pre-decoded lines and the row decoder inputs are fixed. In the RPcache, the memory-to-cache is dynamic, and as a result the static connections between the outputs of the 3-to-8 pre-decoders and the inputs of the final NOR gates in the row decoder are replaced with dynamic connections via switches controlled by the permutation registers. The extra hardware cost associated with the dynamic connections is very low: for each switch, a NAND3 is used to generate the control signal, and for every 3 address bits, 8 switches and 8 NAND3 gates are needed. Note that due to the drain capacitance of the switches the pre-decoded lines may be more heavily loaded and may lead to longer delay. This can be resolved by inserting repeaters in the wires or having duplicated wires each driving a smaller number of rows.

In the proposed new design, rather than controlling the connections between the predecoded lines and the inputs of the final NOR gates, we control the connections between the address lines and the inputs of the decoder. The 3-8 predecoders are removed and their logic corresponding to each row – a NAND3 gate, is moved to sit beside the word line driver. The switches control how address bits are connected to the NAND3 gate, and thus control which cache line is activated given an index. This implements the dynamic memory-to-cache mapping. The hardware required is less in the new design than in the RPcache, i.e., (6 switches, 3 inverters, 1 NAND3 gate) vs. (8 switches, 8 NAND3 gates) for every 3 address bits. Note that the switches and related circuitry of the new address decoder (shown in the upper right corner of Figure 4-8) is only one example design and can be further optimized in terms of hardware cost and load on the long wires. Since our cache has longer index bits, the output of the NAND3 gate corresponding to the extra address bits needs to be ANDed with the output of the NOR gate. This is done by replacing the first inverter in the word line buffer string with a NAND2 gate. By properly adjusting the transistor sizes of the NAND2 gate, no extra delay is introduced. Compared with the RPcache address decoder design, the new design requires less hardware for implementing switches, has lower loads of the long wire and routes address lines instead of predecoded lines along the edge of the memory cell array, reducing the number of long wires and improving power efficiency.

Since the new decoder design is largely based on the traditional decoder design, the extra implementation overhead is minimized. The extra overhead for combinational logic is very low. In the example shown in Figure 4-8, for each cache line that probably contains several hundreds of memory cells and port switches, the extra circuits required only include 3 NAND3 gates, 10 inverters and 18 switches, and all these devices are about the minimal size since they are all minimally loaded. The overhead for storage, i.e., the LNregs, is also low. We assume that the LNregs are laid out aside the memory cell array and implemented with the same memory cells. Since each cache line is associated with one LNreg, the overhead of LNregs relative to the overall cache storage is $(n+k+d)/M$, where n,k,d are defined as in Figure 4-3, M is the total number of memory cells in each cache line including data, tag and flags. In a 64KB cache with 64-bit address and 64-byte cache line size, n=10 and M $\approx$ 64x8+42+6=560, where 42 is an approximation of the tag size and 6 is a rough estimation of the number of flags and ECC bits. If we allow 4 RMTs and wish to achieve good performance, we can choose d=2 and k=4. The relative overhead of memory storage will be 16/560 $\approx$ 2.9%. In some cache implementations the tag array and the data array may be separated, requiring two sets of address decoders. The overhead will be 5.8% in this case.

B. Implementation issues of SecRAND

Compared with other commonly used replacement algorithms such as LRU, pseudo LRU and FIFO, the random replacement algorithm requires the least hardware cost to implement, due to its stateless nature [277]. Similarly, our SecRAND is stateless and

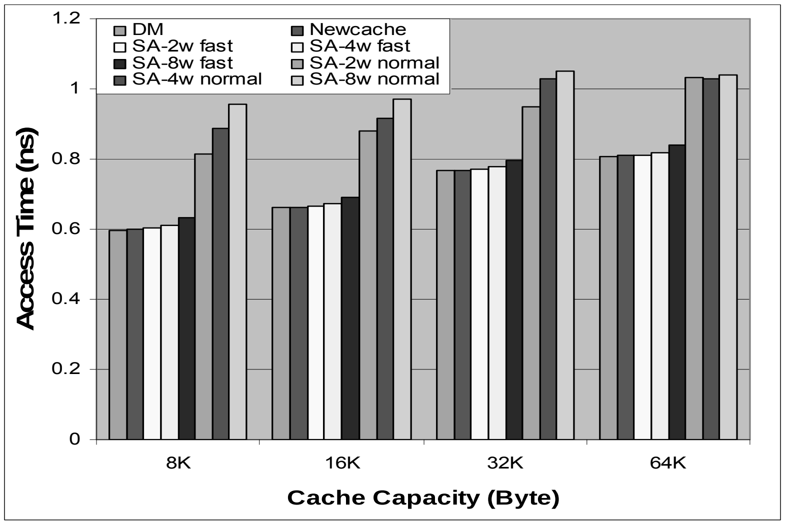


Figure 4-9. Cache access time comparison

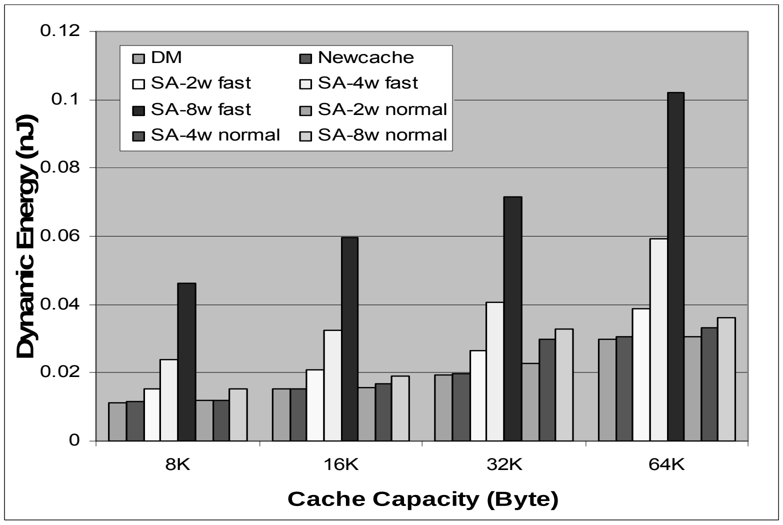


Figure 4-10. Dynamic read energy

enjoys the same advantage. Although SecRAND requires condition checks, these checks are simple and stateless, thus can be trivially implemented with simple combination logic. The security of SecRAND relies on the quality of the random source. This requires a true or pseudo random number generator (RNG or PRNG) on chip. The design of these is out-of-scope for this dissertation. We assume that for any system interested in security, a good RNG or a PRNG [278] is already implemented.

4.3 Analyses and Evaluations

4.3.1 Performance: cache access time

The performance of a cache architecture depends on short access times and low miss rates. We use CACTI 5.0 [279] to explore the design space and find the optimal access times and power consumption. The code corresponding to the address decoder is modified to model the logic shown in Figure 4-8. More accurate transistor level simulation is also performed using HSPICE. The transistor netlists corresponding to the circuit used in CACTI are constructed with the 65nm Predictive Technology Model (PTM) [280].

To accurately model the long wires in the decoder circuitry, we manually extract the parameters of long wires based on the geometrical information generated by CACTI. We focus on fast L1 caches since these are more impacted than L2 and L3 caches. Figure 4-9 shows the results on overall cache access time generated by CACTI.

The extra delay introduced by our proposed cache, referred to as "Newcache" in the discussion below, is always within 1% range of the access time of a traditional direct-mapped (DM) cache. We also compared the access times of commonly used set-associative (SA) caches that are 2-way, 4-way or 8-way set-associative. The "fast" caches are optimized for speed whereas the "normal" caches are optimized for both speed and power efficiency. The data are generated by configuring CACTI with fast mode and normal mode, respectively. Although a fast SA cache could have an access time close to that of our cache, the power consumption is significantly higher – up to 4 times higher than our Newcache, as shown in Figure 4-10. Table 4-2 shows the HSPICE results for a traditional direct-mapped cache versus our proposed Newcache. In all cases, the extra delays are no greater than 5ps, which is less than 1% of the overall access times.

Table 4-2. HSPICE Results on Address Decoder Delay (normalized results in parenthesis)

	8KB	16KB	32KB	64KB
Traditional	0.149ns (1)	0.149ns (1)	0.226ns (1)	0.192ns (1)
Proposed cache	0.151ns (1.013)	0.151ns(1.013)	0.230ns(1.018)	0.197ns(1.026)

4.3.2 Performance: miss rate analysis

4.3.2.1 Theoretical analysis

Cache misses have been classified as compulsory misses, capacity misses or conflict misses [281]. Compulsory misses (e.g., on a cold start) are common to all caches. Capacity misses (e.g., when the program's working size exceeds the size of the cache) only depend on cache size. Conflict misses depend on both the cache organization (e.g., set-associativity) and capacity. To reduce conflict miss rate, a traditional way is to increase associativity, which however impacts cache access time and power efficiency. Increasing capacity can reduce capacity misses as well as conflict misses. However, this is often not feasible in practice due to the limited silicon real estate budget.

In contrast, we show, for the first time, that conflict misses can be largely independent of cache capacity. Our analysis shows that, regardless of its real capacity, our proposed Newcache with an $(n+k)$-bit index has *less conflict misses* than a traditional direct-mapped cache with 2^{n+k} cache lines. The total number of misses in our Newcache has the following bounds:

$$|Miss(Newcache,2^n)| \le |CompulsoryMiss| + |CapactiyMiss(2^n)| + |ConflictMiss(DM,2^{n+k})| \quad (4.1)$$

$$|Miss(Newcache,2^n)| \ge \max\{|Miss(DM,2^{n+k})|,|Miss(FA,2^n)|\} \quad (4.2)$$

where *Miss*(*Arch*, *Size*) denotes the set of misses in a cache of type "Arch" with a capacity of "Size" and |A| is the number of elements in set A. Detailed analysis can be found in Appendix 4. In (4.1), the left side of the equation can be decomposed to the same first 2 terms as the right side plus a third term: *ConflictMiss*(*Newcache*,2^n). Hence, (4.1) shows that the conflict misses of our new cache is less than or equal to that of a direct-mapped cache with 2^{n+k} cache lines. Indeed, as verified in the next section, this bound is asymptotically tight and is a good approximation of the true miss rate in real configurations. This means that *the conflict misses of our proposed Newcache are largely independent of its actual cache capacity*. The conflict misses are indeed dependent on the size of the larger LDM cache, 2^{n+k}, rather than on the actual cache size, 2^n. This property of our proposed cache gives cache designers the ability to control the conflict miss rate at the desirable level by choosing the proper number of index bits, while choosing the capacity independently based on cost or other needs. This avoids the speed and power penalty due to higher associativity and allows finer-grained control on allocating capacity to the cache and making the best use of the resource. This property also enables other benefits that traditional caches can not provide, as we will show in section 4.4.

4.3.2.2 Simulation results

For experimental confirmation of miss rates, we simulated our proposed Newcache and traditional direct mapped (DM), set-associative (SA) and fully-associative (FA) caches on a cache simulator derived from sim-cache and sim-cheetah of the simplescalar toolset [282]. We run all 26 SPEC2000 benchmarks for 1 billion instructions with appropriate fast forward counts ranging from 2 million instructions to 3 billion instructions. Figure 4-11 illustrates the accuracy of the bounds we derived in equations (4.1) and (4.2). The bounds are normalized to the real miss rate to show the relative accuracy. The simulation is done for our proposed caches with 64-byte lines for n = 6 to 10 (i.e., 4K bytes to 64K bytes capacity), with cache indices that are k=3 to 4 bits longer. Except for one point, the bounds are always within the 10% range of the real miss rate, and when $n+k$ or k gets larger, the accuracy increases. Indeed, the derived bounds are asymptotically tight, meaning that the equality in (4.1) holds when k and $n+k$ are large.

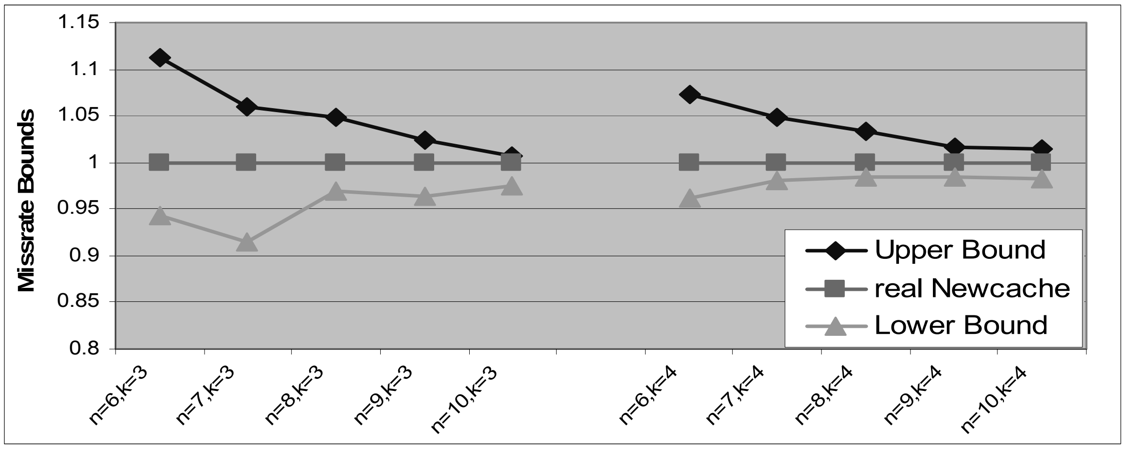


Figure 4-11. Accuracy of the miss rate bounds

Table 4-3 compares the miss rates of our Newcache with the DM cache and the 2-way and 4-way SA caches with LRU replacement. FA caches and 8-way SA caches with RAND replacement are also included to show the effectiveness of our SecRAND replacement algorithm. The lowest miss rate in each column is highlighted in bold (and normalized to 1 in parenthesis). The miss rates of our new caches are in the last 2 rows – our Newcache almost always achieves the lowest miss rates achieved in each column by traditional caches.

Table 4-3. Miss Rate Comparison (relative to best miss rate, in parenthesis)

	4KB	8KB	16KB	32KB	64KB
DM	0.133	0.093	0.068	0.055	0.048
SA-2way, LRU	0.101	0.075	0.057	0.045	0.041
SA-4way, LRU	0.096	0.068	**0.053 (1)**	**0.042 (1)**	**0.040 (1)**
SA-8way, RAND	0.095	0.071	0.054	0.044	0.041
FA, RAND	**0.090 (1)**	**0.067 (1)**	**0.053 (1)**	0.044	**0.040 (1)**
Newcache k=4, SecRAND	0.093 (1.033)	0.068 (1.015)	0.054 (1.019)	0.044 (1.048)	0.041 (1.024)
Newcache k=6, SecRAND	**0.090 (1)**	**0.067 (1)**	**0.053 (1)**	0.044 (1.048)	**0.040 (1)**

4.3.3 Power Efficiency Analysis

We analyze the power efficiency of the proposed cache with regard to two aspects: the per access energy of the cache and the overall power consumption. The cache miss rates are obtained from simulation of all SPEC2000 benchmarks. The power penalty of misses, i.e., the per access energy of L2 cache is obtained using CACTI 5.0.

Modern caches are usually organized as a set of subarrays to achieve fast timing and low power dissipation, as shown in Figure 4-5. The main sources of dynamic power include the power for routing address bits in and data bits out via H-trees, and the power on word lines and bit lines since they are heavily loaded. As our Newcache is direct-mapped, only a minimum number of subarrays need to be activated in each access, which minimizes the power consumed on word lines and bit lines, giving the low per access energy.

Figure 4-10 shows the dynamic read energy data generated by CACTI. The impact of the changes on the overall power consumption compared to DM caches is very low – less than 2%. This is because the percent of energy consumed by the modified structures in our proposed Newcache architecture is low. The new address decoder (excluding word lines since they are not changed) consumes just a few percent more than a traditional DM cache, and the whole decoder power consumption is normally less than 5% of the overall dynamic power. The LNregs consume little power because they are a small amount of memory compared with the size of the cache and have low switching activities – the contents of LNregs need to be changed only during an index miss. Furthermore, unlike accesses to other memory cells, most accesses to LNregs do not involve power-consuming bit-line charging and discharging. Only writes to LNregs require bit-line operations, which occur only when index misses happen. The increase in leakage power in our Newcache is mainly due to the memory cells in LNregs, which is small relative to the overall cache. Hence, the leakage power increase is also low.

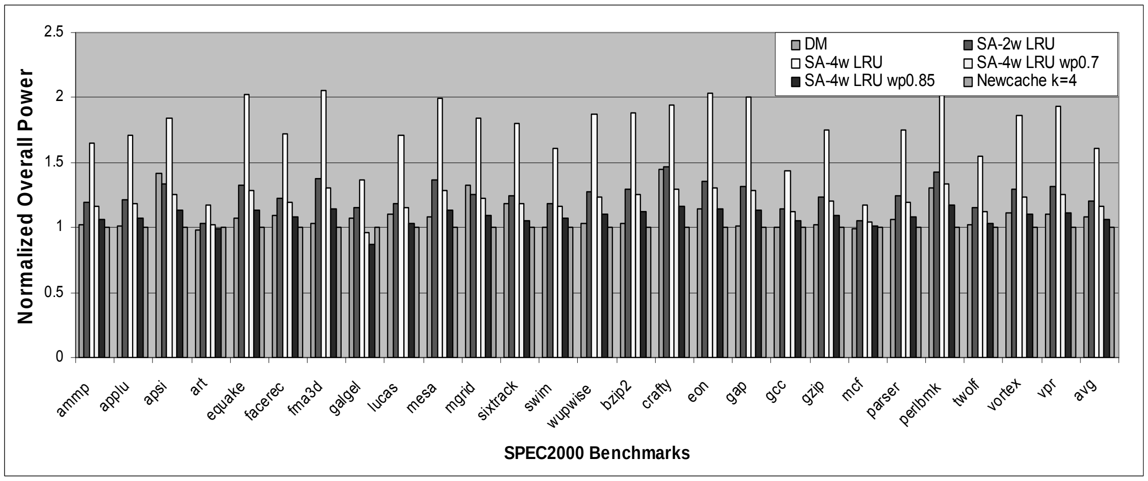


Figure 4-12. Comparison of the overall power consumption

Figure 4-12 shows the results comparing the overall power consumption normalized to our Newcache. We compare traditional SA caches as well as advanced low power SA caches – the way-predicting (wp) SA cache. For example, "SA 4w LRU wp0.7" means a 4-way set-associative way-predicting cache with prediction accuracy of 0.7, and LRU replacement algorithm. All caches are 32KB with 64Byte cache lines. The miss rates of the cache impact the overall system power consumption. A higher miss rate means more accesses to the larger caches or the main memory which consume more power. Our Newcache is more power efficient than the others due to its low miss rate and low per access energy. On average, the 4-way SA cache consumes 61% more power than our Newcache, the 2-way SA cache 20% more, the DM cache 8% more, the 4-way way-predicting cache 16% and 6% more with 0.7 [283] and 0.85 accuracy [284], respectively.

4.3.4 Security Analysis

The proposed cache adopts the randomization approach on cache misses to mitigate information leakage, which is achieved by the SecRAND replacement algorithm. Similar to the analysis of RPcache, we model the information leakage channel as a classic discrete time synchronous channel. The input symbol of the channel is the line number of the cache line accessed by the victim that would cause an eviction and the output symbol is the line number of cache line for which the attacker observes an eviction. Note that the same physical cache line may have different line numbers from the victim and attacker's points of view (e.g., in the proposed cache, they may use different RMTs). To make the capacity of this channel zero, the randomization should meet the following requirement for all protected cache lines:

$$P(j \mid i) = P(j' \mid i) \quad \forall i, j, j' \tag{4.3}$$

where $P(j \mid i) = \Pr(output = j \mid input = i)$. In other words, given an access at line i by the victim that would cause an eviction, the attacker can observe an eviction at any line number with equal probability. From the attacker's point of view, although the attacker can observe a cache eviction, he has no idea which cache line was accessed by the victim.

Below we show that the proposed cache meets this condition. Given a cache miss that causes an eviction that leaks information, the following cases need to be considered.

1) The miss is an index miss. According to Figure 4-4 (4th column), a random cache line R is selected for eviction with equal probability. In other words, for any victim's access that would cause an eviction, all cache lines have the same probability to be evicted, i.e., $P(j \mid i) = P(j' \mid i) \quad \forall i, j, j'$.
2) The miss is a tag miss that involves protected cache lines. As shown in Figure 4-4 (3rd column), the line to be evicted is also randomly selected with equal probability, i.e., $P(j \mid i) = P(j' \mid i) \quad \forall i, j, j'$.

Clearly, the proposed randomization mechanism satisfies equation (4.3), and thus achieves zero channel capacity.

4.4 Additional Benefits

Fault tolerance: Memory-to-cache remapping is a common technique used in fault-tolerant cache design. In traditional caches, a memory block mapped to a faulty line/set is statically remapped to another good line/set [285-287]. Such schemes increase the number of conflict misses since the remapped cache line/set is now shared by more memory addresses. They also increase the number of capacity misses since the faulty lines reduce cache capacity. The proposed cache architecture can provide fault tolerance in a similar manner using remapping, but with better performance. As shown in section 4.3.2, due to the dynamic memory-to-cache mapping of our Newcache architecture, a cache of size *s* with *p* faulty cache lines is equivalent to a cache of size *s-p*, which has the same conflict miss rate as shown by (4.1). In other words, faulty cache lines in our proposed cache only increase capacity misses, but not conflict misses.

Hot-spot mitigation: Due to spatial and temporal locality, the references to a small number of cache lines account for a majority of the total cache references. The more frequently accessed cache lines generate more heat, causing hot spots. Such unevenly distributed cache line accesses however are mostly avoided in our proposed Newcache. The SecRAND replacement algorithm maps memory blocks to randomly selected physical cache lines, which avoids clustering of frequently accessed cache lines.

Optimization for power efficiency: With the ability of mapping memory blocks to arbitrary physical cache lines, our Newcache architecture can also facilitate low power design. For example, by adaptively turning off cache lines based on a program's working set, the power efficiency of the cache can be further improved with minimal impact on performance. An analysis similar to that in the discussion of fault tolerance can show that turning off cache lines in the proposed cache will cause fewer additional cache misses than in traditional caches.

Benefits for cache partitioning and locking: In traditional caches such as set-associative caches, cache partitioning is not trivial and has many restrictions [269]. A set-associative cache can be partitioned in two ways: horizontal partitioning and vertical partitioning.

Horizontal partitioning divides cache sets into subgroups, each of which forms a partition. One issue with this scheme is that the number of cache sets in each partition has to be a power of 2. This severely limits the flexibility of choosing a partition size. In addition, the address decoder has to be redesigned so that it can be reconfigured to index different numbers of cache sets. Vertical partitioning partitions cache "ways" (degrees of associativity) into subgroups. As most caches have limited associativity, the number of partitions can be very limited. In addition, the partitions have lower associativity than the original cache, thus incurring higher conflict miss rates. Cache line locking is a more flexible way to "partition" a cache, as in PLcache [10]. It however also suffers from higher conflict miss rates. In a set-associative cache, the locked line(s) in a cache set reduce the effective associativity of the set, thus incurring more conflict misses. In contrast, as shown in section 4.2, our Newcache does not have restrictions on the number of physical cache lines in a cache. Therefore cache partitioning and locking mechanisms built upon our proposed cache has the highest flexibility in allocating cache lines to a partition. Moreover, as shown in the discussion of fault tolerance, partitioning a cache incurs fewer additional cache misses in our Newcache than in traditional caches, thus providing better performance.

4.5 Summary

The presence of caches alleviates the increasingly severe memory wall problem, enabling high performance. It however introduces security problems, causing information leakage and leading to cache-based side channel attacks. As the information leakage is due to the inherent cache behavior – cache hits and misses, it is hard to eliminate without compromising performance.

Chapters 3 and 4 aim to identify the root causes of cache attacks and their impact on cache designs, understand the proper roles that software and hardware can play in solving the problem, and propose effective solutions that can achieve security without compromising other design goals such as performance and power efficiency. Our analysis shows that each of cache hits and cache misses can cause information-leaking interference and have different implications on cache design. We also found that the strengths of software countermeasures and hardware countermeasures are complementary: Software can be designed to avoid hit-based interference, but has no control over cache evictions which cause miss-based interference. Hardware can be designed to avoid miss-based interference, but cannot prevent hit-based interference without losing the performance provided by caching.

We then described three cache architectures that can mitigate cache side channel attacks: the PLcache, the RPcache and the Newcache. They are based on two general approaches for hardware mechanisms that mitigate information leakage in caches. The first approach aims to eliminate cache interference whereas the second approach allows interference but removes information leakage through randomization. The PLcache is a realization of the first approach. It provides cache line locking mechanisms which prevent undesirable cache line evictions, thus achieving security. Compared to simple static cache partitioning, the PLcache achieves partitioning via locking, which is more flexible and reduces cache underutilization. The RPcache is based on randomization. It imposes minimal restrictions on normal cache behavior and introduces little performance impact. The randomization technique used by RPcache can achieve provable security, based on information theoretic arguments. The experimental evaluations show that the proposed cache designs can achieve security with low hardware cost, causing little performance impact.

Finally, we described our Newcache solution, which is a novel cache architecture that can achieve security while providing even higher performance and power efficiency than traditional caches. It uses the randomization approach like RPcache with a novel dynamic memory-to-cache remapping, longer index bits than required by the physical cache size, and a random replacement algorithm. Newcache achieves the low access time and low power advantages of Direct-mapped caches and the low miss-rates of set-associative caches. In addition, Newcache achieves the security of leak-free operation against software cache-based side-channel attacks. Furthermore, our Newcache architecture can improve the performance when providing fault tolerance, hot-spot avoidance and cache partitioning or cache locking.

Appendix 4

In this analysis, we consider three cache types, our proposed Newcache, DM and FA, as explained in Table 4-4. We assume LRU replacement policy for FA caches and Newcache for the ease of analytical analysis. RAND and SecRAND algorithm should lead to similar properties in a statisitcal sense since they statisically approximate LRU: even though each cache line can be evicted with equal probability in each individual cache miss, statistically the more frequently accessed lines have higher probability of residing in the cache, and a cache line residing in the cache without being accessed for a long time has a higher probability of being evicted.

Table 4-4: Caches considered

Caches Considered:	Description
Newcache of size *s*	our Newcache, with more index bits referring to a larger Logical Direct Mapped cache than the size of the physical cache (for simplicity we assume $s=2^n$)
DM cache of size 2^{n+k}	Direct Mapped cache (which is the LDM cache in Fig. 4-1). The physical cache of the proposed architecture holds a subset of the lines in the LDM cache.
FA caches of sizes 2^n	Fully Associative cache (for calculation of capacity misses)

Proof of the upper bound (4.1):
The proof of the upper bound is based on three facts. We first define the terms that will be used. The *reuse distance d* is the number of *distinct* block addresses between two consecutive appearances of the same block address. *Compulsory misses* are those due to the first access of the data and have a resue distance $d = \infty$. *Capacity misses* are those due to insufficient cache capacity. In a cache with *m* blocks, a miss is a capacity miss if the block address has a reuse distance $d > m$. *Conflict misses* are those that are neither compulsory misses nor capacity misses, i.e., $d \leq m$.

Fact 1: An index miss in the proposed Newcache is either a compulsory miss or a capacity miss but the opposite is not necessarily true.

Proof: In an index miss, the index of the address is not found in the LNregs, which means that this index either never appeared before, or there were more than 2^n distinct indices since the last appearance of the current index. In other words, the reuse distance of the address is greater than 2^n, and hence an index miss is always a capacity miss or a compulsory miss. On the other hand, a capacity miss or a compulsory miss is not necessarily an index miss. For example, at the first time an address is accessed, the same index may already exist in one LNreg due to a previous access to another address with the same index, and hence leads to a tag miss. This proves Fact 1.

Fact 2: A conflict miss is always a tag miss in Newcache but a tag miss is not necessarily a conflict miss.This is indeed the contraposition of Fact 1.

Fact 3: Considering the misses that occur in the cache architectures we examined, the following relationship holds:

$$Miss(Newcache,2^n) \subseteq Miss(DM,2^{n+k}) \cup Miss(FA,2^n) \qquad \text{(i)}$$
$$CompulsoryMiss \cup CapacityMiss(2^{n+k}) \subseteq Miss(DM,2^{n+k}) \cap Miss(FA,2^n) \qquad \text{(ii)}$$

where *Miss*(*Arch*, *Size*) denotes the set of misses in a cache of type “Arch” with a capacity of “Size”.

Proof: In the proposed cache, the index misses are always misses in the FA cache, by Fact 1. The tag misses are always misses in the LDM cache since whenever an index conflict occurs in the physical cache it must occur in the LDM cache. This proves (i). To prove (ii), consider the compulsory misses and capacity misses in the LDM cache. They are first a subset of $Miss(DM,2^{n+k})$ that includes all misses of the LDM cache. Also, since they have reuse distances $d > 2^{n+k}$, they must also be misses in a FA cache of size 2^n. In other words, they belong to $Miss(DM,2^{n+k}) \cap Miss(FA,2^n)$.
With (i) and (ii), we have

$$\begin{aligned}
&|Miss(Newcache,2^n)| \\
&\quad \leq |Miss(DM,2^{n+k})| + |Miss(FA,2^n)| - |Miss(DM,2^{n+k}) \cap Miss(FA,2^n)| \\
&\quad \leq |Miss(FA,2^n)| + |Miss(DM,2^{n+k})| - |CompulsoryMiss \cup CapacityMiss(2^{n+k})| \\
&\quad = |Miss(FA,2^n)| + |ConflictMiss(DM,2^{n+k})| \\
&\quad = |CompulsoryMiss| + |CapacityMiss(2^n)| + |ConflictMiss(DM,2^{n+k})| \quad \square
\end{aligned}$$

Proof of the lower bound (4.2)

As mentioned earlier in section 4.x, at any time the real physical cache stores a subset of the cache lines in the conceptual *LDM* cache. Therefore, given an arbitrary memory address, if it hits in the physical cache, it must also hit in the *LDM* cache. On the other hand, if it hits in the *LDM* cache, it may not necessarily hit in the physical cache – it will miss if the line being accessed in the *LDM* cache is not yet mapped into the physical cache. We therefore have $|Miss(Newcache,2^n)| \geq |Miss(DM,2^{n+k})|$. On the other hand, The proposed cache should have higher miss rate than the fully associative cache of the same size since it has conflict misses that the fully associative cache does not have, i.e., $|Miss(Newcache,2^n)| \geq |Miss(FA,2^n)|$. □

Chapter 5

Fast Covert Channels in Microprocessors

5.1 Introduction

Over the last few decades, modern processor architectures have evolved dramatically and become very complicated. Numerous new features were introduced, which can lead to various forms of information leakage. Theoretically, all these information mechanisms – including leakage by resource use and leakage by event reporting – can be exploited to construct covert channels. In practice, due to high noise level and/or weak control of the sender and receiver over the channel media, many of these channels are not very practical. This chapter focuses on some very dangerous channels. In our study, we have identified extremely fast covert channels in processors – orders of magnitude faster than traditional covert channels. These channels are analyzed and potential countermeasures are discussed.

In particular, this chapter discusses Simultaneous Multi-Threading (SMT) based and control speculation based covert channels. SMT allows multiple threads to simultaneously execute on the same processor and closely interact with each other, leading to efficient symbol transmission and channel synchronization. SMT can facilitate covert channels exploiting various on-chip resources in general. Control speculation in IA-64 processors is a reporting mechanism that allows the direct observation of a variety of architectural and micro-architectural events. It provides a convenient and noiseless observation mechanism and therefore improves the communication quality.

5.2 SMT-based Covert Channels

Simultaneous Multi-Threading (SMT) is a processor architecture approach that allows multiple threads to simultaneously execute on the same hardware chip, sharing and competing for processor resources [288-290]. The main purpose of SMT is to maximize the use of on-chip resources and therefore to boost CPU performance with low hardware cost. In a SMT processor, almost all on-chip resources are shared, including the function units, physical register files, the whole cache system, the execution pipeline as well as various buffers and queues. Some commercial processors have implemented SMT, e.g., Intel's processors with HyperThreading technology [289].

SMT provides an ideal environment for the sender and the receiver to interact with each other. In most traditional covert channels, the operations of the sender and the receiver are serialized. The sender and receiver take turns to execute, which normally involves expensive context switches (since the sender and the receiver belong to different security domains, context switches between domains usually take a much longer time than normal context switches do). In other words, the transmission of each symbol would incur two context switches and therefore the highest possible information rate of such channels would not exceed half of the frequency that the system can perform context switches even if the symbol transmission itself takes zero time. In contrast, SMT allows parallel execution of the sender and the receiver and completely avoids the need for context switches. Furthermore, compared with other covert channels where the sender and receiver can run in parallel (e.g., in multi-processor systems), the resources being exploited in SMT processors are manipulated in a much finer-grained and more tightly coupled manner. For example, embedding information into and extracting information from shared on-chip resources such as caches takes much less time than exploiting high-level system resources such as file systems or even hardware resources that are off-chip, e.g., the main memory. Covert channels in SMT therefore can achieve significantly higher information rates than traditional covert channels do.

Among the on-chip resources that are exploitable, some are extremely easy to use and enable very high information rates. For example, due to the large amount of information that can be embedded into caches [121], the cache-based channel in SMT is reported to be extremely fast – orders of magnitude faster than traditional cache channels. In our study, we identified another very fast channel – the functional unit based covert channel (the SMT/FU channel). Below we describe the exploit scenario of the SMT/FU channel and discuss possible countermeasures for the SMT/FU channel as well as other SMT-based channels.

5.2.1 Exploiting Scenario of SMT/FU Channels

In SMT processors, in each cycle, the function units are dynamically allocated to threads currently running on the chip. This allows one thread to affect another thread's execution in a very fine-grained manner. For example, if one thread tries to use all the ALUs available on the chip for a few cycles, other threads may be slowed down. Such a tightly coupled execution environment for multitasking systems is ideal for covert channels, especially the covert timing channels.

Assume that the sender S and receiver R are two processes that belong to different security domains, e.g., S is a HIGH security process and R is a LOW security process in a MLS system. Figure 5.1 shows the pseudo code of S and R. For descriptive clarity, we assume that the system only contains these two threads (some processors indeed support only two simultaneous threads, e.g., a class of Intel Pentium 4 HyperThreading processors). The effects of other threads on this channel will be discussed in the next section. We assume that S utilizes the integer multiplier as the shared resource to modulate R's behavior. R senses the modulated signal by comparing its progression with a timer T. To send a bit '1', S calls MULTIPLY() to execute a fixed number of multiply instructions, e.g., 100 to 1000 instructions, which tries to use up all the multipliers. It

Sender S

```
int bit;
…
…
do {
    bit = get_bit();
    if ( bit == 1 )
         MULTIPLY();
    else
         NULL();
} while ( !TX_end() );
…
```

Receiver R

```
int time, dt;
…
…
time = 0;
do {
    dt = time;
    RUN();
    time = get_time();
    STORE(time-dt);
} while ( !RX_end() );
…
```

Figure 5.1. Pseudo code for SMT/FU channel

calls NULL() which executes several hundreds or thousands of `nop` instructions to send a bit '0'. To sense S's behavior, R executes multiply instructions at a constant rate by calling RUN(). Therefore when S sends '1's, R will be slowed down. Information can be recovered by measuring the time it will take for R to execute a fixed number of multiply instructions. To sense the slowdown more accurately, R can utilize an over-sampling technique. In other words, R performs multiple measurements for each symbol that S sends, i.e., R checks timer T more frequently than necessary, e.g., once per 10-100 instructions.

As a demonstration, a SMT/FU channel was implemented on a Pentium 4 HT processor, which supports two simultaneous threads. Figure 5.2 shows an example of the observed waveform, which plots the variation of time in CPU cycles that are needed by the receiver to execute a certain amount of operations, e.g., the RUN() function in Figure 5-1. For illustration purpose, the bit rate is slowed down (~100Kbits per second) during the generation of the figure to achieve clearer waveforms. In practice, the transmission rate can be much higher.

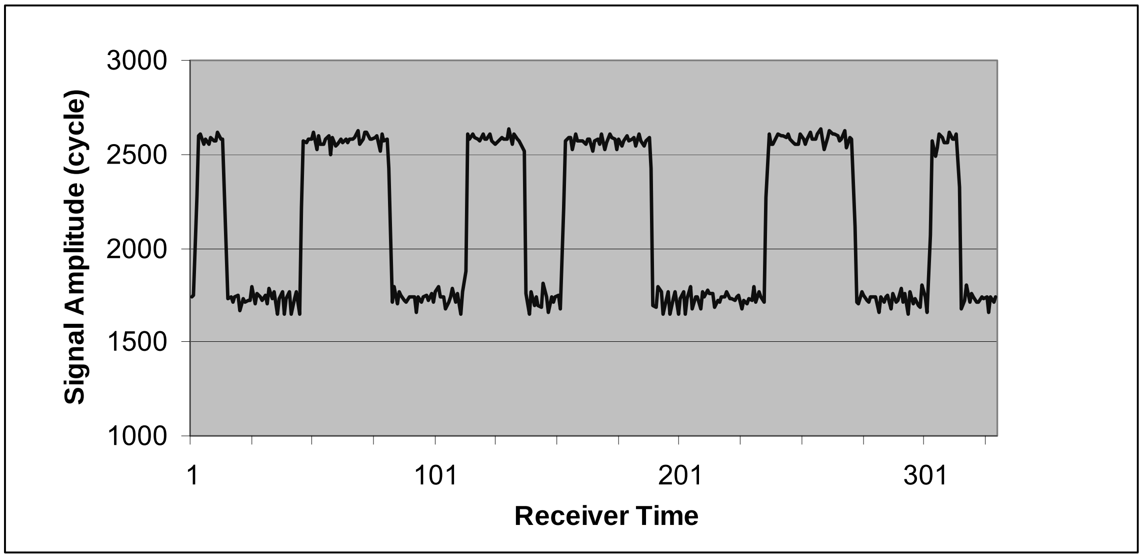


Figure 5.2. SMT/FU channel: observed signal waveform

The SMT/FU channel is a very fast covert channel. The competition for resources is performed in *each cycle*, unlike most known covert channels which rely on system level operations for resource management. The system level operations are at a much coarser time granularity than a processor cycle. Also, the SMT based channel is inherently easy to synchronize. This is because the processor clock is the global clock for both the sender and the receiver. In practice, the SMT/FU channel can easily achieve an information rate on the order of megabits per second.

In SMT processors that support more than two threads, the threads other than the sender and the receiver also compete for the shared functional units, thus introducing noise into the channel. However, the SMT/FU channel can still be effective in that:

- Coding techniques such as the error-correcting code which are commonly used in communications channels can also be applied here to overcome errors due to noise and ensure reliable communication.
- The sender and the receiver can minimize the "noise" from other threads. They can choose to utilize the functional units that are usually unused by other threads. For example, in a database system, most services are integer applications. The floating point units therefore are the ideal resources that can be exploited for covert communication.
- The load of a system is often not uniform. When the system load is light, it is very likely that only the sender and the receiver are ready threads in the system. The security of the system can be compromised as long as the receiver gets chances to receive information from the sender, especially when the covert channel is fast.

5.2.2 A Practical Implementation

This section presents a practical implementation of the SMT/FU channel on a 2.8GHz Pentium 4 HT processor. Two implementation issues need to be considered for achieving high information rate: the implementation of MULTIPLY(), NULL() and RUN() as shown in Figure 5.1, and a synchronization scheme.

The design goal of MULTIPLY(), NULL() and RUN() is to make the interference between MULTIPLY() and RUN() as strong as possible and the interference between NULL() and RUN() as light as possible, and at the same time the interference has small variations – less noise in timing measurements. There are two ways for MULTIPLY() and RUN() to make use of `imul` instructions to contend for the integer multipliers: 1) using independent `imul` instructions; or 2) using a chain of dependent `imul` instructions. We tested all four possible combinations and found that a MULTIPLY() implemented with dependent `imul` chain and a RUN() implemented with independent `imul` instructions give the best result. The implementation of NULL() has even more options. In addition to `nop` instructions, any instruction that has little impact on RUN()'s execution can be used to implement NULL(). We tested "`nop`", "`xchg %ebx, %ebx`", "`pause`", and "`mov (%ebp), %eax`" instructions which are known as equivalent to `nop` instructions. Our experiments show that the implementation with `mov` instructions gives the best result. Figure 5.3 shows the implementation code of the three functions.

```
MULTIPLY()

…
asm volatile (“ \
    imul $1,%ecx; \
    imul $1,%ecx; \
    …
    …
    imul $1,%ecx; \
“);
…
```

```
NULL()

…
asm volatile (“ \
    mov (%ebp),%eax; \
    mov (%ebp),%eax; \
    …
    …
    mov (%ebp),%eax; \
 “);
…
```

```
RUN()

…
asm volatile (“ \
    imul $1, %eax, %ecx; \
    imul $1, %eax, %ecx; \
    …
    …
    imul $1, %eax, %ecx; \
“);
…
```

Figure 5.3. Implementation code of MULTIPLY(), NULL() and RUN()

The bit synchronization can be achieved with the common time stamp counter available in most Intel x86 processors. Both S and R can read the time stamp counter using `rdtsc` instructions and learn the current time in processor cycles. In our implementation, S and R agree in advance that each bit is transmitted over a time interval of 1024 cycles, starting from time “xxx…xx0000000000” to “xxx…xx1111111111”. To send a bit, S reads the timer and starts to run a loop of MULTIPLY() or NULL(), depending on the new bit that should be sent, when he sees the start of a new transmission interval. S stops the current transmission until he observes the start of the next transmission interval and starts the transmission of the next bit. To receive the transmitted bits, R runs a loop of RUN() and keeps observing the timer. Whenever he sees that the boundary of a transmission interval is passed, he records the current received bit and starts to receive the next bit. If for some reason S observes a skip of one or more transmission intervals, S skips the corresponding number of bits so that the number of bits sent is always correct. Similarly, R skips a proper number of bits if he observes a skip of some transmission intervals. In this way, S and R are always synchronized. Such a channel is a binary erasure channel [271] since the locations of the corrupted symbols (the skipped transmission intervals) are known. The channel capacity of this channel is (1-P_e) where P_e is the probability of symbol corruption. In our experiments, we did not see any skip of transmission intervals before a context switch occurs. Also, the noise in the received signal is very low.

In this implementation, since it takes 1024 cycles to transmit a bit and processor clock rate is 2.8GHz, the information rate is $2.8 \times 10^9/1024 \approx 2.7$Mbps. The code can be further optimized and achieve a higher information rate.

5.2.3 Countermeasures of SMT-based Channels

SMT-based channels utilize the parallel execution of the sender and the receiver threads and the tightly shared on-chip resources. One can attack either one of these two points to mitigate the problem.

Software/system level approaches: As the OS scheduler controls when a thread is able to run on which processor, it is natural to mitigate the SMT-based channels at the system level by disallowing parallel execution of the sender and the receiver threads. The following approaches can be applied depending on needs:

- Disabling SMT: The simplest and most straight forward mitigation technique is to disable the SMT feature of a processor. This can be done either at the hardware level by choosing non-SMT mode (e.g., in Intel's HT processors), or at the software level by the scheduler which at any time only schedules one threads on the processor. The advantage of this approach is its simplicity and low cost in implementation. For example, the HT feature of Intel processors can be disabled in BIOS, i.e., neither the operating system software nor application software need to be modified to support this mitigation method. It however may cause performance degradation due to the underutilization of processor resources. The degree of performance degradation depends highly on the type of the workloads of the system. According to previous work on the evaluation of SMT implementation of IBM Power5 [290], disabling SMT may lead to a performance loss of up to ~40% in some cases while achieving a performance gain of up to ~10% in some other cases. Similar observations were also found in Intel's HT implementation. Disabling SMT therefore may be an acceptable solution in some systems while in other systems a better solution is needed.
- Advanced scheduling schemes: In the literature, Lattice scheduling [68] was proposed to reduce the capacity of traditional covert channels in a system. With lattice scheduling, the processes of the same security class tend to be scheduled together and the transitions between different classes are minimized. This slows down the transmission procedure of the covert channel. A similar idea can also be applied here to mitigate SMT-based covert channels. The scheduler always schedule processes of the same security class to run simultaneously on the processor chip. When there is no other process of the same class ready to run, only the current process is scheduled to execute. Such an advanced scheduling scheme would minimize the waste of resource utilization and lead to better performance.

Processor level approaches: In addition to disallowing parallel execution of the sender and receiver threads, SMT-based covert channels can be mitigated by eliminating information leakage due to resource sharing. A few alternatives of this approach are listed below:

- Resource partitioning: The interference between simultaneous threads can be eliminated by partitioning the originally shared resources and allocating different partitions to different threads. In processors, resources can be designed to have a "partitioned mode", and when two threads of different security classes are executing in parallel, the resources are partitioned rather than shared. This may require new architecture design, e.g., the partitioned cache or PLcache (described in Chapter 3), or modifications of existing features, e.g., the fairness control logic available in the processor.
- Non-interference sharing policy: If the receiver thread always has higher priority in contending resources, i.e., he always wins in competing for resources, he will not be able to sense the existence of the sender thread. We refer to such a resource sharing policy as the non-interference policy. Such a policy however

contradicts the concept of fairness, and may not be a practical solution in many systems.

- Randomization: Information leakage due to resource sharing can also be mitigated via randomization. Unlike resource partitioning, resource sharing is still allowed but the interference that causes information leakage is randomized such that it carries no useful information. A randomization based approach therefore can avoid resource underutilization due to partitioning, and can achieve good performance. Examples of this are the cache interference randomization in the RPcache and the Newcache (described in Chapters 3 and 4). Randomization techniques, however, are not generally applicable. For example, randomly allocating function units to different threads will not get better performance and security when compared to static partitioning of function unit resources.

In addition to the above mitigation techniques that avoid or reduce resource sharing, the channels can also be mitigated by making the signal observations noisy. Since most covert channels in processors take advantage of the high resolution on-chip time stamp counter, making the timer unavailable to threads when necessary or reducing the resolution of the timer can be a generally effective way to reduce the information rate of covert channels. Most modern processors provide a way to control how the on-chip timer is accessed. For example, in Intel x86 processors, the operating system can disallow a user-level program to access the on-chip time stamp counter by setting the time stamp disable (TSD) bit in register CR4. Providing a low resolution timer can be achieved with a software method for an existing processor, or with a hardware method that requires modification of current processor design. In the first case, the operating system can set the TSD bit and force a user-level program to learn time through an API call which provides the low resolution time. In the second case, a simple implementation of a low resolution timer is to return the value of the on-chip time stamp counter with a proper number of LSB bits cleared.

5.3 Covert Channels due to Control Speculation in IA-64

To hide the long latency that load instructions may introduce, control speculation in IA-64 allows a load instruction (`ld.s`) to execute speculatively [291] – this means that the compiler can hoist a load instruction before its controlling conditional branch instruction. Since the load is executed without knowing if it should actually be executed (depending on the result of a conditional branch instruction that precedes it), it may cause an exception that should not occur. This exception thus is deferred and not triggered during the execution of the load instruction, and will be handled at a later time when it is known whether the exception should occur or not. In IA-64, the following mechanism is implemented to allow deferral of exceptions. As shown in Figure 5.4, each general purpose register is extended with a one-bit flag called the NaT (Not a Thing) bit. If the speculative load instruction would cause an exception, the NaT bit of the target register will be set. At a later time, this bit is checked, e.g., by using the `chk.s` instruction, and the recovery code will be executed if necessary.

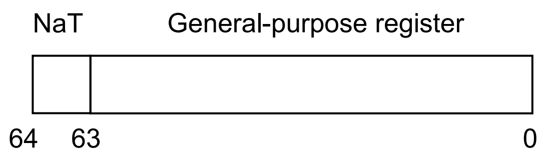


Figure 5.4. 65-bit general-purpose register in IA-64

Figure 5.5 shows a segment of sample code that utilizes control speculation. In a non-speculative ISA, the load instruction can only be scheduled after the conditional branch instruction, while in IA-64 with control speculation, it can be moved far ahead of the branch. The `ld.s` is a speculative load with deferred exceptions. The `chk.s` is a check speculation instruction, which checks the NaT bit of register R1, and triggers any exceptions at that point if they should occur.

```
 Non-speculative ISA

…
…
conditional branch
ld R1 <- [x]
```

```
 IA-64 with control speculation

ld.s R1 <- [x]
…
…
conditional branch
chk.s R1
```

Figure 5.5. Sample code for IA-64 control speculation

5.3.1 Exploiting Scenario

The key idea of control speculation is to defer the exception and allow the program itself to handle the exception. In other words, it makes the exception visible to the program. In IA-64, this is visible through the NaT bit. In practice, TLB misses or TLB access bit violations are typical examples of `ld.s` exceptions which can be deferred [292]. In addition to the deferral of exceptional conditions, some other events, e.g., long latency cache misses, may be deferred automatically by hardware based on implementation-dependent criteria. Such deferral is referred to as spontaneous deferral ([291] vol.2 pp.2:88).

The above mechanism inadvertently opens a covert channel in the system. A sender can encode information by making a change in the system's status, e.g., evicting a page translation of the receiver from the TLB. The receiver can then observe the change by using `ld.s` instructions which will detect the change and set the value of the NaT registers accordingly. Below we assume an IA-64 processor that supports spontaneous deferral of L3 cache misses to illustrate the exploiting scenario. Note that although the current generation of IA-64 processors does not support spontaneous deferral of long latency cache misses, future generations may implement this as indicated in [291] vol.2 pp.2:88, particularly due to the increasingly larger speed gap between the fast processor and slow memory. We wish to emphasize the severity of this channel before real damage is done.

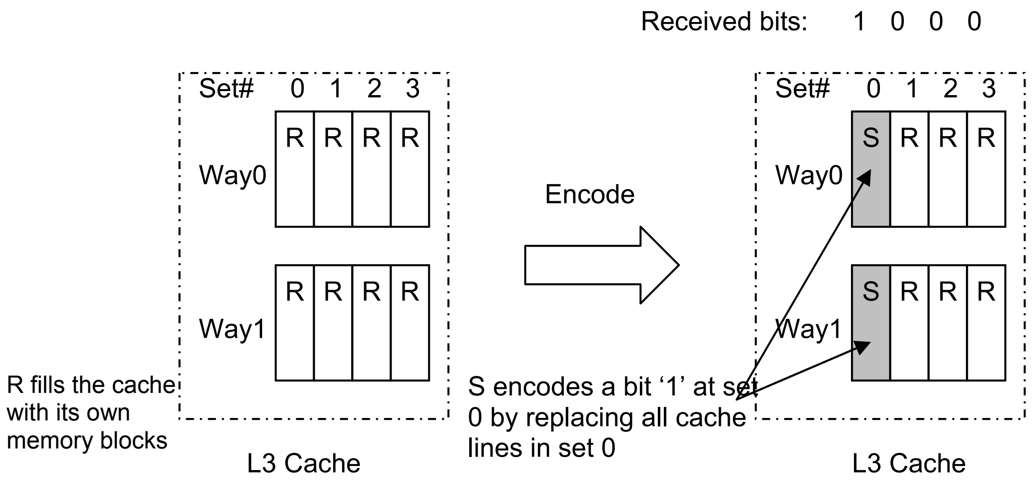


Figure 5.6. Encoding in L3 cache (N=2-way associativity, M=4 sets)

Assume that the processor has an N-way set-associative L3 cache with M cache sets in each way. Without loss of generality, Figure 5-6 shows a simplified example where N=2 and M=4. Assume that the receiver R accesses a large continuous region of memory before it relinquishes the CPU such that the whole cache is filled with R's memory. We also assume that only the receiver R and the sender S exist in the system and S will then gain the CPU. S can then selectively replace the cache lines with its own memory by accessing its memory at certain addresses. For example, S can access `0x`………`0` and `0x`………`1` to replace both cache lines in the first cache set with its own data. Assume that S relinquishes the CPU after it does this and R runs again. R then tries to load `0x`………`0`, `0x`………`2`, `0x`………`4` and `0x`………`6` to R1, R2, R3 and R4 respectively, using `ld.s` instruction. Because the first cache set now only contains S's data, a miss will occur when R accesses `0x`………`0`. Therefore the NaT bit of R1 will be set. Since all other cache sets still keep R's memory, the other 3 loads of R will hit on the cache and the NaT bits of R2, R3 and R4 will remain 0. Using the `tnat` (Test NaT) instruction, the NaT bits can be moved to predicate registers and can then be moved to general purpose registers using `mov` instructions. In this way R receives a string of "1000" from S.

In summary, the following operations are required to set up the covert channel. Assume that initially the cache is filled with R's memory blocks.

S's operations: To encode a bit '1' at a cache set *i*, S manages to access memory at proper addresses such that all cache lines in cache set *i* are replaced by S's memory blocks. To encode a bit '0' at a cache set *i*, S leaves the cache set unchanged. After all cache sets have been encoded, S relinquishes the CPU.

R's operations: R first decodes the bits that S encoded at each cache set by using `ld.s` instructions to access a continuous region of memory. If a bit '1' is encoded at a cache set, the corresponding `ld.s` instructions will set the NaT bits of the target registers. R then uses `tnat` and `mov` instructions to move the decoded bits to general purpose registers. After all bits have been decoded, R manages to fill the whole cache with its memory again using non-speculative memory access instructions. R then relinquishes the CPU.

In this way each time S can send a "packet" of M bits of information to R. In the above discussion, we assumed that S and R are the only threads in the system. When other threads exist, including the OS itself, they may also "pollute" the cache by replacing cache lines with their own memory contents. Furthermore, we ignored the cache line replacements caused by S and R's own codes. All these issues will introduce "noise" to the channel. However, as the L3 cache usually has high associativity, it is inherently resistant to such noise. For example, the memory accesses of other threads may cause some cache lines of a cache set to be replaced. However, at a cache set, unless all cache lines are replaced in the cache set, at least one of R's `ld.s` instructions accessing that cache set will still hit in the cache. If during the decoding of the bit encoded in a cache set, a 1 is decoded only if all R's accesses to the cache set miss, the effect of evictions causing noise is minimized. In a cache with high associativity, the probability that a bit is flipped, i.e., all cache lines in a cache set are replaced, is low. Furthermore, error-correcting code can also be applied here, which can provide further protection against noise.

Another assumption we made in the discussion is that S and R are scheduled on the CPU one after the other. In other words, each "packet" sent by S will be received by R. However, in a real system, R may be scheduled on the CPU twice without S being scheduled in between. Then R will receive one extra "packet". It is also possible that S is scheduled on the CPU twice without R being scheduled in between. In this case a "packet" is dropped. Fortunately since usually the M parameter of an L3 cache is relatively large, a few bits in the packet can be encoded as a sequence number. Using this method, both the "inserted" packets and the packets that are dropped out can be detected and therefore the synchronization is not a problem. Such a channel is indeed an erasure channel [271] which is well known in communication theory. A formal discussion on the impact of the asynchronism of S and R will be presented in chapter 6.

5.3.2 Information Rate Estimation

In this section we estimate the peak information rate of the speculation-based covert channel, i.e., we assume that S and R are scheduled one after the other and they are the only two threads ready to run, e.g., when the load of the system is light. The effect of the scheduling algorithm can be estimated separately, as shown in chapter 6.

In a highly associative cache, using all cache lines in a cache set to encode a bit may be overkill since the probability of the "noise" causing all cache lines in a set to be evicted is very low. To avoid unnecessarily high overhead for bit encoding, we assume that L out of N cache lines in a cache set are exploited by S. The value of L can be determined by examining the average conflict miss rate of programs. For example, it has been observed in the past that given a fixed cache capacity, increasing the cache associativity to above a certain number (e.g., 4-8) does not help much to reduce conflict misses, i.e., the probability that a program occupies more than that number of cache lines in a cache set is low. We therefore assume that L is no greater than 8. We also ignore the bits that are encoded as the sequence number. Let D denote the number of cycles needed to replace a cache line. Then to send a packet, the number of cycles that S needs can be calculated as:

$$T(S) = b \cdot L \cdot D \tag{5.1}$$

where b is the number of '1's in the packet. To calculate the number of cycles needed by R to receive a packet, three time components need to be considered: the time for decoding a 1 bit, the time for decoding a 0 bit, and the time for refilling cache lines that have been evicted. We assume that R performs cache miss detections and cache line refilling in a single scan to save time. Since in modern high performance processors, the time for accessing slow memory (to refill evicted cache lines) is usually overlapped with the time for cache hits and cache probing, the time corresponding to cache accesses that hit in the cache and the speculative loads that probe the cache for miss detection can largely be hidden in the time that is spent in accessing main memory. We therefore make an approximate estimation:

$$T(R) \approx T(S) = b \cdot L \cdot D \tag{5.2}$$

The peak information rate can then be calculated as [17]:

$$R_{\max} \approx \frac{Mf}{T(S) + T(R) + 2T_c} \tag{5.3}$$

where T_c represents the cycles needed for process context switch and f is the operating frequency of the processor.

To have a sense of the information rate in real systems, we consider the following system: the L3 cache is 16-way associative and 2MB in size with 128-byte cache lines. Therefore N=16 and M = 1024. Assume that $L = 4$ and on average half of the bits in a packet is '1's, i.e., b = M/2 = 512. D is estimated as 200 cycles which is a normal memory access delay. We assume that on average context switch time T_c is around 10000 cycles and the processor operates at 2GHz. With these parameters we can calculate R_{max} as:

$$R_{\max} \approx \frac{1024 \times 2 \times 10^9}{512 \times 4 \times 200 + 512 \times 4 \times 200 + 2 \times 10000} \approx 2.4 \times 10^6 \ bps$$

5.3.3 Countermeasures

A straight forward method to block the speculation based channels is to disallow a process to handle exceptions by itself. This requires that the processor provide mechanisms that allow the OS to control the ability of processes on exception handling. Fortunately, the existing IA-64 processors (i.e., Intel's Itanium processors) allow the OS to switch the processor mode which deals with exceptions in different ways. For example, the Itanium processor can be configured in a *no-recovery* mode which only defers fatal exception conditions. In other words, the application code will not be able to handle most exceptions, including the TLB/cache misses. A drawback of this countermeasure is that it may introduce considerable performance degradation. Compilers that make heavy use of speculations to expose more parallelism [293-294] may need to handle exceptions frequently and therefore rely on light-weight exception handling mechanisms to avoid high performance penalty. Having the OS handle exceptions would significantly increase the exception handling overhead. A possible way to circumvent this problem is to augment the processor with the finer-grained control of the exception handling mode. Instead of having a single switch that controls the entire chip, if each hardware context, representing a process or a thread, is tagged with its capability in handling exceptions and the hardware reports exceptions differently based on the tag, the OS can avoid the low performance no-recovery mode whenever possible. For example, in a Multi-Level Security (MLS) system where each process has a security level, e.g., a High security level or a Low security level, to prevent information from being exposed to the Low security processes which are untrusted, the OS may assign no-recovery mode only to the Low security processes. The OS can optimize performance even further, by tracking the system resource usage. For example, if at a moment all processes that are actively sharing the system resources belong to the same security level or security domain, there is no need to worry about information leakage between these processes, and the OS can allow all processes to enjoy the light-weight exception handling mechanism. The OS needs to selectively assign no-recovery mode to processes when processes from different security levels or domains are active at the same time.

Note that as mentioned in chapter 1, control speculation is a mechanism that causes leakage by event reporting, i.e., it does not produce the events that leak information but only exposes the events that were originally not visible. For a given channel medium, information can be extracted using different observation mechanisms. For example, although the information embedded in the L3 cache can be observed using control speculation, disabling control speculation does not really close the channel – the information can also be observed based on cache access timing. Therefore although closing the observation mechanisms is necessary – it makes it harder for the observer to extract information from the channel medium, it is only part of the efforts that eventually close the covert channels.

5.4 Remarks

Compared with traditional covert channels, the new covert channels in processors that we described are extremely fast. Table 5.1 shows a comparison of the information rates of new covert channels in processors and traditional OS-level and hardware-level covert channels. Three traditional covert channels are included for comparison. The bus contention channel [61] exploits the shared bus in multiprocessor systems. The inode table channel encodes a bit by making the inode table either full or not full. The upgraded dir channel uses the existence of a folder to indicate if a 1 or a 0 is sent.

Table 5.1. Information rate[1] comparison of covert channels

Architectural level covert channels			**Traditional covert channels**		
SMT/Cache	SMT/FU	Control Spec.	Bus-contention channel	Inode Table Channel	Upgraded Dir Channel
~3.2M[2]	~2.7M	~2.4M	~1K	~50	~0.5

[1] in bps (bits per second).
[2] Data obtained from [121] on a 2.8GHz Pentium 4 HT processor.

Note that the information rates of traditional covert channels were obtained in computers in 1990's and the same channel in modern computers should be much faster. However, even if we assume a linear increase in such traditional covert channel rates with a 100X increase in processor clock rate, the processor-based covert channels are still orders of magnitude faster than the traditional hardware-based (bus contention channel) as well as OS-based covert channels.

Also note that although this chapter mainly focuses on covert channels, the same information leakage mechanisms that lead to fast covert channels may be exploited in side channel attacks as well. For example, in the access-driven cache based side channel attacks, the attacker currently relies on timing measurement to distinguish cache hits and cache misses, which is noisy and not 100% accurate. If however in a processor the control speculation mechanism allows the detection of long latency cache misses (e.g., via supporting spontaneous deferrals), the attacker would then be able to distinguish cache hits and misses without any noise or errors, e.g., by using the same exploiting method as described in section 5.3.1. This would greatly help improve the cache based side channel attacks. In general, control speculation provides a reliable way for attackers to collect information like various hardware level events that were previously invisible to software, and hence may improve existing side channel attacks or even introduce new attacks.

We also note that although the SMT/FU based channel is not as helpful as the control speculation based channel in exposing information that would allow the inference of data or address information of a program, it may still reveal information about the instruction mix, e.g., helping the identification of different operations in crypto algorithms like squaring operations versus table lookups, if these operations are significantly different in terms of instruction mix.

5.5 Summary

This chapter presents our work on the identification and analysis of new fast covert channels in processors. Two classes of covert channels are discussed, including the SMT-based covert channels and the control speculation based covert channels. SMT is a processor architectural feature that allows multiple threads to execute simultaneously on the same chip, sharing and competing for most on-chip resources on a per cycle basis, which maximizes the resource utilization. SMT allows the sender and the receiver of a covert channel to run in parallel, avoiding bit transmission overhead due to expensive context switches. At the same time, the tightly-coupled resource sharing in SMT processors allows faster interaction between the sender and the receiver through resource contention, i.e., information bits can be transmitted and received more quickly. These two factors significantly increase the information rate of covert channels in SMT processors. Another processor architectural feature that enables fast covert channels is control speculation. Control speculation allows a program to handle various types of exception events, as well as other events such as long latency cache misses, by itself, i.e., these originally invisible events are now directly visible to the program. Control speculation therefore provides a convenient and noiseless observation mechanism for covert channels. To illustrate the severity of these two types of covert channels, two specific covert channels are presented: the SMT/FU channel exploits the shared functional units (the integer multipliers) and the control speculation based channel makes use of the L3 cache as the channel media. The analysis and experiments show that these two channels can achieve very high information rates – on the order of mega bits per second – which is orders of magnitude faster than traditional covert channels. This chapter also discusses possible software and hardware countermeasures for both classes of covert channels.

Chapter 6

On Covert Chanel Modeling and Analysis

6.1 Introduction

Despite decades of work in the area of covert channel modeling and analysis, there are still some fundamental questions that have gone unanswered in the past. One of these is about the classification of covert channels as storage channels versus timing channels. While this classification was widely used in practice, there are covert channels that are hard to classify. Researchers admitted that the difference between storage channels and timing channels is unclear. In this chapter, we will present an abstract channel model which helps improve the understanding of the nature of covert channels and clarify some misconceptions. It can also help identify new channels.

Another question that this chapter will cover is about the capacity estimation of covert channels. In the literature, covert channels are often modeled as certain forms of communication channels, to which information theory can then be applied for calculating channel capacities. There are also experimental methods that directly estimate the channel bit transmission rate based on measured results as well as some best-case assumptions. Both types of methods assume that the channels are synchronous or the synchronization overhead is negligible. This assumption is valid in the sense that the goal is to estimate the maximum attainable information rate over the channel. However, one significant difference between covert channels and real communication channels is that covert channels are not actually "channels" intended for communications. They are typically not synchronous and the asynchronism is indeed an inherent property of covert channels. To fully capture the properties of covert channels, the impact of asynchronous nature on capacities also needs to be characterized. The existing capacity estimation methods obviously can not capture the asynchronous aspect of covert channels due to their assumption of synchronous models. In this chapter, we address several fundamental questions that were not answered in the literature, which however allow us to fully characterize covert channels in channel capacity estimations.

6.2 Storage Channel and Timing Channel Revisited

Intuitively, the key difference between covert storage channels and covert timing channels is whether the channel exploits timing characteristics. This explanation however is vague: the exact meaning of "exploiting timing characteristics" can hardly be defined accurately. Indeed, the definition of "time" itself is vague. When the sender and the receiver do not have access to time references, which is often the case in covert channels, they have to derive their own view of time using other methods, e.g., via the observation of some events. Their view of time may be totally different, and can be uncorrelated to the real physical time. Such ambiguity has led to questioning of the classification in the literature [26]. Some researchers even believed that there is indeed no difference between these two types of covert channels [17]. Without a clear definition of storage channels and timing channels, it would be impossible to answer questions such as "are all storage/timing channels identified? Can all storage/timing channels be identified?"

Another reason that makes the classification of some covert channels hard is that such channels exhibit characteristics of both. For example, in the disk arm channel described in chapter 2, the information is embedded into the position of the disk arm, which is a typical characteristic of storage channels, while the extraction of the information is by comparing the completion timing of a few disk accesses.

In the subsequent sections, we will present an abstract model of covert channels, which consists of two parts: 1) basic resources and mechanisms that allow information leakage in processors as the components of a communication channel, and 2) a channel use model. The channel use model addresses the issue with regard to the definition of time and models the asynchronous nature of covert channels. We will then illustrate how existing work can fit into this proposed model, with confusions clarified, and propose a new categorization of covert channels.

6.2.1 Information Leaking Mechanisms in Processors

Since information leakage is a form of information transfer and distribution, it can be considered as a communication problem. We model an information leakage channel as a communication channel, as shown in Figure 6-1. The proposed channel consists of a sender, a receiver, and the channel medium. The sender modulates information onto the channel medium through a *modulation mechanism*, and the receiver extracts the information from the medium through an *observation mechanism*.

Despite the similarity between the information leakage channel and a traditional communication channel, there are several key differences. First, the sender in an information leakage channel may not always transfer information on purpose. For example, in side channel attacks, the sender is the victim that by no means wishes to transmit information and the information is leaked out unintentionally. Second, unlike in communication systems where the mechanisms for accessing the channel medium are optimized for the sender and the receiver, in information leakage channels, the sender and the receiver may not have much control on what mechanisms they can use to access the channel medium, and the mechanisms (i.e., the modulation and observation mechanisms) are not designed for communication and often not suitable for communication. The focus of this work therefore is not how to design an efficient communication system, but what modulation/observation mechanisms are available in a processor and how these mechanisms can be exploited for information leakage.

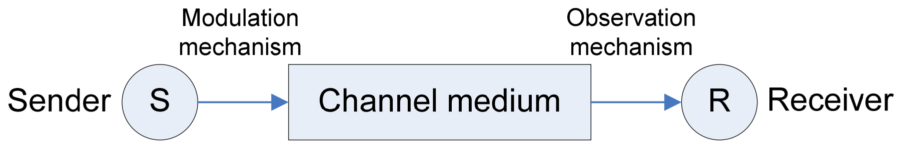


Figure 6-1. An abstract channel model for information leakage

6.2.1.1 Types of Channel Mediums

The ways with which the sender encodes information into the channel medium and the receiver extracts information from it, often depend on the channel medium’s properties. The channel medium, i.e., the resources in processors, can be categorized as two types:

Stateful resources: A stateful resource is one where its use has effect on later uses of the resource. For example, whether a memory access will hit or miss in caches is dependent on previous memory accesses. Examples of stateful resources include all kinds of caches such as data and instruction caches, TLBs, branch predictors as well as other components with a memory effect.

Stateless resources: A stateless resource is one where each use is independent of previous uses. For example, the use of buses and combinatorial functional units with no memory has no effect on their later use.

6.2.1.2 Modulation Mechanisms and Observation Mechanisms

Modulation mechanisms allow the sender to modulate information on the channel medium. The information can be modulated through *temporal encoding* or *spatial encoding*. In temporal encoding, the information is encoded into the amount of time a resource is used. For example, in a uniprocessor system, a process can modulate information over its CPU time, e.g., by using a longer CPU time to indicate a 1 and a shorter CPU time to indicate a 0. In spatial encoding, the information is expressed with the spatial status of the resource. For instance, the sender can encode information into the cache by selectively evicting cache lines. Figure 6-2 shows an example that encodes four bits into a direct mapped cache with four cache lines. The receiver first manages to initialize the cache state by loading his data into the cache to occupy all cache lines. The sender then selectively evicts the receiver’s cache lines: the eviction of a cache line means a 1 and no eviction means a 0. In this example, the encoded bits are 1011.

Similarly, the observation mechanisms may extract information by exploiting temporal characteristics, or spatial characteristics of the resource. Temporal observation mechanisms always involve timing measurements while spatial observation mechanisms do not. A good example of the use of temporal observation mechanism is the detection of cache misses. By using the `rdtsc` instruction (read timer) in x86 processors, a process can measure the number of cycles it takes for a memory access to complete, which indicates whether a cache hit or miss has occurred. A good example of spatial observation mechanisms is the reporting mechanism mentioned in section 1.3.2. For instance, in IA-64 processors that support control speculation, a process can directly learn if a page fault has occurred by checking the value of a NaT (Not a Thing) register.

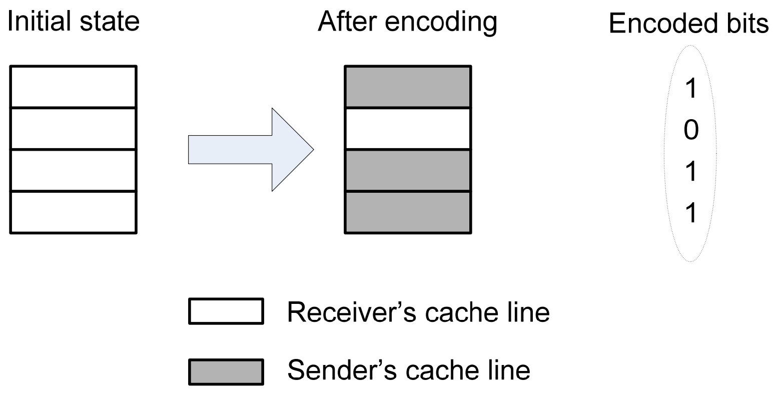


Figure 6-2. Spatially encoding four bits into a four-entry direct mapped cache

The type of observation mechanism used to extract information can be independent of the type of the modulation mechanism used to encode the information, i.e., spatially encoded information may be extracted with both spatial and temporal observation mechanisms, and temporal encoded information may be extracted with both temporal and spatial observation mechanisms. For example, to extract a bit expressed by the presence of a page in memory, a process can detect if an access to the page generates a page fault by measuring the access delay – a temporal observation mechanism, or by checking the value of a NaT register in an IA-64 processor – a spatial observation mechanism.

The modulation and observation mechanisms are closely related to the leakage by resource use and the leakage by event reporting discussed in section 1.3.2. Indeed, the mechanisms that leak information during resource use can always be used as modulation mechanisms since they allow information about object values and addresses to be modulated over the resources being used. The reporting mechanisms on the other hand provide ideal spatial observation mechanisms as the information embedded in the channel medium can be directly read out from architecturally-visible registers. Compared with other observation mechanisms, e.g., those employing timing measurements, the reporting mechanisms allow convenient and noiseless observation.

The type of channel medium also has an impact on how modulation and observation mechanisms can be used. Compared with stateful resources, stateless resources usually have more restrictions. Since stateless resources cannot remember the encoded information, the receiver normally has to be able to run in parallel with the sender so that observations can be made in time before the encoded information disappears. For example, in the bus contention channel, the sender and the receiver have to be running simultaneously on two processors that share the bus, both contending for the bus to modulate information over the bus usage as well as observe the information from the bus usage. The use of a stateful resource does not have such restrictions and therefore can be used in more situations.

6.2.2 A Channel Use Model of Covert Channels

A fundamental difference between a covert channel and a real communication channel is that the sender and the receiver in covert channels often don't have the same "view" of time, e.g., the time elapse rate observed by the sender and the receiver can be totally different and uncorrelated to physical time. Another significant difference is that the sender and the receiver in covert channels are typically much more restricted in accessing the channels, e.g., they often don't have the control on when they can use the channel and may not be able to perform an operation when they wish to. The reasons are twofold. Firstly, as a commonly adopted countermeasure against covert timing channels, good time sources are typically not allowed to be exposed to the sender and the receiver (e.g., user processes in operating systems). Secondly, in computer systems, the resource management is typically out of the sender and the receiver's control, i.e., the sender and the receiver may not use chip resources or make an operation unless the resource manager allows them to do so. This is particularly true in systems where covert channels are a concern. For example, in such systems not only the resource manager (e.g., the process scheduler) is not under the sender and the receiver's control, very often specific methods like lattice scheduling [68] and fuzzy time [25] are also adopted to further mitigate covert channels. As a result, in a covert channel, it may be very hard for the sender and the receiver to reliably cooperate with each other. Also they often can not rely on time to gain synchronization, e.g., by agreeing on a fixed operation interval, since their view of how time elapses may be totally different.

Due to the above reasons, we model the sender and the receiver as entities that operate independently in transmitting and receiving bits and do not have the ability to reliably coordinate with each other. The access patterns to the channel that the sender and the receiver may have are determined by the nature of the system (e.g., the specific scheduling algorithm that is being used) and therefore are considered as a property of the channel instead of something under the sender's or the receiver's control. Note that the sender and the receiver may still be able to coordinate if there are mechanisms (e.g., a feedback path) outside the covert channel being discussed, but with only the forward channel as depicted in Fig 6.1, they can't. The effect of mechanisms allowing coordination will be discussed in section 6.3.

With such a channel model the channel is clearly not synchronized. Bits sent by the sender may drop out – if the receiver does not make an observation promptly, and extra bits may be inserted – if the sender does not send a bit between two adjacent channel observations at the receiver side. Also the sender and the receiver are not aware of bit insertions and dropouts as they don't know and cannot predict if the sender has sent a bit or the receiver has made an observation on the channel. Note that since the error rates of the channel, e.g., the bit dropout/insertion rate, are determined by the sender and the receiver's asynchronous access patterns to the channel, they indeed represent the asynchronous aspect of the channel.

An implication of not having a common interpretation of time is that the meaning of "exploiting timing characteristics" is vague and can hardly be defined accurately. For example, a sequence of events that have temporal effects on the sender may become a single event that carries no timing information at all in the receiver's view – if during the time when the events happen the receiver has a much slower "clock" it may only see the accumulated results of all the events. The sections below rigorously define time used in our channel model and describe how channel symbols are constructed – which relies on the definition of time.

In the broadest sense, as suggested in [26], “the passage of time can be characterized by sequences of events which can be distinguished one from another by the observer.” For example, the time presented by a time stamp counter in a processor can be viewed as a sequence of events which increment a value, and each event is distinguishable due to the corresponding counter value. We adopt this view of time and the definition of a clock as a time reference.

In a covert channel, the sender and the receiver need to find time references (i.e., sequences of events, which may not resemble a real clock at all and may be totally uncorrelated to real physical time) to construct their own view of “time”. To obtain the current time, the observer needs to make an observation and compare the observation event with the time reference event sequence. The order of the events provides the observer with the current time. In other words, if the observation occurs between events T_i and T_{i+1}, the observer knows that the current time $T_i < t < T_{i+1}$. The difference between T_i and T_{i+1} is the resolution of the “clock”.

The above discussion shows that, all forms of timing measurements in essence are the comparison of the order of events. We therefore define:

Definition 6.1: Given an observer and his reference event sequence $T = <T_i>$, the observer’s view of the time can be defined as the comparisons of the ordering of the observation events O_i and the events in T.

Note that the choice of the event sequence is often channel specific and may vary significantly. Some channels may seek event sources that approximately represent physical time. For example, the sender or receiver can construct his own timer by launching a thread that self-increments a counter by executing a loop. Other channels may choose an event sequence that does not resemble a timer at all. In the disk arm channel where the order of completion of two disk accesses carries the information, the completion event of one of the disk accesses itself can be chosen as the reference. Such an event sequence does not really represent the real time.

With this definition of time, we can clarify the difference between covert storage channels and covert timing channels. For clarity of the discussion, we model a computer system as an abstract machine where a process is modeled as an active *subject* and its states are modeled as passive *objects*. A subject consists of sequences of *operations* that manipulate the *value* of objects.

From the observer’s point of view, information can only be delivered to it via the things that it can “see”. By “see”, we mean any means by which the observer can learn the value of an object. We then define an observer’s visible space as follows:

Definition 6.2: The *visible space V* of an observer is the set of all objects that the observer can *see*. $V(i)$ is a snapshot of the *visible space* at the i-th observation, i.e., O_i, made by the observer.

In a covert channel, to transfer information to the receiver, if the sender is able to change the value of an object in the receiver’s visible space V_R, he can directly encode information in the value of that object. In this case, V_R is the symbol space of the covert channel. For example, if V_R contains a single binary object, the symbol space is {0,1}. If, however, the sender has no way to alter the value of any object in the receiver's visible space, he can not directly encode information into V_R. As illustrated below, he can still transfer information by encoding information into the extended symbol space V_R^2, V_R^3,…, etc., where $V_R^2 = V_R \times V_R$, $V_R^3 = V_R \times V_R \times V_R$,…, and so on. In the binary case, $V_R^2 = \{0,1\} \times \{0,1\} = \{00, 01, 10, 11\}$.

When the sender cannot alter the value of any object in V_R, if he can impact the *order* of value changing events of some objects in V_R, he can encode information into the extended symbol space. Below is an example. Consider a V_R that contains two binary objects a and b, with initial values of 0 and 1, respectively. a is going to have a $0 \rightarrow 1$ transition and b is going to have a $1 \rightarrow 0$ transition. These transitions will occur anyway and the sender has no way to change the transitions. The sender however is able to control the order of these transitions. Let $V_R(i)$ and $V_R(j)$ denote the two snapshots of the receiver's visible space when he makes his i-th and j-th observations and the two transitions are observed in these two snapshots. The sender can encode 1 bit of information into the extended symbol space $V_R(i) \times V_R(j)$ by controlling which transition occurs first, e.g., if a's transition occurs first, a 0 is sent, otherwise a 1 is sent. Let a symbol in $V_R(i) \times V_R(j)$ be denoted as $\begin{pmatrix} a(i)a(j) \\ b(i)b(j) \end{pmatrix}$, where $x(t)$ is the value of object x in the t-th observation. The symbol $\begin{pmatrix} 01 \\ 11 \end{pmatrix}$ would indicate a 0 and $\begin{pmatrix} 00 \\ 10 \end{pmatrix}$ would indicate a 1.

Indeed, the above two types of information transfer methods reveal the difference between storage channels and timing channels. In the first method, to receive a bit of information, the receiver only needs to do a single observation and no comparison of event order is involved. The second method however encodes information in the order of events and requires multiple observations to receive a bit. Since timing measurements are essentially comparisons of event orders, the first type of channels do not involve timing measurements whereas the second type of channels do. The classification of storage channels versus timing channels therefore can be based on this distinction.

Definition 6.3: A covert channel is a storage channel if the information to be transmitted is encoded into the receiver's visible space V_R. A covert channel is a timing channel if the information can *only* be encoded into an extended symbol space V_R^n, $n > 1$.

6.2.3 On Covert Channel Classification

Although section 6.2.2 has clarified the issue of the vague definition of time, there are still covert channels that are hard to classify. These channels, e.g., the disk arm channels, exhibit characteristics of both storage channels and timing channels. Indeed, the difficulty in the conventional classification is due to the lack of recognition of the modulation mechanism and the observation mechanism as we proposed in chapter 3 and shown in Figure 6.1. In a covert channel, the transmission of information always involves two steps: 1) modulating information over the channel medium via the modulation mechanism at the sender side, and 2) extracting information from the channel medium via the observation mechanism at the receiver side. Each step can exploit either spatial or temporal methods.

It is therefore natural for covert channels to have mixed characteristics of storage channels and timing channels.

The above discussion indeed shows that the conventional classification of storage channels vs. timing channels is incomplete. A clearer classification should take both modulation mechanism and observation mechanism into account. We first make the following definitions. Let V_S and V_R denote the visible spaces of the sender and the receiver, respectively. V_S is the set of all objects that the sender is able to access.

Definition 6.4: A modulation mechanism is spatial if the information is directly encoded in V_S, or temporal if the information can *only* be encoded in the extended symbol space V_S^n, $n > 1$.

Definition 6.5: An observation mechanism is spatial if the information can be directly extracted from V_R, or temporal if the information must be extracted from the extended symbol space V_R^n, $n > 1$.

Indeed, the classification based on Definition 6.3 is a classification from only the receiver's perspective. We propose a new classification of covert as follows:

SS channels: covert channels that exploit spatial modulation and spatial observation. Example: the file lock channel [17].

ST channels: covert channels that exploit spatial modulation and temporal observation. Example: the disk arm channel [36].

TS channels: covert channels that exploit temporal modulation and spatial observation. Example: see example 6.1 below.

TT channels: covert channels that exploit temporal modulation and temporal observation. Example: the CPU time channel (Example 2 in Section 2.2.1).

Example 6.1: the selective observation channel

For clarity, we assume a simple system that contains only the sender, the receiver, and a random number generator (RNG). The RNG is visible to both the sender and the receiver, and neither the sender nor the receiver can affect the RNG's output. The sender has no way to communicate with the receiver, but can somehow control when the receiver can make an observation, e.g., by occupying the CPU until he wishes to let the receiver make an observation. The information transfer can then be achieved as follows. The sender keeps observing the RNG and checks if its value equals to the current symbol he wants to send, e.g., if the least significant 3 bits is "000". He will not let the receiver make an observation until he sees that the value to be sent appears in the output of the RNG. In this way, the sender can select the desired values for the receiver to observe. In this channel, in order to send a symbol, the sender has to check if a desired event occurs right before his current observation event, which is a temporal modulation method. The receiver simply reads the output of RNG to receive the symbol, which is clearly a spatial observation method. Hence, this selective observation channel is a TS channel.

6.2.4 Remarks

The above discussion not only clarifies the problem in covert channel classification, but also shows how covert channels can be constructed. This can help us understand the capabilities as well as limitations of existing covert channel identification methods and also identify new covert channels. For example, our definitions and classifications can help to understand which covert channels can be identified and which can not be identified by the popular shared resource matrix (SRM) method [15]. The identification of a covert channel in the SRM method is based on the analysis of subjects' capabilities, M (modify) or R (read), in accessing resources. A potential covert channel from A to B is identified if subject A has M access to a resource and subject B has R access to the resource. This indeed only captures the information transferred in symbol space V_S and V_R. The covert channels in which A does not have M access to the resource but is able to encode information in the extended symbol space ${V_S}^n$ or B does not have R access to the resource but is able to extract information from ${V_R}^n$ can not be identified. In other words, the SRM method can identify SS channels, but may not be able to identify ST, TT, or TS channels. It is possible that the definition of M and R can be made more general to catch certain temporal effects and allow the identification of more covert channels. This however is based more on experience and can not be generally applied. Our new definitions and classification can also help identify new types of covert channels. For example, to the best knowledge of the author, the class of TS channels we defined has not been reported in the literature.

6.3 Capacity Estimation of Asynchronous Covert Channels

6.3.1 Capacity Degradation

As explained in section 6.2, unlike traditional communication channels, covert channels are often asynchronous, and the sender and the receiver may not be able to reliably coordinate with each other, causing symbol insertions and dropouts. For example, consider a uniprocessor system where the communicating subjects are processes. To transmit a symbol, the sender has to make a change in the system and the receiver receives it by detecting the change. As there is only one CPU in the system, at any time only one of the two processes can be active. In other words, the sender has to relinquish the CPU after it sends a symbol so that the receiver can get the CPU to read the symbol. In most operating systems, the scheduler determines when and who can gain the CPU. Depending on the scheduling algorithm, it is very likely that the sender is woken up twice without the receiver being able to run in between, or the receiver is woken up twice without the sender being able to run in between. In the former case a symbol is dropped, while in the later case an extra symbol is inserted.

In addition, unlike in communication channels, coherent time references are often unavailable in covert channels. Coherent time reference plays an important role in many communication systems. Even without other synchronization methods, the operations of sender and the receiver can be synchronized as long as the local timers at the two sides of the channel are coherent enough. Being aware of this, and because time references are known as key components in exploiting covert timing channels, high assurance systems normally disallow processes to access timers and have even made efforts to remove all event sources that can serve as such time references to user processes [25].

Note that although the operations of sending and receiving symbols are asynchronous in most covert channels, one may still synchronize the symbol transmission with certain techniques. Figure 6-3 shows an example. The sender makes a change on the S-R variable once a symbol is sent; the receiver checks the S-R variable and reads the symbol when ready; the receiver then makes a change on the R-S variable to inform the sender; the sender checks the R-S variable and sends the next symbol once the last symbol has been received.

There may be other methods that can maintain the correct order of operations, but in essence they do the same thing: let the sender know if the receiver has read the previous symbol and let the receiver know if a symbol has arrived. With such information, each time when the sender is able to perform an operation it can determine whether a new symbol can be sent. At the receiver side, each time when the receiver is able to make an observation, he is able to tell if a new symbol is ready to receive. However, due to the asynchronous nature of the covert channels, it is very likely that the sender finds that the previous symbol has not been read by the receiver and it has to give up the CPU and wait for the next chance, wasting some time. Similarly, some time is also wasted at the receiver side. The channel capacity is therefore reduced. This observation distinguishes our work from previous work where the synchronous model excludes this part of the time in symbol transmission – only the time associated with transmitted symbols was taken into account.

Intuitively, lack of synchronization should make communication harder. It's however unclear how asynchronism would impact channel capacity, how synchronization would help, and what the cost would be. The sections below address all these questions.

6.3.2 Capacity Estimation

To show the impact of asynchronism on channel capacity, we answer two sets of questions:

1. *Existence and capacity*: Without any form of synchronization, is reliable communication still possible? If the answer is yes, what's the capacity of such channels?

2. *Construction and capacity*: How can reliable synchronization mechanisms be constructed for asynchronous covert channels? What is the capacity of such a synchronized channel? Compared with the capacity of an inherently synchronous channel, what is the degradation of capacity due to the asynchronous effect?

The first question indeed is asking if synchronization is always necessary. Previous work all assume a synchronized form of communication. But it is not clear if it is the only way to achieve reliable communication. In fact, another interesting question is: can an asynchronous form of communication have higher information rate than the synchronous one as the overhead associated with synchronization is totally avoided?

6.3.2.1 Deletion-insertion channel

As explained above, asynchronous operations at the two sides of the channel may lead to loss of real symbols and insertion of false symbols. Such a channel can be modeled as a deletion-insertion channel. We adapt the definition in [295] as follows:

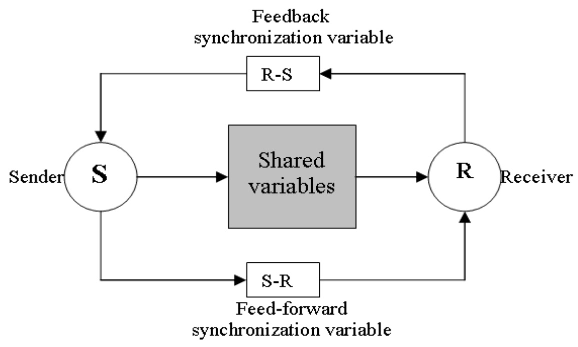


Figure 6-3. Synchronization mechanism using two variables

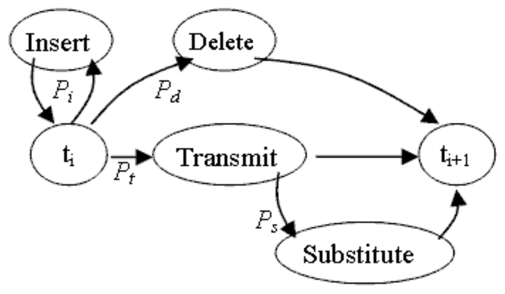


Figure 6-4. Insertion-Deletion channel with probabilities *P_d*, *P_i*, *P_t* and *P_s*, of deletions, insertions, transmissions and substitutions

Definition 6.6: A *binary deletion-insertion channel* is a channel with four parameters: P_d, P_i, P_t and P_s, which denote the rates of deletions, insertions, transmissions and substitutions, respectively.

The symbols to be transmitted are imagined entering a queue, waiting to be transmitted by the channel. Each time the channel is used, one of four events occurs: with probability P_d the next queued bit is deleted; with probability P_i an extra bit is inserted; with probability P_t the next queued bit is transmitted, i.e., is received by the receiver, and with probability P_s of suffering a substitution error (see Figure 6-4).

A deletion-insertion channel should not be confused with an erasure channel. In an erasure channel, channel symbols may be corrupted or lost, which is similar to the substitution or deletion in a deletion-insertion channel. However, the receiver of an erasure channel knows exactly which symbols are corrupted or dropped while in a deletion-insertion channel, the receiver knows nothing about any deletion, insertion or substitution (corruption) of symbols. This makes the recovery of a message much harder.

6.3.2.2 Capacity of deletion-insertion channels

Intuitively, a channel with symbol insertions and drop-outs is hard to use and not efficient. However, as maintaining synchronized communication also introduces overhead, we wish to know how fast the information can be delivered over such channels, compared to the synchronized channels.

Dobursthin [296] first showed that the fundamental theorem of information theory concerning the existence of an upper bound for the transmission rate, for which the error probability can be made arbitrarily small, holds. This implies that reliable communication is possible even if no reliable synchronization mechanisms are available. However, deriving the capacity of such a channel is very hard, and the exact capacity is still unknown after decades of research. A variety of approximations of the capacities and numerical bounds can be found in [297-298].

Despite availability of several capacity bounds, we give an upper bound of the capacity for the purpose of comparison with synchronized channels. Consider a deletion-insertion channel and an erasure channel which are identical except that in the erasure channel the location of symbol drop-outs and insertions are known. Because the two channels are identical except that the erasure channel has more knowledge about the transmission errors, the capacity of the deletion-insertion channel is no greater than the capacity of the erasure channel.

Theorem 6.1. An upper bound of the capacity of a deletion-insertion channel is the capacity of the erasure channel:

$$C_{max} = N(1-P_d) \tag{6.1}$$

where N is the number of bits per symbol and P_d is the deletion probability.

The derivation of the capacity of an erasure channel can be found in many information theory textbooks such as. Note that here we use the term *erasure channel* to refer to the one that is identical to the corresponding deletion-insertion channel except that the locations of symbol insertions/drop-outs are known. In the rest of the paper, we will use erasure channel to refer to this specific erasure channel unless otherwise specified.

The above capacity bounds including (6.1) are very hard, if not impossible, to achieve in practice. Using existing coding schemes such as convolutional code and watermark code, some work [295, 299-300] have shown reliable communication over such channels. However, they all showed that the capacity is quite low and in practice sophisticated coding techniques are required.

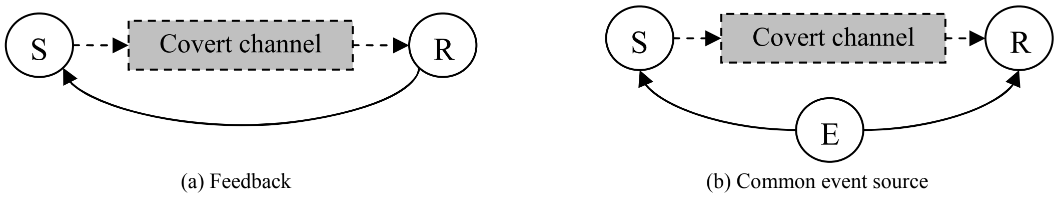


Figure 6-5. Two fundamental synchronization methods.

6.3.2.3 Synchronization and capacity estimation

To answer the second set of questions for constructing reliable synchronization for asynchronous covert channels, we consider fundamental synchronization methods and investigate how these methods impact channel capacity. Synchronization can be achieved with or without feedback. Figure 6-5(a) shows the most general form of a feedback based synchronization mechanism. It consists of only a feedback path without any assumption on how the feedback path is used, and specific techniques can all be explained with this general abstraction. For example, in Figure 6-3, two synchronization variables are used, one serves as the feedback path and the other serves as the feed forward path. The feedback variable is one form of the feedback path in Figure 6-5(a). The feed forward variable can be viewed as a special use of the feed forward channel in Figure 6-5(a): part of the feed forward channel is dedicated for synchronization and the rest of the channel transmits data. Note that such separation of the feed forward channel is not necessary – it is indeed not optimal. Given a fixed bandwidth of the feed forward channel, using a separate feed forward synchronization variable reduces the effective bandwidth for information transmission. This will be proved in subsequent sections.

Synchronization can also be achieved without feedback. The sender and the receiver may luckily share a common time reference (or at least have time references that are coherent at the two sides of the channel) and thus are naturally synchronized. Alternatively, the sender is able to use some method to let the receiver know of his operation timing. Indeed, these two methods are essentially the same from the perspective of synchronization. They both allow the sender and the receiver to have a common view of time, i.e., they virtually have a common time reference, as shown in Figure 6-5(b).

In the following discussion, we assume that the feedback path and the two paths from the event source E to the sender S and the receiver R are perfect. This simplifies the analysis, and is also a requirement for deriving the maximum information rate. To focus on the synchronization problem, we assume that the channel is noiseless, i.e., $P_s = 0$ and $P_t = 1 - P_i - P_d$ where P_d and P_i and Ps are the probabilities of symbol deletion, insertion and substitution, respectively.

A. Channels with feedback

We now show that the capacity of a channel with deletions can achieve the capacity of an erasure channel by utilizing feedback. We then extend the result to channels with insertions.

Lemma 6.1. The upper bound of the capacity of a deletion channel with perfect feedback is the capacity of the erasure channel.

Proof: Consider a deletion channel with a deletion probability P_d and its corresponding erasure channel. Add perfect feedback path to both of them. Since the erasure channel knows where the symbol drop-outs occurs which the deletion channel does not know, the erasure channel knows more information than the deletion channel. Therefore the erasure channel with feedback will gain equal or higher capacity than the deletion channel with feedback. Since an erasure channel is a memoryless channel and it is well known that adding feedback will not increase the capacity of a memoryless channel [271], the upper bound of the capacity of a deletion channel with perfect feedback is the capacity of the erasure channel. □

In Lemma 1 we only show an upper bound of the capacity, we now show that the bound is tight.

Theorem 6.2. The capacity of a deletion channel with perfect feedback equals the capacity of the erasure channel.

Proof: Here we construct a protocol by which the capacity of the erasure channel can be achieved. The protocol is as follows: let the receiver notify the sender via the feedback path once it receives a symbol. The sender will keep resending the symbol until it knows that the symbol has been received. Therefore no dropouts will occur. While the probability of deletion is P_d, a symbol gets through with probability of 1- P_d, therefore the effective information rate is $N(1-P_d)$, which is the capacity of an erasure channel with an erasure probability P_d. Since the upper bound of the capacity (Lemma 6.1) can be achieved, it is the actual capacity. □

When symbol insertions are present in the channel, a theorem similar to theorem 6.2 can be proved. We first define an extended erasure channel as follows:

Definition 6.7: An *extended erasure channel* is a channel where symbols may be inserted and/or dropped but the locations of all insertions and dropouts are known.

Lemma 6.2. The upper bound of the capacity of a deletion-insertion channel with perfect feed back is the capacity of the equivalent extended erasure channel, i.e.,

$$C_{upper\text{-}bound} = N(1\text{-}P_d).$$

The proof is similar to that for Lemma 6.1. Note that the insertion probability P_i does not affect this upper bound. This can be explained as follows. Under the assumption of noiseless asynchronous channels, the symbol insertions are due to the receiver's double (or multiple) channel observations before the sender sends a new symbol. If the receiver checks the channel more frequently, P_i will increase. This however should not change the capacity of the extended erasure channel, given that the symbol transmission rate of the sender does not change. This is because given a time interval, although the receiver receives more symbols due to his higher channel observation rate, the inserted symbols will simply be discarded in the extended erasure channel since the receiver knows exactly which symbols are inserted, and therefore the number of effective symbols received remains unchanged, i.e., the capacity is not affected by P_i.

A lower bound of the capacity can also be derived with a constructive protocol, based on the synchronization variables as shown in Figure 6-3. It can be shown that under certain conditions, this lower bound and the upper bound shown in Lemma 6.2 asymptotically converge.

Lemma 6.3. A lower bound of the capacity of a deletion-insertion channel with perfect feed back is:

$$C_{lower-bound} = (N-1)(1-P_d) \tag{6.2}$$

where N is the number of bits contained in each symbol.

Proof: A simple protocol can be constructed based on the synchronization mechanism depicted in Figure 6-3. Assume that one of the N bits contained in a channel symbol is reserved as the S-R variable which serves as the feed-forward synchronization variable and is initialized to a value that the sender and the receiver have agreed upon a priori, e.g., '0'. The protocol is as follows: let the receiver notify the sender via the feedback path once it receives a symbol and let the sender notify the receiver via the feed-forward S-R variable by flipping its current value once it sends a new symbol. The sender will keep resending the symbol, without changing S-R variable's value, until he knows that the symbol has been received. On the other hand, when the receiver gets a chance to access the channel, he checks the S-R variable. If the value remains unchanged, no new symbol has been sent from the sender, and the receiver will simply wait for the next chance to access the channel. If however the value of the S-R variable did change, he reads the channel and notifies the sender via the feedback path that he received a new symbol. With such a protocol, no drop outs will happen since the sender will not send a new

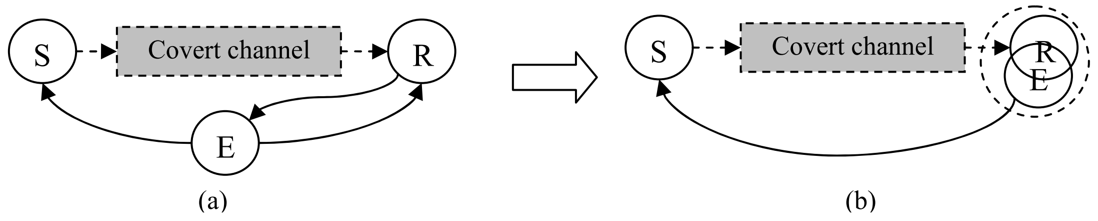


Figure 6-6. Using common events won't get better capacity than using feedback

symbol until the receiver tells him that the current symbol has been received, and no insertions will happen since the receiver can always tell if he is reading an old symbol that he has received. Since in each symbol the effective number of bits used for carrying true data information is (N-1) bit, and the probability of a symbol successfully getting through the channel is (1-P_d), the information rate of the channel with this protocol is (N-1)(1-P_d). Note that insertion probability P_i does not matter here because regardless of how many insertions the receiver has received, the receiver always knows that they are insertions and can simply discard them. It has no impact on the rate that the sender sends symbols as we have explained in the proof of Lemma 6.2. □

The above capacity can be achieved using this protocol and therefore is a lower-bound of the actual capacity. Under certain conditions, e.g., when $N \rightarrow \infty$, this lower bound and the upper bound shown in Lemma 6.2 asymptotically converge. In other words, with such a simple protocol, the theoretical channel capacity can practically be achieved if N is sufficiently large. In systems where N is small, more advanced coding scheme may be used to achieve higher information rate than this lower bound.

Theorem 6.3. A lower capacity bound of Lemma 6.3 and the upper capacity bound of Lemma 6.2 asymptotically converge when $N \rightarrow \infty$.

Proof: the proof is straight forward

$$\lim_{N\to\infty}\frac{C_{lower-bound}}{C_{upper-bound}} = \lim_{N\to\infty}\frac{(N-1)(1-P_d)}{N(1-P_d)} = \lim_{N\to\infty}\frac{(N-1)}{N} = 1 \qquad \square$$

B. Channels with common event source

There may be several ways to exploit a common event source E for synchronization. For example, E can be a self-incrementing counter which serves as a common clock for the sender and receiver. However, as we show below, exploiting E will not get higher capacity than using a feedback path in general.

If one more path from R to E is added, as shown in Figure 6-6(a), E may gain more information. Therefore an equal or higher information rate may be achieved than without the added path. In the best case, E and R communicate with each other without any overhead, i.e., they indeed can be regarded as one single party and such a configuration actually becomes the synchronization method using feedback, as shown in Figure 6-6(b). Therefore a similar system using feedback will get equal or better performance for channel capacity.

6.3.3 Remarks

We have answered the two sets of questions we posed. For the first question, we show that reliable communication over non-synchronous channels without synchronization is possible, but it is not as effective as synchronized communication and requires complicated coding schemes. For the second question, we show that the capacity of an asynchronous channel with either feedback or a common event as the synchronization mechanism has a capacity that is equal or close to $N(1\text{-}P_d)$. Comparing to the ideal synchronous channel which has a capacity of N, that means the capacity degradation due to asynchronous effects is roughly proportional to P_d, the probability of deletions.

According to the above discussion, with a good feedback path, synchronization is not a problem for a covert channel in general. Furthermore, with the help of the feedback, the theoretical capacity of the channel can be practically achieved using a very simple protocol. This has interesting implications for a multi-level security (MLS) system. Since the legal information flow (from low to high) can serve as a perfect feedback path, one may always exploit it to achieve the channel capacity. In other words, covert channels in MLS systems are relatively easy to exploit in general and tend to be fast.

Note that since we've shown that the capacity degradation due to asynchronous effects is roughly P_d, to estimate the capacity of a given covert channel, one could first use traditional methods to estimate the physical capacity C. The probability of deletion P_d should then be estimated. The real capacity can then be estimated as $C(1\text{-}P_d)$. Note also that the capacity degradation modeled in our method is independent of the synchronization mechanisms used and does not include any specific overhead introduced by such mechanisms. Such degradation is inherent due to the asynchronous nature of operations. It is unavoidable even if efficient mechanisms are deployed.

The study of the capacity of asynchronous channels not only provides a more accurate capacity estimation, but also provides an estimation of the covert channel reduction effectiveness of different system implementations. Since the asynchronous effects of the covert channel are often determined by the system implementation, e.g., the scheduler algorithm, the results can also be used to evaluate the effectiveness of candidate system implementations, parameterized with P_d , in reducing covert channel capacities.

Finally, although our results are derived in the context of capacity estimation of covert channels, it may provide meaningful insights to researchers in other areas. Recently some ongoing work [301] in the communication community also shows interest in the capacity bounds of channels with asynchronous behaviors. Although the problems and models are different, similar insights may apply. It would be interesting to study the connections in future work.

6.4 Summary

In this chapter, some fundamental questions about covert channels that went unanswered in the past are discussed and clarified. With respect to the conventional classification of covert channels as storage channels vs. timing channels, two sources of ambiguity are discussed. The vague definition of time is first clarified and a new definition is proposed. Definition 6.1 shows that from an observer's perspective, the view of time can be accurately modeled as the order of the observer's observation events and the reference event sequence. The classification of storage channels and timing channels is then shown incomplete by introducing the concept of modulation and observation mechanisms. Based on the channel model proposed in chapter 3, the mixed characteristics of some covert channels that make the conventional classification difficult are shown as a natural property of covert channels, since both modulation mechanisms and observation mechanisms can exploit either spatial or temporal characteristics. We proposed a clearer classification of covert channels which categorizes them into SS, ST, TS, and TT channels. This new classification can help show the capabilities and limitations of existing covert channel identification methods and can help find new types of covert channels.

With respect to capacity estimation of covert channels, this chapter discussed the impact of the asynchronism of covert channels on channel capacities. We attempt to answer two sets of fundamental questions. The first set of questions ask if synchronization is always necessary and what is the capacity of channels without synchronization mechanisms. Our answer is that reliable communication over asynchronous channels without synchronization is possible, but it is not as effective as synchronized communication and requires complicated coding schemes. The second set of questions ask how synchronization mechanisms can be constructed and what the inherent impact of asynchronism is on channel capacity. We show that synchronization can be achieved by using feedback or a common event source, and regardless of the form of the synchronization mechanism used in a covert channel, the capacity degradation due to asynchronous effects is roughly proportional to P_d, the probability of symbol deletions. Such degradation is inherent due to the asynchronous nature of operations. It is unavoidable even if efficient mechanisms are deployed. With the recognition of this inherent impact, covert channel capacity can be estimated more accurately. Furthermore, as P_d is typically determined by the system implementation, e.g., the scheduler algorithm, it can also be used to evaluate the effectiveness of candidate system implementations, parameterized with P_d , in reducing covert channel capacities.

Chapter 7

Conclusions

Microprocessors as the central processing units of modern computer systems are highly shared by different programs, processes or virtual machines. Like in any shared resources, interference between users often reflect the activities of the users and thus leak out useful information. Even worse, several unique properties make microprocessors an even better place for information leakage. First, microprocessors are fast, and channels based on processor level interferences hence can often be orders of magnitude faster than those at software level. Second, a microprocessor is the most information-rich point for attackers to snoop because any information in the system essentially has to be processed in the central processor, and thus has the possibility to be leaked out. Third, the processor level sharing often breaks software level isolation mechanisms like virtual machines (VMs). For example, two logically isolated VMs can still be running on the same physical microprocessor and share caches. It has been demonstrated in several recent work that sensitive information like cryptographic keys can be leaked out through shared caches. Last, the increasing prevalence of mobile computing and cloud computing makes the information leakage problem even more challenging. User data won't be constrained in local systems forever. Instead, they will be travelling through and exposed to a much wider spectrum of platforms, including public platforms like public clouds. This may significantly increase the risk of leaking out sensitive user information, and users often don't have much control over the problem.

In the past, related research are mostly in the area of side channel attacks and covert channel analysis, which focused on either specific hardware/software targets such as cryptographic devices and software ciphers, or system and software level covert channel issues. Unlike the previous work, this dissertation focuses on architectural and micro-architectural level information leakage problems, which did not receive much attention in the past. The results demonstrate that, in modern microprocessors, there are rich mechanisms that allow covert channels that are much faster than traditional ones, and enable significant side channel attacks that can impact not only embedded devices but also general purpose systems.

7.1 Contributions

This dissertation focuses on processor architectural and micro-architectural level information leakage problems and the contributions are of two categories. The work first investigates real attacks that are of high significance, analyzes concrete problems and proposes novel and effective solutions. The work then generalized the problem with abstract modeling and classification, based on which theoretical analyses are performed. The generalized discussion helps clarify misconceptions that were unanswered in the past and allows better modeling and clearer classification of information leakage channels. The better understanding of the nature of the problem also helps identify new information leakage channels.

More specifically, the main contributions of the dissertation include:

- (in chapter 3): A thorough study of the cache-based side-channel attacks, which are the most significant processor level attacks. This work addresses the problem by identifying and attacking the common root cause of all cache-based software side-channels and thus avoids being attack specific. Solution strategies were identified based on efficient dynamic partitioning (e.g., the PLcache) and sharing with interference randomization (e.g., the RPcache). The latter strategy uses an information-theoretical analysis to show that we can eliminate or mitigate most cache based attacks without compromising performance and has theoretically proved the cache leakage security.
- (in chapter 4): A new cache architecture, Newcache, that is not only secure but also high performance. The proposed new cache architecture inherits the short access time from the Direct Map cache architecture while still maintains low miss rates by employing dynamic address remapping. A simple security-aware random replacement algorithm is also proposed which makes the cache immune to most cache based attacks. Moreover, the proposed architecture can bring additional benefits in terms of fault tolerance, thermal and power optimization, and more performance-friendly cache partitioning and locking mechanisms.
- (in chapter 5): Identification of several new fast covert channels, which are based on popular features in modern processors such as Simultaneous Multi-Threading (SMT) and the IA-64 control speculation feature. The new channels are prototyped and demonstrated to be orders of magnitude faster than traditional channels. Various countermeasures to these channels are also discussed.
- (in Chapter 6): Development of an abstract framework, which allows the information leakage problem to be discussed in a generalized form. The framework is constructed in a way that past work such as the categorization of timing and storage channel, channel analysis and identification methods like SRM, can be related and explained within the framework. The framework is more general and can clarify misconceptions that went unanswered in the past, reveal the limitation of existing work, and help identify a new type of covert channel. It allows more rigorous and accurate channel classification and may better facilitate channel analysis and identification.
- (in Chapter 6): With regard to channel capacity estimation, all past work based the capacity calculation on synchronized channel models (though sometimes

implicitly). This work for the first time pointed out that because covert channels are typically not intended for communications, they are inherently asynchronous and hence there is inherent capacity degradation due to the asynchronism, which was not recognized in the past. To fully understand the effect of the channels' asynchronous nature, this work answered two sets of fundamental questions: 1) is reliable communication possible without any form of synchronization, and if the answer is yes, what's the capacity of such a channel? 2) How can reliable synchronization mechanisms be constructed for asynchronous channels? What is the capacity of such a synchronized channel? What is the degradation of capacity due to the asynchronous effect? Answering these questions not only allow more accurate channel capacity estimation but also helps understand how covert channels can be constructed effectively and how to mitigate them.

7.2 Future work

The first area that future research should continue to pursue is the search for new vulnerabilities and exploits. While many problems have been identified, e.g., those based on caches, branch predictors, control speculation and simultaneous multi-threading etc., more are yet to be explored. For example, despite the intensive studies on the leakage of address information (e.g., through external address bus directly or through cache hit/miss patterns or branch predictors indirectly), the leakage of data flow information was not receiving much attention in the literature. Indeed, architectural features like value prediction and data speculation can leak out information in a similar way as caches do. Similar to the timing effects of cache hits and misses, correct predictions lead to shorter execution time and wrong predictions cause the program to run longer. The data-dependent execution time therefore can leak out information about the data values being processed. Some functional units like the divide units may also exhibit data-dependent latencies and thus are vulnerable to the information leakage problem. More recently, researchers have applied the concept of speculation in arithmetic logic design to improve performance, which essentially optimizes the circuit such that the critical path circuit is fast but does not guarantee 100% correctness, and in very rare cases the result is incorrect which requires extra cycles to recalculate the right result. As a result, functional units like adders that are implemented in this way may also exhibit data-dependent timing and thus can leak out information. Indeed, as the basis of a large variety of performance optimizations, the concept of speeding up the common cases while allowing slow speed in rare cases is fundamentally insecure because the patterns of fast and slow operations almost always leak out useful information. In the literature, all these issues were not investigated and their implications in terms of security are not clear.

Another line of research that future work should pursue is software and hardware defenses that ensure security without compromising performance. The design of cryptographic algorithms that are immune to side channel attacks, on general purpose processors in particular, is one such topic. Another interesting direction is compiler techniques as well as other software approaches that allow not only small kernels like crypto ciphers but also larger software programs to be resistant to information leakage. Software/hardware co-design is also an area receiving more attention, as software and hardware often have strengths that are complementary to each other. For example, in the case of cache based attacks, while hardware design can easily avoid inter-process interference (e.g., via cache partitioning or randomized line replacement), it is hard for hardware to eliminate interferences within a process because the interferences are often due to expected behavior and are unavoidable. For example, if two pieces of code in the same process try to access the same memory address, the first access will interfere with the second access, i.e., by making the second access hit in the cache and thus observe a short access time. This is exactly what a cache is expected to do to improve performance, and thus hardware has no way to avoid this unless caching is completely turned off. From the software side, however, while the software programmer can not control how other processes can interfere with the software she is developing, she has the ability to design the code to be free of interference from itself. Therefore while software or hardware alone can not avoid all cache interferences, software/hardware co-design may together solve the problem.

The third area that future research should explore is the methodologies that allow more systematic analyses of processors for information leakage problems. Systematically analyzing a complicated processor is certainly very hard, however it is also very important as whole system security relies on knowing and defending against attacks in all aspects instead of defeating just one or a few of them. Most existing work are rather *ad-hoc* and attack specific. To the best knowledge of the author, no established work has been reported in this area.

References

[1] M. Steil, "17 Mistakes Microsoft Made in the Xbox Security System," presented at the 22nd Chaos Communication Congress, Berliner Congress Center, Berlin, Germany, 2005.

[2] A. One, "Smashing the stack for fun and profit," in *Phrack* vol. 7, ed, 1996.

[3] C. Cowan*, et al.*, "Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade," in *DARPA Information Survivability Conference and Exposition*, 2000.

[4] "FreeBSD Kernel Arbitrary Memory Disclosure," *http://www.osvdb.org/16091*.

[5] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory," in *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6*, San Jose, California, 1996.

[6] B. W. Lampson, "A note on the confinement problem," *Commun. ACM,* vol. 16, pp. 613-615, 1973.

[7] M. H. Lipasti*, et al.*, "Value locality and load value prediction," in *Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, Cambridge, Massachusetts, United States, 1996, pp. 138-147.

[8] Y. Sazeides and J. E. Smith, "The predictability of data values," in *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, Research Triangle Park, North Carolina, United States, 1997, pp. 248-258.

[9] P. Raghavan*, et al.*, "Dynamic schemes for speculative execution of code," *Perform. Eval.,* vol. 53, pp. 125-142, 2003.

[10] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *Proceedings of the 34th annual international symposium on Computer architecture*, San Diego, California, USA, 2007, pp. 494-505.

[11] Z. Wang and R. B. Lee, "A novel cache architecture with enhanced performance and security," in *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 83-93.

[12] Z. Wang and R. B. Lee, "Covert and Side Channels Due to Processor Architecture," in *Proceedings of the 22nd Annual Computer Security Applications Conference*, Miami Beach, Florida, USA 2006, pp. 473-482.

[13] Z. Wang and R. B. Lee, "Capacity Estimation of Non-Synchronous Covert Channels," in *Proceedings of the Second International Workshop on Security in*

Distributed Computing Systems (SDCS) (ICDCSW'05) - Volume 02, 2005, pp. 170-176.

[14] Z. Wang and R. B. Lee, "New constructive approach to covert channel modeling and channel capacity estimation," in *Proceedings of the 8th international conference on Information Security*, Singapore, 2005, pp. 498-505.

[15] R. A. Kemmerer, "Shared resource matrix methodology: an approach to identifying storage and timing channels," *ACM Trans. Comput. Syst.,* vol. 1, pp. 256-277, 1983.

[16] J. C. Huskamp, "Covert Communication Channels in Timesharing Systems," *University of California Technical Report UCB-CS-78-02,* 1978.

[17] V. Gligor, "A guide to understanding covert channel analysis of trusted systems," *Technical Report NCSC-TG-030, National Computer Security Center,* 1993.

[18] ed: National Computer Security Center, Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28-STD 1985.

[19] C.-R. Tsai*, et al.*, "On the Identification of Covert Storage Channels in Secure Systems," *IEEE Trans. Softw. Eng.,* vol. 16, pp. 569-580, 1990.

[20] M. Gasser, *Building a secure computer system*: Van Nostrand Reinhold Co., 1988.

[21] A. S. Tanenbaum, *Modern Operating Systems*: Prentice Hall, 2001.

[22] Wikipedia. Covert Channel [Online].

[23] J. K. Millen, "20 Years of Covert Channel Modeling and Analysis," presented at the IEEE Symposium on Security and Privacy, 1999.

[24] S. B. Lipner, "A comment on the confinement problem," in *Proceedings of the fifth ACM symposium on Operating systems principles*, Austin, Texas, United States, 1975.

[25] W. M. Hu, "Reducing timing channels with fuzzy time," presented at the IEEE Computer Society Symposium on Research in Security and Privacy, 1991.

[26] J. C. Wray, "An analysis of covert timing channels," presented at the IEEE Computer Society Symposium on Research in Security and Privacy, 1991.

[27] C. Abad, "IP Checksum Covert Channels and Selected Hash Collision," UCLA Technical Report, http://gray-world.net/papers/ipccc.pdf, 2001.

[28] C. G. Girling, "Covert Channels in LAN's," *IEEE Trans. Softw. Eng.,* vol. 13, pp. 292-296, 1987.

[29] S. Cabuk*, et al.*, "IP covert timing channels: design and detection," in *Proceedings of the 11th ACM conference on Computer and communications security*, Washington DC, USA, 2004.

[30] G. Shah*, et al.*, "Keyboards and covert channels," in *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, Vancouver, B.C., Canada, 2006.

[31] Z. Wang*, et al.*, "Mutual Anonymous Communications: A New Covert Channel Based on Splitting Tree MAC," presented at the the 26th IEEE International Conference on Computer Communications (INFOCOM 2007), Minisymposium, 2007.

[32] S. Li and A. Ephremides, "A covert channel in MAC protocols based on splitting algorithms," presented at the IEEE Wireless Communications and Networking Conference (WCNC 2005), 2005.

[33] S. Li and A. Epliremides, "A network layer covert channel in ad-hoc wireless networks," presented at the the 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON 2004), 2004.
[34] N. B. Lucena*, et al.*, "Covert Channels in IPv6," in *Proceedings of Privacy Enhancing Technologies (PET)*, 2005, pp. 147-166.
[35] S. Zander*, et al.*, "A Survey of Covert Channels and Countermeasures in Computer Network Protocols," *IEEE Communications Surveys & Tutorials,* vol. 9, pp. 44-57, 2007.
[36] P. A. Karger and J. C. Wray, "Storage Channels in Disk Arm Optimization," presented at the IEEE Symposium on Security and Privacy, 1991.
[37] D. E. Denning, *Cryptography and Data Security*. Massachusetts: Addison-Wesley, 1983.
[38] G. R. Andrews and R. P. Reitman, "An Axiomatic Approach to Information Flow in Programs," *ACM Trans. Program. Lang. Syst.,* vol. 2, pp. 56-76, 1980.
[39] R. Feiertag, "A Technique for Proving Specifications are Multilevel Secure," Technical Report CSL-109, Computer Science Laboratory, SRI International, Menlo Park, California, 1980.
[40] J. Rushby, "The Security Model of Enhanced HDM," in *Proceedings of the 7th DOD/NBS Computer Security Conference*, pp.120-136, September 1984.
[41] S. T. Eckmann, "Ina FIo: The FDM Flow Tool," in *Proceedings of the 10th National Computer Security Conference*, Baltimore, Maryland, pp.175-182, September 1987.
[42] J. McHugh, "An Information Flow Tool for Gypsy," in *Proceedings of the 17th Annual Computer Security Applications Conference*, pp.191, 2001.
[43] J. McHugh and D. I. Good, "An Information Flow Tool for Gypsy," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp.46-48, April 1985.
[44] J. McHugh and R. L. Akers, "A Formal Justification for the Gypsy Information Flow Tool," Technical Report, Computational Logic Inc., Austin, Texas, 1987.
[45] C.-R. Tsai*, et al.*, "A Formal Method for the Identification of Covert Storage Channels in Source Code," in *Proceedings of the lEEE Symposium on Security and Privacy*, Oakland, California, pp. 74-86, April 1987.
[46] R. A. Kemmerer and P. A. Porras, "Covert Flow Trees: A Visual Approach to Analyzing Covert Storage Channels," *IEEE Trans. Softw. Eng.,* vol. 17, pp. 1166-1185, 1991.
[47] J. Goguen and J. Meseguer, "Security policies and security models," presented at the IEEE Symposium on Security and Privacy, 1982.
[48] J. McHugh, "Covert Channel Analysis: A Chapter of the Handbook for the Computer Security Certification of Trusted Systems," Technical Memorundum 5540:080A, Naval Research Laboratory, Washington D.C., 1995.
[49] J. McLean, "Security Models," in *Encyclopedia of Software Engineering*, ed: John Wiley & Sons, 1994.
[50] M. A. Bishop, *Computer Security: Art and Science*: Addison-Wesley 2002.

[51] J. A. Goguen and J. Meseguer, "Unwinding and inference control," presented at the Proc. of the IEEE Symposium on Research in Security and Privacy, pp.75-86, 1984.
[52] J. T. Haigh*, et al.*, "An Experience Using Two Covert Channel Analysis Techniques on a Real System Design," *IEEE Trans. Softw. Eng.,* vol. 13, pp. 157-168, 1987.
[53] J. K. Millen, "Covert Channel Capacity," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp.60-66, 1987.
[54] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*: University of Illinois Press, Urbana, Illinois, 1963.
[55] J. K. Millen, "Finite-state noiseless covert channels," in *Proceedings of the Computer Security Foundation Workshop II*, pp.81-86, 1989.
[56] I. S. Moskowitz and A. R. Miller, "The channel capacity of a certain noisy timing channel," *IEEE Transactions on Information Theory,* vol. 38, pp. 1339-1344, 1992.
[57] I. S. Moskowitz and A. R. Miller, "Simple Timing Channels," in *Proceedings of the 1994 IEEE Symposium on Security and Privacy*, 1994.
[58] I. S. Moskowitz*, et al.*, "An Analysis of the Timed Z-channel," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp.2-11, 1996.
[59] M. H. Kang and I. S. Moskowitz, "A pump for rapid, reliable, secure communication," in *Proceedings of the 1st ACM conference on Computer and communications security*, pp.119-129, 1993.
[60] M. H. Kang*, et al.*, "A Network Pump," *IEEE Trans. Softw. Eng.,* vol. 22, pp. 329-338, 1996.
[61] J. W. Gray, "On Introducing Noise into the Bus-Contention Channel," in *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, 1993.
[62] J. Giles and B. Hajek, "An information-theoretic and game-theoretic study of timing channels," *IEEE Transactions on Information Theory,* vol. 48, pp. 2455-2477, 2002.
[63] B. R. Venkatraman and R. E. Newman-Wolfe, "Capacity estimation and auditability of network covert channels," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp.186-198, 1995.
[64] C.-R. Tsai and V. D. Gligor, "A Bandwidth Computation Model for Covert Storage Channels and Its Applications," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp.108-121, April 1988.
[65] S.-P. W. Shieh and V. D. Gligor, "Auditing the Use of Covert Storage Channels in Secure Systems," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp.285-295,1990.
[66] G. J. Popek and C. S. Kline, "Verifiable Secure Operating System Software," in *Proceedings of the National Computer Conference*, pp.145-151, 1974.
[67] J. Giles and B. Hajek, "The jamming game for packet timing channels," in *Proceedings of IEEE International Symposium on Information Theory*, pp.51, 2000.
[68] W.-M. Hu, "Lattice Scheduling and Covert Channels," in *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, 1992.

[69] M. H. Kang, *et al.*, "Design and Assurance Strategy for the NRL Pump," *IEEE Computer Magazine,* vol. 31, pp. 56-64, 1998.

[70] M. H. Kang, *et al.*, "The Pump: A Decade of Covert Fun," in *Proceedings of the 21st Annual Computer Security Applications Conference*, pp.352-360, 2005.

[71] S. P. Skorobogatov and R. J. Anderson, "Optical Fault Induction Attacks," presented at the Cryptographic Hardware and Embedded Systems - CHES 2002, 2003.

[72] P. Kocher, *et al.*, "Differential Power Analysis," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, 1999.

[73] T. S. Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software," presented at the Cryptographic Hardware and Embedded Systems - CHES 2000, 2000.

[74] J. Waddle and D. Wagner, "Towards Efficient Second-Order Power Analysis," presented at the Cryptographic Hardware and Embedded Systems - CHES 2004, 2004.

[75] M. Joye, *et al.*, "On Second-Order Differential Power Analysis," presented at the Cryptographic Hardware and Embedded Systems - CHES 2005, 2005.

[76] L. Goubin and J. Patarin, "DES and differential power analysis: The duplication method," presented at the Cryptographic Hardware and Embedded Systems - CHES 1999, 1999.

[77] S. Chari, *et al.*, "Template Attacks," presented at the Cryptographic Hardware and Embedded Systems - CHES 2002, 2002.

[78] C. Archambeau, *et al.*, "Template Attacks in Principal Subspaces," presented at the Cryptographic Hardware and Embedded Systems - CHES 2006, 2006.

[79] T. S. Messerges, *et al.*, "Investigations of power analysis attacks on smartcards," in *Proceedings of the USENIX Workshop on Smartcard Technology*, 1999, pp. 151-162.

[80] P. N. Fahn and P. K. Pearson, "IPA: A New Class of Power Attacks," presented at the Cryptographic Hardware and Embedded Systems - CHES 1999, 1999.

[81] J. Kelsey, *et al.*, "Side Channel Cryptanalysis of Product Ciphers," *Journal of Computer Security,* vol. 8, pp. 141-158, 2000.

[82] K. Schramm, *et al.*, "A New Class of Collision Attacks and Its Application to DES," presented at the 10th International Workshop on Fast Software Encryption (FSE 2003), 2003.

[83] E. Brier, *et al.*, "Correlation Power Analysis with a Leakage Model," presented at the Cryptographic Hardware and Embedded Systems - CHES 2004, 2004.

[84] F.-X. Standaert, *et al.*, "Power Analysis of an FPGA: Implementation of Rijndael: Is Pipelining a DPA Countermeasure?," presented at the Cryptographic Hardware and Embedded Systems - CHES 2004, 2004.

[85] K. Schramm, *et al.*, "A Collision-Attack on AES Combining Side Channel- and Differential-Attack," presented at the Cryptographic Hardware and Embedded Systems - CHES 2004, 2004.

[86] S. Mangard, "A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion," presented at the Information Security and Cryptology - ICISC 2002, 2003.

[87] C. Lauradoux, "Collision attacks on processors with cache and countermeasures," *Western European Workshop on Research in Cryptology - WEWoRC 2005,* pp. 76-85, 2005.

[88] G. Bertoni*, et al.*, "AES Power Attack Based on Induced Cache Miss and Countermeasure," presented at the International Symposium on Information Technology: Coding and Computing - ITCC 2005, 2005.

[89] J. Bonneau, "Robust Final-Round Cache-Trace Attacks Against AES," Cryptology ePrint Archive, Report 2006/374,2006.

[90] E. Oswald and S. Mangard, "Template Attacks on Masking - Resistance Is Futile," in *Proceedings of the RSA Conference 2007 Cryptographers' Track (CT-RSA 2007)*, 2007.

[91] S. Mangard and K. Schramm, "Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations," presented at the Cryptographic Hardware and Embedded Systems 2006 (CHES 2006), 2006.

[92] R. Novak, "SPA-Based Adaptive Chosen-Ciphertext Attack on RSA Implementation," presented at the Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems (PKC 2002), 2002.

[93] B. d. Boer*, et al.*, "A DPA Attack against the Modular Reduction within a CRT Implementation of RSA," presented at the Cryptographic Hardware and Embedded Systems - CHES 2002, 2002.

[94] P.-A. Fouque*, et al.*, "Attacking Unbalanced RSA-CRT Using SPA," presented at the Cryptographic Hardware and Embedded Systems - CHES 2003, 2003.

[95] W. Schindler, "A Combined Timing and Power Attack," presented at the Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems (PKC 2002), 2002.

[96] J.-S. Coron, "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems," presented at the Cryptographic Hardware and Embedded Systems - CHES 1999, 1999.

[97] E. Oswald, "Enhancing Simple Power-Analysis Attacks on Elliptic Curve Cryptosystems," presented at the Cryptographic Hardware and Embedded Systems - CHES 2002, 2002.

[98] K. Itoh*, et al.*, "Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA," presented at the Cryptographic Hardware and Embedded Systems - CHES 2002, 2002.

[99] L. Goubin, "A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems," presented at the Public Key Cryptography - PKC 2003, 2003.

[100] S. B. Örs*, et al.*, "Power-Analysis Attacks on an FPGA - First Experimental Results," presented at the Cryptographic Hardware and Embedded Systems - CHES 2003, 2003.

[101] K. Lemke*, et al.*, "DPA on n-Bit Sized Boolean and Arithmetic Operations and Its Application to IDEA, RC6, and the HMAC-Construction," presented at the Cryptographic Hardware and Embedded Systems - CHES 2004, 2004.

[102] S. Mangard*, et al.*, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*: Springer, 2007.

[103] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," presented at the Advances in Cryptology - CRYPTO '96, 1996.

[104] J. F. Dhem*, et al.*, "A Practical Implementation of the Timing Attack," in *Proceedings of the The International Conference on Smart Card Research and Applications*, 1998, pp. 167-182.

[105] W. Schindler, "Optimized timing attacks against public key cryptosystems," *Statistics and Decisions 20,* pp. 191-210, 2002.

[106] W. Schindler*, et al.*, "Improving Divide and Conquer Attacks against Cryptosystems by Better Error Detection / Correction Strategies," in *Proceedings of the 8th IMA International Conference on Cryptography and Coding*, 2001, pp. 245-267.

[107] W. Schindler, "A Timing Attack against RSA with the Chinese Remainder Theorem," presented at the Cryptographic Hardware and Embedded Systems - CHES 2000, 2000.

[108] D. Brumley and D. Boneh, "Remote timing attacks are practical," in *Proceedings of the 12th conference on USENIX Security Symposium*, Washington, DC, 2003, pp. 1-14.

[109] O. Aciiçmez*, et al.*, "Improving Brumley and Boneh timing attack on unprotected SSL implementations," in *Proceedings of the 12th ACM conference on Computer and Communications Security (CCS'05)*, 2005, pp. 139-146.

[110] D. Page, "Theoretical use of cache memory as a cryptanalytic side-channel," Technical Report CSTR-02-003, Department of Computer Science, University of Bristol, 2002.

[111] O. Aciiçmez and Ç. K. Koç, "Trace-Driven Cache Attacks on AES," Cryptology ePrint Archive, Report 2006/138,2006.

[112] Y. Tsunoo*, et al.*, "Cryptanalysis of block ciphers imple-mented on computers with cache," presented at the Proc. International Symposium on Information Theory and its Applications, 2002.

[113] Y. Tsunoo*, et al.*, "Cryptanalysis of DES Implemented on Computers with Cache," presented at the Cryptographic Hardware and Embedded Systems - CHES 2003, 2003.

[114] D. J. Bernstein, "Cache-timing Attacks on AES," available at: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf,2005.

[115] M. Neve*, et al.*, "A refined look at Bernstein's AES side-channel analysis," in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, 2006.

[116] J. Bonneau and I. Mironov, "Cache-Collision Timing Attacks Against AES," presented at the Cryptographic Hardware and Embedded Systems - CHES 2006, 2006.

[117] O. Aciiçmez*, et al.*, "Cache Based Remote Timing Attack on the AES," presented at the Topics in Cryptology -- CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, 2007.

[118] D. A. Osvik*, et al.*, "Cache Attacks and Countermeasures: the Case of AES ", Cryptology ePrint Archive, Report 2005/271,2005.

[119] D. A. Osvik, *et al.*, "Cache Attacks and Countermeasures: The Case of AES," presented at the Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, 2006.
[120] M. Neve and J.-P. Seifert, "Advances on Access-Driven Cache Attacks on AES " presented at the Selected Areas in Cryptography - SAC'06, 2006.
[121] C. Percival, "Cache Missing for Fun and Profit," available at: http://www.daemonology.net/papers/htt.pdf,2005.
[122] O. Aciiçmez, *et al.*, "On the power of simple branch prediction analysis," in *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, 2007, pp. 312-320.
[123] O. Aciiçmez, *et al.*, "Predicting Secret Keys via Branch Prediction," presented at the Topics in Cryptology -- CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, 2007.
[124] O. Acıiçmez, *et al.*, "New Branch Prediction Vulnerabilities in OpenSSL and Necessary Software Countermeasures," in *Proceedings of the 14th IMA International Conference on Cryptography and Coding*, 2007, pp. 185-203.
[125] NSA, NSA TEMPEST Documents, http://www.cryptome.org/nsa-tempest.htm,2003.
[126] D. Agrawal, *et al.*, "The EM Side-Channel(s): Attacks and Assessment Methodologies," presented at the Cryptographic Hardware and Embedded Systems (CHES 2002), 2002.
[127] J.-J. Quisquater and D. Samyde, "A new tool for non-intrusive analysis of smart cards based on electro-magnetic emissions: the SEMA and DEMA methods," presented at the Eurocrypt rump session, 2000.
[128] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (EMA): measures and countermeasures for smart cards," presented at the Smart Cards Programming and Security (e-Smart 2001), LNCS vol. 2140, 2001.
[129] K. Gandolfi, *et al.*, "Electromagnetic analysis: Concrete results," presented at the Cryptographic Hardware and Embedded Systems - CHES 2001, LNCS vol. 2162, 2001.
[130] S. Mangard, "Exploiting Radiated Emissions – EM Attacks on Cryptographic ICs," in *Proceedings of Austrochip*, Linz, Austria, 2003.
[131] V. Carlier, *et al.*, "Electromagnetic Side Channels of an FPGA Implementation of AES," Cryptology ePrint Archive, Report 2004/145,2004.
[132] D. Agrawal, *et al.*, "Multi-channel Attacks," presented at the Cryptographic Hardware and Embedded Systems - CHES 2003, 2003.
[133] J.-J. Quisquater and D. Samyde, "Automatic code recognition for smart cards using a Kohonen neural network," in *Proceedings of the 5th conference on Smart Card Research and Advanced Application Conference*, San Jose, CA, 2002.
[134] M. G. Kuhn, "Compromising Emanations: Eavesdropping Risks of Computer Displays," University of Cambridge, Computer Laboratory, UCAM-CL-TR-577, 2003.
[135] M. G. Kuhn, "Electromagnetic Eavesdropping Risks of Flat-Panel Displays," *Privacy Enhancing Technologies*, pp. 88-107, 2005.
[136] D. Asonov and R. Agrawal, "Keyboard Acoustic Emanations," in *Proceedings of 25th IEEE Symposium on Security and Privacy*, 2004, pp. 3-11.

[137] A. Shamir and E. Tromer, "Acoustic cryptanalysis - On nosy people and noisy machines," http://people.csail.mit.edu/tromer/acoustic/, 2004.
[138] R. Anderson, *Security engineering: A Guide to Building Dependable Distributed Systems*. New York: Wiley & Sons, 2001.
[139] R. Anderson and M. Kuhn, "Tamper resistance: a cautionary note," in *Proceedings of the Second USENIX Workshop on Electronic Commerce*, 1996, pp. 1-11.
[140] R. Anderson and M. Kuhn, "Low Cost Attacks on Tamper Resistant Devices," in *Proceedings of the 5th International Workshop on Security Protocols*, 1998, pp. 125-136.
[141] P. Gutmann, "Data remanence in semiconductor devices," in *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, Washington, D.C., 2001.
[142] D. P. Maher, "Fault Induction Attacks, Tamper Resistance, and Hostile Reverse Engineering in Perspective," in *Proceedings of the First International Conference on Financial Cryptography*, 1997, pp. 109-122.
[143] J.-J. Quisquater and D. Samyde, "Eddy current for magnetic analysis with active sensor," *Proceedings of Esmart,* 2002.
[144] S. P. Skorobogatov and R. Anderson, "Optical Fault Induction Attacks," presented at the Cryptographic Hardware and Embedded Systems - CHES 2003, 2003.
[145] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Journal of Cryptology,* vol. 4, pp. 3-72, 1991.
[146] L. Hemme, "A Differential Fault Attack Against Early Rounds of (Triple-)DES " *Cryptographic Hardware and Embedded Systems - CHES 2004,* vol. 3156 (LNCS), pp. 254-267, 2004.
[147] C. Giraud and H. Thiebeauld, "A Survey on Fault Attacks," presented at the Smart Card Research and Advanced Applications VI - 18th IFIP World Computer Congress, 2004.
[148] J. Blömer and J.-P. Seifert, "Fault Based Cryptanalysis of the Advanced Encryption Standard (AES) " presented at the Financial Cryptography, 2003.
[149] C.-N. Chen and S.-M. Yen, "Differential Fault Analysis on AES Key Schedule and Some Countermeasures," presented at the Information Security and Privacy, 8th Australasian Conference, ACISP 2003, 2003.
[150] P. Dusart*, et al.*, "Differential Fault Analysis on A.E.S," presented at the Applied Cryptography and Network Security (ACNS 2003), 2003.
[151] C. Giraud, "DFA on AES," IACR eprint archive, Report 2003/008, 2003.
[152] G. Piret and J.-J. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD," presented at the Cryptographic Hardware and Embedded Systems (CHES 2003), 2003.
[153] J. J. Hoch and A. Shamir, "Fault Analysis of Stream Ciphers," presented at the Cryptographic Hardware and Embedded Systems - CHES 2004, 2004.
[154] E. Biham*, et al.*, "Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4," presented at the Fast Software Encryption (FSE 2005), 2005.
[155] D. Boneh*, et al.*, "On the Importance of Checking Cryptographic Protocols for Faults," presented at the Advances in Cryptology - EUROCRYPT'97, 1997.

[156] D. Boneh*, et al.*, "On the Importance of Eliminating Errors in Cryptographic Computations " *Journal of Cryptology,* vol. 14, pp. 101-119, 2001.
[157] M. Joye*, et al.*, "Chinese Remaindering Based Cryptosystems in the Presence of Faults " *Journal of Cryptology,* vol. 12, pp. 241-245, 1999.
[158] J. Blömer*, et al.*, "Sign Change Fault Attacks On Elliptic Curve Cryptosystems," presented at the Fault Diagnosis and Tolerance in Cryptography, 2006.
[159] I. Biehl*, et al.*, "Differential Fault Attacks on Elliptic Curve Cryptosystems," in *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO 2000)*, 2000, pp. 131-146.
[160] F. Bao*, et al.*, "Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults," in *Proceedings of the 5th International Workshop on Security Protocols*, 1998, pp. 115-124.
[161] M. Joye*, et al.*, "RSA-type Signatures in the Presence of Transient Faults," *Proceedings of the 6th IMA International Conference on Cryptography and Coding,* vol. 1355 (LNCS), pp. 155-160, 1997.
[162] O. Goldreich, "Towards a theory of software protection and simulation by oblivious RAMs," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, 1987, pp. 182-194.
[163] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *Journal of the ACM,* vol. 43, pp. 431-473, 1996.
[164] M. G. Kuhn, "Cipher Instruction Search Attack on the Bus-Encryption Security Microcontroller DS5002FP," *IEEE Transactions on Computers,* vol. 47, pp. 1153-1157, 1998.
[165] R. M. Best, "Preventing Software Piracy with Crypto-Microprocessors," *Proc. IEEE Spring COMPCON'80,* pp. 466-469, 1980.
[166] X. Zhuang*, et al.*, "HIDE: an infrastructure for efficiently protecting information leakage on the address bus," in *Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, Boston, MA, USA, 2004.
[167] J.-S. Coron*, et al.*, "Statistics and secret leakage," *ACM Transactions on Embedded Computing Systems,* vol. 3, pp. 492-508, 2004.
[168] M.-L. Akkar*, et al.*, "Power Analysis, What Is Now Possible..." presented at the Advances in Cryptology - ASIACRYPT 2000, 2000.
[169] M. Joye and S.-M. Yen, "The Montgomery Powering Ladder," presented at the Cryptographic Hardware and Embedded Systems - CHES 2002, 2003.
[170] J. Bos and M. Coster, "Addition chain heuristics," presented at the Advances in cryptology - Crypto 1989, 1989.
[171] D. E. Knuth, *The Art of Computer Programming*, 2nd ed. vol. 2: Semi Numerical Algorithms: Addison Wesley, 1981.
[172] Ç. K. Koç, "Analysis of Sliding Window Techniques for Exponentiation," *Computers and Mathematics with Applications,* vol. 30, pp. 17-24, 1995.
[173] C. D. Walter, "Exponentiation Using Division Chains," *IEEE Transactions on Computers,* vol. 47, pp. 757-765, 1998.
[174] C. D. Walter, "MIST : An Efficient, Randomized Exponentiation Algorithm for Resisting Power Analysis " presented at the Topics in Cryptology -- CT-RSA 2002, 2002.

[175] M. A. Hasan, "Power Analysis Attacks and Algorithmic Approaches to Their Countermeasures for Koblitz Curve Cryptosystems," *IEEE Transactions on Computers,* vol. 50, pp. 1071-1083, 2001.

[176] M. Joye and C. Tymen, "Protections against Differential Analysis for Elliptic Curve Cryptography," presented at the Cryptographic Hardware and Embedded Systems - CHES 2001, 2001.

[177] P.-Y. Liardet and N. P. Smart, "Preventing SPA/DPA in ECC systems using the Jacobi form," presented at the Cryptographic Hardware and Embedded Systems - CHES 2001, 2001.

[178] E. Oswald and M. Aigner, "Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks," *Cryptographic Hardware and Embedded Systems - CHES 2001,* vol. 2162 (LNCS), pp. 39-50, 2001.

[179] J.-C. Ha and S.-J. Moon, "Randomized Signed-Scalar Multiplication of ECC to Resist Power Attacks," presented at the Cryptographic Hardware and Embedded Systems - CHES 2002, 2002.

[180] T. Izu and T. Takagi, "A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks," *Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems (PKC),* vol. 2274 (LNCS), pp. 280-296, 2002.

[181] J.-S. Coron, "Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems " presented at the Cryptographic Hardware and Embedded Systems - CHES 1999, 1999.

[182] T. S. Messerges*, et al.*, "Power Analysis Attacks of Modular Exponentiation in Smartcards," presented at the Cryptographic Hardware and Embedded Systems - CHES 1999, 1999.

[183] K. Itoh*, et al.*, "DPA Countermeasures by Improving the Window Method," *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems,* vol. 2523 (LNCS), pp. 303-317, 2003.

[184] M. Ahn*, et al.*, "A Random M-ary Method Based Countermeasure against Side Channel Attacks," presented at the Computational Science and Its Applications - ICCSA 2003, 2003.

[185] C. D. Walter, "Sliding Windows Succumbs to Big Mac Attack," presented at the Cryptographic Hardware and Embedded Systems - CHES 2001, 2001.

[186] S. Chari*, et al.*, "Towards Sound Approaches to Counteract Power-Analysis Attacks," presented at the Advances in Cryptology - Crypto 1999, 1999.

[187] D. Chaum, "Blind signatures for untraceable payments," presented at the Advances in Cryptology -- CRYPTO '82, 1983.

[188] D. Chaum, "Showing credentials without identification: transferring signatures between unconditionally unlinkable pseudonyms," presented at the Advances in Cryptology - AUSCRYPT '90, 1990.

[189] M.-L. Akkar and C. Giraud, "An Implementation of DES and AES, Secure against Some Attacks," presented at the Cryptographic Hardware and Embedded Systems - CHES 2001, 2001.

[190] J.-S. Coron and L. Goubin, "On Boolean and Arithmetic Masking against Differential Power Analysis," presented at the Cryptographic Hardware and Embedded Systems - CHES 2000, 2000.

[191] L. Goubin, "A Sound Method for Switching between Boolean and Arithmetic Masking," presented at the Cryptographic Hardware and Embedded Systems - CHES 2001, 2001.

[192] T. S. Messerges, "Securing the AES Finalists Against Power Analysis Attacks," in *Proceedings of the 7th International Workshop on Fast Software Encryption (FSE '00)*, 2000, pp. 150-164.

[193] J. D. Golic and C. Tymen, "Multiplicative Masking and Power Analysis of AES," presented at the Cryptographic Hardware and Embedded Systems - CHES 2002, 2003.

[194] E. Trichina*, et al.*, "Simplified Adaptive Multiplicative Masking for AES," presented at the Cryptographic Hardware and Embedded Systems - CHES 2002, 2003.

[195] D. May*, et al.*, "Random Register Renaming to Foil DPA," presented at the Cryptographic Hardware and Embedded Systems - CHES 2001, 2001.

[196] J. A. Ambrose*, et al.*, "RIJID: random code injection to mask power analysis based side channel attacks," in *Proceedings of the 44th annual conference on Design automation*, 2007, pp. 489-492.

[197] Y. Ishai*, et al.*, "Private Circuits: Securing Hardware against Probing Attacks " presented at the Advances in Cryptology - CRYPTO 2003, 2003.

[198] T. S. Messerges*, et al.*, *Method and Apparatus for Preventing Information Leakage Attacks on a Microelectronic Assembly*: US Patent 6,295,606, 2001.

[199] E. Trichina, "Combinational logic design for AES subbyte transformation on masked data," Cryptology ePrint Archive, Report 2003/236, 2003.

[200] E. Trichina and T. Korkishko, "Small Size, Low Power, Side Channel-Immune AES Coprocessor: Design and Synthesis Results," in *Proceedings of the Fourth Conference on the Advanced Encryption Standard (AES)*, 2004.

[201] S. Mangard*, et al.*, "Side-Channel Leakage of Masked CMOS Gates," presented at the Topics in Cryptology - CT-RSA 2005, 2005.

[202] W. Fischer and B. M. Gammel, "Masking at Gate Level in the Presence of Glitches," presented at the Cryptographic Hardware and Embedded Systems - CHES 2005, 2005.

[203] S. Moore*, et al.*, "Improving Smart Card Security using Self-timed Circuits," in *Proceedings of the 8th IEEE International Symposium on Asynchronous Circuits and Systems - ASYNC '02*, 2002, pp. 23-58.

[204] J. J. A. Fournier*, et al.*, "Security Evaluation of Asynchronous Circuits," presented at the Cryptographic Hardware and Embedded Systems - CHES 2003, 2003.

[205] M. Bucci*, et al.*, "Three-Phase Dual-Rail Pre-charge Logic " presented at the Cryptographic Hardware and Embedded Systems - CHES 2006, 2006.

[206] T. Popp and S. Mangard, "Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints," presented at the Cryptographic Hardware and Embedded Systems - CHES 2005, 2005.

[207] D. Suzuki and M. Saeki, "Security Evaluation of DPA Countermeasures Using Dual-Rail Pre-charge Logic Style," presented at the Cryptographic Hardware and Embedded Systems - CHES 2006, 2006.

[208] K. Tiri*, et al.*, "A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards," presented at the Proc. 28th European Solid-State Circuits Conf. (ESSCIRC 02), 2002.
[209] Z. Toprak and Y. Leblebici, "Low-power current mode logic for improved DPA-resistance in embedded systems," presented at the IEEE International Symposium on Circuits and Systems (ISCAS 2005), 2005.
[210] F. Macé*, et al.*, "A Design Methodology for Secured ICs Using Dynamic Current Mode Logic," presented at the Integrated Circuit and System Design, 2005.
[211] M. W. Allam and M. I. Elmasry, "Dynamic current mode logic (DyCML): a new low-power high-performance logic style," *Solid-State Circuits, IEEE Journal of,* vol. 36, pp. 550-558, 2001.
[212] J. Daemen and V. Rijmen, "Resistance against implementation attacks: A comparative study of the AES proposals," in *Proceedings of the Second Advanced Encryption Standard (AES) Candidate Conference*, 1999.
[213] G. B. Ratanpal*, et al.*, "An On-Chip Signal Suppression Countermeasure to Power Analysis Attacks," *IEEE Transactions on Dependable and Secure Computing,* vol. 1, pp. 179-189, 2004.
[214] A. Shamir, "Protecting Smart Cards from Passive Power Analysis with Detached Power Supplies," presented at the Cryptographic Hardware and Embedded Systems - CHES 2000, 2000.
[215] J. F. Dhem, "Design of an effcient public-key cryptographic library for risc-based smart cards," Ph.D. Thesis, Université catholique de Louvain (UCL), Crypto Group - Laboratoire de microélectronique (DICE),1998.
[216] C. D. Walter, "Montgomery Exponentiation Needs no Final Subtractions," *Electronics Letters,* vol. 35, pp. 1831-1832, 1999.
[217] C. D. Walter, "Montgomery's Multiplication Technique: How to Make It Smaller and Faster " presented at the Cryptographic Hardware and Embedded Systems - CHES 1999, 1999.
[218] G. Hachez and J.-J. Quisquater, "Montgomery Exponentiation with no Final Subtractions: Improved Results," presented at the Cryptographic Hardware and Embedded Systems - CHES 2000, 2000.
[219] E. Biham, "A Fast New DES Implementation in Software," in *Proceedings of the 4th International Workshop on Fast Software Encryption (FSE)*, 1997, pp. 260-272.
[220] E. Brickell*, et al.*, "Software mitigations to hedge AES against cache-based software side channel vulnerabilities," IACR ePrint Archive, Report 2006/052,2006.
[221] E. Brickell*, et al.*, "Mitigating cache/timing attacks in AES and RSA software implementations," presented at the RSA Conference 2006, session DEV-203, http://2006.rsaconference.com/us/cd_pdfs/DEV-203.pdf, 2006.
[222] D. Page, "Partitioned Cache Architecture as a Side-Channel Defense Mechanism " Cryptology ePrint Archive, Report 2005/280,2005.
[223] B. Pfitzmann, "Information Hiding Terminology - Results of an Informal Plenary Meeting and Additional Proposals," in *Proceedings of the First International Workshop on Information Hiding*, 1996, pp. 347-350.

[224] F. A. P. Petitcolas*, et al.*, "Information Hiding - A Survey," *Proceedings of the IEEE, special issue on protection of multimedia content,* vol. 87, pp. 1062-1078, 1999.

[225] N. F. Johnson*, et al.*, *Information Hiding: Steganography and Watermarking - Attacks and Countermeasures*: Springer, 2000.

[226] "Encyclopedia of Cryptography and Security," Tilborg and H. C. A. van, Eds., ed: Springer, 2005, p. 159.

[227] M. Miller and J. B. M. Miller, *Digital Watermarking: Principles and Practice*: Morgan Kaufmann, 2001.

[228] J. Dittmann*, et al.*, "Media-independent watermarking classification and the need for combining digital video and audio watermarking for media authentication," presented at the International Conference on Information Technology: Coding and Computing, 2000.

[229] N. Cvejic and T. Seppanen, *Digital Audio Watermarking Techniques and Technologies: Applications and Benchmarks* IGI Global, 2007.

[230] R. G. van Schyndel*, et al.*, "A digital watermark," in *IEEE International Conference on Image Processing - ICIP '94*, 1994, pp. 86-90.

[231] I. J. Cox*, et al.*, "A Secure, Robust Watermark for Multimedia," in *Proceedings of the First International Workshop on Information Hiding*, 1996, pp. 185-206.

[232] M. Barni*, et al.*, "A DCT-domain system for robust image watermarking," *Signal Process.,* vol. 66, pp. 357-372, 1998.

[233] W.-N. Lie*, et al.*, "Robust image watermarking on the DCT domain," presented at the IEEE International Symposium on Circuits and Systems - ISCAS 2000, 2000.

[234] C. I. Podilchuk and W. Zeng, "Digital image watermarking using visual models," in *Proceedings of the SPIE Conference on Human Vision and Electronic Imaging II*, 1997, pp. 100-111.

[235] D. Kundur and D. Hatzinakos, "A Robust Digital Image Watermarking Scheme Using the Wavelet-Based Fusion," in *Proceedings of the International Conference on Image Processing (ICIP '97)*, 1997, pp. 544-547.

[236] A. A. Reddy and B. N. Chatterji, "A new wavelet based logo-watermarking scheme," *Pattern Recogn. Lett.,* vol. 26, pp. 1019-1027, 2005.

[237] L. Boney*, et al.*, "Digital Watermarks for Audio Signals," in *Proceedings of the 1996 International Conference on Multimedia Computing and Systems (ICMCS '96)*, 1996, pp. 473-480.

[238] M. Ramkumar*, et al.*, "A robust data hiding scheme for images using DFT," presented at the International Conference on Image Processing - ICIP '99, 1999.

[239] J. J. K. O. Ruanaidh*, et al.*, "Phase watermarking of digital images," in *International Conference on Image Processing - ICIP '96*, 1996, pp. 239-242.

[240] I. J. Cox*, et al.*, "Secure Spread Spectrum Watermarking for Multimedia," *IEEE Transactions on Image Processing,* vol. 6, pp. 1673-1687, 1997.

[241] J. R. Smith and B. O. Comiskey, "Modulation and Information Hiding in Images," in *Proceedings of the First International Workshop on Information Hiding*, 1996, pp. 207-226.

[242] W. Bender*, et al.*, "Techniques for Data Hiding," *IBM Systems Journal,* vol. 35, pp. 313-336, 1996.

[243] J. Fridrich, "Robust Bit Extraction from Images," presented at the IEEE International Conference on Multimedia Computing and Systems (ICMCS '99), 1999.

[244] R. B. Wolfgang and E. J. Delp, "A Watermark For Digital Images," presented at the IEEE International Conference on Image Processing (ICIP'96), 1996.

[245] J. T. Brassil, *et al.*, "Copyright protection for the electronic distribution of text documents," *Proceedings of the IEEE,* vol. 87, pp. 1181-1196, 1999.

[246] P. Wayner, "Mimic functions," *Cryptologia,* vol. XVI(3), pp. 285-299, 1992.

[247] US-CERT, http://www.us-cert.gov/.

[248] SecureFocus, http://www.securityfocus.com/.

[249] INSECURE.ORG, http://seclists.org/.

[250] The Mozilla Organization. "Javascript lambda replace exposes memory contents". http://www.mozilla.org/security/announce/mfsa2005-33.html, 2005.

[251] S. Byers, "Information leakage caused by hidden data in published documents," *Security & Privacy, IEEE,* vol. 2, pp. 23-27, 2004.

[252] P. Gutmann, "Lessons Learned in Implementing and Deploying Crypto Software," in *Proceedings of the 11th USENIX Security Symposium*, 2002.

[253] D. Engler*, et al.*, "Bugs as deviant behavior: a general approach to inferring errors in systems code," *SIGOPS Oper. Syst. Rev.,* vol. 35, pp. 57-72, 2001.

[254] "Wu-ftpd core dump vulnerability," http://www.insecure.org/sploits/ftp.coredump2.html.

[255] "Solaris (and others) ftpd core dump bug," http://www.insecure.org/sploits/ftpd.pasv.html.

[256] "Coredump hole in imapd and ipop3d in slackware 3.4," http://www.insecure.org/sploits/slackware.ipop.imap.core.html.

[257] "Security Dynamics FTP server core problem," http://www.insecure.org/sploits/solaris.secdynamics.core.html.

[258] J. Chow*, et al.*, "Understanding data lifetime via whole system simulation," in *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, San Diego, CA, 2004.

[259] Arkoon Security Team. "Information leak in the Linuxkernel ext2 implementation". http://arkoon.net/advisories/ext2-make-empty-leak.txt, March 2005.

[260] N. Provos, "Encrypting virtual memory," in *Proceedings of the 9th conference on USENIX Security Symposium - Volume 9*, Denver, Colorado, 2000.

[261] J. A. Halderman*, et al.*, "Lest We Remember: Cold Boot Attacks on Encryption Keys," Center For Information Technology Policy, Princeton University,February 2008.

[262] D. Lie*, et al.*, "Architectural support for copy and tamper resistant software," *SIGPLAN Not.,* vol. 35, pp. 168-177, 2000.

[263] G. E. Suh*, et al.*, "AEGIS: architecture for tamper-evident and tamper-resistant processing," in *Proceedings of the 17th annual international conference on Supercomputing*, San Francisco, CA, USA, 2003.

[264] R. B. Lee*, et al.*, "Architecture for Protecting Critical Secrets in Microprocessors," in *Proceedings of the 32nd annual international symposium on Computer Architecture*, 2005.

[265] J. S. Dwoskin and R. B. Lee, "Hardware-rooted trust for secure key management and transient trust," in *Proceedings of the 14th ACM conference on Computer and communications security*, Alexandria, Virginia, USA, 2007, pp. 389-400.
[266] D. Champagne and R. B. Lee, "Scalable architectural support for trusted software," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, 2010, pp. 1-12.
[267] D. Champagne*, et al.*, "The Reduced Address Space (RAS) for Application Memory Authentication," in *Proceedings of the 11th international conference on Information Security*, Taipei, Taiwan, 2008, pp. 47-63.
[268] M. Matsui, "Linear cryptanalysis method for DES cipher," in *Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, Lofthus, Norway, 1994, pp. 386-397.
[269] P. Ranganathan*, et al.*, "Reconfigurable caches and their application to media processing," in *Proceedings of the 27th annual international symposium on Computer architecture*, Vancouver, British Columbia, Canada, 2000, pp. 214-224.
[270] P. Shivakumar and N. Jouppi, "Cacti 3.0: An integrated cache timing, power, and area model," *Technical report, COMPAQ Western Research Lab,* 2001.
[271] T. M. Cover and J. A. Thomas, *Elements of information theory*: Wiley-Interscience, 1991.
[272] J. Kong*, et al.*, "Deconstructing new cache designs for thwarting software cache-based side channel attacks," in *Proceedings of the 2nd ACM workshop on Computer security architectures*, Alexandria, Virginia, USA, 2008, pp. 25-34.
[273] J. Kong*, et al.*, "Hardware-software integrated approaches to defend against software cache-based side channel attacks," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, 2009, pp. 393-404.
[274] M-Sim v2.0, http://www.cs.binghamton.edu/~jsharke/m-sim/.
[275] M. Zhang and K. Asanovic, "Highly-Associative Caches for Low-Power Processors " presented at the Kool Chips Workshop, MICRO-33, Monterey, CA, December 2000.
[276] C. Zhang, "Balanced Cache: Reducing Conflict Misses of Direct-Mapped Caches," in *Proceedings of the 33rd annual international symposium on Computer Architecture*, 2006, pp. 155-166.
[277] H. Al-Zoubi*, et al.*, "Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite," in *Proceedings of the 42nd annual Southeast regional conference*, Huntsville, Alabama, 2004, pp. 267-272.
[278] V. Fischer*, et al.*, "True Random Number Generator Embedded in Reconfigurable Hardware," in *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, 2003, pp. 415-430.
[279] S. Thoziyoor*, et al.*, "CACTI 5.0," PHL Techincal Report HPL-2007-167.
[280] Predictive Technology Model. http://www.eas.asu.edu/~ptm.
[281] M. D. Hill and A. J. Smith, "Evaluating Associativity in CPU Caches," *IEEE Trans. Comput.,* vol. 38, pp. 1612-1630, 1989.
[282] http://www.simplescalar.com.

[283] M. D. Powell, *et al.*, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," in *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, Austin, Texas, 2001, pp. 54-65.

[284] K. Inoue, *et al.*, "Way-predicting set-associative cache for high performance and low energy consumption," in *Proceedings of the 1999 international symposium on Low power electronics and design*, San Diego, California, United States, 1999, pp. 273-275.

[285] P. P. Shirvani and E. J. McCluskey, "PADded Cache: A New Fault-Tolerance Technique for Cache Memories," in *Proceedings of the 1999 17TH IEEE VLSI Test Symposium*, 1999, p. 440.

[286] M. Mutyam and V. Narayanan, "Working with process variation aware caches," in *Proceedings of the conference on Design, automation and test in Europe*, Nice, France, 2007, pp. 1152-1157.

[287] H. Lee, *et al.*, "Performance of Graceful Degradation for Cache Faults," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, 2007, pp. 409-415.

[288] D. M. Tullsen, *et al.*, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 1995, pp. 392-403.

[289] D. T. Marr, *et al.*, "Hyper-Threading Technology Architecture and Microarchitecture," *Intel Technology Journal,* vol. 6, pp. 4-15, 2002.

[290] H. M. Mathis, *et al.*, "Characterization of simultaneous multithreading (SMT) efficiency in POWER5," *IBM Journal of Research and Development,* vol. 49, pp. 555-564, 2005.

[291] "Intel Itanium Architecture Software Developer's Manuals," vol. 1-3, available at http://www.intel.com/design/itanium2/documentation.htm.

[292] R. Zahir, *et al.*, "OS and compiler considerations in the design of the IA-64 architecture," *SIGARCH Comput. Archit. News,* vol. 28, pp. 212-221, 2000.

[293] A. Raman, *et al.*, "Speculative parallelization using software multi-threaded transactions," in *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, Pittsburgh, Pennsylvania, USA, 2010, pp. 65-76.

[294] H. Kim, *et al.*, "Scalable Speculative Parallelization on Commodity Clusters," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 3-14.

[295] M. C. Davey and D. J. C. Mackay, "Reliable communication over channels with insertions, deletions, and substitutions," *Information Theory, IEEE Transactions on,* vol. 47, pp. 687-698, 2001.

[296] R. L. Doburshin, "Shannon's Theorems for Channels with Synchronization Errors," *Problemy Peredachi Informatsii,* vol. 3, pp. 18-36, 1967.

[297] N. D. Vvedenskaya and R. L. Doburshin, "The Computation on a Computer of The Channel Capacity of a Line with Symbol Drop-out," *Problemy Peredachi Informatsii,* vol. 4, pp. 92-95, 1968.

[298] A. S. Dolgopolov, "Capacity Bounds for a Channel with Synchronization Errors," *Problemy Peredachi Informatsii,* vol. 26, pp. 27-37, 1990.

[299] K. S. Zigangirov, "Sequential Decoding for A Binary Channel with Drop Outs and Insertions," *Problemy Peredachi Informatsii,* vol. 5, pp. 22-30, 1969.
[300] D. Leigh, "Capacity of Insertion and Deletion Channels," *Project Report, available at* *http://www.inference.phy.cam.ac.uk/is/papers/*, 2001.
[301] J. Luo and A. Ephremides, "On the throughput, capacity, and stability regions of random multiple access," *IEEE/ACM Trans. Netw.,* vol. 14, pp. 2593-2607, 2006.