ON THE PREVENTION OF CROSS-VM CACHE-BASED SIDE CHANNEL ATTACKS

ZAKIRA INAYAT

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

ON THE PREVENTION OF CROSS-VM CACHE-BASED SIDE CHANNEL ATTACKS

ZAKIRA INAYAT

THESIS SUBMITTED IN THE FULFILMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

UNIVERSITY OF MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Zakira Inayat

Registration/Matric No: WHA130033

Name of Degree: Doctor of Philosophy

Title of Project Paper/Research Report/Dissertation/Thesis: ("On the Prevention of Cross-VM Cache-Based Side Channel Attacks"):

Field of Study: Information Security (Computer Science)

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:

Designation:

ON THE PREVENTION OF CROSS-VM CACHE-BASED SIDE CHANNEL ATTACKS

ABSTRACT

The state-of-the-art Cloud Computing (CC) has been commercially popular for shared resources of third party applications. A cloud platform enables to share resources among mutually distrusting CC clients and offers cost-effective, on-demand scaling. With the exponential growth of CC environment, vulnerabilities and their corresponding exploitation of the prevailing cloud resources may potentially increase. While it provides numerous benefits to the CC tenant, however, resource sharing and Virtual Machine (VM) physical co-residency raising the potential for sensitive information leakages such as side channel (SC) attacks. In particular, physical co-residency features allow attackers to communicate with another VM on the same physical machine and leak the confidential information due to inadequate logical isolation. We investigate SC attacks involving the CPU cache and identify that traditional prevention mechanisms for SC attacks are not appropriate for prevention of cross-VM cache-based SC attacks. We go on to demonstrate the prevention mechanisms, however, the existing prevention techniques either require the client to change the software or the underlying hardware and suffer from performance degradation leading to reduce cache usage and increase overhead. To address this problem and improve performance, we investigate that new technique such as dynamic cache partition is necessary to mitigate these sorts of attacks in a cloud environment which is hypervisor-based and does not need the client to change their software and the underlying hardware. Finally, we propose new hypervisor-based mitigation technique, implementing them in a state-of-the-art cloud system which guarantees the security and performance feature of the system. The proposed prevention mechanism is evaluated using various benchmarking experiments. The evaluation results show that merging our proposed method into hypervisor can prevent cross-VM cache-based SC attacks without affecting the performance of hypervisor. Our dynamic partitioned (HBP-DCP based) hypervisor improves the bearable load by increasing the number of request per second by 45% and by decreasing the average response time by 5.58%. Moreover, improve cache utilization that each VM has access to by increasing cache read/modify/write, cache read, and cache write bandwidth in combine by 53.5% and increasing the cache access time by 15.53%, as a result substantially increase the efficiency as significant.

Keywords: Cloud Computing, Cache-based SC Attacks, Cross-VM SC Cache-based SC Attacks, Countermeasure, Dynamic Cache Partition

PENCEGAHAN SERANGAN SALURAN SISI BERASASKAN SILANG-VM CACHE

ABSTRAK

Kajian semasa dalam bidang Pengkomputeran Awan (CC) secara komersialnya telah popular dalam perkongsian sumber aplikasi pihak ketiga. Platform awan membenarkan perkongsian sumber di antara pelanggan CC yang saling tidak mempercayai dan menawarkan penskalaan yang kos efektif dan berdasarkan permintaan. Dengan pertumbuhan persekitaran CC yang pesat, kelemahan dan eksplotasi yang berkaitan antara sumber awan semasa, berpotensi boleh meningkat. Walaupun ia menyediakan pelbagai faedah kepada penyewa pengkomputeran awan, perkongsian sumber dan fizikal mesin maya (VM), ia boleh meningkatkan potensi untuk kebocoran maklumat sensitif seperti serangan saluran sisi. Secara khususnya, ciri-ciri fizikal residensi bersama membolehkan penyerang untuk berkomunikasi dengan VM lain pada mesin fizikal yang sama dan membocorkan maklumat sulit yang disebabkan oleh kekurangan pengasingan logik. Kami menyiasat tentang serangan saluran sisi yang melibatkan cache CPU dan mengenalpasti bahawa mekanisma pencegahan tradisional bagi serangan saluran sisi tidak sesuai untuk pencegahan serangan saluran sisi berasaskan silang-VM cache. Kami memilih untuk menunjukkan mekanisma pencegahan, walau bagaimanapun, teknik-teknik pencegahan sedia ada sama ada memerlukan pelanggan untuk menukar perisian atau perkakasan asas akan menyebabkan kemerosotan prestasi yang boleh mengurangkan penggunaan cache dan meningkatkan overhed. Bagi menangani masalah ini dan meningkatkan prestasi, kami menyiasat teknik baru iaitu pemetakan cache secara dinamik. Ia adalah perlu untuk mengatasi serangan di dalam persekitaran awan yang berasaskan *hypervisor* tanpa perlu menukar perisian dan perkakasan pelanggan. Akhir sekali, kami mencadangkan mitigasi baru berasaskan *hypervisor*, melaksanakannya dalam sistem awan yang mengikut aliran semasa bagi menjamin keselamatan dan ciri-ciri prestasi sistem. Mekanisma pencegahan yang dicadangkan dinilai dengan menggunakan pelbagai eksperimen penandaarasan. Keputusan penilaian menunjukkan bahawa penggabungan kaedah cadangan kami ke hypervisor boleh mencegah serangan SC berasaskan silang-VM cache tanpa menjejaskan prestasi hypervisor. Pemetakan secara dinamik (berasaskan HBP-DCP) *hypervisor* telah meningkatkan tanggungan beban dengan pertambahan jumlah permintaan setiap saat sebanyak 45% dan pengurangan purata masa respon sebanyak 5.58%. Selain itu, ia juga meningkatkan penggunaan cache di mana setiap VM mempunyai akses dan peningkatan jalur lebar bagi operasi baca/kemaskini/tulis cache sebanyak 53.5%, peningkatan masa capaian cache sebanyak 15.53% dan keputusan ini menunjukkan kecekapan meningkat secara purata.

Keywords: Pengkomputeran Awan, Serangan SC, Serangan saluran sisi berasaskan silang-VM cache, tindak balas, Pemetakan cache dinamik

ACKNOWLEDGEMENTS

ALLAH Almighty provides the courage, knowledge, and resources to every human being in this world. I am thankful to ALLAH Almighty for blessing me in every form of human quality such that I have reached this point of life and completed my PhD thesis. My hearty thanks must go to my advisors, family (Especially in Laws), and friends who have supported and encouraged me through difficult times of life. I am highly thankful to my supervisor Prof. Dr. Abdullah Gani who has patiently provided the vision, encouragement and advice necessary for me to proceed through this doctoral program and complete my PhD. I'd like also to extend my gratitude to my co-supervisor, Dr. Nor Badrul Anuar for his deep commitments and continued help and support. Their continuous support and guidance helped me producing a valuable piece of research reported in this thesis.

I would like to sincerely thank my dearest and loveliest parents for their faith in me and allowing me to be as ambitious as I wanted. I owe them everything and I hope that this work makes them proud. I would also like to gratefully express my special appreciation and thanks to my beloved husband, Rahat Ali, for his great support, encouragement and unwavering and unconditional love. Words cannot express how grateful I am to my dearest brother Shahid Anwar for his great support. He had always been a tremendous mentor for me.

Finally, I would like to thank the Malaysian Ministry of Higher Education, Malaysia for the financial support and assistance of the entire period of my PhD. I am also thankful to FSKTM support staff at WISMA RND and University for lending their support and resources. I dedicate this thesis to my beloved parents and husband for their constant support and unconditional love, who believes in me under any circumstances, cheers on my trivial achievements, and always stands beside me in the face of difficulties.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

Symbols	Description
AES	Advance Encryption Standard
CC	Cloud Computing
CMOS	Complementary metal–Oxide–Semiconductor
CMP	Chip Multi-Processing
Cross-VM	Cross-Virtual Machine
DES	Data Encryption Standard
DPA	Differential Power Analysis
FA	Fault Analysis
HBP-DCP	Hypervisor-based Prevention Mechanism using Dynamic Cache Partitioning
IaaS	Infrastructure as a Service
IPS	Intrusion Prevention System
IRS	Intrusion Response System
ISA	Instruction Set Architecture
IT	Information Technology
KSM	Kernel Same-page Merging
LLC	Last Level Cache
MFN	Machine Frame Number
OS	Operating System
PaaS	Platform as a Service
PFN	Physical Frame number
PLcache	Partition Locked Cache
RPCache	Random Permutation Cache
PTP	Prime + Trigger + Probe
SaaS	Software as a Service
SC	Side Channel
SMP	Symmetric Multi-Processing
SMT	Simultaneous Multi-Threading
SPA	Simple Power Analysis
SPT	Shadow Page Table
SSL	Secure Sockets Layer
TLB	Translation Lookaside Buffer
VM	Virtual Machine
VMM	Virtual Machine Manager

CHAPTER 1: INTRODUCTION

This chapter introduces the basis of the research work carried out in this thesis. The background of our initial research domain, Cloud Computing (CC) and Side Channel (SC) is provided. It explains the key motivations in the establishment of a research problem of the thesis leading to highlight our research problem and objectives. The research problem is highlighted from a broad perspective in the form of statements of the problem. The research aim and objectives are highlighted in the domain of side channel attacks in CC. Furthermore, the research methodology employed to address the research problem is presented.

The structure of the remainder of the chapter is as follows: Section 1.1 presents the background knowledge of the field of research namely CC, cache-based SC attack, and cross-VM SC attacks. Section 1.2 presents the motivation of inspiring the research provided in this thesis. In Section 1.3, the established research problem is presented. Section 1.4 provides the research aim and objectives. In Section 1.5, the research methodology employed to address the research problem is defined. Finally, Section 1.6 presents the layout of the rest of the thesis.

1.1 Background

Cloud Computing (CC) can be defined as a new paradigm that delivers computing and IT as services. The cloud resources on-demand concept has attracted end users to utilize various CC services, such as “Software, Platform, and Infrastructure” as-a-service (“SaaS, PaaS, and IaaS”) at low cost (Zhang, Cheng et al. 2010). As a new paradigm, CC acquires more importance and brings unique features and vulnerabilities in today Information Technology (IT) industry. Specifically, it introduces multi-tenancy to facilitate the users to share computing physical resources provisioned over the Internet and offers cost-effective, on-demand scaling to the CC tenants. Moreover, it establishes the new concept in computing namely mutually distrusting co-resident clients as a valid execution state. Although mutually distrusting co-resident and multi-tenancy provide numerous benefits to the CC tenants, this paradigm introduces a new concept known as client's co-residence and VM's physical co-residency. However, the security vulnerabilities arise from these well-known concepts because it enables a new form of sensitive information leakage. One of the security vulnerability to CC is the SC attacks which exploit the information leakage channel at the micro-architectural level. The CC infrastructure relies on the virtualized servers that provide the required logical isolation between guest VMs through sandboxing. However, this isolation was described to be imperfect in the past research work which exploited the information leakage channel to extract the sensitive information across co-located VMs. Co-residence clients and physical co-residency of VMs allow the attacker's VM to communicate with the victim's VM running on the same physical device that by design they are unable to have access (Ristenpart, Tromer et al. 2009).

Since CC is not equivalent to physically separated systems and due to an inadequate logical isolation, it facilitates the co-located malicious VM to use the SC attacks to leak sensitive information about the victim VM functionality and exploit the correlation between the software and hardware. SC attacks use the unconventional methods including cache access and timings to extract and transfer confidential data in a way that violate security policies have been identified as a major issue in implementing cryptographic algorithms. Although cryptographic algorithms provide security to the sensitive information from attackers by encrypting and decrypting sensitive data. However, CC is a big concern for cryptographers because they are putting their data and program out there away from their trusted computers (Ristenpart, Tromer et al. 2009). The encryption keys of the cryptographic algorithms e.g., Advance Encryption Standard (AES) are extracted by the attackers using simple spying processes by the attacker to analyze information about cache lines, which have been accessed. In addition, AES in various well known cryptographic libraries namely OpenSSL, polarssl, and libcrypto are vulnerable to information leakage attacks, when running in different hypervisors' including XEN and VMware used by cloud service providers. The current VM in the processor analyzes this cache information. Although the cache data is protected, the metadata about cache is not fully protected (Tromer, Osvik et al. 2010).

Since SC attacks are physical attacks, they require the fundamental characteristics of computation including power consumption, timings it takes to run a program and exploitation of hardware to extract the secret information of the cryptographic algorithms (e.g., encryption key). This attack typically works by creating the correlation between the functionality of the underlying hardware in the physical device and the software. Moreover, this correlation can be used to exploit the co-located VMs by interpreting the internal execution of the software program from the observed phenomena of hardware at a specified time. This allows SC attack to be conducted in an environment where the attacker and the victim have access to the same hardware in the absence of any prevention mechanism. In order to exploit the physical properties of the machine, the attacker and victim have to access the same hardware by using hardware and software SC attacks.

Although in a traditional system, gaining access to the same physical device as a target was a difficult task in establishing SC attacks. However, CC environment makes it easy to accomplish SC attacks. In a non-virtualized environment, it is difficult to launch the SC attack as compared to in a virtualized environment. This is because, in a virtualized environment, the attacker and the victim are co-located on the same physical machine. Since SC attacks are used to extract the cryptographic information, thereby, can be implemented on all those devices which used cryptography concepts for securing their data such as smart cards, mobile phones, tablets, personal computers, and servers (Fisk, Fisk et al. 2002).

SC attacks are categorized into various types according to the specific piece of hardware medium they target and exploit and have been discussed in Chapter 2 in detail. Since CPU caches are the high interacting and sharing devices between processes and are always been targeted by the adversaries. Therefore, it enables us to categorize the SC attacks in this thesis, specifically cache-based SC attacks and the prevention mechanism based on the exploited hardware medium and physical characteristics of computation.

The state-of-the-art literature shows that a large number of cache-based SC attacks have been studied in the past in non-virtualized multilevel systems including database, Operating System (OS) and networking (Zander, Armitage et al. 2007). For instance, Bernstein's proposed SC attack based on the cache access time variation (Bernstein 2004). The author used the access time information (whether the data is being accessed from the cache or from main memory) to extract AES key. Moreover, the co-residency feature of CC makes cache-based SC attacks more effective in the virtualized environment. In 2009, the first cache-based SC attack became visible in the community when Ristenpart et al. (Ristenpart, Tromer et al. 2009) successfully implemented this attack in the virtualized environment by using the co-residency feature. Because of these information leakage channels, information security in a public or shared cloud environment is a general concern that must be considered.

Since these attacks are always implemented by using the hardware or software channels, therefore, the defensive mechanisms for such attacks are also implemented on the hardware channel as well as through software (Zhang, Juels et al. 2012). Although hardware-based prevention mechanisms provide security from SC attacks, these mechanisms require changing the underlying hardware. The changing of hardware would take longer time as well as expensive and the SC attacks would be succeeded before the changing of hardware. Therefore, the software-based prevention mechanism is required which prevents the SC attacks before occurring and which is hypervisor-based that comply with the cloud model and does not need to change the software and the underlying hardware.

1.2 Motivation

CC is a rapidly growing technology in terms of both research work and commercial applications. Over the last five years, CC has grown exponentially from its origin to the existing vast research and application development industry. It is predicted that CC market will grow approximately to over $45.90 billion by 2018. Despite the characteristics such as dynamic provisioning, multi-tenancy, scalability, and ease of integration as shown in Table 2.1 in detail, CC is vulnerable to SC attacks because of its easy accessibility and distributed infrastructure. In spite of this threat to CC, the users of the cloud are increasing day by day as shown in Figure 1.1.

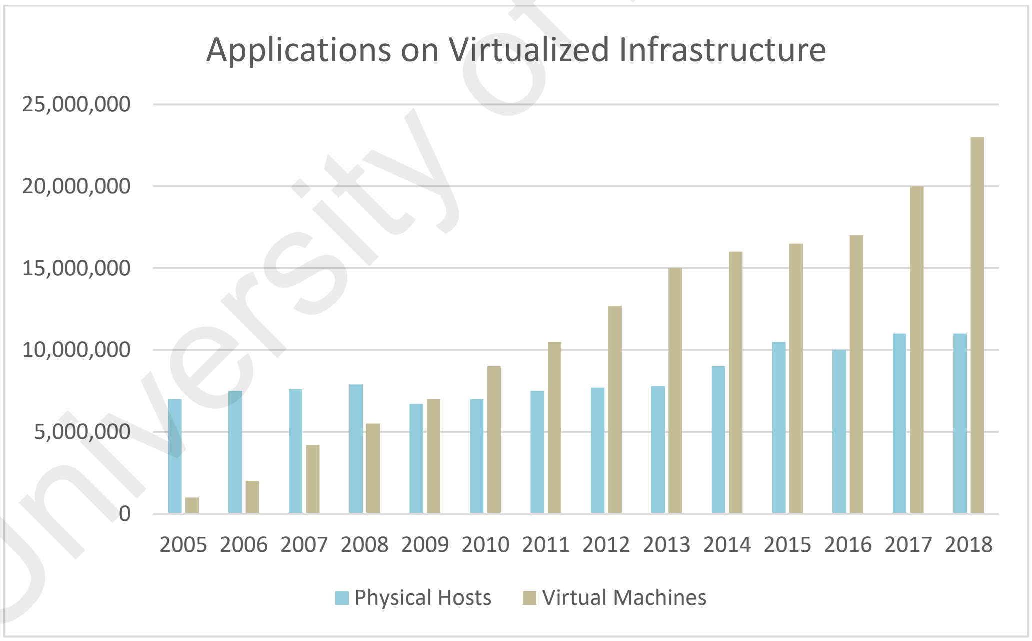


Figure 1.1: Year wise Progression of Cloud Computing Usage (Irazoqui, Inci et al. 2014)

Figure 1.1 shows the gradual increase from standard technology to virtualized environment. Moreover, it shows an increase in the number of cloud users compared to physical machine users. The statistic shows that from 2013 to 2017, there is approximately one billion increase in the number of online users of cloud-based service through all over the world. The increase in cloud-based service users is estimated from the fact that in 2013 the cloud users was 2.4 billion and it is predicted that it would be approximately exceeded to 3.5 billion in 2018 (Portal 2016).

According to a recent Tech News report (NEWS 2015), Apple announced a major vulnerability to the security of iPhones that some Canadians attacked the iPhones and iPads with malware that could extract their iCloud passwords and other personal confidential data. Similarly, according to Digital Forensics Association (DFA), from 2009 to April 2016, the lost records' quantity in data breaches ranged from 1 to 140,000,000 with an average of 407,926 (DFA 2016). Even if a packets contains only single bit can be covertly transmitted , 26 GB of data could be leak on a large Internet Site through SC attacks (Zander, Armitage et al. 2007). Moreover, the CERT statistics in 2017 indicated a 50% increase in the information leakage from insider attackers and reported more than 40 % of SC attacks (Cert. 2017.). These reports prove that the effect of SC attacks are unavoidable. This is the reason that motivated researchers to explore information leakage channel namely SC attacks in cloud environment.

CC is a distributed computing paradigm that enables on-demand access to a shared pool of scalable computing resources. As a new design paradigm in computing, the goal of CC is computing consolidation and multi-tenancy. Multitenancy employs virtualization to share computing physical resources among CC customers. Since CC provide the logical isolation to cloud resources through sandboxing mechanism across guest VMs and does not provide the physical isolation. Therefore, data is vulnerable to information leakage due to the concept of co-residence clients and physically co-residence VMs provided by CC (Irazoqui, Eisenbarth et al. 2015). Unlike other multilevel systems (i.e., OSs, databases, networking etc.,), CC allows attackers to access the same hardware and perform malicious activities among their own users. Specifically, attackers exploit the physical characteristics of computation and hardware side-channels to access the place such as cache that by design is restricted to them and gain the information. To limit this leakage channel, the cache must be divided across the VMs through software mechanism.

Unlike encryption, which protects confidential information from being decoded by unauthorized persons, SC attacks aim to attack the encryption systems and to hide the occurrence of communication. Since the evidence of the existence of communication is sufficient to detect the physical properties of computation. So encryption is unable to prevent attackers from detecting the pattern of communication (Zander, Armitage et al. 2007). Therefore, cloud provider, criminals, terrorist company, or government organization have the interest to hide their confidential communication.

The Cloud features affected by cross-VM SC attacks are data outsourcing, multi-tenancy, and co-residency. Presently, more than 79% of organizations attempt to utilize data outsourcing, because approximately 75% of the total ownership cost is assigned to manage of in-house huge storage. Since any co-resident VM can perpetrate cross-VM attacks through a covert channel. Therefore, end-users trust will be declined on cloud-based application.

Although there are several defensive mechanisms such as firewall, cryptography, and access control, however, these are unable to protect cloud environment from SC attacks. Moreover, some prevention mechanisms need to change the client software and the underlying hardware. Therefore, there is a need for a prevention mechanism for cross-VM cache-based SC attacks which is software-based and does not need to change the client software and the underlying hardware. In this thesis, the software-based prevention mechanism is proposed. Our proposed prevention mechanism for cross-VM cache-based SC attacks, need to be followed by the two key points of the cloud model. First, it does not need to modify the software on the client-end of interfaces it intends to run and second it does not require the modification of underlying hardware.

1.3 Statement of the Problem

Although side-channel attacks have existed in the multilevel system including databases and OSs in the past (Osvik, Shamir et al. 2006), the novel co-residency feature of CC makes them more effective in virtualized environment. Due to shared technology, the attacker is no longer required to gain unlawful or restricted access to the victim's hardware, which bypasses the first line of defense against such attacks. Because a side-channel requires the exploitation of the underlying hardware and software, each defensive mechanism must also be specifically adapted for the underlying hardware and software channel. Therefore, it enables us to categorize side-channel attacks and the defensive mechanism based on the hardware and software channel they exploit. Since each channel provides unique vulnerabilities. The CPU cache is one of the most frequently used pieces of shared hardware and often deals with sensitive data. Thus it become one of the most common targets for use in a SC attack as it can more easily be used to extract useful data at a high rate. An attack made over this channel is referred to as a cache-based SC attack.

Multiple prevention mechanisms are available to prevent cache-based SC attacks in multiprocessing systems including OS, databases, and networking. However, these existing mechanisms are unable to prevent the cross-VM cache-based SC attacks, as the cloud facilitates the users with the shared resources (Kim, Chandra et al. 2004, Percival 2005). Determined adversaries have the ability to place malicious hosts in the cloud environment on the same machine as a target host (Ristenpart, Tromer et al. 2009). The malicious hosts are then able to monitor and manipulate the shared cloud resources, including caches and other hardware resources in order to leak critical information from the target hosts. In a cloud environment, the prevention mechanisms are divided into hardware-based and software-based prevention mechanisms, and hardware-based are implemented on the hardware level (Kim, Chandra et al. 2004, Percival 2005). However, hardware based countermeasures are unable to provide an immediate solution to the problem. They will take the time to develop and are failed to protect the existing hardware. In addition, hardware-based prevention mechanisms are expensive and need special design hardware to support cache control, or need individual cache, need to disable the cache, or need to change the replacement policy of cache (Osvik, Shamir et al. 2006). Consequently, for immediate mitigation of SC attack, software-based prevention solutions are required. Because software-based solution can be implemented in the already existed architecture. Therefore, there is a need for a software-based mechanism for the prevention of cross-VM cache-based SC attacks.

One of the software-based prevention mechanism for cross-VM cache-based SC attacks is static partition which use the page coloring technique to partition the entire cache into static parts for various VMs during boot time. Page coloring allows the hypervisor to limit the cache usage of any application and VMs. However, this method can only configure the cache usage of each virtual machines at boot time and once the VM is created we cannot change its configuration. For instance, if we partition the entire cache into 16 parts during boot time and currently one VM is executing, then only one part would be assign to that VM and the remaining 15 parts will be idle. We cannot change the entire cache partition according to running VMs during execution. Moreover, static cache partition degrades the performance in term of bearable load, cache utilization, and cache access time. The VMs are only allowed to allocate the memory at boot time from the same partition that belonging to the same VM.

Since a single VM running in a 4-way partitioned system, therefore, one-quarter of the total memory is assigned to that VM may lead to wasted resources. Consequently, the memory resources are maximized even balancing of loads. Therefore, there is a need for a preventive mechanism for SC attacks which dynamically partition the entire cache for each VM upon the creation of new VM. Once the VM is created then we would be able to configure the entire cache for various VMs. For instance, if one VM is created then the whole cache memory is assigned to that VM on a dynamic basis. If two or three VMs are created then the cache memory is divided into 2 or 3 partitions accordingly and would be assigned to those VMs. Moreover, prevention mechanism of SC attacks, need to be followed by the two key points of the cloud model. First, it does not need to modify the software on the client-end of interfaces it intends to run and second it does not need the modification of underlying hardware.

Our goal is to provide a defense capable of preventing cache-based side-channels in the Cloud while not interfering with the Cloud model and without degrading the system performance. Using the code base of an open source hypervisor, Xen (Project 2016), we have conducted our solution based on dynamic cache partition demonstrate to inhibit cache-based side-channels from occurring within a Cloud server. In our solution, all cache monitoring and cache partitioning operations are done transparently within the hypervisor or VMM. Therefore it is applicable to commodity operating systems such as Windows, of which the source code is unavailable. Second, because guest OSes are black boxes to the VMM, this single mechanism is portable across all the OSes supported by the VMM. Our solution is implemented in the hypervisor, therefore it provides cache partitioning both within and across OSes and also provides more flexibility and opportunities for the whole-system optimization.

1.4 Statement of Objectives

This research is undertaken with the aim to prevent cache-based SC attacks across VMs and in CC with minimum overhead in terms of bearable load, cache utilization, and cache access time. The research aim is accomplished by addressing the following objectives:

- To study the existing SC attacks in virtualized and non-virtualized environment involving CPU-cache to gain an insightful understanding to the performance limitations of current state-of-the-art prevention mechanisms for these attacks.

- To investigate the identified problem by conducting the cross VM cache-based SC attacks in the real environment and applied the existing prevention mechanism based on the static cache partition and unveiling the impact of existing prevention mechanism on the load, cache utilization, and cache access time as well as on the cloud model.
- To propose a prevention mechanism based on the dynamic cache partition for the prevention of cache based SC attacks across VMs that leads to an efficient cache utilization among various VMs.
- To evaluate and validate the performance of our dynamic cache based prevention mechanism considering three metrics namely: computing load, cache utilization, and memory access rate and compare it with the state-of-the-art prevention mechanisms.

1.5 Research Methodology

The research carried out in this thesis used the following four main steps in order to achieve the four objectives defined in Section 1.4. The proposed research methodology along with the detail description of research objectives corresponding to each research phase is given in Figure 1.2.

- The state of the art SC attacks in CC with emphasize on cache-based SC attacks across VMs are reviewed in the first phase. The SC attacks are generally categorized based on the computing location, on the way of implementation, and on the way of accessing the modules. Moreover, the SC attacks based on the computing location are classified into: intra-VM, cross-platform, and cross-VM SC attacks. Similarly, the SC attacks-based on the implementation are divided into sequential and parallel SC attacks. In addition, the SC attacks based on the way of accessing the module is further divided into invasive, non-invasive, and semi-invasive attacks. Since all the aforementioned attacks have been discussed in the non-virtualized environment including database, networking and OS for many years. Therefore, in this research work our focus is on the cache-based SC attacks in the virtualized environment (e.g., Cross-VM cache-based SC attacks). The aim of this thesis is to explore the SC attacks involving CPU-cache and their mitigation techniques in a state-of-the-art cloud system to improve security in CC. We also categorized the cross-VM cache-based SC attacks according to the hardware medium they target and exploit, the ways they access the module and the method they use to extract the confidential information. Through a comprehensive literature review, we identify the most significant research problem to cross-VM cache-based SC attacks to address in this research.

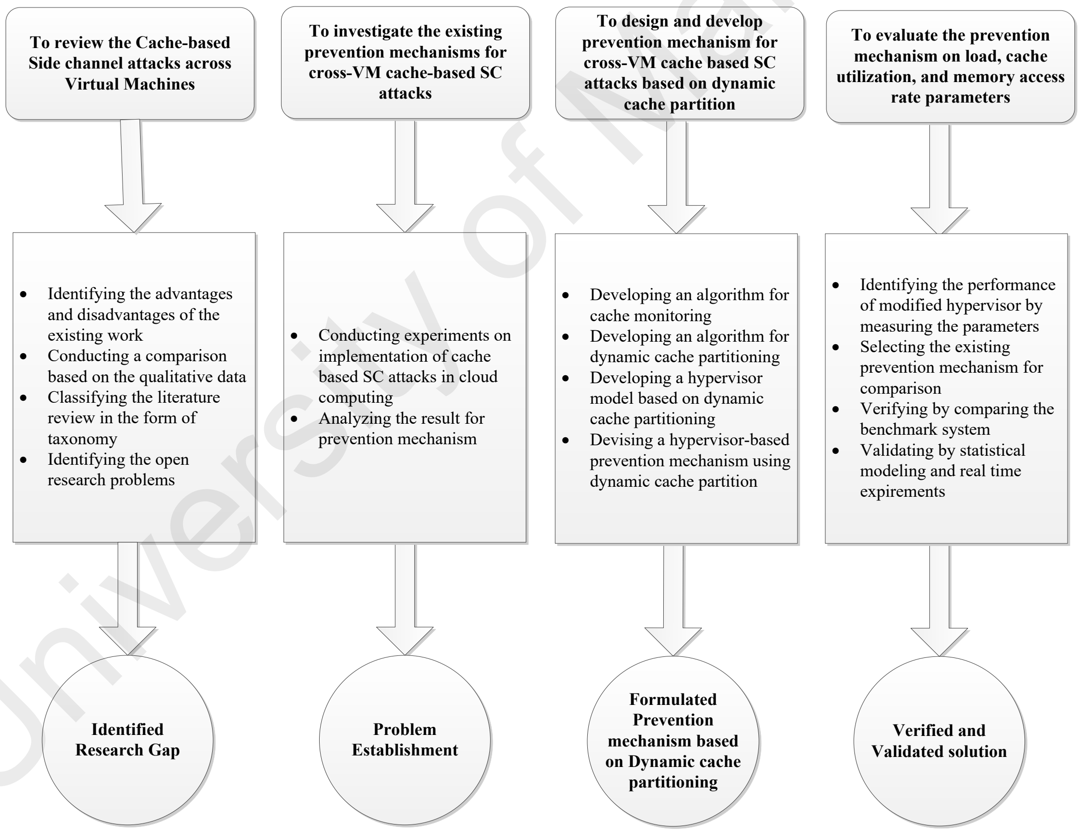

To review the Cache-based Side channel attacks across Virtual Machines
To investigate the existing prevention mechanisms for cross-VM cache-based SC attacks
To design and develop prevention mechanism for cross-VM cache based SC attacks based on dynamic cache partition
To evaluate the prevention mechanism on load, cache utilization, and memory access rate parameters
Identifying the advantages and disadvantages of the existing work
Conducting a comparison based on the qualitative data
Classifying the literature review in the form of taxonomy
Identifying the open research problems
Conducting experiments on implementation of cache based SC attacks in cloud computing
Analyzing the result for prevention mechanism
Developing an algorithm for cache monitoring
Developing an algorithm for dynamic cache partitioning
Developing a hypervisor model based on dynamic cache partitioning
Devising a hypervisor-based prevention mechanism using dynamic cache partition
Identifying the performance of modified hypervisor by measuring the parameters
Selecting the existing prevention mechanism for comparison
Verifying by comparing the benchmark system
Validating by statistical modeling and real time expirements
Identified Research Gap
Problem Establishment
Formulated Prevention mechanism based on Dynamic cache partitioning
Verified and Validated solution

Figure 1.2: Research Methodology

- The second phase of this research involves the investigation of the identified problem and verification of its significance through experimental analysis between two VMs using Xen hypervisor in a real CC environment. By real implementation of cache-based SC attacks between two VMs in Xen and VMWare hypervisor, we analyzed that these attacks are extracting the secret cryptographic key via cache information and are very dangerous in the virtualized environment. The static cache partition at boot time as a solution to these attacks is exercised to reveal degradation in the performance in terms of load, cache utilization, and cache access time.

- In the third phase of this research work, we implement and design HBP-DCP (Hypervisor-based Prevention Mechanism using Dynamic Cache Partitioning) prevention mechanism that prevent cross-VM cache-based SC attacks to alleviate the identified problem. HBP-DCP consists of two algorithms: one for cache monitoring and one for page allocation to each requested VMs. These algorithms are embedded into the source code of page allocator of existing hypervisor. The basic objective of cache-based SC attacks is that target VM1 traces the cache access and access time variation of the victim VM2 to extract the secret information of secret cryptographic key of the encryption algorithms (e.g., AES, DES). To prevent the cache access between VMs e.g., victim VM1 from attacker VM2, our proposed HBP-DCP prevention mechanism divide the cache into partitions on dynamic basis that no VM would access the partition assigned to another VM. In addition, it divide the cache into different color on dynamic basis and assign the specific color to each VM.
- We implemented and evaluated the performance of our proposed HBP-DCP prevention mechanism through benchmarking experiments in the last phase of our research. A set of standard computation benchmarking along with matrix multiplication and customized benchmark are used to evaluate the performance of our proposed HBP-DCP prevention mechanism. A real testbed environment is created by using Xen hypervisor. Load testing, cache utilization, and memory access rate are the performance evaluation metrics in this experiment. We synthesize the result of these three parameters using modified (dynamic partitioned /HBP-DCP/secure): the case of our solution) with the result of the unmodified (default/insecure) and the static

partitioned hypervisor. Moreover, we devised a statistical model to analyze and validate the result of performance evaluation metrics. The statistical model is devised using regression model which is a predominant observation-based modeling and analysis method. The statistical model is validated using split-sample validation approach. The empirical results of our performance evaluation are validated through the statistical regression model.

1.6 Thesis Layout

The research entitled "On the prevention of cross-VM cache-based SC attacks" is comprising of an extensive study. Therefore, the thesis has been divided into chapters for a clear reader understandability. The thesis is comprised of 7 chapters and the layout of the thesis is presented in Figure 1.3.

Chapter 2 aims to review the research undertaken in the field of cross VM cache-based SC attacks. The chapter describes knowledge about the CC and the vulnerability of SC attacks to identify and classify the SC attacks across VMs and in CC. Moreover, in this chapter, cross-VM cache-based SC attacks are focused and the detail about the prevention mechanism for these attacks are provided which discover the deficiency of the existing solution. We provide qualitative critical analysis in the aforementioned research direction based on the metrics derived from the proposed taxonomy. The research problems are identified by the literature review expose the need for the prevention mechanism based on the dynamic cache partition for the cross-VM cache-based SC attacks. Furthermore, several research issues are identified for the future research direction.

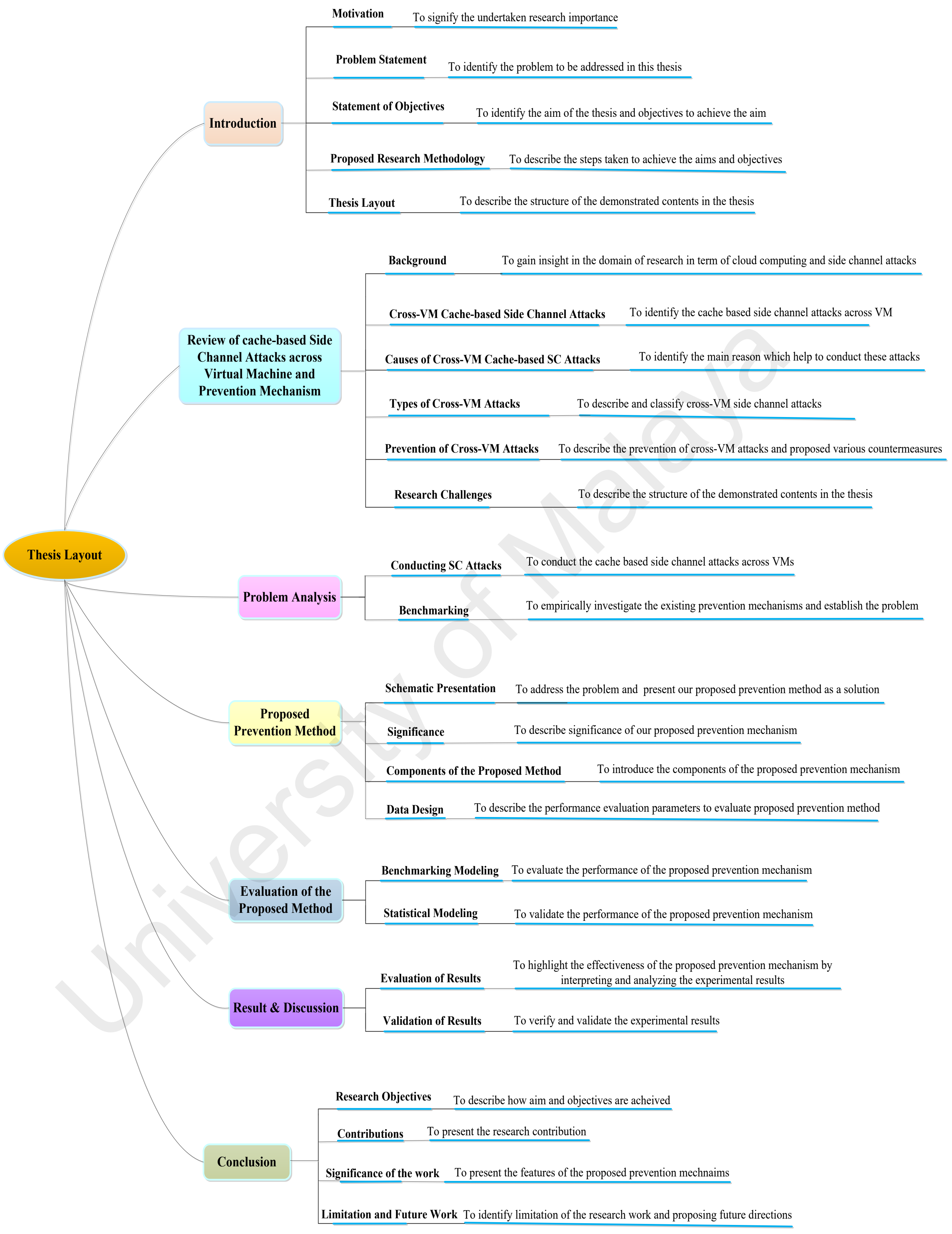
In Chapter 3, we conducted the cache-based SC attacks in the cloud environment in single VM and across VMs. Using series of experiments for conducting these attacks by using the Prime + Probe and Flush + Reload techniques in Linux and across VM, we analyzed that CC is vulnerable to the dangerous information leakage attacks.

Chapter 4 describes HBP-DCP mechanism for the prevention of cache-based SC attacks across VM in the CC environment. The objectives and assumption undertaken by the technique are presented. Moreover, the schematic presentation of the proposed prevention mechanism is presented and each component of the technique is described in detail. The significance of the proposed technique is highlighted and the performance evaluation parameter is derived.

Chapter 5 reports on the performance evaluation methodology for the HBP-DCP technique. The experimental setup is explained with accompanying benchmarks and the devices. The data collection method regarding the experimental and evaluation methods namely statistical modeling and benchmarking is described that have been utilized to evaluate and validate the proposed technique performance. The benchmarking application is described and the technique to evaluate the statistical modeling is also demonstrated.

In **Chapter 6**, we present the result of the experimental performance evaluation of the HBP-DCP technique to prove its efficiency and significance. The experimental evaluation is based on three parameters, namely load, cache utilization, and memory access rate. We compare and contrast the result of benchmarking with the statistical model result to validate the performance of the proposed method.

Finally, **Chapter 7** concludes this work by revisiting the aim and objective of this research that how it is fulfilled. The main contribution of the research is summarized and the significance and the method proposed in this thesis are highlighted. The future research directions and limitations conclude the chapter.



Thesis Layout
Introduction
Motivation
To signify the undertaken research importance
Problem Statement
To identify the problem to be addressed in this thesis
Statement of Objectives
To identify the aim of the thesis and objectives to achieve the aim
Proposed Research Methodology
To describe the steps taken to achieve the aims and objectives
Thesis Layout
To describe the structure of the demonstrated contents in the thesis
Review of cache-based Side Channel Attacks across Virtual Machine and Prevention Mechanism
Background
To gain insight in the domain of research in term of cloud computing and side channel attacks
Cross-VM Cache-based Side Channel Attacks
To identify the cache based side channel attacks across VM
Causes of Cross-VM Cache-based SC Attacks
To identify the main reason which help to conduct these attacks
Types of Cross-VM Attacks
To describe and classify cross-VM side channel attacks
Prevention of Cross-VM Attacks
To describe the prevention of cross-VM attacks and proposed various countermeasures
Research Challenges
To describe the structure of the demonstrated contents in the thesis
Problem Analysis
Conducting SC Attacks
To conduct the cache based side channel attacks across VMs
Benchmarking
To empirically investigate the existing prevention mechanisms and establish the problem
Proposed Prevention Method
Schematic Presentation
To address the problem and present our proposed prevention method as a solution
Significance
To describe significance of our proposed prevention mechanism
Components of the Proposed Method
To introduce the components of the proposed prevention mechanism
Data Design
To describe the performance evaluation parameters to evaluate proposed prevention method
Evaluation of the Proposed Method
Benchmarking Modeling
To evaluate the performance of the proposed prevention mechanism
Statistical Modeling
To validate the performance of the proposed prevention mechanism
Result & Discussion
Evaluation of Results
To highlight the effectiveness of the proposed prevention mechanism by interpreting and analyzing the experimental results
Validation of Results
To verify and validate the experimental results
Conclusion
Research Objectives
To describe how aim and objectives are acheived
Contributions
To present the research contribution
Significance of the work
To present the features of the proposed prevention mechnaims
Limitation and Future Work
To identify limitation of the research work and proposing future directions

Figure 1.3: Summary of Chapters Presented in this Thesis

CHAPTER 2: LITERATURE REVIEW

This chapter presents a literature review on the cache-based cross-VM SC attacks and countermeasure for these attacks. The purpose of this chapter is to detail the literature work related to our problem domain in order to identify the potential research issues in the field of SC attacks and their countermeasures in virtualized environment. The primary research issues identified through the literature review is that with the exponential growth of CC environment, vulnerabilities and their corresponding exploitation of the prevailing cloud resources may potentially increase. CC supports multi-tenancy, physical co-residency features which enable resource sharing among mutually distrusting CC clients and offers cost-effective, on-demand scaling. Although, these features provides numerous benefits to the CC tenant, however, resource sharing and VMs physical co-residency enable a new form of sensitive information leakage such as SC attacks. Unlike encryption, which protects information from being decoded by unauthorized persons, SC attacks aim to attack the encryption systems and to hide the existence of communication. Initially, SC attacks were identified as the main threat on multi-level secure systems i.e. OS, database, and networks. More recently the focus of researchers has shifted toward SC attacks in CC. The target of this article is to explore SC attacks, especially cache-based cross-VM SC attacks and countermeasure in CC and how they compare to traditional SC attacks and countermeasure. The taxonomies are devised with reference to cache-based cross VM SC attacks and countermeasures for these attacks. Qualitative comparison of the state-of-the-art research works is detailed in each section. The chapter also provides the basic knowledge of the technical elements found in the thesis such as cache-based SC attacks, Cross-VM cache-based SC attack, and countermeasures for these attacks.

The rest of this chapter is organized is as follows: Section 2 discusses the background detail of cache-based SC attacks and to classify the cache-based SC attacks into different types. Section 3 describes the SC attacks in the cloud environment. Section 4 provides the prevention mechanism for cross-VM SC attacks followed by the discussion on the existing cross-VM SC attacks and proposed countermeasure. Finally, Sections 6 conclude this chapter by comparing existing approaches and providing a general design approach for prevention of SC attacks.

2.1 Background

This section describes the background detail about the CC, cache-based SC attacks, and previous work related to SC attacks in the cloud. It also describes techniques to implement cache-based SC attacks. Since the cloud users use the same hardware and the computational properties of hardware channel namely power consumption and time are mostly used for these types of attack. Since a cache is the most accessed hardware, most targeted hardware channel for SC attacks, therefore, this study includes a detail description of cache-based SC attacks and their typical prevention techniques.

Although there are existing surveys which explored SC attacks in detail (Osvik, Shamir et al. 2006). However, they investigated the cache-based SC attacks in the non-virtualized environment including database, networking, and OS. To the best of our knowledge, this is the first survey which explores the cross-VM cache-based SC attacks as well as cache-based SC attacks in CC and proposed some countermeasures in the virtualized environment. The aim of this thesis is to explore the SC attacks involving CPU-cache and their mitigation techniques in a state-of-the-art cloud system to improve security in CC. We categorized the SC attacks according to the hardware medium they target and exploit, based on the ways of accessing the module and the method used to extract the confidential information. We also investigate countermeasures for their prevention, required to improve the security in CC.

2.1.1 Cloud Computing

CC can be defined as a new paradigm that delivers computing and IT as a service as shown in Figure 2.1. The cloud resources on-demand concept has attracted end users to utilize various CC services, such as “Software, Platform, and Infrastructure” as-a-service (“SaaS, PaaS, and IaaS”) at low cost (Zhang, Cheng et al. 2010). However, CC is a big concern for cryptographers because they are putting their data and program out there away from their trusted computers (Ristenpart, Tromer et al. 2009). Therefore, security in CC is a critical issue given the distributed infrastructure and user-friendly nature of this technology. Cyber threats to the cloud environment are different from the threats to traditional systems (Security 2010).

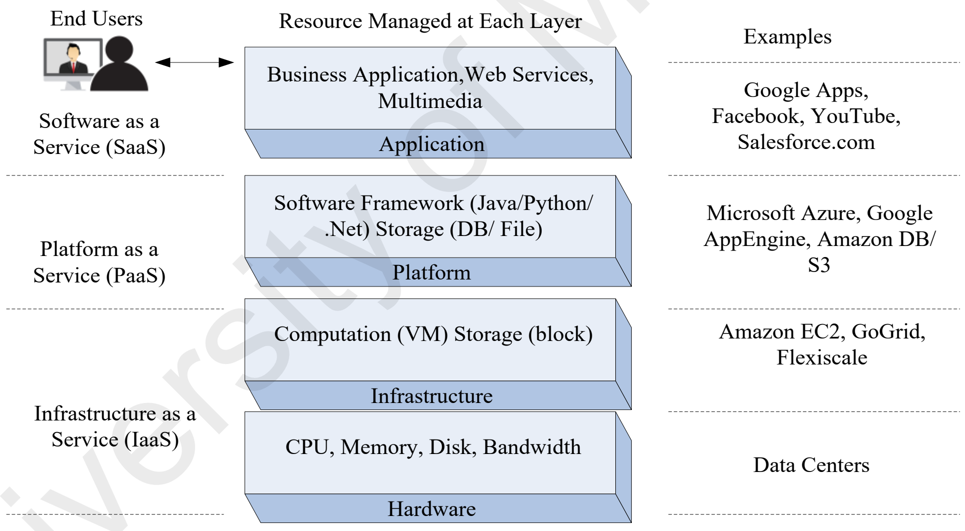

End Users
Resource Managed at Each Layer
Examples
Business Application,Web Services, Multimedia
Application
Google Apps, Facebook, YouTube, Salesforce.com
Software as a Service (SaaS)
Software Framework (Java/Python/ .Net) Storage (DB/ File)
Platform
Microsoft Azure, Google AppEngine, Amazon DB/ S3
Platform as a Service (PaaS)
Computation (VM) Storage (block)
Infrastructure
Amazon EC2, GoGrid, Flexiscale
Infrastructure as a Service (IaaS)
CPU, Memory, Disk, Bandwidth
Hardware
Data Centers

Figure 2.1: Layered Model of Cloud Computing

As cloud service providers offer their customer unlimited use of shared cloud resources, this makes the cloud environment vulnerable to attacks. Furthermore, CC facilitates end users with a set of API and software interfaces, opening a window for intruders. As company delivers services (SaaS, PaaS, IaaS) from cloud provider in a scalable way, they provide an opportunity for intruders to gain an inappropriate level of control over the cloud resources and this shared technology of CC enables intruders to extract information in the form of SC attacks. Table 2.1 describes the characteristics of

CC. CC introduces a multitenancy feature, however, this new concept of co-residence client and physical co-residency enables hardware and software covert and SC attacks.

Table 2.1: Characteristic of CC

Characteristics	**Description**
Dynamic provisioning	Mobile users execute their application in a flexible way without any advance reservation for cloud resources
Scalability	The deployment of mobile applications meet the unpredictable demand
Multi-tenancy	Multi-tenancy provides sharing technology of cloud resources
Ease of integration	Multiple cloud services from different cloud service providers can be integrated to meet user demands

Despite these characteristics, CC is vulnerable to SC attacks because of its easy accessibility and distributed infrastructure. Although there are several defensive techniques such as firewall, cryptography, and access control, however, are unable to protect cloud environment from SC attacks. Therefore, there is a need for a preventive mechanism for SC attacks.

2.1.1.1 Virtualization

Besides the benefits of multi-tenancy and physical co-residency, CC has another characteristic called virtualization. Virtualization involves the abstraction of the physical machine to OSs in multiple VM on the same physical device isolated by the Virtual Machine Manager (VMM) or hypervisor. In virtualization, the hypervisor namely the XEN and VMware are responsible for the communication between VM as shown in Figure 2.2. Although the hypervisor uses sandboxing techniques to provide logical isolation across guest VMs for modern virtualization, this logical VM isolation is not equal to physical isolation. It is also not sufficient if the attacker uses the SC attacks to circumvent them because VM uses the same hardware, which is a serious threat to VM logical isolation (Ristenpart, Tromer et al. 2009). The literature shows that attackers can use the SC attacks to acquire detail about the memory access pattern of another program such as the cryptographic algorithm that performs the encryption with an unknown private key. These SC attacks affect and observe the cache state and then analyze the effect on the encryption's execution time, during, or after the execution of encryption. Since the VM resides on the same physical hardware, it is at risk to SC attacks in virtualized environment and this has been a known problem for the last 10 years (Ristenpart, Tromer et al. 2009). For instance, Ristenpart et al. (Ristenpart, Tromer et al. 2009) successfully implemented the cache-based SC attack in the virtualized environment for the first time and violate and break through the logical isolation supplied by a sandboxing mechanism. In fact, he is not only able to co-locate two VM on the identical physical device but also able to extract the key stroke by a victim VM. As described in the following section, the prevention mechanism of SC attacks need to be followed by the two key points of the cloud model.

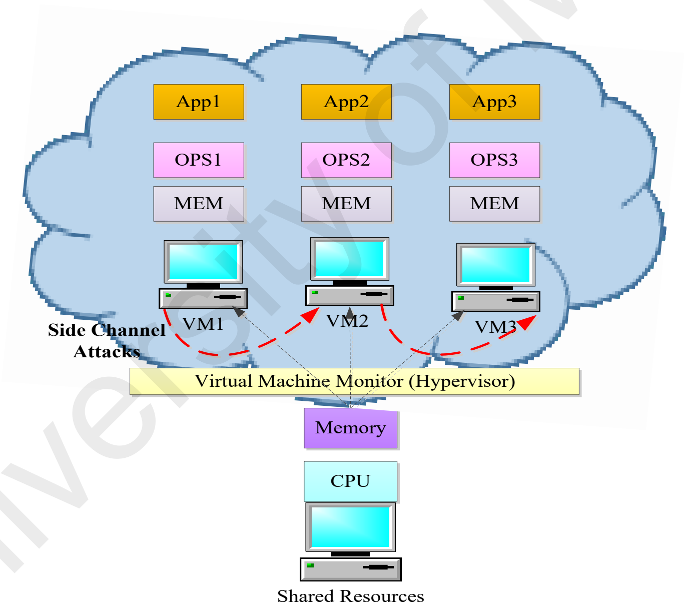


Figure 2.2: Virtualization

2.1.1.2 Cloud Model

We refer the cloud model in this thesis as a specific relationship that the CC has established with its users and the underlying hardware. The two key points that have been highlighted by the cloud model has become commonplace in the CC environment (IBM 2012). According to the first key point, the users have no knowledge or permission to change the cloud software they intend to run and is always able to run canonical software on the cloud. The second point is that the users always run the software that does not need to change the underlying hardware of cloud because the cloud is built on the canonical hardware. According to these two key points of the cloud model, any modification to the CC must comply with the listed two points. Therefore all the solution comply with the cloud model if:

- If it does need any modification in the underlying hardware
- If it does not need the clients to change their software which they intend to run on the cloud

If a solution is developed according to these two points then it complies with the cloud model and can be easily applied to the CC environment without altering the already established functionality of CC. We design our solution server based to keep in mind the above mentioned two point and therefore transparent to the clients and the underlying hardware. The client does not need to change their software as well as does not required to change the underlying hardware.

2.1.2 Side Channel Attacks

Traditionally, in cryptography, cryptographic devices are thought of as black boxes. It means that the only way attackers can gain access to these devices. Since the data and computation are by giving them input and receiving the output of the computation, what is going on within the devices is completely hidden from the attackers. Attackers use physical attacks e.g., SC attacks to gain more information about the data used in the devices. Over the last decade, side channels that transfer confidential data in a way that violate security rules have been identified as a major issue in implementing cryptographic algorithms. Although overt channels utilize the system's secure data object to transmit confidential information in a way that does not violate the security rules. These channels use the data object to hold the information including buffers, files, shared memories, and thread signals. These data objects are normally viewed as a data container. On the other hand, covert or side channels use system resources or entities to transfer information between subjects that are not normally viewed as a data container. In this chapter, a survey is conducted on the cross-VM cache-based SC attacks.

SC attacks are the physical attacks that use the physical process to extract the secret information of the cryptographic algorithms such as encryption key. The computation is a physical process that involves the use of all kinds of physical characteristics of computation such as the timings it takes to run a program, the characteristic of the power consumed during a program execution, electromagnetic radiation, acoustics, and temperature to leak the confidential information. This attack typically works by creating the correlation between the functionality of the underlying hardware in the physical device and the software and this correlation can be used to infer the internal execution of the software program at a specified time. Although the state-of-the-art literature studied these attacks for numerous years in the context of a multi-level embedded system and smart cards, the literature showed that the microprocessor is also vulnerable to these attacks (Bernstein 2005, Percival 2005, Osvik, Shamir et al. 2006). Traditionally, to accomplish a physical attack in multilevel embedded systems (e.g., database and OS) is a difficult task because it requires gaining physical access to the system. However, in a virtualized environment, because of resources sharing, gaining access to a system is very easy.

Smart cards are the most targeted device for SC attacks and because of the noisy nature of these attacks, it is very difficult to collect sensitive information and gain physical access or proximity. However, a virtualized environment makes it possible to gain physical access to the system. The more traditional attacks are used to attack and extract the information from a general-purpose computer e.g., Attacks that authorize an attacker to acquire physical access to the secret data of the entire system by exploiting flaws in OS. SC attacks can be implemented on all devices including mobile phone, PC, tablet, and server, which use a cryptographic algorithm for securing information. For instance, the web browser has an embedded cryptographic algorithm called RSA, which is widely used by the Secure Sockets Layer (SSL) today for secure communication and electronic data transfer over the Internet. Moreover, these attacks are generally categorized into hardware-based channel including power analysis, bus probing and a software-based channel including timing attacks, cache attacks, and memory attacks.

2.1.3 Taxonomy of Side Channel Attacks

SC attacks can be categorized into different types based on the computing location (virtualized and non-virtualized), the implementation, and the ways of accessing the module. The detail of each category is given in the following sections as well as in Figure 2.3.

2.1.3.1 Side Channel Attacks based on the Computing Location

These attacks are categorized based on virtualized and non-virtualized environments as shown in Figure 2.3. In addition, the attacks are also classified on the basis of whether the victim and the attacker have existed in the same cores or in different cores as shown in Figure 2.4. The SC attacks have been studied in a multiprocessing system including the database, OS, and in networking for many years. In these systems, the SC attacks are implemented on the same OS and on same cores and also on the different cores in the same OS. However, in virtualized environment, the attacks are implemented on the different guest OS either on the same or on different cores.

(a) Intra-VM Side Channel Attacks

These are also called process level SC Attacks. Malicious processes P1 and P2 are positioned in the same OS in the domain unit (Dom U) and in the same hardware with different security levels. In the single VM, one higher level secure process P1 (attacker) leaks the confidential information from the process P2 having a low-security level (victim) using the SC attack. These attacks can be implemented in the guest VM where the attacker and the victim have existed in the same or on different cores in the single guest VM. However, process level SC attacks for the traditional personal computer have been surveyed for many years, and several mature defensive mechanisms and analysis techniques have been mentioned in the literature (Brickell, Graunke et al. 2006). The state of the art literature shows several defensive mechanisms for intra-VM or process level attacks (Bernstein 2005, Aciiçmez 2007, Acıiçmez, Brumley et al. 2010). The detail of each one is given in Table 2.2.

(b) Cross Platform Side Channel Attacks

These attacks are also called network level SC attacks. Malicious processes P1 (attacker) and P2 (victim) are placed in different OSs and on different hardware platforms. The network is the main source of communication between these two processes P1 and P2, therefore, these processes use network storage and timing channels to transfer the confidential data in such a way that violates the policy of the system security. SC attacks are mainly based on the entire network, the literature showed the study on these attacks in the non-virtualized environment since 1987 (Zander, Armitage et al. 2007, Irazoqui, Eisenbarth et al. 2015). These attacks can also be implemented in the cloud environment but its prevention solution is already available in the literature (Brickell, Graunke et al. 2006, Osvik, Shamir et al. 2006).

(c) Cross-VM Side Channel Attacks

These are the OS level SC attacks. Malicious processes P1 (attacker) and P2 (victim) are situated in distinct domains but the underlying hardware platform is same. Cross-VM SC attacks are introduced by the hypervisor managed multi-tenancy and VM Co-residency features (Ristenpart, Tromer et al. 2009, Suzaki, Iijima et al. 2011, Wu, Xu et al. 2012, Zhang, Juels et al. 2012). Confidential information (e.g., extraction of a cryptographic key) may be leaked by the SC attacks among VMs and competitive companies that are physically co-located, which will bring huge economic losses to the CC. Cross-VM cache-based SC attacks are further categorized into shared memory-, CPU-load-, and cache-based attacks. In SC attacks based on the shared memory SC attacks, different memory access intervals are used to extract the secret key of any cryptographic algorithm and sensitive information about the memory. In CPU-load based SC attacks, the physical characteristics of computation (e.g., physical resources) such as CPU execution time is used to extract the confidential information and the secret key of any cryptographic algorithm. While in cache-based SC attacks the different cache access latencies (e.g., cache miss and cache hit) are used to transmit and extract data covertly. Details of the cache-based SC attacks are given in the following section. In this thesis, our focus is on the cross-VM cache-based SC attacks which we will elaborate in detail in the upcoming section.

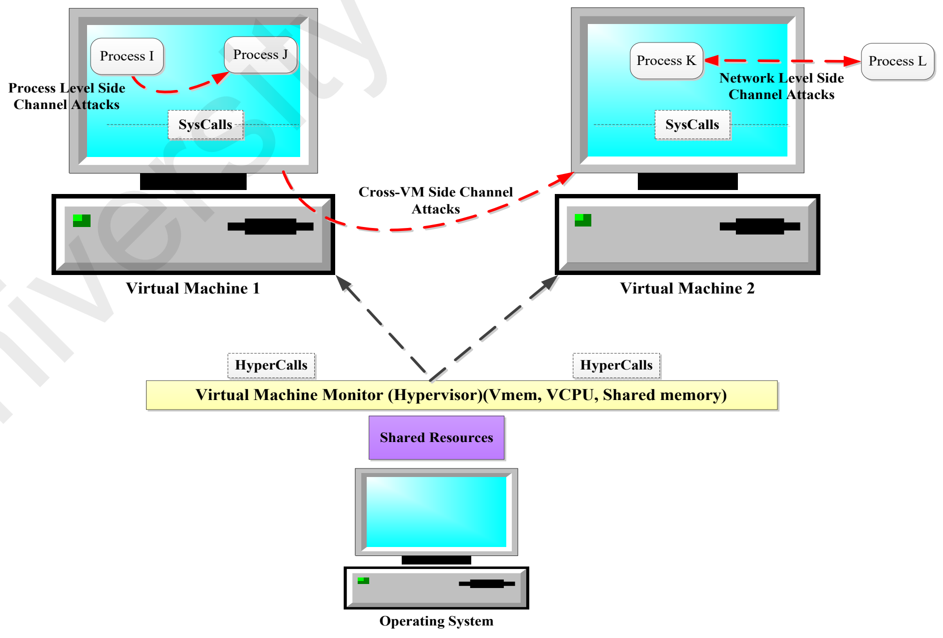

Process I
Process J
Process Level Side Channel Attacks
SysCalls
Process K
Process L
Network Level Side Channel Attacks
SysCalls
Cross-VM Side Channel Attacks
Virtual Machine 1
Virtual Machine 2
HyperCalls
HyperCalls
Virtual Machine Monitor (Hypervisor)(Vmem, VCPU, Shared memory)
Shared Resources
Operating System

Figure 2.3: Types of Side Channel Attack in Hypervisors (XEN)

2.1.3.2 Side Channel Attacks based on Implementation

SC attacks can also be classified into parallel and sequential attacks, based on the implementation as given in Figure 2.4. These attacks are differentiated as to whether they are conducted on parallel or sequential access to the cache memory of CPU. These types of attacks are the most known cache-based SC attacks in CC. The following section describes these two types of attacks in detail.

(a) Sequential Side Channel Attacks

In order to establish cache-based SC attacks, the victim and the attacker need to share some portion of cache memory. In present-day hardware, two different approaches are used to share caches between multiple cores. One approach is that cache is assigned to one CPU core or the cache is accessed by two processes sequentially while the other is for them to have parallel access or the CPU cache is shared between different CPU cores. Sequential access requires a process context switch to be on the same CPU core, whereas concurrent access can be achieved by having a shared cache between CPU distinct cores based on hardware restriction. The literature shows that there is a lot of research for both types of the channel (Wu, Xu et al. 2012). In both types the sequential access is typically seen as more portable, as the concurrent access is to a cache is only allowed by some systems. Sequential SC attacks work in a way that the receiver (attacker) will wait for a message to be read until the sender (victim) writes a message. Due to the ordering, there is a clear window in which the cache can be flushed for prevention purposes that are when the context switch occurs between the attacker and the victim. All other cache-based sequential SC attacks rely on this mechanism, making it a well-known example of a canonical SC attack. Moreover, all cache-based cross-VM SC attacks have been based on this fundamental method; an effective restriction of its principles could, therefore, prevent all current SC attacks in the cloud (Zhang, Juels et al. 2012).

(b) Parallel Side Channel Attacks

Parallel SC can be achieved by adapting sequential SC on a shared-cache system having Last Level Cache (LLC). In this approach, the probing (attacker) and target (victim) processes are located on distinct cores but have concurrent access to the shared LLC. Although the access to a cache memory in both sequential and parallel attacks are the same, the parallel access method does not require to trigger between two VMs. This is because there is no clear gap in the Trigger and the Probe steps and both have occurred at the same time. Although similar to the sequential method, the process originates with the probing executing the "Probe" step, however, unlike sequential this method has no context switch so the target process is started after probe step. Once the cache is primed, like the sequential, the target VM can execute the "Trigger" step rather than the "Trigger" and "Probe" steps are executing concurrently. However, in comparison to a sequential SC attack, the parallel technique is not so reliable as an attack medium because the more noise in the system makes them unreliable, and also because while one VM reads a cache line, the other VM modifies another cache line.

To date, the literature described that only a sequential SC attack can do a very serious destruction in the cloud (Zhang, Juels et al. 2012). Although a parallel channel attack is difficult to conduct, as it still holds the ability to be applied in such an attack and gain unauthorized access to the information about a VM. In addition, it is difficult to flush the cache in parallel access, because the VM might change the cache, rendering it useless and generating too much overhead. Parallel cache-based SC attack can be avoided by restricting the ability of co-resident VMs on the physical machine from evicting one another's data from the cache memory.

2.1.3.3 Side Channel Attacks based on the Way of Accessing the Module

Anderson et al. (Anderson, Bond et al. 2006), categorized SC attacks into invasive, non-invasive and semi-invasive attacks based on whether these attacks have direct or indirect access to the device as shown in Figure 2.4.

(a) Invasive/ Hardware Side Channel Attacks

In this section we discuss the physical attacks involving the interaction of attackers with the chip package and direct physical access to the components by depackaging the chip. The well-known example of this is the direct connection between a wire and a data bus to observe the transfer of data. In addition, these attacks involve the probing or modification of the chip once it is opened. Invasive attacks can be achieved by getting direct access through electrical to the internal parts of the main crypto processor. For instance, to capture signal of a bus line, the attackers place a micro probing needle that can open a hole to get direct access to the passivation layer of a microcontroller chip. These attacks are not limited to a smart card but can also be performed on Complementary metal–Oxide–Semiconductor (CMOS) components. However, these attacks are expensive, since they require the individual or physical access of the compromised devices.

(b) Non-invasive/ Software Side Channel Attacks

SC attacks, also known as passive non-invasive attacks, exploit the directly accessible interface of the cryptographic devices. However, these attacks do not leave behind any evidence because the cryptographic device is not permanently modified. These attacks involve playing with the clock signal and voltage, which exploit the physical characteristics of computation (e.g., the unintentional leakage) such as execution time and the power consumed to run a process. The device's computation process can be observed or manipulated by local non-invasive attacks. For instance, the fluctuation in the current in a power analysis attack, consumed by the devices can be measured with the high accuracy, and by correlating the measurements obtained with the computations of the underline hardware the value of cryptographic keys can be extracted. These attacks are dangerous as the owner of the compromised device is often unaware that the secret key has been stolen.

Non-invasive attacks are further divided into power analysis and fault analysis SC attacks. Power analysis is further categorized into Simple Power Analysis (SPA), Differential Power Analysis (DPA), and Fault Analysis (FA) attacks. In SPA attacks, the attackers try to leak information and the encryption key by observing the power consumption of the device (Mangard 2002). While in a DPA (Kocher, Jaffe et al. 1999), instead of looking for a direct relation between the secret data and the power consumption, the attackers try to check the variance in power consumption over many iterations of the algorithm. The power consumption of a unit is generally used to observe the internal execution while an encryption operation is being performed. SPA and DPA are the non-invasive SC attacks that allow the attackers to attack and harm the tamper-resistance device by analyzing their power consumption (Countermeasures). DPA is a most dangerous security threat for all the electronic devices which use cryptography for performing encryption. The countermeasure for SPA and DPA attacks include hardware, software, and protocol prevention solution that secure tamper-resistance electronic devices from SC attacks. However, FA attacks generate fault in a system and investigate the encryption algorithm to extract secret keys by using this faults (Aumüller, Bier et al. 2002). Fault analysis attacks can be further categorized into conventional and differential fault analysis. A conventional FA attack (Li, Sakiyama et al. 2010) aims to retrieve secret data by analyzing the result of faulty encryptions. While in differential FA attacks (Biham and Shamir 1997), the attacker encrypts the same plaintext twice, once with and once without an induced error. The attacker then tries to identify the round in which the fault occurred by looking at the difference between the two obtained ciphertexts.

(c) Semi-invasive SC Attacks

Compared with the non-invasive attacks, semi-invasive attacks are very difficult to implement as they involve the opening or depackaging of the chip. However, these attacks have implemented without the requirement of an expensive equipment in comparison to invasive attacks. Furthermore, the implementation of these attacks requires only a short time. They can be achieved by depackaging the chip to get direct access to the chip surface but without harming the chip passivation layer or making any illegal electrical entry other than with the authorized interface. These attacks could be accomplished using UV light, X-rays, electromagnetic fields, laser, and another source of ionizing radiation. For example, the attacker can ionize a transistor by using a laser beam and thus changing the flip-flop's state that holds the device's protection state (Aciiçmez, Koç et al. 2007).

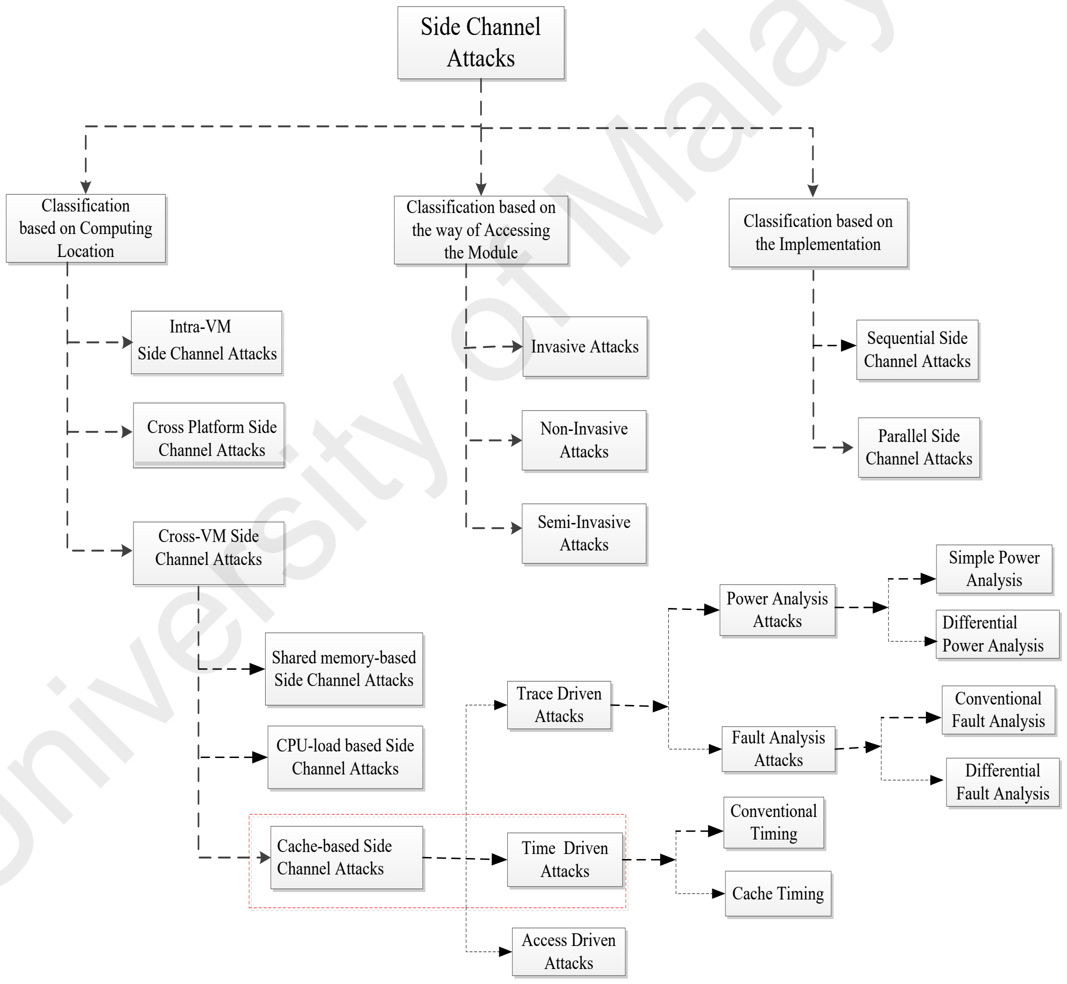

Side Channel Attacks
Classification based on Computing Location
Classification based on the way of Accessing the Module
Classification based on the Implementation
Intra-VM Side Channel Attacks
Cross Platform Side Channel Attacks
Cross-VM Side Channel Attacks
Invasive Attacks
Non-Invasive Attacks
Semi-Invasive Attacks
Sequential Side Channel Attacks
Parallel Side Channel Attacks
Shared memory-based Side Channel Attacks
CPU-load based Side Channel Attacks
Cache-based Side Channel Attacks
Trace Driven Attacks
Time Driven Attacks
Access Driven Attacks
Power Analysis Attacks
Fault Analysis Attacks
Simple Power Analysis
Differential Power Analysis
Conventional Fault Analysis
Differential Fault Analysis
Conventional Timing
Cache Timing

Figure 2.4: Taxonomy of Side Channel Attacks

Existing literature shows that these three attacks are local and can be easily prevented, however, remote attacks are more challenging to prevent since they are not dependent on the quality of the crypto processor hardware (Smith 2003). Having discussed the vulnerability issues concerning CC, this current research focuses mainly on the defensive mechanism of cache-based SC attacks (cross-VM attacks) in CC. Our proposed HBP-DCP solution is based on the cache-based time-driven SC attacks. The detailed classification of these attacks is given in Figure 2.4.

2.2 Cross-VM Cache-based Side Channel Attacks

SC attacks existed in the past in multilevel systems including database, OS, and networking (Zander, Armitage et al. 2007), however, the co-residency feature of the CC makes cross-VM cache-based SC attacks more effective in this paradigm (Ristenpart, Tromer et al. 2009). It was very difficult to gain physical access to the system in the past, but with shared resources, in the cloud, physical access can be easily accomplished (Chang and Ramachandran 2016). Cross-VM Cache attacks are purely software based, and they extract the full encryption key of the well-known cryptographic algorithms including RSA, AES without any direct or physical interaction with the cryptographic devices (Zhang, Juels et al. 2012). These attacks are deployed very easily and are efficient as they require a short time to break the well-secured systems. Moreover, these attacks use the spying process to collect information about the accessed cache line for extracting the cryptographic key from Linux encrypted partition. Irazoqui et al. (Irazoqui G 2014) conducted the Bernstein's correlation attack in a virtual environment for the first time to show the implementation of cross-VM SC attacks on KVM, VMware, and Xen.

Similarly, Irazoqui et al. (Irazoqui, Inci et al. 2014) established the Flush + Reload cache-based SC attack across VM executing on a VMware hypervisor. They used a memory deduplication technique known as transparent page sharing for launching the SC attack and recovered the AES key in a very short time from the AES implemented in OpenSSL 1.0.1. One of the main features of cache-based SC attacks is the memory deduplication, which has been explained in the earlier section of this thesis. To this extent, CPU Cache is seen as the attackers' most targeted device in the cloud due to the device's high shared interaction between processes, cores, and VM. This interaction leads to crosstalk between processes and VM, thus leaking the most fine-grain information of computation (power, time) to attackers. Although the virtual memory mechanism secures the stored data in the cache memory from SC attacks by, the “metadata” have the most fine-grain information about the cache information and pattern of the memory access (i.e. the addresses of which are being accessed) is not fully protected. Several approaches for measurement that exploit crosstalk between processes have been identified. One approach is to measure the effect of the cache on the encryption algorithm (requiring accurate timings). Another approach analyzes the effect of the encryption algorithm on the cache status. Despite using the partitioning method, which includes sandboxing and memory protection, these attacks allow an unauthorized program (attacker) to attack the victim processes on the same physical device running in parallel. These methods provide the logical isolation but are unable to secure communication between processes that are physically located on the same domain.

In comparison to hardware (physical) SC attacks, software cross-VM cache-based SC attacks have a more serious impact on the systems and clients or cloud users. Since almost all modern microprocessors contain cache, physical access to a system very easy in the cloud, making the software attacks much easier to accomplish, and are also effective on disparate platforms (Bernstein 2005, Percival 2005, Osvik, Shamir et al. 2006). Consequently, this makes cross-VM cache-based SC attacks as a new weapon for the adversaries and a much-discussed topic in the literature. These attacks can be achieved without exploiting bus and memory probing since it is not must for software cache-based SC attacks to gain physical access. The attacker can exploit the system by acting like a legitimate user performing a normal operation without the requirement to find the system flaws to perform unauthorized operations. The attacker and the victim are two processes that do not have the same address space, therefore this always makes the attacker able to leak confidential information about the victim's activity.

The cross-VM cache-based SC attacks are also called remote attacks involving faraway observation of the normal input and output data of the device. Timing observation, cryptanalysis, analysis of the protocol, and SC attacks on the programming interfaces of applications are examples of remote timing attacks. The cross-VM cache based SC attacks are further categorized into time-, access-, and trace-driven attacks, which are explained in detail in the following section. Timing analysis attacks conducted on shared caches memory have been widely studied in the cryptanalysis of cryptographic algorithm, e.g. (Bonneau and Mironov 2006, Acıiçmez, Schindler et al. 2007, Intel 2007, Brumley and Hakala 2009, Tromer, Osvik et al. 2010, ARM 2012) in a non-virtualized environment. In this research work, we elaborate the cross-VM cache-based SC attacks in detail. To the best of our knowledge, no prior works have conducted a survey on cross-VM cache-based SC attacks and countermeasure. Therefore, the main contribution of this chapter is to thoroughly study the literature on cross-VM cache-based SC attacks and proposed countermeasure to these attacks.

2.3 Causes of the Cross-VM Cache-based Side Channel Attacks

Cross-VM attacks are conducted between the two VMs (victim and Attacker) in a virtualized environment. In this section, the main causes in the memory management system are described that allow the information leakage in virtualized environment. Although sandboxing provides logical isolation across guest VMs, this isolation is considered to be imperfect and the attacks exploit the memory deduplication and huge pages to leak the secret information across VM boundaries. Since the cache is the most interactive device between VM, it often becomes the targeted device for SC attacks in the modern computers. Therefore, the source of information leakage by using cache in X86 computer is shown in the Figure 2.5.

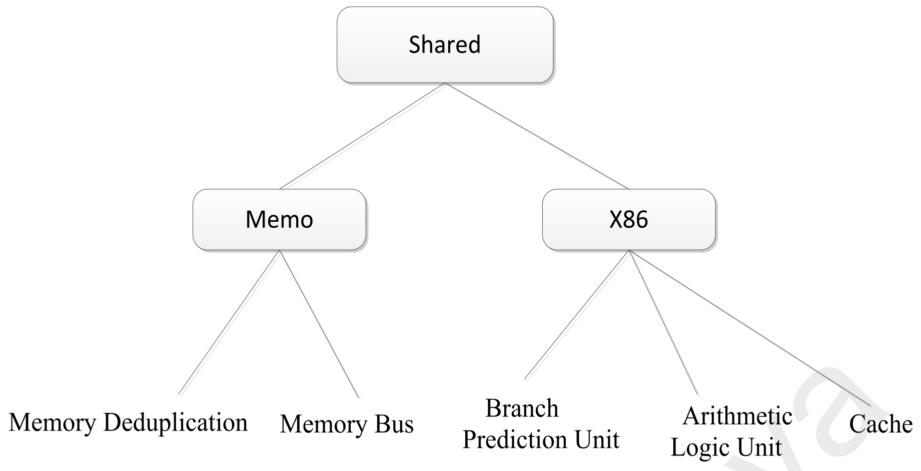


Figure 2.5: Sources of Information Leakage on Shared Hardware

2.3.1 Last Level Cache Memory

The cache memory is located between RAM and CPU cores to remove the delay added by the accessing of the data. The main objective of the cache memory is to decrease the required time for accessing data from the main memory. Modern CPU have more than one cache memory to improve the computation performance by improving the efficiency of cache access. A unit of a cache memory is called line, which consists of a fixed number of bytes. There are a fixed number of cache lines in each multiple cache sets and these cache lines in a cache set is called an associative. The cache is divided into L1, L2, and L3 level. The associative of L1 and L2 cache memory are 8-way associative while the L3 cache memory is a 12-way associative (Handy 1998).

Cache is classified into inclusive and exclusive on the basis of the design approach. In the inclusive design approach, the data is stored in the L1 cache and is also duplicated in the L2 and L3 cache at the same time. While in the exclusive design, the data is never shared between all the cache levels. In modern Intel processors including Core I5 and Core I7, the L3 or LLC is shared between all CPU cores. The salient characteristic of the LLC is that it is by design an inclusive cache memory. Therefore, the data stored in the L1 and L2 caches is also copied in the LLC. Consequently, in the case of a cache miss in the L1 cache, the data will be checked in L2 in order to decrease the cache miss rate. Furthermore, if the data is flushed or evicted from the LLC, it will automatically be erased from all the other levels of the processor's cache.

Although shared cache has some advantages such as increased utilization of cache space, decreased cache miss rate, faster inter-core communication through shared LLC (L3 and L2), and the elimination of undesired replication of cache lines to reduce aggregate cache footprint. However, the major disadvantage of shared LLC is the uncontrolled contention can occur by allowing CPU-cores to access the shared LLC on a freely basis. Consequently, a scenario can occur where one core can easily access and evict the useful content of LLC (L3and L2) belonging to another core result a high LLC miss rate. This cache miss rate degrades the overall performance of the application and system. Similarly, one core can easily extract the useful data of another core can cause SC attacks.

The cache is divided into cache lines having fixed size of l bytes. A cache line contains the information that can be fetched or written at the time of cache access. When the CPU accessed the data stored in the memory for the first time, it first queries the cache memory for data, if it is in the cache then the required time for fetching the data will be low. This is because the memory line that contains the regained data is loaded into the cache memory. If the same data is retrieved again from the identical memory line, then for the same data access the access time will be minimized and this is called a cache hit. However, if the needed data is not available in the cache then the CPU will fetch the data from the main memory and the required time for fetching the data will be high and this is called cache miss. The CPU fetches the data from the main memory when the cache miss occurs and stores a copy in the cache. Therefore, encryption time for a cryptographic algorithm directly depends on the position of the accessed table, which in turn depends on the internal confidential state of the cipher. The secret key of the encryption algorithm can be extracted by exploiting this timing information. In the case of unavailability of cache lines, the data that is not recently being accessed are removed to create a space for the input lines to cache. Therefore, the eviction of cache lines from the cache memory is based on the not recently accessed cache line policy.

TLB (Translation-Lookaside Buffer) is the fastest hardware cache of virtual to physical address translation is also called address translation cache. Upon each virtual to physical translation, the hardware first checks the TLB cache whether the virtual memory reference is already present in the TLB or not. If present in the TLB then the translation is performed very quickly without consulting with the page table. TLB improve the performance of the system by making virtual to physical translation possible. The hardware can handle the TLB misses entirely by using page table base register that exactly tell the location of the page table in memory. On the TLB miss the hardware check the exact page table and extract the translation and update the translation in the TLB.

Cache hit rate= number of hits/ total number of access

When the TLB cache accesses the memory for the first time this misses always occur, however, spatial locality improve the TLB performance. The elements of the array are tightly couple so always TLB miss occurs only for accessing the first element of the array. The idea behind hardware is to take advantage of locality. The functionality and performance of TLB cache are always dependent on the spatial and temporal locality features of cache. According to temporal locality the data or instruction that are recently been accessed will likely be accessed again in the future (e.g., loop variables). In contrast, with spatial locality, the data and the information in the nearby location of the already accessed will be likely accessed in the future. Consequently, when the data is retrieved from main memory by the processor, the copy of that data with nearby memory data will be put in the cache memory to minimize the future access delay of data. The spatial locality facilitates the CPU by storing the entire bigger block of data along with the data in nearby locations. The execution performance can be improved by storing the entire block of data because the data that is located nearby the originally retrieved values are likely to be retrieved again. With address space identifier (ASID), the TLB is able to hold and differentiate translation from the different process without any confusion during context switching.

How do the attackers work? To exploit the timing information, the attacker chooses a cache sized memory buffer and set the cache to a known state before the victim processes an execution. The Attacker accesses all the lines in the buffer, loading the cache with its data. When the victim executes, the victim replaces some memory in the cache. The attacker then measures the time to access the buffer cache (Liu, Yarom et al. 2015). Access to the cache line is faster than to evict lines. CPU caches are the most targeted hardware devices by adversaries due to the high-rate interactions between processes, shared among VMs or Cores, and have the most fine-grain information about the computing processes. In the past, SC attacks are applied on L1 and L2 caches, and in virtualized environment L2 cache in Core 2 duo system is the most targeted device for cache-based SC attacks (Figure 2.6). Most of the attacker are still using L2 cache for launching cache-based SC attacks (Godfrey and Zulkernine 2014).

However, in modern PC including Core i5 and Core i7, the LLC or L3 is the most targeted device for SC attacks. This is because every core has their own L1 and L2 cache but the L3 cache is shared between every core in modern architecture as shown in Figure 2.7. Consequently, the attackers always target LLC (L3) for SC attacks. Flush and Reload attacks exploit the cache behavior and can be mostly implemented by using LLC. Figure 2.6 shows the architecture of Core i5 processor in which the L2 cache is always shared between cores and VMs. Every cores have their own L1 data cache and instruction cache.

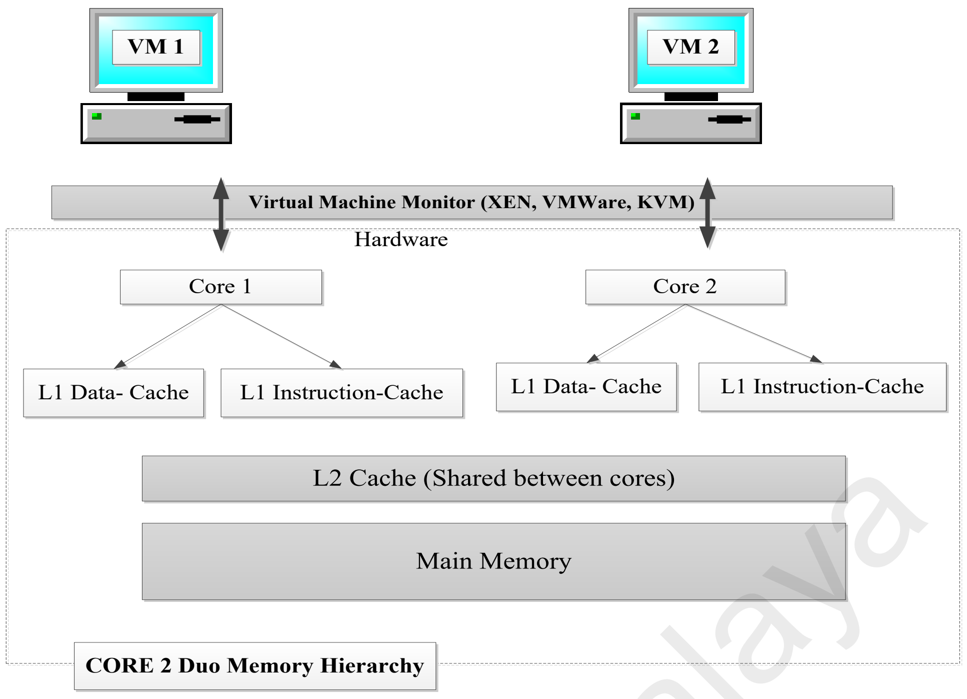


Figure 2.6: Virtual Machine CORE 2 Duo Memory Allocation Hierarchy

In contrast to Core i5, Core i7 process have their L1 and L2 cache but L3 cache is always shared between cores and VMs as shown in Figure 2.7. The access time for accessing information from main memory or from L1 or L2 cache closer to main memory is more than from accessing it from L3 cache closer to the core. Cache-based SC attacks

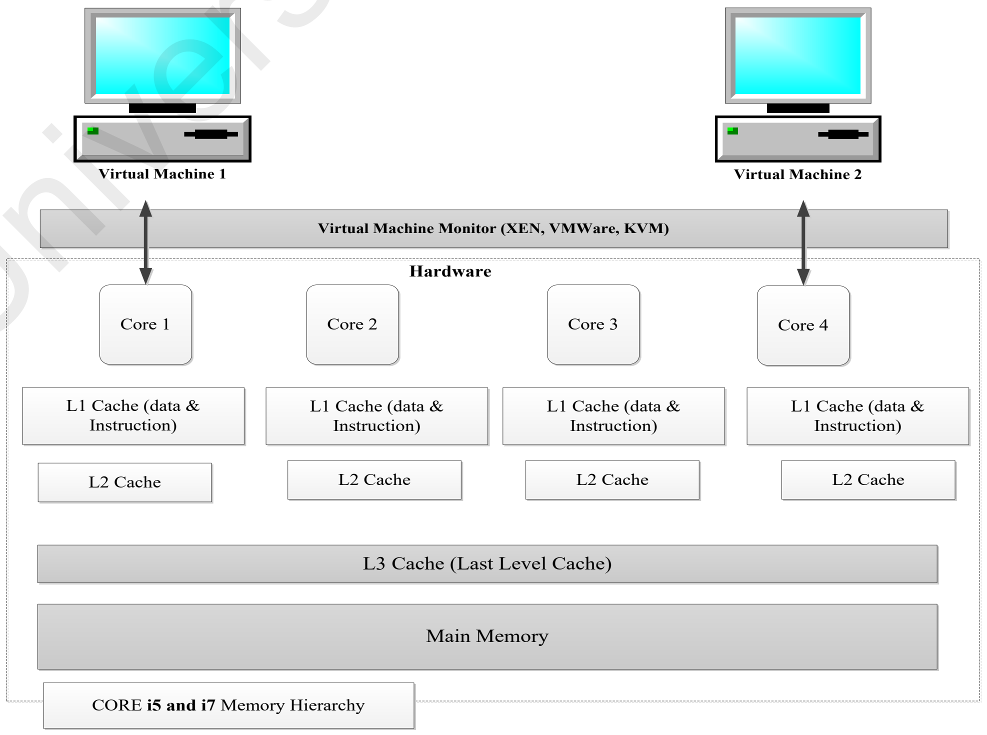

Virtual Machine 1
Virtual Machine 2
Virtual Machine Monitor (XEN, VMWare, KVM)
Hardware
Core 1
Core 2
Core 3
Core 4
L1 Cache (data & Instruction)
L1 Cache (data & Instruction)
L1 Cache (data & Instruction)
L1 Cache (data & Instruction)
L2 Cache
L2 Cache
L2 Cache
L2 Cache
L3 Cache (Last Level Cache)
Main Memory
CORE i5 and i7 Memory Hierarchy

Figure 2.7: Virtual Machine CORE i7 Memory Allocation Hierarchy

exploit this cache access timing difference. There is a need for a prevention mechanism that hides this timing difference from the attacker and which does not need any changes in the software or hardware by the client (Mishra, Pilli et al. 2017).

2.3.2 Memory Deduplication

One of the major causes of SC attacks is content aware sharing or memory deduplication. By using content aware sharing, the same pages are recognized and loaded by the disk location (Miłós, Murray et al. 2009). By merging the identical pages and making a single copy of the redundant data, many VMs are able to run on the host system (e.g., Hypervisor) (Xiao, Xu et al. 2013). This technique improves the memory efficiency by reducing the space and bandwidth requirements for data storage of multiple clients. However, deduplication has a great impact on the security of the system and it opens the door for cache-based SC attacks. The memory deduplication leaks sensitive information due to the deficiency in the Intel x86 processor and the Flush + Reload attack exploits this deficiency to monitor memory lines. The recent statistics (Russell 2010) showed that deduplication is the most impactful storage technology and in the near future, 75% of all backups will apply this. The memory deduplication mechanism, which first appeared in the Linux kernel version 2.6.32, is KSM (Kernel Same-page Merging) (Suzaki, Iijima et al. 2011). KSM is a memory saving feature and has also been suggested for virtualization such as Satori (Miłós, Murray et al. 2009). However, this approach is a big security threat for cryptographic algorithms in virtualized environment (Gullasch, Bangerter et al. 2011). The memory deduplication feature is enabled by default in some hypervisor namely VMware ESXI and Virtual Box. However, recognizing it as a major threat to security, Amazon never enabled this memory deduplication feature on their compute cloud server EC2.

The memory deduplication can be exploited by one of the low noise cache-based SC attacks called flush reload attack. All the current LLC attacks (e.g., flush + reload attack on L3) require deduplication. In these attacks, the target of the two processes is to access the same physical memory location. This means no identical contents are stored in the physical memory since the memory deduplication feature has eliminated the redundant data from the memory allowing the cross-sharing of data between processes. However, this mechanism creates a security vulnerability in CC. The system must protect data shared between two non-cooperating processes. Due to the additional copy operation, the access time to the normal page and de-duplicated page is different. Therefore, in virtualized environment, the attackers can easily get the memory access information from victim VM because the victim and attacker VM are collocated on the same physical machine (Suzaki, Iijima et al. 2011, Suzaki, Iijima et al. 2011). For instance, the attacker can easily detect whether the de-duplicated page exists in the collocated VM or not by requesting the same page from the memory. Although the adversary cannot modify or corrupt the data in the cache, parallel access rights and cross sharing can be exploited to extract secrets from the process executed in VM.

2.3.3 Big Data Deduplication

In this era, as the volume of the data is increasing on daily basis, everyone is thinking about for online storage to move and store data on the cloud side. Since this data is stored in a huge amount, it is therefore needed to remove the redundant data for improving the performance. In order to eliminate the repeated data, data deduplication mechanism is used. Data deduplication is one of the data compression mechanism use for big data which eliminate redundant copies of the data stored in multiple places in the storage of big data. Although this mechanism is used to improve the utilization of big data storage and also minimize the number of packets or bytes to be sent (Yu and Guo 2016). However, this is because of data deduplication that big data storage is vulnerable to SC attacks. Big data deduplication is one of the major cause of SC attacks. The overall cost can be reduced by providing the same services to multiple clients and this can be achieved by deduplication mechanism. Since data is generated from different resources in a big data environment, everyone needs to think about the security of big data in CC. Similarly, the data is shared between different VM in the virtualized environment. If you think about to store a huge amount of data (big data) on the cloud side and to share your data with another VM then you have to care about the security and cost as well. The security of the big data storage is a major issue in CC which demotivates the cloud user and they are not further trusting to move their data to cloud side. The big data has different characteristics and is not equivalent to normal data, thereby security requirement for big data is different.

Data duplication can be categorized into various types including granularity, location, and ownership on the basis of distinct criteria. Based on data granularity, the deduplication is further divided into a file- and bloc-level deduplication. In file-level deduplication, the big data is reduced by removing the redundant file. While in the block-level, the redundant block of data is removed in the non-similar file (Stanek, Sorniotti et al. 2014). According to location, the deduplication is divided into the client- and server-side deduplication. The deduplication of redundant data performed on the client side is called client-side deduplication otherwise target-based (server) deduplication. In the target or server-based mechanism, the server does all the deduplication while the source or client is completely unaware of the deduplication. The server-side deduplication mechanism improves the overall storage but does not have improvement in the bandwidth. While the deduplication on the client side improves both data storage and bandwidth making the system more vulnerable to SC attacks. By using deduplication the SC attackers can easily determine the big data storage. To store big data on the server side is more secure as compared the big data storage on the client side. Based on the data ownership, the deduplication mechanism is further classified into single-user and cross-user deduplication. In a cross-user deduplication mechanism, the data interchange between two users. The storage and bandwidth can be improved by using cross-user deduplication. Although cross-user deduplication is more effective in CC, however, it gives a chance to the attacker to leak the confidential information (Harnik, Pinkas et al. 2010). In (Wang, Cao et al. 2016), the author proposed the attribute-based encryption which secures the big data storage from the SC attacks and is able to provide security to the big data.

2.3.4 Huge Pages

Although cloud service providers and the virtualization company have disabled the memory deduplication feature for the mitigation of cache-based SC attacks, another security risk for the virtualized and non-virtualized environment in the form of huge pages has come into existence. Huge pages are another root cause that attackers use for launching SC attacks. The attackers gain the knowledge about the physical addresses of the memory by using large size pages. The attacker takes the opportunity of the translation of the virtual to physical addresses. All the processes have no direct accessed to the physical address instead they are using the virtual addresses. The memory is divided into continuous fix block called memory pages. The virtual memory is used to load these memories when they are not present in the main memory. When some pages needed by a process is not retrieved from the main memory then page fault occur and the required pages are loaded from other storage. Therefore, before access to memory, a translation stage between virtual to physical address is needed. Modern computer architectures consist of Translation Lookaside Buffer (TLB) for the purpose to avoid the latency of virtual to physical address translation.

The TLB behaves like a small cache is first observed before memory management unit. If the memory is divided by increasing the size of the page into fewer pages then this can be used to avoid TLB misses (Weisberg and Wiseman 2009, performance Feb 2016). The TLB misses will be reduced than 4KB pages because the translation between virtual and physical addresses have significantly been reduced. Due to this reason, state-of-the-art processors use the huge size pages of 1MB. The usage of huge pages is very effective in virtualized environment where different VM use the same hardware resources on the same physical machine. The huge pages by default are enabled in all VMM or hypervisor including KVM, VMware, and XEN. Therefore, unlike Yarom's Flush + Reload (Yarom and Falkner 2014) that only works when the deduplication is enabled, many other attacks can be launched by exploiting huge pages to extract the secret information in virtualized and non-virtualized environment (Irazoqui, Eisenbarth et al. 2015). Liu et al. (Liu, Yarom et al. 2015) conducted SC attacks on the L3 cache by exploiting huge memory pages. They extracted the cryptographic key from the ElGamal encryption algorithm. Picking up from Liu et al (2015), Inci et al.(Inci, Gulmezoglu et al. 2015) conducted prime + probe attacks in the cloud environment. They used the huge pages to extract the information about co-location and also the cryptographic key of ElGamal algorithm.

2.4 Types of Cross-VM Cache-based Side Channel Attacks

Cross-VM cache-based attacks are categorized into time-, trace-, and access-driven. The detail of each category is given in the following section.

2.4.1 Time driven Side Channel Attacks

In cryptography, an attack in which the attacker observes the execution time of the cryptographic algorithm and use this information to compromise a cryptosystem is called time-driven attacks. In addition, the attackers try to extract the cryptographic key by learning the system's sensitive information and by analyzing the computation's time of processes. It is an extremely powerful in CC because of the memory deduplication and logical isolation. In CC, the sandboxing provides only the logical isolation, which is not equal to the physical isolation. Therefore, the attacker in one VM can easily measure the computation time of any encryption algorithm by accessing the cache to determine the encryption key in use on the victim VM on the underlying hardware. The two co-resident

VM can easily access each other's execution process and cache if the Transparent Page Sharing between the two VMs is enabled. The victim and the attacker VM could be on the same physical machine and could be remote. The attacks associated with the class of time-driven attacks are explained in (Osvik, Shamir et al. 2006, Tromer, Osvik et al. 2010) and the detailed example is given in Table 2.2.

In these attacks, the timing difference can be exploited by associating the cache to a prior state known to a victim cryptographic operation. In addition, the following two methods are used to extract information from the victim's operation. The first method is based on the time measurement which it takes for the victim to execute the cryptographic operation. As this time is related to the cache's state, when the victim executes the operation, the attacker can evaluate the accessed cache lines by the victim and extract the secret information (Bernstein 2005, Acıiçmez, Schindler et al. 2007). In the second method, the attacker's time for accessing the data after the victim's operation is measured (Aciiçmez 2007, Brumley and Hakala 2009, Acıiçmez, Brumley et al. 2010). The changes in this time are dependent on the changes in the cache state before and after the victim operation. In the literature, this problem has gained a lot of attention. Time-driven attacks are further categorized into active and passive attacks. In a passive time-driven attack, the total computation time of the victim's process is measured by the attacker, in contrast, the attacker observes the state of the cache in active time-driven attacks.

The main challenges in the measurements of timings in the time-driven attacks are the increased level of noise (such as network latency and increased access time) and unpredictability of correlation of timings. Many cryptographic algorithms lack a proper defensive mechanism for cache based timing attacks. Therefore, the timing attacks can easily be implemented on any cryptosystem. For instance, libgcrypt (used in GNUTLS and GPG) and Cryptlib are not secure from the timing attacks. A defensive mechanism against the timing attacks is present in the OpenSSL 0.9.7 as an option. However, this option is not enabled in common applications such as the Apache SSL module and mod SSL and therefore they are vulnerable to time-driven attacks. The following examples show that cryptosystems are vulnerable to time-driven SC attacks.

Tsunoo et al. (Liu, Ge et al. 2016) implemented the initial practical results for cache-based time-driven attacks and the authors were able to break the Data Encryption Standard (DES) in 90% of their attempts. They found that the internal table lookup collision in the cryptographic algorithms is the main cause for time-driven attacks. Various attacks associated with the class of timing attacks on AES were explained in the subsequent papers (Bernstein 2005, Osvik, Shamir et al. 2006, Acıiçmez, Schindler et al. 2007, Tromer, Osvik et al. 2010). In some of them, the first or the last round of AES algorithm is required. These attacks execute the overall execution time of encryption algorithms. The detail of these attacks is given in Table 2. Osvik et al. (Osvik, Shamir et al. 2006) introduced the time-driven attack on the second round of AES algorithm for analyzing the timings information. Similarly, Weiß et al. (Weiß, Heinz et al. 2012) described the most relevant class of time-driven SC attack. In this work, they implemented a time-driven SC attack against an embedded uniprocessor in virtualized environment. In cache collision attack against AES (Bonneau and Mironov 2006), the authors conducted cache-based timing attacks in which they extracted the secret key by exploiting the cache collision due to the internal state of the table lookup operation in AES.

2.4.2 Trace–Driven Side Channel Attacks

In these attacks, the attacker's process has the ability to capture a profile of the cache activity during the execution of the cryptographic algorithms. To launch this attack, the attackers need to access the profile in which they observe and extract the profile of the cache activity from other profile content. These attacks (Bertoni, Zaccaria et al. 2005, Lauradoux 2005) are related to the class of trace-driven attacks. The result of this attack produces a cache hit if the victim accesses the cache and cache miss for every access to memory. Therefore, it is very easy to trace the S-box accesses for encryption algorithm including AES and DES during the execution time. As opposed to a time-driven attack, Osvik et al. (Osvik, Shamir et al. 2006) in his research conducted the two trace-driven attacks including Prime + Probe and Evict + Time and their impact on AES algorithm. They further investigated that both techniques can be applied during the attack implementation to recover the encryption key of any cryptosystem.

In an Evict + Time attack, the cache is evicted before the encryption and then the cache access is investigated in term of a cache hit and cache miss. While in the Prime + Probe procedure, the cache is filled prior to encryption and after it has checked which cache line has or has not been accessed. The information can be further used to extract the encryption key. By using these attacks some features of the device are continuously monitored throughout the cryptographic operation, for example, a processor leaks information by analyzing electromagnetic radiation (e.g., (Gandolfi, Mourtel et al. 2001, Quisquater and Samyde 2001) and by the power consumption of the device (e.g., (Kocher, Jaffe et al. 1999)). These attacks became powerful by the ability to continuously monitor the processor computation but the limitation is in physical proximity of the device to the timing and power measurements, an idea which was first introduced by Kocher in the year of 1999 (Kocher, Jaffe et al. 1999). Here we described the continuation of these attacks by measuring the cache access latency.

2.4.3 Access-Driven Side Channel Attacks

The most powerful attack is called an access-driven, in which the attacker tries to investigate which cache line has been observed during the execution of cryptographic algorithms. These different memory accesses are the main threat to cryptographic software since the variations in the computation time provide information about the secret key. These attacks evaluate the cache memory working with a fine-grain information, rather than analyzing the overall computation time of the executable program. In these attacks, the attacker and the victim's programs are executed side by side on the same host machine. The attacker executes a program on the same physical system that is executing the cryptosystem and observes the operation of the shared architectural component to extract confidential information about instruction and data cache.

The usage of a shared architectural components such as the instruction cache (Aciiçmez 2007, Acıiçmez, Brumley et al. 2010), data cache (Percival 2005, Tromer, Osvik et al. 2010) floating-point multiplier (Aciicmez and Seifert 2007), or branch prediction cache (Aciiçmez, Koç et al. 2007) is monitored by the attacker's program to extract secret information about the cryptographic key. To implement this attack, the researchers (Ristenpart, Tromer et al. 2009, Gullasch, Bangerter et al. 2011, Zhang, Juels et al. 2012, Yarom and Falkner 2014) exploit a shared hardware cache between both VMs and filled the cache with their own data. The target victim VM changed the cache by overwriting some of its data, including information about the secret key. When they rewrite their information in the cache, the attacker is able to detect the private encryption key. The most effective and common method for implementing an access-driven SC attack as conducted by Osvik et al. (Osvik, Shamir et al. 2006) is to Prime the cache and then Probe, hence it is called the prime + probe protocol. Similarly, Neve et al, (Neve and Seifert 2006) introduced access-driven SC attacks in which they target the last round of AES. They showed in their research that the whole key can be extracted with a limited set of encryption in a very short time. In addition, in (Neve and Seifert 2006, Gullasch, Bangerter et al. 2011), the authors illustrated that these attacks are successful in a single core, non-virtualized environment by attackers involved in game OS process scheduling. Traditionally, unlawful access into a non-virtualized environment is very difficult, however in virtualized environment, the co-residency of guest VMs make it possible to gain access quite easily.

In a virtualized environment, Ris-tenpart et al (Ristenpart, Tromer et al. 2009) implemented the first access-driven attack on modern Symmetric Multi-Processing (SMP) and multi-core architectures. Their attack is able to provide information about the cache utilization of guest VM, however, is unable to extract the cryptographic secrets. In line with that, Percival, et al (Percival 2005) conducted an access-driven attack on the data cache. In their attack, the shared architectural component is monitored to extract data from the data cache during the execution of the RSA cryptosystem. Similarly, in (Owens and Wang 2011), the authors implemented an access-driven attack to exploit the memory deduplication in the victim VMware ESXI hypervisor for fingerprinting the OS. Zhang et al. (Zhang, Juels et al. 2012) described the cross-VM SC attacks in a virtualized SMP by extracting a cryptographic key from the VM. They perform the successful sequential SC attacks by using the CPU cache and do serious damage to virtualized environment by extracting the cryptographic secret from unwary hosts.

In the existing study, Gullasch et al. (Gullasch, Bangerter et al. 2011) implemented a Flush + Reload SC attack that accessed specific memory lines in the AES memory by utilizing cache behavior. The authors used the processor's clflush instruction to expel the observed memory lines and using this information, they extract the secret key in less than 100 encryptions. While this attack accesses specific memory lines, it generates a false alarm by frequently interrupting the victim process. These authors (Acıiçmez, Brumley et al. 2010, Gullasch, Bangerter et al. 2011) introduced the access-driven attacks called asynchronous, meaning that in the trigger step, the attackers do not require the precise time information of victim operations. The CPU with Simultaneous Multi-Threading (SMT) feature or the OS process schedulers is more vulnerable to these attacks; SMP settings are not vulnerable to these attacks. The class of asynchronous access-driven attack is further extended by Zhang et al. (Zhang, Juels et al. 2012) to VMs running on virtualized SMP systems. Furthermore, by using this attack they have extracted the most fine-grain data from a victim VM across VMs in a virtualized environment. More specifically, by using cache-based timing attacks an ElGamal decryption is recovered from the victim VM. The authors used a hidden Markov model to reduce the errors and cope with noise (e.g., network latency). The significance of this work is that the authors have extracted the fine grain data across VMs for the first time, unlike Ristenpart et al. (Ristenpart, Tromer et al. 2009) who managed to achieve the usage of CPU and recovered keystroke patterns by co-location of VM. Table 2.2 describes the attack based on execution environment and architecture. The execution environment categorizes the attacks whether they are conducted in the virtualized or non-virtualized environment. According to the architecture, Table 2.2 shows whether the attacks have conducted on a single core or multi-core. In addition, it also describes the target of attacks (e.g., Attacks on AES, RSA, and ElGamal or any other encryption algorithm).

Table 2.2: Side channel Attack in Virtualized and Non-Virtualized Environment

Type of Attacks	**Ref**	**Title**	**Description**	**Name of Attacks**	**Method**	**Target Of Attacks**	**Execution Environment**	Architecture
Acces-Driven	(Neve and Seifert 2006)	Advances on access-driven cache attacks on AES	To scrutinize the cache behavior with a finer granularity, rather than evaluating the overall execution time.	Side Channel	Prime + Probe	AES	Non-Virtualized	Single-Core
	(Ristenpart, Tromer et al. 2009)	Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds	To detect the co-residency of virtual machine and then leak the information (such as aggregate cache-usage)	Side Channel	Prime + Probe	Leak information about cache pattern	Virtualized	Multi-Core
	(Acıiçmez, Brumley et al. 2010)	New result on intrusion cache attacks	To monitor the instruction cache for leaking the timing information of cache	Side Channel	Prime + Probe	DSA, Information leakage	Non-Virtualized	Multi-Core
	(Gullasch, Bangerter et al. 2011)	Cache games–bringing access-based cache attacks on AES to practice	To extract the confidential key with a less than 100 encryption by using timing difference of cache access	Side Channel	Flush+Reload	AES	Non-Virtualized	Single-Core
	(Owens and Wang 2011)	Non-interactive OS Finger printing through memory de-duplication technique in virtual machines.	To conduct access-driven attack to exploit the memory deduplication in the victim VMware ESXI hypervisor for fingerprinting the OS	Side Channel	Prime + Probe	Information leakage	Virtualized	Multi-Core
	(Zhang, Juels et al. 2012)	Cross-VM Side Channels and Their Use to Extract Private Keys	To extract ElGamal decryption key by using cache timing attack	Side Channel	Prime + Probe	ElGamal	Virtualized	Multi-Core

Table 2.2: Continue…..

	(Suzaki, Iijima et al. 2011)	Wait a minute! A fast, Cross-VM attack on AES	To extract the AES key by exploiting page deduplication feature in VMware ESXI	Side Channel	Flush+ Reload	AES	Virtualized	Multi-Core
Trace-Driven	(Gandolfi, Mourtel et al. 2001)	Electromagnetic Analysis: Concrete Results	To conduct attack on three different CMOS chips to extract cryptographic key	Side Channel	N/A	RSA, DES	Non-Virtualized (Smart Cards)	Single-core
	(Quisquater and Samyde 2001)	Electromagnetic analysis (ema): Measures and counter-measures for smart cards	To establish simple and differential electromagnetic attack on the implementation of RSA, DES, cryptographic token and SSL accelerator	Side Channel	N/A	DES, RSA	Non-Virtualized (Smart Cards)	Single-Core
	(Bertoni, Zaccaria et al. 2005)	AES power attack based on induced cache miss and countermeasure	To leak information by using a power side channel of MIPS microprocessor	Side Channel	Flush+Reload	AES	Non-Virtualized	Multi-Core
	(Aciiçmez 2007)	Yet another micro architectural attack: Exploiting I-cache	To discover that during execution of RSA encryption the main cause for leakage information is instruction cache likewise the data cache	Side Channel	Prime + Probe	RSA	Non-Virtualized	Single-Core
	(Aciiçmez, Koç et al. 2007)	On the Power of simple branch prediction analysis.	To extract information by analyzing the branch prediction cache	Side Channel	Prime + Probe	Extract key	Non-Virtualized	Single-Core

Table 2.2: Continue…..

	(Yarom and Falkner 2014)	Flush+ reload: a high resolution, low noise, L3 cache side-channel attack	To extract the private encryption keys of RSA from a victim program across cores and across VM hosted by VMware and KVM	Side Channel	Flush+Reload	RSA	Virtualized	Multi-Core
Time-Driven	(Bernstein 2005)	Cache-timing attacks on AES	To attack the AES algorithm and extract the cryptographic key	Side Channel	Prime + Probe	AES	Non-Virtualized	Single-Core
	(Percival 2005)	Cache missing for fun and profit	To describe a cache-based SC attack on RSA on processors having simultaneous multithreading.	Side Channel	Prime + Probe	RSA	Non-Virtualized	Multi-Core
	(Brumley and Boneh 2005)	Remote timing attacks are practical	To launch a cache-based timing attack to extract confidential keys from a library used in web server and SSL applications such as OpenSSL-based	Side Channel	Prime + Probe	RSA	Non-Virtualized	Single-Core
	(Wang and Lee 2006)	Covert and Side Channels due to Processor Architecture	To identify two new attacks namely Simultaneous Multithreading and speculation	Covert and Side Channel	Prime + Probe	RSA	Non-Virtualized	Single-Core
	(Osvik, Shamir et al. 2006)	Cache attacks and countermeasures: the case of AES	To describe time-drive side channel attacks which neither require the plaintext or cipher text	Side Channel	Evict + Time	AES	Non-Virtualized	Multi-Core
	(Bonneau and Mironov 2006)	Cache-collision timing attacks against AES,	To extract cryptographic key by using cache-based timing attack	Side Channel	Evict + Time	AES	Non-Virtualized	Multi-Core

Table 2.2: Continue…..

	(Acıiçmez, Schindler et al. 2007)	Cache based remote timing attack on the AES	To establish cache-based timing attack to measure the cache timing for extracting secret key by using statistically infer information	Side Channel	Prime + Probe	AES	Non-Virtualized	Single-Core
	(Weiß, Heinz et al. 2012)	A cache timing attack on AES in virtualization environments	To launch a cache-based time-driven attack against an embedded ARM processor inside L4Re virtual machine	Side Channel	Prime + Probe	AES	Virtualized	Single-Core
	(Irazoqui G 2014)	Fine grain Cross-VM Attacks on Xen and VMware are possible!	To conduct the Bernstein's correlation attack in a virtual environment for the first time to show the implementation of cross-VM SC attacks on KVM, VMware, and Xen.	Side Channel	Prime + Probe	AES	Virtualized	Multi-Core
	(Irazoqui, Eisenbarth et al. 2015)	S $ A: A Shared Cache Attack That Works across Cores and Defies VM Sandboxing--and Its Application to AES	To conduct time-driven cache-based attacks targeting L3 cache by using huge pages	Side channel	Prime + Probe	AES	Virtualized	Multi-Core

2.5 Prevention of Cross-VM Cache-based Side Channel Attacks

Although hypervisor enforces logical isolation to the cloud resources through a mechanism called sandboxing, however as compared to physical isolation this logical isolation has some security implications. For instance, we know that co-locating VMs on the same platform are not physically isolated and can easily leak sensitive information of each other's which give a great opportunity to the attackers to do security interference. Many researchers have shown the applicability of SC attacks to extract this confidential information and have also demonstrated the mitigation of these security interferences in this section. SC attacks use the fundamental characteristics of sensitive computation (e.g., power consumption, execution time, and the electromagnetic field sharing of a processor core with an attacker) to leak this confidential information. Computations unintentionally leak confidential data through either hyper threading or time division.

The main idea of SC attacks is that most cryptographic algorithms have memory access pattern that are data dependent, which can be easily observed by cache miss and hit rate. Most of the existing prevention mechanisms are adhoc and are unable to prevent SC attacks because they are designed to prevent only specific attacks. No general prevention mechanism has been proposed in the past which could prevent all types of attacks as well as cache-based SC attacks (Singh and Chatterjee 2017). The proposed approaches for the prevention mechanisms for the mitigation of cache-based SC are categorized into three types. The first approach is to come up with new cache designs (e.g. (Wang and Lee 2006, Wang and Lee 2007, Zhang, Juels et al. 2011, Irazoqui, Inci et al. 2014, Zhou, Reiter et al. 2016)). In the second approach, Aviram et al. (Kong, Aciiçmez et al. 2009) described that cache-based SC attacks in CC can be mitigated by forcing VM execution to be deterministic, however, further research is still needed for this approach. The third approach is to construct cryptographic algorithm in such a way that it can block the cache-based access timing attacks (e.g., (Domnitser, Jaleel et al. 2012), (Intel 2007)). The existing countermeasure for SC attacks has several limitations including high overhead, the requirement to change the hardware and software, application-specific, or inappropriate to mitigate the SC attacks (Page 2003).

Similarly, the operation and implementation of Flush + Reload is associated with the combination of four factors (Yarom and Falkner 2014) including the flow of data between sensitive information and memory access patterns, memory deduplication between victim and attacker VM, the analysis and the measurement of high-resolution interval, and the unrestricted use of clflush instruction. Preventing any of these can mitigate SC attack. Techniques such as new cache designs, disallow cache sharing between VM, partitioning the cache among tenants, and forced determinism could potentially mitigate the SC attacks. However, these techniques will not be widely adapted in the future because they require hardware changes (Coppens, Verbauwhede et al. 2009). One technique proposed by Zhang et al. (Zhang, Juels et al. 2011) that enable its guest VMs to detect the exclusive usage of the physical machine. In addition, it verifies the success or failure of the cache isolation policies implemented by the service provider. In addition to the data cache, other architectural SC includes the instruction cache (Acıiçmez, Brumley et al. 2010), the shared functional units (Wang and Lee 2006, Aciicmez and Seifert 2007), and the branch target cache (Aciiçmez, Koç et al. 2007, Zhang and Reiter 2013), all of which have been exploited in the cryptosystem. The countermeasures for these attacks are divided into hardware-based and software-based. Table 2.3 shows that existing solution for SC attacks requires to changing either hardware or software consequently affect the overall performance in term of load testing and cache utilization.

Table 2.3: Required Modification in the Existing Solutions of Cache-based Side Channel Attacks

Implemented Solution Types	Source	Hardware	Performance degradation
Obscure Cache-Data Correlation	Y	N	N
Delay Timing Information	Y	N	N
Normalize Cache State	N	N	Y
Custom Hardware	N	Y	N
Disable Cache	N	N	Y
Cache Warming	N	N	Y
Cache Partitioning	N	N	Y
Cache Flushing	N	N	Y

Y for need to change, N mean does not need to change

2.5.1 Existing Countermeasures

The existing countermeasures for cache-based SC attacks are divided into hardware-based and software-based countermeasure which are explained in the following section in detail.

2.5.1.1 Hardware-based Countermeasure

The literature shows that cache-based SC attacks are mostly prevented by a hardware-based solution that mainly focuses on altering the replacement policies of cache (Kim, Chandra et al. 2004, Percival 2005). Although some of these solutions are effective, the existing processors are unable to employ this because they need a special support of hardware. For instance, Osvik et al. (Osvik, Shamir et al. 2006) proposed hardware-based solutions to disable the cache or to utilize an individual cache for concurrent threads. Few of the new proposed solutions include eviction strategies which minimize the eviction of data of one thread used by another one (Percival 2005). In (Page 2003), the partitioned cache initially designed for multimedia applications is exploited to block cache-based SC attacks. The Instruction Set Architecture (ISA) is altered by adding new instructions to make the cache a prominent part of the existing architecture that can define a partition, cache line size, as well as other parameters. However, the authors also claimed that the costs of the cache design and its computation can be high. As compared to software-based solution, a hardware based solution cannot provide an efficient countermeasure and takes more time to develop to block these attacks. As a result, this problem can be solved by using an efficient countermeasure such as software-based solution.

Over the last decade, when the attention in the literature was given to SC attacks (Inci, Gulmezoglu et al. 2015) many proposals for the mitigation of SC attacks (Page 2003, Osvik, Shamir et al. 2006) were put forward. These proposed prevention mechanisms were typically categorized into eight types as shown in Table 3. Various methods to implement these prevention mechanisms, such as changing the usage of the cache to cryptosystem and altering the hardware channel. In addition, the hardware-based solution proposed (Kong, Aciiçmez et al. 2009, Domnitser, Jaleel et al. 2012), included coming up with new designs for a shared cache. However, these existing prevention mechanisms would require either modifying the source code, altering the cryptographic algorithm, changing the hardware (e.g., changing cache design), or creating high computation cost in term of high overhead. Designing new caches will take longer time and during this time the SC attacks do a lot of damage. Therefore, there is a need for software-based prevention mechanisms for the quick mitigation of SC attacks.

2.5.1.2 Software-based Solutions

Most of the existing prevention mechanisms for cache-based SC attacks are software-based and are associated to a specific cryptosystem. The basic phenomenon of this prevention mechanism is to edit the software in a new method that the SC attacks cannot be established. Such as, to prevent SC attacks on AES, many types of mechanisms have been proposed, such as 1) The AES tables must be loaded into the cache prior to executing an encryption so that all accesses to AES create cache hit and hence have constant encryption time, 2) During the AES execution, only mathematical operations should be used instead of table lookups. For instance, Brickell et al. (Brickell, Graunke et al. 2006) proposed a software solution, which mitigates the SC attacks by changing the implementation of cryptographic algorithms including RSA and AES. Similarly, the researchers in (Coppens, Verbauwhede et al. 2009, Aviram, Hu et al. 2010, Shi, Song et al. 2011, Zhang and Reiter 2013, Godfrey and Zulkernine 2014) proposed software-based countermeasure for the quick mitigation of SC attacks. Over the last decade, hardware-based solutions have been used for the mitigation of SC attacks, however, recent security activities motivate the implementation of software-based prevention mechanisms by improving the software isolation properties.

Table 2.4: Countermeasures for Cross VM Cache-based Side Channel Attacks

Ref	Title	Description	Attack Types	Implementation Type	Used Method	Limitation
(Aumüller, Bier et al. 2002)	STEALTHMEM: System-Level Protection Against Cache-Based Side Channel Attacks in the Cloud	To propose system level design in which hypervisor or OS give an individual access to a particular sectioned portion of the cache to each VM.	Trace-driven Time-driven	Software-based	- Locking page - Page partitioning	Require user interaction for client side modification which does not comply to the cloud model
(Page 2005)	Partitioned Cache Architecture as a Side-Channel Defense Mechanism	To propose a cache partitioning method against SC attacks that use data cache and access to SBOX through this cache	Access-driven	Hardware-based	- Partitioning Cache	- High overhead - Require hardware modification
(Brickell, Graunke et al. 2006)	Software mitigations to hedge AES against cache-based software side channel vulnerabilities	To secure encryption algorithms (RSA and AES), they proposed new implementation of AES and RSA	Access-driven	Software-based	- Compact S-Box table - Frequently randomized Table - Pre-loading of relevant cache line	- Performance degradation - Require software modification
(Osvik, Shamir et al. 2006)	Cache attacks and countermeasures: The case of AES	To describe an active timing SC attack and then prevent this attack by disabling the cache or use separate cache for simultaneous thread	Access-driven	Hardware-based	- Disabling cache	- Need hardware change
(Wang and Lee 2006)	Covert and Side Channel due to processor architecture	To propose new cache design for mitigation of attacks	NA	Hardware-based	- Selective cache partitioning - Random permutation cache	- Require hardware change - High Overhead - Application specific
(Wang and Lee 2007)	New cache designs for thwarting software cache-based side channel attacks.	To propose hardware based mitigation technique by designing new cache or by dividing the existing cache to hide cache access pattern	Time-driven	Hardware-based	- Locking cache line - Cache partitioning	- Require hardware modification - Performance Degradation

Table 2.4: Continue….

(Intel 2007)	Faster and timing-attack resistant AES-GCM	To construct a constant time cryptographic AES implementation that mitigates cache-based timing attacks	Time-driven	Software-based	- AES implementation	- Require software changes - Runtime time overhead
(Kong, Aciiçmez et al. 2009)	Determinating timing channels in compute clouds	To mitigate timing side channels in virtualized environment by forcing the execution of VM to be deterministic	Time-driven	Software-based	- deterministic execution	Need fine grain timing information
(Zhang, Juels et al. 2011)	Non deterministic caches: a simple and effective defense against side channel attacks	To introduce the cache decay approach which controls cache and randomly select the interval of the cache to create non-deterministic behavior of the cache.	Access-driven	Hardware-based	- Cache decay approach	- Require hardware modification - High overhead
(Shi, Song et al. 2011)	Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring.	To partitioned the cache and reserved a small portion of the cache for each VM and core by using page coloring technique	Access-driven	Hardware-based	- Partitioning of cache - Page coloring	Require the software changes
(Domnitser, Jaleel et al. 2012)	A fast and cache-timing resistant implementation of the AES	To construct a new implementation of cryptographic algorithm that resists side-channel attacks	Access-driven	Software-based	- Lookup-based Implementation	- Runtime overhead - Less data memory - Application and Attacks specific
(Kong, Aciicmez et al. 2013)	Architecting against software cache-based side-channel attacks	To propose a prevention mechanism that hides the cache access by different cores	Access-driven	Hardware-Software integrated	- Preloading - Informing Load - Software Permutation scheme	Require Code changes Performance degradation
(Irazoqui, Inci et al. 2014)	Deconstructing New Cache Designs for Thwarting Software Cache Based Side Channel Attacks	To propose new cache design namely Random permutation cache and Partition lock cache	Time-driven	Hardware-based	- Cache partitioning	- Unable to prevent the attack which built either on cache collision or cache sharing - Need hardware modification

Table 2.4: Continue….

(Godfrey and Zulkernine 2014)	Preventing Cache-Based Side-Channel Attacks in a Cloud Environment	To implement a server side solution which complies Cloud model and does not need any changes in the software or underlying hardware	Access-driven	Hypervisor-based	- Cache flushing - Static Cache partitioning	- High Overhead - Misuse Cache Utilization
(Crane, Homescu et al. 2015)	Thwarting Cache Side-Channel Attacks Through Dynamic Software Diversity	To propose a solution which use dynamic software diversification to change the observable execution features while preserving semantic of program and just changing the replica at the machine level instruction	Access-driven Time-driven	Software-based	- Diversifying Transformation - Inserting random memory load	Require the changes in cryptographic algorithms
(Zhou, Reiter et al. 2016)	A Software Approach to Defeating Side Channels in Last-Level Cache	To proposed CacheBar approach which automatically detects the concurrent access to shared pages and prevents them from evicting memory contents	Access-driven	Software-based	- Copy on access for physical pages - Cache ability management	Running Overhead
(Liu, Ge et al. 2016)	CATalyst: Defeating Last-Level Cache Side Channel Attacks in Cloud Computing	To propose a solution that protects the square-and-multiply algorithm in GnuPG 1.4.13 by dividing the cache into secure and non-secure partition	Access-driven	Hardware-based	Cache partitioning	Dependent on intel cache specific design

2.5.2 Proposed Countermeasures

In the following sections, we discuss several prevention mechanisms that prevent the exploitability of the shared level cache and SC attacks.

2.5.2.1 Disable Huge Size Pages

The main cause of SC attacks is the utilization of huge pages due to which attacks take benefit of the additional memory address. Thus the attack by using huge pages can be conducted in both virtualized and non-virtualized environment since in Linux and in the CC, huge pages are enabled in advance in the hypervisor. In particular, Irazoqui et al. (Irazoqui, Eisenbarth et al. 2015) established SC attacks by using huge pages. These attacks can be prevented by not allowing the guest VM to use huge size pages. The VMM or hypervisor is responsible for making decisions about huge size pages, based on precise parameters such as the size or memory space resources that are needed by the programming code.

2.5.2.2 Cache Partition Using Cache Coloring

Cache coloring is a software-based approach which is used for mapping memory pages to cache lines and for the purpose of a cache hit optimization. The author in (Taylor, Davies et al. 1990) introduced this as an OS performance optimization technique to improve the performance between the physical and virtual memory. This technique is designed to ensure that accesses to contiguous pages in virtual memory make the best use of the processor cache. For instance, two instructions or two VMs that are consecutive in the memory can evict one another data in the cache. Cache coloring solves this problem by mapping the two consecutive memory addresses into non-consecutive locations of the cache. Furthermore, the author used the cache coloring approach for the prevention of cache-based SC attacks (Shi, Song et al. 2011) by dividing the cache into the various portion. The partition of cache for each individual VMs is always implemented using cache coloring approach which is used in the earlier research for the cache performance improvement (Jin, Chen et al. 2009). There is two type of cache partition mechanism namely static and dynamic cache partition. Static partition degrades the overall performance of the system by decreasing the usable portion of the cache for each individual VMs. In contrast dynamic partition improve the usable part of cache for each VMs by diving the cache on the fly according to VMs requirement. Moreover, this thesis conducted the dynamic cache partition approach for the prevention of cache-based SC attacks as well as improve the cache usage for the various VMs.

2.5.2.3 Private LLC Cache Slices

The SC attacks can be prevented by making the cache slices per private VM, same as to the prevention mechanism presented in (Wang and Lee 2007). Author's means that two co-located VMs are unable to use the identical cache slice at the same period of time or a guest VM is not allowed to use it when it is being used by another co-located VM. Therefore, the attacker cannot reach the victim's cache slice and may not able to decide on the usage of memory lines by the attacker. However, this prevention mechanism requires a change in the cache structure and a reduction in the length of the cache slices for a VM. The creation of multiple Guest VMs on the multiple cache slices can also be restricted.

2.5.2.4 Controlling Clflush Instruction

In X86 architecture the deficiency of authorization check for the clflush instruction is the major cause for the SC attacks (Yarom and Falkner 2014). These attacks can be mitigated by controlling or limiting the power of the clflush instruction. Clflush instruction is used to enforce the memory coherence in the devices that do not support memory coherence (Intel 2007). In addition, it also improves the efficiency of the program by controlling the usage of cache memory and flushing the lines from the cache that is not in use during the program execution.

2.5.2.5 Preventing Page Sharing

The Flush + Reload SC attacks can be mitigated by preventing the sharing of memory between the victim and the attacker programs. However, this prevention mechanism will degrade the overall performance as well as affect the functionality of the system by opposing the increased page sharing trend in the OS and the hypervisor. The complete prevention of memory sharing would increase the demand of memory for the OS and the hypervisor and therefore it is not a good solution. However, this solution can be adapted to prevent the sharing of the personal credential by changing the program loader code. Furthermore, there is another possible solution for preventing a Flush + Reload attack is to disable deduplication, which is disabled by default in the XEN hypervisor only (Suzaki, Iijima et al. 2011, Suzaki, Iijima et al. 2011).

2.5.2.6 Prefetching Cache Memory

The prefetching of lookup tables or T tables of any encryption algorithm into the cache before execution of the attacker program blocks the SC attacks. The attacker will never analyze the difference in the memory access time because all the data will be loaded into the cache prior to execution of attacker program. However, the prefetching of 4kb size T tables requires more memory and more time. Consequently, it would increase the AES encryption time and overall performance of the system. The assembly version of AES in OpenSSL used the prefetching technique to stop cache memory information leakage because of T table's access.

2.5.2.7 Flushing Cache Memory

Flushing a cache increases the execution time on a modern processor, therefore, a cache must be flushed only when it is needed. It has been disregarded as a prevention mechanism for traditional cache-based SC due to the generation of large amounts of overhead (Osvik, Shamir et al. 2006). Flushing the T tables after the execution of the encryption algorithm (e.g., AES) from the cache memory produces the same result as cache prefetching. The attackers check the access time difference by locating the T tables in the memory. The generated overhead from cache flushing is based on two main elements: the proportion of flushes to context switches and the context switching ratio in the system. These two factors in combination generate the major source of overhead that reduces the system performance. Additional context switches are needed for the increased flushes and more overhead is generated when more flushes are needed. Cache flushing for mitigating the cache-based SC attack in a non-virtualized environment is more expensive (Osvik, Shamir et al. 2006) because in the non-virtualized environment, the high rate cache flushing and overhead are required on the process-level. However, in virtualized environment, the prevention of SC attacks on process-level is not required, but the prevention of SC attacks across VM-level is required. By comparison, in the cloud system, the context switch rate between VMs is comparatively much lesser than the rate between programs in a regular OS because very few VMs are created when compared to processes or programs.

2.5.2.8 Hardware Masking of Addresses

This prevention mechanism is conducted on the hardware level, which applies a mask to the offset field. During the usage of huge pages, this mask to the offset field is applied on the basis of some of the non-set addressing bits in the physical address. Since the offset field is completely hidden from the users and they have no longer control over the offset field. Thereby, the user is unable to lead the particular set, which he desires to target in the LLC memory and is unable to decide whether the victim has used that particular set or not (Irazoqui, Eisenbarth et al. 2015). The detail description of countermeasures is given in Figure 2.8.

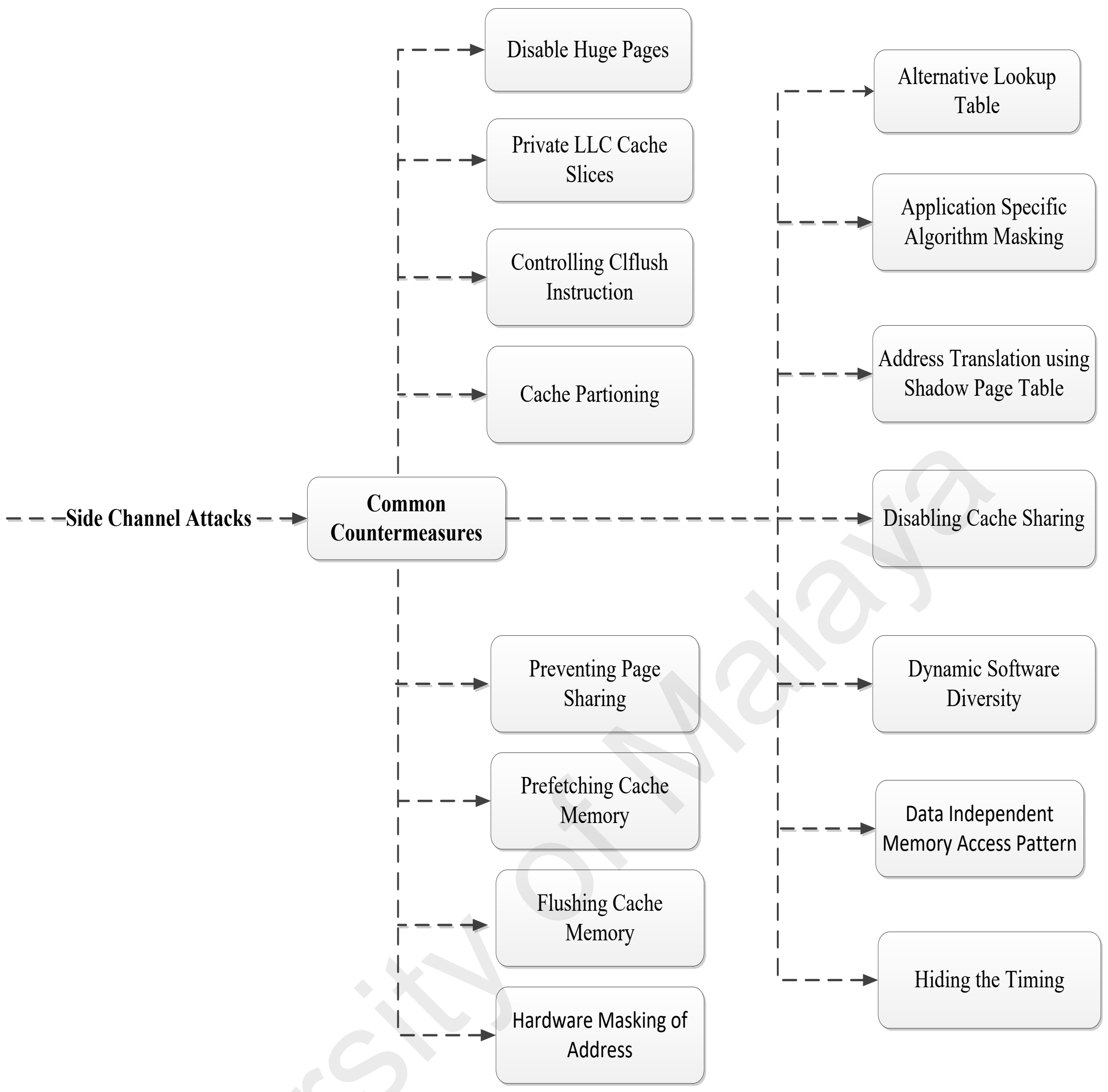


Figure 2.8: Overview of Countermeasure for Side Channel Attacks

2.5.2.9 Address Translation using Shadow Page Table

In many architectures including X86, the CPU uses the shadow page table for address translation. In the case of virtualized environment, the VMMs use the shadow page tables for a virtual to virtual memory translation. For example, the shadow page tables are not only responsible for the translation from VM's virtual memory to the hypervisor (e.g., XEN and VMware) virtual memory but are also responsible for applying a mask based on the non-cache-addressing bits. Therefore, the guest user is unaware of the masking value applied by the guest VM, and he is unable to control the set that the LLC (L3) that contain his data.

2.5.2.10 Dynamic Software Diversity

Dynamic software diversity approach is a protection mechanism against both online and offline cache based SC attacks. The countermeasure for the attacks which rely on the static properties of computation includes diversification of the program representation. On the other hand, SC attacks rely on the dynamic properties of computation including memory access timings, time of execution, and the power consumption. Consequently, for the prevention of SC attacks the diversification will randomize the program's execution instead of representation.

2.6 Research Challenges

Through the detailed study of literature, we analyzed and found that there are some research challenges related to prevention mechanisms. These research challenges are listed in the following subsection.

2.6.1 Efficient Cache Utilization

Since the cache-based SC attacks are implemented using the shared L3 cache. The two VMs using the same cache can evict and also extract each other confidential data. There are prevention mechanisms which divide the shared L3 cache into a partition to restrict the individual VMs to a specific part of the cache. However, this degrades the cache usage for each VMs which affect the overall performance of the system. There must be a mechanism which could divide the cache into a partitions in such a way which does not reduce the cache utilization of individual VM and without affecting the performance of the system.

2.6.2 Server Side Solution Transparent to Guest OS and Client Software

The literature showed that the attention has been given to SC attacks in the non-virtualized environment since the 70s. Many prevention mechanisms including software-based and hardware-based have been proposed to mitigate such attacks (Page 2003, Osvik, Shamir et al. 2006, Tromer, Osvik et al. 2010). These include modification of the hardware functionality (e.g., cache), to disable the hardware channel, modifying the encryption algorithms (e.g., AES and RSA), or breaking the correlation between hardware and program's execution by altering the victim. Unfortunately, implementing any of these defensive mechanisms would either require the cloud users to change their software intended to be executed in the cloud (Aumüller, Bier et al. 2002, Shi, Song et al. 2011), or to customize all the underlying hardware needed to be used in the cloud (e.g., cache) (Page 2003). Both of these solutions contradict the relationship between the cloud model and their users, as they would either restrict the client to change their software or needed the hardware changes, consequently restrict the client to use the cloud. In the virtualized environment, the hardware and software based solution are inappropriate where VM is dynamically added and removed, so the security requirement is also changing. Thereby a server-based prevention solution is required in virtualized environment which is transparent to the clients and the underlying hardware and does not need the changing of client software or hardware and always comply to the user relation with the cloud.

2.6.3 Predicting Cache Contention

If several applications or VMs are accessing an identical part of the cache in parallel as it is in the case of SMT or Chip Multi-Processing (CMP) and if there is not enough cache associativity. Then the two executing applications or VMs displace each other data in order to fill with their own data as well as to extract confidential information. In this cache, the cache contention occurs when the VMs request for the displaced data has to be re-fetched. There must be a cache contention co-aware scheduling where various VMs running in parallel on the shared cache is to be aware of co-aware scheduling. Cache contention co-aware scheduling can reduce the drawback initiated by cache contention by an appropriate mapping of VMs to various CPU Cores and a proper scheduling of VMs.

2.6.4 Determining Optimal Cache Partition Policy at run time with low Overhead

When two or more VMs running on the same physical system, they shared the LLC (L2 or L3) even both are on the different cores. There are existing mechanisms including cache partition, assign per VM cache, assign part of cache to secure algorithm, which divides the cache for individual VM and restrict each VMs cache utilization. However, these cache partitions degrade the cache usage for an individual user and consequently degrade the overall performance of the system. Therefore there must be a mechanism which monitors and determine the required cache of each VM and optimally partitions the cache accordingly.

2.6.5 Improving the Xen Credit Scheduler

Some attackers perform the attacks based on the core-private-cache (e.g., L1 and L2) by exploiting the Xen credit scheduler to extract the fine-grained information of the same cores for two different VMs. There must be a mechanism to improve the functionality of Xen scheduler to restrict the L1 (data & instruction cache), L2, and L3 cache usage in order to not interfere with each other data.

2.6.6 Hiding Memory Access Pattern

The cryptographic algorithms have data-dependent memory access pattern, which can be easily extracted by the attackers by observing the associated hit and miss rate of cache memory. During encryption and decryption cache attacks depend on certain statistics to leak the confidential information in the form of cryptographic key. The information leakage is due to the low-level detail provided by the CPU namely the structure of the cache memory, specifically forms a shared LLC which all the VMs compete for and thus is affected by each VM. For instance, the attacker VM takes benefit from this shared resources. Although virtual memory mechanism protects each VM data is cache, however, the metadata about cache content and the memory access pattern is not fully protected and is available to all running VMs. Therefore, there must be a mechanism in the OS kernel which hides the memory access pattern of that physical system on which different VM is running.

2.6.7 Cache-Aware Scheduler for Optimum Cache Partition

The scheduler must be cache-aware, that scheduler has the ability to monitor the cache utilization of individual VM and decides the partition accordingly. Since various VMs shared the local cache section, there is a change for cache interference to happen between VMs. In this case, there must be a global coordination amongst schedulers on each core for using cache in a proper manner. The likelihood of cache contention as a result of static and dynamic cache partition can be reduced by proper scheduling of VMs and by sharing the information on page color usage.

2.6.8 Soft Isolation as a Solution

In hard isolation, the hardware is dedicated to every VM, however it degrades the performance and efficiency in term of reducing cache usage. In contrast, soft isolation such as scheduler based prevention mechanism (Varadarajan, Ristenpart et al. 2014) improve the performance and reduce the risk of sharing through better scheduling. Although hard solution is more effective, however it cannot be applied in the existing real processor because it is based on cache replacement policies (Kim, Chandra et al. 2004, Qureshi and Patt 2006). In contrast, soft solution is not based on the hardware replacement policies and it used the page coloring technique to change the source code of the Xen scheduler or OS scheduler and can be very deployed without any additional hardware cost (Tam, Azimi et al. 2007, Shi, Song et al. 2011).

2.7 Discussion

In the virtualized environment, the hardware and software based solution are inappropriate where VM is dynamically added and removed, so the security requirement is also changing. Thereby a hypervisor-based prevention solution is required in virtualized environment which does not need the changing of client software or underlying hardware and always comply to the user relation with the cloud. Our research work in this thesis addresses the software attacks and does not consider the physical attacks such as bus probing and the analysis of power. In addition, our focus is on software solutions instead of the hardware solutions. In (Godfrey and Zulkernine 2014) the author referred to the cloud relation with the user and underlying hardware as the cloud model. According to the cloud model the prevention mechanism, which does not need hardware or software changes is implemented for the prevention of SC attacks. In order to implement any defensive mechanism, the two key points are highlighted that have become commonplace in CC. Since the users' is completely unaware of the cloud environment, they may not have the permission to change their canonical software they intend to run on the cloud. Secondly, a CC can be easily maintained and expanded, because it is built from canonical hardware. In order to maintain the practicality of the CC, these two key factors must be maintained.

One solution to SC attacks is the modification of the encryption algorithms (AES, RSA, and ElGamal). The solution is based on writing constant time AES algorithm because the variable timing AES has created an opportunity for attackers to launch an attack. However, to write constant time AES encryption algorithm is very difficult because constant time AES is unacceptable for many application. Kim *et al*. (Aumüller, Bier et al. 2002) have implemented a prevention mechanism for active time-driven and trace-driven cache-based SC attacks in the cloud environment. Hyperthreading, preemptive scheduling, and multi-core OSs are the leakage channel which forms the basis for these time-driven and trace-driven attacks. In preemptive scheduling, the attackers VM and victim VM may use a single CPU core and its cache. In hyperthreading, multiple hardware threads execute on a single CPU core. While in multi-core the attackers and victim may be executing in parallel on a separate CPU core with a shared L3 cache. This framework is compatible with the existing server hardware and tenant software and it will not affect the system performance. In their prevention solution, they have given an individual access to some part of the cache known as stealth page to prevent cache-based SC attacks. However, in order to access these stealth pages by using software application, their prevention mechanism requires to changing the software being run in the guest VM. Since the modification of client software violates the cloud model and describes the requirement for a transparent prevention mechanism to the client. The state of the art literature showed very little work describing the severity of other side channels including power consumption, electromagnetic radiation in the cloud as compared to cache-based SC attacks. To this extent, the most interactive device is the CPU cache, thereby, is the commonly targeted channel to exploit for the successful SC attacks in the cloud. This is because it generates one of the highest and reliable communication speeds. Thereby, this chapter describes the prevention mechanisms for cache-based SC attacks as opposed to any other channel (Zhang, Juels et al. 2012).

Misiu et al., proposed a preventive mechanism for SC attacks without changing or affecting cloud model. In addition, if a prevention mechanism can be compiled without change or effect the existing hardware and software then it can be easily adapted to the existing cloud system without any interference to the cloud functionality. For this, the author used the Xen source code for the prevention of SC attacks without change the client side or hardware. Wang et al., (Wang and Lee 2006) proposed the hardware-based mitigation methods which reduce the cache accesses by designing new caches, or by caches partitioning with dynamic or other efficient methods. However, this prevention mechanism requires changing the underlying hardware and software. Mitigation of SC attacks is very necessary at a hardware level since these channels of hardware level are not affected by the strong mechanism of software isolation. The attackers steal the information from the victim through shared functional unit that is dynamically allocated to each of processes in every cycle. In this sharing process, one process can interfere with another leading process through a side channel. In their paper, two methods are proposed one is Selective Partitioning and the other is novel Random Permutation Cache (RPCache). Selective partitioning by hardware (or software) can prevent the simultaneous multithreading/functional unit covert channel problem. The RP cache solution is implemented by using distinct memory location to cache mappings between a process that need isolation from each other and is used to mitigate software cache-based SC attacks. It can also find which cache location is used by another process. The main advantage is low-performance degradation. While the disadvantage is extra overhead when two cache sets are swapped.

Weiß et al. (Weiß, Heinz et al. 2012) conducted a cache timing attack on AES for the first time in an L4Re VM running on an ARM processor inside a Fiasco.OC microkernel. The attack is implemented using Bernstein's correlation attack and the target of this attack is many popular AES encryption algorithms including the one in OpenSSL. The extraction of the most fine-grain information from inside a VM is the significance of this work (AES vs. ElGamal keys in (Zhang, Juels et al. 2012)). The cipher text determines the entry of the loaded table by a byte of the cipher state. Hence, information about the confidential key of AES can be extracted by accessing the cache directly that during execution which table data have been inserted into cache like trace-driven attacks. The corresponding information can also be extracted by monitoring the behavior of timing during various AES executions over time, like time-driven cache-based SC attacks.

Yarom et al. (Yarom and Falkner 2014) demonstrate a trace-driven flush + reload attack which accesses specific memory lines to evicts the data from the LLC and extract the encryption key of the RSA algorithm. Furthermore, he noted that this attack could be applied in CC. Similarly, Tiri et al. (Liu, Yarom et al. 2015) proposed an analytical model which forecasts the symmetric key of ciphertext against timing attacks based on the lookup table and length of cache lines. The access-driven cache attacks need the attacker to monitor that which lines of cache have been monitored (like trace-driven attacks), but similar to timing-driven SC attacks it does not need detailed information about the cache that how and in what order the cache was accessed for data. Therefore, these classes of attacks can be varied with each other based on the attacker's access capabilities. The software-based countermeasure is needed for the mitigation of these types of attacks. Because hardware based solution takes time and degrades the overall performance. The hypervisor-based software solution for these type of attacks is cache flushing and cache warming, however, it degrades the overall performance of the system in term of CPU speed and load (Godfrey and Zulkernine 2014).

In (Godfrey and Zulkernine 2014), the author proposed a purely software-based server-side defense for cache-based SC attacks in the cloud. To make it fully deployable on the cloud model, the author implemented the solution in such a way that it does not require the software used to run CC or hardware changes for the prevention of SC cache-based attacks between co-resident VM. The prevention mechanism for SC attacks should be invisible and secure from cloud provider as well as from client and only visible to the cloud developer. Wang (Wang and Lee 2007) mitigate SC attacks by redesigning or partitioning the cache. The author in (Wang and Lee 2007) identified two main solutions namely Selective Partitioning such as Partition-Locked cache (PLcache) to hide the access pattern of the cache by locking cache lines and the other is novel Random Permutation cache (RPcache) to complicate patterns of the cache by randomizing cache mappings. One of the solutions is to minimize the interference between cache lines by partitioning the cache. The other class of solution is to allow sharing by randomizing the interference between cache lines so that no useful confidential information can be extracted. The RPcache solution is implemented by using distinct memory location to cache mappings between the processes that need isolation from each other and is used to mitigate software cache-based SC attacks. It can also find which cache location is used by another process. These hardware-based solutions, however, is unable to provide practical defensive mechanism until CPU designer and cloud provider purchase and integrate them into CPUs and cloud providers purchase them. The main advantage is low-performance degradation. While the disadvantage is extra overhead when two cache sets are swapped.

2.8 Conclusion

CC is a shared open environment, which has its own characteristics and features such as on-demand services and multi-tenancy. Specifically, it introduces multi-tenancy to facilitate the users to share computing physical resources provisioned over the Internet on-demand scaling. While multi-tenancy has many benefits, this paradigm introduces a new concept known as clients' co-residence and VM's physical co-residency. This co-residency arise security vulnerabilities to CC and enables a new form of sensitive information leakage including SC attacks. Although there are many benefits to adopting CC, however, security is the most significant barrier to adoption. In order to gain the trust of clients, cloud provider must consider the security in CC. VM managers (VMMs) namely XEN and VMware for modern virtualization systems enforce logical isolation between VM by using sandboxing mechanism. Since this logical isolation is not equivalent to physical isolation, the attackers can easily circumvent this logical isolation by using SC attacks.

In this thesis, different hardware and software, specifically CPU cache-based SC attacks and their countermeasure have been discussed. CPU cache is one of the most prone hardware devices targeted by adversaries due to its high rate of interactions and sharing between processes. In addition, several methods have been described by which the attacker can observe the memory pattern of the victim process. e.g., one that executes encryption algorithm with an unknown private key. These methods are categorized into various type based on cache state. In one method, the affect the cache state is observed and then measure and analyses the consequence on the encryption algorithm running time, and in second methods the state of the cache is investigated after or during encryption. The second method is found to be noise-resistant and particularly effective. For 10 years it is a known problem in a virtualization environment. The most past attacks applied on the L1 cache which exploits the hyper threading or scheduler weaknesses. However, the existing LLC attacks (L2 or L3) such as the prime probe, flush reload, and LLC attacks require memory deduplication and usage of huge pages. Some attacks do not have restrictions such as hyper-threading and memory sharing. There is a need for prevention mechanisms which is hypervisor-based and does not need any software by the client or the changing of the underlying hardware. The hypervisor-based software solution for these type of SC attacks is cache flushing and cache warming, however, it degrades the overall performance of the system in term of CPU speed and load.

CHAPTER 3: PROBLEM ANALYSIS

In this chapter, we aim to analyze the existence of the cross-VM cache-based SC attacks and the performance of the existing prevention mechanism to mitigate these attacks in the cloud environment. Although cache-based SC attacks and the prevention mechanism for these attacks in multiprocessing systems namely in networks, in OS, and in database systems have already been studied by the researcher for many years. However, in CC this is a new topic for research. As we already discussed in detail in Chapter 2 when two or more VMs run on a multiple cores system. One VM would be able to disturb the cache access of another VM and can extract secret information, even if every VM is running on a dedicated core. This situation has created a security risk in the form of cache-based SC attacks in the cloud environment. Therefore, in this thesis, cache-based SC attacks and the preventive countermeasures for these attacks in CC have been discussed. The main objective of this chapter is to analyze the aforementioned research problems discussed in Section 2.6 to establish the problem. The measurement parameter to analyze and establish the problem is the implementation of the cache based SC attacks in the cloud environment and to analyze the overhead of the existing prevention mechanism for cross-VM cache-based SC attacks in term of CPU's load and speed and cache usage. The problem analysis for this thesis consists two parts.

The first part of the problem analysis is to implement the cache-based SC attacks in the cloud and a non-cloud environment. This analysis is accomplished on the basis of Prime + Probe and Flush + Reload method to check the existence of information leakage through shared devices. We carried out the software based SC attacks, in which victim program play the role of sender and the attacker program as a receiver. The attacker places some code in the cache during the execution of victim program. The attackers observe the difference in the cache access time and execution time of encryption algorithms through software side channel to leak some confidential information to extract the full encryption key or part of the encryption key.

The second part of the problem analysis is to formulate the overhead of the existing prevention mechanism for cross-VM cache-based SC attacks. The initial findings are verified through apache, cachebench, and cachegrind benchmarking experiments on a real cloud environment. The results from two hypervisors including unmodified (default/unsecure) and static partitioned unveil the performance degradation in terms of bearable load, cache utilization and memory access time due to static cache partition as a prevention mechanism. The remainder of this chapter is organized as follows. In Section 3.1, we present the experimental methodology for launching the SC attacks and for the performance analysis of the prevention mechanism. In Section 3.2, we provide the detailed investigation of the SC attacks and implement SC attacks by using flush + reload and prime + probe techniques. In Section 3.3, static cache partition as a solution to SC attacks is presented. Section 3.4 describe the performance evaluation of the static partitioned based prevention mechanism by using various benchmarking experiment and the chapter is concluded in Section 3.5.

3.1 Experimental Methodology

In this section, we report the details of the experimental setup for this study. In order to evaluate the prevention mechanism for SC attack, we implemented the cache-based SC attack in native OS, XEN, and VMware hypervisor. Because for the analysis of prevention mechanism, it is needed to implement the attack first. For this, we utilize the customized version of XEN hypervisor on Ubuntu and creates two VMs. We changed the XEN source code according to our requirement and used different benchmarks. The main focus of our work is to prevent the cache-based cross-VM SC attacks and to evaluate the computational overhead of prevention mechanism in term of CPU load and CPU speed.

Table 3.1: Experimental Environment in Problem Analysis

Items	Detail
CPU Processor	Intel Core i5-3450 CPU @ 3.10GHz, 4 cores, Hyper Threading disabled
L1 Data-cache	32KB, 8 way associative, line size 64
L1 Instruction-cache	32KB, 8 way associative, line size 64
L2 Cache	256KB, 8 way associative, line size 64
L3 Cache	6144KB, 12 way associative, line size 64
Memory	11915MB DDR3 @1333MHz
VMM	Xen Hypervisor with static cache partition
Virtual Machines	HVM guest, 1GB memory, 1 dedicated core for individual VM
Guest OS	Ubuntu 12.04.5

3.2 How cache based side channel attack works

As described in Section 2.4, the root causes of the cache-based cross-VM SC attacks are the memory sharing and cache interference and specifically multitenancy and co-residency in virtualized environment. PTP (Prime + Trigger + Probe) and Flush + Reload are the two methods widely used for conducting SC attacks and the main causes for these attacks are cache interference e.g., memory deduplication and usage of the huge pages. Flush + Reload attacks are conducted by sharing some physical memory pages between the attacker and victim (Zhang, Juels et al. 2012, Yarom and Falkner 2014). The PTP technique does not require the sharing of memory pages between the attacker and the victim. Instead, the PTP attacks are conducted by sharing the same CPU cache set between the attacker and the victim (Irazoqui, Eisenbarth et al. 2015). PTP is mostly used to launch time-driven attacks by using the whole cache set while access-driven attacks are mostly conducted by using the Flush + Reload method (Tromer, Osvik et al. 2010), which uses a specific single cache line. In the access-driven channel, the value sent by the sender is written, and then the receiver reads and stores that value. While in timing channels signaling information of the sender is observed and decoded by the receiver by modulating the use of resources over time.

These two methods are proven to be conducted on the systems when the memory deduplication feature is enabled by the VMM to share some pages between the attacker and the victim processes (Yarom and Falkner 2014). Due to the sharing pages capability, the attacker can know about the eviction of a particular memory line from all levels of the cache. The spy observes the memory access timings to leak the secret information. The Flush + Reload technique is a variation of Gullasch's attack (Gullasch, Bangerter et al. 2011) which can be adapted for use in the multi-core and in virtualized environments. Gullasch et al. (Gullasch, Bangerter et al. 2011) also conducted attacks on specific memory lines. However, the victim process is frequently interrupted by the attacker and as a result, it generates much false-positive. Similarly, Yarom et al. (Yarom and Falkner 2014) conducted the Flush + Reload attack to extract the secret key of RSA across different cores in virtualized environment. Later on, Irazoqui et al. (Suzaki, Iijima et al. 2011) conducted the Yarom's attack on cross-VM hosted by VMware in virtualized environment to extract the cryptographic key of AES algorithm. In our thesis, we used the Prime + Probe method to implement the Gorka's attack. The basic algorithm which is used for both prime + probe and flush + reload techniques in order to implement cache-based SC attack is as follows:

Table 3.2: Algorithm for Implementing Cache based Side Channel Attacks

Sender queries	**Receiver queries**

(Wait for receiver to perform some queries)	**for** $i := 0$ **to** $N-1$ **do**
	{Put *Cache (i)* into the *cached* state}
	Access memory maps to *Cache* (i);
	end for

for $i := 0$ **to** $N-1$ **do**
if *DSend* (i) = 1 **then**
{Put *Cache* (i) into the *flushed* state}
Access memory maps to *Cache* (i);
end if
end for
	(Wait for sender to prepare the cache)

(Wait for receiver to read the cache)	**for** $i := 0$ **to** $N-1$ **do**
	Timed access memory maps to *Cache (i)*;
	{Detect the state of *Cache (i)* by latency}
	if *AccessTime* > *Threshold* **then**
	DRecv (i) := 1; {*Cache (i)* is *flushed*}
	else
	DRecv (i) := 0; {*Cache(i)* is *cached*}
	end if
	end for

3.2.1 Implementation of Cross-VM Cache-based SC Attack by using Flush + Reload technique

In order to conduct cache-based SC attacks, Flush + Reload technique is used. This technique consists the three phases.

Step 0: attacker will flush the cache (Flush)

Step1: Target victim access cache line and do some operation

Step 3: attacker measures the delay by reloading memory lines

During the first phase, the observed memory line is flushed from all the levels of cache including L1, L2, and L3. In the second phase, the attacker will wait until the target victim access to the memory line and do some operation before the third phase. During the third phase, the spy measures the delay in the memory access timings by reloading the memory lines. If the victim accesses the memory line during the wait phase, the cache hit occurs and the reload time will be short because the monitored line will be available in the cache memory. While a cache miss occurs when the victim is unable to access the cache memory lines, the reload time will be high because the lines will be required to access from the main memory. The flush reloads attack is always conducted on the LLC.

These attacks have become more powerful and dangerous due to two properties: First, unlike the prior attacks which access some specific cache set, the attackers try to access specific memory lines. Consequently, the Flush + Reload does not require any further processing for detection and does not generate a false alarm. Flush reload attacks are only conducted on the X86 architecture and are unable to be conducted on the ARM architecture, although the ARM architecture has instruction for the eviction of cache lines. However, it does not allow an unprivileged user process to use the eviction intrusion to selectively evict the information from the cache memory (ARM 2012).

3.2.1.1 Flush + Reload Attack Scenario

We have implemented Gorka flush + reload attack in the VMware ESXI 5.5.0. We used two VMs one for an attacker and the other for a victim, both are communicating through local IP connection. Where we have executed the attacker in one VM and victim program on another VM and on different cores. For the purpose of this attack, the authors in (Ristenpart, Tromer et al. 2009) have been solved the co-location problem which ensures that two VMs (attacker and victim) are running on the same physical machine. The VMM such as XEN and VMware provide memory overcommitment feature, however, Irazoqui et al. (Irazoqui G 2014) exploited this feature especially focusing on memory deduplication in their attack which Suzaki et al. (Suzaki, Iijima et al. 2011) have implemented in their research. The hypervisor (VMM) has the ability to search regularly for the same pages and merge the identical pages and make a single copy of the redundant data. Once the VMM execute this, this scenario enables the cache-based SC attack because both the attacker and victim will access the same physical memory. The implementation of this attack is such that target program is executed in VMware ESXI 5.5.0 running Ubuntu 12.04.5 64 bits, kernel version 3.11 for encryption using C implementation of the AES OpenSSL 1.0.1f. We have conducted all experiments on a machine having features an Intel i5-3450M four core clocked at 3.10GHz. Core i5 has three level caches including L1, L2, and L3 but in this attack, the L3 cache is used because this LLC is always shared between programs. The attacker and victim share this L3 cache for launching SC attack. The attack steps are as follow:

(a) Flushing step

In this step, the attacker flush the desired memory from the L3 cache by using the clflush command hence make sure that if needed next time they have to be accessed and retrieved from the main memory. It is important to note that clflush command not flush the desired memory lines from the L3 cache of the corresponding cores but it also flushes from L1 and L2 of the of all the different cores in the same physical machine. Because if it only flushed the cache of the corresponding, the attack would only work if the victim and attacker's program were co-locating on the same core.

(b) Target accessing step

In this step, the attacker waits for the victim to run a fragment of code, which might use the cache memory lines that have been flushed by the attacker in the first step.

(c) Reloading step

The attacker reloads the previously flushed memory lines in this step and takes all the measurements such as measure the reload time it takes for these flushed lines. On the basis of these reloading time, the attacker knows whether the memory lines accessed by the victim or not in case if accessed by victim the corresponding memory line would be present in the cache (cache hit) otherwise will not be present in the cache if not accessed by the victim (cache miss). The attacker takes the advantage of this timing difference between a cache hit and a cache miss and can easily detect the encryption key by analyzing the victim activity.

(d) Discussion

The victim is an encryption server receive encryption quires through socket connection and in response sends back the ciphertext. The attacker sends the encryption queries to the victim. In this attack, a package of 16 bytes (the plaintext) sends to the encryption server unlike Bernstein's attack (Bernstein 2004), where the server receives packages of 40 bytes (the plaintext). The attacker does not know about the confidential encryption key used by the encryption server. The victim program receives the encryption queries sent by the attacker program. All the measurement such as the required time for the reload step is performed on the attacker side. In this attack, only a single line is required to monitor. The flushing step is always occurs before encryption and reloading can be done after encryption, i.e. the attacker will not interfere with the attack process.

In this attack, the attacker first discovers the offset of the T tables' addresses with respect to the beginning of the library. After gaining this information, the attacker is able to refer to any memory line that the value of T table holds, even the ASLR (Address Space Layout Randomization) is activated. Then, it sends encryption queries to the encryption server and receives the interrelated ciphertext. After each encryption, the attacker checks the value of the chosen T table by Flush + Reload technique whether its value have been accessed or not.

We assume that the attacker monitors the memory line corresponding to the T table first position, where T is the lookup table is applied to the i-th byte of the targeted AES state before the last round. It is further assumed that n T table can adjust in the memory lines, for instance, for this attack the memory lines will hold the first n T table position. If any value of the T table entries is equal to s_i in the memory lines (i.e. S_i Ɛ {0,…., n} if the first n T table entries in the memory line) then the accesses memory line will be present in the cache with a high probability shows that these memory lines been accessed by the encryption server. However, s_i with a change value means that accessed memory lines are not loaded in this step. The probability that encryption process did not access the specific T table memory lines is given as:

$$\Pr\left[\text{no access to T}\left((\text{i})\right)\right] = \left(\left(1 - \frac{t}{256}\right)^{l}\right) \tag{3.1}$$

Here *l* represents the number of accesses to the particular T table. Since each encryption uses 40 access to each of the T table, therefore l=40 for OpenSSL 1.0.1 AES-128. The probability that the cache line is not accessed is Pr (no access to T(i)) = 28%. Therefore, we can easily distinguish about the accessing of memory lines whether it is accessed or not. In order to distinguish whether the line is accessed or not is to measure the reload time for the targeted memory lines. If the reload time is high it shows that the memory lines are accessed and the low reload time shows that the memory line is not accessed.

The key recovery step is then executed after all the measurement performed which takes less than half a minute. The result is shown in Figure 3.1. The vertical access refers to the correct bytes of key and the horizontal access refers to the number of encryption needed to recover the key. Due to the noise ratio in the virtualized environment we need different number of encryption for the attack in the Linux and for the cross-VM attacks. We need 100 thousand encryption for the correct bytes of key in Linux and 400 thousand encryption for recovering the correct bytes of key in cross-VM attack scenario.

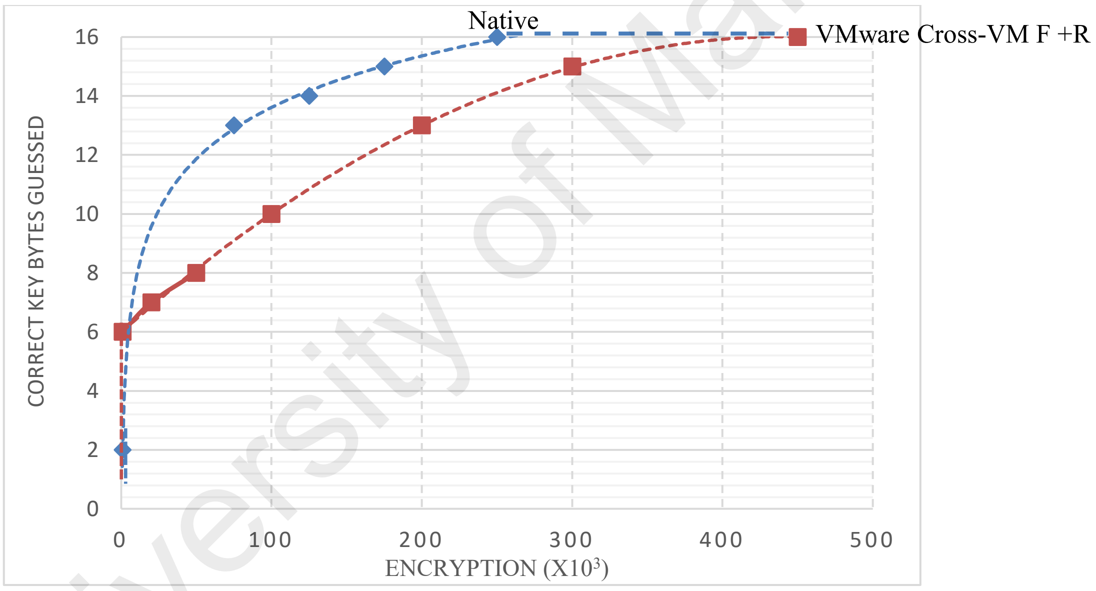


Figure 3.1: Number of the Key Bytes of AES Key Correctly Guessed vs Number of Needed Encryption

3.2.2 Implementation of Cross-VM cache-based SC attack by using Prime + Probe technique

Similarly, the prime + probe method is used in order to conduct cache-based SC attacks. The document form of SC attack was conducted by Ristenpart *et al.* (Ristenpart, Tromer et al. 2009), by using PTP which consists of the following three steps:

Step 0: attack fills the cache (prime)

Step1: Victim evicts cache lines while performing encryption

Step 3: attacker probes data to determine if the set was accessed or not

In the PRIME step, the attacker filled the CPU cache with his own data; in the IDLE step the attacker waits for some random interval for the victim to perform some operation; then finally in the PROBE step, the execution is resumed and the cache is refilled to measure the delay between the cache access and memory access time. In addition, during the first phase, the cache memory is divided into accessed and un-accessed categories. In the prime phase, it generates cache hit and cache miss based on the accessed and not accessed. Then finally, the probing instance has more values, which indicates how much time is needed to access the cache line with a primed cache. Then the delay in the memory access timings are observed to extract the secret information. This memory access timings are considered to be an easier way to exploit the cryptosystem and functionality of the system and is more difficult to control. The purpose of Ristenpart's PTP attack was to check if a cache-based SC attack could be established between the two co-located VM on the same physical machine. This attack requires that the attacker VM and target VM must be placed on the same physical machine. This SC was conducted such that the probing instance or first VM could collect a message that the target instance or second VM encodes in its usage of the cache.

3.2.3 Experimental Setup

We have implemented the cache-based SC attack by using prime + probe method. In native setup we have installed the attacker and the victim programs on the same core within same OS. For the cross-VM attack setup, the attacker and the victim programs have been installed on different guest VM which has different cores and different OS but both are sharing the same hardware.

3.2.3.1 Attack1 Setup: Attack in Native Operating System and in Single VM

In the native OS, the PTP technique is implemented in such a way that the attacker and the victim programs are executing on the same core. Since VM is a guest operating system, the execution of cache-based SC attack using PTP technique would be same as SC attack in native OS. In both environment the attacker and victim programs will be executed on the same core. In the PTP technique, the probing instance (attacker) first divides the cache lines into the touched and the untouched category. Cache lines in the touch category have been accessed by the target and in the untouched category have not been accessed by the target. After categorization step, the probing instance primes the monitored cache by filling his own data. Now in the victim accessing stage, the attacker waits for the victim to perform some operation causing the eviction of some cache lines that were primed by the attacker in the first stage. Now in the probing stage, the attacker access the prime data again. When the attacker reloads the data from the set that has been used by the victim causing a higher probe time because some of the primed cache lines have been evicted. However, if the victim program did not use any set of the cache lines in the primed set, causing a low probe time because all the primed cache lines will still reside in the cache. The probing instance has a series of values which represent the access latency for cache lines.

The latency difference for cache lines in the touched category compared to the untouched category shows that the victim instance was trying to communicate. Later on Wu *et al*. (Wu, Xu et al. 2012) refined the PTP technique, where they conducted a high-speed channel by communicating a “1” or a “0” . The generation of these “1” and “0” depends on the variation of timing category whether it is positive or negative (assuming the variation is above a specific threshold value). In addition, their attack has the ability to transfer bit-streams of over 190kb/s. So far, this SC attack is the most reliable and robust cache-based SC attack in a virtualized environment. This technique is mostly used for sequential cache-based SC attacks, making it a good example of a canonical attack. Since all the cache-based SC attacks in the cloud rely on this basic PTP technique, a successful prevention of its principle could prevent all the present cache-based SC attacks.

Therefore, in our research, we have implemented and analyzed Wu *et al*. attack for our problem analysis (Wu, Xu et al. 2012).

3.2.3.2 Attack2 Setup: Cross-VM Attacks

We have also implemented Gorka et al. attack in which they used the PTP technique for the extraction of cryptographic key targeting the AES algorithm running in victim VM. Unlike Gorka flush + reload attack this attack is not rely on the deduplication, instead, it uses the huge pages for conducting the SC attack by using PTP method. This method can be used to extract information from any encryption algorithm (i.e. ELGamal, RSA) but Gorka's target is the AES algorithm. AES is also stored in a cache and the attacker can leak the detail of AES to extract the cryptographic key. Unlike normal information leakage, the leakage from AES algorithm is very dangerous. Because the attackers can detect the complete key and by using that key they can easily detect the confidential information. AES in most famous cryptographic libraries including OpenSSL, PolarSSL, and Libgcrypt are vulnerable to Gorka attack when runing in the most popular hypervisor such as Xen and VMware used by popular cloud service providers (CSP) namely Amazon and Rackspace. The attack on AES has existed in the literature for many years, however, in virtualized environment, this attack has been introduced in 2009.

The implementation of this attack is executed in Ubuntu 12.04.5 64 bits, kernel version 3.11. We have executed target process using the C-implementation of AES in OpenSSL 1.0.1f for encryption. This is used when OpenSSL is configured with no-asm and no-hw option. We want to remark that this is not the default option in the installation of OpenSSL in most of the products. We have conducted all experiments on a machine having features an Intel i5-3450M four core clocked at 3.10GHz. The cache hierarchy of Core i5 has three-level: It is important to note that L1 and L2 cache are private to each core while L3 (LLC) is divided into slices and shared among all cores. When attacker and victim are in the same core they use the L1 cache. When both attacker and victim are in the different core they use the shared L3 cache. The L1 cache line size is 64 bytes and is 8-way associative, with 215 bytes of size. The size of the L2 cache line is 64 bytes and a total size of 218 bytes cache and is 8-way associative. The L3 cache is 12-way associative with a 64 bytes cache line size and a total size of 222 bytes. The L2 cache in combined with the memory deduplication feature performed by the VMM allows the attacker to learn about cache accesses by the victim program. The attack scenario is such that one VM receiving the encryption queries with a secret key. The attacker VM is co-located with the victim encryption server but on different cores.

The communication between the attacker and victim is carried out through local IP and by using this connection the attacker start the spy process and sends the plaintext to the encryption server. The attacker start measuring the usage of L3 cache on the reception of cipher text. There are four main steps in the Gorka attack on AES in Xen Hypervisor: in the first step the attacker gain the knowledge about the LLC (L2 or L3), cache slice number, and the cache lines that fills one of the sets in L3 cache. In the second step, attacker tries to know about the set which T table occupies, because these T table needs to be accessed again for recovering the secret key. In the third step the attacker perform the prime, reprimes, and request encryption steps on the desired set to check whether the cache lines have been accessed or not. Finally in the last step, the attacker recover the cryptographic key used by the server by utilizing the measurements taken in step 3.

3.2.4 Experimental Results

In this section, we show the proof of the existence of SC attacks in virtualized environment by conducting the experiment and analyzing the results. We perform the SC attack and analyze the results in native OS, and in XEN hypervisor.

3.2.4.1 Result in Native Operating System

This is the basic setup in which we have executed the attack. We have executed two program in the native OS in the same core. This experiment is based on the number of encryption needed to recover the correct bytes of key. The Figure shows that due to low noise in a native operating the number of encryption for recovering the correct key is very low as compared to the virtualized environment. To distinguish L3 cache access from main memory is more susceptible to noise as compared to differentiate between L1 cache access and main memory access. Therefore, while L3 cache is mostly used for SC attacks, its make the SC attack more challenging. Figure 3.2 shows the result.

3.2.4.2 Result of Attacks in Single VM and in Cross-VM

In this scenario we executed the attack in virtualized environment in which two VMs: one is the attacker and the other is a victim are communicating through local IP. Due to the noise ratio in virtualized environment more encryptions are needed to recover the correct bytes of the key as shown in Figure 3-2. We have implemented Gorka attack and analyze the different results in our thesis. It is important to note that attacker has the administrator privileges in the cross-VM attacks due to the sharing resources in virtualized environment. In the first stage, the spy process recognizes the L3 cache access pattern in our Intel i5-3450M system and by using this method we can detect the division of L3 cache into a slice. The spy process makes us able to know about the cache division into two slices and that the selection method of the slice is based on the parity of the first non-set addressing bit (i.e., a 17th bit). Thereby, for filling set in the odd slice we need 16 odd lines and to fill a set in the even slice we need 16 even lines. In the second stage, the spy process recognizes the set in the LLC (L2 or L3) that each T table cache lines of the hold.

In order to recognize the set, each possible set is monitored according to the obtained offset from the shared library of the Linux feature. The set reserving of T table cache line is used in the encryption process around 90% of times while the set will remain unused around 10% of the time where 500 random encryptions in a cross-VM scenario in Xen hypervisor were observed. It is to be noted that monitoring time for an unused set is more stable which is in the range of 200-300 cycles as compared to the monitoring time of a set used by T-tables which is 90% around of the time. The last step is to execute Gorka attack to recover the AES key used by encryption server. Valid ciphertexts are to be considered for the step of key recovery that are below the average time. The measurements are taken in the customized Xen hypervisor-based when the corresponding last line of T table is monitored and the key is 0xe1 in this case.

The attack was analyzed in native, single-VM, and in cross VM-scenario requiring 275.000 and 650.000 encryption respectively to recover 16 bytes key. It is shown in the Figure 3.2 that in the single-VM and cross-VM environment more number of encryption is needed as compared to the non-noisier environment such as native OS scenario.

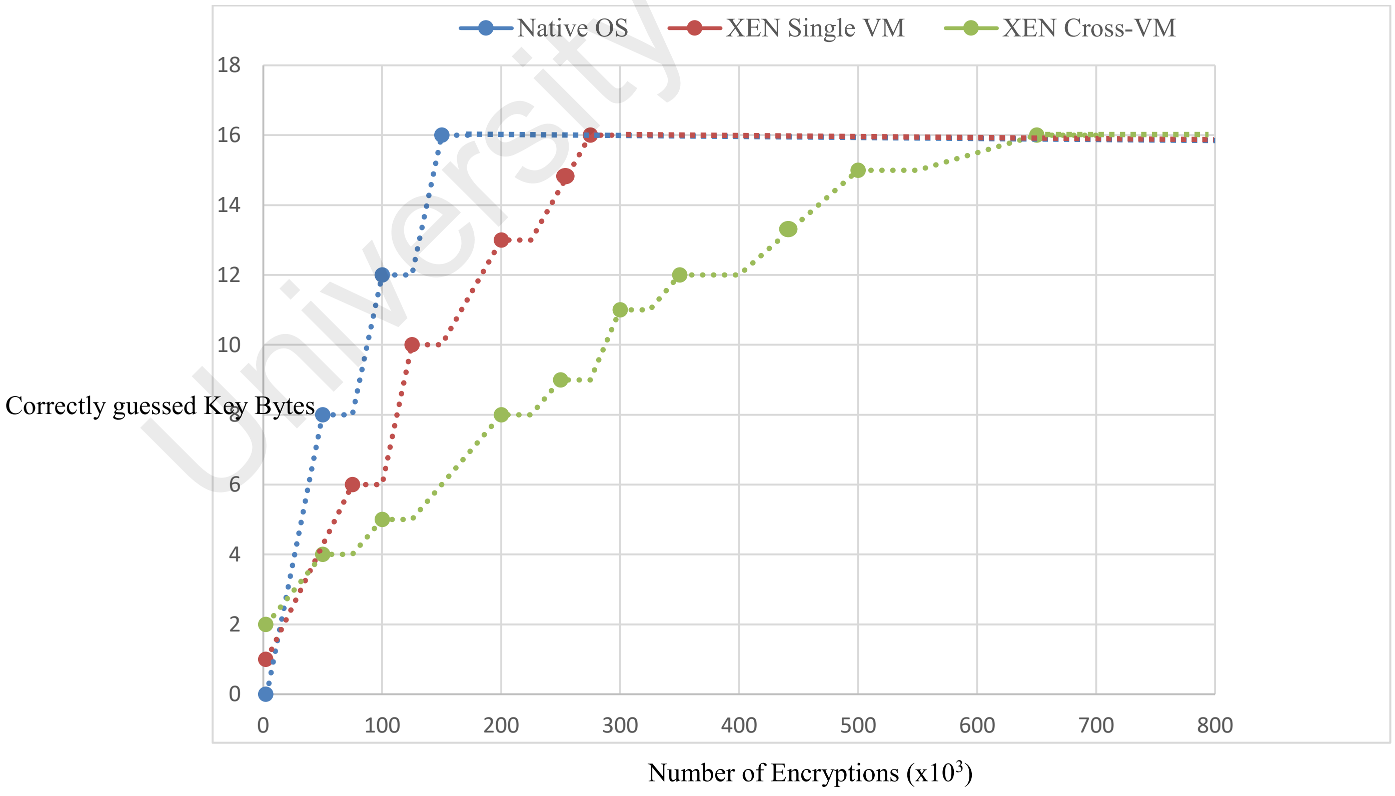


Figure 3.2: Number of Recovered Key Bytes Correctly Guessed vs Number of Requested Encryption for Native OS, Single-VM, and Cross-VM in XEN

For example, the encryption key consists of 16 bytes. In the native setup, it is clear from the Figure that 150.000 encryption is needed to recover the 16 bytes key. While in the single-VM scenario 250.000 encryption is needed to recover the 16 bytes key. This is because of noise that in the cross-VM scenario 650.000 encryption is needed to recover the whole key as compared to another scenario in which low encryption is needed to recover the whole bytes of the key.

Table 3.4 describes the correctly recovered key in number of bytes in both single VM and cross-VM. Single VM means that attack is conducted in single in which the attacker and victim programs are in the same VM. In cross-VM scenario, both the attacker and the victim programs are in different VM and in different cores. Table 3.2 shows that the required number of requested encryption for correctly recovered the whole bytes of key in single VM is less than as compared to cross-VM. Because in cross VM the external noise effect the results. We believe that due to noise SC attacks require a high number of encryption in the cloud environment as compared to non-cloud environment. Table 3.2 shows the result of cache-based SC attacks in native, single VM, and in a cross-VM scenario in XEN and VMware.

Table 3.3: Comparison of Correctly Recovered Key in Single and Cross-VM

In Single Virtual Machine (Single-Core)		**In Cross-Virtual Machine(Multi-Core)**	
Number of requested encryption	Number of correctly recovered key bytes	Number of requested encryption	Number of correctly recovered key bytes
10,000	1	30,000	2
90,000	6	60,000	2
130,000	10	100,000	4
150,000	10	200,000	8
200,000	13	260,000	9
250,000	13	300,000	11
260,000	14	350,000	12
265,000	14	450,000	13
270,000	15	500,000	15
275,000	16	650,000	16

Similarly, Table 3.5 describes the comparison of cache-based SC attacks in both Xen and VMware. This table shows the result of the SC attacks in both Xen and VMware implemented by flush + reload and prime + probe methods. The need number of requested encryption for correctly recovering the number of key bytes are different in each scenario.

Table 3.4: Comparison of Cache-based Side Channel Attacks in XEN and VMware

Name of Attack	**Scenario**	**Platform**	**Technique**	**No of Encryption**
Flush + Reload	Native OS	i5-3450	L3 Cache Flush + Reload	250
	Cross-VM	i5-3450	L3 Cache Flush + Reload	450
Prime + Probe	Native OS	i5-3450	L3 Cache	150
	Single-VM	i5-3450	L3 Cache (Gorka attack)	275
	Cross-VM	i5-3450	L3 Cache prime + probing	650

3.3 Prevention Mechanism

As we discussed in Chapter 2 in detail, the prevention mechanism is divided into two types: Software-based and hardware-based. The hardware based prevention mechanism need to change the underline hardware while for the software-based the client need to change their software which violates the CC concepts. There are several existing prevention mechanisms for cache-based SC attacks. These mechanisms may also prevent cross-VM cache-based SC attacks. For instance, one prevention method is to rewrite the software (AES) in a way that does not allow the known attacks to occur (Brickell, Graunke et al. 2006). Similarly, there is another prevention mechanism that needs the non-standard hardware to refine the processor architecture for the prevention of cache-based SC attacks (Wang and Lee 2008). Some research work proposed the disability of cache sharing for mitigating cache-based SC attacks (Oswald, Mangard et al. 2005). The author in (Shi, Song et al. 2011) used the cache partition by using dynamic page coloring technique. However, this approach requires the client to change their software which is against the cloud rule. To use the cloud resources we need to follow the cloud rules and according to cloud rule, we are unable to change the canonical software and hardware. In addition, if a prevention mechanism can be compiled without change or effect the existing hardware and software then it can be easily adapted to the existing cloud system without any interference to the cloud functionality.

For instance, Misiu et al. (Godfrey and Zulkernine 2014), proposed a hypervisor preventive mechanism for SC attacks without changing or affecting cloud model. We evaluated this hypervisor-based solution using the Gorka's cross-VM cache-based SC attack. In order to evaluate the performance of hypervisor-based solution, the Gorka's attack was given an ideal condition to execute in. Specifically, for attack, the attacker VM (probing instance) and the victim VM (target instance) and dom 0 VMs are running on a hypervisor and all the three VM were pinned to different CPU cores. This is the ideal configuration for launching cross-VM attacks and any variation in this configuration would make difficult the success of the SC attack. The author in (Godfrey and Zulkernine 2014) used static cache partitioning approach for the prevention of SC attacks without changing the client side or hardware. However, this hypervisor-based solution degraded the overall performance of XEN hypervisor by reducing cache usage, because if the cache size is 4MB and we divide it into 4 parts and make static partition of 1MB. For example, if there are 4 active VMs. Two of them are using the cache and the other 2 are not using the cache, however, each time hypervisor boot they will make the static partition according to the active VM. The cache will be wasted and the performance will be degraded. Because if one VM needed the more cache memory than the assigned one and the other needed less memory than the assigned one. Then the assigned memory to each VMs cannot be assigned other VM on the needed basis.

The authors in (Shi, Song et al. 2011) have tried to partition the cache dynamically. In their proposed approach, they are given a small portion of the cache to the secure encryption algorithms (AES, DES) to maintain the efficiency. However, their solution is unable to secure the VMs from the leakage attacks. Secondly, their solution needs the client's software to change and to be informed about their partition approach to take advantage of the partition. Our hypervisor-based solution by contrast partition the cache dynamically and it does not need to change the client's software, mean the software and hardware do not need to change. The reason for hypervisor-based solution using dynamic cache partition comes from the fact that we are dynamically partitioning the cache on the hypervisor side to make the clients unaware of the solution. In addition, client's software does not need any changes for using this solution. Our solution will be compatible with the existing software and hardware and will not degrade the system performance, which will fulfill the CC criteria. Our solution applies preventive mechanism rather than a reactive mechanism. Since according to our solution, two VMs cannot access the same cache lines, therefore there is no chance to create side channel between two VMs. We utilized a set of benchmark and Phoronix test suite for the evaluation.

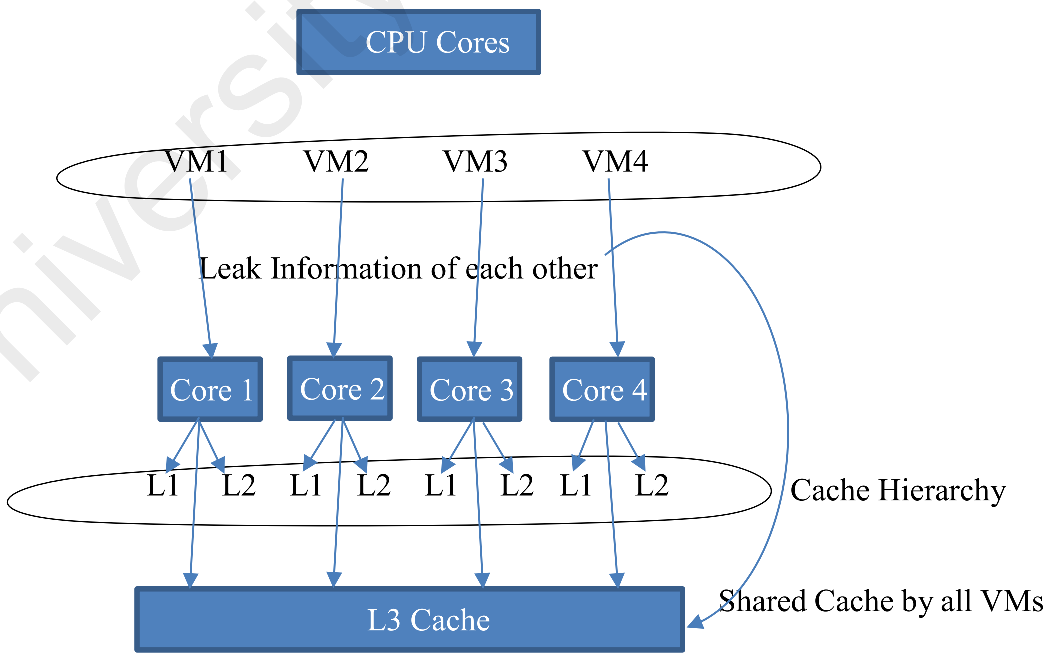


Figure 3.3: Problem Visualization

3.3.1 Cache Partitioning as a Prevention Mechanism

When multiple VMs run on multi-cores system, because of shared L3 cache one VM can extract the information of another VM by disturbing the cache access, even every VM is running on a dedicated core. Therefore fair partition of cache and cache-sharing is not only considered in a multi-programming system but also is a hot topic for virtualized environment. Cache partitioning mechanism is based on page coloring technique. Page coloring is a software-based approach for memory mapping to leads that how memory pages mapped to the specific cache set or cache lines (Soares, Tam et al. 2008, Zhang, Dwarkadas et al. 2009). Furthermore, the memory management module is controlled by page coloring approaches to ensure that a group of memory pages having the same color will be mapped to particular cache lines. The figure shows the mapping of memory pages to cache lines during memory management process. Page coloring technique is divided into static and dynamic types (Tam, Azimi et al. 2007, Jin, Chen et al. 2009). Static page coloring is the intuitive approach for the prevention of cache-based SC attacks (Jin, Chen et al. 2009) which provides a strong degree of isolation between VMs. However, this approach limits the number of VMs and degrade the overall performance in term of reducing cache usage for individual VM. Although static partition of the cache can reduce the eviction rate of cache data, consequently prevent the cache-based SC attacks. However, it reduces the memory usage or the size of the usable part of cache for individual VM. As a cache is divided into a static portion, it became smaller and many VMs compete for the same portion usage of cache making that portion the more efficient. These conflicting factors degrade the overall performance of this static partition based prevention mechanism. The static partition using page coloring technique is shown in the following Figure 3.4.

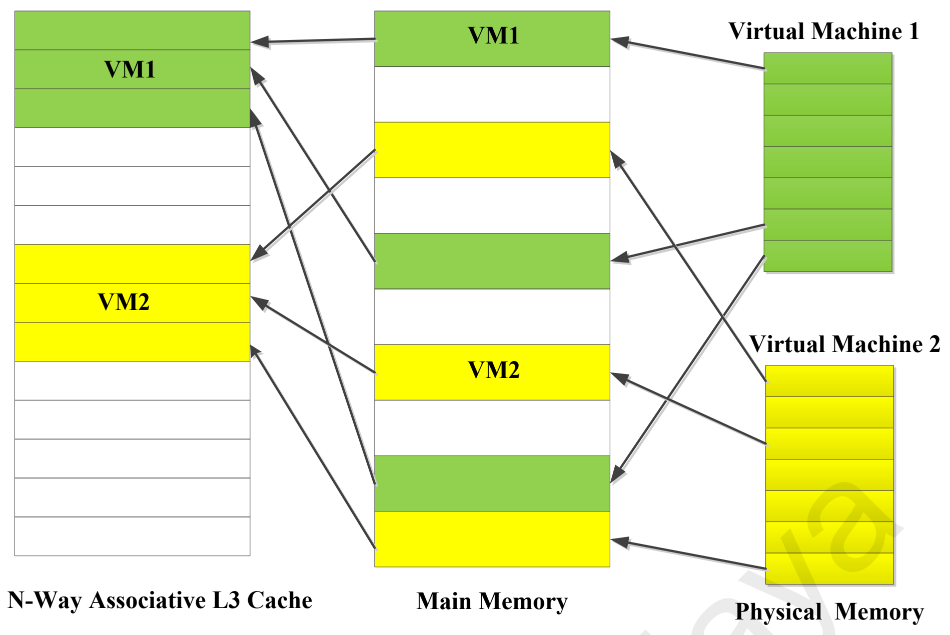


Figure 3.4: Static Cache Partition Using Page Coloring

Since the performance of many applications depends on the cache size and utilization, however, in virtualized environment, this approach proportionally limited the access to a cache set that a VM can use to effect the overall performance. In addition, depending on the cache total size and set associativity, the page coloring system is able to provide a very limited number of colors that impose a restriction on the number of running VM on a cache during VM provisioning. Since the cache color for a system is calculated as:

$$\text{Number of Colors} = \left(\frac{\textit{Cache Size}}{\textit{Number of way associativity} \times \text{Page Size}}\right) \qquad (3.2)$$

Moreover, since the partition is static it cannot be changed at runtime. It means that the partition cannot be scaled up or down after booting the hypervisor correspond with the number of running VMs. Therefore, the efficiency of static partition solution depends on correctly balancing the number of VMs correspond to a number of partitions. If the number of VMs are not correctly balancing correspond to the number of partition then this little mismatch can lead to a significant overhead and can degrade the overall performance very badly. For instance, if the shared cache (L3) is divided into four partitions statically during boot time and the number of created VMs are eight then this mismatch can lead to significance overhead in term of cache usage. Similarly, if the number of partition is four and the number of created VM is one then other parts of cache will be wasted and also has a bad impact on the speed because the cache size will be reduced. Since the size of cache partition is static and it cannot be just changed unless the system is rebooted.

To address this limitation, the author in (Shi, Song et al. 2011) proposed dynamic cache partition approach. To maintain the efficiency, they have assigned a small portion of the cache to the encryption algorithms. In their solution, they partitioned the cache based on the page coloring technique and assigned a secure color to the encryption algorithms e.g., AES and DES. It means that when two or more VMs would be using cache, the hypervisor will not partition the cache. However, the hypervisor would be partitioned the cache based on the execution of any encryption algorithm. Although this prevention mechanism maintains the cache efficiency, it requires the client to change their software e.g., encryption Algorithms, which does not comply with the cloud model. Moreover, it gives a small portion of the to the encryption algorithm cache by using a page coloring technique, mean it just secure the clients and programs which use the encryption algorithm. For instance, if one VM wants to leak information from another VM, then their solution is unable to prevent the information leakage across VMs. Therefore, there is a need for server-based prevention mechanism which is transparent to guest VM and the underlying hardware.

Moreover, this prevention mechanism prevents the extraction of the cryptographic key as well as the normal information leakage. This prevention mechanism follows the cloud rule as we discussed in Chapter 2 as a cloud model that does not need to change the client software or the underlying hardware. Our solution does not need to reboot the system every time for partitioning the cache on the provisioning of new VM. Additionally, if the cache is divided dynamically according to the requirement of each VMs, and the individual VM does not have to worry about that other VM will evict its data. Consequently, this improves the overall performance by increasing the cache hit and cache miss rate and also the cache usage for each VM. This solution is more preventive rather than reactive. There are two types of prevention mechanism: Intrusion response system (IRS) and Intrusion prevention system (IPS). Since we cannot wait for a cache channel might be forming we completely prevent the channel to form or occur, therefore our solution is IPS e.g., Proactive response rather than IRS e.g., Reactive response.We utilized the standard workload such as apache benchmark and Cache bench benchmark from the open sourced Phoronix test suite for the evaluation of the amount of overhead generated by the static-based partitioned hypervisor. These benchmarks are used because these are open source.

3.3.2 Phoronix Test Suite

We utilized the apache and cachebench benchmark from the Phoronix test suite to evaluate our proposed dynamic cache partitioned solution. The Phoronix test is commonly used for the evaluation for the performance measurement of various system attributes and subsystems. These attributes include cache usage, CPU load testing, cache access rate and how the Xen hypervisor is able to handle the high load distributed among different VMs. However, among the available tests, we focused on standard workload namely Apache benchmark and the Cachebench benchmark to evaluate the various performance attributes most significant in the virtualized environment. Apache was chosen because it is the most widely used software for load testing and typically find in the Cloud. Our solution is also related to virtualizing environment therefore, we chose this benchmark and Cachebench was chosen because it shows the cache usage in more detailed form.

3.4 Evaluation Parameters

The following test we performed by using the above mentioned benchmark.

- Load testing with varying numbers of VMS and partitions
- Cache Utilization with varying numbers of VMs and partitions
- Memory access rate with varying numbers of VM and partitions

3.4.1 Load Testing with varying numbers of VMs and Partitions

The Apache benchmark is a standard benchmarking tool based on HTTP webserver (Foundation. 2013). We utilized apache benchmark from the open sourced Phoronix Test Suite for the evaluation of the partitioned hypervisor. This benchmark is chosen because this typical software is found very easy to use and frequently available in the cloud. Table 3.4 shows the performance of the static partition in term of load testing that the modified hypervisor based on the static partition can tolerate how much load in term of request per second. As shown in the table that with increasing number of VMs and partitions the bearable load in term of number of request per second is decreasing.

Table 3.5 : Load Testing with Varying Number of VMs and Partitions

	Number of Requests per Second				
Number of Partitions	**With 1 VM**	**With 2VM**	**With 4VM**	**With 8 VM**	**With 16 VM**
Default (1)	3200	3500	3200	1500	700
Partition (2)	3200	3200	3100	1500	700
Partition (4)	2800	3100	3100	1500	600
Partition (8)	2600	2900	2800	1400	600
Partition (16)	2200	2100	2000	1100	400

We're assuming that if there are four cores in the system then we will have 4 VM. Dom 0 will divide the cache partition according to cache associative. If the cache is 24way and VM is 4. Then 24/4=6. Each VM will get 6 partitions. We concluded from the various experiments that if we will partition the cache into 16 at boot time then the overall performance of the system will be degraded even one VM is executing on the partitioned (16 parts) cache.

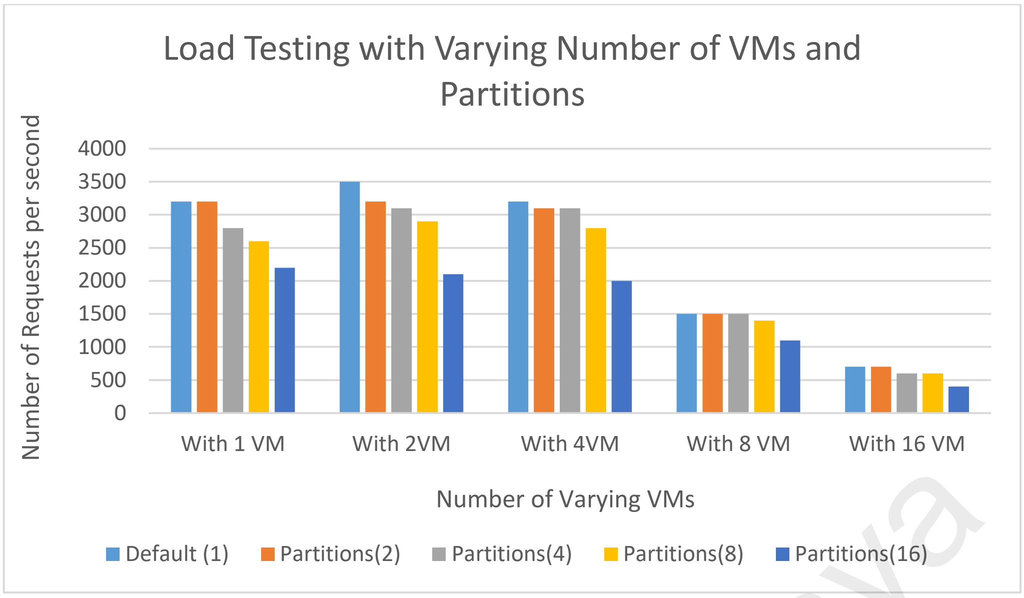


Figure 3.5: Load Testing in Static Partitioned Hypervisor with Varying Number of VMs and Partitions

In Figure 3.5, the result of the conducting experiment in static partitioned hypervisor is shown. We analyzed from this load experiments that as the number of partitions increases the request per second is decreased which show the bearable load of a system or load that a system can tolerate. The problem with the static partition during boot time is: once we create the static partitions at boot time we cannot change the partitions until we boot the system. For instance, once we divide the cache into 16 partitions and during this time one VM is running then one VM will be executing on one part of the cache and the remaining 15 partitions will be idle during execution of VM. Because we cannot change the 16 partitions into single partition according to the executing single VM. Moreover, we cannot change the partition into one partition with respect to the creation or execution of one VM. We have concluded from the experiments that the ideal distribution with minimum overhead would be an equal amount of VMs and amount of partitions for each set of partitions. Although static partition prevents overhead from cross-VM cache evictions, however, it would be very difficult to detect the number of VMs and number of partitions at boot time.

3.4.2 Cache Utilization with varying numbers of VMs and Partitions

We conducted this experiments by using cachebench benchmark. Cache utilization is investigated for static portioned hypervisor to check the amount of data accessed in bytes by each one. Cachebench includes various benchmark, however, we used the Cache Read/Modify/Write to evaluate the different level of cache in term of accessed data. We analyzed by conducting this experiment that in the static cache partition the amount of cache bandwidth would be decreases with increasing number of VMs and number of partitions. Consequently, static cache partition generates much more overhead as the number of VMs and partitions increases. Table 3.7 shows the result of a statistically partitioned hypervisor.

Table 3.6: Cache Utilization with Varying Number of VMs and Partitions

	Cache Bandwidth of Read/Modify/Write (MB per Second)				
Number of Partition	**1VM**	**2VMs**	**4 VMs**	**8 VMs**	**16 VMs**
Default (1)	17923	17128	15289	13567	13889
Partition (2)	15628	14035	13878	11228	12556
Partition (4)	14289	13582	12728	10988	10454
Partition (8)	10989	9366	8800	8487	8000
Partition (16)	4896	4098	3789	3567	3089

Figure 3.6 shows a gradual decrease in the cache utilization for each VM as the number of VMs and partitions increases. For 1 VM the cache bandwidth is more as compared to the 16 VMs and partitions. It means when the number of VMs and partition is increases the performance will be degraded because the cache bandwidth in term of cache read/write/modify bandwidth will be decreased. Similarly, if the number of partition is increased then the cache bandwidth for each VM will decrease even if one VM is running.

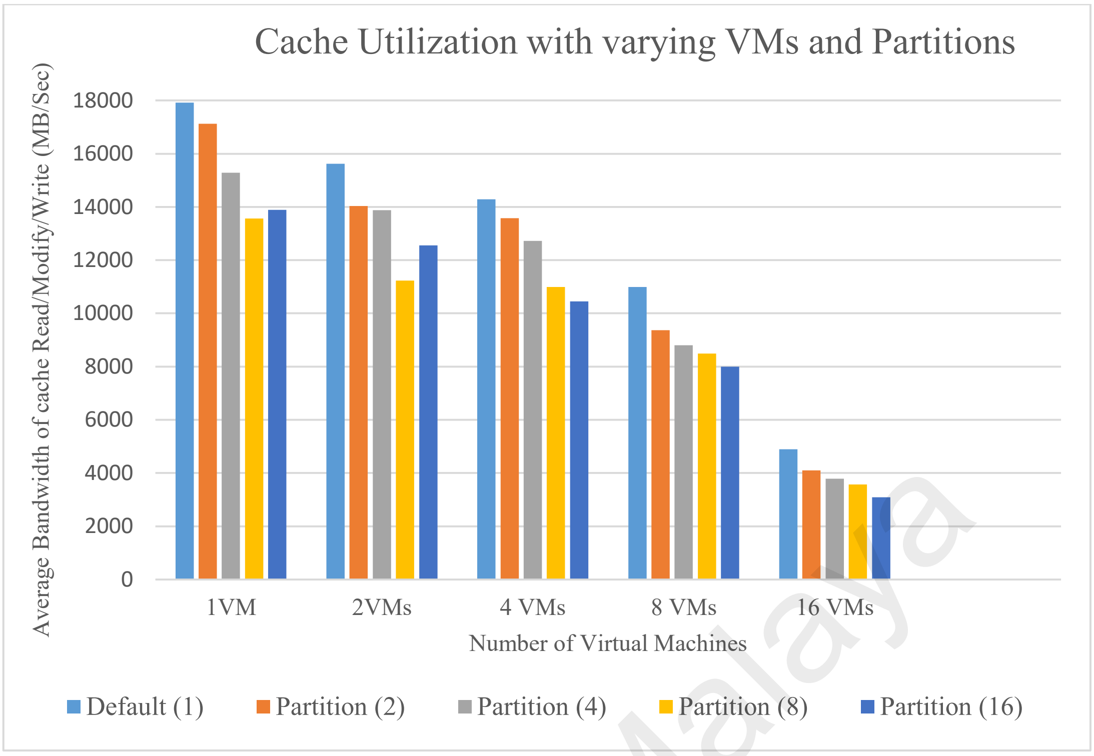


Figure 3.6: Cache Utilization with Varying Number of VMs and Partitions

3.4.3 Memory Access Rate with varying numbers of VMs and Partitions

The average memory access time is a valuable parameter to evaluate the performance of a memory hierarchy configuration. When a processor demand to execute an item from the main memory, it sends a load request to the cache memory. If the item resides in the cache it will generate a cache hit and in the case of absence, it will generate a cache miss. These cache miss and hit rate are used to calculate the memory access rate. We have calculated the total cache references, cache miss, and cache hit rate by using a cachegrind benchmark for the purpose to determine cache access rate. Then by using these values, we have calculated the memory access rate by our own designed program in the static cache partitioned-based hypervisor. Figure 3.7 shows the access latency for varying number of partitions with increasing number of VMs. We have observed from the analysis that the performance for 2-way and 4-way partitions are same as the default hypervisor.

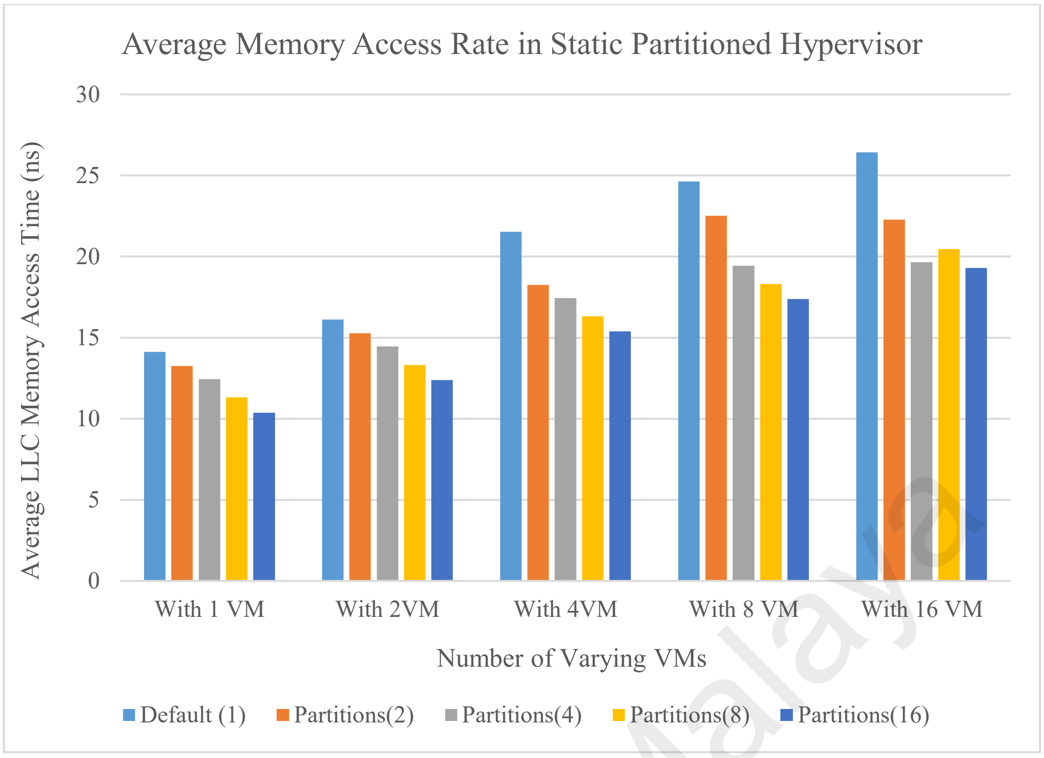


Figure 3.7: Cache Access Rate in Static partitioned based Hypervisor

However, when we increase the number of partition and number of VMs, there is a gradual decrease in the cache access time. When there is many numbers of VMs and partitions then there are chances for eviction of each other data that's why the memory access rate would be increased as the data will not be present in the cache memory and will be coming from the main memory. Since the program is running on the partitioned hypervisor, therefore, the performance will be degraded. If our design program for the calculation of cache access rate is run in the default/unmodified hypervisor then it will generate more cache hit as compared to the program which is running in the static partitioned based hypervisor. Because in the partitioned hypervisor case, our design program gets a portion of the entire cache. This proves that the overall performance will be degraded with the static cache partition.

3.5 Conclusion

In this chapter, the implementation of SC attacks by using two methods in the different hypervisors is presented. Then the prevention mechanism for SC attacks has been discussed in detail. According to the studied literature, one intuitive defense to mitigate cache-based SC attacks is the static partition of the cache is to divide the cache into statistical partition according to the different scheme of color assignment and strictly assign distinct page color to different VMs, so they access different parts of cache. We conducted the real-time experiment for the prevention of SC attacks based on the static partition solution and analyzed the load, cache utilization, and memory access time of this existing solution. Consequently, we analyzed that there are some bearable differences in the load, and memory access time between unmodified and modified (based on static cache partition) hypervisor as we know security is always comes with some overhead. However, there is a very large difference in the cache utilization of modified and unmodified hypervisor.

Although the static cache partition is simple and provides isolation, it potentially decreases the cache set for utilization. Consequently, the number of cache sets in typical processor cores is very limited which could reduce the number of executable VMs in a shared cache when using a static partition of cache. Consider, for instance, a system having 4 cores with 64-page color and 16 MB of memory for individual color. Therefore, this system support 64 VMs with one color each and every VM limited to footprint not more than 16MB. Moreover, cache utilization is low when all cores and all VMs are active. It was observed that once the static partition is created during the boot time, it cannot be changed after creation of more VMs until we boot the system and change some changes. In addition, existing page coloring mechanisms either is unable to adaptively adjust cache partitions efficiently or they unable to identify phase transition of application. To mitigate this problem, we extend page coloring with dynamic cache partitioning capability by adding recoloring mechanism. In this dynamic partition mechanisms, every VMs get their partition upon the creation without booting the system. There are some other approaches to prevent cache-based SC attacks, however, they all require the changes in the client software and in the underlying hardware. The prevention mechanism for SC attacks can be improved in term of cache utilization and performance improvement.

CHAPTER 4: HYPERVISOR-BASED PREVENTION MECHANISM USING DYNAMIC CACHE PARTITIONING: HBP-DCP

This chapter aims to present the details of our Hypervisor-based Prevention Mechanism using Dynamic Cache Partitioning (HBP-DCP) for the prevention of cross-VM cache-based SC attacks. We describe the building blocks and components of the proposed prevention mechanism and describe their functionality. The HBP-DCP is comprised of three main components, namely, admission control, cache usage monitor, and cache partitioner (Color-Aware Page Migrator). The admission control interprets and analyzes the user requests and takes a decision based on the availability, capability, and price of VM. The cache usage monitor analyzes and measures the number and utilized cache of executing VMs. The cache partitioner then divides/partition the cache according to executing VMs. The required amount of cache varies from VM to VM and the number of VMs also varies, therefore, it is difficult to fulfill the demand of the each VM for the different requested amount of cache in a static partition. Our approach divides the cache according to the executing number of VMs and facilitates the VMs requested amount of cache on the fly and also can prevent the cache-based SC attacks across VMs.

In the following sections, details of the HBP-DCP are provided. In Section 4.1, the overview of the system requirements for HBP-DCP. Section 4.2 presents the VM creation/provisioning in detail. Section 4.3 describes the various component of the proposed prevention mechanism, such as admission control, cache usage monitor, and cache partitioner in detail. In section 4.4 the significance of the proposed prevention mechanism is provided. Section 4.5 describes the data designing followed by concluding remarks in 4.6 for this section.

4.1 Hypervisor-based Prevention mechanism using Dynamic Cache Partitioning

In this subsection, we present the overview of our HBP-DCP mechanism that is capable of preventing cross-VM cache-based SC attacks. As already discussed in section 2.1.2, the basic idea behind cache-based SC attacks is the shared resources in CC as VMs resides on the same physical devices and can easily extract each other data by using shared L3 cache. Since it is clear from the name that hypervisor-based prevention has three main phases. The first phase is that Xen hypervisor will check the VM request. This usually includes whether the request is generated from the new VM or from existing VM. The second phase is to check the cache usage that how many VMs already exist in the cache and how much cache is assigned to those VM. The third phase is to reconfigure the cache and re-divide the cache according to the requirement of the current running VMs. This approach is to rewrite the software (Source code of Xen Hypervisor) in a way that no known and unknown cache-based SC attacks between VMs can succeed. This solution is more preventive than reactive. There are two types of prevention mechanism: Intrusion response system (IRS) and Intrusion prevention system (IPS). Since we cannot wait for a cache channel might be forming we completely prevent the channel from being forming or occurring, therefore our solution is IPS (e.g., Proactive response) rather than IRS (e.g., Reactive response).

4.1.1 Features of the Proposed HBP-DCP Prevention Mechanism

The aim of the HBP-DCP is to mitigate cross-VM cache-based SC attacks. The proposed prevention mechanism has the following features that distinguish it from the existing prevention mechanism:

- Generalizable: The fundamental cause of any type (e.g., Trace-driven, Time-driven, and access-driven) of cache-based SC attacks in the virtualized environment is the cache memory. Since the cache is the most interactive devices between VMs and it is always been targeted for SC attacks. As our proposed prevention mechanism is based on the cache memory partition, therefore, it can mitigate any type of cross-VM SC attacks which is based on the cache memory.

- Comply with Cloud Model: The existing solution does not comply with the cloud model as they need the client to change their software or the underlying hardware. As discussed in Section 2.1.1.2, our prevention mechanism confirming to the cloud model because it can be directly implemented into the hypervisor. Furthermore, it is transparent to the cloud model because it does not need any modification in the underlying hardware and in the client software.
- Portability: Hypervisor can be installed almost on every type of computing infrastructure. Since our prevention mechanism is hypervisor based means we have implemented by using the source code of an open source hypervisor. Therefore our prevention mechanism can be ported to any type of the supported software (hypervisor) and computing infrastructure.
- Applicable to Commodity Operating System: To implement the cache partition at the VMM (hypervisor) level is very beneficial. Since all the monitoring and partitioning activity will be done on the hypervisor level. Therefore it is applicable to Commodity OS the source code of which is unavailable such as a window OS. Most of the previous work is done for the multi-programming workload. However, our work enables the cache partitioning across and within the OSs, as it is implemented in the hypervisor and therefore improve the whole system optimization by providing more flexibility.
- Saving Cache Utilization: Hypervisor-based prevention mechanism is based on the dynamic partition of the cache. Therefore the overall performance can be improved by increasing the cache utilization for individual VM because VM is only giving as much more cache memory as they are requested. Unlike static

partition, the dynamic system avoids having to reboot the system every time on the VM provisioning so increase the overall performance of the system.

- Preventive rather than Reactive: Our solution is more preventive rather than reactive. Since we cannot examine when SC attacks might occur, we simply ensure that the two VMs would not be able to access the same cache lines for the purpose to create SC attack. Preventive mean early prevention before occurring of the attacks while reactive mean prevents attacks after occurring. Because once the attack occurs, it will harm the system even in a minute, therefore, early prevention of attack is more beneficial than post prevention.

4.2 System Architecture

We devise HBP-DCP mechanism for the mitigation of cross-VM cache-based SC attacks. This prevention mechanism is based on the open source code of Xen hypervisor. Since the source code of the Xen hypervisor is open source and freely available. Therefore, we chose Xen hypervisor for the creation of VM and for the implementation of our solution. Furthermore, our solution is also hypervisor-based and will be added to the existing source code of Xen. However, this prevention mechanism can be applied in other hypervisors namely VMware ESXI because it is general approach and is based on the cache partitioning. This prevention mechanism is enabled by admission control and VM provisioning rather than SC channel attacks. Therefore, we need to explain these terms in our thesis according to the requirement of our HBP-DCP prevention mechanism.

Furthermore, for this thesis, first of all, we have implemented the attack on the shared LLC (L3) cache, the detail of which is given in Chapter 3 in detail. Since we need to check the cross-VM cache-based attack. Therefore, VM provisioning is must to create two VMs on the Xen hypervisor. Since both the implementation of attack and solution are based on the shared LL cache, therefore, LLC (L3) must be in the system. The salient characteristic of the LLC is that it is by design an inclusive cache memory. Therefore, the data stored in the L1 and L2 caches is also copied in the LLC. Consequently, in the case of a cache miss in an L1 cache, the data will be checked in L2 in order to decrease the cache miss rate. Furthermore, if the data is flushed or evicted from the LLC, it will automatically be erased from all the other levels of the processor's cache.

Although shared cache has some advantages such as increased utilization of cache space, decreased cache miss rate, faster inter-core communication through shared LLC (L3 and L2), and the elimination of undesired replication of cache lines to reduce aggregate cache footprint. However, the major disadvantage of shared LLC is the uncontrolled contention can occur by allowing CPU-cores to access the shared LLC on a free basis. Moreover, HBP-DCP is always activated when the user sends a request to admission control for VM creation and when the VM provisioning phase will be activated by assigning VM to the specified client. The system requirements for the implementation of our prevention mechanism includes Xen hypervisor and Intel Core i7 with shared LLC (L3). We choose Core i7 having 4 cores because in this modern architecture each core has a dedicated L1 (instruction and data cache) and L2 cache but the L3 cache is shared amongst all cores. Therefore, the state of the art cache-based SC attacks target L3 cache.

Our proposed HBP-DCP prevention mechanism consists the following four components, namely; (a) Admission Control module has been used as a general mechanism to enforce the fair usage policy of resources on server, (b) After verification and availability of resources, once the admission control grants the request, then after this stage the global scheduler will assign the physical id of the underlying hardware to VM on which new VM will be created, (c) Cache Usage Monitor has the ability to check the status and utilization of cache of the underlying physical device and the executing VMS on the fly, (d) and Cache Partitioner (Color-aware Page Migrator) repartitions the cache dynamically according to the requested VM. The high level components of the proposed prevention mechanism is depicted in Figure 4.1.

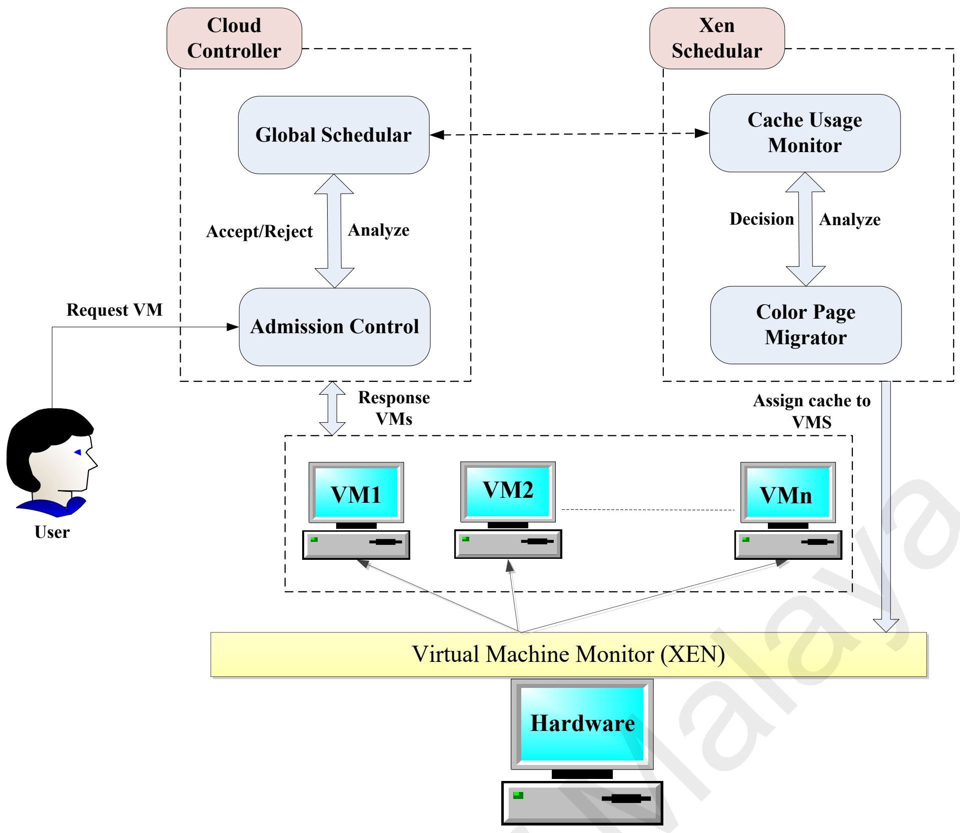


Figure 4.1: Proposed Hypervisor-based Prevention Mechanism Using Dynamic Cache Partitioning

Since our proposed prevention mechanism is based on the VM provisioning (When VMs create and demand cache) and page coloring (assign the separate part of cache to VM) technique. Admission control and global scheduler are based on the VM provisioning, because these components are always activated with VM provisioning. Therefore, we need to describe VM provisioning and page coloring terms according to the requirement of our proposed prevention mechanism in the following section. Moreover, we describe the Xen paging mechanism.

4.2.1 Virtual Machine Provisioning

VM provisioning is a management process for a system that creates new VMs on the physical host server and computing resources are allocated to support these VMs. These computing resources consist the entire cores or CPU cycles, Input/output cycles, storage and memory spaces. Xen enables users to instantiate the guest operating systems (VM) on the fly to execute whatever they desire and require. Furthermore, admission control is performed/activated based on the provisioning or creation of new VM as shown in Figure 4.2. Each VM have to pay in some fashion for the resources it requires. We use this same basic approach to building Xen, which multiplexes physical resources at the granularity of an entire OS and is able to provide performance isolation between them. The task of building the initial guest OS structures for a new domain is mostly delegated to Domain0 which uses its privileged control interfaces to access the new domain's memory and inform Xen of initial register state. VMs sees the allocated space whether the thick or thin allocation is provided by Xen hypervisor. In thick allocation, the whole virtual disc is provided to VMs while in thin approach only the required part of the virtual disk is provided to VMs. VMs sees all the time the allocated virtual disk space but only used the amount of capacity required to hold the current files. These virtual disks are allocated to each VMs on the fly on the physical disk according to the requested user need.

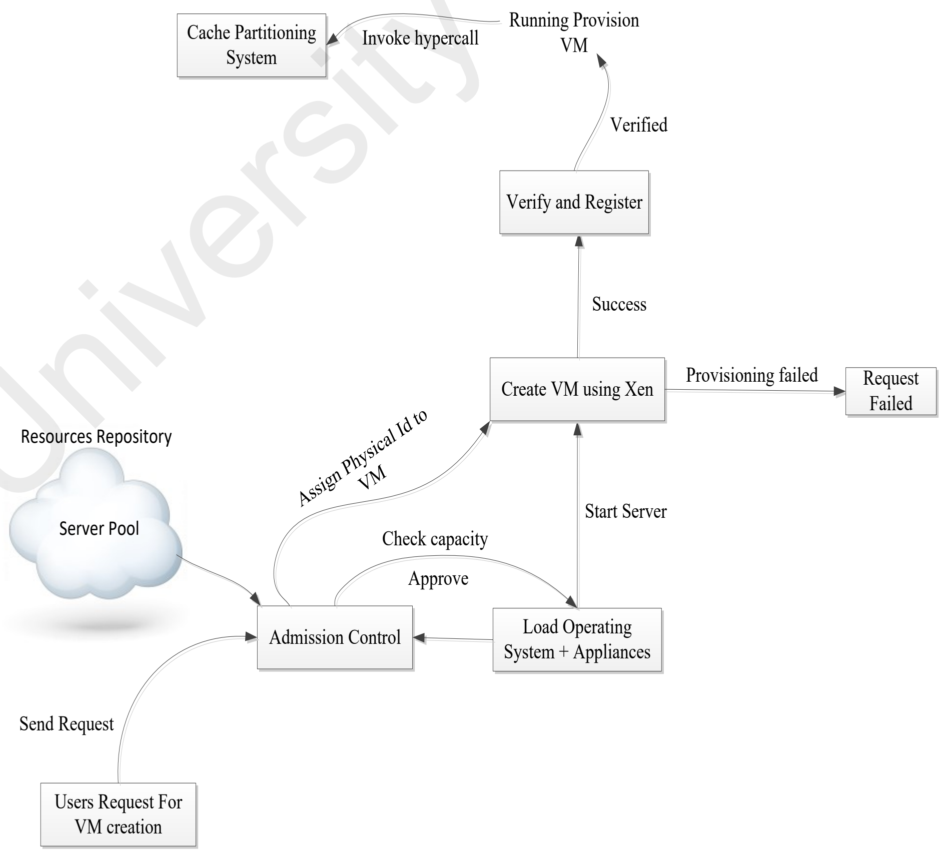


Figure 4.2: Process of VM Provisioning

4.2.2 Page Coloring

Our proposed prevention mechanism using dynamic cache partition is based on the page coloring technique. Page coloring is a classical software based page allocation technique that directs how pages are mapped to the cache memory lines. It is the basis for the fine grain division of LLC namely L2 and L3 for the purpose of a cache hit optimization. Despite, the primary use of the cache coloring as an optimization approach, the particular mapping of the memory addresses to cache lines can be exploited for the security of the system by mimicking VMs isolation across LLC (L2 or L3). In addition, the memory management module is controlled by the page coloring systems to ensure that the group of pages having the same color is assigned to the same cache lines to enforce the security of the system.

In modern OS, the OS access physical memory and L3 cache by physical address. In limited cache associative, there should be overlapped on bit field between physical page number and L3 cache set number. Furthermore, there are some overlapped bit between the set number of cache associative and the page number of machines which are directly controlled by the cache coloring. These bits can be used to group the memory pages into distinct color. For instance, the size of the physical page is 4KB and to represent page offset there must be 12 bit. The remaining bits are assigned to the physical page number. The size of the L2 cache is 512KB, 61-way associative, and the size of a cache line is 64B. So the physical page number has 3 lower bits that are overlapped with the higher 3 bits of cache set number. This overlapped part is called page color.

Similarly, in Figure 4.3, the physical page number has 4 bits that are overlapped with the higher 4 bits of the cache. These 4 overlapped bits show the cache partition into 16 colors. Furthermore, 5 overlapped bit partition the cache into 32 part and for 3 overlapped bits the cache will be divided into 8 partitions. This number can be varied according to cache's associativity, the size of the cache and cache line. In addition, the OS has full control on these overlapped bits. OS decide how and which virtual page to be mapped into which physical page. The hardware by itself fixes the mapping of the physical page into a cache, which is the most important requirement for page coloring technique. OS can use its control of virtual to physical mapping to control indirectly the mapping of physical pages to cache lines. Distinct color could be mapped into distinct cache sets. The steps for the color assignment are as follow:

- Each set has own color
- The same color has to be assigned the pages mapped on the same set
- A VM own one or more color
- Hypervisor assigns to VMs only pages of their own color

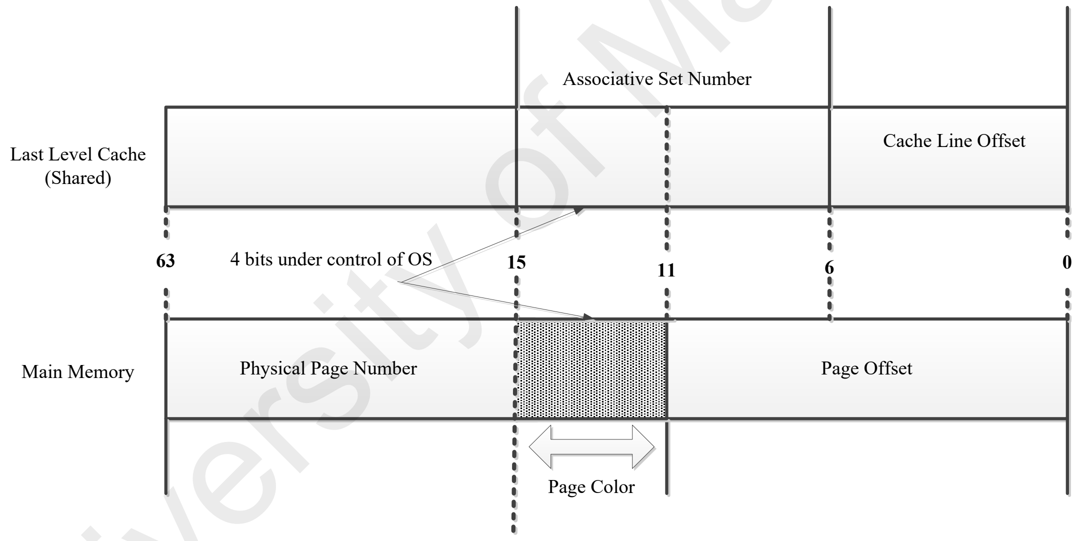


Figure 4.3: Mapping between the Physical Address and Cache Lines (Overlapped Bits are Used for Page Coloring)

4.2.3 Paging Mechanism in Xen Hypervisor

As we discussed in the previous section that our proposed prevention mechanism uses page coloring for the allocation of pages in Xen hypervisor. Therefore, in our thesis, we need to discuss the Xen paging mechanism. In a traditional non-virtualized environment, the OS is responsible for the assignment of physical memory to the running process inside its own virtual address space. During the memory access, the virtual address of process must be translated by the memory management unit to traverse the corresponding process page table set up by the OS. However, in the hypervisor (e.g., virtualized environment) one more indirection of memory translation from guest to host is performed. Xen hypervisor supports para–virtualization and full-virtualization. The performance of para-virtualization is better as compared to full virtualization, but it need change in the guest OS source code. Therefore, for our prevention mechanism using dynamic cache partition, we focused on full virtualization. Under full virtualization mode, Xen is responsible for the translation of three address space namely machines-, guests-, and linear address-space. The machine address-space is called the real machine address-space; while the guest's view of the real machine address space is the guest address space is also known as pseudo-physical address space, and linear address space is provided by processor's MMU is a flat contiguous address space.

As shown in Figure 4.4, there are three memory namely virtual memory, physical memory, and machine memory. Virtual memory is mapped by application inside the guest OS. While in physical memory the host presents physical pages to VMS and actual pages allocated by the host in machine memory. Furthermore, when executing on the hypervisor, the guest OS translates the guest virtual address to the physical address of guest OS, and the hypervisor (VMM) maintain a mapping from the guest physical address to machine physical address. Furthermore, this real memory address is used for accessing the memory. The machine, the guest, and the linear address spaces are manipulated in a unit knows as page frames. Particularly, the frame number of the machine physical address space is called Machine Frame Number (MFN) while guest's pseudo Physical Frame number is known as PFN.

Furthermore, hypervisor detects the same pages in the memory of each guest VM and maps these identical pages to the same physical memory. Hypervisor allocates frame numbers and mapped a unique PFN of guest OS (VM) to a specific MFN (Barham, Dragovic et al. 2003). The Xen hypervisor used the two hardware page table namely guest physical to machine physical translation (P2M) in order to translate the guest physical address to machine physical address and machine physical to guest physical translation (M2P) in order to translate the machine to guest physical address. In addition, Xen hypervisor used the shadow page table for the guest virtual address to host physical address. P2M and M2P are the arrays of frames numbers indexed by either by machine or physical frames, particularly P2M table for the mapping of PFN to MFN while the M2P table is for the MFN to PFN mapping.

In full virtualization mode, the guest OS considers itself as a real machine. Therefore, in page tables the frame numbers that they used to be MFN, which are in fact PFN. The PFN entries filled in the page table cannot be accessed directly without translation to corresponding MFN before the page table be committed to MMU. Xen handles this problem by using a mechanism called Shadow Page Table. OS creates and maintains shadow page table for the original page table in each VM for its virtual address spaces without modification. But MMU hardware does not use these shadow page tables, these tables just for the direct virtual to physical mapping. It uses the TLB for the translation of virtual pages of a guest to machine page of a physical system. These tables are loaded into the hypervisor (VMM) on context switching. VMM keeps its tables consistency with the OS in such a way that it’s V→M consistent with the OS V→P. VMM maps page table of OS as read-only. When OS tries to write to the page table then traps to VMM. VMM applies write to shadow page table and OS page table and returns. This process is called memory tracing. Original page table used by MMU will be locked on the creation of shadow page table for further changes, the effect of every write to original page will be captured and propagated to shadow page. A shadow page pool will be created by Xen for every guest OS, then Xen is responsible for allocating a free page from the shadow page pool to every guest OS which tries to access these pages as page table, or recycles a shadow page to make it a target page shadow based on less frequently used and on no availability of free page. Since it is common for page tables to reference each other, Thereby Xen is responsible to organizes all shadow pages with both page level as keys and PFN into a hash table.

Modern computer architecture has the hardware support for the memory virtualization techniques. For instance, Extended Page Table (ETP) feature is enabled in Intel CPU by Intel VT (Virtualization Technology) (Technology. 2016), the hardware MMU first walk through the shadow page table used by the guest OS for each memory access by each VM to translate from guest virtual address to guest physical address. Then access a separate page table namely ETP setup by the hypervisor for the translation of guest physical address to machine physical address. Thus, the conceptual P2M and M2P tables we have mentioned above just map to the EPT table in the case of Intel architecture.

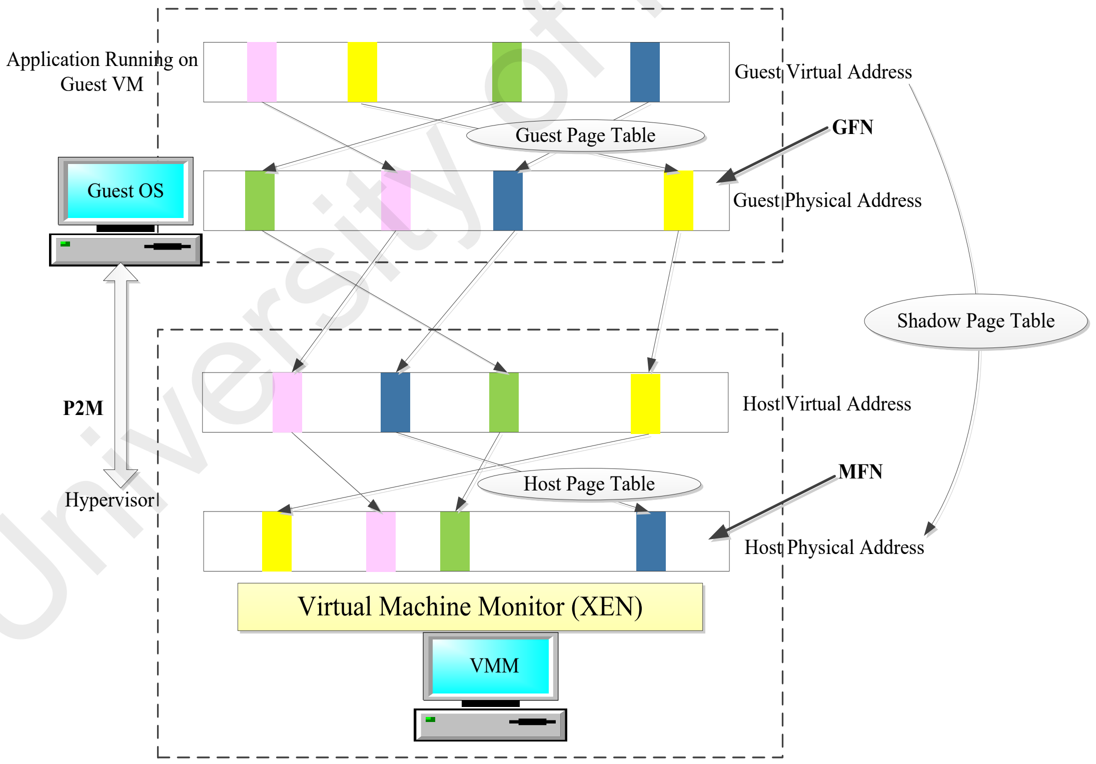


Figure 4.4: Paging Mechanism in Hypervisor

4.3 Components of the Proposed HBP-PDC Prevention Mechanism

The following section describes the components of the proposed prevention mechanism which is admission control, Xen scheduler, cache usage monitor, and cache partitioner. Although our prevention mechanism is mainly based on two components namely cache usage monitor and color-aware page migrator. Cache usage monitor is responsible for assigning initial page colors to the new creating VM and for monitoring VM cache usage metrics. While the responsibility of color-aware page migrator is to allocate page frames of a specific color. However, if the admission control accepts the request for the new VM creation then the global scheduler will be activated and the main components of our HBP-DCP will be activated based on the admission control and global scheduler approval for new VM. Therefore, there is a need to explain these components.

4.3.1 Server Side Admission Control

The responsibility of admission control module is to enforce the fair utilization of server resources. It uses various strategies to decide which user requests to be accepted in order to minimize the performance impact, avoiding the overloading of resources and penalties of service level agreement that decrease cloud provider's profit (Wu, Garg et al. 2012). Admission control regulates the number of active cloud users based on the utilization of the system or the policy manually defined by the system administrator. In order to share the resources among various devices, a request from clients will be processed and queued according to any scheduling policy namely round robin and FIFO defined by the system administrator or selected on the fly based on the system load and other metrics.

Furthermore, whenever the clients send a request for VM, the admission control will communicate with the VMM (hypervisor) whether the VM can be created or not. Thereafter, the admission control phase verifies the software platform availability and analyses if the new request can be accepted then it will decide whether to queue it up in an already initiated VM or by initiating a new VM. Hence, if both conditions are satisfied then the request is transferred and the id of the physical CPU will be assigned to the requested VM. Hence, firstly, the admission control checks if the new request can be queued up by waiting for all accepted requests on any initiated VM. If this request cannot wait in any initiated VM, then the admission control checks if it can be accepted by initiating a new VM provided by the cloud provider. Once the request is granted by the admission control then the global scheduler is responsible for the creation and scheduling of VMs on the specific hypervisor of the underlying physical device from which user request is generated for the VMs creation.

4.3.2 Global Scheduler vs Xen Scheduler

The scheduling and admission control are interlinked for VMs creation and resources allocation to users. The global scheduler is responsible for delivering or rejecting services to every user according to their request based on the admission control decision (Wu, Garg et al. 2012). Once the admission control grants the request, then after this stage, the global scheduler will assign the physical id of the underlying hardware to different VMs on which the user send a request to the admission control for the creation of new VM. The whole process of our proposed prevention mechanism is inter-related with the creation of VMs. The Xen scheduler will check the cache memory after the creation of new VM.

Xen scheduler is the local scheduler on the individual physical machine. Xen being a virtualization hypervisor closely models the OS on which it is run. Therefore, the scheduling of VM in the hypervisor is same as the process or thread scheduling in OS. Just like the process in OS has multiple threads that can be processed on different cores, the VMs have multiple virtual CPUs (VCPUs) that can be run on different physical CPUs (PCPUs). Xen scheduler balances the load of one or more virtual CPU across physical CPU. The basic difference between the OS and hypervisor is that the number of VCPUs is static as compared to the process in OS because VMs and VCPUs are created and deleted on a rare basis. In contrast, the process or thread are created and deleted on a continuous basis. Since Xen scheduler controls the cache memory according to the new requested VMs. Therefore, we change the existing code of Xen scheduler by adding code for monitoring the cache utilization and cache partition module.

4.3.3 Cache Usage Monitor

During the physical system booting all cache memory is allocated to the Dom0 by default. When the other guest VMs e.g., Dom U are executed, then they share the same memory which is allocated to Dom0. The responsibility of cache usage monitor (CUM) is to measure the cache for running VMs upon the creation of new VM and to make a decision about the partition adjustment. Moreover, the responsibility of CUM upon the new VM creation is to monitor the cache utilization, to assign the initial page colors to VM, and to readjust the color assignment according to the requirement of VMs. The CUM, reserve a memory pool for the page coloring during runtime, and partition this memory according to the underlying cache infrastructure into different colors. The memory pool is used in order to serve all page request. The free pages having the same color inside the memory pool are linked together to form multiple list. The CUM divide the cache into N portion of contiguous pages on a physical system having M processing cores. Each cache section is then assigned to a particular CPU core, this specific core will be considered a local core for a color if the color belongs to this core. All the core other than local core will be considered as remote cores. When new VM is created then the CPM will search the core having light weight and allocate the whole cache portion of the core to this VM. Consequently, in a system, if the total number of page color is C then the color assigned to every VM will be C/M. This means that the cache will be fully utilized by N co-running VMs.

One intuitive defense to mitigate cache-based SC attacks is the static partition scheme is to divide the cache into statistical partition according to the different schemes of color assignment and strictly assign distinct page color to different VMs, so they access different parts of cache. Although the static cache partition is simple and provides isolation, it potentially decreases the cache utilization and consequently limits the number of VMs. To mitigate this problem, we extend page coloring with dynamic cache repartitioning capability by adding recoloring mechanism. In this scheme, we first assign a default-sized partition and then gradually increase the size thorough re-partition (e.g., re-coloring). We devised an algorithm for cache usage monitor which is presented as follows.

Algorithm 1 Cache Usage Monitor

```
1:  Input: Current VM, Cache Miss Rate
2:  if New VM creation = True then
3:  Cache Usage Monitor: Pass the cache miss rate of the current VM to cache
    usage monitor function
4:  Function Cache_Usage_Monitor (Cache miss rate)
5:  Assignment = Assignment of (current VM)
6:  if cache miss rate > High-Threshold then
7:     if IsNotShared (Number of VMs do not sharing the same color) = False
    then
8:        IsShared← True
9:        Return
10:    End if
11:    New = Assign_Color (c)
12:    Assignment += new
13:    IsNotShared ← False
14: End if
15: End if
16: End Function
17: Function Assign_Color (number)
18:    New ← ϕ
19:    While number > 0 do
20:       If need-cache() then
21:         new += pick_remote ()
22:      else
23:         new + = pick_local()
24:         end if
25:         number ← number -1
26:      end While
27:      return new
28:  End Function
```

Thus the different parts of the cache will be assigned to different VMs to improve the security of the system without impacting the overall performance of the system. Because every VM can access different cache on a dynamic basis according to their requirements. The system will be secure from the SC attacks because no VM can access the partition assigned to another VM. The purpose of differentiating the VMs sharing the same color to the VMs not sharing the same color is useful for restricting two VMs to not interfere with each other data by assigning the new color which is not already shared to the new created VM. There are two approaches coloring and recoloring. In the cache usage monitor algorithm, the function Assign-Color is triggered to assign a specific color to a new created VM, whenever it run outs of memory with its pre-existing color. Here c is the configurable number of color. Moreover, in cache usage monitor algorithm, the number of VMs using the same color as the definition of IsShared and refer the other one IsNotShared when the VMs do not share the same cache. While the recoloring function in color-page allocator has explained in the following section is invoked when a cache demand exceeds its current assignment. This is determined by observing the ratio of the cache miss rate to the total number of cache accesses by hardware performance counter in modern processor over a period of time. Cache usage monitoring is responsible for assigning an initial color to VM. We set up high-threshold and low-threshold as a global variable for the cache miss rate when the system starts. VMs with the cache miss rate greater than the high-threshold are the one required more space on the cache. While VM with cache miss rate lower than the low-threshold is the ones willing to provide empty space on cache for re-partitioning.

Function cache_usage_monitor in algorithm 1 is used to activate recoloring. It takes the cache miss rate of the existing VMs upon the time of new VM creation. The need_cache() function returns true if current VM has already been using the entire section of the local cache. The number of VMs sharing the same color (IsShared) act as a signal to indicate that a VM needs more page colors. Conversely, the number of VMs do not share the same color (IsNotShared) is used to indicate when more page colors are available. Functions pick_remote select a color in a section of remote cache belonging to the existing VM. Similarly, pick_local choose a color in a local cache section, owned by the current VM. Cache usage monitor considers both the number of cache references and cache miss rate. If the cache reference for a VM is small then the cache miss rate for that specific VM will be considered as zero.

4.3.4 Color-Aware Page Migrator

Color-aware page migrator is responsible for allocating page frame of specific color. Upon receiving an allocation request from VM, the color-aware page migrator communicate with the CUM to determine the colors already assigned to the requesting VM. The color-aware page migrator then gets one of these colors in a round robin manner and returns a page from the memory pool with that specific color. For instance, when a new VM request for a page, the Xen hypervisor (VMM) allocate pages according to P2M table created by CUM in term of page coloring. Therefore, the requested data for individual VM will be placed in separate cache lines, this improves the security because one VM cannot access or evict the data of another VM. In our prevention mechanism, a memory pool is used to handle all the request from various VMs. Inside the memory pool, free pages having the same colors are linked together to make multiple lists. The per-VM color assignment is done in the cache usage monitor in order to reduce the complexity of our prevention mechanism.

Once the admission control approves the request for new VM creation then upon the receiving request of allocation for new VM, the color-aware page migrator (CPM) communicate with the CUM to find the color already assigned to the created VMs. The CPM then select one of these colors in a round robin manner and return a page of that specific color to the requesting VM from the memory pool. If the memory pool does not consist that requesting color then the page migrator assigned another color to the VM. If the requesting color is not available then the page allocator of Xen hypervisor populate the memory pool and organize the free pages in a machine using a buddy system, similar to the Linux memory management mechanism. To implement our prevention mechanism we have added our color-aware page migrator in addition to buddy system for memory management. Using these color bits, the Xen hypervisor can make the color-aware page migrator to control the mapping between physical pages and cache memory and can assign a specific color page to the requesting VM.

4.4 HBP-DCP Prevention Mechanism Algorithm

Section 4.3 explains the basic building blocks of the proposed prevention mechanism. In this section, we will present our proposed algorithm which starts from the interaction between the components of the proposed prevention mechanism. The existing prevention algorithm change the cache configuration at boot time based on the new VM creation. Our proposed prevention mechanism is different from the static page coloring method in that it allows cache usage of adjusting VMs on the fly. We achieve the dynamic cache partition between VMs by changing the physical location of a VM's logical page through a set of hypercalls in the Xen source code. The dynamic cache partition algorithm is based on the cache coloring approach which has already implemented in the OS for page allocation (Tam, Azimi et al. 2007). However, the basic steps in generic dynamic cache allocation can be summarized in the following steps:

1. Wait for new VM request.
2. Admission control approves the request for new VM creation and send hypercalls to hypervisor.
3. The cache usage monitor component in hypervisor is responsible to get the current partition of cache and the number of currently executing VMs.
4. Reconfigure existing VMs by shrinking its cache size through page recoloring.
5. Register a new cache partition for new VM with Xen hypervisor.

6. Create the requested VM with the new cache partition on dynamic basis and assign a color page to the VM which represent a unique cache line.
7. Consequently, it prevents the cache-based SC attacks to occur because every VMs get individual unique partition in the cache and is unable to access and evict each other data.

Based on these generic steps and the components of our proposed solution, we present our proposed cache usage monitor and dynamic cache partitioning algorithms to serve the end users as algorithm 1 and algorithm 2 for the prevention of cross-VM cache-based SC attacks.

Algorithm 2 Dynamic Cache Partitioning Algorithm

```
1:  Input: L3 cache detail, VM-ID
2:  Include Xen memory header file
3:  While(1)
4:  Input (src_mfn, dst_color)
5:  Allocate a free machine page from the cache usage monitor
6:  New free machine page ← dst_mfn
7:  Store (src_mfn in SPT)
8:  Remove all write permission in SPT where each entry pointing to src_mfn
9:  Enable → dirty page
10: If guest tries to access and modify src_mfn
11: then mark page as ← dirty page
12: Before starting the above steps lock the SPT or activate shadow lock
13: if (Copy src_mfn to dst_mfn)
14: for each copy
15: check →whether the content is changed during the copying process
16: if (unchanged) then
17: Update P2M and M2P mapping
18: End While
```

Figure 4.5 shows the flow of our HBP-DCP prevention mechanism. As shown in the figure when the user request for VM creation then the responsibility of the admission control in combining with the global scheduler is to assign the physical id on a physical machine to that created VM. Once the admission control assign the physical id then the global scheduler send the hypercall to the Xen scheduler. Now VM is created, the responsibility of cache usage monitor in our HBP-DCP mechanism is to monitor the existing running VMs and their cache utilization. Now the responsibility of color-aware page migrator to repartition the entire cache according to the new created VM. For instance, if currently one VM is running then the entire cache is assigned to that VM. Now if 2 more VMs are created and the demand of each VM for cache is different. Then HBP-DCP mechanism monitor the entire cache and repartition the entire cache according to the demand of 3 created VMs.

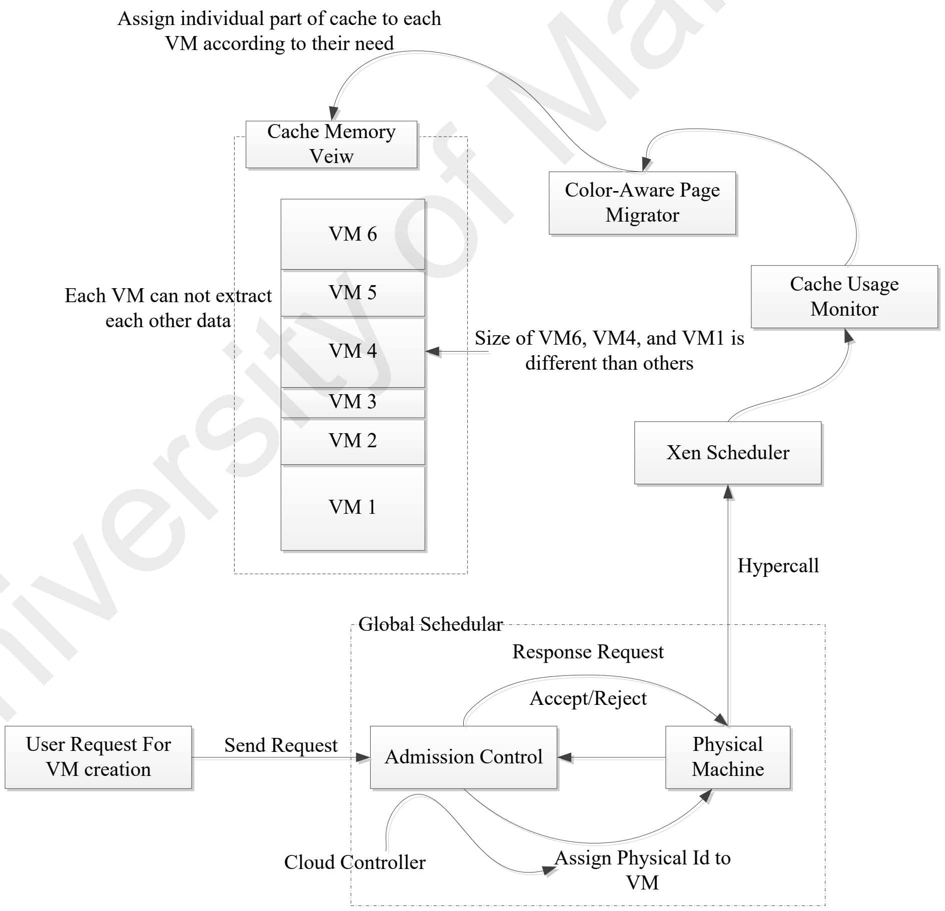


Figure 4.5: Flow of the Prevention Mechanism Using Dynamic Cache Partitioning

4.5 Data Design

In this section, we present the features of our performance evaluation system namely performance evaluation parameters and methods. We introduce and describe our performance evaluation parameters. These parameters are selected to evaluate and efficiently analyzes the lightweight characteristics of our proposed prevention mechanism. In addition, we describe the methods to evaluate and validate the performance of the proposed prevention mechanism.

4.5.1 Performance Evaluation Metrics

In this section, we describe the evaluating process of our proposed prevention mechanism. It describes the criteria by which we evaluate the effectiveness of our solution and the environment for conducting our experiment. We also describe the parameters or metrics by which we evaluate our proposed prevention mechanism and compare to the existing state-of-the-art prevention mechanism. These parameters are used to determine under what conditions our proposed HBP-DCP prevention mechanism for cross-VM cache-based SC attacks can be practically implemented into a commercial cloud environment. Table 4.1 shows the evaluation parameters with the measurement unit.

Table 4.1: Metric for Performance Evaluation of the proposed Prevention Mechanism

Evaluation Metrics	Description	Unit
Load Testing	To evaluate the performance of a system in term of generated overhead from normal (low) to peak (high) load to find peak for the system	Seconds
Cache Usage	To calculate cache usage by measuring performance of the memory hierarchy more specifically the level (L1, L2, and L3) of cache	MB/Sec
Memory Access Rate	To calculate the time required to access data from memory	Nanoseconds

(a) Load Testing

We measure the performance of our proposed prevention mechanism under the normal and peak condition and consider the load testing which is mostly used method for performance testing. The load testing of any system can be better expressed by the maximum amount of work a system can handle without performance degradation. These load testing can be done by many methods, however, we used Apache benchmark explained in next chapter which is open source and easily benchmarks for the cloud.

(b) Cache Usage

We measure the performance of the memory hierarchy (e.g., cache usage) of the system after deploying our prevention mechanism, specifically our focus is to parameterize and evaluate the performance of cache level (e.g., L1, L2, and L3) present on and off the processor. The performance of the system means that how much raw bandwidth in megabyte per second after the dynamic partition of cache.

(c) Memory Access Rate

Memory access time is calculated during translation of guest virtual pages to machine physical page. This is the total time, the computer takes to read data from a storage device such as computer memory, physical memory, and cache or another mechanism. The unit of measurement for memory access rate is commonly nanoseconds or milliseconds. If the memory access time for any instruction is low then it is considered to be a better access time as compared to high access time. For instance, if the memory access time is 10ns for reading 100MB then it is considered faster than the 50ns for accessing the same data.

4.5.2 Data Collection Tool

Although there are different methods and tool for generating and analyzing a load of a system and also the data can be gathered by using different approaches. However, in order to analyze the load of a system, we used Apache benchmark. In which we check that a system can handle how many numbers of requests per second in a modified and unmodified hypervisor. Similarly, there are a number of programs and benchmark for extracting the cache usage, however, cachebench was expected to give more detailed cache usage. Auto-generating of load testing and cache usage is beneficial in order to avoid man-made mistakes and improving data integrity, accuracy and data reliability at the stages of analyses and synthesis. The memory access rate and boot time are collected by using our customized designed program. The program calculates and analyzes data gathered by measuring the number of cycles required to access a set of memory addresses.

4.5.3 Performance Evaluation Method

The performance of this prevention mechanism is evaluated using benchmarking experiments on the modified (static and dynamic partitioned) and unmodified Xen hypervisor and on different numbers of VMs. The data are collected for analysis by using many synthetic workloads. The statistical model is used to validate the results of our benchmarking experiments. We developed the statistical model using the independent replication model to train the regression model. The split-sample approach is used to validate the identified statistical model. The validated model are used to generate the load testing, cache usage, and memory access rate. Data analysis and synthesis testify the proposed prevention mechanism performance.

4.6 Conclusion

In this chapter, we presented our proposed HBP-DCP prevention mechanism. The basic objective of the HBP-DCP mechanism to prevent the cross-VM cache-based SC attacks. HBP-DCP consists of cache usage monitor and color aware page migrator that monitor the cache after VM creation and assign a specific part of cache to each VM. These VMs will not interfere with each other data nor evict each other data. Moreover, HBP-DCP mechanism partitions the entire cache dynamically according to VM creation. If we partition the entire cache statically during boot time, then this will degrade the cache usage and consequently the overall performance of the system. For instance, if there is one VM in the running state but the cache is already partitioned into 4 equal parts during boot time. Then this running VM will get the one small part of the entire cache while the other three parts will be idle or wasted. As a result, the cache utilization will be degraded and will affect the overall performance of the system. To solve this problem prevention mechanism based on dynamic cache partition is proposed in our solution (HBP-DCP) which will decide to partition and assign the cache to different VMs according to their requirement on a dynamic basis.

CHAPTER 5: EVALUATION

In this chapter, we describe the performance evaluation approaches to evaluate the proposed our modified dynamic partitioned hypervisor based on HBP-DCP mechanism for cross-VM cache-based SC attacks based on dynamic cache partitioning. For this purpose, we analyze the performance difference between unmodified (default/insecure), static partitioned and our modified (dynamic partitioned/secure) hypervisors by considering three performance metrics, namely load testing, cache utilization, and cache access rate. The main motive of this chapter is to discuss and analyzes the data collection methods, experimental setup, evaluation parameters, and to analyze the performance of the proposed algorithms. To evaluate the proposed prevention mechanism and its lightweight features, we utilized standardized synthetic benchmarking experiments namely Apache benchmark and cachebench benchmark from the Phoronix test suite and also cachegrind benchmark. In addition, we also used customized program to measure the average number of cycles needed to access a set of memory addresses or the average memory access time. The evaluation process also describes that how the results were conducted and how many observation is performed in order to evaluate the proposed prevention mechanism. Moreover, the data collection method for the proposed HBP-DCP prevention mechanism is also described.

The evaluation results are validated through statistical modeling. We have used independent replication method in order to build our statistical model and validate the proposed prevention mechanism using split-sample approach. In another set of experiments, we build a separate test-bed for the comparison of our proposed prevention mechanism using dynamic cache partition to the static cache partition to describe the lightweight features of our mechanism. Finally, we demonstrate the statistical data analysis methods used to analyze and synthesis the results. The rest of this chapter is organized as follow: Section 5.1 presented the process of evaluation at a high level.

Section 5.1.1 described the experimental method along with the data collection and data generation method. The benchmarks and their input data are listed in Section 5.2. In Section 5.3, the evaluation method is presented to explain how the statistical models are validated. Section 5.4 described our parametric analysis in terms of bearable load, cache usage, and memory access time by each hypervisors and demonstrated the statistical data analysis observation used in this thesis to evaluate and synthesis the result. Finally, Section 5.5 conclude the chapter.

5.1 Evaluation Process

The proposed prevention mechanism (HBP-DCP) is designed to prevent the cross-VM cache-based SC attacks using dynamic cache partition. In this prevention mechanism, the page coloring technique is implemented to divide the cache dynamically according to the new VM request and demand. An efficient algorithm is developed to measure the cache for new created VM and assign the specific color to the new requesting page of VM. Then the color-aware page migrator component in the (HBP-DCP) receive these input and allocate a separate partition in the cache to the new requesting VM on the fly. Multiple compute intensive benchmarks application are selected from the Phoronix test suit to analyze and evaluate the performance of the system. Since our solution is based on the hypervisor source code, thereby, the data is collected is to run the application benchmark in the unmodified, static partitioned, and modified (dynamic partitioned) hypervisors. For load testing of the three hypervisors, we used Apache benchmark. For evaluating the cache utilization of unmodified, static partitioned and dynamic partitioned hypervisors, the data is collected through the cachebench benchmark. Similarly, the memory access rate of unmodified, static partitioned, and our modified (dynamic partitioned) hypervisors are evaluated through collecting data through our designed customized benchmarks. The evaluation of our secure HBP-DCP hypervisor is based on acquiring the answers to the two research questions: Does our secure hypervisor prevent the cross-VM cache based

SC attacks and what is the performance difference between our secure (modified/dynamic partitioned) and insecure (unmodified/default) and static partitioned hypervisors. In order to answer these questions, we have simulated a single server cloud environment. We have conducted a cross-VM cache-based SC attack by using PTP technique in this environment. Moreover, we have subject both static partitioned and our dynamic partitioned hypervisor to a series of workload under different configuration for the purpose of checking system behavior. The resulting completion indicates system overhead.

5.1.1 Experimental Setup

In this section, we explain the needed hardware and software for conducting our experiment and describe the methodology used to benchmark the unmodified, static partitioned and our modified (dynamic cache partitioned) hypervisors when collecting the data about load testing, cache usage, memory access pattern. We conducted real-time experimental analysis for the evaluation of our proposed prevention mechanism. There are various reason behind the utilization of real-time experiments. Firstly, in the field of CC environment, simulation tools are not much mature and is unable to provide the technical capabilities to conduct the research work of this nature. Secondly, simulation tools simulate the real time metrics and consequently generates probabilistic estimation and more overhead. Therefore, simulation tools are more vulnerable to result in estimation that can lead to low accuracy. Moreover, the real-time analysis provides the detailed knowledge of the system evaluation parameters that affect the performance of the prevention mechanism.

Our performance evaluation for the proposed prevention was conducted on a machine with Intel processor i7 having the quad-core processor and one hardware thread per core and 4GBytes memory. We created two VMs namely the victim VM1 and attacker VM2. These two co-located VMs can evict each other data to form a cross VM cache-based SC attack by sharing the same hardware. We have solved this problem by using HBP-DCP mechanism to prevent the two co-located VM from being able to evict each other data from the cache by using dynamic cache partition. By using HBP-DCP we have assign separate part of cache to each VM, so these two VMs are unable to access each other cache. By assigning such partition, each VM would be able to use a part of the shared cache without interfering with each other to ensure that no VM outside of its partition can access the cache lines. To comply with the cloud model our dynamic partition would need to be implemented entirely through software means.

The studied literature showed that some authors have done static cache partition, however, this static partition degrades the cache usage that further degrades the overall performance. Some authors have done dynamic cache partition, but that solution would need to be required the clients to change their software's or the underlying hardware which does not obey the cloud model. Therefore, we need a prevention mechanism based on dynamic cache partition that obey or comply with the cloud model. Since our prevention mechanism is hypervisor-based, we changed the source code of Xen hypervisor by implementing our solution in the page allocation algorithm of Xen memory management. Now we need to evaluate our proposed prevention mechanism performance in term of load testing, cache utilization, and memory access rate by analyzing the benchmark results in the unmodified, static partitioned, and our HBP-DCP (dynamic partitioned) hypervisors. To evaluate the performance of the proposed prevention mechanism, we selected two standard and different synthetic benchmark. The selected synthetic benchmarks are Apache and cachebench from the Phoronoix test suit for evaluation of load testing and utilization of cache. These benchmarks have been discussed in Chapter 3 in detail for the purpose of problem analysis.

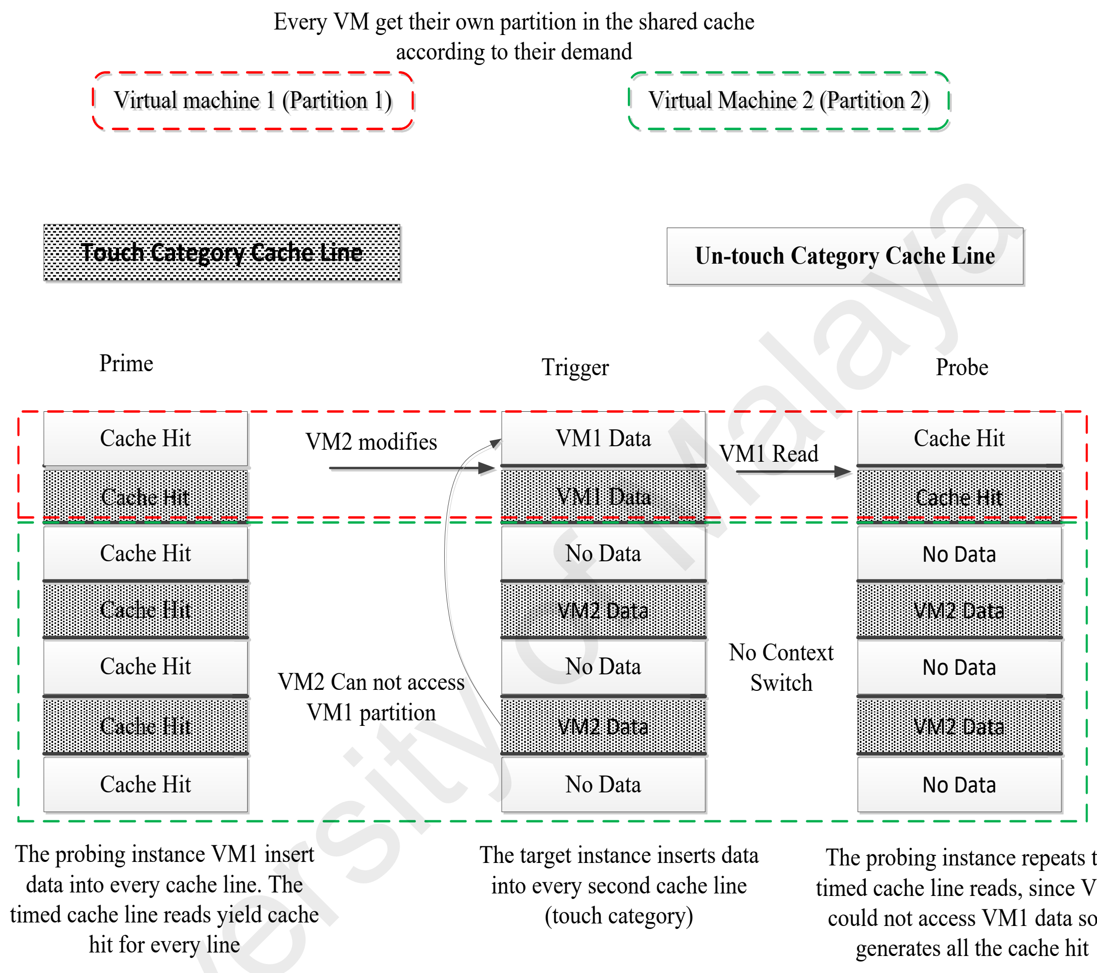
The primary data for evaluating the performance of our proposed prevention mechanism is gathered by conducting the experiments in three scenarios. In the first scenario, the attacks and the benchmarks are executed in the unmodified or default hypervisor. In the second scenario, the attacks and the benchmarks application are executed in the modified hypervisor is known as a statically partitioned hypervisor. Finally, in the third scenario, the attacks and the benchmarks are executed in our modified hypervisor is referred to as dynamic partitioned hypervisor. In our solution, we changed the source code of the open source customized hypervisor. The outcome of our solution is a new hypervisor which dynamically assigned the cache to individual VM based on their cache requirement. Consequently, no two VMs are able to evict or extract each other data nor disrupt each other communication. The limitations of the static partition can be solved here by assigning the cache partition during runtime based on the need and number of executing VMs.

5.1.2 Effect of our HBP-DCP based Hypervisor on the Cross-VM SC Attack

The hypervisors (Unmodified, Static partitioned, and our modified based on the dynamic cache partition (HBP-DCP)) are evaluated under the same cache-based SC attack. The cache-based attacks were conducted by using prime + probe and flush & reload methods as we explained in detail in Section 3.2.3 in Chapter 3. We have implemented cache-based SC attack in three scenario namely in native OS, in single VM, and across-VMs in virtualized environment. In the PTP technique, the side channel receiver and sender programs are installed on two separate guests VMs. The receiver program is called the probing instance while the receiver program performs the function of the target instance. Both sender and receiver programs were executed simultaneously by co-located VMs on the test-bed machine and pinned to separate CPU cores such that L3 cache could not be used as an SC attacks. The attack was conducted between the target (victim) and the probing instance (attacker) by sending an identifiable string of 160 bits from the target to probing. In order to verify the consistency, the attack was executed on each hypervisor ten times. Similarly, in order to conduct the cross-VM cache-based SC attack by using the flush & reload technique, the flush & reload process (attacker) is installed in one VM and the AES process (victim) is installed in another VM. Both attacker and victim processes were executed simultaneously by co-located VMs on the same physical machine and pinned to separate CPU cores. Consequently, the attacker extract the encryption key of AES algorithm which we have explained in the chapter 3 in detail.

The SC attacks were given an ideal condition for working to verify the secure (HBP-DCP) and insecure hypervisor ability. Specifically, the Dom0, target instance, and the probing instance were pinned to separate CPU cores and were the only VMs running on the hypervisor. This configuration depicts the best possible condition for cache-based SC attacks. Any variation in this setup would prevent the attack success. Our experimental analysis concludes that if an attack can be prevented under these favorable conditions then the same prevention mechanism would work for the environment more unfriendly to the success of attacks. The viability of the implemented prevention mechanism should not be affected by these configurations. The experimental results of our proposed prevention mechanism on the PTP and F&R techniques are presented in Figure 5.1. Dynamic cache partition as a solution would be able to assign different partition of shared cache to the individual VM so each VM would be unable to access each other data as shown in the Figure 5.1.

As we see in Figure 5.1, there are two VMs accessing the shared cache, so our prevention mechanism divides the cache into two partitions according to the requirement of each individual VM. The red dotted line part is reserved for VM1 and green dotted line shows the VM2 partition. For instance, VM1 is the probing instance and the partition of VM1 maps to the first two cache lines and the partition for VM2 (Target instance) maps to the last six cache lines out of the eight shown. When the probing instance tries to access and prime the cache lines it would be able to access the partition of the first two cache lines which has already been assigned to it and cannot access the other six lines. Now when the target instance tries to access and modify the cache lines it can just access their own assigned six partitions and therefore unable to evicting VM1's data from the cache lines.



Every VM get their own partition in the shared cache according to their demand
Virtual machine 1 (Partition 1)
Virtual Machine 2 (Partition 2)
Touch Category Cache Line
Un-touch Category Cache Line
Prime
Trigger
Probe
Cache Hit
Cache Hit
Cache Hit
Cache Hit
Cache Hit
Cache Hit
Cache Hit
VM2 modifies
VM2 Can not access VM1 partition
VM1 Data
VM1 Data
No Data
VM2 Data
No Data
VM2 Data
No Data
VM1 Read
No Context Switch
Cache Hit
Cache Hit
No Data
VM2 Data
No Data
VM2 Data
No Data
The probing instance VM1 insert data into every cache line. The timed cache line reads yield cache hit for every line
The target instance inserts data into every second cache line (touch category)
The probing instance repeats the timed cache line reads, since VM2 could not access VM1 data so it generates all the cache hit

Figure 5.1: Effect of Dynamic Cache Partition on the PTP technique

When the probing instance once again tries to access cache lines it would see no difference from when it is left and therefore no communication would occur. Our prevention mechanism is more than preventive as compared to reactive, it prevents the cache-based SC attacks from occurring rather than reactive response after occurring the attacks. Since there are two types of response for attacks namely preventive and reactive response. In prevention response, the attacks would be able to prevent at the start before occurring and it would not be occur. While, in reactive response, once the attacks occur then the attacks would be cured or blocked after occurring. Therefore, the preventive response is better than reactive response because once the attack occur it can damage the system or extract the confidential information in a very short time.

If a VM is given a cache of half the size but does not have to worry about data being evicted from it by other VMs then it may end up yielding a greater cache hit/miss ratio. We are partitioning the entire memory pool to guarantee the complete security of the system as compared to the work done by Shi et al. (Shi, Song et al. 2011) which have attempted to partition a small memory of the cache and given a portion of memory by using secure color to the encryption algorithm. By default, the hypervisor's memory is not bound to a specific partition as we are not aware of any side-channel attack that targets the hypervisor. However, this could be easily implemented using the same technique.

Table 5.1 describes the correctly recovered whole key in number of bytes in both single VM and cross-VM. Single VM means that attack is conducted in single in which the attacker and victim programs are in the same VM. In cross-VM scenario, both the attacker and the victim programs are in different VM and in different cores.

Table 5.1: Comparison of Correctly Recovered Key in Single and Cross-VM in Unmodified Hypervisor (Insecure/Default)

In Single Virtual Machine (Single-Core)		**In Cross-Virtual Machine(Multi-Core**	
Number of requested encryption	Number of correctly recovered key bytes	Number of requested encryption	Number of correctly recovered key bytes
10,000	1	30,000	2
90,000	6	60,000	2
130,000	10	100,000	4
150,000	10	200,000	8
200,000	13	260,000	9
250,000	13	300,000	11
260,000	14	350,000	12
265,000	14	450,000	13
270,000	15	500,000	15
275,000	16	650,000	16

Table 5.1 shows that the required number of requested encryption for correctly recovered the 16 bytes of key in single VM is 275,000 and in cross-VM is 650,000. Since in cross VM the external noise effect the results, therefore, the number of requested encryption in single VM is less than as compared to cross-VM. We believe that due to noise SC attacks require a high number of encryption in the cloud environment as compared to non-cloud environment.

The cross-VM cache-based attack is conducted in the modified (Secure/HBP-DCP based) hypervisor. The evaluation result of cross-VM cache-based attacks is shown in Table 5.2. The result in Table 5.2 describes the correctly recovered key in number of bytes in both single VM and cross-VM in the presence of our HBP-DCP prevention mechanism. Single VM means that attack is conducted in single VM in which the attacker and victim programs are in the same guest VM/operating system.

Table 5.2: Comparison of Correctly Recovered Key in Single and Cross-VM in Modified Hypervisor (Secure/Dynamic Partitioned/HBP-DCP)

In Single VM (Single-Core)		**In Cross-VM (Multi-Core)**	
Number of requested encryption	Number of correctly recovered key bytes	Number of requested encryption	Number of correctly recovered key bytes
20,000	0	20,000	0
60,000	0	70,000	0
100,000	0	100,000	0
130,000	0	200,000	0
180,000	0	280,000	0
200,000	0	300,000	0
250,000	0	350,000	0
265,000	0	475,000	0
270,000	0	520,000	0
275,000	0	650,000	0

As shown in the Table 5.2, in cross-VM scenario, both the attacker (Flush & Reload) and the victim (AES) programs are in different VMs and in different cores. The correctly recovered key in both cases is zero. This zero byte result shows that our HBP-DCP prevention mechanism is capable to prevent cross-VM cache-based SC attacks. Because by implementing our HBP-DCP mechanism, no VM has the ability to communicate with each other for the purpose to leak confidential information.

5.2 Benchmark Applications

This section describes the evaluation process of our HBP-DCP solution. It presents the criteria by which we evaluate the effectiveness of our proposed solution and the environment in which we conducted the experiments. We also describe the evaluation metrics by which we compare our proposed solution to the existing state-of-the-art prevention mechanisms. To evaluate the performance behavior of our proposed prevention mechanism, we have considered two synthetic benchmarks namely Apache and cachebench. We also ustilized two customized benchmarks namely: one is a program for checking the memory access time and the other is a synthetic compute intensive program with the different granularity of execution input.

We have discussed these benchmarks in Chapter 3 also because the same benchmarks were utilized for the evaluation of performance of the static cache partitioning mechanism. Selected benchmark applications investigate the load that modified (dynamic partitioned) hypervisor can tolerate, the cache utilization, and the memory access rate after partitioning the cache dynamically. These evaluation metrics are used to determine under what condition our solution can be practically implemented into a commercial cloud environment. There are different reasons behind choosing these benchmark application. Firstly, selecting the benchmark applications, it is ensured that the chosen benchmark is an open source. Secondly, our solution is based the on Xen hypervisor source code (coded in C/C++). Therefore, it is considered during the selection process that the selected benchmark is coded in C/C++ language. According to the above constraints, we have selected Apache benchmark[1], cachebench[2], cachegrind[3] and customized benchmark namely memory access time.

5.2.1 Apache Benchmark

The Apache benchmark is a standard command line program used as HTTP web server benchmarking tool. Apache benchmark was chosen because it is open source, commonly available, frequently used a benchmarking service, and mostly used for web services one would see as cloud-based applications. In addition, this benchmark was selected for performance experiments because we believe that being a robust benchmark it constitutes a credible Cloud workload. Apache benchmark is used to fire requests to a server in order to find that in how much time and how fast the server could process these requests. We used the apache benchmark to analyze the performance difference between our modified based on dynamic cache partitioning, static partitioned, and unmodified hypervisors in term of load testing.

5.2.2 Cachebench Benchmark

Since our HBP-DCP prevention mechanism is based on the dynamic cache partition that is directly related to the cache. Thereby, in order to check the impact of our solution on the cache usage, we must choose a benchmark which gives a more detailed information about the cache usage. In our case, cachebench is a more suitable benchmark because it is designed for the cache usage description. Cachebench is a synthetic benchmark designed to evaluate the performance of the memory architecture and also to empirically parameterize the performance of cache levels namely L1, L2, and L3 present on and off the CPU processor. The performance is calculated in term of raw bandwidth in megabytes

[1] https://en.wikipedia.org/wiki/ApacheBench

[2] https://openbenchmarking.org/test/pts/cachebench

[3] http://valgrind.org/info/tools.html

per second such as cache read/modify/write, cache read and cache write bandwidth. The objective of this benchmark is to establish high computation rate which gives the optimal cache usage and to verify the effectiveness of compiler optimization. Moreover, the purpose of using this benchmark is to verify the memory footprint of our proposed prevention mechanism because the requirement for many application depends on the resources in term of memory footprint. We check the memory footprint in term of the cache hit and cache miss. Thus this benchmark gives us a good basis for our proposed prevention mechanism performance.

5.2.3 Cachegrind Benchmark

We have used the cachegrind benchmark from the valgrind test suit for conducting the data about cache miss and cache hit rate of all level cache including L1, L2, and L3. These cache miss and cache hit is then used by our designed program to calculate the average memory access rate and the effective memory access rate in case of our modified (dynamic partitioned), static partitioned, and unmodified hypervisor. The parameters along their measurement unit calculated as a result of these benchmarks are shown in the following Table 5.3 in detail. Since our prevention mechanism is based on the partition of cache memory, therefore by using this benchmark we analyzed the result of cache access rate and memory access rate by executing matrix program in the modified, static partitioned, and unmodified hypervisor.

Table 5.3: Parametric Evaluation with Benchmarking

Factor	Parameters	Calculated by	Unit
Load Testing	To calculate the load of modified (partitioned) and unmodified hypervisor in request per seconds	Apachebench	Seconds
Cache Utilization	To calculate bandwidth of a memory by changing array sizes in MB/s	Cachebench	MB/Sec
Memory Access Time	To calculate the time in nanoseconds required to access data from memory	Customized Benchmark + Cachegrind	Nanoseconds

5.3 Evaluation methods

In order to analyze the reliability and validity of our research, several statistical analyses are performed on the collected data through benchmark tools and executing experiments in a different scenario. A statistical model is used to represent and analyze generated data by an average and a standard deviation. The statistical model always implies dependent and explanatory variable. Computation behind the statistical modeling allows us to show the significance of our research. We present each of the statistical methods that are used in this research in the following section.

5.3.1 Descriptive statistics

The descriptive statistics is used in this research in order to analyze data and to highlight the significance of achievement of our modified HBP-DCP based hypervisor in terms of cache utilization and prevention capability as compared to the static partitioned and unmodified (insecure) hypervisors. In descriptive statistic, minimum, maximum, mean and the standard deviation are determined. The desire descriptive data is acquired based on the collected data are summarized in the graphical and tabular form to accomplish the desired objectives.

5.3.2 Confidence Interval

According to the sample central limit theorem, approximately 95 % of the sample means fall within 1.96 standard deviations of the population mean, showed that the sample is greater than or equal to 30 ($n \geq 30$). Therefore, all the experiments in this research are executed 30 times for the performance evaluation of individual variable to verify that the obtained value is under one of the representative samples. In the data sample, the measurement of the central tendency of each experiment is calculated based on the sample mean (-X), for the reason to discover that sample mean is a better point estimate of the population mean as compared to median or mode. Data sampling includes a range of intervals determined from the specified confidence level, a statistics, and the factor of sampling error; hence the sample mean can differ from the population mean. The level of confidence is the probability that the parameter is truly captured by the confidence range. The most common Confidence Levels (CL) are 90%, 95%, and 99%. Therefore, the interval estimate of each sample is determined in order to signify the goodness of the calculated point estimate. The interval estimate for each sample mean of the primary data is calculated with approximately 95% confidence interval of the sample means within 1.96 standard deviations by using the following equation. Therefore, for reporting the parametric results we raise the readability and confidence of the results up to 95%. Equation 5.1 is used to calculate the margin of error in the sample (Intervals 2004).

$$\mathrm{M} = \mathrm{Z} * \left(\frac{\sigma}{\sqrt{n}}\right) \tag{5.1}$$

Whereas, M is the margin of error and Z indicates the value based on the confidence interval percentage and σ is the standard deviation and n is the number of samples. Equation 5.2 is used to calculate the confidence interval estimates for each sample mean (X) of the primary data with a 95% confidence interval (Intervals 2004).

$$\mu = \overline{\mathrm{X}} \pm 1.96 \left(\frac{\sigma}{\sqrt{n}}\right) \tag{5.2}$$

Whereas, σ is used to indicate the standard deviation in the sample values and n shows the size of sample space.

5.3.3 Paired Samples T-Test

In this research, we performed the Paired Samples T-Test to ensure that there is a significant difference between the mean values of the identical measurement performed in three different hypervisors namely unmodified (insecure), static partitioned, and modified (dynamic partitioned–based, the case of our solution) execution modes. In our study, the unmodified, static partitioned and the modified (dynamic partitioned) hypervisors parametric values are paired data of the same workload into three different execution modes. We use this test to ensure that the execution modes of the unmodified, static partitioned, and modified (dynamic partitioned) hypervisors have a significant impact on the load, cache utilization, and memory access time or not. In other words, we can conclude with the help of the generated results from the Paired Sample T-Test that the bearable load, cache utilization, and memory access time in the unmodified (insecure) static partitioned, and modified (secure as a case of our solution) hypervisors modes have a significant difference. Furthermore, our modified HBP-DCP based hypervisor has the ability to prevent cross-VM cache based SC attacks in the cloud environment.

5.3.4 Linear Regression

In this section, we explain our statistical analysis modeling. Using the statistical model results, we can verify and validate the results of the conducted experiments in this research work. We produce the statistical modeling of our performance parameters including load testing, cache utilization, and memory access rate by employing the independent replication method to generate independent datasets. These datasets consist of load testing, cache utilization, and memory access rate for the new independent workload in the unmodified (default/insecure), static partitioned and modified (dynamic partitioned/HBP-DCP) hypervisors.

Moreover, we train the linear regression model to identify the correlation between the load and the transferred number of request per seconds as well as between the cache size and memory bandwidth in term of memory access rate. These regression models are used to generate the load, cache utilization, memory access rate to validate the findings of the performance evaluation parameters generated via experimental analysis. We leverage split-sample approach and perform calibration-validation exercise to validate our regression model. Therefore, partial datasets are used to build and train the model and the remaining for validation of the model. We randomly split the sample into two different size samples to perform validation and identification of the correlation between dependent and independent variable. The model is valid in the case if the result values support each other. The following section describe the parametric evaluation

5.4 Evaluation Metrics

In this section, we present the data collected in a number of experiments by using the aforementioned benchmarks for the evaluation of the modified hypervisor based on the proposed prevention mechanism (HBP-DCP) for the cache-based attacks across VMs. The data are presented from the perspective of performance metrics (i) load testing, (ii) cache utilization, and (iii) memory access time in three different scenario, namely (i) conducting of attack in the static partitioned hypervisor, (ii) conducting of attack in the unmodified (default/insecure) (iii) conducting of attack in the modified (dynamic partitioned/ HBP-DCP) hypervisors. The experimental setup used in benchmarking analysis is shown in the following Table 5.4.

Table 5.4: Experimental Environment in benchmarking Analysis

Items	Detail
CPU Processor	Intel Core i5-3450 CPU @ 3.10GHz, 4 cores, Hyper Threading disabled
L1 Data-cache	32KB, 8 way associative, line size 64
L1 Instruction-cache	32KB, 8 way associative, line size 64
L2 Cache	256KB, 8 way associative, line size 64
L3 Cache	6144KB, 12 way associative, line size 64
Memory	11915MB DDR3 @1333MHz
VMM	Xen Hypervisor with dynamic cache partition
Virtual Machines	HVM guest, 1GB memory, 1 dedicated core for individual VM
Guest OS	Ubuntu 12.04.5

5.4.1 Load Testing

The load testing is investigated in order to support the load that a modified hypervisor based on our proposed method HBP-DCP can tolerate. We have tested a load of hypervisors in three modes, namely unmodified (default/insecure), static partitioned, and modified (dynamic partitioned/secure) hypervisor by sending multiple requests through Apache benchmark. We have created 10 VMs on each hypervisor namely unmodified, static, and our modified hypervisors and have checked a load of each hypervisor by sending many concurrent requests. For this, first, we have checked a load of unmodified (insecure), static partitioned, and modified (secure is our solution) hypervisors without creation of any VMs. Then after this, we have created 1VM, 2VM, 3VM, 4VM, 5VM, 6VM, 7VM, 8VM, 9VM, and 10 VMs on each hypervisor respectively and have checked the bearable load of each hypervisors in each case. In each hypervisor, for load testing, we have analyzed the average response time and the maximum number of requests per second that hypervisors can tolerate under a large number of connections or simultaneous users.

Table 5.5 shows the load testing of the unmodified, static partitioned, and dynamic partitioned (HBP-DCP) hypervisors in term of sending the concurrent requests and checking the average response time per request. We generate the different types of loads for the system in form of sending the concurrent requests to the server and run the experiment 30 times for 1 to 30 concurrent users. The Min and Maximum in the Table 5.5 representing the minimum and maximum ranges of generated number of requests per second and average response time per request for varying number of concurrent requests/users. In order to analyze that for what scenario the system will fail to work, we executed the system for the different workload. In this case, we do not have created any VM. The number of concurrent requests varies from 10 to 150. The number of concurrent requests means that our modified hypervisor can handle how many numbers of concurrent users. Table 5.5 shows that the difference in the average number of requests per second and the response time per request in the unmodified, static partitioned ,and in our modified (dynamic partitioned) hypervisors is significant as the T-test and P-test prove it. As shown in the table, the p-values for number of request per seconds and response time in the static and dynamic partitioned hypervisors are .037 and .421 and T-values are 2.323 and 1.923 respectively. These values prove the significance of the results.

Table 5.5: Load Testing of Unmodified, Static Partitioned, and Dynamic Partitioned Hypervisors without any VM and with Varying Number of Concurrent Requests

	Unmodified (Default/insecure)		**Static Partitioned Hypervisor**		**Modified (Dynamic Partitioned/secure)**	
Concurrent Requests	Number of Requests per Second	Average Response Time per request	Number of Requests per Second	Average Response Time per request	Number of Requests per Second	Average Response Time per request
10	5502	1.749	5070	12.222	5105	5.851
20	5530	3.480	5007	25.324	5115	15.806
30	5047	5.882	4834	36.234	4949	20.65
40	4943	7.827	4237	38.765	4462	22.988
50	4904	9.788	4317	52.342	4535	25.629
60	4999	12.218	4295	54.232	4910	38.979
70	4980	13.995	4805	68.454	4827	39.954
80	4903	16.517	4628	79.345	4843	45.271
90	5150	18.162	4400	85.332	4924	49.13
100	5199	19.894	4349	98.393	4964	55.837
110	5280	22.823	4255	106.347	4970	58.846
120	5301	24.766	4195	120.776	4618	59.481
130	5377	30.858	4322	158.711	4751	65.154
140	4998	50.101	4275	185.872	4695	98.152
150	3640	159.041	3408	245.743	3430	201.03
Mean	5050.2	26.47	4426.47	91.21	4739.86	53.52
Median	5047	16.517	4322	79.345	4843	45.271
Min	3640	1.749	3408	12.222	3430	5.851
Maximum	5530	159.041	5070	245.743	5115	201.03
Std. Deviation	442.592	38.664	407.70	64.63	410.050	47.00
Confidence. Int.	223.978	19.566	206.32	32.71	207.510	23.78
P-Value	.037	.0421	0.022	0.039	.0283/0.022	0.039
T-Value	1.992	1.721	2.099	1.826	1.992/2.099	1.826

The relationship between the number of requests and response time is that the average response time per request is increasing as the number of the concurrent users are increasing. Although there is a small increase in load in the case of modified (HBP-DCP) hypervisor as compared to unmodified as shown in a table. However, the modified (HBP-DCP/secure) hypervisor has the ability to secure the CC environment from cache-based SC attacks as compared to the unmodified (insecure) hypervisor. Since we know that security always comes with some overhead, therefore this is not a big difference in both hypervisors.

Table 5.6 shows the load testing of unmodified (default/insecure) hypervisor in term of sending the concurrent requests. In this case, we have created multiple VMs namely 1VM to 10VMs and have checked the bearable load in case of each VM. For instance, first, we have created 1VM and have checked the load for 10 to 100 concurrent users. Similarly, we repeated the same experiment for 2VM, 3VM, 4VM, 5VM, 6VM, 7VM, 8VM, 9VM, and 10VM respectively. The bearable load in term of number of requests per second is shown in the following table.

Table 5.6: Number of Requests per Second in Unmodified (Default/insecure) Hypervisor with Varying Number of VMs and Concurrent Users/Requests

Execution Traces	**Number of Requests per Second**									
Number of concurrent users	**1VM**	**2VM**	**3VM**	**4VM**	**5VM**	**6VM**	**7VM**	**8VM**	**9VM**	**10VM**
10	3311	3246	3302	3198	3186	3203	3222	3240	3207	3150
20	3256	3225	3124	3154	3106	3162	3208	3204	3099	3132
30	3180	3207	3223	3134	3223	3144	3198	3229	3213	3128
40	3298	3185	3258	3118	3218	3132	3182	3284	3282	3112
50	3238	3162	3285	3110	3284	3127	3166	3289	3285	3173
60	3258	3146	3244	3107	3229	3120	3150	3248	3247	3168
70	3189	3121	3240	3112	3192	3112	3142	3259	3238	3072
80	3156	3102	3162	3102	3185	3114	3123	3172	3211	3039
90	3138	3088	3203	3105	3216	3112	3105	3198	3202	3023
100	3094	3066	3265	3069	3215	3110	3090	3208	3195	2988
Mean	3211.8	3154.8	3230.6	3120.9	3205.4	3133.6	3158.6	3233.1	3217.9	3098.5
Median	3213.5	3154	3242	3111	3215.5	3123.5	3158	3234.5	3212	3120
Std. Deviation	71.502	60.818	54.922	34.824	45.043	29.507	44.485	38.173	52.669	64.429
Confidence .Int.	44.317	37.694	34.040	21.584	27.918	18.289	27.572	23.659	32.644	39.933

Table 5.7 shows the load testing of our modified (secure/dynamic partitioned) hypervisor in term of sending the concurrent request. In this case, we have repeated the same procedure as we done for the unmodified hypervisor. We have created multiple VMs namely 1VM to 10VMs and have checked the bearable load in case of each VM. For instance, first, we have created 1VM and have checked the load for 10 to 100 concurrent users. Similarly, we repeated the same experiment for 2VM, 3VM, 4VM, 5VM, 6VM, 7VM, 8VM, 9VM, and 10VM respectively. We have calculated the average

for each VMs. The approximate difference in average for each VM is 10 number of request in both unmodified and modified hypervisors.

Table 5.7: Number of Request per Second in Modified (Dynamic Partitioned) Hypervisor with Varying Number of Virtual Machines and Concurrent Requests

Execution Traces	**Number of Requests per Second**									
Number of concurrent users	**1VM**	**2VM**	**3VM**	**4VM**	**5VM**	**6VM**	**7VM**	**8VM**	**9VM**	**10VM**
10	3301	3209	3301	3226	3164	3202	3228	3201	3238	3250
20	3243	3195	3114	3176	3085	3141	3192	3082	3076	3092
30	3171	3166	3219	3142	3138	3119	3214	3201	3204	3218
40	3286	3145	3247	3130	3119	3093	3274	3242	3200	3212
50	3225	3130	3236	3115	3112	3075	3273	3255	3253	3233
60	3249	3120	3224	3103	3105	3068	3232	3237	3149	3168
70	3178	3102	3210	3081	3093	3122	3241	3221	3159	3072
80	3141	3080	3192	3065	3071	3090	3162	3155	3122	3039
90	3120	3038	3187	3043	3055	3033	3181	3192	3113	3023
100	3043	3006	3175	3022	3021	3017	3201	3182	2992	2988
Mean	3195.7	3119.1	3210.5	3110.3	3096.3	3096	3219.8	3196.8	3150.6	3129.5
Median	3201.5	3125	3214.5	3109	3099	3091.5	3221	3201	3154	3130
Std. Deviation	80.38	65.00	49.32	61.83	41.25	53.69	37.18	50.37	79.19	97.54
Confidence .Int.	49.82	40.29	30.57	38.32	25.57	33.28	23.04	31.22	49.08	60.45

Table 5.8 shows the number of requests and the average response time per request for both unmodified (insecure) and modified (dynamic partitioned/secure) hypervisor. The average number of requests per second is for the unmodified hypervisor is 3189.42 and for modified is 3150.1. Similarly, the average response time per request for the unmodified hypervisor is 18.42 and for modified is 19.52. There is only 1.24% difference in bearable load in term of number of requests per second in both hypervisor and only 5.8% difference in the average response time per request. However, this is acceptable because our modified (dynamic partitioned) hypervisor has the ability to prevent cross-

VM cache-based SC attacks. The T-test in the following Table 5.8 proves the significant difference in the load of both unmodified and modified hypervisor.

Table 5.8: Load Testing in Modified and Unmodified Hypervisor with Varying Number of VMs (Average for 10 to 100 Concurrent Request/users for each VM)

	Unmodified (Default Hypervisor)		**Modified (Dynamic Partitioned/secure) Hypervisor**	
Number of VMs	**Average Number of Requests per Second**	**Average Response Time per request**	**Average Number of Requests per Second**	**Average Response Time per request**
1	3231.8	16.024	3210.7	18.132
2	3228.8	17.108	3189.1	18.203
3	3216.6	17.901	3172.5	19.209
4	3210.9	18.479	3160.3	19.111
5	3198.4	18.551	3146.3	19.351
6	3187.6	18.732	3142.4	19.232
7	3172.6	18.98	3134.8	19.34
8	3161.1	19.201	3123.8	20.001
9	3149.9	19.57	3115.6	21.37
10	3136.5	19.611	3105.5	21.211
Min	3136.5	16.024	3105.5	18.132
Mean	3189.42	18.42	3150.1	19.52
Median	3193.0	18.6	3144.4	19.3
Maximum	3231.8	19.6	3210.7	21.4
Std. Deviation	33.46	1.13	33.30	1.08
Confidence Int.	20.74	0.70	20.64	0.67
P-Value	0.008	0.019	0.008	0.019
T-Value	2.53	2.220	2.53	2.220

Table 5.9 shows the bearable load in a statically partitioned hypervisor with varying number of VMs and number of partitions. Since the partition is created during boot time, therefore, we are unable to change the partitions during execution of VMs. As shown in the table the bearable load in term of the number of request per second is decreasing as the number of VMs and partitions are increasing. For instance, if we created 16 number of partitions in the cache or we divided the cache into 16 parts and during runtime, we need only one VM. Then this configuration cannot be changed during runtime and therefore, one part of the cache would be assigned to single created VM and the remaining

15 parts of cache will be wasted. Consequently, degrade the performance in term of bearable load because this single VM having limited part of cache accept the low number of requests per second. For instance, if the number of created VM is one and the number of partitions is 16, then the number of request per second will be very low regardless the number of VMs. As shown in Table 5.9, if the number of VMs is greater than the number of partition then the performance will be degraded. For instance, if there are 16 VMs and the number of partitions is 2 then during runtime it will be difficult to manage the partitions accordingly.

Table 5.9: Load Testing with Varying Number of VMs and Partitions in Static Partitioned Hypervisor

		Number of Requests per Second				
Number of concurrent users	Number of Partitions	With 1 VM	With 2VM	With 4VM	With 8 VM	With 16 VM
10	1	3200	3500	3200	1500	700
10	2	3200	3200	3100	1500	700
10	4	2800	2900	3100	1500	600
10	8	2300	1900	1700	1400	600
10	16	1900	1800	1600	1100	400

Conversely to the static partitioned mechanism, in the dynamic partitioned based hypervisor, the number of cache partitions is not decided during boot time as shown in Table 5.10. When VM is created then the cache is divided into partition accordingly. For instance, if one VM is created then the whole cache is assigned to single VM during runtime. While in the case of 8 or 16 VMs the whole cache is divided into 8 or 16 parts respectively. As compared to static partition, the dynamic partition improves the performance in term of the bearable load. For example, in the static partition, once we create 16 partitions at boot time then in the case of one VM creation the cache is divided into 16 parts. Consequently, degrade the performance because the other 15 parts of cache will be idle during execution. While in dynamic partition the case is different because the whole cache is assigned to that single VM. In the case of 8 or 16 VMs creation, the cache will be divided into 8 or 16 parts respectively.

Table 5.10: Load Testing with Varying Number of VMs in Dynamic Partitioned Hypervisor

	Number of Requests per Second				
Number of concurrent users	1VM/Partition	2VM/Partition	4VM/Partition	8 VM/Partition	16VM/Partition
10	3301/1	3209/2	3197/4	3176/8	2764/16
20	3243/1	3195/2	3186/4	3162/8	2726/16
30	3221/1	3166/2	3169/4	3140/8	2698/16
40	3196/1	3145/2	3147/4	3125/8	2645/16
50	3175/1	3132/2	3116/4	3104/8	2622/16

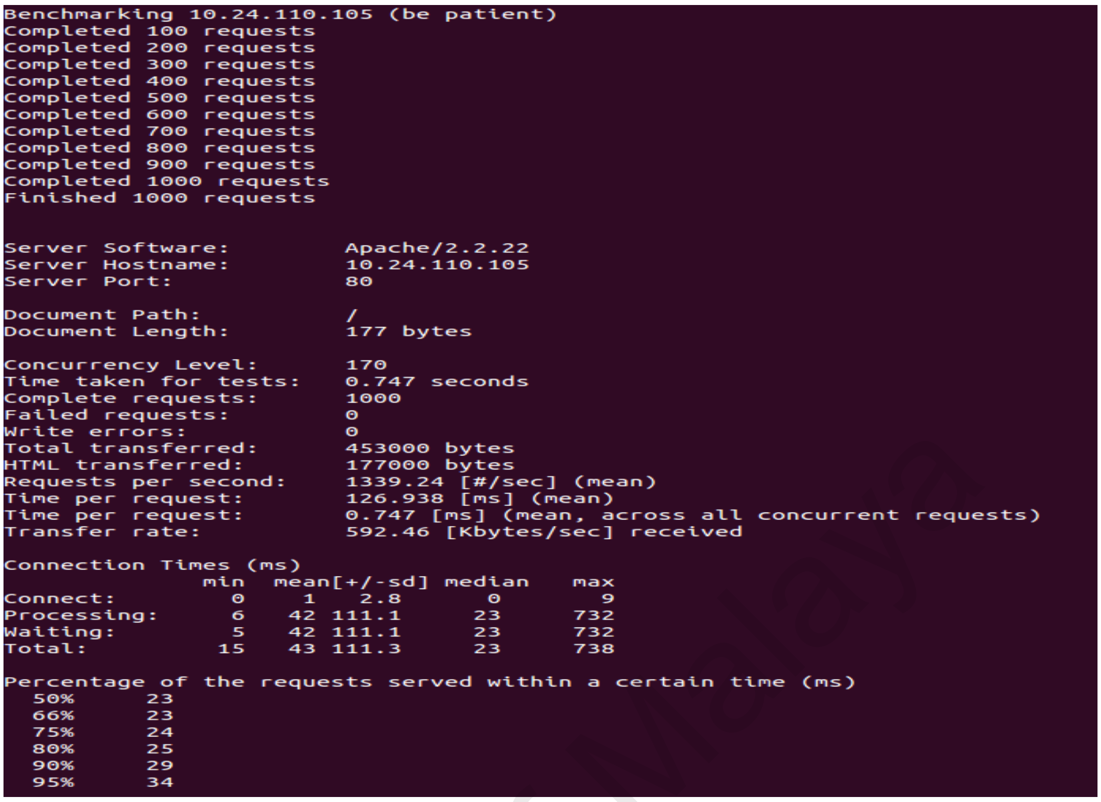
Table 5.11 shows the comparison of load testing in static and dynamic partitioned hypervisors. In both hypervisors, the bearable load is compared in term of the average number of requests per second and response time per request for the 1,2,4,8, and 16 cache partitions. As shown in the table, in the static partitioned hypervisor, the number of requests per second is decreasing with the increasing number of VMs and partitions. For instance, the average number of request per second 1977.28 in the static partitioned hypervisor and 3157.88 in our dynamic partitioned (HBP-DCP) hypervisor. Similarly, the average response time in the static partitioned hypervisor is 19.33 and 18.28 in our dynamic partitioned (HBP-DCP). The number of request per second is increased by 45.33% and the average response time per request is decreased by 5.58%. Therefore, the bearable load in term of request per second and average response time is improved in our dynamic partitioned (HBP-DCP based) hypervisor. Since the number of partitions is predefined during boot time, we cannot change during runtime. For instance, if we partitioned the entire cache into 16 parts and we are executing one VM then only single part of the entire cache will be assigned to that executing VM and the remaining 15 parts will be idle. Consequently, degrade the performance in term of the load. While in our

dynamic partitioned hypervisor, the cache partition is not predefined but decided during runtime according to the number of executingVMs. For instance, if we create one VM then the entire cache will be assigned to that single VM while if we create 16 VMs then the cache the cache will be divided into 16 parts. Therefore, the overall performance will be improved.

Table 5.11: Comparison of Load Testing in Static-Partitioned and Dynamic-Partitioned-based Hypervisors with Varying Number of VMs and Partitions (Average for 10 to 100 Concurrent Request for each VM)

	Static Partitioned-based Hypervisor (1,2,4,8,16 Static Partitions)		**Dynamic Partitioned-based Hypervisor (Dynamic Partition)**	
Number of VMs	Average Number of Requests per Second	Average Response Time per request	Average Number of Requests per Second	Average Response Time per request
1	2684.5	18.132	3210.7	17.024
2	2660.8	18.203	3189.1	17.108
4	2540.2	19.111	3160.3	17.479
8	1400.6	20.001	3123.8	18.201
16	600.3	21.211	3105.5	19.611
Min	600.3	18.132	3105.5	17.024
Mean	1977.28	19.33	3157.88	18.28
Median	2540.2	19.111	3160.3	18.479
Maximum	2684.5	21.211	3210.7	19.611
Std. Deviation	937.13	1.30	43.81	1.18
Confidence Int.	821.42	1.14	38.40	1.04
T-Value	2.813	1.927	2.813	1.927
P-Value	0.11	.045	0.11	.045

Figure 5.2 shows the result of sending 170 concurrent request during execution. It shows that with increasing number of concurrent request in each hypervisor namely unmodified (default/insecure), static partitioned, and our modified HBP-DCP (dynamic partitioned) hypervisors the number of request per second is decreasing. While the average response time per request in increasing. The response time per request is 126ms. The 126ms response time is showing the fastest request. If the response time per request is or less than 175ms then it will be considered the fastest requests while the slowest requests have the response time is or greater than 224ms.



```
Benchmarking 10.24.110.105 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        Apache/2.2.22
Server Hostname:        10.24.110.105
Server Port:            80

Document Path:          /
Document Length:        177 bytes

Concurrency Level:      170
Time taken for tests:   0.747 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      453000 bytes
HTML transferred:       177000 bytes
Requests per second:    1339.24 [#/sec] (mean)
Time per request:       126.938 [ms] (mean)
Time per request:       0.747 [ms] (mean, across all concurrent requests)
Transfer rate:          592.46 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   2.8      0       9
Processing:     6   42 111.1     23     732
Waiting:        5   42 111.1     23     732
Total:         15   43 111.3     23     738

Percentage of the requests served within a certain time (ms)
  50%     23
  66%     23
  75%     24
  80%     25
  90%     29
  95%     34
```

Figure 5.2: Result of Apache Benchmark with Varying Number of Concurrent Requests

Statistical model is designed in order to validate the results of performance evaluation produced via experimental analysis. The result of statistical model for load testing are presented here. We have designed the load estimation model to test the load value for each VM based on the two variables namely request per seconds and the average time per request. In order to build the statistical model, we have taken 80 % data for the training and 20% for the validation. We have taken these two values to train our model as much as possible to avoid biased results. The output of the statistical model is shown in the following Table 5.12. The statistical model for load is shown in the following Eq.5.3.

R⟵lm (Load ~ Number of Request + Time per Request)

$$W = \sum_{i=1}^{10} (L_{VMi} + NR_{VMi} + TPR_{VMi}) \quad (5.3)$$

Where i is = 1 to 10 (Number of Virtual Machines from 1 to 10)

The detail statistics of the statistical model of our linear regression are summarized in Table 5.12. The R value shows significance correlation between the number of request per second and the response time per request. The average R-squared value in the table testifies that 99.20% of the load value can be explained using number of request per second and response time per request. The F-statistics in the table ensure that available data is appropriate to be used for linear regression and P-value shows the significance of the result.

Table 5.12: Regression Statistics Summary for Load Testing of Varying VMs

Number of VM	P-Value	R-Squared	Adjusted R-Squared	F-Statistic
1VM	4.968e-05	0.993	0.9894	281.8
2VM	0.0008928	0.9907	0.9845	160.3
3VM	7.348e-07	0.991	0.988	329.5
4VM	3.633e-12	0.9998	0.9998	1.951e+04
5VM	2.77e-09	0.9999	0.9999	3.8e+04
6VM	1.09185-03	0.5488	0.9884	3.649
7VM	4.819e-06	0.9978	0.0067	909
8VM	1.529e-07	0.9996	0.9994	5112
9VM	5.636e-14	1	0.9999	7.824e+04
10VM	8.348e-11	0.9996	0.9994	6861

5.4.2 Cache Utilization

Since our HBP-DCP prevention mechanism for cross-VM cache-based SC attacks is based on the dynamic partition of cache for each VM and the performance impact of our prevention system depends on the cache functionality. Therefore, we used cache utilization as our evaluation parameter that how much our solution effect the cache bandwidth in term of cache read/write/modify, cache read, and cache write. Cache utilization is investigated for unmodified, static partitioned, and modified (HBP-DCP/Dynamic partitioned) hypervisors to check the amount of data accessed in bytes by each one. We used the cache write, cache read and cache Read/Modify/Write from the cachebench benchmark to evaluate the different level of cache in term of accessed data. Each one executes repeated access to data items on varying vector lengths. For each vector length, timings are taken based on the number of iterations. The total amount of accessed data in bytes is calculated by computing the product of vector length and number of iteration. A bandwidth figure (e.g., Megabytes as being 1024^2 or 1048576) in megabytes per second is then computed by dividing this total data accessed by the total time. Moreover, the average access time in nanoseconds per each data item is computed and reported. For cache usage metric, the cache writes, cache read, cache read/modify/write are conducted to evaluate both unmodified (insecure) and modified (secure: our solution) hypervisor. Cachebench is used to run these three benchmarks in order to measure the time in nanoseconds and bandwidth in MB/sec. Cache read calculates the read bandwidth by varying vector length. The resulting bandwidth will be high for the cases, where the vector length is less than cache size because the data will be coming from the cache.

Cache size and vector size are both independent variables. Cache size is how much data is stored "locally" in some sense. Vector length is the amount of data to transmit and can thus be any number. The measures of interest come when the vector is larger than the cache as shown in the Table. As shown in Table 5.15, there is no significant difference between the modified (dynamic partitioned) and unmodified (default) hypervisor with increasing number of VMs despite the expectation to the contrary. To investigate why we analyzed the source code of cachebench. It is clear from the code that cachebench obtain its reading by measuring the response time for cache small sections at any specific time. The dynamic cache partition will not affect the cache performance because these sections are very small and have enough cache to work.

This benchmark shows that a program with both low and high memory footprint should not be negatively affected by the dynamic partitioning of the cache. In contrast, static partition does not affect the program with low partition, however, it has a negative impact on the program with high memory footprint. Because, in the static partition, the small program can entirely fit in the assigned smaller partition of cache to each VM while the large memory footprint program can not fit entirely in the small partition assigned to each VM at boot time. Conversely, in a dynamic partition, the program with both small and large memory footprint can fit entirely in the assigned cache to each VM. Because in the dynamic partition, if for example, 2 VMs are running then the whole cache would be assigned to these 2 VMs and therefore large memory footprint program will not degrade the overall performance. For instance, if a program just needs 8KB or 200KB of the cache at any specific time then it will have no negative impact on the cache performance if there are more than 10 partitions because, during dynamic partition, 512kb is assigned to each VMs. However, if one or two VMs are running then the whole cache memory will be assigned to one or two VM, in Core i5 the 2MB cache will be assigned to both VM. Therefore, a program with the low memory footprint has no negative impact on the cache performance during cache partitioning.

Table 5.13 and 5.14 present the data related to the cache utilized by varying vector lengths in each VM e.g., 1VM, 2VMs, 3VMs, 4VMs, 5VMs, 6VMs, 7VMs, 8VMs, 9VMs, and 10VMs, which are collected in unmodified (insecure/default) and modified (Secure/HBP-DCP/dynamic partitioned) hypervisors. Each Table summarizes the bandwidth for varying vector lengths with 95% confidence interval for 30 number of iteration for each VM e.g., 1VM to 10VMs. Similar to load testing for each VM, we present bandwidth in MB/Sec for Cache Read/Modify/Write e.g., the total amount of data accessed in bytes with 95% confidence interval to enable reliability of our data. The benchmark Read/Modify/Write in cachebench is used to determine the bandwidth (how much data is accessed) by varying vector lengths. The data in both tables show that there is an acceptable difference between the cache utilization in term of bandwidth MB/Sec in both hypervisors even the modified hypervisor has the ability to prevent cross-VM cache-based SC attacks.

Table 5.13: Cache Utilization of Unmodified Hypervisor

		Bandwidth (MB/Sec) for Read/Modify/Write of Varying Virtual Machines									
Execution Traces	C-Size	1VM	2VM	3VM	4VM	5VM	6VM	7VM	8VM	9VM	10VM
1	256	18010.78	16320.03	18256.65	18374.66	18141.36	17673.91	18284.43	18238.125	17875.55	17407.48
2	336	17444.21	17436.02	17457.82	17422.24	17386.05	16942.28	17339.97	17350.963	16928.82	16410.65
3	424	18974.97	18986.7	18945.75	18856.22	18751.78	18322.14	18853.09	18800.384	18362.71	17874.6
4	512	20105.25	20115.28	20052.72	20023.84	19463.41	19475.4	19977.81	19914.018	19428.63	19272.44
5	680	21626.97	21622.58	21562.56	21515.29	20.941.81	20967.44	21505.72	21399.052	20785.12	20229.52
6	848	22669.82	22657.75	22573.36	22589.29	21877.59	21968.81	22543.28	22422.787	21678.64	21255.02
7	1024	23445.77	23420.64	23232.63	23360.52	22723.52	22716.04	23204.62	23123.022	22385.02	22119.99
8	1360	24449.76	24442.52	24338.14	24404.33	23706.65	23698.67	24335.17	24064.946	23251.82	23678.41
9	1704	25102.69	25120.47	25057.39	25002.8	24339.59	24354.7	24983.84	24711.307	23901.93	24322.07
10	2048	25571.66	25580.1	25541.13	25480.36	24769.92	24809.76	25470.57	25241.943	24329.22	24760.35
11	2728	26168.16	26145.53	26085.84	26030.6	25354.95	25398.21	26060.53	25892.23	24811.41	25340.84
12	3408	26540.02	26520.56	26457.35	26437.22	25731.23	25719.91	26410.5	26235.549	25208.98	25704.39
13	4096	26774.5	26779.57	26650.52	26695.41	25913.12	25894.68	26667.05	26499.885	25381.6	25930.89
14	5456	27044.14	27061.88	27038.92	27045.49	26313.17	26277.96	27000.67	26839.088	24930.03	26262.08
15	6824	27276.03	27300.06	27240.12	27254.47	26466.41	26475.72	27199.62	27048.748	25161.71	26457.36
16	8192	27422.45	27445.8	27344.61	27355.97	26526.88	26599.68	27340.21	27156.973	8192	27422.45
17	10920	27607.14	27608.08	27522.84	27519.55	26758.4	26771.99	27468.5	27318	10920	27607.14
18	13648	27597.66	27707.58	27633.22	27610.15	26864.49	26867.85	27561.91	27459.636	13648	27597.66
19	16384	27778.46	27771.29	27690.99	27708.51	26943.71	26925.28	27612.92	27512.886	16384	27778.46
20	21840	27863.58	27862.95	27784	27788.43	27021.39	27012.86	27697.67	27589.881	21840	27863.58
21	27304	27913.91	27913.22	27875.83	27821.62	27069.35	27063.09	27791.73	27607.307	27304	27913.91
22	32768	27944.52	27931.52	27885.8	27704	27090.11	26997.19	27809.4	27699.925	32768	27944.52

Table 5.13: Continue…

23	43688	27977.26	27982.49	27909.43	27895.51	27065.76	27126.44	27870.49	27536.417	43688	27977.26
24	54608	28017.56	27994.23	27949.03	27890.55	27161.52	27160.8	27893.35	27589.301	54608	28017.56
25	65536	28033.89	28025.99	27944.67	27910.99	27173.82	27156.75	27887.54	27764.86	65536	28033.89
26	87376	28046.01	28042.92	27983.92	27945.97	27184.51	27214.36	27853.62	27674.065	87376	28046.01
27	109224	28051.91	28063.72	27998.85	27960.02	27204.26	27211.66	27787.4	27803.388	109224	28051.91
28	131072	28063.83	28073.39	27993.38	27947.32	27204.54	27221.6	27975.84	27696.146	131072	28063.83
29	714760	28079.79	28082.25	27978.91	27927.25	27232.66	27242.34	27951.38	27783.632	714760	28079.79
30	218448	28132.23	28120.33	27989.24	27950.55	27240.56	27250.34	27982.32	27790.339	218448	28132.23
	Min	17444.21	16320.03	17457.82	17422.24	17386.05	16942.28	17339.97	17350.96	16928.82	16410.65
	Median	27349.24	27372.93	27292.37	27305.22	26526.88	26537.70	27269.92	27102.86	25314.25	26331.57
	Maximum	28132.23	28120.33	27998.85	27960.02	27240.56	27250.34	27982.32	27803.39	27848.38	27002.85
	Std. Deviation	3302.39	3451.20	3271.43	3258.45	3051.04	3190.86	3269.59	3227.04	2936.11	3254.17
	Confidence Int.	1181.72	1234.97	1170.65	1166.00	1091.78	1141.81	1169.99	1154.76	1050.65	1164.47

Table 5.14: Cache Utilization of Modified Hypervisor

		Bandwidth (MB/Sec) for Read/Modify/Write of Varying Virtual Machines									
Execution Traces	**C-Size**	**1VM**	**2VM**	**3VM**	**4VM**	**5VM**	**6VM**	**7VM**	**8VM**	**9VM**	**10VM**
1	256	17490.78	16300.03	18236.65	18354.66	18121.36	17653.91	18264.43	18218.125	17855.55	17387.48
2	336	17419.21	17411.02	17432.82	17397.24	17361.05	16917.28	17314.97	17325.963	16903.82	16385.65
3	424	18555.97	18967.7	18926.75	18837.22	18732.78	18303.14	18834.09	18781.384	18343.71	17855.6
4	512	20073.25	20093.28	20030.72	20001.84	19441.41	19453.4	19955.81	19892.018	19406.63	19250.44
5	680	21577.97	21599.58	21539.56	21492.29	20100.11	20944.44	21482.72	21376.052	20762.12	20206.52
6	848	22630.82	22636.75	22552.36	22568.29	21856.59	21947.81	22522.28	22401.787	21657.64	21234.02
7	1024	23393.77	23393.64	23205.63	23333.52	22696.52	22689.04	23177.62	23096.022	22358.02	22092.99
8	1360	24414.76	24417.52	24313.14	24379.33	23681.65	23673.67	24310.17	24039.946	23226.82	23653.41
9	1704	25065.69	25103.47	25040.39	24985.81	24322.59	24337.7	24966.84	24694.307	23884.93	24305.07
10	2048	25535.66	25564.11	25525.13	25464.36	24753.92	24793.76	25454.57	25225.943	24313.22	24744.35
11	2728	26121.16	26125.53	26065.84	26010.6	25334.95	25378.21	26040.53	25872.23	24791.41	25320.84
12	3408	26502.02	26497.56	26434.35	26414.22	25708.23	25696.91	26387.5	26212.549	25185.98	25681.39
13	4096	26735.35	26753.57	26624.52	26669.41	25887.12	25868.68	26641.05	26473.885	25355.6	25904.89
14	5456	27023.14	27040.88	27017.92	27024.49	26292.17	26256.96	26979.67	26818.088	24909.03	26241.08
15	6824	27214.03	27273.06	27213.12	27227.47	26439.41	26448.72	27172.62	27021.748	25134.71	26430.36
16	8192	27375.15	27419.8	27318.61	27329.97	26500.88	26573.68	27314.21	27130.973	25220.89	26492.44
17	10920	27473.24	27580.08	27494.84	27491.55	26730.4	26743.99	27440.5	27290.12	25561.34	26701.01
18	13648	27532.36	27688.58	27614.22	27591.15	26845.49	26848.85	27542.91	27440.636	25960.02	26818.27

Table 5.14: Continue…

19	16384	27741.4	27751.29	27670.99	27688.51	26923.71	26905.28	27592.92	27492.886	26024.53	26832.16
20	21840	27831.28	27833.95	27755.21	27759.43	26992.39	26983.86	27668.67	27560.881	25670.56	26879.6
21	27304	27790.91	27890.22	27852.83	27798.62	27046.35	27040.09	27768.73	27584.307	26147.74	26943.03
22	32768	27793.32	27901.52	27855.81	27674.21	27060.11	26967.19	27779.4	27669.925	25913.64	26967.68
23	43688	27884.12	27959.49	27886.43	27872.51	27042.76	27103.44	27847.49	27513.417	26315.71	26979.85
24	54608	27975.22	27969.23	27924.03	27865.55	27136.52	27135.8	27868.35	27564.301	26828.74	26878.88
25	65536	27970.82	28002.99	27921.67	27887.99	27150.82	27133.75	27864.54	27741.86	26082.77	26711.79
26	87376	27995.41	28021.92	27962.92	27924.97	27163.51	27193.36	27832.62	27653.065	26283.22	26591.34
27	109224	28016.86	28038.72	27973.85	27935.02	27179.26	27186.66	27762.4	27778.388	26367.67	26504.82
28	131072	28028.22	28047.39	27967.38	27921.32	27178.54	27195.6	27949.84	27670.146	27108.16	26445.88
29	714760	27997.21	28060.25	27956.91	27905.25	27210.66	27220.34	27929.38	27761.632	27163.41	26379.05
30	218448	28107.33	28100.33	27969.24	27930.55	27220.56	27230.34	27962.32	27770.339	27828.38	26135.57
	Min	17419.21	16300.03	17432.82	17397.24	17361.05	16917.28	17314.97	17325.96	16903.82	16385.65
	Median	27294.59	27346.43	27265.87	27278.72	26470.15	26511.20	27243.42	27076.36	25288.25	26310.07
	Maximum	28107.33	28100.33	27973.85	27935.02	27220.56	27230.34	27962.32	27778.39	27828.38	26979.85
	Std. Deviation	3357.68	3450.40	3270.67	3257.68	3129.66	3190.09	3268.82	3226.26	2935.50	3253.39
	Confidence Int.	1201.51	1234.69	1170.37	1165.72	1119.92	1141.54	1169.71	1154.48	1050.43	1164.19

Similarly, Table 5.15 shows the bandwidth of cache read and cache writes by varying vector length i.e., c-size for both unmodified and modified hypervisor generated via experiment.

Table 5.15 Average Bandwidth (MB/Sec) of Cache Read and Cache Write of Varying VMs (1VM-10VM) in Un-Modified and Modified Hypervisor

Average Bandwidth (MB/Sec) of Varying VMs (VM1-VM10)					
		Unmodified (default/insecure) Hypervisor (1VM-10VMs)		**Modified (Dynamic-Partitioned) Hypervisor (1VM-10VMs)**	
Execution Traces	C-Size	Bandwidth of Cache Read (MB/Sec)	Bandwidth of Cache Write (MB/Sec)	Bandwidth of Cache Read (MB/Sec)	Bandwidth of Cache Write (MB/Sec)
1	256	1935.88	2139.74	1903.88	2127.74
2	336	1938.39	2173.34	1908.39	2161.34
3	424	1949.24	2178.98	1917.24	2166.98
4	512	1948.09	2042.00	1923.09	2030.00
5	680	1323.09	1933.28	1288.09	1921.28
6	848	1931.31	2068.73	1916.31	2056.73
7	1024	1939.8	2163.17	1924.8	2151.17
8	1360	1921.37	2156.73	1896.37	2144.73
9	1704	1952.07	2225.12	1927.07	2213.12
10	2048	1957.94	2224.35	1942.94	2212.35
11	2728	1969.46	2236.61	1954.46	2224.61
12	3408	1963.77	2273.94	1948.77	2261.94
13	4096	1975.43	2293.04	1960.43	2281.04
14	5456	1963.89	2214.23	1948.89	2199.23
15	6824	1968.47	2096.07	1947.47	2081.07
16	8192	1976.26	2182.09	1941.26	2167.09
17	10920	1397.06	2157.74	1342.06	2142.74
18	13648	1331.05	2019.30	1306.56	2004.30
19	16384	1330.59	2142.71	1235.59	2127.71
20	21840	1317.53	2296.77	1302.53	2281.77
21	27304	1313.01	2528.70	1268.01	2513.70
22	32768	1300.64	2310.62	1275.64	2295.62
23	43688	1009.88	2005.95	1044.88	1990.95
24	54608	1085.77	2329.79	1070.77	2314.79
25	65536	1310.22	1935.00	1295.22	1920.00
26	87376	1297.13	1930.16	1272.13	1915.16
27	109224	1353.34	2019.69	1328.34	2004.69
28	131072	1330.98	2233.06	1335.98	2218.06
29	174760	1191.13	2385.88	1276.13	2370.88
30	218448	1086.98	2133.25	1271.98	2118.25
Mean		1608.99	2167.67	1595.82	2153.97
Std. Deviation		358.86	138.41	346.29	138.36
Confidence Int.		128.41	49.53	123.92	49.51
T-Value		1.981	2.012	1.981	2.012
P-Value		.0437	.0411	.0437	.0411

Cache read and cache write perform repeated access to the data item on varying vector lengths such as varying c-size in Table 5.14. The average of cache read for unmodified is 1595.82 and for modified is 1608.99 and the average for cache write is 2153.97 for unmodified and 2167.67 for modified are almost 14. The difference between the cache read of both unmodified and our modified (dynamic partitioned) is .821% and in the cache write is .634%. This is acceptable difference because our modified hypervisor has the ability to prevent cross-VM cache-based SC attacks. The T-test proves the significant difference of cache read and cache write in both unmodified and modified hypervisor.

Table 5.16 shows the average bandwidth of cache read/write/modify benchmark in the static partitioned hypervisor. As the expected performance of our dynamic partitioned hypervisor depends on the utilization of cache. Therefore, we execute the cache read/write/modify to evaluate the cache utilization of static partitioned hypervisor. In Table 5.16, the Min and Maximum in the first column representing the minimum and maximum ranges of cache Read/write/Modify bandwidth in case of varying number of VMs and partitions.

Table 5.16: Bandwidth of Cache Read/Write/Modify in Static Partitioned Hypervisor

	Average Bandwidth of cache Read/Write/Modify (MB per Second)				
Number of Partition	**1VM**	**2VMs**	**4 VMs**	**8 VMs**	**16 VMs**
1	17923	17128	15289	13567	13889
2	15628	14035	13878	11228	12556
4	14289	13582	12728	10988	10454
8	10989	9366	8800	8487	8000
16	4896	4098	3789	3567	3089
Min	4896	4098	3789	3567	3089
Maximum	17923	17128	15289	13567	13889
Mean	12363.4	11347.9	10508.9	9281.6	9280.9
Std. Deviation	5054.6	5041.8	4648.9	3806.1	4266.6
Confidence Int.	4430.5	4419.2	4074.9	3336.1	3739.8

In static partitioned hypervisor, the average bandwidth of cache read/write/modify is decreasing with increasing number of VMs and partitions. Even in the case of 1VM or 2 VMs, if the partitions are 16 then the cache bandwidth will be low because the cache is divided into 16 parts. While in dynamic partitions, the bandwidth will be high in case of 1 or 2 VMs because the entire cache will be divided into 1 or 2 partitions. For instance, in a static partition, if for 1VM the number of partition is 1 the bandwidth will be 17923 as shown in Table 5.16. While for 1VM, if the number of partitions is 16 then the bandwidth will 4896 which is very low as compared to 17923 in the case of 1 partition.

Table 5.17 shows the comparison of cache read/write/modify in both static and dynamic partitioned hypervisors. We have created 1VM, 2VM, 4VM, 8VM, and 16VM in both hypervisors for checking the bandwidth of cache read/write/modify with varying number of partitions. As compared to the dynamic partitioned hypervisor, the average bandwidth of cache read/write/modify in the static partitioned hypervisor is decreasing with increasing number of VMs and partitions. For instance, the average bandwidth of cache read/modify/write in the static partitioned hypervisor is 13012.8 and 18234.7 in our dynamic partitioned (HBP-DCP) hypervisor.

Table 5.17: Average Bandwidth of cache Read/Modify/Write in Static and Dynamic Partitioned Hypervisors

Average Bandwidth (MB/Sec) of Cache Read/Modify/Write with varying VMs and partitions		
Number of Partitions	Static Partitioned (Average of 1VM, 2VM, 4VM, 8VM and 16VMs)	Dynamic partitioned (HBP-DCP) (Average of 1VM, 2VM, 4VM, 8VM and 16VMs)
1	14745.5	19645.848
2	13641.8	18629.448
4	13896.8	18110.181
8	11567.4	17012.904
16	11246.3	17585.894
Min	11246.3	17012.9
Maximum	14745.5	19645.85
Mean	13012.8	18234.7
Std. Deviation	1532.059	1008.6
Confidence Int.	1342.884	884.1
P-Value	0.00012	
T-Value	6.311	

Thus the average bandwidth of cache Read/Modify/Write is improved by 33.32% in our HBP-DCP based hypervisor as compared to static partitioned hypervisor. Consequently improves the cache utilization. Moreover, the T-test values namely P-value and T-value in the Table 5.17 shows the validity of the results. Because in the static partitioned hypervisor, the number of partitions is static and predefined during boot time. Therefore, 16 partitions cannot change for 1 VM while in the case of the dynamic partitioned hypervisor, the number of partitions will be changed according to created VMs.

Similarly, Table 5.18 shows the comparison of cache read in both static and dynamic partitioned hypervisors. We have analyzed the cache read bandwidth by varying number of VMs and partitions in both static partitioned and dynamic partitioned hypervisors. The average bandwidth of the cache read in static partitioned hypervisor is decreasing with increasing number of VMs and partitions as shown in Table 5.18. For instance, the average cache read bandwidth for 2 VMs in the case of 16 partitions is 946.3 in the static partitioned hypervisor while in the dynamic partitioned hypervisor is 1225.375. The cache read bandwidth of dynamic partitioned (HBP-DCP) is more as compared to static partitioned. Since the 16 partitions are defined during boot time so the partition cannot change even 2 VMs are running.

Consequently, degrade the cache read bandwidth. While in dynamic if 2 VMs are executing then the partitions will be changed into 2 according to the executing VMs. Thus improve the average cache read bandwidth. For instance, the average bandwidth of cache read in the static partitioned hypervisor is 1164.16 and 1474.07 in our dynamic partitioned (HBP-DCP) hypervisor. Thus the average bandwidth of cache write is improved by 23.493% in our HBP-DCP based hypervisor. Furthermore, the P-value and T-value shows the validity of the result as the P-value is less than 0.05. This improvement in the cache read bandwidth improves the cache performance in term of cache utilization as compared to static partitioned hypervisor.

Table 5.18: Average Bandwidth of Cache Read in Static and Dynamic Partitioned Hypervisors

Average Bandwidth (MB/Sec) of Cache Read with varying VMs and partitions		
Number of Partitions	Static Partitioned (Average of 1VM, 2VM, 4VM, 8VM and 16VMs)	Dynamic partitioned (HBP-DCP) (Average of 1VM, 2VM, 4VM, 8VM and 16VMs)
1	1345.5	1585.212
2	1264.8	1552.323
4	1196.8	1532.333
8	1067.4	1475.134
16	946.3	1225.375
Min	946.3	1225.375
Maximum	1345.5	1585.212
Mean	1164.16	1474.07
Std. Deviation	158.827	144.673
Confidence Int.	139.215	126.809
P-Value	0.006	
T-Value	3.225	

Similarly, in Table 5.19, the bandwidth of cache write for both static partitioned and modified (dynamic partitioned) hypervisors is shown. Similarly to the other two bandwidth cache read bandwidth for the static partitioned hypervisor is less than the dynamic partitioned hypervisor. The average bandwidth of cache write in the static partitioned hypervisor is 1383.374 and 1908.416 in our dynamic partitioned (HBP-DCP) hypervsior. Thus the average bandwidth of cache write is improved by 32% in our HBP-DCP based hypervisor. Furthermore, the P-value and T-value shows the validity of the result. Consequently, improves the cache performance in term of cache utilization as compared to static partitioned hypervisor.

Table 5.19: Average Bandwidth of Cache Write in Static and Dynamic Partitioned Hypervisors

Average Bandwidth (MB/Sec) of Cache Write with varying VMs and partitions		
Number of Partitions	Static Partitioned (Average of 1VM, 2VM, 4VM, 8VM and 16VMs)	Dynamic partitioned (HBP-DCP) (Average of 1VM, 2VM, 4VM, 8VM and 16VMs)
1	1710.201	2024.191
2	1630.451	2010.763
4	1445.134	1975.354
8	1202.543	1910.541
16	928.541	1621.232
Min	928.541	1621.232
Maximum	1710.201	2024.191
Mean	1383.374	1908.416
Std. Deviation	320.644	166.462
Confidence Int.	281.052	145.908
P-Value	0.005	
T-Value	3.249	

Table 5.20, 5.21, and 5.22 shows the average bandwidth in MB/Sec calculated from the cache Read/Modify/Write, cache read, and cache write benchmarks for each VM including VM 1 to 10VMs in the unmodified and modified hypervisor. The cache read/write/modify generate much more memory traffic as compared to the cache read and cache write because in the cache read/write/modify the data items must be first read from the cache to register and then back to memory/cache. Therefore, the bandwidth for cache read and cache write is less than the bandwidth of read/write/modify as shown in Table 5-20. The average for unmodified is 25625.64 and 25572.21 for modified hypervisor. The difference is almost .208% between both unmodified and modified hypervisor. This difference is acceptable because the modified hypervisor has the ability to prevent cross-

VM cache-based SC attacks. The T-test result proves the significant difference between both values.

Table 5.20: Comparison of cache Read/Modify/Write in Unmodified and Modified (HBP-DCP) Hypervisors

	Average Bandwidth (MB/Sec) of Cache Read/Modify/Write	
Number of VMs	**Unmodified (Insecure)**	**Modified (dynamic-partitioned)**
1	25683.831	25645.848
2	25680.515	25629.448
3	25634.193	25610.181
4	25644.917	25612.904
5	25667.956	25585.894
6	25670.929	25613.862
7	25594.038	25570.093
8	25572.166	25543.093
9	25558.599	25538.542
10	25542.249	25522.272
Min	25542.25	25522.27
Maximum	25683.93	25645.85
Mean	25622.96	25586.69
Std. Deviation	53.77	42.11
Confidence Int.	33.32	26.10
T-Value	1.746	
P-Value	0.0417	

Table 5.21 shows the result of bandwidth in MB/Sec calculated by cache read benchmark in the unmodified and modified (dynamic partitioned/HBP-DCP) hypervisor. The average bandwidth of Cache Read for unmodified is 1546.433 and for our modified (dynamic partitioned) hypervisor is 1514.567. The difference in cache Read bandwidth of both is 2.08% which is acceptable because our modified hypervisor has the ability to prevent cross-VM cache-based SC attacks. Since the cache Read/Modify/Write benchmark first read the values from the cache and then will write in the cache. Therefore, the resulting bandwidth for cache Read/Modify/Write benchmark will be high as compared to cache read and cache write benchmarks for both modified and unmodified

hypervisor as shown in Table 5.20, 5.21, and 5.22. Furthermore, the T-test (P-value and T-value) shows the validity of the result.

Table 5.21: Comparison of Cache Read in Unmodified and Modified (HBP-DCP) Hypervisor

	Average Bandwidth (MB/Sec) of Cache Read	
Number of VMs	**Unmodified (Insecure)**	**Modified (dynamic-partitioned)**
1	1610.157	1585.22
2	1585.222	1552.323
3	1572.211	1549.333
4	1566.777	1532.134
5	1553.221	1525.888
6	1540.554	1518.122
7	1529.212	1484.111
8	1519.454	1475.223
9	1505.291	1467.989
10	1482.234	1455.323
Mean	1546.433	1514.567
Std. Deviation	38.820	42.466
Confidence Int.	24.060	26.320
T-Value	1.751	
P-Value	0.0421	

Table 5.22 show the resulting bandwidth of the cache write benchmark in both unmodified and modified (dynamic partitioned/HBP-DCP) hypervisor. The average bandwidth for cache write in the unmodified hypervisor is 2184 and in the modified hypervisor is 2126. The resulting difference in percentage in both hypervisor is almost 2.69%. Which is acceptable because our modified (partitioned hypervisor) has the ability to prevent cache-based SC attacks between VMS. Furthermore, the T-value and P-value show that the result is significant as the T-value is less than 2.2 and P-value is less than 0.05.

Table 5.22: Comparison of Cache Write in Unmodified and Modified (HBP-DCP) Hypervisor

	Average Bandwidth (MB/Sec) of Cache Write	
Number of VMs	**Unmodified (insecure)**	**Modified (dynamic-partitioned)**
1	2273.121	2224.191
2	2252.256	2210.763
3	2245.177	2195.354
4	2225.878	2162.823
5	2197.891	2134.871
6	2183.199	2117.783
7	2162.872	2101.234
8	2135.752	2067.553
9	2113.882	2047.812
10	2059.432	2006.232
Mean	2184.946	2126.862
Std. Deviation	67.897	72.739
Confidence Int.	42.082	45.083
T-Value	1.854	
P-Value	0.0307	

We have designed the cache bandwidth model to test the cache utilization for each VM e.g., 1VM, 2VMs, 3VMs, 4VMs, 5VMs, 6VMs, 7VMs, 8VMs, 9VMs, and 10VMs based on the two variables namely total time in nanoseconds and the total amount of data accessed in bytes. In order to build the statistical model, we have taken 80 % data for the training and 20% for the validation. We have taken these two values to train our model as much as possible to avoid biased results. Similar to a statistical model for load testing, in order to present a reliable and accurate estimation model of cache utilization in term of cache read/modify/write, cache read, and cache write, we perform linear regression using measured real data in cachebench benchmark in both unmodified and modified hypervisor. We use data set of cache utilization including cache read/modify/write, cache read, and cache write and use them for training the regression model to produce the bandwidth model. For validation of our proposed model, we use the split sample approach. Hence the cache utilization model can be presented as follow:

R$\longleftarrow$ **lm** (Data Accessed ~ Total Iteration × (C-size)

R⟵lm (C-utilization~ Total Time × (Total Amount of Data Accessed in Bytes)

$$B_m(W_{i)} = \sum_{i=1}^{10}(T_{VMi} \times Data_{VMi}) \quad (5.4)$$

Where $B_m (W_i)$ is the total bandwidth in MB/Sec calculated by the total amount of data accessed in bytes divided by the total time. The bandwidth is totally depends on the vector length. Because if the vector length is less than cache size then, in this case, the whole data will come from the cache and the resulting bandwidth will be high otherwise the resulting bandwidth will be low. Timings are taken for every vector length based on a number of iteration. The number of iteration is then multiplied by the vector length to compute the total amount of accessed data in bytes. Finally, the total amount of accessed data in bytes is divided by the total time to compute the bandwidth. The output of the statistical model is shown in the following Table 5.23.

Table 5.23: Regression Statistics Summary for Cache Utilization of Virtual Machines

Number of VM	P-Value	R-Squared	Adjusted R-Squared	F-Statistic
1VM	3.238e-05	0.9910	0.9899	1.502e+04
2VM	0.0008928	0.9907	0.9889	3.8e+04
3VM	4.338e-04	0.9991	0.9899	3.649
4VM	2.633e-03	0.9997	0.9999	1.951e+04
5VM	1.2345e-09	0.9998	0.9998	3.8e+04
6VM	1.09185-04	0.9898	0.9887	3.649+04
7VM	4.323e-05	0.9989	0.9941	2.345e+03
8VM	2.512e-04	0.9996	0.9994	40189e+02
9VM	3.623e-08	0.9930	0.9989	7.824e+04
10VM	5.316e-11	0.9986	0.9999	2.502e+03

The R value shows the significant correlation between the vector length and the bandwidth. The P-value and R-squared values in Table 5.23 for each VM testify the significance of the result using the statistical model for cache utilization. The detail statistics of the statistical model are summarized in Table 5.23. The R value shows significance correlation between the number of cache size and the bandwidth of cache/read/write. The average R-squared value in the table testifies that 99.61% of the load value can be explained using cache-size and cache read/modify/write. The F statistics in the table ensure that available data is appropriate to be used for linear regression and p-value shows the significance of the result.

5.4.3 Memory Access Rate

CPU cache is used to increase the speed of the memory access for the data which is most commonly accessed. However, our proposed HBP-DCP prevention mechanism divides the cache on the fly according to VMs requirement. Therefore, it is needed to check memory access rate as a performance parameter in order to check the evaluation of our prevention mechanism whether it will effect on the memory access rate or not. Although profiling cache memory operation requires collaboration from the hardware, however, it is also possible to collect information through software. The average memory access time is a valuable parameter to evaluate the performance of a memory hierarchy configuration. When a processor demand to execute an item from the main memory, it sends a load request to the cache memory. If the item resides in the cache it will generate the cache hit and in the case of absence, it will generate the cache miss. These cache miss and hit rate are used to calculate the memory access rate. We have checked the memory access rate in each hypervisor namely unmodified (Default), static partitioned, and modified (dynamic partitioned) to check the performance difference between each one.

As compared to the RAM storage, the access to the cache memory is faster due to the high latency of RAM storage. The total memory access time is calculated by the Eq. 5.8 while considering the cache and memory of the system as a target location. In Eq. 5.5, Hit_{Rate} represents the amount of data accessed from the cache memory. Alternatively, $Miss_{Rate}$ describes the fraction of data accessed from the main memory. Moreover, $Cache_{AccessTime}$ and $RAM_{AccessTime}$ represent the total time to access the data from the cache and main memory respectively (Rixner, Dally et al. 2000). For instance, time to access main memory is 100 ns in the majority of the system. The cache access time is proven to be 10 times faster than the main memory. Consider a program that yields a hit ratio with .92 for a reading request then the effective memory access time will be calculated by the following Eq. 5.5 and Eq. 5.6.

$$\text{Effective}_{\text{AccessTime}} = \text{HIT}_{\text{Rate}} \times \text{Cache}_{\text{AccessTime}} + \text{Miss}_{\text{Rate}} \times \text{RAM}_{\text{AccessTime}} \qquad (5.5)$$

$$\text{Effective}_{\text{AccessTime}} = .092 \times 10 + (1\text{-}0.92) \times 100 \approx 17\text{ns} \qquad (5.6)$$

The browsing experience can be improved by a high cache hit ratio while reducing costs in terms of energy, bandwidth, and computation power. Therefore the effectiveness of the caching system by monitoring the cache hit and cache miss ratio. We have written a customized program for checking the total memory access rate of a matrix program in the unmodified, static partitioned and HBP-DCP based hypervisors. For the program, we calculated the cache miss and cache hit for the matrix program by using cachegrind which is under the valgrind tool suit. We have collected the results for the average memory access time by executing our programs in the unmodified (default), static partitioned, and modified (dynamic partitioned) hypervisors.

Table 5.24 and 5.25 present the data related to the memory access rate by a matrix program which is collected in the unmodified (insecure/default) and modified (secure/dynamic partitioned) hypervisors execution modes for eight granularity levels of matrices, respectively. Each Table summarizes the memory access rate in term of total LLC memory access with 95% confidence interval for 30 number of iteration for each VM e.g., 1VM, 2VM, 3VM, 4VM, 5VM, 6VM, 7VM, 8VM, 9VM,10VM in eight intensity levels. Similar to cache utilization, we present the LLC references e.g., memory access with 95% confidence interval to enable reliability of our data. The small value of error estimate based on 95% confidence interval at the end of Table 5.24 and 5.25 testify the result of collected LLC references data.

Table 5.24: Last Level Cache (LLC) Memory Accesses in Unmodified Hypervisor

	Total LLC Memory References in the Unmodified (Insecure) Hypervisor							
	Matrix Multiplication Granularity							
Number of VMs/ Partitions	**300x300**	**400x400**	**500x500**	**600x600**	**700x700**	**800x800**	**900x900**	**1000x1000**
1/1	60973	114071	2315516	4054171	5065081	7212454	9121649	11264682
2/2	60974	114070	2315544	4054171	5065083	7212455	9121651	11264683
3/3	60975	114072	2316643	4054294	5065084	7212457	9121653	11264685
4/4	60978	114082	2316651	4054297	5065088	7212461	9121655	11264687
5/5	60979	114084	2316652	4054299	5065089	7212463	9121657	11264688
6/6	60980	114084	2316654	4054200	5065088	7212465	9121658	11264689
7/7	60982	114085	2316658	4054201	5065090	7212466	9121660	11264691
8/8	60984	114086	2316659	4054301	5065091	7212469	9121661	11264693
9/9	60980	114088	2316663	4054302	5065094	7212470	9121664	11264693
10/10	60981	114089	2316664	4054308	5065095	7212471	9121667	11264698
Min	60973	114070	2315516	4054171	5065081	7212454	9121649	11264682
Mean	60978.6	114081.1	2316430.4	5054248.44	5065087.55	7212462.22	9121657.5	11264688.9
Median	60979.5	114084	2316653	4054294	5065088	7212463	9121657.5	11264688.5
Maximum	60984	114089	2316664	4054302	5065094	7212470	9121667	11264698
Std. Deviation	3.60	7.26	474.64	60.43	4.16	5.89	5.70	4.98
Confidence Int.	2.23	4.50	294.18	37.45	2.58	3.65	3.53	3.08

Table 5.25: Last Level Cache (LLC) Memory Accesses in Modified Hypervisor

	Total LLC Memory References in the dynamic partitioned Hypervisor							
	Matrix Multiplication Granularity							
Number of VMs/Partitions	**300x300**	**400x400**	**500x500**	**600x600**	**700x700**	**800x800**	**900x900**	**1000x1000**
1/1	60970	114069	2315503	4054168	5065080	7212452	9121644	11264678
2/2	60969	114068	2315501	4054168	5065079	7212450	9121645	11264677
3/3	60966	114058	2316640	4054291	5065080	7212455	9121644	11264677
4/4	60976	114080	2316648	4054294	5065087	7212459	9121650	11264683
5/5	60972	114082	2316649	4054296	5065088	7212461	9121652	11264684
6/6	60973	114082	2316651	4054197	5065087	7212463	9121653	11264685
7/7	60963	114083	2316655	4054198	5065089	7212464	9121655	11264687
8/8	60965	114084	2316656	4054298	5065090	7212467	9121656	11264689
9/9	60976	114086	2316660	4054299	5065093	7212468	9121659	11264689
10/10	60977	114087	2316661	4054305	5065094	7212469	9121662	11264694
Min	60963	114058	2315501	4054168	5065079	7212450	9121644	11264677
Mean	60970.7	114077.9	2316422.4	4054251.4	5065086.7	7212460.8	9121651	11264684.3
Median	60971	114082	2316650	4054292.5	5065087.5	7212462	9121652	11264684.5
Maximum	60977	114087	2316661	4054305	5065094	7212469	9121662	11264694
Std. Deviation	4.95	9.56	485.13	60.00	5.38	6.70	6.29	5.72
Confidence Int.	3.07	5.93	300.68	37.19	3.33	4.15	3.90	3.54

Table 5.25 shows the average memory access for the varying matrix granularity level. The average memory access of modified (HBP-DCP base) hypervisor is less as compared to the unmodified (default/insecure) hypervisor. Since in modified hypervisor, the cache is divided into partitions, therefore, it effect the average cache access and it will be reduced. The average memory refernces rate is increasing with increasing number of matrix size. For instance the average memory access for 300×300 is less than as compared to 1000×1000. The average standard deviation for unmodified (insecure) hypervisor is 70.83 and for modified (HBP-DCP) is 72.96. This small difference validate the result.

Table 5.26 shows the descriptive statistics of experimental results in unmodified (default/insecure) and HBP-DCP (dynamic partitioned /secure) hypervisors including minimum, maximum, and mean of the total cache references are summarized in eight intensity levels. This statistics shows that there is minor difference between the standard deviation of both hypervisors e.g., .05% difference in both modes is acceptable. As descriptive statistics in the Table 5.26 shows that the difference between both hypervisors is significant even the HBP-DCP based hypervisor has the ability to prevent cross-VM cache-based SC attacks.

The total memory references are calculated by using cachegrind benchmark as shown in Table 5.24 and 5.25. Then these cache references are used to calculate the cache hit rate and average memory access rate. The equations for LLC hit rate and for memory access time are as follows:

$$\text{Cache Hit Rate} = (\text{Cache Hits} / (\text{Cache Hits} + \text{Cache Misses})) \times 100\% \tag{5.7}$$

$$\text{Avg. Memory Access Time} = \text{Hit Rate} \times T_C + \text{Miss Rate} \times M \tag{5.8}$$

Where cache hit rate is calculated by the above Eq.5.7 and miss rate=1-hit rate. T_C is the time to access data from the cache which is mostly 10ns. LL cache miss is the memory accesses percentage that does not find the desired information in the main memory and is determined by the cachegrind benchmark and miss rate is calculated from the formula (1-hit rate).

Table 5.26: Descriptive statistics of LLC Memory Accesses Data Generated by Standard Experimentation

	Mode	Min	Mean	Median	Maximum	Std.Dev	Conf.Int
Mat. Mult.	Unmodified	60967	60967	60975	60983	5.51	1.97
(300×300)	(HBP-DCP)	60963	60970.7	60971	60977	4.95	1.77
Mat. Mult.	Unmodified	114060	114080.1	114084	114091	9.79	3.50
(400×400)	(HBP-DCP)	114058	114077.9	114082	114087	9.56	3.42
Mat. Mult.	Unmodified	1315504	2316425.7	2316653	2316667	485.30	173.66
(500×500)	(HBP-DCP)	2315501	2316422.4	2316650	2316661	485.13	173.60
Mat. Mult.	Unmodified	4054171	4054254.8	5054296	4054312	60.41	21.62
(600×600)	(HBP-DCP)	4054168	4054251.4	4054293	4054305	60.00	21.47
Mat. Mult.	Unmodified	5065081	5065088.7	5065088.5	5065099	5.70	2.04
(700×700)	(HBP-DCP)	5065079	5065086.7	5065088	5065094	5.38	1.92
Mat. Mult.	Unmodified	7212452	7212463.3	7212264	7212476	7.51	2.69
(800×800)	(HBP-DCP)	7212450	7212460.8	7212462	7212469	6.70	2.40
Mat. Mult.	Unmodified	9121649	9121657.4	9121658	9121671	7.07	2.53
(900×900)	(HBP-DCP)	9121644	9121652	9121653	9121662	6.29	2.25
Mat. Mult.	Unmodified	11264681	11264681	11264689	11264703	6.78	2.43
(1000×1000)	(HBP-DCP)	11264677	11264684.3	11264685	11264694	5.72	2.05

M is the time to access information or data from the main memory. Then by using these values, we can calculate the average access time by using our own written customized program. Each level of memory including L1, L2, and L3 (LLC) cache will have different values for these parameters.

Table 5.27 shows the cache hit, miss rate, and cache access time of LLC memory in the unmodified hypervisor. In Table 5.27, the LLC references, LLC misses, and Miss Rate are calculated for unmodified (insecure) by using cache grind benchmark. Then we have written a customized program in order to determine the average cache access time that can be determined by Eq. 5.8. In the customized program, the LLC references and misses are used to calculate LLC hit values by using the LLC hit = LLC references – LLC miss. The LLC hit rate over a period of time is calculated by dividing the cache hits by the combined number of hits and misses and then multiply by 100. Moreover, the LLC hit rate is determined over time by using Eq. 5.7.

Table 5.27: Average Cache Access Rate, Cache Hit, and Miss Rate of LLC in Unmodified (Default/Insecure) Hypervisor

		Average Cache Access Rate of 1VM to 10VMs					
Varying Number of VMs	**Matrix Size**	**LL Cache References**	**LL Cache Hit**	**LL Cache Miss**	**Hit Rate (Read + Write)**	**Miss Rate (Read + Write)**	**Cache Access Time (ns)**
1VM-10VM	300 x 300	60978.6	51777.6	9201	84.91%	15.09%	23.58
1VM-10VM	400 x 400	114081.1	100372.1	13709	87.98%	12.02%	20.82
1VM-10VM	500 x 500	2316430.4	2179943.4	136487	94.11%	5.89%	15.30
1VM-10VM	600 x 600	5054248.4	4863139.4	191109	96.22%	3.78%	13.40
1VM-10VM	700 x 700	5065087.6	4855126.6	209961	95.85%	4.15%	13.73
1VM-10VM	800 x 800	7212462.22	6902988.22	309474	95.71%	4.29%	13.86
1VM-10VM	900 x 900	9121657.22	8841722.22	279935	96.93%	3.07%	12.76
1VM-10VM	1000 x 1000	11264689	11035641	229048	97.97%	2.03%	11.83
Mean		5026204.32	4853838.82	172365.50	0.94	0.06	15.66
Median		5059668.00	4859133.00	200535.00	0.96	0.04	13.80
Std. Deviation		4081294.32	3984289.65	112395.60	0.05	0.05	4.22
Confidence Int.		2828140.70	2760921.13	77884.74	0.03	0.03	2.92

Table 5.28 shows the average access rate for 1VM, 2VMs, 3VMs, 4VMs, 5VMs, 6VMs, 7VMs, 8VMs, 9VMs, and 10VMs in the static partitioned hypervisor. Similarly to unmodified and modified (dynamic partitioned) hypervisors, the LL cache references and LL cache miss are calculated by using cache grind benchmark. The cache hit rate and memory access rate is calculated by using Eq. 5.7 and 5.8 respectively. We calculated LL cache references for 300×300 in single VM then in the case of 2 VMs and up to 10 VMs. Similarly, for each corresponding matrix multiplication workload, we calculated the LL

cache reference in the case of 1VM, 2VM, 3VM, 4VM, 5VM, 6VM, 7VM, 8Vm, 9VM, and 10 VM.

Table 5.28: Average Cache Access Rate, Cache Hit and Miss Rate of LLC in Static Partitioned Hypervisor (1, 2, 4, 8, and 16 partitions)

		Average Cache Access Rate of VM1 to VM10					
Varying Number of VMs	**Matrix Size**	**LL Cache References**	**LL Cache Hit**	**LL Cache Miss**	**Hit Rate (Read + Write)**	**Miss Rate (Read + Write)**	**Cache Access Time (ns)**
1VM-10VM	300 x 300	50920.7	40498.7	10422	79.53%	20.47%	28.42
1VM-10VM	400 x 400	94027.9	77023.9	17004	81.92%	18.08%	26.28
1VM-10VM	500 x 500	1716372.4	1532183.4	184189	89.27%	10.73%	19.66
1VM-10VM	600 x 600	3554201.4	3141106.4	413095	88.38%	11.62%	20.46
1VM-10VM	700 x 700	4465036.7	4003805.7	461231	89.67%	10.33%	19.30
1VM-10VM	800 x 800	6812410.8	6006936.8	805474	88.18%	11.82%	20.64
1VM-10VM	900 x 900	8421601	7530557	891044	89.42%	10.58%	19.52
1VM-10VM	1000 x 1000	9464634	8473379	991255	89.53%	10.47%	19.43
Mean		4322400.61	3850686.36	471714.25	0.87	0.13	21.71
Median		4009619.05	3572456.05	437163.00	0.89	0.11	20.06
Std. Deviation		3644422.22	3256216.36	389674.67	0.04	0.04	3.56
Confidence Int.		2525409.35	2256401.35	270025.81	0.03	0.03	2.47

The total cache references for varying granularity level in the static partitioned hypervisor is less than the dynamic partitioned hypervisor. In the static partitioned hypervisor, if the of VMs is equal to the number of partitions then the cache access rate will be high. However, we have analyzed the average cache access rate in all cases where for one VM there may be 8 partitions or conversely for 8 VMs there may be single partition. This configuration degrade the performance in term of cache access rate.

Therefore, the cache access time in static partitioned hypervisor is less than dynamic partitioned (HBP-DCP) hypervisor.

Table 5.29: Average Access Rate, Cache Hit, and Miss Rate of LLC Memory in Modified (Dynamic Partitioned/HBP-DCP) Hypervisor

		Average Cache Access Rate of VM1 to VM10					
Varying Number of VMs	**Matrix Size**	**LL Cache References**	**LL Cache Hit**	**LL Cache Miss**	**Hit Rate (Read + Write)**	**Miss Rate (Read + Write)**	**Cache Access Time (ns)**
1VM-10VM	300 x 300	60970.7	50748.7	10222	83.23%	16.77%	25.09
1VM-10VM	400 x 400	114077.9	96973.9	17104	85.01%	14.99%	23.49
1VM-10VM	500 x 500	2316422.4	2176433.4	139989	93.96%	6.04%	15.44
1VM-10VM	600 x 600	4054251.4	3713356.4	340895	91.59%	8.41%	17.57
1VM-10VM	700 x 700	5065086.7	4716055.7	349031	93.11%	6.89%	16.20
1VM-10VM	800 x 800	7212460.8	6609186.8	603274	91.64%	8.36%	17.53
1VM-10VM	900 x 900	9121651	8462507	659144	92.77%	7.23%	16.50
1VM-10VM	1000 x 1000	11264684	10705629	559055	95.04%	4.96%	14.47
Mean		4901200.61	4566361.36	334839.25	0.91	0.03	18.29
Median		4559669.05	4214706.05	344963.00	0.92	0.01	17.02
Std. Deviation		4095601.74	3852117.36	259681.19	0.04	0.06	3.87
Confidence Int.		2838055.06	2669332.09	179946.58	0.03	0.04	2.68

Similarly, in Table 5.29, the LLC references, LLC misses, and Miss Rate are calculated for our modified HBP-DCP based (dynamic partitioned/ secure) hypervisor by using cache grind benchmark. Then LLC references and misses are used to calculate LLC hit values by using the LLC Hit = LLC References – LLC Miss. The LLC Hit rate over a period of time is calculated by dividing the cache hits by the combined number of hits and misses and then multiply by 100. The T-value and P-value prove the significant difference between the average LLC memory access time for each VM including 1VM, 2VMs, 3VMs, 4VMs, 5VMs, 6VMs, 7VMs, 8VMs, 9VMs, and 10VMs of modified (dynamic partitioned) and unmodified hypervisors.

Table 5.30 presents the data related to the memory access rate by a matrix program which is collected in the unmodified (insecure/default) and modified (secure/dynamic partitioned) hypervisors execution modes for eight granularity levels of matrices, respectively. This Table summarizes the memory access rate with 95% confidence interval for 30 number of iteration for each VM e.g., 1VM to 10VMs in eight intensity levels. Since the cache access time is dependent on the cache miss rate, therefore, the cache access time is decreasing with increasing number of cache miss rate. The cache miss rate is increasing with increase in the matric multiplication workload. Therefore, the cache access time is decreasing with increasing matrix multiplication workload.

Table 5.30: Average Cache Access Rate of Varying VMs in Unmodified and Modified Hypervisors

Average Cache Access Rate of varying VMs (1VM to 10VMs)			
		Unmodified (Insecure/Default) Hypervisor	**Modified (Dynamic Partitioned/Secure) Hypervisor**
Varying Number of VMs	**Matrix Size**	**Cache Access Time (ns)**	**Cache Access Time (ns)**
1VM-10VM	300 x 300	23.58	25.09
1VM-10VM	400 x 400	20.82	23.49
1VM-10VM	500 x 500	15.30	15.44
1VM-10VM	600 x 600	13.40	17.57
1VM-10VM	700 x 700	13.73	16.20
1VM-10VM	800 x 800	13.86	17.53
1VM-10VM	900 x 900	12.76	16.50
1VM-10VM	1000 x 1000	11.83	14.47
Mean		15.66	18.29
Median		13.80	17.02
Std. Deviation		4.22	3.87
Confidence Int.		2.92	2.68
T-Value		1.959	
P-Value		0.0354	

As shown in Table 5.30, the average LLC memory access time for unmodified (default/insecure) hypervisor is 15.66 and for modified (dynamic partitioned /secure) is 18.29. The almost difference in both is 15.32%, however, this is acceptable because the modified hypervisor has the ability to prevent cross-VM cache-based SC attacks. Moreover, the T-value and P-values prove the significance of the result as the T-value is less than 2.2 and P-value is less than 0.05.

Table 5.31 shows the comparison of average access time of varying number of VMs and partitions in both static partitioned and dynamic partitioned hypervisors. The LL cache access time is calculated based on the total LL cache references. As shown in Table 5.28 and 5.29, the LLC cache references is increasing according to the increasing corresponding matrix multiplication workload. Therefore, cache access time is decreasing with increasing matrix multiplication workload. The effective access time is 17ns. For the static partition, the overhead will be low if the number of VMs is equal to the number of partitions. However, unlike dynamic cache partitioned hypervisor, the number of VMs and partitions cannot be equal in all cases. Since the partitions are configured during boot time. The average access time for static partitioned is 21.71 and for dynamic partitioned is 18.29. The cache access time of our HBP-DCP based hypervisor is improved by 17.1% as the cache access time will be high for the high miss rate. The cache access time is calculated based on the total access rate and miss rate. Since the total cache access rate in static partitioned hypervisor is less than and the miss rate is greater than our dynamic partitioned (HBP-DCP) hypervisor. If the miss rate is high the cache access time will be high. Therefore, the average cache access time of our dynamic partitioned hypervisor (HBP-DCP) is less than static partitioned hypervisor. The T-value and P-value prove the significant difference between both results as the T-value is less than 2.2 and P-value is less than .05.

Table 5.31: Comparison of Average Cache Access Rate of Varying VMs in Static and Dynamic-Partitioned Hypervisors

Average Cache Access Rate of varying VMs (VM1 to VM10)			
		Static-Partitioned Hypervisor	**Dynamic Partitioned Hypervisor (HBP-DCP)**
Varying Number of VMs	**Matrix Size**	**Cache Access Time (ns)**	**Cache Access Time (ns)**
1VM-10VM	300 x 300	28.42	25.09
1VM-10VM	400 x 400	26.28	23.49
1VM-10VM	500 x 500	19.66	15.44
1VM-10VM	600 x 600	20.46	17.57
1VM-10VM	700 x 700	19.30	16.20
1VM-10VM	800 x 800	20.64	17.53
1VM-10VM	900 x 900	19.52	16.50
1VM-10VM	1000 x 1000	19.43	14.47
Mean		21.71	18.29
Median		20.06	17.02
Std. Deviation		3.56	3.87
Confidence Int.		2.47	2.68
T-Value		1.884	
P-Value		0.0431	

We have designed the cache access time model to test the cache access time of a matrix program for each VM e.g., 1VM, 2VMs, 3VMs, 4VMs, 5VMs, 6VMs, 7VMs, 8VMs, 9VMs, and 10VMs based on the four variables namely cache hit rate, cache access time, cache miss rate, and memory access time in nanoseconds. In order to build the statistical model, we have taken 80% data for the training and 20% for the validation. We have taken these two values to train our model as much as possible to avoid biased results. Similar to a statistical model for load testing and cache utilization, in order to present a reliable and accurate estimation model of cache access time, we perform linear regression using measured real data in cache grind benchmark in both unmodified and modified hypervisor. We use data set of cache references in term of cache hit rate and cache miss rate and also cache access time and memory access time and use them for training the regression model to produce the cache access time model.

For validation of our proposed model, we use the split sample approach. Hence the cache access time model can be presented is as follow:

R⟵lm (Memory Access Time~ Hit Time + (Miss Rate × Miss Penalty))

Where Hit time is the time to access the cache and Miss penalty is the time to access the RAM (main memory).

$$\text{AMAT}_m(W_{i)} = \sum_{i=1}^{10}(Hit\ Rate_{VMi}\ \times\ T_{CVMi} +\ Miss\ Rate_{VMi}\ \times M_{VMi}) \qquad (5.9)$$

Where AMAT$_m$ (W$_i$) is the total memory access rate in Nanoseconds calculated by the total amount of hit rate multiplied by the time to access data from the cache. Miss Rate is the memory accesses percentage that does not find the desired information and is determined by the cache grind benchmark. M is the time to access information or data from the main memory. The data generated from a statistical model is given in Table 5.32.

Table 5.32: Regression Statistics Summary of Memory Access Rate for varying VMs

Number of VM	P-Value	R-Squared	Adjusted R-Squared	F-Statistic
1VM	3.138e-05	0.9910	0.9815	1.302e+02
2VM	0.0018928	0.9911	0.9817	3.7e+03
3VM	3.238e-04	0.9981	0.9919	3.349
4VM	1.233e-03	0.9915	0.9989	1.351e+04
5VM	2.2345e-03	0.9819	0.9918	3.7e+04
6VM	2.11185-04	0.9818	0.9917	3.6e+04
7VM	3.223e-04	0.9916	0.9831	1.35e+03
8VM	2.512e-03	0.9917	0.9914	40189e+02
9VM	2.623e-03	0.9913	0.9915	5.824e+04
10VM	4.316e-12	0.9915	0.9918	3.502e+03

The detail statistics of the statistical model of our linear regression are summarized in Table 5.32. The R value shows significance correlation between the cache miss rate and average memory access time. The average R-squared value in the table testifies that

98.95% of the memory access time value can be explained using cache miss and cache hit rate. The F-statistics in the Table ensure that available data is appropriate to be used for linear regression and p-value shows the significance of the result.

5.5 Conclusion

In this chapter, we describe the methodology used to evaluate and validate the result collected from analyzing the performance in two modes of unmodified (default/insecure) and modified (dynamic partitioned /secure) hypervisors. Benchmarking experimentation is the method to evaluate and validate our HBP-DCP prevention mechanism based on three performance parameters: Load testing, cache utilization, and memory access rate for both modes. Moreover, statistical modeling is also performed in order to evaluate and validate the experimental results obtained by benchmarking for both unmodified and modified modes. Furthermore, the observation-based analysis namely regression analysis is used to devise our statistical modeling. The statistical model is validated through the split-sample approach and the results are reported. The result of performance evaluation of our HBP-DCP mechanism is described in the next chapter that will be used to signify the weakness and strength of our proposed prevention mechanism.

CHAPTER 6: RESULTS AND DISCUSSION

In this chapter, we discuss and report the evaluation results of our proposed prevention mechanism through benchmarking experiments and statistical analysis and compare it with the other prevention mechanisms. The data about load testing, cache utilization, and memory access time are presented, analyzed and synthesized for modified and unmodified hypervisors. Finally, the conducted results are validated through statistical modeling using independent replication approach.

The rest of the chapter is organized as follows: The results of benchmarking experiments that analyze the load testing, cache utilization, and memory access time of modified and unmodified hypervisor are described and evaluated in Section 6.1. In Section 6.2, the results are described. In Section 6.3 the performance evaluations are carried out and further discuss is also provided. Finally, Section 6.4 conclude the chapter.

6.1 Performance Evaluation Results

In this section, we present the performance evaluation generated through benchmarking experimental analysis. Our performance evaluation analysis focuses on two features: the successful mitigation or inhibition of cache based SC attacks, and emerging the performance difference between unmodified (default/insecure) and modified (dynamic partitioned/secure/HBP-DCP) hypervisors. The results are revealing the usefulness of our solution in the cloud environment. First of all, we verify that our prevention mechanism is able to prevent cross-VM SC attacks by conducting the attack experiments in both unmodified and modified (HBP-DCP based) hypervisors. We have sent a 20-bit string from one VM to another VM on the separate core and separate physical machine and create a successful communication in the unmodified/insecure hypervisor. We observed the result and describe the vulnerability of the unmodified hypervisor. By contrast, in our secure hypervisor, we tried to implement the same attacks by sending 20-

bit strings but yielded 0 bits of a successful communication across VMs over all twenty attempts in case of our modified hypervisor based on the dynamic cache partition. Moreover, we present and compare the benchmarking results. It contains the four main subsections. In the first, subsection the data related to the bearable load of both hypervisors (modified and unmodified) are presented. While in the second, and third subsection the collected data about cache utilization and memory access time are described respectively. The experimental analysis is conducted to evaluate the performance of the proposed prevention mechanism.

6.1.1 Load Testing

This Section describes the results obtained from the benchmarking tools and statistical modeling. The results are presented in Section 5.5.1 in the previous Chapter 5. In these experiments, we have compared the bearable load of both unmodified and modified hypervisors by sending a various number of concurrent requests and have checked the response time and number of requests per second for each VM.

Figure 6.1 and 6.2 show the bearable load in term of a maximum number of requests per seconds in both unmodified (default/insecure) and static partitioned hypervisor without any VMs and partition. In the Figure 6.1, the y-axis shows the response time and the x-axis shows the number of requests per second. If the number of users are 100 and if we send 10 concurrent requests then the unmodified hypervisor is able to handle 100 to 900 requests in 35ms and 1000 in 40ms. The response time per request is increasing with the increase in the number of concurrent requests in both hypervisors. Figure 6.2 shows the load testing in term of a number of request per second in the static hypervisor without the creation of any VM. Similar to the unmodified hypervisor, if the total request or users in the static partitioned hypervisor is 100 and number of concurrent requests are 10. Then in this hypervisor, 100 requests will be handled in 35ms while the remaining 200 to 800 will be handled in 50ms.

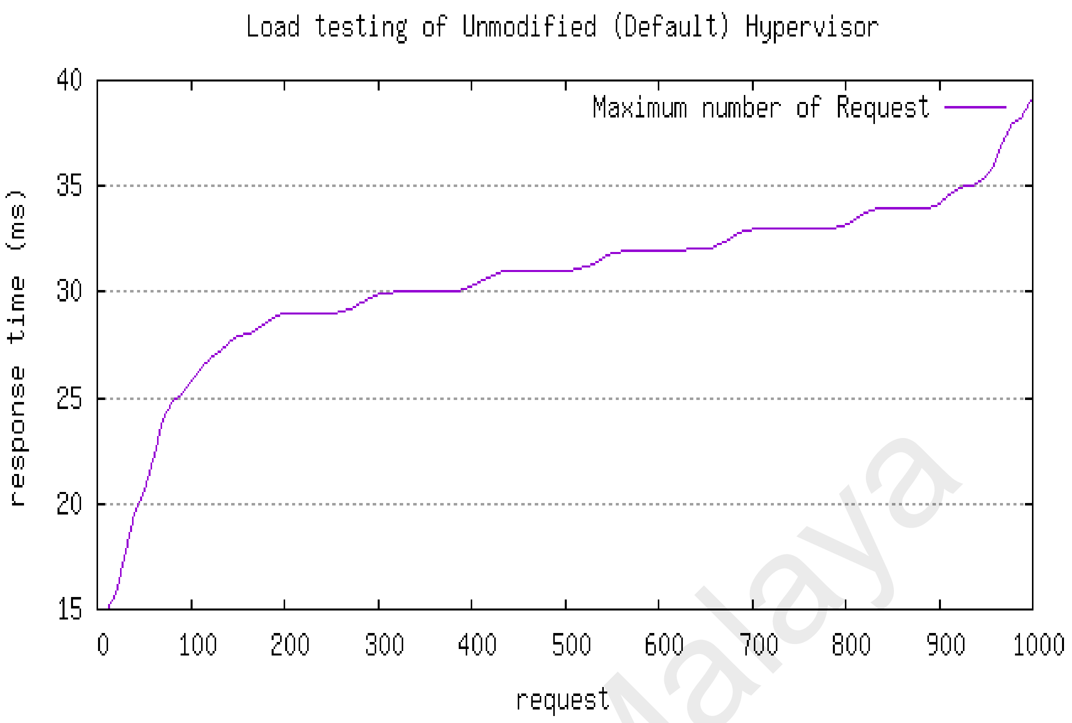


Figure 6.1: Load Testing of Unmodified (Default/Insecure) Hypervisor

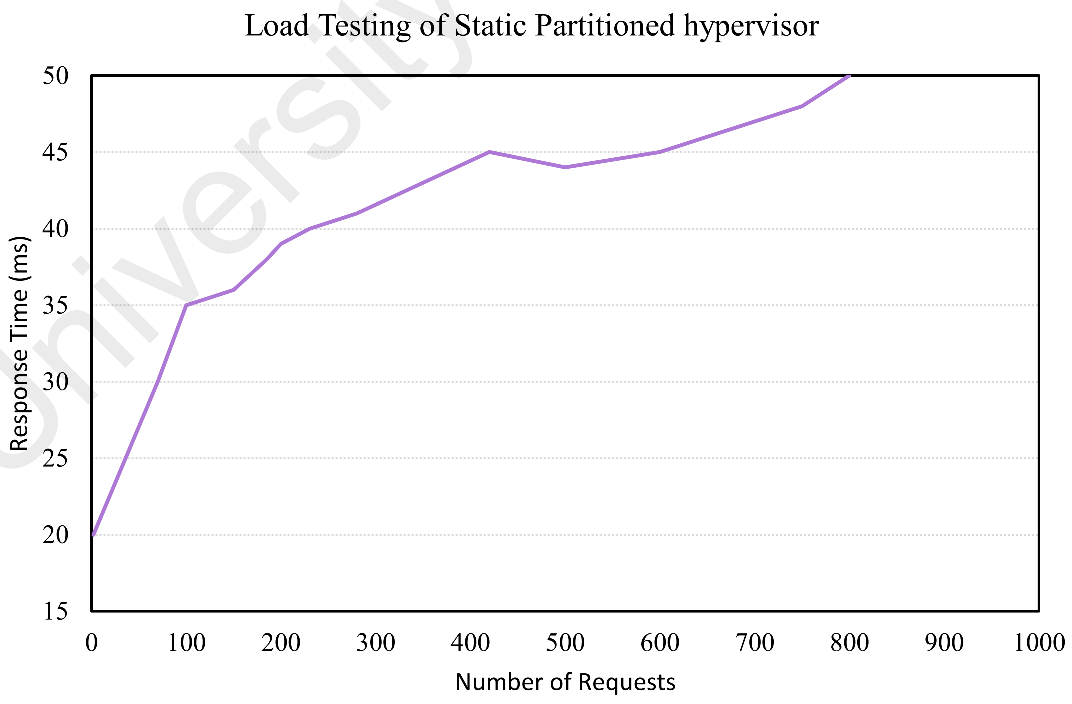


Figure 6.2: Load Testing of Static Partitioned Hypervisor

Figure 6.3 show the load testing in term of number of requests per second and the response time in modified (dynamic partitioned) hypervisor. In the figure, the response time for 600 requests is 35 while 43ms for 900 requests. The maximum number of requests per second is 1000 and is greater than static partitioned hypervisor.

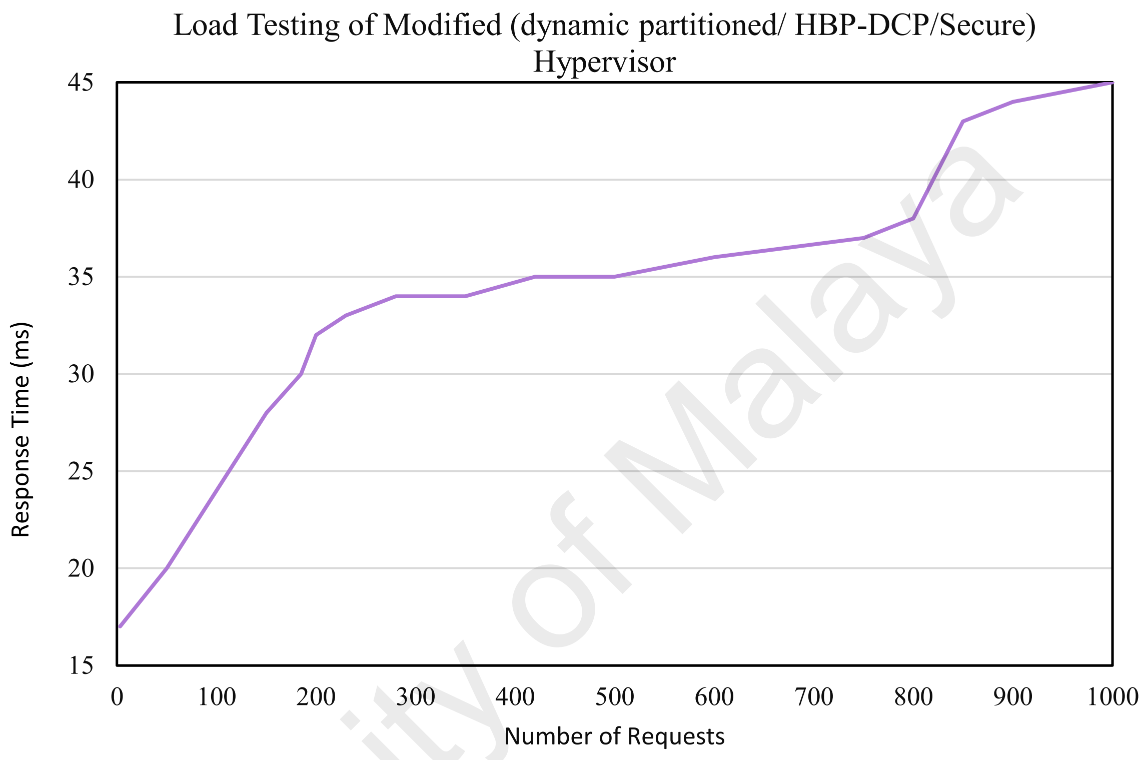


Figure 6.3: Load Testing of Modified (Dynamic Partitioned/Secure/HBP-DCP) Hypervisor

Figure 6.4 shows the comparison of average response time in unmodified, static partitioned, and dynamic partitioned (HBP-DCP) hypervisors. We have compared the response time between all hypervisors without the creation of any VM. In the figure, the x-axis shows the average response time while the y-axis shows the number of concurrent request for 15 data traces including 10 to 150. The average response time in both hypervisors is increasing with the increase in the number of concurrent users. As shown in the graph, the average response time of static partitioned hypervisor is more as compared to our HBP-DCP hypervisor. The response time per request for 150 concurrent requests or users in unmodified hypervisor is 159.041ms, 225.743ms in static partitioned, and 201.03ms in dynamic partitioned hypervisor. Although the response time in the dynamic partitioned (HBP-DCP) hypervisor is a little bit high as compared to the response time of unmodified hypervisor, however, our HBP-DCP based hypervisor has the ability to prevent cross-VM cache-based SC attacks. Since we know that security always comes with some overhead, therefore, the minor changes in the average response time is acceptable. Moreover, we have shown in Table 5.5 in the previous Chapter 5 by using P-value and T-value that the difference in the response time and the number of requests per second between both hypervisors are significant.

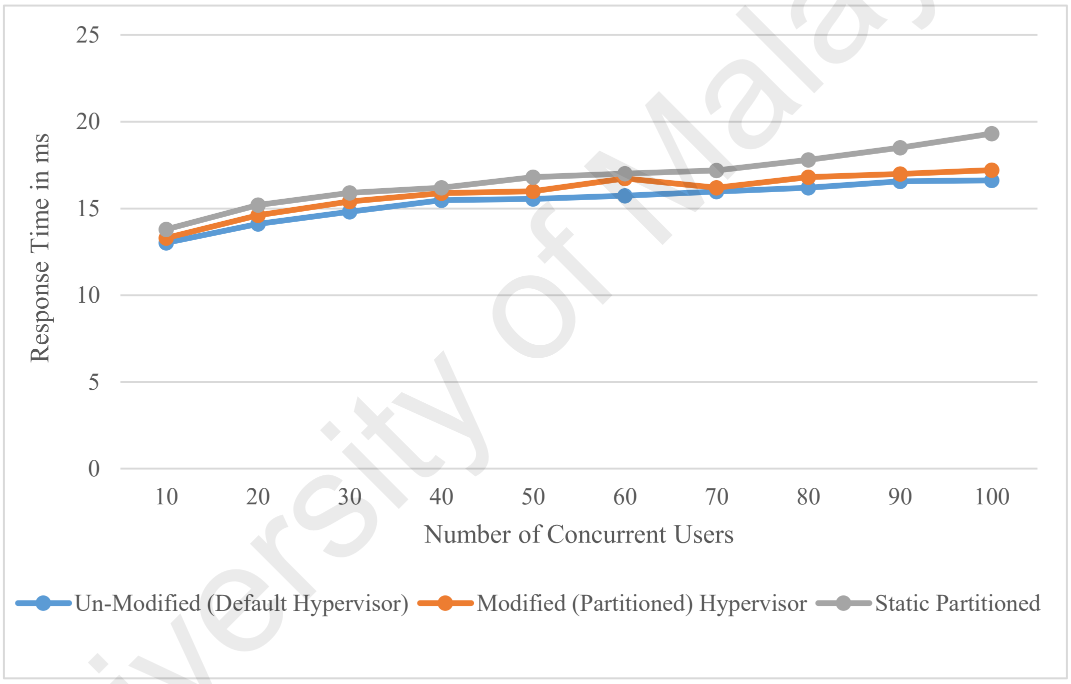


Figure 6.4: Average Response Time for Concurrent Request without VMs for Modified (Default) and Unmodified (Dynamic Partitioned) Hypervisor

Similarly, Figure 6.5 shows the comparison of load testing in term of how many numbers of request per second will be handled by each hypervisors namely: unmodified, static partitioned, and dynamic partitioned (HBP-DCP) without the creation of any VM when the number of concurrent requests or users are increased. In the figure, the x-axis shows the number of concurrent requests while the y-axis shows the number of request per seconds. As shown in Figure 6.4, the average response time is increasing with the increase in the number of concurrent requests. However, the number of requests per second is not increasing with the increase in the number of concurrent requests neither difference in the number of requests for both hypervisors. In contrast to response time, the requests per second is decreasing by increasing number of concurrent requests. The number of request per second in static partitioned hypervisor is less is compared to the unmodified and dynamic partitioned hypervisor. The result obtained for all 10 to 150 concurrent requests in the modified hypervisor is closer to the unmodified hypervisor. However, the modified hypervisor has the ability to prevent cross-VM cache-based SC attacks.

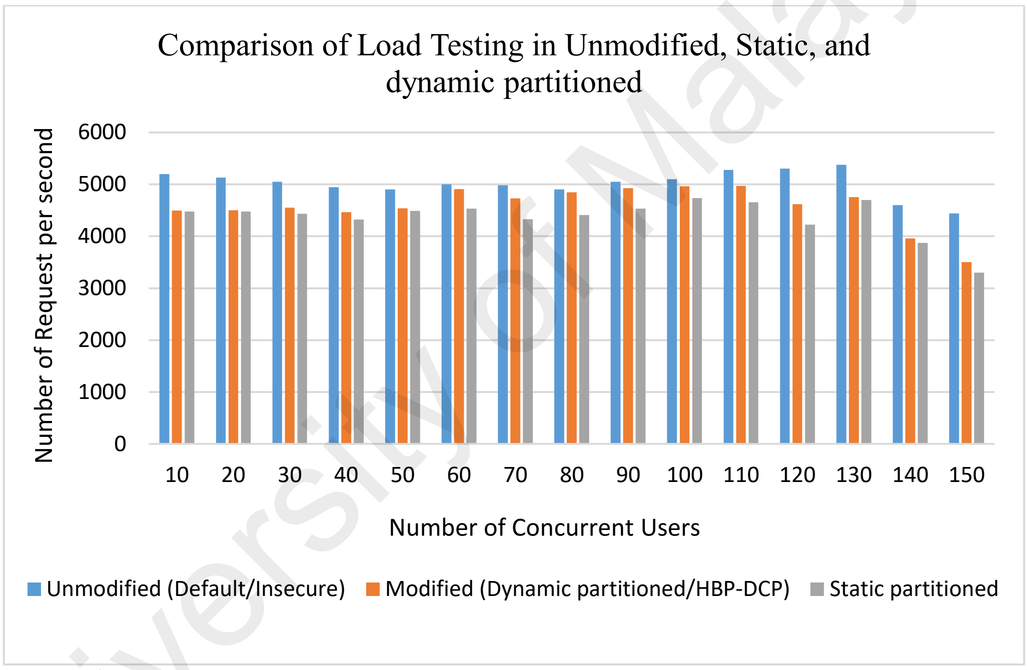


Figure 6.5: Number of Requests per Second Time for (10-150) Concurrent Request without VMs for both Unmodified and Modified (Partitioned) Hypervisor

Figure 6.6 shows the load testing in term of a number of requests per second for concurrent requests with the varying number of VMs for the unmodified hypervisor. In the figure, the y-axis shows the number of requests per second and the x-axis shows the number of virtual machines. We have found and compared the number of request for each VM. As shown in the figure that the number of requests is decreasing as the number of concurrent requests is increasing for each VM including 1VM to 10VMs.

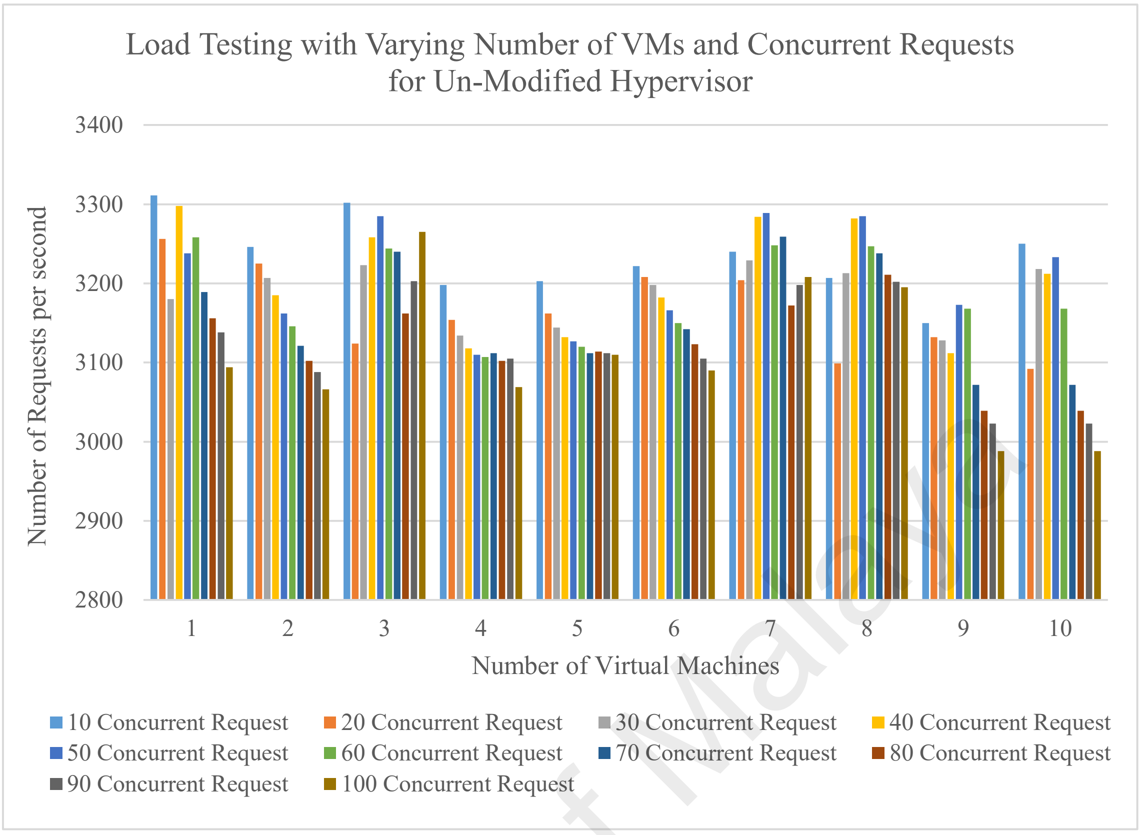


Figure 6.6: Number of Requests per Second in Unmodified (Default) Hypervisor with Varying Number of VMs, Partitions, and Number of Concurrent Requests

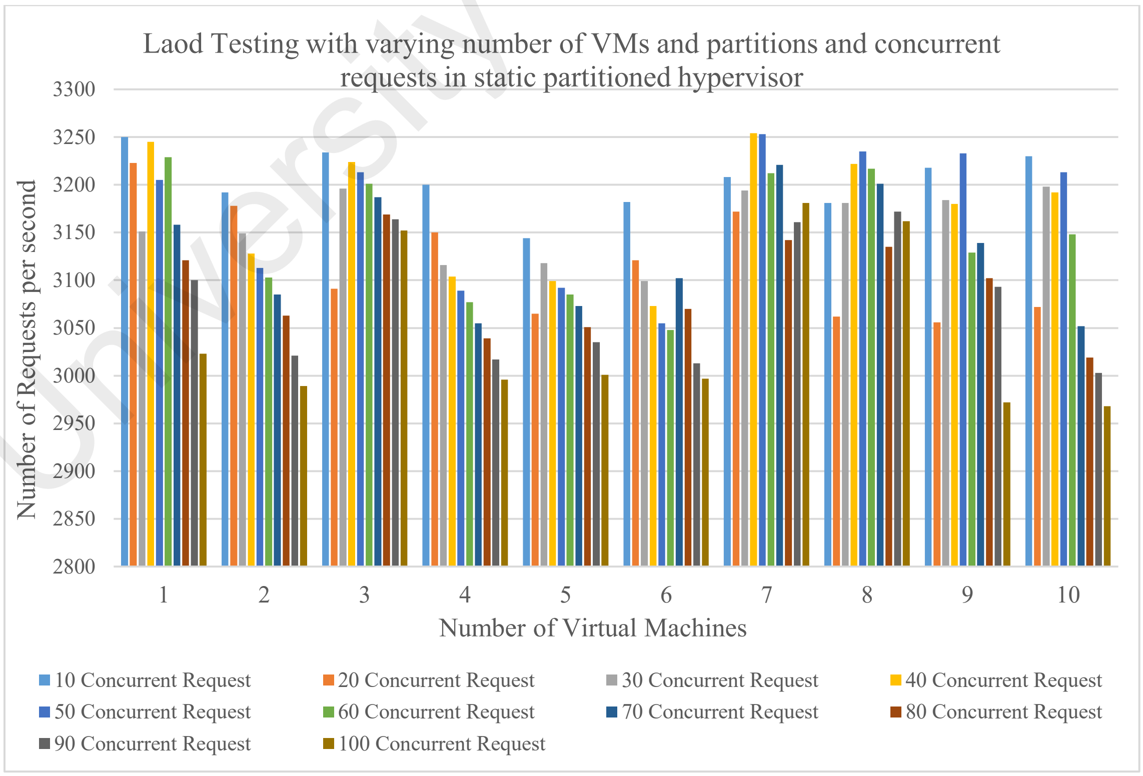


Figure 6.7: Number of Requests per Second in Static Partitioned Hypervisor with Varying Number of VMs, Partitions, and Number of Concurrent Requests

Figure 6.7 shows the load testing in term of a number of requests per second for varying VMs and partitions in the static partitioned hypervisor. Similar to unmodified hypervisor the number of requests per second is decreasing as the number of concurrent users or requests is increasing from 10 to 100 concurrent requests. However, the number of request per second is 3250 which is less as compared to number of requests in the dynamic partitioned hypervisor which is 3303. We have compared the result of static partitioned hypervisor with the dynamic partitioned hypervisor. The difference is almost 50 to 55 number of request per second for each VM. This small amount of difference validates the results collected from the unmodified (insecure) hypervisor when compared with the static partitioned and modified (dynamic partitioned/ secure) hypervisor.

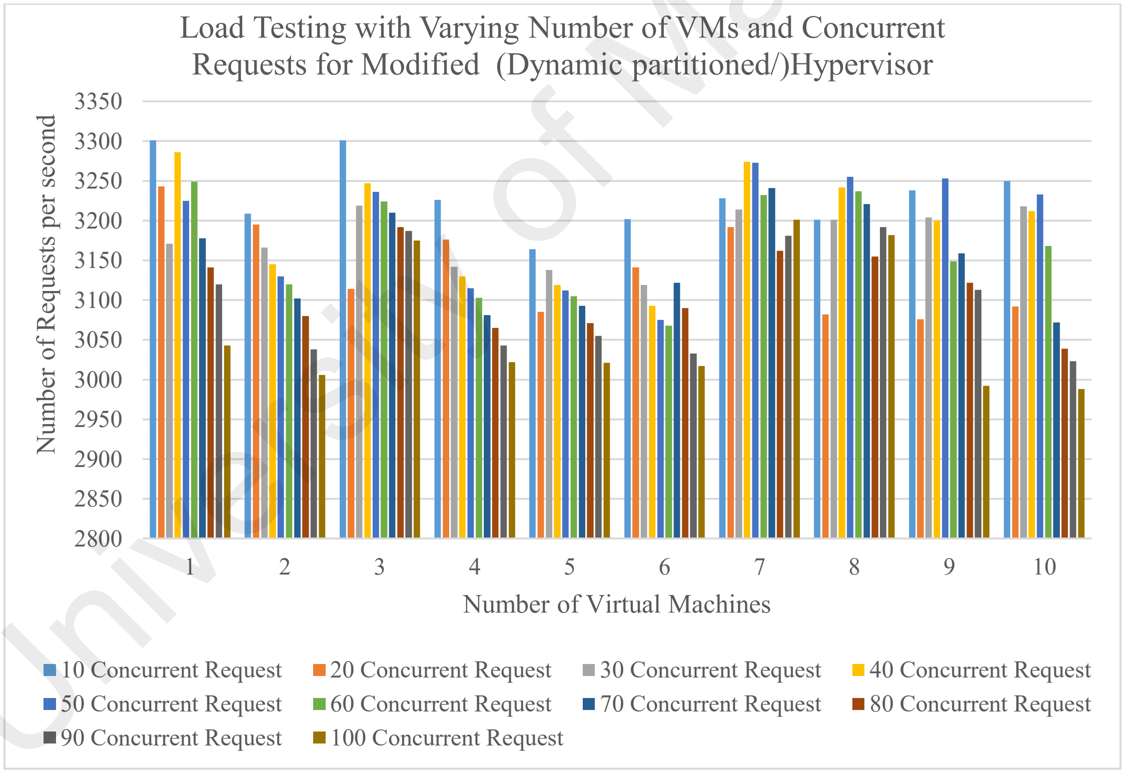


Figure 6.8: Number of Requests per Second in Modified (Dynamic Partitioned/HBP-DCP) Hypervisor with Varying Number of VMs, and Number of Concurrent Requests

Similarly, Figure 6.8 shows the load testing in term of a number of requests per second for varying VMs in the modified (dynamic partitioned/secure) hypervisor. Similar to unmodified hypervisor the number of requests per second is decreasing as the number of concurrent users or requests is increasing from 10 to 100 concurrent requests. We have compared the result of both modified and unmodified hypervisor. The difference is almost 15 to 20 number of request per second for each VM. This small amount of difference validates the results collected from the unmodified (default/insecure) hypervisor when compared with the modified (dynamic partitioned/secure) hypervisor.

Figure 6.9 shows the comparison of average bearable load in term of a number of requests per second for varying VMs and partitions in each hypervisors namely unmodified, static partitioned, and modified (dynamic partitioned) hypervisors. In the figure, the x-axis shows the number of request and the y-axis shows the number of VMs from 1VM to 10VMs. The figure shows that the request per second in the dynamic partitioned hypervisor is greater than static partitioned while less than unmodified hypervisor. However, the P-value and T-value in Chapter 5 prove the significance of the results. Moreover, this result is acceptable as the dynamic partitioned hypervisor prevent the cross VM cache-based SC attacks and improve the security in CC environment.

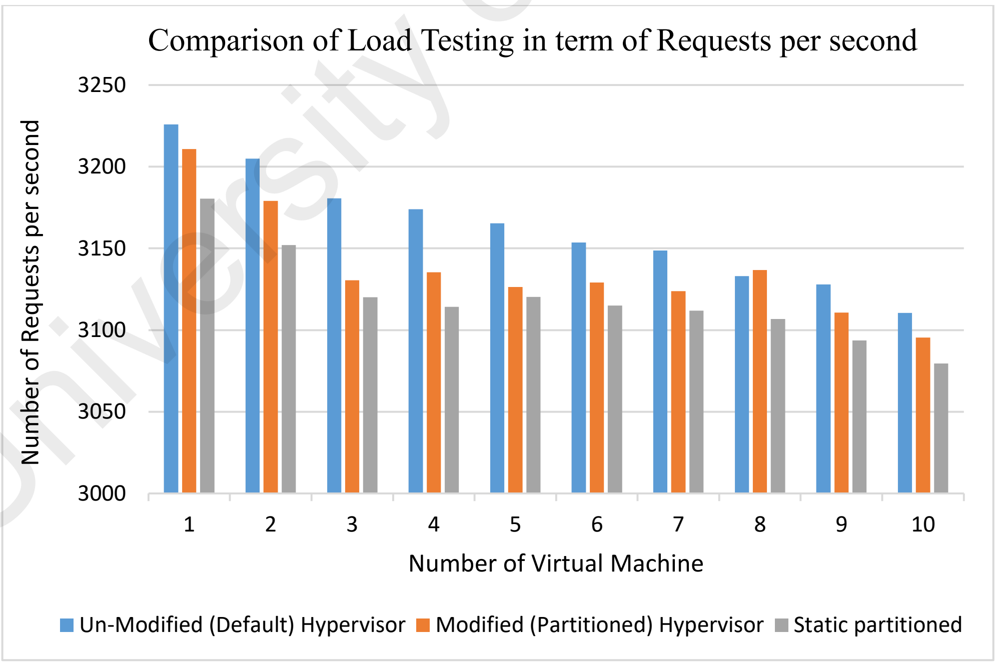


Figure 6.9: Average Number of Request per Second with Varying Number of VMs in Unmodified, Static Partitioned, and Modified (Dynamic Partitioned/HBP-DCP) Hypervisors

The ranges of an average number of requests per second are from 3110.5 to 3225.8 in unmodified and the ranges in the static partition hypervisor are from 3079.5 to 3180.5, and in the modified hypervisor are from 3095.5 to 3210.7. The average difference for all VMs including 1VM to 10VMs is almost 15 in the unmodified and modified (dynamic partitioned). However, the average difference between the static partitioned and dynamic partitioned hypervisor is 30 number of requests per second. This small amount of difference validates the results collected from the modified (secure) hypervisor when compared to the results of the unmodified (dynamic partitioned/ insecure) hypervisor.

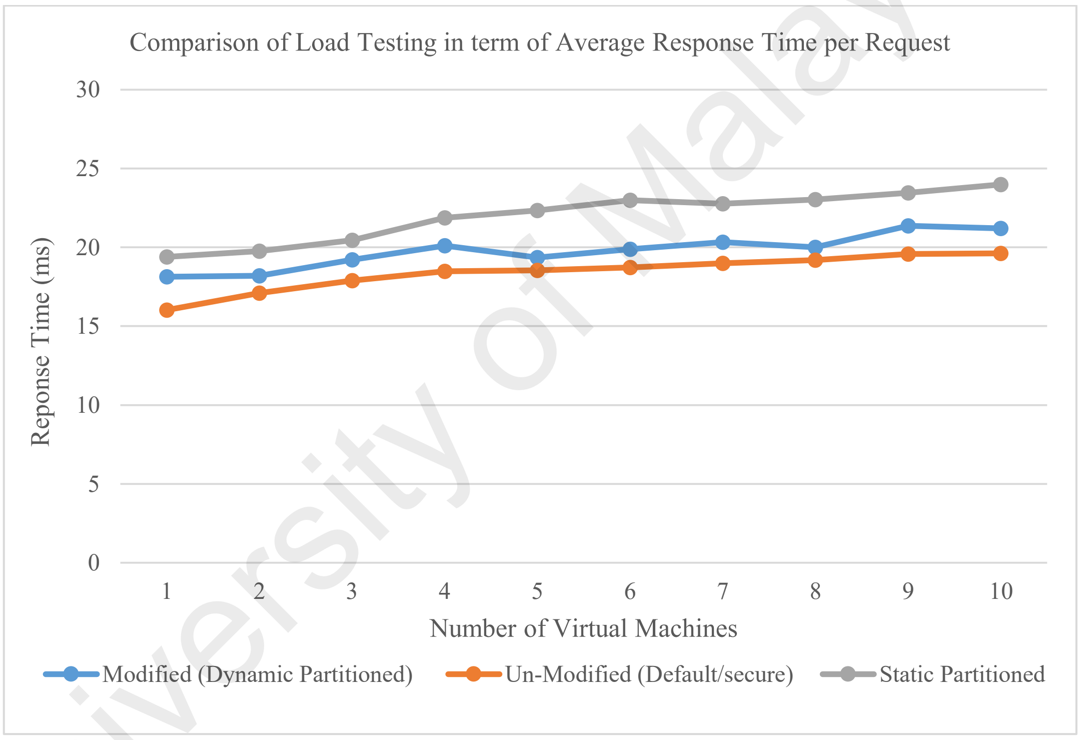


Figure 6.10: Average Response Time for Concurrent Request with Varying Number of VMs in Unmodified (Default) and Modified (Dynamic Partitioned) Hypervisor

Figure 6.10 shows the average response time for varying number of VMs, partitions, and concurrent requests. In the figure, the y-axis shows the response time per request and the x-axis shows the number of VMs from 1VM to 10VMs. In contrast to number of request per second in Figure 6.9, here the response time is increasing as the number of concurrent requests and VMs is increasing. The response time for 5VMs and 8VMs is almost same. However, in the other VMs, the small amount of difference validates the dynamic partitioned hypervisor results with the ones collected from the unmodified and static partitioned hypervisors.

6.1.2 Cache Utilization

In this section, we analyze the cache utilization in term of the Read/Write/Modify bandwidth, read, and write bandwidth calculated by cache Read/Write/Modify, cache read, and cache write benchmarks in both unmodified and modified hypervisors. Figure 6.11 and 6.12 present the data related to the comparison of cache utilization in term of Read/Write/Modify bandwidth in each hypervisor namely unmodified, static partitioned, and dynamic partitioned. We have executed the experiment for 30 execution traces for each VM and with varying number of VMs. The aim of this analysis is to compare the status of cache in the unmodified (default/insecure), static partitioned, and modified (dynamic partitioned/secure) hypervisors. Dynamic partitioned hypervisor or HBP-DCP based on dynamic cache partitioning is our solution. The figures clearly represent that there is a very small amount of difference in the cache utilization of both hypervisors even the modified hypervisor has the ability to prevent cross-VM cache-based SC attacks. In Figure 6.11, the y-axis shows the average bandwidth of cache read/modify/write and the x-axis shows the number of varying VMs. Here in this figure we did not mention the partitions since the partitions in the dynamic partitioned (HBP-DCP) is decided during runtime. While in Figure 6.12, the y-axis shows the average bandwidth in MB/Sec for read/write/modify and the x-axis shows the number of varying VMs and partitions. Here in this figure, we mentioned the partitions since the partitions are decided during boot time and cannot be changed during execution or runtime. The average bandwidth ranges of cache read/write/modify for the unmodified hypervisor is from 25542 to 25683 while for unmodified hypervisor the range is from 25522 to 25645. The highest value of the unmodified hypervisor is 25683 MB/Sec that is bigger than modified (HBP-DCP) by 38 MB/Sec. Moreover, Read/Write/Modify bandwidth is almost same in both unmodified (secure) and modified (dynamic partitioned/HBP-DCP/secure) hypervisor. However, the bandwidth in both hypervisors including HBP-DCP and in unmodified is decreasing as the number of VMs is increasing. However, the significant difference as shown in the previous chapter validate the results and enable the cloud providers to use the modified hypervisor based on HBP-DCP to improve the security of virtualized environment.

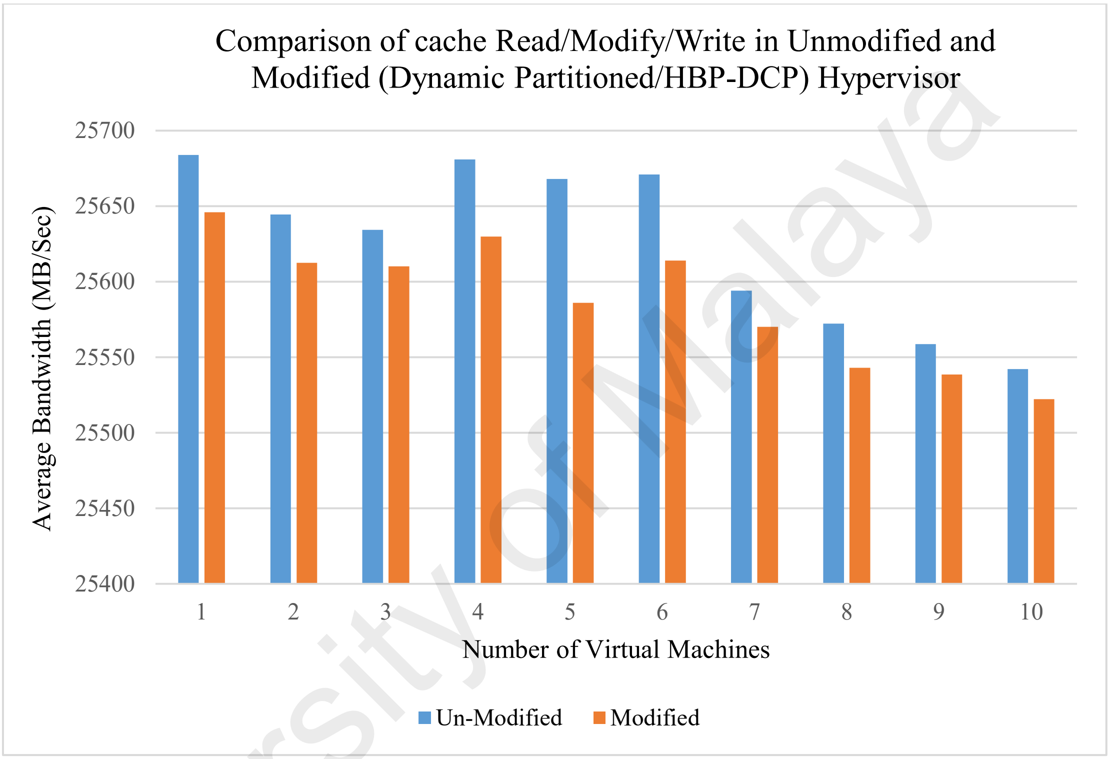


Figure 6.11: Cache Read/Modify/Write Bandwidth in Unmodified and Modified (Dynamic Partitioned) Hypervisors

Similarly Figure 6.12 shows the comparison of cache Read/Modify/Write bandwidth in the static partitioned and dynamic partitioned hypervisors. In contrast to the dynamic partitioned hypervisor, the number of partitioned is predefined in the static partitioned hypervisor. Therefore, we have taken 1,2,4,8, and 16 partitions for both static partitioned and dynamic partitioned hypervisors. There is a big difference in both hypervisors even for the same VM and partitions as shown in the figure. Since the partitions in the static partitioned based hypervisor cannot be changed, therefore, the performance is degraded in the static partitioned hypervisor as compared to our HBP-DCP solution. For instance, in static partitioned hypervisor, if we make 16 partitions and is only one VM is executing the only single partition will be assigned to that VM and the remaining 15 partitions will be idle. While in case of our dynamic partitioned (HBP-DCP), the partitions is defined during runtime according to running VMs, therefore, if one VM is running the whole cache would be assign to that single VM. Consequently improve the performance in term of cache utilization. The average bandwidth of cache read/modify/write in the static partitioned hypervisor is 13012.8 and 18234.7 in our dynamic partitioned (HBP-DCP) hypervisor. Thus the average bandwidth of cache write is improved by 43.32% in our HBP-DCP based hypervisor. Consequently improves the cache utilization. Because in the static partitioned hypervisor, the number of partitions is static and predefined during boot time**.** We have shown the significance of the results in the previous Chapter 5 by using P-value and T-values.

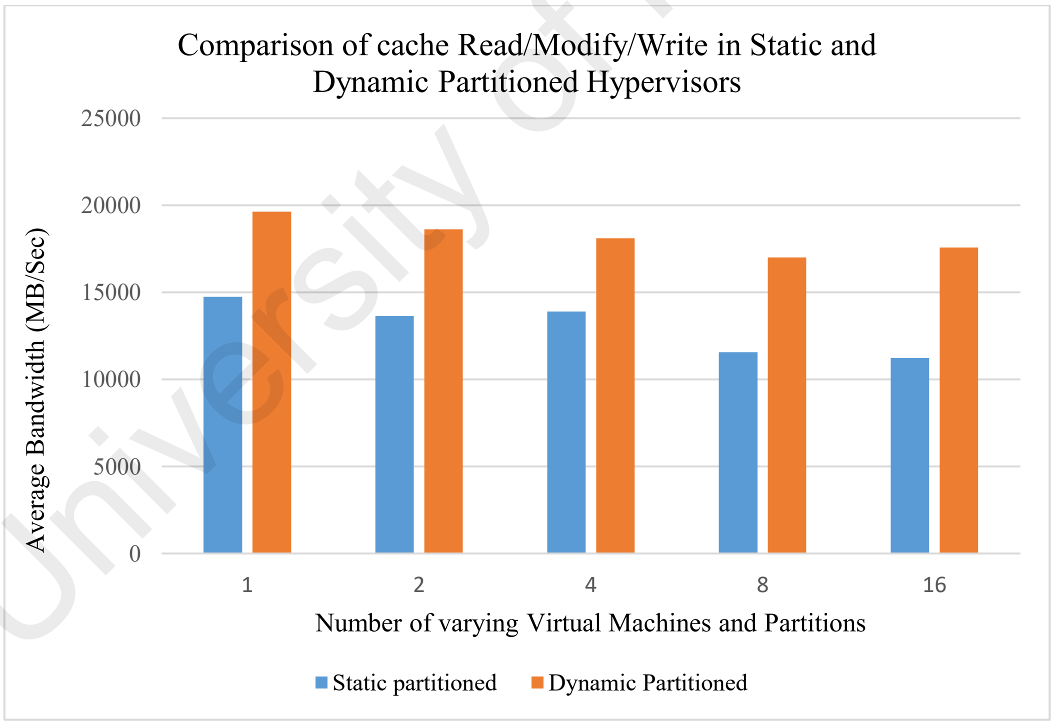


Figure 6.12: Cache Read/Modify/Write Bandwidth in Static Partitioned and Modified (Dynamic Partitioned/HBP-DCP) Hypervisors

Similarly, Figure 6.13 shows the bandwidth of cache read in both unmodified and modified (HBP-DCP based) hypervisors. The y-axis shows the bandwidth for cache read and the x-axis shows the number of VMs. The read bandwidth in both hypervisors is decreasing with the increasing number varying VMs. We have executed the experiment for 30 execution traces for each VM and with varying number of VMs. The aim of this analysis is to compare the status of cache read bandwidth in the unmodified (default/insecure) and modified (dynamic partitioned/secure) hypervisors after HBP-DCP based on cache partitioning as our solution. The figures clearly represent that there is a very small amount of difference in the cache utilization in term of cache read of both hypervisors and also in the previous chapter we have shown that the difference is significant even the modified hypervisor has the ability to prevent cross-VM cache-based SC attacks.

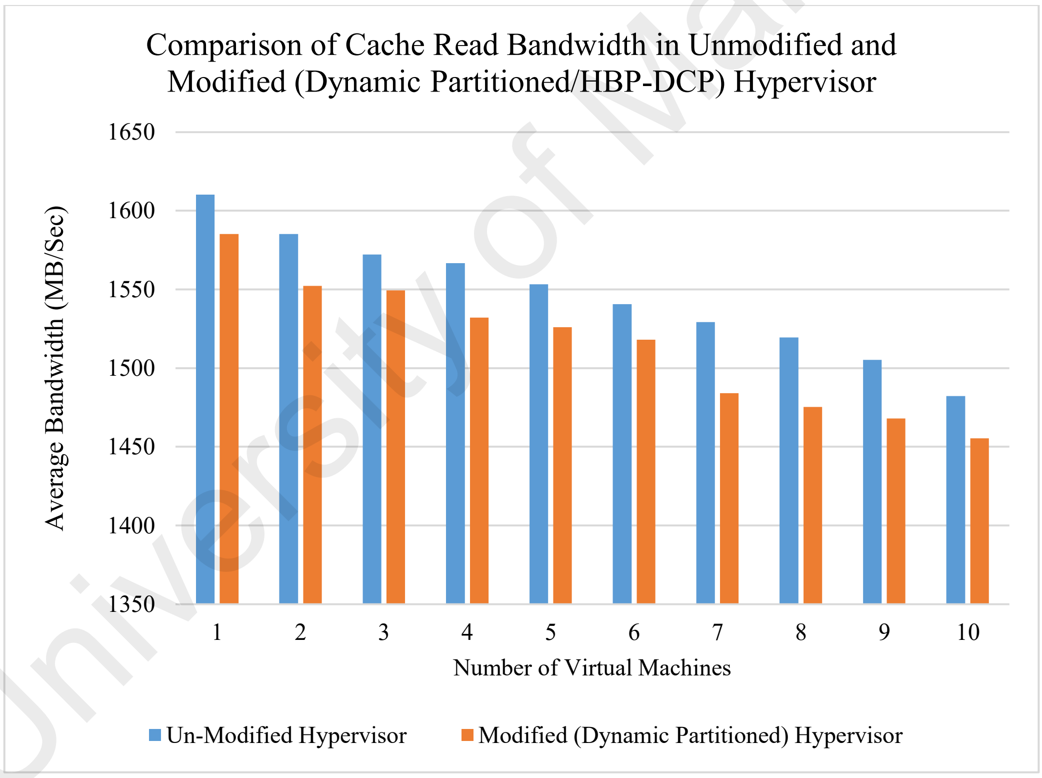


Figure 6.13: Cache Read Bandwidth in Unmodified and Modified (Dynamic Partitioned/HBP-DCP) Hypervisor

The ranges of average bandwidth for cache read is from 1482 to 1610 in the unmodified (insecure/default) hypervisor and from 1455 to 1585 in the modified (secure/partitioned) hypervisor based on HBP-DCP. The difference is almost same and we have shown in the previous chapter that the difference between both hypervisors is significant. There is a little difference between both, however, modified hypervisor based on HPB-DCP has the ability to prevent cross-VM cache-based SC attacks.

Similarly, the comparison of cache read bandwidth in both static partitioned and dynamic partitioned hypervisors is shown in the Figure 6.14. Similar to Figure 6.13, the average bandwidth of cache read in static partitioned hypervisor is low as compared to our HBP-DCP based hypervisor due to the predefined static partitions during boot time.

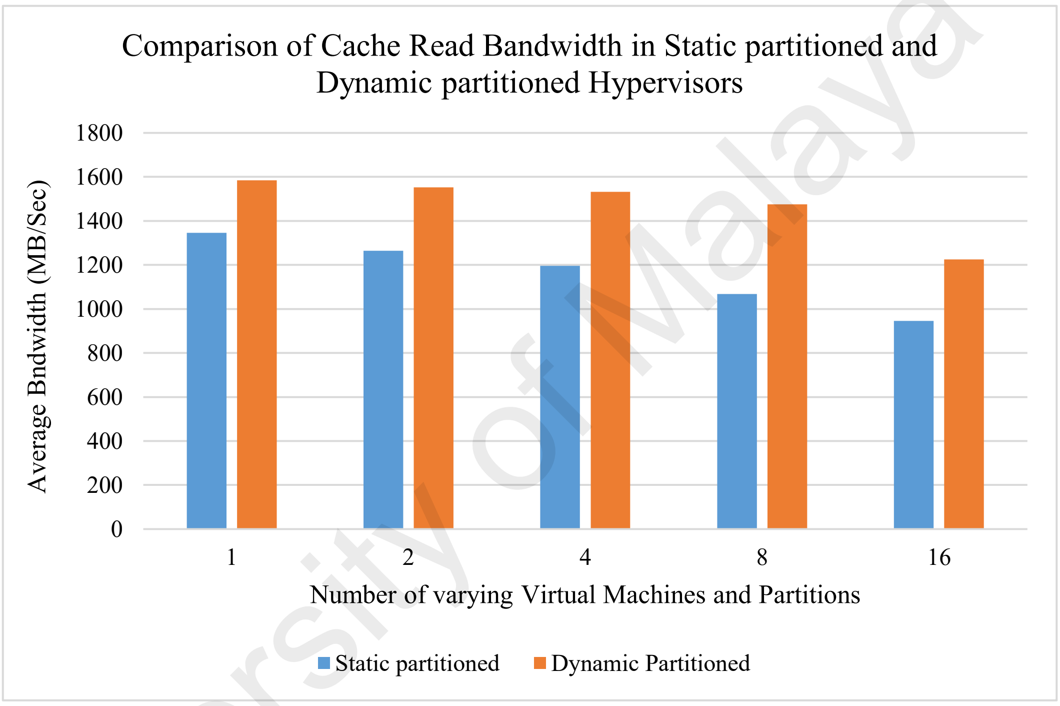


Figure 6.14: Cache Read Bandwidth in Static Partitioned and Modified (Dynamic-Partitioned/HBP-DCP) Hypervisor

The ranges of average bandwidth for cache read are from 946.3 to 1345.5 in the static partitioned hypervisor and the ranges in the modified (secure/dynamic partitioned) hypervisor based on HBP-DCP are from 1225.375 to 1585.212. The p-value and t-value in Chapter 5 validate the significance of the result. We have improve the cache utilization by using dynamic partitioned hypervisor up to 45%.

Figure 6.15 shows the bandwidth for cache write in both unmodified and modified hypervisors. The y-axis shows the bandwidth for cache write and the x-axis shows the number of VMs. The cache write bandwidth in both hypervisors are decreasing with the increasing number varying VMs. We have executed the experiment for 30 execution traces for each VM and with varying number of VMs. The aim of this analysis is to compare the status of cache read bandwidth in the unmodified (default/insecure) and modified (partitioned/secure) hypervisors after HBP-DCP based on cache partitioning as our solution. The difference in both hypervisors is significant, even the modified hypervisor (HBP-DCP) has the ability to prevent cross-VM cache-based SC attacks.

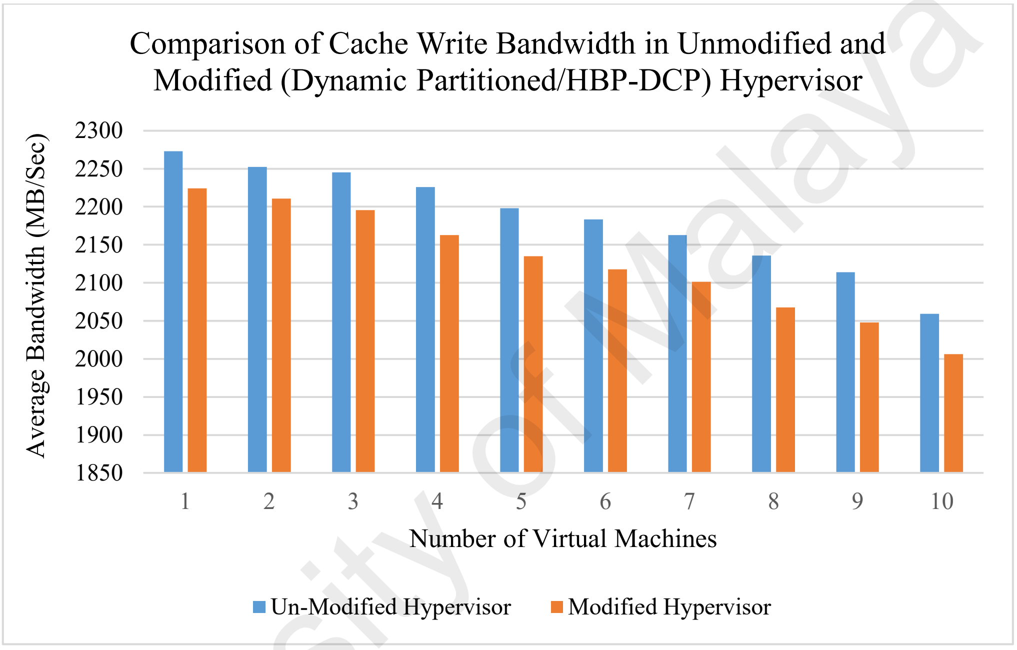


Figure 6.15: Cache Write of Unmodified and Modified Hypervisor

Figure 6.16 shows the comparison of the cache write bandwidth in both static partitioned and modified (dynamic partitioned/HBP-DCP) hypervisors. The cache write bandwidth ranges for static partitioned hypervisor are from 998.541 to 1710.201 and are from 1621.232 to 2024.191 in the dynamic partitioned hypervisor. As shown in the figure the average bandwidth for cache write in the static partitioned hypervisor is less than from HBP-DCP (dynamic partitioned) hypervisor. Since for 16VMs if there are single partition in the static partitioned hypervisor, then it will be difficult to maintain the writing in the small part of cache for the 16 VM as compared to write in 16 partitions for 16VMs in dynamic partitioned (HBP-DCP) hypervisor. Because in static partitioned hypervisor, the partition is predefined during boot time while in dynamic partitioned, the partitioned will be created during runtime according to the number of VMs.

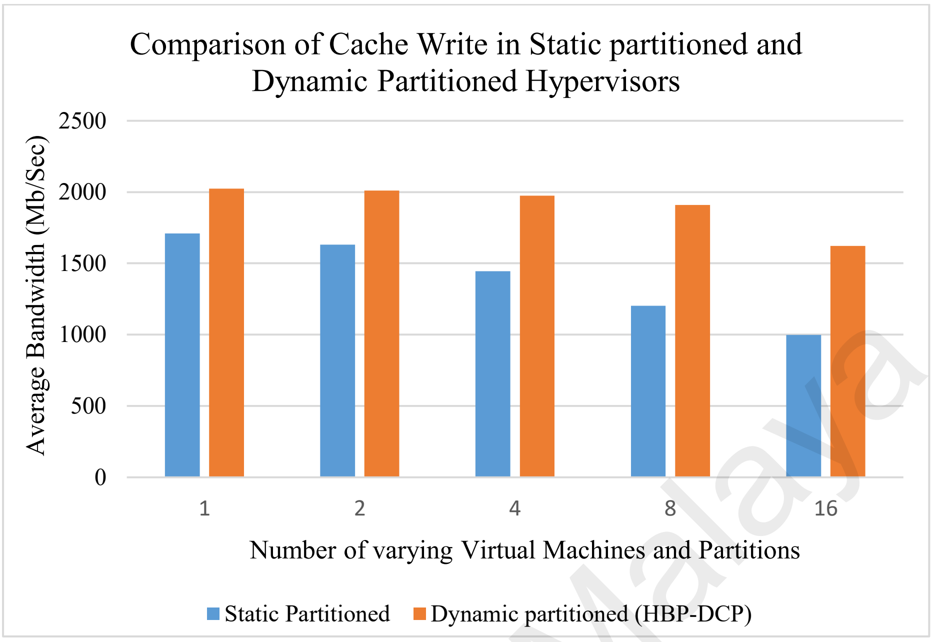


Figure 6.16: Cache Write of Static Partitioned and Dynamic Partitioned (HBP-DCP) Hypervisor

6.1.3 Memory Access Rate

In this section, we compare the performance of modified hypervisor based on the HBP-DCP mechanism with the unmodified hypervisor and with the static partitioned hypervisor. The parameters used for the comparison is the memory access time calculated from the cache hit and miss rate. The average memory access time is a valuable parameter to evaluate the performance of a memory hierarchy configuration. Figure 6.12 and 6.13 present the graph related to the LLC memory references by a matrix program which are collected in the unmodified (insecure/default) and modified (secure/partitioned) based on HBP-DCP hypervisors execution modes for eight granularity levels of matrices, respectively. The aim of finding the total LLC memory references is to analyze the total cache access time for varying granularity level. In the previous chapter, we have given the detail for cache access time. For cache access time, we have to find the total cache references, cache hit, and cache miss rate. We have found these values by using cachegrind benchmark and the using these values in our designed program to calculate the average memory access time. Each Figure summarizes the memory access rate in term of total LLC memory access with 95% confidence interval for 30 number of iteration for each VM e.g., VM1 to VM10 in eight intensity levels.

Figure 6.17 shows the LLC memory references or cache access for matrix multiplication program with 8 granularity level. In the Figure, the y-axis shows the total memory accesses and the x-axis shows the number of VMs from VM1 to Vm10. Each diagonal bar in the figure represents the mean value of LLC memory reference or accesses measured in the unmodified hypervisor mode of 30 iterations for each corresponding matrix multiplication workload (300x300 to 1000x1000). The LLC memory references are increasing with increasing number of workload for each VM in the unmodified hypervisor.

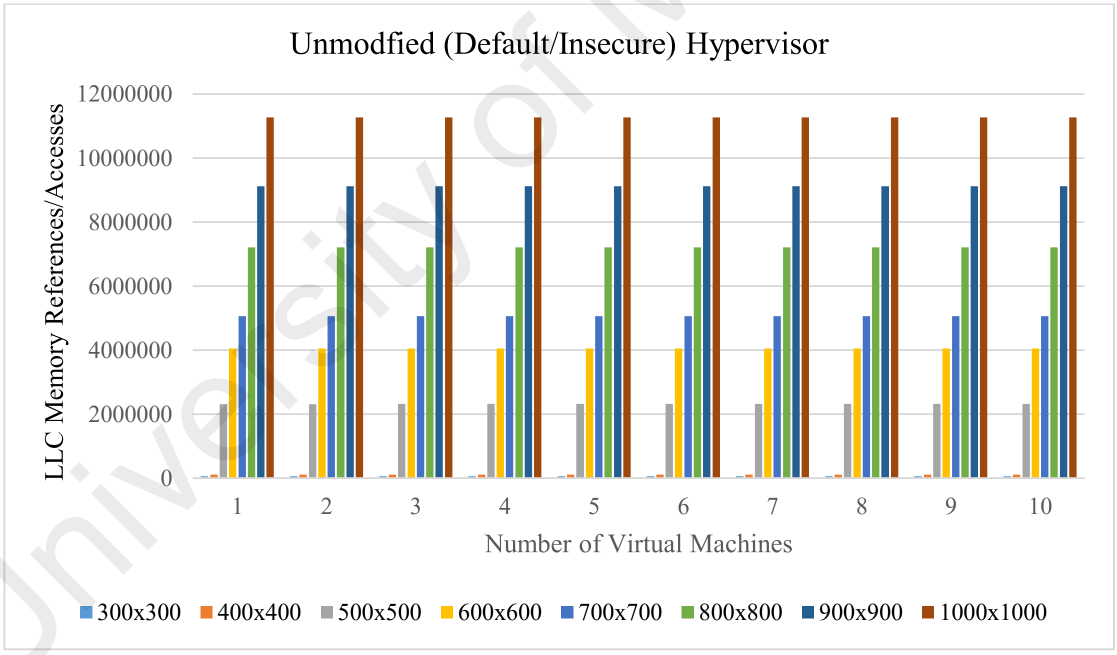


Figure 6.17: Average LLC Memory References in Unmodified Hypervisor for Varying VMs (1VM-10VMs)

Figure 6.18 shows the LLC memory references or cache access for matrix multiplication program with 8 granularity level in the static partitioned hypervisor. As compared to the unmodified and dynamic partitioned hypervisor, the average memory accesses is low. Since the partitions cannot be changed and for the configuration of 1VM and 16 partitions the VM will access just single partition and the remaining 15 partitions will be idle during VM execution. Therefore, the memory reference will be low in the static partitioned hypervisor. In the Figure 6.18, the y-axis shows the total memory accesses and the x-axis shows the number of VMs from VM1 to Vm10. Each diagonal bar in the figure represents the mean value of LLC memory reference or accesses measured in modified hypervisor mode of 30 iterations for each corresponding matrix multiplication workload (300x300 to 1000x1000). The LLC memory references are increasing with increasing number of workload for each VM in the modified hypervisor. For instance, the memory references for 1000×1000 are greater than 300×300 workload references.

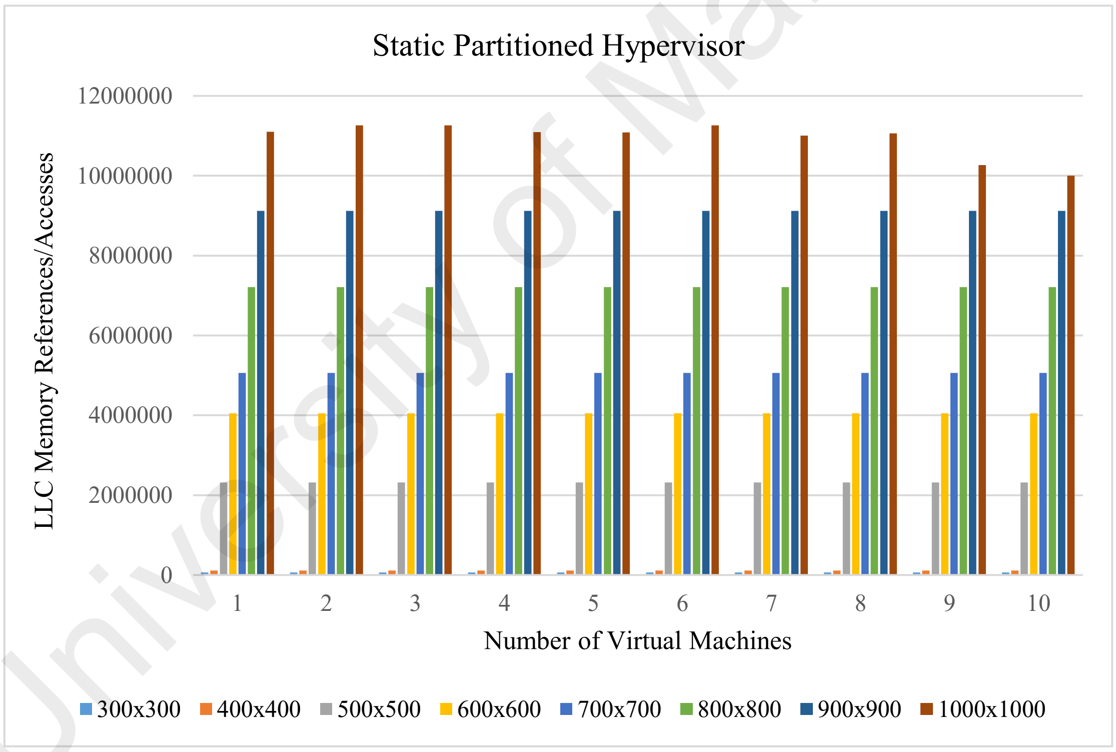


Figure 6.18: Average LLC Memory References in Static Partitioned Hypervisor for Varying VMs (1VM-10VMs)

Figure 6.19 shows the LLC memory references or cache access for matrix multiplication program with 8 granularity level in modified (HBP-DCP) hypervisor. In the Figure, the y-axis shows the total memory accesses and the x-axis shows the number of VMs from VM1 to Vm10. Each diagonal bar in the figure represents the mean value of LLC memory reference or accesses measured in modified hypervisor mode of 30 iterations for each corresponding matrix multiplication workload (300x300 to 1000x1000). The LLC memory references are increasing with increasing number of workload for each VM in the modified hypervisor. In the previous chapter, we have shown that the difference between both hypervisors is significant even though the modified (HBP-DCP) hypervisor has the ability to prevent cross-VM cache-based SC attacks.

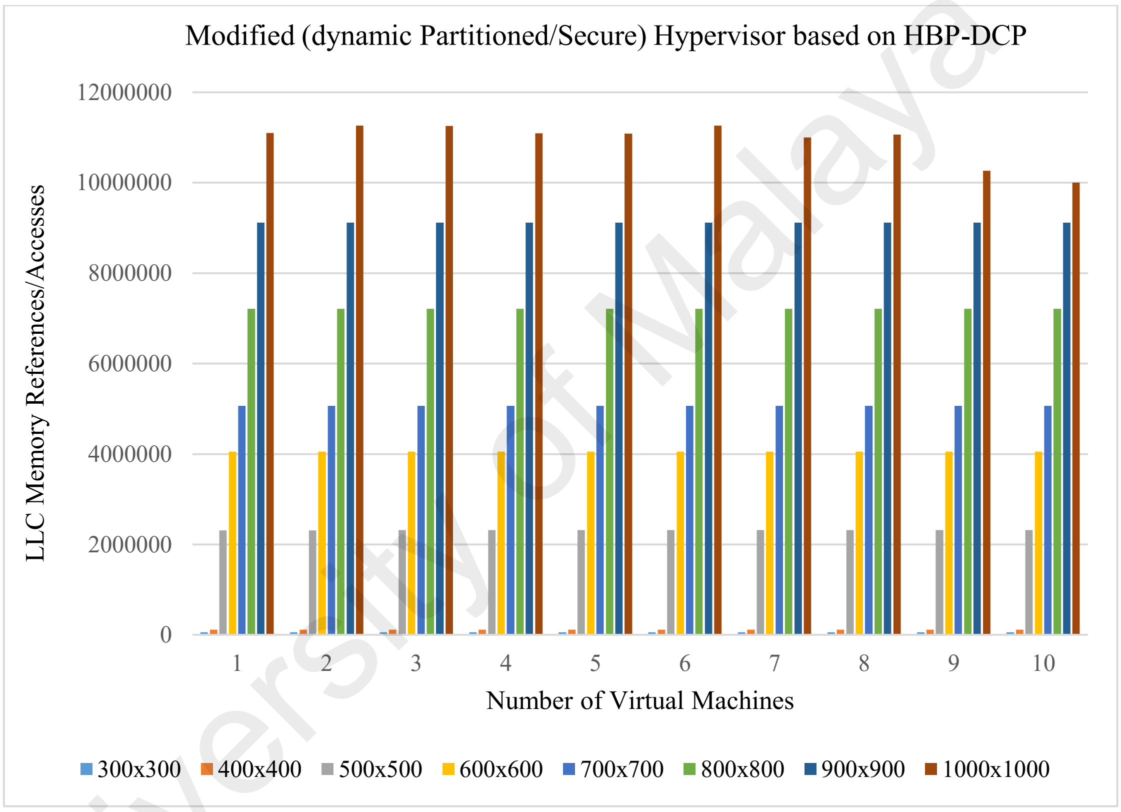


Figure 6.19: Average LLC Memory References in HBP-DCP based Hypervisor for Varying VMs (1VM-10VMs)

Figure 6.20 shows the comparison of average LLC memory access time in unmodified, static partitioned, and dynamic partitioned (HBP-DCP) hypervisors with varying number of VMs and eight different granularity level. In the Figure, the y-axis represents the total cache references and the x-axis represents the various granularity level of matrix multiplication for both unmodified and HBP-DCP based hypervisor. The diagonal bar represents the average LLC cache references for the varying workload and varying VMs from 1VM-10VMs. The graph in Figure clearly depicts the increasing complexity as the matrix multiplication intensity from left to right be increases in both hypervisors. However, the growth of workload has a significant impact on the total cache references when the workloads are executed in both modified and HBP-DCP hypervisor. In Chapter 5, we have also shown that the differences between the total cache references in both modes are significant. Moreover, the modified HBP-DCP based hypervisor has the ability to prevent cross-VM cache-based SC attacks.

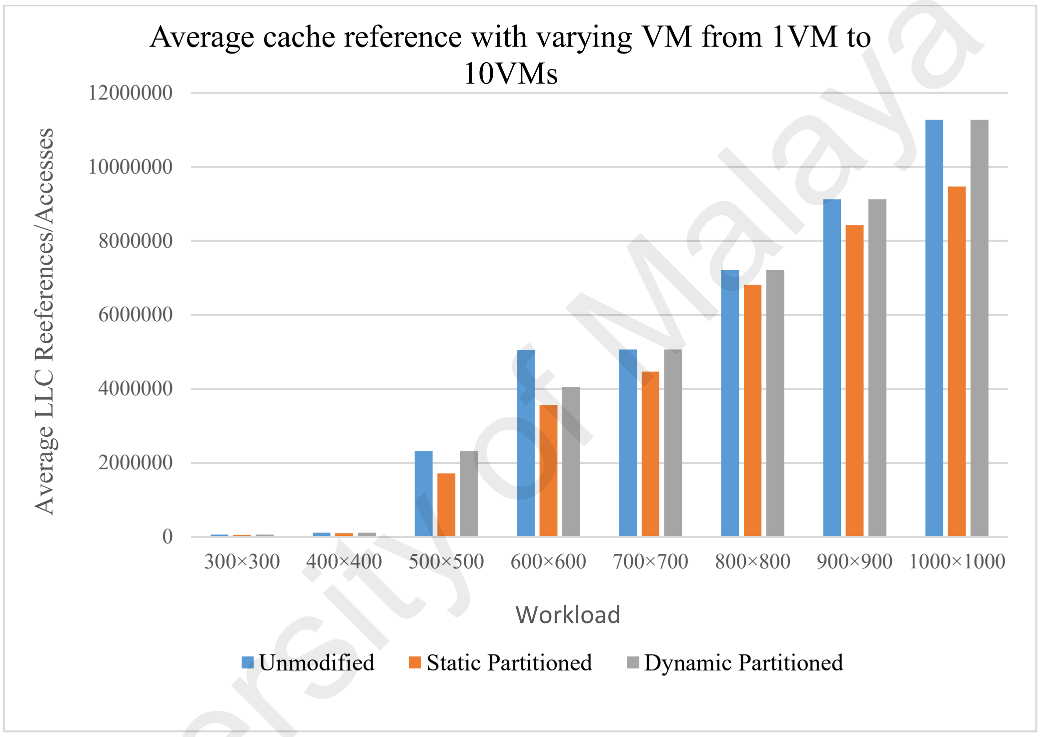


Figure 6.20: Comparison of LLC Memory References in Unmodified, Static and HBP-DCP based (Dynamic partitioned) Hypervisors

Figure 6.21 show the LLC memory hit rate in three hypervisors namely: unmodified static partitioned, and dynamic partitioned or HBP-DCP based hypervisors. In the figure, the y-axis represents the hit rate and the x-axis shows the workload in term of varying granularity level. The hit rate for the unmodified hypervisor is greater than as compared to the modified (HBP-DCP based) hypervisor. Moreover, the hit rate is increasing with increasing matrix granularity. The ranges of LLC hit rate for unmodified hypervisor are from 87.97% to 98.56% and for modified (HBP-DCP based) hypervisor are from 84.84% to 96.14%. There is almost 2% difference in both hypervisors. Similarly, the ranges of

LLC hit rate for static partitioned hypervisor are from 77.22% to 91.34%. There is almost 5% difference in the dynamic partitioned and static partitioned hypervisors which validate the result based on 95% confidence interval. However, in Chapter 5, we have shown that this difference is significant. Moreover, the HBP-DCP based hypervisor has the ability to prevent cross-VM cache-based SC attacks.

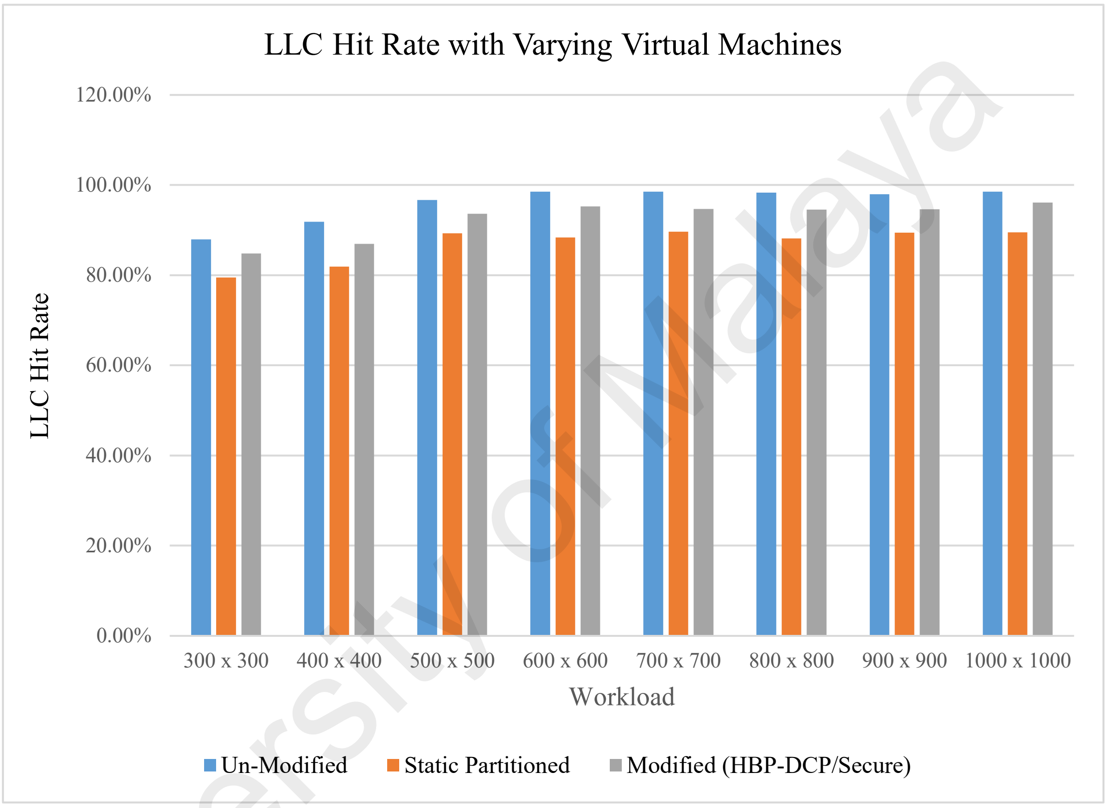


Figure 6.21: Average LLC Memory Hit Rate with Varying VMs

Figure 6.22 shows the LLC memory miss rate in both unmodified and HBP-DCP based hypervisors. In the figure, the y-axis shows the miss rate and the x-axis shows the workload in term of varying granularity level. Unlike to the hit rate, the miss rate for the unmodified hypervisor is less than as compared to the modified (HBP-DCP based) hypervisor. Moreover, the miss rate is decreasing with increasing matrix granularity. Because the miss rate is calculated in Chapter 5 by (1-hit rate) formula. Therefore, miss rate depends on the hit rate. Since the hit rate for the unmodified hypervisor is greater than HBP-DCP, therefore, miss rate will be lower than HBP-DCP. The ranges of LLC miss rate for unmodified hypervisor are from 1.44% to 12.03% and for modified (HBP-DCP based) hypervisor are from 3.86% to 15.16%. There is almost 3% difference in both unmodified and modified hypervisors. However, in the previous Chapter 5, we have shown by T-value and P-value that this difference is significant. Moreover, the HBP-DCP based hypervisor has the ability to prevent cross-VM cache-based SC attacks.

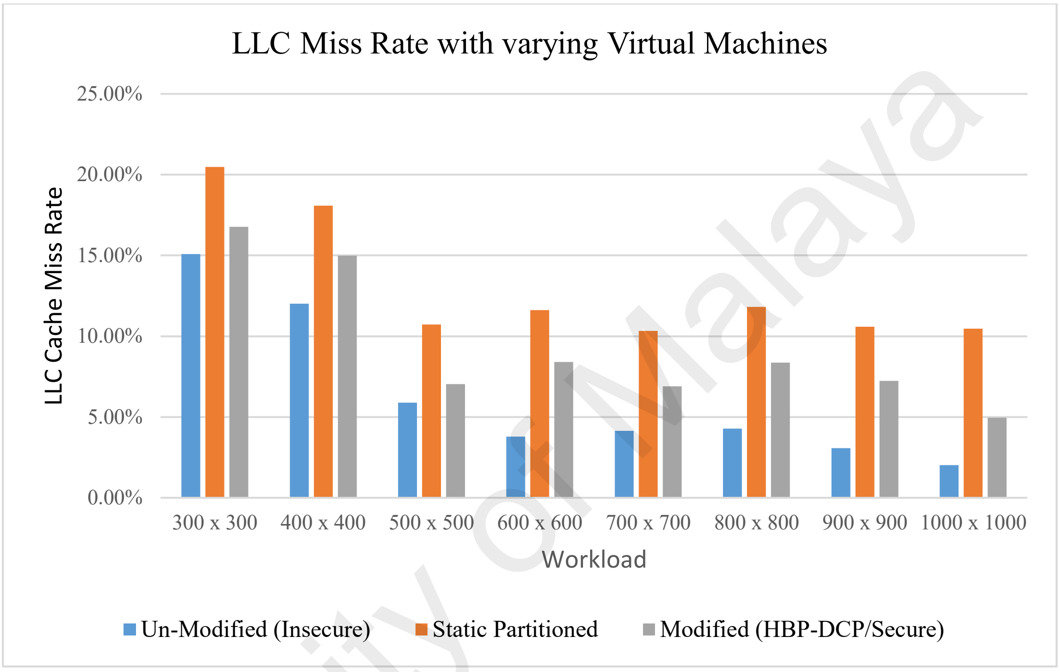


Figure 6.22: Average LLC Memory Miss Rate with Varying VMs

Figure 6.23 shows the comparison of average LLC memory access time in both unmodified and modified (HBP-DCP) hypervisors with varying VMs and eight different granularity level. The timing benchmark is our own design program to observe the memory access rate in term of the cache hit and miss rate. For cache access time, we have first calculated the LLC references, LLC miss and hit rate by using cachegrind benchmark. Then we have calculated the LLC memory access time by using our own designed program. Since the LLC access time is calculated from the cache miss and cache hit rate as shown in Eq. 5.7 and 5.8 and the hit rate is increasing with increasing workload, however, the miss rate is decreasing with increasing workload. Therefore the LLC access time is decreasing with increasing workload in term of matrix granularity. The Figure shows that the LLC access time is decreasing by increasing workload in term of matrix multiplication granularity. The ranges of LLC access time for unmodified hypervisor are from 11.3 to 20.82 and for modified (HBP-DCP based) hypervisor are from 13.47 to 23.64. The average difference in both hypervisors is almost 2. Previously in Chapter 5, we have proved the significance in the difference by P-value and T-value. Moreover modified (HBP-DCP based) hypervisor has the ability to prevent cross-VM cache-based SC attacks.

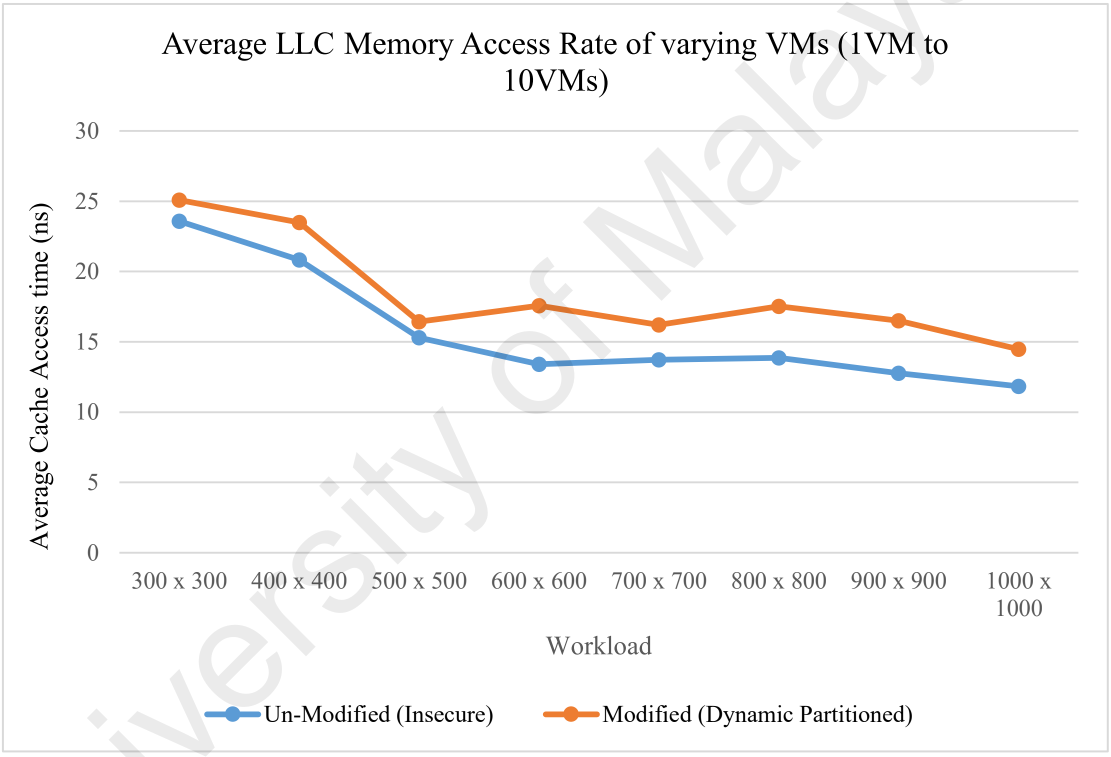


Figure 6.23: Comparison of Average LLC Memory Access Time in both Unmodified and HBP-DCP (Dynamic Partitioned) Hypervisors

Figure 6.24 shows the comparison of average LLC memory access time in both static partitioned and our HBP-DCP (Dynamic Partitioned) hypervisors with varying VMs and eight different granularity level. As shown in the figure the memory access time for our dynamic partitioned hypervisor as low as compared to the static partitioned hypervisor. Since the cache references is low and cache miss is high in the static partitioned hypervisor, therefore, the LLC memory access time will be high. The average access time for static partitioned is 21.71 and for dynamic partitioned is 18.29. The cache access time of our HBP-DCP based hypervisor is improved by 17.1% as the cache access time will be high for the high miss rate. The cache access time is calculated based on the total access rate and miss rate. Since the total cache access rate in static partitioned hypervisor is less than and the miss rate is greater than our dynamic partitioned (HBP-DCP) hypervisor. If the miss rate is high the cache access time will be high. Therefore, the average cache access time of our dynamic partitioned hypervisor (HBP-DCP) is less than static partitioned hypervisor. The T-value and P-value prove the significant difference between both results as the T-value is less than 2.2 and P-value is less than .05.

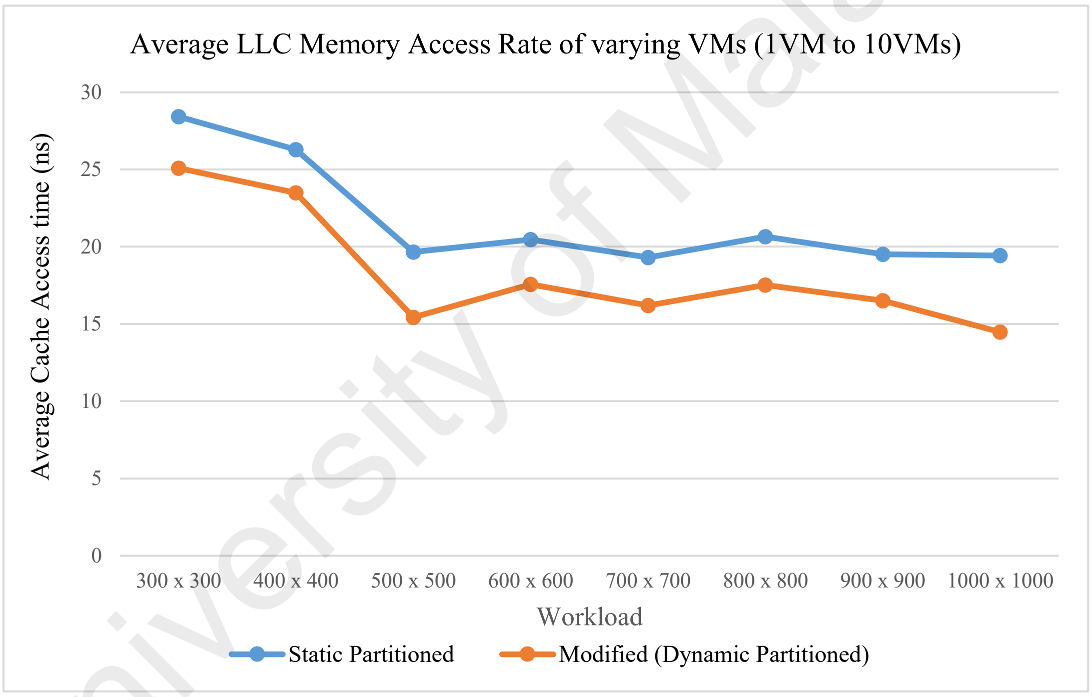


Figure 6.24: Comparison of Average LLC Memory Access Time in Static Partitioned and HBP-DCP (Dynamic Partitioned) Hypervisors

6.2 Conclusion

In this chapter, the experimental results of modified hypervisor based on HBP-DCP prevention mechanism is discussed to prove the efficiency of the proposed prevention mechanism on the basis of load testing, cache utilization, and cache access time and prevention of attacks. The experiments for load testing were investigated based on a number of requests per second and the average response time per request. Since our HBP-DCP prevention is based on the cache partition, therefore, the experiments were investigate based on the cache utilization and cache access time. The cache utilization is based on that how much bandwidth of cache would be utilized during writing and reading. In addition, the experiments are carried out in both unmodified (default/insecure) and modified (HBP-DCP based/secure) hypervisor.

In addition, we validate the system against cross-VM attacks while demonstrating that they can be prevented without client-side or hardware modifications. First, we validated our results by conducting the cross-VM cache-based SC attacks in unmodified and modified hypervisors. For this, we created two VMs: Victim and Attacker VMs. Then analyzing the performance by sending the 16-bit stream from attacker VM to Victim VM and check whether both hypervisors can prevent the attacks or not. Using the code base of an open source hypervisor, Xen (Project 2016), we have conducted our solution based on dynamic cache partition demonstrate how to inhibit cache-based side-channels from occurring within a cloud server. Our HBP-DCP prevention mechanism prevents communication along a shared cache by partitioning the cache dynamically into multiple segments using a technique known as cache coloring.

Then we analyzed the load testing, cache utilization, and cache access time in both unmodified (insecure), static partitioned, and modified (dynamic partitioned/HBP-DCP-based/secure) hypervisors. We observed that load testing showed an average 3189 in term of average number of request per request per second and 18.42 in response time in the unmodified hypervisor. While in modified (dynamic partitioned), the average number of request per second is 3157.88 and average response time is 18.28. As compared to the unmodified hypervisor the computing load in term of request per second in the modified hypervisor is increased by 1.008% and the response time is decreased by .07%. This is acceptable difference since we know security always comes with some overhead and the modified hypervisor has the ability to prevent cross-VM cache based SC attacks. In contrast to this, the computing load in static partitioned hypervisor in term of number of request per second is 1977.28 and average response time per request is 19.33. On the other hand, in modified (dynamic partitioned) hypervisor, the number of request per second is 3157.88 and average response time is 18.28. The computing load is improved in the dynamic partitioned hypervisor as compared to the static partitioned hypervisor. Because the average number of request per second in the dynamic partitioned hypervisor is increased by 45.98% and the average response time per request is decreased by 5.58%. Similarly, as compared to the static partitioned hypervisor the average bandwidth of cache read/modify/write is improved by 43.32% in our HBP-DCP based hypervisor. Consequently improves the cache utilization that each VM has access to by increasing cache read/modify/write, cache read, and cache write bandwidth in combine by 53.5%. Moreover, the cache access time is improved by 15.53%, as a result substantially decrease the overhead as significant by 20%. However, the modified hypervisor based on our proposed HBP-DCP prevention mechanism has the ability to prevent cross-VM cache based SC attacks. We then compare this solution to the current state of the art. In our comparison, we find that the dynamic partitioned hypervisor is more secure against side-channels regardless of the number of partitions we assign.

CHAPTER 7: CONCLUSION

This chapter presents the overall conclusions of this thesis and emphasizes the qualitative feature of the HBP-DCP mechanism. The conclusive analysis is carried out by considering on the aim and objectives set of research in the first chapter of the thesis. We identified the future research work and research contribution is also highlighted.

The rest of this chapter is also organized is as follows. In Section 7.1, the aim and research objectives of this study listed in Chapter 1 are reexamined. Section 7.2 describes the contribution of this research work. The significance of this work among existing prevention mechanism in CC is described in Section 7.3. Section 7.4 elaborates the scope and limitation of this research work and the future research direction are highlighted for further enhancement.

7.1 Research Objectives

This research work aimed to prevent the cross-VM cache-based SC attacks while maintaining the performance to solve the problem of static cache partitioning as a prevention mechanism. We described four research objectives in section 1.4. We investigate that how we could attain the research aim by completing the following research objectives.

Objective 1: To study the cache-based SC attacks in the non-virtualized and virtualized environment from the perspective of conducting and preventing these attacks to gain insights into performance limitations of current state-of-the-art prevention solutions.

The first objective was to investigate critically analyze the current state-of-the-art cross VM cache-based SC attacks and their prevention mechanisms such that insight is gained leading to their prevention and performance limitations. This research objective was conducted by a thorough review of the most credible work published in articles collected from online scholarly digital libraries, such as IEEE, ACM, Elsevier, and Web of Science using the University Malaya access portal. In order to ensure thorough browsing of the recent literature in the journals and conferences about cross VM cache-based SC attacks in CC, techniques for conducting SC attacks in OS, single VM and across VM, and prevention mechanisms for these attacks are visited. We organized the recent work, devised proposed taxonomy, and provided a qualitative comparison for cross-VM cache-based SC attacks, and prevention mechanisms for these attacks.

The main purpose of this thorough study was to analyze and synthesize the recent work in order to identify the research problems and challenges in the prevention mechanism for cross-VM cache-based SC attacks. We found that current prevention mechanism based on static cache partition is unable to handle attacks prevention efficiently. Therefore, a dynamic cache partition is required to prevent the cross-VM cache-based SC attacks in the CC.

Objective 2: To investigate the identified problem by conducting the cache-based SC attacks in the real environment and applying the existing prevention mechanism based on the static cache partition and unveiling the impact of existing prevention mechanism on the cache utilization as well as on the cloud model.

The second objective of this research study was to investigate and analyze the overhead in the existing prevention mechanisms in CC. Prevention of SC attacks can be applied by using hardware and software. We investigated the aforementioned SC prevention mechanism with the perspective of hypervisor-based (software) prevention mechanism for the cross-VM cache-based SC attacks. The investigation revealed that hardware based solution is costly, as they need to change the underlying hardware and is unable to provide the pro-active prevention. Furthermore, it does not comply with the cloud model as they need to change the client software and the underlying hardware. On the other hand, the software-based mechanisms provide security to the encryption algorithms rather than the overall information leakage across VMs and is comply with the cloud model. We further examined the cross-VM execution of cache-based SC attacks. We found that static cache partition, an existing software solution for prevention of cross-VM cache-based attacks degrade the performance in term of bearable load, cache utilization, and cache access time and consequently generated overhead.

Objective 3: To propose a prevention mechanism based on the dynamic cache partition for the prevention of cache-based SC attacks across VMs that leads to an efficient cache utilization among various VMs.

The third objective of this research study was to design a hypervisor-based prevention mechanism using dynamic cache partition (HBP-DCP) for the prevention of cross-VM cache-based SC attacks. The HBP-DCP is a hypervisor-based (e.g., software-based) mechanism complies with the cloud model which indicates that it does not need the changing in client software or the underlying hardware. For the cache monitoring, we devised a cache monitoring algorithm based upon the VM creation in the Xen Scheduler. This algorithm analyzes and reports the current state of the cache to the cache partitioner algorithm. The cache partitioner algorithm then re-partition the cache according to the number and requirement of VM based on the cache coloring approach.

Objective 4: To evaluate and validate the performance of our dynamic cache based prevention mechanism considering three metrics namely: computing load, cache utilization, and memory access rate and compare it with the state-of-the-art prevention mechanisms.

The fourth objective of this study was attained by evaluating the proposed mechanism via benchmarking experiment by creating 10 VMs on a desktop computer having all level cache (e.g., L1, L2, L3). We performed the performance experiments for all parameters and observe the results for 30 workload execution under the identical condition and every workload is repeated for 30 times for the sake of reliability for each of 10 VMs in both modified and unmodified hypervisor to experiments. Our performance results unveil that utilizing our proposed prevention mechanism prevent the cross-VM cache-based SC and also improve the cache utilization to 53.5% , load by 45% and cache access time by 15.53% while generating less than 5% overhead as compared to the static partitioned prevention mechanism.

We develop a statistical model in order to validate the performance results of our proposed HBP-DCP prevention mechanism. Regression analysis is used for the purpose to derive the accurate statistical model of our four performance evaluation parameters namely: load testing, cache utilization, and memory access rate. We validate our performance results of the HBP-DCP mechanism by using split-sample validation approach. We compared the findings of benchmarking to the statistical modeling to validate our proposed prevention mechanism. Validation results confirm that leveraging our proposed prevention mechanism can prevent cross-VM cache-based SC attacks without affecting the performance of the system and improve the load, cache utilization, and cache access time.

7.2 Contribution

In this Section, we have highlighted the contribution of this research work. We presented the contribution in term of the scholarly articles in list of publications and presented papers at the end of thesis. This research work produced several contributions to the body of knowledge in following aspects.

- **Taxonomy of Cross-VM Cache-based Side Channel Attacks:** We produced taxonomies from the existing literature for the cache-based SC attacks and prevention mechanisms. We comprehensive reviewed the Cache-based SC attacks from the cross-VM point of view and prevention mechanism by critical analyzing of the selected state-of-the-art research work extracted from scholarly articles such as ACM, IEEE, and Elsevier. Our comprehensive studied literature is presented in Chapter 2 and published in (Anwar, Inayat et al. 2017) led to the identification of our research problem.

- **Cache Monitoring Algorithm:** We devised a cache monitoring algorithm for the page allocator system of hypervisor. The cache monitoring algorithm examines the cache status upon the new request from the admission control for new VM creation. In addition, this algorithm is efficient in assigning the different partition of the cache to each VMs according to the VMs requirement.
- **Hypervisor-based Prevention Mechanism (HBP-DCP):** We devised a hypervisor-based prevention mechanism (HBP-DCP) for the prevention of cross-VM cache-based SC attacks. HBP-DCP mechanism is based on the dynamic cache partition for each VMs. The cache monitoring was integrated with the cache partitioning (page allocating) algorithms in the existing page allocator of hypervisor to enable the hypervisor to partition the cache dynamically according to the new VM requirement when new VM is created.
- **Performance Evaluation and Validation:** The analytical evaluation results of the system are generated through benchmarking and statistical modeling. Performance evaluation using benchmark analysis is performed on the modified (Dynamic partitioned/secure) and unmodified (Default/insecure) hypervisor. We developed a statistical model of the benchmarking parameters of HBP-DCP mechanism for the prevention of cross-VM cache-based SC attacks. The statistical model is generated via observation-based modeling approach in which dataset of independently replicated data is generated to train the regression model. The model is validated using split-sample approach is used to validate the performance of our proposed prevention mechanism. The process and result of performance evaluation and validation are presented in Chapters 5 and 6

respectively. Statistical and schematic analysis of the results unveiled the feasibility, functionality, lightweight nature of our proposed prevention mechanism and advocate that the objectives and aim of this study are fulfilled and is realized.

7.3 Significance of the work

Several significant features that are considered during design and development of HBP-DCP prevention mechanism could distinguish it from the existing prevention mechanism for cache-based SC attacks are briefly presented as follows:

First, HBP-DCP prevention mechanism complies with the cloud model. In particular, unlike hardware mechanism, HBP-DCP is hypervisor based (e.g., software based) prevention mechanism which does not need changes in any client software or the underlying hardware. Therefore, it can be embedded into the hypervisor and in the cloud model, because, it obey the cloud rules.

Second, this attack is based on the cache (the most interactive device). Since our prevention mechanism is based on the dynamic cache partition, therefore, it is generalizable in the sense that it can prevent all types of SC attacks which is based on the cache and in all type of hypervisor (e.g., XEN and VMWare) in which VMs can be created.

Third, our HBP-DCP prevention mechanism can be ported to any type of the supported software (hypervisor) and computing infrastructure. Since our HBP-CP prevention mechanism is hypervisor-based means we have implemented by using the source code of an open source hypervisor. Therefore, HBP-DCP can be installed almost on every type of computing infrastructure and it is applicable to the commodity OS.

Fourth, HBP-DCP is based on the dynamic partition of the cache. Therefore the overall performance has improved by increasing the cache utilization for each VMs because each VM is only giving as much more cache memory as they are requested at runtime. For instance, if there is 2MB L3 cache, and 1 VM is running the whole 2MB would be assigned to 1VM, consequently improve the overall performance in term of load, cache utilization, and memory access rate.

Fifth, HBP-DCP is a preventive mechanism rather than reactive. Since we cannot examine when SC attacks might occur, we simply ensure that the two VMs would not be able to access the same cache lines for the purpose to create SC attacks. Preventive means early prevention before occurring of the attacks while reactive means prevents attacks after occurring. Because once the attack occurs, it will harm the system even in a minute, therefore, early prevention of attack is more beneficial than post prevention.

7.4 Limitation and Future Work

The HBP-DCP prevention mechanism prevent cross-VM cache-based SC attacks with a minimum cache access rate and by improving computing load, cache utilization, and memory access rate. Our HBP-DCP (dynamic partitioned) mechanism can be entirely implemented within the hypervisor and do not interfere to the cloud model (does not need to change the client side's software or the underlying hardware). The HBP-DCP prevention mechanism can prevent any type of attacks in which cache is involved and therefore it is generalizable to all types of the hypervisor which is used for VM creation.

However, HBP-DCP is always activated upon the VM creation, and assign the specific color page of the cache memory that matches the color of the requested VM, the limitation of this prevention mechanism is it is unable to detect the cache memory requirement of each VMs upon the creation time. For instance, if two VMs are requested for cache then it is unable to detect that how much amount of cache is required to VM1 and how much to VM2. On the other hand, the decision that how many pages should be migrated and which one page among all pages should be migrated first is very difficult.

In our future work, we will consider cache management policy. We will focus on that when VM is created then we should be able to predict all the cache requirements of that specific VM. Furthermore, the cloud computing environment is also vulnerable to other SC attacks likewise the cache-based SC attacks. It arises difficulty for the cloud provider because SC attacks based on a specific medium often require their own unique solutions. Therefore, each channel will required further work to develop a solution customized to its specific vulnerabilities.

REFERENCES

Aciiçmez, O. (2007). Yet another microarchitectural attack:: exploiting I-cache. Proceedings of the 2007 ACM workshop on Computer security architecture, ACM.

Acıiçmez, O., B. B. Brumley and P. Grabher (2010). New results on instruction cache attacks. Cryptographic Hardware and Embedded Systems, CHES 2010, Springer**:** 110-124.

Aciiçmez, O., Ç. K. Koç and J.-P. Seifert (2007). On the power of simple branch prediction analysis. Proceedings of the 2nd ACM symposium on Information, computer and communications security, ACM.

Acıiçmez, O., W. Schindler and Ç. K. Koç (2007). Cache based remote timing attack on the AES. Topics in Cryptology–CT-RSA 2007, Springer**:** 271-286.

Aciicmez, O. and J.-P. Seifert (2007). Cheap hardware parallelism implies cheap security. Fault Diagnosis and Tolerance in Cryptography, 2007. FDTC 2007. Workshop on, IEEE.

Anderson, R., M. Bond, J. Clulow and S. Skorobogatov (2006). "Cryptographic processors-a survey." Proceedings of the IEEE **94**(2): 357-369.

Anwar, S., Z. Inayat, M. F. Zolkipli, J. M. Zain, A. Gani, N. B. Anuar, M. K. Khan and V. Chang (2017). "Cross-VM Cache-based Side Channel Attacks and Proposed Prevention Mechanisms: A survey." Journal of Network and Computer Applications.

ARM, A. (2012). "Architecture Reference Manual. ARMv7-A and ARMv7-R edition." ARM DDI C **406**.

Aumüller, C., P. Bier, W. Fischer, P. Hofreiter and J.-P. Seifert (2002). Fault attacks on RSA with CRT: Concrete results and practical countermeasures. International Workshop on Cryptographic Hardware and Embedded Systems, Springer.

Aviram, A., S. Hu, B. Ford and R. Gummadi (2010). Determinating timing channels in compute clouds. Proceedings of the 2010 ACM workshop on Cloud computing security workshop, ACM.

Barham, P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield (2003). Xen and the art of virtualization. ACM SIGOPS operating systems review, ACM.

Bernstein, D. J. (2004). "Cache-timing attacks on AES, URL: http://cr.yp.to/papers.html#cachetiming.".

Bernstein, D. J. (2005). palms.ee.princeton.edu, Cache-timing attacks on AES, Technical report.

Bertoni, G., V. Zaccaria, L. Breveglieri, M. Monchiero and G. Palermo (2005). AES power attack based on induced cache miss and countermeasure. Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on, IEEE.

Biham, E. and A. Shamir (1997). Differential fault analysis of secret key cryptosystems. Annual International Cryptology Conference, Springer.

Bonneau, J. and I. Mironov (2006). Cache-collision timing attacks against AES. Cryptographic Hardware and Embedded Systems-CHES 2006, Springer**:** 201-215.

Brickell, E., G. Graunke, M. Neve and J.-P. Seifert (2006). "Software mitigations to hedge AES against cache-based software side channel vulnerabilities." IACR Cryptology ePrint Archive **2006**: 52.

Brumley, B. B. and R. M. Hakala (2009). Cache-timing template attacks. International Conference on the Theory and Application of Cryptology and Information Security, Asiacript, Springer 2009.

Brumley, D. and D. Boneh (2005). "Remote timing attacks are practical." Computer Networks **48**(5): 701-716.

Cert. (2017.). "http://www.cert.org/news/article.cfm?assetid=493750&article=039&year=2017, Cert Statistics. 2017."

Chang, V. and M. Ramachandran (2016). "Towards achieving data security with the cloud computing adoption framework." IEEE Transactions on Services Computing **9**(1): 138-151.

Coppens, B., I. Verbauwhede, K. De Bosschere and B. De Sutter (2009). Practical mitigations for timing-based side-channel attacks on modern x86 processors. 2009 30th IEEE Symposium on Security and Privacy, IEEE.

DPA Countermeasures, D. "https://www.rambus.com/inventions-dpa-countermeasures/, Inventions Security, Oct, 22, 2016."

Crane, S., A. Homescu, S. Brunthaler, P. Larsen and M. Franz (2015). Thwarting Cache Side-Channel Attacks Through Dynamic Software Diversity. NDSS.'15, 8-11 February 2015, San Diego, CA, USA. Copyright 2015 ...

DFA (2016). "http://www.businessinsurance.com/article/99999999/NEWS070101/110909913/digital-forensics-association-data-breach-report."

Domnitser, L., A. Jaleel, J. Loew, N. Abu-Ghazaleh and D. Ponomarev (2012). "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks." ACM Transactions on Architecture and Code Optimization (TACO) **8**(4): 35.

Fisk, G., M. Fisk, C. Papadopoulos and J. Neil (2002). Eliminating steganography in Internet traffic with active wardens. Information Hiding, Springer.

Foundation., A. S. (2013). "Apache http server benchmarking tool,."

Gandolfi, K., C. Mourtel and F. Olivier (2001). Electromagnetic analysis: Concrete results. Cryptographic Hardware and Embedded Systems—CHES 2001, Springer.

Godfrey, M. and M. Zulkernine (2014). "Preventing cache-based side-channel attacks in a cloud environment." Cloud Computing, IEEE Transactions on **2**(4): 395-408.

Godfrey, M. M. and M. Zulkernine (2014). "Preventing cache-based side-channel attacks in a cloud environment." IEEE Transactions on Cloud Computing **2**(4): 395-408.

Gullasch, D., E. Bangerter and S. Krenn (2011). Cache games--bringing access-based cache attacks on AES to practice. Security and Privacy (SP), 2011 IEEE Symposium on, IEEE.

Handy, J. (1998). The cache memory book, Morgan Kaufmann.

Harnik, D., B. Pinkas and A. Shulman-Peleg (2010). "Side channels in cloud services: Deduplication in cloud storage." IEEE Security & Privacy **8**(6): 40-47.

IBM (2012). "https://www.ibm.com/developerworks/topics/master%20the%20mainframe%202012/."

Inci, M. S., B. Gulmezoglu, G. Irazoqui, T. Eisenbarth and B. Sunar (2015). Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud, Cryptology ePrint Archive, Report 2015/898, 2015. http://eprint. iacr. org.

Intel, R. (2007). "and IA-32 architectures optimization reference manual." Intel Corporation, May.

Intervals, C. (2004). " Confidence Intervals and Hypothesis Testing, http://statweb.stanford.edu/~susan/courses/s141/TTestLecture.pdf."

Irazoqui, G., T. Eisenbarth and B. Sunar (2015). S $ A: A Shared Cache Attack That Works across Cores and Defies VM Sandboxing--and Its Application to AES. 2015 IEEE Symposium on Security and Privacy, IEEE.

Irazoqui G, I. M., Eisenbarth T, Sunar B. (2014). "Fine grain Cross-VM Attacks on Xen and VMware are possible! IACR Cryptology ePrint Archive. 2014a;2014:248." IACR Cryptology ePrint Archive. 2014a;2014:248. **2014**: 248.

Irazoqui, G., M. S. Inci, T. Eisenbarth and B. Sunar (2014). "Fine grain Cross-VM Attacks on Xen and VMware are possible!" IACR Cryptology ePrint Archive **2014**: 248.

Irazoqui, G., M. S. Inci, T. Eisenbarth and B. Sunar (2014). Wait a minute! A fast, Cross-VM attack on AES. Research in Attacks, Intrusions and Defenses, Springer**:** 299-319.

Jin, X., H. Chen, X. Wang, Z. Wang, X. Wen, Y. Luo and X. Li (2009). A simple cache partitioning approach in a virtualized environment. Parallel and Distributed Processing with Applications, 2009 IEEE International Symposium on, IEEE.

Kim, S., D. Chandra and Y. Solihin (2004). Fair cache sharing and partitioning in a chip multiprocessor architecture. Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques, IEEE Computer Society.

Kocher, P., J. Jaffe and B. Jun (1999). Differential power analysis. Advances in Cryptology—CRYPTO'99, Springer.

Kong, J., O. Aciicmez, J.-P. Seifert and H. Zhou (2013). "Architecting against software cache-based side-channel attacks." IEEE Transactions on Computers **62**(7): 1276-1288.

Kong, J., O. Aciiçmez, J.-P. Seifert and H. Zhou (2009). Hardware-software integrated approaches to defend against software cache-based side channel attacks. 2009 IEEE 15th International Symposium on High Performance Computer Architecture, IEEE.

Lauradoux, C. (2005). "Collision attacks on processors with cache and countermeasures." WEWoRC **5**: 76-85.

Li, Y., K. Sakiyama, S. Gomisawa, T. Fukunaga, J. Takahashi and K. Ohta (2010). Fault sensitivity analysis. International Workshop on Cryptographic Hardware and Embedded Systems, Springer.

Liu, F., Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser and R. B. Lee (2016). Catalyst: Defeating last-level cache side channel attacks in cloud computing. 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE.

Liu, F., Y. Yarom, Q. Ge, G. Heiser and R. B. Lee (2015). Last-level cache side-channel attacks are practical. IEEE Symposium on Security and Privacy.

Mangard, S. (2002). A simple power-analysis (SPA) attack on implementations of the AES key expansion. International Conference on Information Security and Cryptology, Springer.

Miłós, G., D. G. Murray, S. Hand and M. A. Fetterman (2009). Satori: Enlightened page sharing. Proceedings of the 2009 conference on USENIX Annual technical conference.

Mishra, P., E. S. Pilli, V. Varadharajan and U. Tupakula (2017). "Intrusion detection techniques in cloud environment: A survey." Journal of Network and Computer Applications **77**: 18-47.

Neve, M. and J.-P. Seifert (2006). Advances on access-driven cache attacks on AES. Selected Areas in Cryptography, Springer.

NEWS, T. (2015). "http://www.thestar.com/business/tech_news/2015/09/21/apple-store-hack-targets-iphone-ipad-apps.html , Accessed on 3 dec ".

Osvik, D. A., A. Shamir and E. Tromer (2006). Cache attacks and countermeasures: the case of AES. Topics in Cryptology–CT-RSA 2006, Springer**:** 1-20.

Oswald, E., S. Mangard, N. Pramstaller and V. Rijmen (2005). A side-channel analysis resistant description of the AES S-box. International Workshop on Fast Software Encryption, Springer.

Owens, R. and W. Wang (2011). Non-interactive OS fingerprinting through memory de-duplication technique in virtual machines. Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International, IEEE.

Page, D. (2003). "Defending against cache-based side-channel attacks." Information Security Technical Report **8**(1): 30-44.

Page, D. (2005). "Partitioned Cache Architecture as a Side-Channel Defence Mechanism." IACR Cryptology ePrint Archive **2005**: 280.

Percival, C. (2005). Cache missing for fun and profit, In BSD Con.

performance, V. L. P. (Feb 2016). "http://www.vmware.com/techpapers/2008/large-page-performance-1039.html."

Portal, T. S. (2016). "http://www.statista.com/statistics/321215/global-consumer-cloud-computing-users/, Accessed on July ".

Project, X. D. (2016). "http://wiki.xenproject.org/wiki/Xen_Project_Beginners_Guide."

Quisquater, J.-J. and D. Samyde (2001). Electromagnetic analysis (ema): Measures and counter-measures for smart cards. Smart Card Programming and Security, Springer**:** 200-210.

Qureshi, M. K. and Y. N. Patt (2006). Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on, IEEE.

Ristenpart, T., E. Tromer, H. Shacham and S. Savage (2009). Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. Proceedings of the 16th ACM conference on Computer and communications security, ACM.

Rixner, S., W. J. Dally, U. J. Kapasi, P. Mattson and J. D. Owens (2000). Memory access scheduling. ACM SIGARCH Computer Architecture News, ACM.

Russell, D. (2010). "Data deduplication will be even bigger in 2010." Gartner, Feb.

Security, C. (2010). " “Top Threats to Cloud Computing”, Cloud Security Alliance,. http://www.cloudsecurityalliance.org/csaguide.pdf, V. 1.0 ".

Shi, J., X. Song, H. Chen and B. Zang (2011). Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on, IEEE.

Singh, A. and K. Chatterjee (2017). "Cloud security issues and challenges: A survey." Journal of Network and Computer Applications **79**: 88-115.

Smith, S. W. (2003). "Fairy dust, secrets, and the real world [computer security]." Security & Privacy, IEEE **1**(1): 89-93.

Soares, L., D. Tam and M. Stumm (2008). Reducing the harmful effects of last-level cache polluters with an OS-level, software-only pollute buffer. Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society.

Stanek, J., A. Sorniotti, E. Androulaki and L. Kencl (2014). A secure data deduplication scheme for cloud storage. International Conference on Financial Cryptography and Data Security, Springer.

Suzaki, K., K. Iijima, T. Yagi and C. Artho (2011). Memory deduplication as a threat to the guest OS. Proceedings of the Fourth European Workshop on System Security, ACM.

Suzaki, K., K. Iijima, T. Yagi and C. Artho (2011). "Software side channel attack on memory deduplication." SOSP POSTER.

Tam, D., R. Azimi, L. Soares and M. Stumm (2007). Managing shared L2 caches on multicore systems in software. Workshop on the Interaction between Operating Systems and Computer Architecture, Citeseer.

Taylor, G., P. Davies and M. Farmwald (1990). The TLB slice-a low-cost high-speed address translation mechanism. Computer Architecture, 1990. Proceedings., 17th Annual International Symposium on, IEEE.

Technology., I. V. (2016). "http://www.intel.com/, Accessed on April 2016."

Tromer, E., D. A. Osvik and A. Shamir (2010). "Efficient cache attacks on AES, and countermeasures." Journal of Cryptology **23**(1): 37-71.

Varadarajan, V., T. Ristenpart and M. M. Swift (2014). Scheduler-based Defenses against Cross-VM Side-channels. Usenix Security.

Wang, Z., C. Cao, N. Yang and V. Chang (2016). "ABE with improved auxiliary input for big data security." Journal of Computer and System Sciences.

Wang, Z. and R. B. Lee (2006). Covert and side channels due to processor architecture. null, IEEE.

Wang, Z. and R. B. Lee (2007). New cache designs for thwarting software cache-based side channel attacks. ACM SIGARCH Computer Architecture News, ACM.

Wang, Z. and R. B. Lee (2008). A novel cache architecture with enhanced performance and security. 2008 41st IEEE/ACM International Symposium on Microarchitecture, IEEE.

Weisberg, P. and Y. Wiseman (2009). Using 4kb page size for virtual memory is obsolete. Information Reuse & Integration, 2009. IRI'09. IEEE International Conference on, IEEE.

Weiß, M., B. Heinz and F. Stumpf (2012). A cache timing attack on AES in virtualization environments. Financial Cryptography and Data Security, Springer**:** 314-328.

Wu, L., S. K. Garg and R. Buyya (2012). "SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments." Journal of Computer and System Sciences **78**(5): 1280-1299.

Wu, Z., Z. Xu and H. Wang (2012). Whispers in the hyper-space: high-speed covert channel attacks in the cloud. Presented as part of the 21st USENIX Security Symposium (USENIX Security 12).

Xiao, J., Z. Xu, H. Huang and H. Wang (2013). Security implications of memory deduplication in a virtualized environment. 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE.

Yarom, Y. and K. Falkner (2014). Flush+ reload: a high resolution, low noise, L3 cache side-channel attack. 23rd USENIX Security Symposium (USENIX Security 14).

Yu, S. and S. Guo (2016). Big Data Concepts, Theories, and Applications, Springer. SBN 978-3-319-27763-9

Zander, S., G. Armitage and P. Branch (2007). "A survey of covert channels and countermeasures in computer network protocols." Communications Surveys & Tutorials, IEEE **9**(3): 44-57.

Zhang, Q., L. Cheng and R. Boutaba (2010). "Cloud computing: state-of-the-art and research challenges." Journal of internet services and applications **1**(1): 7-18.

Zhang, X., S. Dwarkadas and K. Shen (2009). Towards practical page coloring-based multicore cache management. Proceedings of the 4th ACM European conference on Computer systems, ACM.

Zhang, Y., A. Juels, A. Oprea and M. K. Reiter (2011). Homealone: Co-residency detection in the cloud via side-channel analysis. 2011 IEEE Symposium on Security and Privacy, IEEE.

Zhang, Y., A. Juels, M. K. Reiter and T. Ristenpart (2012). Cross-VM side channels and their use to extract private keys. Proceedings of the 2012 ACM conference on Computer and communications security, ACM.

Zhang, Y. and M. K. Reiter (2013). Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud. Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM.

Zhou, Z., M. K. Reiter and Y. Zhang (2016). "A software approach to defeating side channels in last-level caches." arXiv preprint arXiv:1603.05615.

LIST OF PUBLICATIONS AND PAPERS PRESENTED

Published ISI Journal Articles

1) **Zakira Inayat,** A Gani, NB Anuar, MK Khan, S Anwar, "Intrusion response systems: Foundations, design, and challenges", "Journal of Network and Computer Applications" 62, 53-74.

2) **Zakira Inayat,** A Gani, NB Anuar, S Anwar, MK Khan, "Cloud-Based Intrusion Detection and Response System: Open Research Issues, and Solutions", "Arabian Journal for Science and Engineering", 1-25.

3) S Anwar, **Zakira Inayat**, MF Zolkipli, JM Zain, A Gani, NB Anuar, MK Khan, "Cross-VM Cache-based Side Channel Attacks and Proposed Prevention Mechanisms: A survey" ."Journal of Network and Computer Applications".

4) N Khan, I Yaqoob, IAT Hashem, **Zakira Inayat,** WK Mahmoud Ali, M Alam, "Big data: survey, technologies, opportunities, and challenges", "The Scientific World Journal 2014".

5) S Anwar, J Mohamad Zain, MF Zolkipli, **Zakira Inayat,** S Khan, B Anthony, "From Intrusion Detection to an Intrusion Response System: Fundamentals, Requirements, and Future Directions", Algorithms 10 (2), 39.

6) Nawsher Khan, Noraziah Ahmad, Tutut Herawan, **Zakira Inayat, "**Cloud Computing: Locally Sub-Clouds instead of Globally One Cloud", International Journal of Cloud Applications and Computing, 2(3), 68-85, July-September 2012.

Accepted Conference Articles

1) Nawsher Khan, Ibrar Yaqoob, Ibrahim Abaker, **Zakira Inayat**, Abdullah Gani, et al. "Big Data: Survey, Technologies, Opportunities, and Challenges", June 2013, ABC.

2) Shahid Anwar, Jasni Mohamad Zain, Mohamad Fadli Zolkipli, **Zakira Inayat.** **“**A Review Paper on Botnet and Botnet Detection Techniques in Cloud Computing”, Sep 2014 ISCI 2014 – IEEE Symposium on Computers & Informatics.

3) Shahid Anwar, Jasni Mohamad Zain, **Zakira Inayat**, Mohamad Fadli Zolkipli, Julius Odili, “Response Option for Attacks Detected by Intrusion Detection System”, Aug 2015 The 4th International Conference on Software Engineering and Computer System.

4) Shahid Anwar, Jasni Mohamad Zain, **Zakira Inayat**, et al., “Static Approach Towards Mobile Botnet Detection”, 3rd International Conference on Electronic Design (ICED) Aug 2016

5) Nawsher Khan, A. Noraziah, Tutut Herawan, **Zakira Inayat**, “Cloud Computing: Architecture for Efficient Provision of Services”, Sep 2012, NBis 2012.

Submitted ISI Journal Articles

1) **Zakira Inayat,** A Gani, NB Anuar, MK Khan, S Anwar , Prevention of Cross VM Cache-based Side Channel Attacks using Dynamic cache Partitioning” Submitted in Tier-1 journal, June 15, 2017.

2) Shahid Anwar, Mohamad Fadli Zokipli, Jasni Mohamad Zain, **Zakira Inayat,** et al., “Android Botnets: A Serious Threat to Android Devices”, ISI Index Journal, Submitted Date: 6 Dec, 2016