SECURING THE DATA STORAGE AND PROCESSING IN CLOUD COMPUTING ENVIRONMENT

by

Rodney Owens

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2013

ABSTRACT

RODNEY OWENS. Securing the data storage and processing in Cloud Computing Environment. (Under the direction of DR. WEICHAO WANG)

Organizations increasingly utilize cloud computing architectures to reduce costs and energy consumption both in the data warehouse and on mobile devices by better utilizing the computing resources available. However, the security and privacy issues with publicly available cloud computing infrastructures have not been studied to a sufficient depth for organizations and individuals to be fully informed of the risks; neither are private nor public clouds prepared to properly secure their connections as middle-men between mobile devices which use encryption and external data providers which neglect to encrypt their data. Furthermore, cloud computing providers are not well informed of the risks associated with policy and techniques they could implement to mitigate those risks.

In this dissertation, we present a new layered understanding of public cloud computing. On the high level, we concentrate on the overall architecture and how information is processed and transmitted. The key idea is to secure information from outside attack and monitoring. We use techniques such as separating virtual machine roles, re-spawning virtual machines in high succession, and cryptography-based access control to achieve a high-level assurance of public cloud computing security and privacy. On the low level, we explore security and privacy issues on the memory management level. We present a mechanism for the prevention of automatic virtual machine memory guessing attacks.

ACKNOWLEDGMENTS

This dissertation would not have happened without the help, inspiration, and encouragement of many people along the way.

First and foremost, I want to thank my advisor, Prof. Weichao Wang. I am very fortunate to have the privilege of being his graduate student. Dr. Wang is readily available to provide guidance and direction in my research. His support goes beyond his role as my advisor and I will never forget the fun we've had along the way.

I am honored to have Professors Xintao Wu, Mohamed Shehab, Bill Chu, and Wei Zhao serve on my PhD Committee. Their scrutiny has helped shape this dissertation. I want to especially thank Bill Chu for handling the details necessary for my temporary teaching position in ITIS 3200, and Mohamed Shehab for his guidance in how to steer students towards an education in information security.

I could not have completed my studies at UNCC without the financial support of the Charlotte Information Systems Security Association, the NSF Partnerships for International Research and Education, and Graduate Assistance in Areas of National Need. These programs not only provided financial support, but they all have adjusted my perspectives on information security, teaching, and international research.

Last, but not least, I would like to thank my family and friends, most especially my parents, Roger and Millie, my wife, Anna, and my Savior, Jesus Christ, who's constant support gave me the drive to pursue my goals.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Motivation

With the nature of the modern economy, it is becoming increasingly important to minimize the costs of a computing infrastructure. Organizations are looking to do more with less, by making management easier with less man-power, while decreasing the total cost of ownership associated with service oriented computing resources. Due to these constraints, organizations are increasingly turning to cloud-computing resources. Since the early 2000's, Cloud computing is often a vague term that is gaining popularity. It is important to pin down what, precisely, is being discussed when the term 'cloud computing' is used, in order to effectively discuss the importance of this technological trend in today's world. Currently, the trend is to describe cloud computing as 'Everything as a Service' or 'EaaS.' Infrastructure as a service (IaaS), which is a more specific concept, offers the consumer a greater degree of control than other types of cloud services. The authors of [63] explain that "the consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls)." Software, computer storage, and even a desktop are transformed into services offered by providers such as Google and Amazon. [36] This explanation is adequate for the average consumer, but something more specific is needed to obtain a better understanding of these concepts. Cloud computing, according to [63], is the following:

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

This is a generally accepted definition of cloud computing, as evidenced by its mention in [68], which also considers it the "closest to capturing in a minimal number of words all of the essential ideas of cloud computing." The authors of [104] say that these listed attributes: ubiquitous, convenient, and on-demand are the driving forces behind the increasing popularity of cloud computing. The mobile cloud is the embodiment of these elements. According to [94] the mobile cloud is "mobile services and applications delivered from a centralized (and perhaps virtualized) data center to a mobile device such as a smart phone." In [94] they suppose that "the mobile cloud-stands to significantly increase the overall value of mobility, as well as radically alter the way people live, learn, work, and play." With so many mobile users poised to take advantage of all the services the mobile cloud can offer, the amount of growth is anticipated to be "unprecedented."

Businesses, particularly "small and medium sized enterprises (SMEs)," can benefit from what the 'cloud' has to offer. [68] As [74] states, "Cloud computing means someone else runs your computers and software while you use what they deliver and focus on delivering value." Value here meaning whatever product or service your customers seek from your business. The authors of [74] elaborate: “You can let your people concentrate on doing their jobs, instead of worrying about which new technology needs to be purchased.”

Cloud computing does more than allow your business to become more focused. The benefits are monetary, and easily calculable. In the cloud computing world, the business pays for the outcome, not the technology. [74] The authors of [74] describe it this way:

> You no longer need to own some of your computing equipment, which also means you don’t need staff to maintain it. An external provider can handle all this for you as a service, just like Starbucks manages the coffee-making equipment and the barristers. You get a reduced price through economies of scale.

Normally, a business would need to buy and maintain their own servers and networks. Cost of these operations include: electricity to run the machines, salaries to those knowledgeable enough to keep them functioning and secure, and replacing technologies as they rapidly change. When all the equipment is stored in a large warehouse, in a data center (think Google or Amazon), this reduces all of these costs. Storage as a service (STaaS) is a fundamental cloud service that takes advantage of these large data centers. The total cost for a data warehouse to store data is lower than for individual consumers. The authors of [68] conjecture that “another driver of cloud computing is an evolution from small, distributed, data-oriented computing centers to more cost-effective, very large scale commercial cloud services. This trend is likely to continue.” Cost-effectiveness is the key. Electricity, including heating and cooling, costs are lower. The cost of backup and general maintenance is also lower. The authors of [68] go on to say that “The last few years has seen rapid learning in warehouse-size data centers on the design of cooling systems, environment monitoring, and backup with the result that power utility efficiency is improving."

Another benefit of cloud computing is that businesses do not pay for what they do not need. The authors of [63] describe this as one of the characteristics of cloud computing, "rapid elasticity- capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time." New businesses normally would have to try to anticipate consumer demand (how quickly and to what extent it will increase) when purchasing equipment. They don't want newly purchased servers to become quickly overwhelmed by a surge in high traffic, leading to down time and consumer dissatisfaction. "Computing servers and desktop computers in a modern organization are often underutilized, since IT infrastructure is configured to handle theoretical demand peaks." [104] With cloud computing, this is not an issue. Most of the time cloud services are pay as you go (PAYG). When you no longer need a service, you stop paying for it. "The PAYG model's flexibility lets customers scale up or scale down the work they do with [providers]." [74]

Both [14] and [68] describe concerns about security and privacy as barriers to businesses embracing cloud computing. "A 'one-size-fits-all' approach to providing the latest state-of-the art in security, privacy and trust technologies is neither appropriate nor even possible," [68] points out. The authors of [14] observe these and other barriers, including: "unclear legal jurisdiction and data location issues, complex security and data protection regulations, uncertain trust in suppliers, and lack of guaranteed data access and portability between cloud system." The authors of [39] state the following basic security issue: "For a business, disclosing the personal information of customers or employees, or other business information to a cloud provider is often unrestricted by law because no privacy law or other law applies.” A related security issue involves encryption of data. Data that is sent over the Internet is often encrypted for the sake of privacy and security. The authors of [68] explain this problem:

> Encryption is not a complete solution because data needs to be decrypted in certain situations - so that computation can occur and the usual data management functions of indexing and sorting can be carried out. Thus although data in transit and data at rest are effectively encrypted, the need to decrypt, generally by the cloud service provider, can be a security concern.

The authors of [39] point out an inherent flaw in cloud services:

> A cloud provider will also acquire transactional and relationship information that may itself be revealing or commercially valuable. For example, the sharing of information by two companies may signal a merger is under consideration. In some instances, only the provider’s policy will limit use of that information.

To protect itself, a business should be intimately familiar with the terms of service set out by the cloud provider, and be aware of potential problems. “Ultimately, however, the uptake of cloud services is dependent upon providers and their clients trusting each other” [68] points out. The authors of [14] describes “uncertainties about the way legal and security issues are managed in the cloud environment are strongly correlated with uncertainties about the relationship with and the trustworthiness of cloud providers.” The authors of

[83] observed that “Google’s Terms of Service explicitly disavow any warranty or any liability for harm that might result from Google’s negligence, recklessness, malintent, or even purposeful disregard of existing legal obligations to protect the privacy and security of user data.” If a business gives their data to a cloud service/storage provider, how certain can they be that the provider will act responsibly, and what would the legal repercussions be if they are not responsible? These questions cause businesses to hesitate when considering cloud solutions. The problems of “Data access and portability barriers are less relevant in the short term, but are expected to become more relevant in time for all stakeholders, as intensity of cloud usage increases,” [14] states.

The authors of [68] note a security advantage of cloud computing by stating that “Some threats are better dealt with by warehouse-size data centers (e.g. Distributed Denial of Service attacks, which involve attempts to prevent an internet site or service from functioning). Applications running in the cloud are less vulnerable to these attacks.”

As stated earlier, mobile clouds offer the best that cloud computing has to offer, though with very particular drawbacks. With a mobile device, such as a Smartphone or IPad (using 3G or 4G networks), one has access to a huge variety of applications and information that, without the cloud, would be beyond the capabilities of such mobile devices. Smart phone use is on the rise, with [94] predicting “that up to 60 percent of U.S. mobile phone subscribers could be smart phone users by the end of 2013.” The rise has been so fast that it has been difficult to predict. According to [94] “International Data Corporation’s forecast for smart phone sales in 2010 underestimated the actual growth by half.” These smart phone users are very interested in the mobile cloud. The authors of [94] explain it this way:

> Smartphone users are most attracted to the mobile cloud because it can supplement and augment their devices by offering better security, convenience, and an expanded range of functionality compared to device-centric services. Security is top of mind because users recognize the growing importance of their mobile devices and understand how difficult life would be if their devices were lost, stolen, or damaged. Smartphone users also recognize the key value proposition of the mobile cloud-storing all of their critical information, media content, and apps in the cloud, where they can be readily accessed no matter what happens to their mobile device.

Apple advertises on their website that iCloud gives you music, photos, calendars, contacts, and documents, from any device you are currently using and there is no need to transfer data between your devices as they share the same files. The mobile user has an account that can be accessed on multiple devices with minimum effort and no repurchasing. Apps such as SoundHound and Shazam use a cloud database and an algorithm to identify songs by letting the owner's mobile device "listen" to the music. Something like this would not be possible without access to the huge store of data in the cloud and mobility.

Ultraviolet, a feature that is included with most BluRay purchases, allows you to store your movies in the cloud and access them anywhere, anytime. This concept has appeal because the idea of ownership is more concrete. You own the movie, and have an actual disk that proves it. The mobile cloud (Ultraviolet) merely lets you have increased access to it, and makes your life more convenient.

Business users are also taking advantage of the mobile cloud, and are far more likely to do so if they already utilize cloud services on their PCs. [96] Mobile services they utilize include dropped-call reconnect, visual voicemail, messaging history, mobile conferencing, document management, and “specific business apps that allow them to extend the boundaries of their offices.” [96] Whether for business or personal use, [95] believes mobile users are interested in having access to a virtual desktop infrastructure (VDI) that they can personalize, and would have access to from any mobile device. The idea of being able to utilize a cloud service, but at the same time have something personalized, not one-size-fits-all, greatly appeals to consumers.

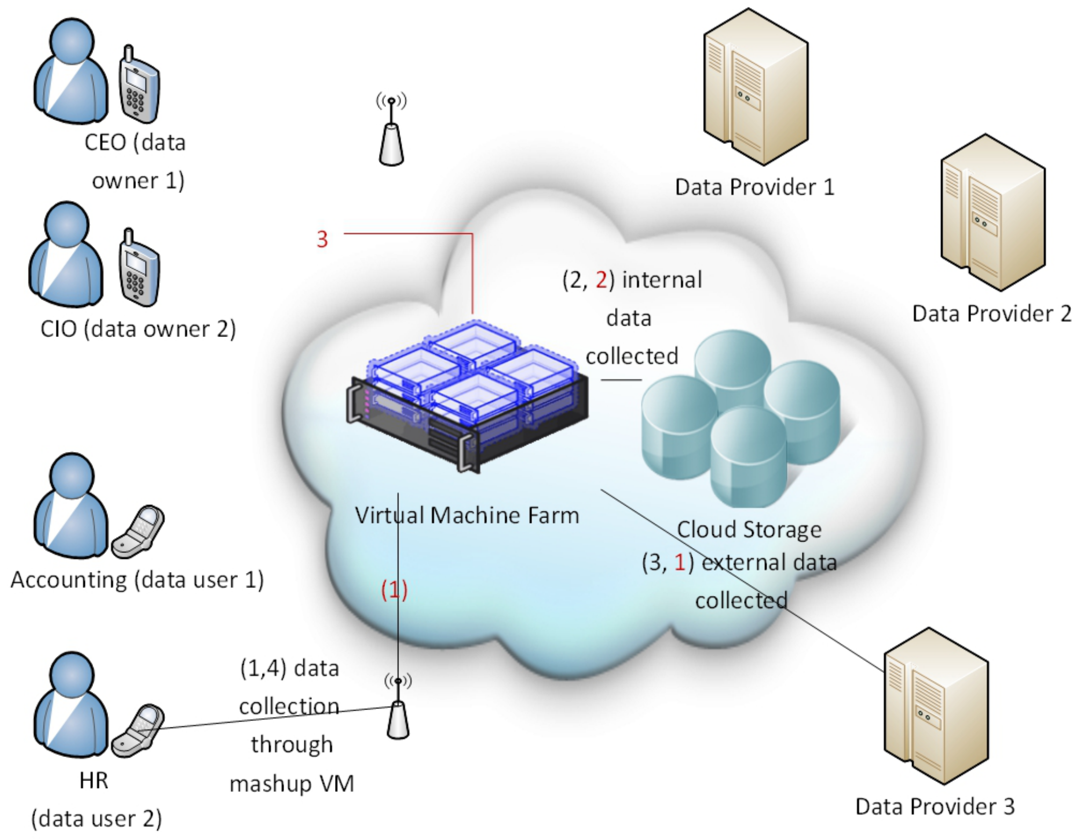
The authors of [94] mention security as being a benefit, but mobile cloud usage involves a certain amount of risk. The authors of [49] quell the enthusiasm for the mobile cloud by pointing out that “mobile devices, particularly embedded devices, require proper set up and protection to be of benefit overall, which include restrictions on the type of data maintained on the device.” The authors of [24] submit that, with the increased popularity of smart phone use, “now is a good time to understand the security threat to the mobile cloud and to begin to prepare for an inevitable increase in security threats.”

One threat is malware, as [24] points out, “smart phones, being sophisticated and fully featured computers, are receiving the growing attention of malware creators.” The authors of [15] state that “there are three main [mobile] security threats – malware, privacy and authenticating access.” He downplays the threat by arguing that, since operating systems are more diverse on mobile devices, attacks will be less likely than with a PC. “It’s hard to know where to focus the attack with the mobile cloud,” [15] conjectures. He states directly afterwards that “There are other attack vectors people could potentially use. For instance, one that will work on any mobile device is an SMS message that can get to the device and persuade it to part with information." In [17]'s stipulations about cloud computing, it requires "Data (including sensitive personal information) must be encrypted in transit and at rest where potentially at risk (e.g., on mobile devices)." Though, as this chapter has pointed out previously, encryption does not guarantee security.

Other institutions, including governmental, scientific and healthcare, may benefit from cloud computing technology. The authors of [68] mention "the rapid evolution and widespread deployment of sensors - in the soil, tree canopies, in gene sequencing machines, in telescopes, on the sea floor." The point of which would be to "communicate their location, identity, and local measurements of their environment wirelessly over low-power computer networks." This data would be compiled and analyzed in the cloud, where data could be shared, and conclusions reached, far more quickly than would otherwise be possible. The authors of [52] grant that "The cloud's vast computing power is making it easier and less expensive for companies and clinicians to discover new drugs and medical treatments. Analyzing data that used to take years and tens of millions of dollars can now be done for a fraction of that amount." "The ability to analyze vast amounts of data in this way is changing lots of industries - including healthcare." The authors of [87] discuss both the benefits and challenges of having all medical records 'in the cloud.' While it enables easy communication between all people in the medical profession (doctors, nurses, pharmacists) it also means risking the privacy of patients. Medical professionals will have to decide if the convenience is worth the risk.

Another trend in the world of cloud computing is mashups. Mashups integrate existing web services and content to create a new web tool. The authors of [56] are more specific, defining a mashup as “a Web-based application that is created by combining and processing on-line third party resources, that contribute with data, presentation or functionality.” Facebook and Google maps are examples of popular mashups. Facebook is littered with mashups, although they call them ‘apps.’ Various games (such as Angry Birds and Words with Friends), and websites such as Pinterest, hulu and Goodreads are part of the mashups available on Facebook. You can search for restaurants in a specific area, and Google maps will show you a map with the exact locations of those restaurants pinpointed. The next step, requesting directions to these places, is also provided on the same page. These services don’t originate from a single place, but are brought together for the convenience of the user. There are two types of mashups- server side and client side. These two categories are based on where the data is ‘mashed up.’ Server side mashups occur on the server, and client side mashups integrate services and content on the client. [71] Usually a combination of server-side and client-side is used in the creation of a mashup. [56]

Security is a problem for mashups, particularly when it comes data integrity and privacy. Not much is being written about these problems. The authors of [56] state that “While security challenges have been identified for mashups only few approaches exist that try to handle security lacks and identity of mashups.” The provider does not have full control over the content of the mashup, so [56] determines that “This requires mechanisms to control the user connection and the data security.” Trust certificates, which would “assure the end-user the trustworthiness of the content and also the integrity of the mapping application,” are recommended for use with mashups. These certificates would work in much the same way as certificates given to online shopping sites. [56] The issue of same-origin policy



1) Data user's mashup requests data internal and external to company. Request is sent to user's mobile cloud virtual machine hosted by Infrastructure/Software-as-a-Service provider.

2) Virtual machine requests data from Storage-as-a-Service provider, hosting secured internal company data.

3) Virtual machine requests data from external unsecured data provider.

4) mashed data is computed and returned to mobile user.

1) Attacker can watch here for mobile and external data provider traffic correlations to compromise user privacy.

2) Attacker can compromise data confidentiality through various techniques such as wire tapping.

3) Attacker can launch malicious VM to compromise parallel mobile user VMs.

Figure 1: A diagram of our target universe.

has been mentioned in certain contexts, such as what [5] provides. First, an explanation of same-origin policy, given by [5]:

> Browsers implement the same-origin policy as a protection mechanism in order to isolate Web applications coming from different domains from each other, under an assumption that different domains represent different originators. As a result, if applications in multiple windows or frames are downloaded from different servers, they will not be able to access each other's data and scripts. Same-origin policies make it difficult for mashups to work, so programmers 'bypass' the problem. What isn't discussed, however, is the security vulnerabilities this opens up. If your bank account, or other similarly private and sensitive data, is part of the mashup anyone who has access to one part of the mashup (something less private) could also have access to your sensitive data. The same-origin policy that normally prevents this has been bypassed, and therefore rendered useless.

Figure 1 gives an illustration of our target universe, without losing generality. In this scenario we have multiple users. Some own the data the organization uses and some simply access this data to perform their tasks. As an example within this target universe, we have a data owner opening their mashup application (this would normally be performed automatically by the user's smartphone). This mashup application requires data that is both internal to the company and external from outside providers. To offload the mashup task to the cloud, the request is sent to the user's mobile cloud-based virtual machine. This virtual machine is hosted at a provider which hosts many virtual machines for many outside users. This same virtual machine, must then request the data the mashup application requires which is, again, both internal and external to the organization. In this example the internal data is stored on another cloud providers STorage-as-a-Service (STasS). The virtual machine must request this data over encrypted channels to build part of the mashup result. Once received, the mashup also requires data hosted by external providers which do not encrypt their transmissions to their end users. The virtual machine makes its un-encrypted request to these providers and receives its unencrypted result. This data is then "mashed up" and returned to the original mobile user.

There are many ways an attacker could compromise the security of our end user. We have chosen to focus on the most important ones to this situation, which are outlined in red in figure 1. We feel these attacks are the most dangerous because they involve the most "legal" actions or stealthy and attack can conduct, thus minimizing the risk of the attacker being caught or the attack even being noticed. This creates "unknown unknown" vulnerabilities. They are unknown, because the entities in the position to detect the attacks do not know how to detect them. They are "unknown unknowns" because these same entities also do not know they even need to be on the lookout for such attacks. It is around these attacks this dissertation is designed.

First, attackers could listen to the traffic from the virtual machine provider to correlate unencrypted requests from data providers and traffic to end users to determine which unencrypted requests match up with encrypted mobile user return traffic. During this completely passive attack, the attacker needs to only listen to one Internet connection (the virtual machine provider) to conduct this correlation attack. The results of this attack would enable the attacker to determine information about many mobile users such as their geographical position, the companies they rent cars from, the hotels they plan on staying at and the list goes on from there.

Second, Attackers must be prevented from tricking the STasS provider the organization's internal data is stored from allowing the wrong users from accessing the content. The struggle is to prevent from taxing the resources of the STasS provider too much, while also making the encryption key space small enough for data owners and data users to easily be able to store without becoming too large. This must still be done in a manner that is versatile enough to allow many different users to have permissions to different sections of data while not accidently giving any users permissions over data they should not have access to, which to the STaaS Provider may have looked like a legitimate access request, thus preventing a breach of confidentiality.

Third, an attacker could legitimately create a virtual machine on the same cloud provider our organization user's virtual machine is hosted. If this virtual machine ends up on the same host, this attacker could perform Operating System or data set fingerprinting through only memory access requests on the attacker's virtual machine. Through our presented "guess and check" strategy, the attacker would be able to confirm "guesses" of memory content and confirm these "guesses" through memory access delays within their own virtual machine, thus providing a compromise of confidentiality. In the case of Operating System fingerprinting, this will further provide the opportunity for "one-hit-knock-out" attacks because the attacker would not have to scan the target virtual machine first with network based scanners. Once the attacker has penetrated the virtual machine, further levels of security such as data integrity can be easily accomplished before an intrusion detection system has time to respond.

1.2 Contribution

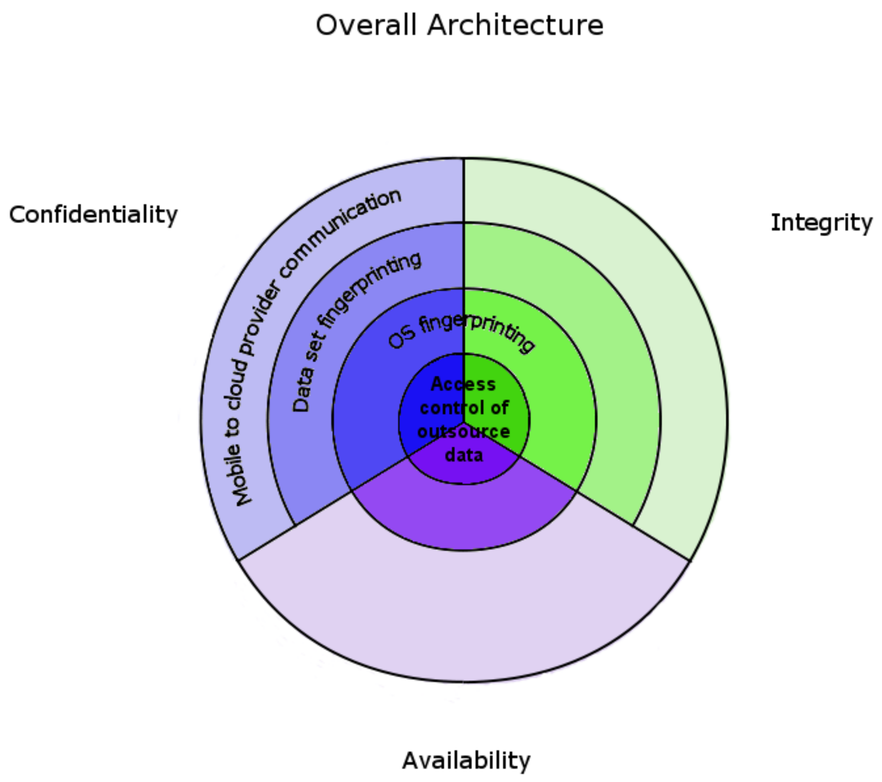


Figure 2: A model of our target areas.

This dissertation explores several approaches for exploiting public cloud computing infrastructures and their possible mitigation strategies. The main contributions of this dissertation are the following:

1. Flexible cryptography-based access control of outsourced data.
2. Methods for securing virtual machines from OS and Data Fingerprinting.
3. Secure mobile-to-cloud-to-provider communication.

A graphical illustration of the scope of this dissertation is presented in Figure 2. This dissertation looks at the low level security of cloud computing by exploring OS and data set fingerprinting in virtual machine hypervisors. Additionally, we look at protecting the confidentiality, integrity, and availability of outsourced data, from the point of view of the cloud computing clients. Finally, we look at the security of the multiparty communication required for mobile-to-cloud-to-provider mashup computing.

1.3 Outline

The remainder of this dissertation begins with chapter 2, which presents our method to provide Secure and Efficient Access to Outsourced Data. Chapter 3 presents virtual machine OS and Data Fingerprinting and our experiments on how to mitigate against attackers who may use these fingerprinting techniques to gain secret insights into penetrating a cloud computing virtual machine. Chapter 4 presents our methods of providing secure communications between both the mobile user to the cloud and the cloud to the information provider using our newly de-duplication attack defended cloud computing virtual machines. Chapter 5 concludes this dissertation.

CHAPTER 2: SECURE AND EFFICIENT ACCESS TO OUTSOURCED DATA

2.1 Introduction

Since the daily operations of modern corporations heavily depend on their information processing capabilities, the costs and overhead to manage their computation resources pose serious challenges to these companies. To free the companies and their manpower from the burden of IT services, the concept of cloud computing has been proposed. In this new environment, a client may choose to outsource its data storage, information processing, or even the whole information infrastructure to a service provider. These new services allow companies to focus more on their core business and leave the IT operations to professionals. While the concept of cloud computing provides a new method for information processing, the security problems must be properly solved before these services can be widely deployed. Since many service providers are untrusted, the confidentiality and privacy of the clients' information must be protected by some mechanisms.

Among various services of cloud computing, enabling secure access to outsourced data lays a solid foundation for information management and other operations. However, more research efforts are needed to achieve flexible access control to large-scale dynamic data. For example, using asymmetric encryption to protect data or metadata [40] will impact the adoption of the outsourcing platform by devices with limited computational power (e.g.

mobile devices). At the same time, user-group-based data encryption may lead to a complicated access hierarchy after a series of grant and revocation operations. [32]

In this chapter, we focus on the data outsourcing scenario investigated in [26, 27, 31, 32]. In this environment, the data can be updated only by the original owner. At the same time, end users with different access rights need to read the information in an efficient and secure way. Both data and user dynamics must be properly handled to preserve the performance and safety of the outsourced storage system. Before presenting the details of the proposed approach, we use an example to illustrate the potential applications.

The world's largest collider accelerator LHC (Large Hadron Collider) can generate about 10 PB (Peta-Bytes, 10^{15} bytes) data each year. [84] The data can be stored on a third party cluster and the European Organization for Nuclear Research may publish new data, update existing records, and delete expired information. The data can be accessed by scientists in different countries. Since the scientists may have different security clearance levels, encryption based access control will be adopted. At the same time, methods must be designed to support dynamic changes of the access rights of the end users.

Enforcing data security in this scenario poses new challenges for researchers. First, since the size of the outsourced data could be huge, we want the server to store only one copy of each data block (the data should be encrypted). Second, since a popular storage outsourcing pricing model is pay-per-use (e.g. Amazon S3), we want to reduce the number of operations on the storage server except for information access and updates. Specifically, we want to avoid data re-encryption caused by changes of user access rights. Last, but not least, we want to provide fine-grained access control to end users.

In this chapter, we propose to develop a new approach [105] that integrates several advanced techniques to solve these problems. First, we encrypt every data block with a different symmetric key and adopt the key derivation method [8, 27] to reduce the number of secrets that the data owner and end users need to maintain. Different from [27], we do not organize users into groups based on their access rights. Our method, although leading to more data encryption keys, will simplify operations during user access right changes. We propose to store the metadata for key derivation with the corresponding data blocks. In this way, we can establish multiple indexes of the blocks to reduce the communication overhead for key distribution during data access. Second, we adopt over-encryption by the server [32] to achieve data isolation among end users even when they have the same access rights. For the servers that refuse to conduct over-encryption, we propose to use lazy revocation [50] to prevent revoked users from getting access to updated data. Finally, we present detailed methods to handle dynamics in both user access rights and outsourced data. To summarize, the contributions of the research include:

- The proposed approach provides fine grained access control to outsourced data with flexible and efficient management. The data owner needs to maintain only a few secrets for key derivation. It does not need to access the storage server except for data updates.
- We propose comprehensive mechanisms to handle dynamics in user access rights and updates to outsourced data. The metadata for key derivation will allow us to establish multiple indexes for different data access patterns of the end users.

The remainder of the chapter is organized as follows. Section 2 discusses the related work. In Section 3, we describe assumptions in the investigated scenario. We also discuss requirements to the proposed approach. Section 4 presents the details of the data access procedure. We introduce the construction of the key hierarchies and the key derivation procedures. The organization of the metadata and data blocks will also be presented. In Section 5, We describe how to calculate the key set. In Section 6, we describe mechanisms to handle dynamics in outsourced data blocks and user access rights. Section 7 investigates the overhead of the proposed approach through simulation. Section 8 studies the safety and scalability of the proposed approach. Finally, Section 9 discusses future extensions and concludes this chapter.

2.2 Related Work

Although the official name of 'cloud computing' is proposed in recent years, the concept of treating data, storage, software, platform, and even infrastructure as a service has been investigated for a long time. Cloud computing is poised to provide computation for public utilities, such as water and electricity, and as such, new challenges arise for the confidentiality, privacy, and integrity of the information and resources in the system. Although many research papers have been published on related topics, security research for cloud computing is still in its early stage. Therefore, below we first review the expected properties of data security in cloud computing and map them to the investigated scenario. We will then discuss the achievements in two research directions: secure remote untrusted storage and key management for access hierarchies, from which our proposed approach benefits.

Requirements of Data Security in Cloud Computing

Different from many fields in which a big gap exists between academic research and industry applications, cloud computing has attracted attentions from both sides since the very beginning. For example, secure data storage and management is an important component of the security guidance recently published by Cloud Security Alliance [23], the membership of which includes the leading corporations in cloud computing such as Sun, eBay, Visa, and McAfee. In the guidance, a secure data outsourcing service should be evaluated from at least the following aspects: (1) strong encryption and scalable key management; (2) user provisioning, de-provisioning, and information lifecycle management; and (3) system availability and performance.

For the first aspect, in this chapter we propose to use data block level symmetric encryption. Since the proposed mechanism does not depend on any specific encryption algorithms, the end users can make their choices based on the requirements of the applications. The key derivation tree structure will allow data consumers to use a few keys to generate all secrets in need. For the second aspect, we provide detailed description on handling dynamics in user access rights and data blocks. For the last aspect, the performance and overhead analysis is conducted in Section 7.

Secure Remote Storage

Securing outsourced data for multi-user accesses can be achieved through encrypted file systems. However, the following analysis will show that existing approaches cannot satisfy the requirements of the example application discussed in Section 2.1.

To allow users to get secure and efficient access to outsourced data files, both data and metadata must be properly protected. An early approach [66] presents the basic idea to encrypt the information and use hash values and digital signatures to guarantee information integrity. FARSITE [1] uses symmetric secrets to encrypt files and uses every reader's public key to encrypt the symmetric keys. In Plutus [50], both files and directory information are encrypted. It uses *sign* and *verify* keys respectively to determine whether or not a user can write or read a file. Since the key generation procedure depends on power-modular computation, the overhead is relatively heavy. SiRiUS [40] adopts a more complicated structure. Every data file has an encryption key and a sign key. At the same time, every user has a public/private key pair. The approach continuously signs the metadata tree to guarantee the freshness of the information and prevent rollback attacks. Every time a secret is revoked, it will generate a new key and reencrypt corresponding data files. In SUNDR [57] the authors use hash trees and chains to prevent fork attacks and guarantee that users have the same view of files. They use update certificates to handle concurrent updates. The advantages and disadvantages of many approaches can be found in [53]. Recently, a data sharing platform for outsourced information using asymmetric encryption is proposed in [88]. We find that all these approaches adopt asymmetric encryption to protect data confidentiality. At the same time, encrypting information at the data block level will make the key management mechanism of secure file systems very cumbersome. Therefore, a new approach is needed to protect the safety of the outsourced data.

Key Management for Access Hierarchies

To enable secure and efficient access to outsourced data, investigators have tried to integrate key derivation mechanisms [19, 20, 59, 114] with encryption-based data access control. Atallah *et al.* [8] propose a generic method that uses only hash functions to derive a descendant's key in a hierarchy. The method can handle updates locally and avoid propagation. Although the proposed key derivation tree structure can be viewed as a special case of access hierarchies, analysis in Section 8 will show that our method serves the studied application better.

In [27], the authors divide users into groups based on their access rights to the data. The users are then organized into a hierarchy and further transformed to a tree structure to reduce the number of encryption keys. This method also helps to reduce the number of keys that are given to each user during the initiation procedure. In [32], data records are organized into groups based on the users that can access them. Since the data in the same group are encrypted by the same key, changes to user access rights will lead to updates in data organization. While a creative idea in this approach is to allow servers to conduct a second level encryption (over-encryption) to control access, repeated access revocation and grant may lead to a very complicated hierarchy structure for key management. In [31], the approach will store multiple copies of the same data record encrypted by different keys. At the same time, when access rights change, reencryption and data updates to the server must be conducted. These operations will cause extra overhead on the server and do not fit into our application scenarios. An experimental evaluation of these approaches can be found in [26].

2.3 Problem Definition

In this section, we briefly sketch out the application scenario under investigation and the system assumptions. We also describe the requirements to the proposed data outsourcing mechanism.

Application Scenario and Assumptions

As the example in Section 1 illustrates, the owner-write-users-read scenario is a popular case in the storage outsourcing applications. Figure 3 provides an abstract illustration of the scenario under investigation. The data owner stores a large amount of information on the service provider. Since the service provider is untrusted, the owner will encrypt the outsourced data before putting them on the server. Here we assume that the smallest information access unit is called a 'block'. This is an abstract concept and it may have different meanings in different systems. To provide fine-grained access control, the encryption will be conducted at the block level. Only the owner can make updates to the outsourced data. Here the operations include updates to data blocks, and deletion, insertion, and appending of blocks. We also assume that there exist pre-distributed secrets between data owner and service provider, and between data owner and end-users. The key distribution and update problem is beyond the scope of this research and we refer interested readers to existing approaches such as [13].

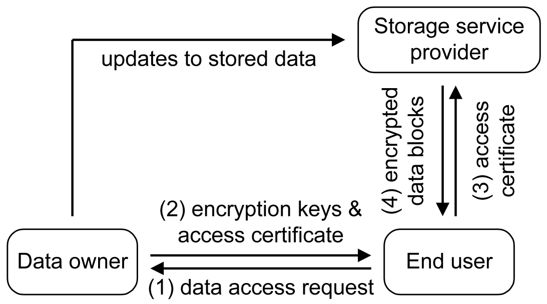


Figure 3: Illustration of the application scenario.

The outsourced data can be accessed by many different end users that are distributed all over the network. Since the end users may use devices with weak processing capabilities such as PDAs, we want to avoid computationally expensive operations such as asymmetric encryption of data blocks. At the same time, we want to reduce the amount of information that is stored on the end users. The access rights of the end users are different and they may change (grant and revocation) as time proceeds. Therefore, right keys must be provided to the end users to control their access.

We assume that the service provider adopts a curious but not malicious model. That means, the provider will not intentionally send wrong data blocks to an end user but it will try to get access to the plaintext of the stored information. To preserve confidentiality of the outsourced data, the owner may ask the service provider to conduct a second level encryption (over-encryption) [32] before the data is sent to the end users. For providers that refuse to offer this service, we adopt the lazy revocation method [50] to reduce information leakage through eavesdropping.

With the introduction of the roles of data owner, service provider and end user, we can describe the data access procedure as follows. An end user will send a data access request to the owner. The owner will refer to its access control matrix and send back corresponding encryption keys through the secure channel between them. At the same time, the owner will send back a data access certificate to the end user. The user will then present the certificate to the service provider. The provider will verify it and send the corresponding encrypted data blocks to the end user. We assume that the end user has information to verify the integrity of the received data. [41, 107] In this way, end users directly communicate with the service provider to get the data blocks and the owner will not become the data transfer bottleneck.

System Requirements

In this section, we describe the requirements on efficiency and security to the proposed approach. First, since the outsourced data could be huge and the service provider may charge the owner based on used space, we want to maintain only one encrypted copy for each data block on the outsourced storage. This will put new challenges to the key management mechanism. Second, in addition to providing the storage space, the service provider may or may not offer the service of over-encryption [32] when it sends the data blocks to end users. The proposed mechanism must properly handle both cases to preserve data confidentiality. Third, since the service provider may have a pay-per-use pricing policy, the data owner should reduce the number of information accesses to the service provider when they are not caused by updates to data blocks. That means we want to avoid data block reencryption when the access rights of end users change. Last but not least, we want to reduce the storage, communication, and computation overhead on the data owner and end users to promote the wide adoption of the proposed approach. Later discussion will show that some of the requirements conflict with each other and the designed approach will try to achieve a tradeoff.

2.4 Secure and Efficient Data Access

In this section, we present the details of the proposed approach. We first introduce key-derivation-based data block encryption. We will then describe the data access procedure. Mechanisms to reduce the overhead on the data owner and to prevent information access from revoked users will also be discussed.

Determining Keys for Data Encryption

As we introduce in Section 3, the smallest information access units are data blocks. Therefore, to provide fine-grained access control, we propose to encrypt every data block with a different secret. Here we do not assume the adoption of any specific symmetric encryption algorithm and leave the choice to the system designers. However, an efficient mechanism must be designed to allow data owner and end users to manage the encryption keys. We assume that the outsourced data contains n blocks $\{D_1, D_2, \cdots, D_n\}$ and each block is encrypted with a randomly generated secret k_i $(i$=1 to $n)$. If the data owner keeps a copy of all these keys, the storage overhead will be linear to n. At the same time, when an end user needs to access l data blocks, the communication overhead between the owner and the user for key distribution will also be linear to l. This overhead can be overwhelming for many end users when we consider that the outsourced data can easily contain millions of blocks. Therefore, a more efficient key management method must be designed.

To solve this problem, we propose to adopt the key derivation method. [8] The basic idea is as follows. Each of the encryption keys k_i can be represented as the XOR result of a public part and a private part. The public part will be stored as metadata with the corresponding block. The private part, on the contrary, will be generated through key derivation hierarchies. Here every key in the hierarchies can be derived from its ancestors. Two hierarchies are used in our example to enable more versatile key assignment for different sections of data blocks they are mapped to. Since the derivation procedure uses a one-way function, we cannot reverse the computation to get the secret keys of the ancestors and sibling nodes. In this way, the data owner needs to maintain only the root nodes of the hierarchies.

During the key distribution procedure, the owner can send the secrets in the hierarchies to end users based on their access rights. The end user will derive the leaf nodes in the hierarchy/hierarchies. It will then combine the derivation results with the public metadata to decrypt the data blocks. The cost of this approach includes the calculation of one-way functions and XOR operations during key derivation. Since previous experiments [100] show that both operations can be accomplished very efficiently, in this paper we propose to trade computation for storage and communication overhead.

While there are different choices of the organization of key hierarchies and key derivation functions, below we present an approach using the binary tree structure and hash functions. Both hierarchies are computed in the same way, but with different root nodes, so we will describe a one hierarchy derivation. Without losing generality, we assume that the outsourced data contains n blocks and $2^{p-1} \leq n < 2^p$. Therefore, we can build a binary tree with height p as follows. The data owner will choose a root secret (root node) $s_{0,1}$. Here the first index of the key represents its level in the hierarchy, and the second index represents its sequence number in the level. For example, for level x in the hierarchy, the sequence numbers are from 1 to 2^x. In this way, for node $s_{i,j}$ in the hierarchy, its parent is $s_{(i-1),(\lceil j/2 \rceil)}$ (when $i \neq 0$), and its children are $s_{(i+1),(2*j-1)}$ and $s_{(i+1),(2*j)}$ (when $s_{i,j}$ is not a leaf node). The nodes in level p will be used as the private parts to calculate the encryption keys k_i. The public parts y_i will be stored with the blocks D_i as metadata. An example hierarchy is illustrated in Figure 4.

Now let us look at the key derivation procedure. The data owner chooses a public hash function $h()$. For any node $s_{i,j}$ in the hierarchy, its left child can be calculated as $s_{(i+1),(2*j-1)} = h(s_{i,j}||(2*j-1)||s_{i,j})$. Here we 'sandwich' the sequence number of the child node with the parent's key and then apply the hash function. We can calculate the right child of $s_{i,j}$ in a similar way. Through repeatedly applying this function, a node can calculate the secrets of all of its descendants. When we reach level p of the hierarchy, the hash results can be XORed with the metadata to get the encryption keys. Since XOR is a bit by bit manipulation, the public part (y_n and z_n in figure 4) that is XORed can be chosen to give the same keys when secrets from different trees point to the same data blocks.

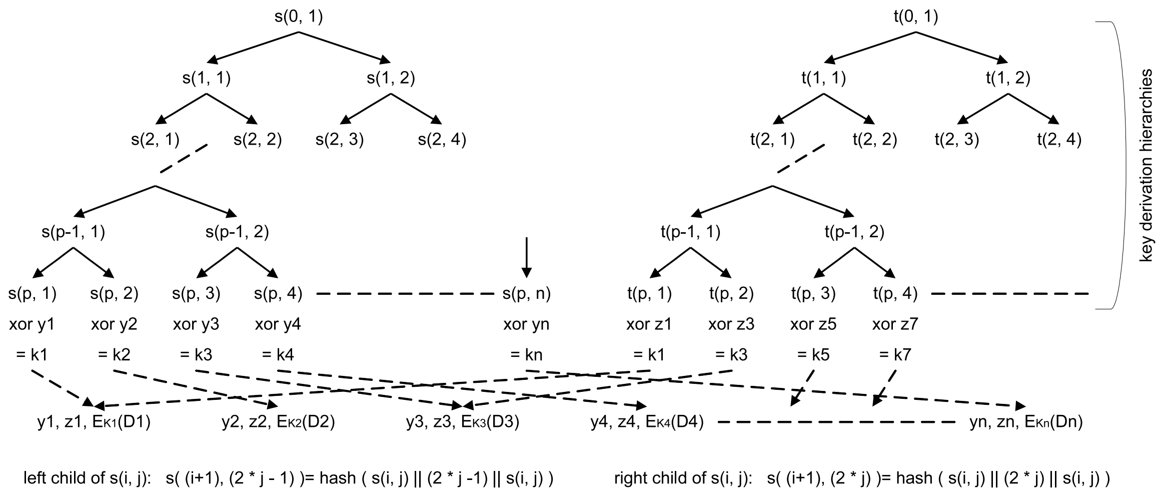


Figure 4: Key derivation hierarchies.

The safety of the approach comes from the one-way property of the hash function. While a node can easily derive its descendants in the hierarchy by applying the public hash function, it has to reverse the function to get the secrets of its siblings or ancestors. If a secure hash function is chosen, the end users cannot decrypt the data blocks that they are not authorized to access.

We need to pay attention to several issues when we establish the key hierarchies and distribute the secrets during information access. First, when we choose the height of the hierarchies, we need to leave some room for the insertion and appending operations to the outsourced data. The details of such operations will be discussed in the next section. Second, we should not disclose encryption keys of the blocks that are temporarily missing from the outsourced data. For example, when $n = 7$ and an end user can access blocks D_5, D_6 and D_7, the owner should send $s_{2,3}$ and $s_{3,7}$ to the user instead of $s_{1,2}$. In this way, later when we append D_8 to the outsourced data and the user cannot access it, we do not need to revoke the secret.

We notice that end users' access rights have a direct impact on the communication overhead of the proposed approach. For example, if one secret in the hierarchy can be used to derive all encryption keys of the data blocks that a user needs to access but not any other keys, the owner needs to send only this secret to the end user. On the contrary, if we have only one key derivation hierarchy and the data blocks that a user wants to access do not share any common ancestors in level $(p-1)$ of the hierarchy, the number of returned keys will be equal to the number of requested blocks.

Two methods can be used to solve this problem and improve the efficiency of the owner. First, we can use the method described in [27]. The basic idea is to organize data blocks with similar access patterns into groups and give them sequential index numbers when they are outsourced. In this way, the end users have a higher probability to access data blocks with consecutive index numbers and the owner can locate a few keys in the hierarchy to satisfy a reading request. The disadvantage, however, is that the block organization can be mapped to only one access pattern.

Second, we can generate multiple key derivation hierarchies, each of which will have a different mapping function between the leaf nodes and the index numbers of the outsourced blocks. Here the metadata for the leaf nodes of each hierarchy must be stored with the data blocks. An example of two hierarchies is illustrated in Figure 4. In the first hierarchy, the sequence numbers of the leaf nodes and the index numbers of the data blocks are identical. In the second hierarchy, the left half of the tree maps to all blocks with odd index numbers, and the right half maps to blocks with even index numbers. During an information access procedure, the owner can examine both hierarchies to locate the smallest key set for secret derivation. The cost of maintaining multiple hierarchies includes the storage overhead of the metadata and hash calculation during secret derivation. A quantitative analysis will be presented in Section 7.

Data Access Procedure

In this part we describe the data access procedure. To prevent revoked users from getting access to outsourced data through eavesdropping, we assume that the service provider will conduct over-encryption [32] when it sends data blocks to end users. To conduct this operation, the service provider and end users need to share a pseudo random bit sequence generator $P()$. [21, 58] Given a *seed*, $P()$ can generate a long sequence of pseudo random bits. The scenarios in which the service provider refuses to offer this operation will be discussed in the next section.

We use $\mathscr{O}$ to represent the data owner, $\mathscr{S}$ to represent the service provider, and $\mathscr{U}$ to represent the end user. We assume that $\mathscr{O}$ shares the pairwise keys k_{OU} and k_{OS} with $\mathscr{U}$ and $\mathscr{S}$ respectively. With these assumptions, the data access procedure works as follows.

1. $\mathscr{U}$ will send a data access request to $\mathscr{O}$.

$$\mathscr{U} \rightarrow \mathscr{O} : \{\mathscr{U}, \mathscr{O}, E_{k_{OU}}(\mathscr{U}, \mathscr{O}, request\ index, data\ block\ indexes, MAC\ code)\}$$

Since only $\mathscr{U}$ and $\mathscr{O}$ know k_{OU}, $\mathscr{O}$ will be able to authenticate the sender. The *request* *index* will be increased by 1 every time $\mathscr{U}$ sends out a request and it is used by $\mathscr{O}$ to defend against replay attacks. The request contains the index numbers of the data blocks that $\mathscr{U}$ wants to access. The Message Authentication Code (MAC) will protect the integrity of the packet.

2. When $\mathscr{O}$ receives this message, it will authenticate the sender and verify the integrity and freshness of the request. It will then examine its access control matrix and make sure that $\mathscr{U}$ is authorized to read all blocks in the request. If the request passes this check, the owner will determine the set of keys $\mathscr{S}'$ in the hierarchies such that (1) $\mathscr{S}'$ can derive the keys that are used to encrypt the requested data blocks; and (2) $\mathscr{U}$ is authorized to know all keys that can be derived from $\mathscr{S}'$. The algorithm to determine $\mathscr{S}'$ from multiple hierarchies will be presented in Section 4.

The owner will then generate the reply to the end user.

$$\begin{aligned} \mathscr{O} \to \mathscr{U}: \quad & \{\mathscr{O}, \mathscr{U}, E_{k_{OU}}(\mathscr{O}, \mathscr{U}, request\ index, ACM\ index, \\ & seed\ for\ P(), \mathscr{S}', cert\ for\ \mathscr{S}, MAC\ code)\} \end{aligned}$$

Here the request index is used to uniquely label this reply. The ACM index is used by $\mathscr{O}$ to label the freshness of the Access Control Matrix (ACM). This index will be increased by 1 every time $\mathscr{O}$ changes some end user's access rights. The updated ACM index will be sent to $\mathscr{S}$ by $\mathscr{O}$ to prevent those revoked users from re-using expired certificates to access data blocks. The seed is a random number to initiate $P()$ so that $\mathscr{U}$ can decrypt the over-encryption conducted by $\mathscr{S}$. $\mathscr{U}$ will use $\mathscr{S}'$ to derive the data block encryption keys. The *cert* in the packet is a certificate for the service provider and it has the following format:

$$\{E_{k_{OS}}(\mathscr{U}, request\ index, ACM\ index, seed, indexes\ of\ data\ blocks, MAC\ code)\}$$

3. The user $\mathscr{U}$ will send $\{\mathscr{U}, \mathscr{S}, \textit{request index}, \textit{cert}\}$ to the service provider. When $\mathscr{S}$ receives this packet, it can verify that the *cert* is generated by $\mathscr{O}$ since only they know the secret k_{OS}. $\mathscr{S}$ will make sure that the user name and request index in *cert* match to the values in the packet. If the ACM index in *cert* is smaller than the value that $\mathscr{S}$ receives from $\mathscr{O}$, some changes to the access control matrix have happened and $\mathscr{S}$ will notify $\mathscr{U}$ to get a new *cert*. Otherwise, the service provider will retrieve the encrypted data blocks and conduct the over-encryption as follows. Using the *seed* as the initial state of $P()$, the function will generate a long sequence of pseudo random bits. $\mathscr{S}$ will use this bit sequence as a one-time pad and conduct the XOR operation to the encrypted blocks. The computation results will then be sent to $\mathscr{U}$.

4. When $\mathscr{U}$ receives the data blocks, it will use the *seed* to re-generate the pseudo random bit sequence and use $\mathscr{S}'$ to derive the encryption keys. It will then recover the data blocks.

This approach adopts two methods to reduce the overhead on the data owner. First, the *cert* that it provides to the end user does not contain a timestamp. Therefore, if the access control matrix has not been changed and the ACM index value has not been updated, a user can reuse previous *cert*s to access the data blocks. Second, during the whole data access procedure, the owner only needs to use the root key(s) to determine $\mathscr{S}'$ by calculating a group of hash functions. Since this computation can be accomplished very efficiently, the data owner will not become the bottleneck in the application.

This approach adopts two methods to prevent revoked users from getting access to the outsourced data. First, when an end user $\mathscr{U}$ loses access to some data blocks, the access control matrix at $\mathscr{O}$ will be updated. This will lead to the increase of the ACM index and the value will be sent to $\mathscr{S}$. In this way, if $\mathscr{U}$ presents the old *cert* to $\mathscr{S}$, it will be rejected. $\mathscr{U}$ can still get access to the data blocks by eavesdropping on the traffic between $\mathscr{S}$ and other end users if it has kept a copy of the key set $\mathscr{S}'$. To defend against such attacks, we ask the service provider to conduct over-encryption before sending out the data blocks. Since for every data request the *seed* is dynamically generated and never transmitted in plaintext, $\mathscr{U}$ will not be able to regenerate the bit sequences of other end users. Therefore, unless $\mathscr{U}$ keeps a copy of the data blocks from previous access, it will not be able to get the information.

2.5 Calculating the Key Set $\mathscr{S}'$

Given a key derivation hierarchy, it is fairly easy to locate the smallest set of nodes that can be used to calculate the encryption keys of the blocks in a data reading request. When multiple key derivation hierarchies exist in the system, it is possible to combine the node sets from different hierarchies and find an even smaller node set to calculate the keys. This procedure, however, can be very computationally expensive. Below we discuss the two problems respectively.

Since the encryption key of a data block will be mapped to only one leaf node in a hierarchy, "Determine the node set in one hierarchy" is a greedy algorithm which can be used to determine the smallest node set in it for a reading request.

When multiple hierarchies have been established for the data blocks, we can calculate a node set N for each of the hierarchies. The procedure to select a minimum number of nodes from these sets to cover the blocks in the request is actually a variation of the set covering problem, which is one of the famous NP-complete problems in Karp's list. [51]

Many heuristic approaches have been proposed for this problem [97] and the data owner can balance between the search and communication overhead.

Determine the node set in one hierarchy

```
$N = \emptyset$; $D = indexes\ of\ data\ blocks\ in\ the\ request$;
$While\ (D \neq \emptyset)$
{
    $choose\ any\ d \in D$;
    $locate\ the\ leaf\ node\ n\ in\ the\ hierarchy\ for\ d$;
    $While\ (the\ user\ is\ authorized\ to\ read\ all\ keys$
            $derived\ from\ n's\ parent)$
      $\{n = n.parent;\}$
    $remove\ the\ indexes\ from\ D\ whose\ keys\ can\ be\ derived\ from\ n$;
    $add\ n\ into\ N$;
}
$return\ N$;
```

2.6 Handling Dynamics in System

Thus far, we have considered only static data and access rights of end users. Although some applications, such as digital libraries, have relatively stable data contents, many scenarios for outsourced data storage require the system to support data dynamics. For example, in the DoE case discussed in Section 1, scientists may conduct new experiments and need to add new information to the outsourced data. At the same time, they may find that some data have been mis-calculated and several data blocks on $\mathscr{S}$ need to be updated.

The proposed approach also needs to support changes to access rights of end users. For example, when DoE is collaborating with researchers in country X, it will temporarily authorize them to read more data. When the collaboration is terminated, the access rights will be revoked. Below we show how to amend the basic scheme to handle dynamic operations in the proposed system. The revised approach still tries to satisfy the requirements described in Section 3.2.

Dynamics in User Access Rights

Dynamics in user access rights can be represented as different combinations of two primitive operations: access right grant and revocation.

Grant Access Right

When an end user $\mathscr{U}$ is authorized to read a data block D_i, the owner will change its access control matrix and increase the value of ACM index. The next time that $\mathscr{U}$ submits a data access request, the owner will recalculate the key set $\mathscr{K}'$ based on the new access rights. The service provider and the end user do not need to change to adapt to this update.

Revoke Access Right

Access right revocation is a more complicated event and we need to discuss two mechanisms based on whether or not the service provider conducts over-encryption during data block transmission.

As we describe in Section 4.2, over-encryption conducted by the service provider can defend against eavesdroppers even when they have the data block encryption keys. Under this condition the owner will update the access control matrix and increase the ACM index by 1. It will repeatedly send the new ACM index to the service provider until it receives a confirmation. At this time the revoked user can no longer use its old *cert*. The owner will calculate a new key set $\mathscr{K}'$ when it receives the next data access request from the revoked user.

If the service provider does not conduct over-encryption, the eavesdropper will be able to get access to data blocks if it has kept a copy of the encryption keys. Since the system design criteria require the owner to reduce the number of accesses to outsourced storage, we propose to adopt the lazy revocation method. [50] In lazy revocation, we assume that it is acceptable for the revoked user to read unmodified data blocks. However, it must not be able to read updated blocks. Lazy revocation trades re-encryption and data access overhead for a degree of security. The details of the method are as follows.

When the access right to data block D_i of the user $\mathscr{U}$ is revoked, the access control matrix in $\mathscr{O}$ will be updated and the ACM index increased. At the same time, $\mathscr{O}$ will label this data block to show that some user's access right has been revoked since its last content update. Before D_i is updated for the next time, the owner will not re-encrypt the block on the outsourced storage. Since the ACM index value has been changed, $\mathscr{U}$ can no longer use its old *cert* to access D_i. However, when another user gets encrypted D_i through the network, $\mathscr{U}$ can eavesdrop on the traffic. Since the service provider does not conduct over-encryption, the data will be transmitted in the same format whoever the reader is. Therefore, should $\mathscr{U}$ have kept a copy of the encryption key, it will get access to D_i. This result, however, is the same as $\mathscr{U}$ has kept a copy of D_i before its access right is revoked.

When the owner needs to change the data block from D_i to D_i', it will check the label and find that some user's access right has been revoked. Therefore, it cannot encrypt the updated data block with the current key. To solve this problem, the owner will encrypt a control block with the secret k_i and put it at the slot for D_i. Since the encryption key does not change, the metadata in the key derivation hierarchies will not be impacted. The control block will contain a pointer to another block in which the updated data is stored. It will also contain enough information for the owner to derive the new encryption key. In this way, when a user retrieves this control block from the service provider, it will submit it to the owner. The owner will derive the new key and send it back to the user. At the same time, a new *cert* will be generated so that the user can get the new block from the service provider. A revoked user will be able to read the control block. However, the owner will not send the new encryption key and the *cert* to it. Therefore, the revoked user cannot get access to the updated data. More details of this method will be introduced when we discuss dynamics in the outsourced data.

The adoption of lazy revocation also explains the reason that we want to encrypt every data block with a different key. If we divide data blocks into groups based on the users that can access them, we can encrypt the blocks in the same group with a single key. In this case, whenever a user's access right is revoked, the data block group needs to be fragmented and many blocks need to be re-encrypted. Larger overhead will be caused on the data owner and service provider. At the same time, the requirement to reduce accesses to outsourced storage by the data owner will also be violated.

Dynamics in Outsourced Data

The data owner may need to conduct various operations on data blocks (e.g. update, delete, insert, append). Below we describe the details.

Block Deletion

When a data block D_i is deleted from the outsourced data, the owner will use a special control block to replace D_i. The special block will be encrypted by k_i and stored at the original slot for D_i on the service provider. The metadata for different hierarchies will not be impacted. At the same time, the owner will label its access control matrix to show that the block no longer exists. The end users can still access this control block but they will not get any useful information from the contents.

Block Update

We assume that the owner needs to modify the i-th block of the outsourced data from D_i to D_i'. Since in Section 6.1 we require the owner to maintain a label to show whether or not some user's access right to this block has been revoked since its last update, below we describe two methods based on the value of the label.

If no user's access right to this data block has been revoked since its last update, the owner can update its content in the current storage place. The owner will first locate the slot in which D_i is stored and derive its encryption key. It will then use the key to encrypt D_i' and write the new block to the storage place. The end users will not be impacted by this operation and they will automatically get the new data when they access the block.

If some user's access right to D_i has been revoked since its last update and the service provider does not conduct over-encryption during data transmission, we cannot encrypt the new block D_i' with the current key. On the contrary, we will encrypt a control block with k_i and write it to the i-th block of the outsourced data. The control block will contain the following information: (1) a pointer to the data block in which D_i' is stored; (2) information used by the data owner to derive the new encryption key of D_i'; (3) information used by the data owner to verify the integrity of the control block. The owner will also use the new secret to encrypt D_i' and write it to the corresponding place in S.

When a user needs to access D_i', it will get the encrypted control block from the service provider and submit it to the owner. The owner will verify the authenticity and integrity of the control block and derive the current encryption key. It will then return the key with a *cert* to the user through a secure channel so that the user can access D_i' from $\mathscr{S}$. A revoked user can get the control block but it will not get the new encryption key and the *cert* from the owner.

While there are different ways to implement the block update operation, below we describe one approach. We assume that the data owner will choose two secret keys k_0' and k_{verify}. The former is used to protect the newly generated encryption keys for the updated data blocks, and the latter is used to verify the integrity of the control blocks. When the owner needs to update D_i, it will use k_i to encrypt the control block and store it in the i-th block of the outsourced data. The control block has the format of:

$$E_{k_i}(E_{k_0'}(k_i', 2^p + i, x), hash_{k_{verify}}(k_i', 2^p + i, x))$$

In the first component, the owner will use k_0' to encrypt the newly generated secret k_i', the index number $2^p + i$ of the block, and the number of times x that D_i has been updated. In the second component, k_{verify} will be used to generate the message authentication code of the information to protect its integrity. The owner will encrypt D'_i with k'_i and store the result in the block with the index number $(2^p + i)$. Figure 5 illustrates the update operations.

When a user $\mathscr{U}$ needs to access the updated data block D'_i, it will first get the encrypted control block from $\mathscr{S}$ and submit it to the data owner. The owner will use k_i and k'_0 to determine the secret k'_i. It will use the secret k_{verify} to examine the integrity of the control block. The owner will return the encryption key and a new *cert* to $\mathscr{U}$ so that $\mathscr{U}$ can get D'_i from the service provider.

This method has several properties. First, we store all metadata in the control blocks on the service provider so that the data owner only needs to store two secrets k'_0 and k_{verify}. Second, since k_{verify} is known to only the owner, attackers cannot generate fake control blocks. Third, every time the data block D_i is updated, the value of x will be increased and the encryption key will be regenerated. At the same time, the control blocks of different data blocks will not be confused since their index numbers have a one-to-one mapping.

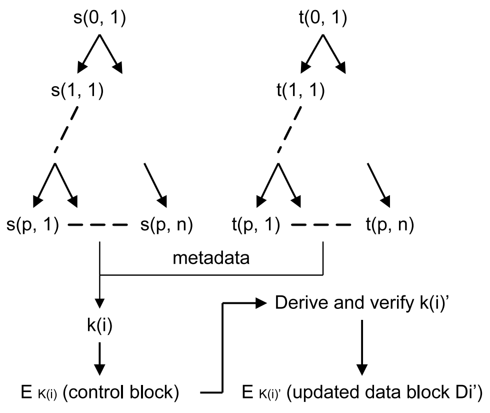


Figure 5: Handling updates to data blocks.

The costs of this method include another group of encryption/decryption operations and a second round of communication between the owner and the user. Although the blocks for the updated data have the index numbers from $(2^p + 1)$ to $(2^p + n)$, we have to clarify that we do not need to double the storage space on the service provider when only a fraction of blocks are updated. Instead, we can maintain a list of the updated blocks and their index numbers.

Block Insertion and Appending

The data owner may need to generate new information and put it on the service provider. Here we do not intentionally distinguish insertion from appending and follow the same procedure to handle the two operations. The data owner will locate an unused block index, generate the encryption key and derive the metadata, encrypt the data block, and store it on the service provider. One trick that can be adopted to improve the efficiency of future data access is to reserve some index numbers in the storage space. Later the data owner can insert the new data blocks into these holes based on their access patterns.

2.7 Analysis of Overhead: an Example

In this part, we use the application scenario described in Section 1 as an example to analyze the computational, storage, and communication overhead of a data access operation.

We assume that the size of the outsourced data is 10 PB and the data block size is 4 KB. Therefore, we have 2.5×10^{12} blocks. It is prohibitively expensive to store all encryption keys on the data owner. Using the method described in Section 4, we find that the height of the key hierarchy is $p = 42$. We assume the data that a user needs to access each time falls into a $4GB$ = 1,000,000 data block range. The access requests may read different fractions of the data. Since the authors of [28, 79] find that scientific databases have very infrequent data update operations, we assume that 0.1% of data blocks have been updated and they have control blocks on the server. Previous research [6, 65] shows that the number of consecutive data blocks that are accessed in scientific applications ranges from 100 to 30,000. Therefore, in our analysis we assume that on average 95% of the data is accessed in chunks of 1,000 consecutive data blocks. The remaining 5% of the accessed blocks distribute randomly and uniformly in the 4 *GB* data. We assume that two key derivation hierarchies as shown in Figure 4 have been established. We adopt 256-bit block encryption keys and 64-bit block indexes in our system. We also assume that the owner, the end users, and the server all use computers with 1-GHz CPU. Combining the information, we show the communication overhead of each party in Table 1.

Table 1: Overhead of the proposed approach.

computational overhead (in machine cycle)			
	owner $\mathscr{O}$	server $\mathscr{S}$	user $\mathscr{U}$
key derivation	$27M$	–	$720M$
one-time pad generation and over-encryption	–	$10G$	$10G$

communication overhead			
	owner $\mathscr{O}$	server $\mathscr{S}$	user $\mathscr{U}$
data blk index #	6KByte	–	10KByte
control blk	–	–	10.5KByte
keys per hierarchy	16KByte	–	–
updated data blk	–	1MByte	–

The table shows only the overhead of the proposed key management mechanism. It does not contain the transmission and decryption of the 1GB outsourced data since that overhead is independent of the adopted key management scheme.

Based on these assumptions, we can calculate the overhead of the data access operations. The computational overhead of the proposed approach comes from two aspects: key derivation using hash functions and over-encryption using a one-time key pad. When the data owner $\mathscr{O}$ receives the index numbers of the data blocks that $\mathscr{U}$ wants to access, it needs to derive the encryption keys from the roots of the hierarchies. It can then compare the two key sets and adopt the one with a smaller size. Since the hash function needs about

20 machine cycles to process one byte [75], we need 1440 machine cycles to accomplish key derivation in one level. For the generation of and encryption with the one-time key pad, previous research shows that algorithms such as ISAAC [82] need 19 machine cycles to generate 32 bits of pseudo random number. At the same time, exclusive-or operation is usually more efficient than random number generation.

Figure 6 illustrates the simulation results of the expected computation time at the owner, the user, and the service provider when 10% to 70% of the data blocks in 4 *GB* are accessed. We can see that the owner and the end user need only a few seconds of computation time to derive the block encryption keys even when 2.8 *GB* data is read. At the same time, the service provider and the user need less than 30 seconds of computation time for the XOR-based double encryption. All of the computation time can be easily hidden in the transmission time of the data.

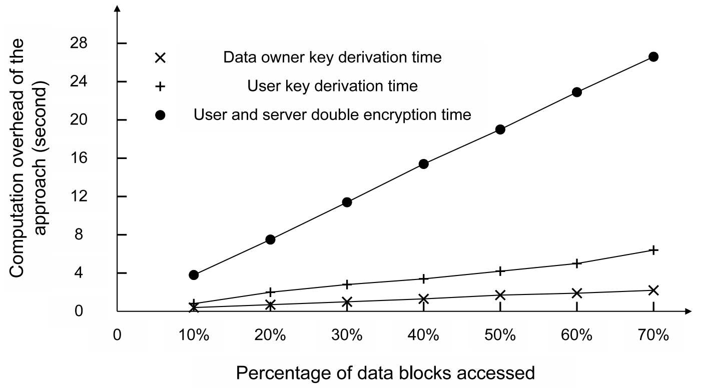


Figure 6: Computation time caused by the proposed approach.

For the communication overhead we focus on the number of key derivation keys that the owner needs to transmit to the end user, especially the reduction caused by the establishment of multiple hierarchies. We experiment with two scenarios of the 5% data blocks whose index numbers are not consecutive. In the first scenario, the blocks are chosen randomly and uniformly from the 4 *GB* data. In the second scenario, they are still chosen randomly but are restricted to the odd index numbers. In this way, they have a better chance to be the descendants of the same secret in the second key derivation hierarchy. We use this simple, yet biased access pattern, to demonstrate the potential advantages of multiple hierarchies. The simulation results are shown in Figure 7.

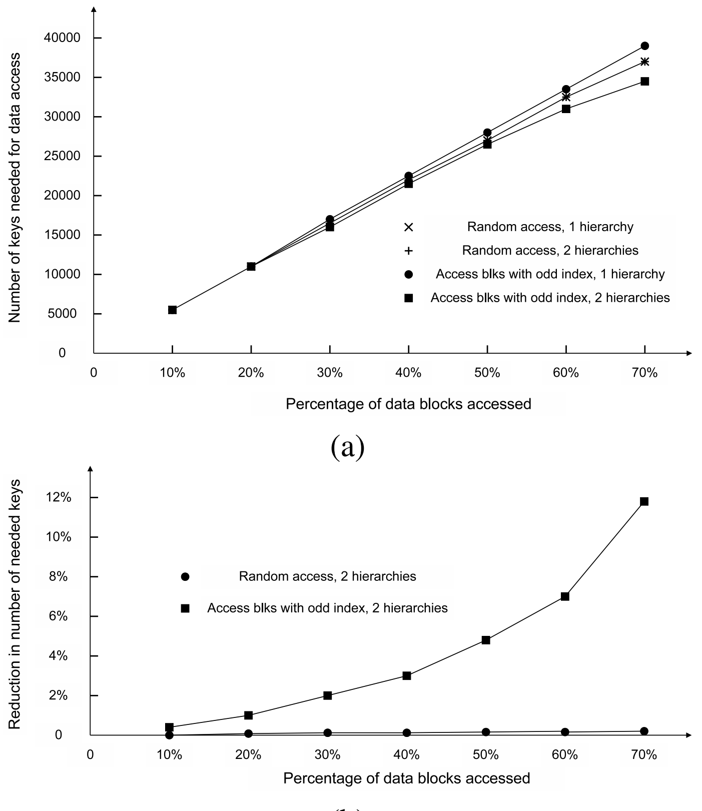


Figure 7: Communication overhead of the proposed approach.

In Figure 7.(a), we illustrate the number of keys that the owner needs to send out for different access requests. We find that when the user reads 2.8 *GB* data, the communication overhead for key distribution is about 40000 keys $\times$ 32 Bytes/key $\approx$ 1.3 *MB*. As a comparison, if every block encryption key is transmitted, the overhead will be 22.4 *MB*. Although the second hierarchy does not bring a lot of benefits to the random access scenario, the reduction in the odd-index only scenario becomes very obvious. Figure 7.(b) provides a better illustration of the results. For the odd-index only scenario, the second key derivation hierarchy saves 12% keys when 2.8 *GB* data is read. Please note that our simulation adopts a very simple access pattern for the key hierarchy establishment. When the data owner has a more accurate access pattern model, a larger reduction could be expected. At the same time, we can use the heuristic solutions to the set covering problem [16] to select keys from both hierarchies to further reduce the communication overhead.

The proposed approach introduces very limited storage overhead. The key derivation mechanism allows the owner $\mathscr{O}$ to store only the root keys of the hierarchies. The end user $\mathscr{U}$ does not need to pre-calculate and store all data block encryption keys. Quite the opposite, it can calculate the keys on the fly when it is conducting the data block decryption operations. The service provider $\mathscr{S}$ needs to store an extra copy of the updated data blocks. When the data update rate is very low in the applications, the extra storage overhead at $\mathscr{S}$ is also low compared to the size of the outsourced data.

One problem that we need to consider is the lengthened data retrieval delay caused by the access to the updated data blocks. We can shorten the response time from both the server and the data owner sides. At the server side, it has a one-to-one mapping among the index numbers of the original data blocks and those of the updated blocks. When the server receives a data access request to a control block, it can send the updated data block together with the control block to the user. This will not compromise the confidentiality of the information since the user still needs to get the new encryption key from the owner. At the data owner side, it can take advantage of the temporal locality of data access to shorten the response time. The owner can maintain a cache of the mapping between the index numbers of updated data blocks and their new encryption keys. In this way, when the owner receives an access request to such a block, it can directly send both the old and the new encryption keys back without waiting for the control block. This operation allows the server and the owner to deliver information of the updated blocks to end users more efficiently by avoiding another round of request/reply.

2.8 Discussion

In this section we discuss several problems of the proposed key management approach.

Comparison to CCS'05 Approach

Consider the difference between our approach and the mechanism proposed by Atallah *et al.* in ACM CCS'05. [8] First, we want to say that Atallah's paper provides a more generic approach to key management in access hierarchies. It has several features that our approach does not provide such as the support of downward and limited depth inheritance and the support of shortcut edges in hierarchies.

However, when we zoom into the application scenarios investigated in this paper, our approach makes special adjustments to adapt to their properties. For example, our approach handles user revocation differently from [8]. When a user's access right changes (but the data blocks do not change), we want to avoid operations to the storage service provider. We adopt two methods: over-encryption and lazy revocation, to achieve this goal. In [8], the authors suggest two schemes, both of which will immediately change the encryption keys and metadata of the impacted data blocks. These methods will cause extra communication and computational overhead for data reencryption to achieve consistent data confidentiality.

Security of the Approach

In previous sections we have described mechanisms to defend against eavesdroppers. Now, we investigate the safety of the proposed approach over collusive attacks and replay attacks of the control blocks.

In collusive attacks, two or more revoked end users may put their stored keys together and try to derive a secret that is not the descendant of any key known to them. Following the proof in [8], we can show that the adversaries have to have a non-negligible advantage in breaking the hash function to accomplish this task. Therefore, the proposed approach is robust against collusive attacks if the hash function is considered safe.

In Section 6.2, we show that an end user needs to send the control block of the updated data to the owner to get the new encryption key. Here a user may send an old control block to the owner to get the encryption key of a previous version of the data block. In this way, an adversary that can access only the current data block will get a copy of the old encryption key. Should it have kept a copy of the encrypted data block by eavesdropping on the network traffic, it will compromise the backward secrecy of the system. To defend against the replay attack, the data owner must verify the freshness of the control block. For example, when the service provider sends the control block to the end user, it can encrypt the user name, the request index, and the hash of the control block with the secret key between $\mathscr{O}$ and $\mathscr{S}$. In this way, the owner will be able to verify whether or not this control block is the latest version.

Extending to Multi-owner Outsourced Data

While in this chapter we consider only the simple case of storage outsourcing with a single data owner, the proposed approach can be extended to the scenarios in which the data has multiple owners and each of them can change data blocks independently. In this part, we discuss techniques to accomplish this extension.

To preserve data consistency, we should have orderly execution of the update operations when multiple owners want to change the data contents. This can be achieved through a semaphore flag at the service provider $\mathscr{S}$. This problem has been extensively studied in Operating Systems and distributed systems for access to shared resources.

The update operations to the multi-owner data are similar to the procedures described in Section 6. When a data owner wants to update a data block, it will store a control block to its original place. The control block will be protected by a secret key shared among the data owners. The control block contains a pointer to the new storage place, the information used by the owners to derive the new data encryption key, and a hash value of the whole block to protect its integrity. To prevent an attacker from sending an old control block or a control block for another data block to the owner, the service provider will link a control block to a specific data access request when the contents are sent back to the end user.

2.9 Conclusions

We propose a mechanism to achieve secure and efficient access to outsourced data in owner-write-users-read applications. We assume that the outsourced data has a very large size and we try to reduce the overhead at the data owner and service provider. We propose to encrypt every data block with a different key so that flexible cryptography-based access control can be achieved. Through the adoption of the key derivation method, the owner needs to maintain only a few secrets. We propose to store the metadata with the blocks so that multiple key derivation hierarchies can be established for different access patterns to reduce the processing overhead of the data access requests. Analysis shows that the key derivation procedure based on hash functions will introduce very limited overhead. We propose to use over-encryption and/or lazy revocation to prevent revoked users from getting access to updated data blocks. We design mechanisms to handle both updates to outsourced data and changes in user access rights. We investigate the computational, storage, and communication overhead of the approach through simulation. We also investigate the scalability and safety of the approach.

Extensions to our approach include the following aspects. First, we plan to design a new scheme for key management based on this approach so that it can be applied to many-write-many-read applications. Second, we want to design dynamic mapping functions among keys in the hierarchy and index numbers of data blocks so that we can progressively reorganize the data blocks based on their access patterns. In this way, we can further reduce the number of keys that the owner sends to the end user. Finally, we plan to integrate existing approaches to access control, provable data possession, and key management for outsourced data to develop a new approach to secure Storage-as-a-Service.

CHAPTER 3: DE-DUPLICATION VULNERABILITY AND DEFENSE

Cloud computing has at its core, virtualization. Without virtualization, cloud computing would be too expensive to be economically feasible. Virtualization, by definition, shares computing resources between different computing applications (such as between different objects within a program all the way to independent Operating Systems), thus greatly increasing the utilization of computing hardware. As with all cases of sharing, there can be violations of trust. Customers may trust that a cloud computing provider is doing everything possible to protect their computing environment from attack, but a malicious customer may find ways to violate the trust in this resource sharing to gain confidential insight into what other customer's virtual environments are working on. A malicious customer may also discover vulnerabilities in, or even gain complete access to, other customer's virtual machines.

One such attack example is described here, along with our mitigation strategies. In this example attack, a malicious customer takes advantage of the copy-on-write security protection mechanism designed to protect user integrity while allowing the performance advantages of memory de-duplication. The malicious customer uses this protection mechanism to facility an attack on confidentiality, more specifically, to launch a "guess-and-check" method of identifying the Operating Systems and data set contents stored in memory on other customer's virtual machines. Once a detailed look at this attack methodology is made, we demonstrate how other customers may protect themselves from this attack while still gaining the advantages of cloud computing with acceptable performance loss.

3.1 Introduction

OS fingerprinting is an essential step for many subsequent penetration attempts and attacks. Only after identifying the type and version of the OS of a system, can an attacker determine the vulnerabilities to exploit. Traditional OS fingerprinting schemes [9, 37, 42, 80, 93] are usually interactive procedures through IP packet analysis, service querying, or chronological exploits. Since these mechanisms usually initiate some interaction with the target system and use the contents and delay of the network packets from the target OS to determine its type and version, the target OS can monitor the network traffic to detect such attempts. It can also disable the responses or generate fake packets to disguise the fingerprinting procedure.

Additionally, with the ever increasing computation capabilities, storage space, and network bandwidth, more and more scientific and military projects are starting to use very large data sets for analysis, computation, and decision making. For example, NASA's Earth Observing System Data and Information System (EOSDIS) can generate two terabytes of data each day. Because of the intellectual property, user privacy, and several other reasons, many of these projects choose to hide the information about what data sets they are using for analysis and computation, even when some of the data sets are public. For example, a survey conducted in 2009 [61] shows that out of 62 groups that are requested to share their data or data sources, only 24 groups comply. To get access to such information, attackers have designed various mechanisms to compromise the operating system and database management system. Different schemes have been designed to defend against such attacks. Virtual machines in the cloud also make it so mobile users can defend themselves against traffic correlation attacks, as will be discussed in Chapter 4.

The proliferation of virtual machine platforms creates a new path for non-interactive OS and Data set fingerprinting. In a VM hypervisor, multiple virtual machines share the same hardware platform. Although perfect isolation among VMs is required by design [99], researchers have identified several mechanisms to break such isolation through the schemes such as side channels. [10] For example, researchers find that the shared cache may become a side channel for the detection of the web traffic access rate or even keystrokes of the co-resident VM instances. [81] As another example, CCCV [70] is a system that can create a covert channel using the CPU loads to secretly transmit information among virtual machines.

In this research we seek to investigate OS fingerprinting [73] and data set identification [72] in virtual machine hypervisors with the memory de-duplication functionalities enabled (such as VMware ESX and ESXi [101], Extended Xen [43, 110], and KSM (Kernel Samepage Merging) [7] of the Linux kernel). The memory de-duplication technique takes advantage of the similarity among memory pages so that only a single copy and multiple handles need to be preserved in the physical memory, as shown in Figure 8. Here each of the two virtual machines $VM1$ and $VM2$ needs to use three memory pages. Under the normal condition, six physical pages will be occupied by the VMs. If memory de-duplication is enabled, we need to store only one copy of multiple identical pages. Therefore, the two VMs can be fit into four physical pages (note that we have both inter- and intra-VM memory de-duplication). This technique can reduce the memory footprint size of VMs and the performance penalty caused by memory access miss. However, it will break the isolation among VMs and introduce new vulnerabilities of non-interactive OS and data set fingerprinting. The objective here is to exploit the vulnerability by constructing concrete attacks on the VMware ESXi hypervisor long side a widely used scientific visualization software package called ParaView [18], and investigate the mechanisms to defend against such attacks.

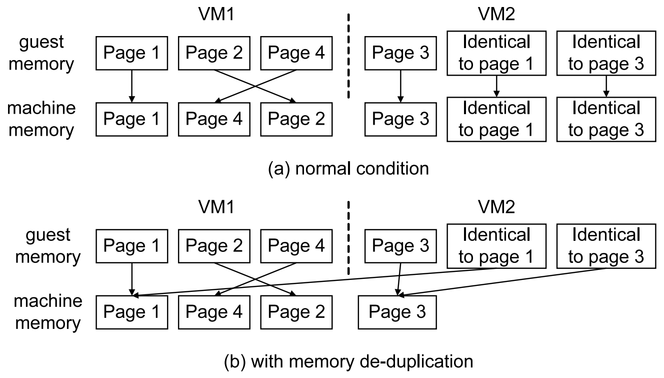


Figure 8: Memory de-duplication reduces the OS footprint size.

The overview of our approach is as follows. When we detect that the target VM has been launched, we will use the mechanisms described in [81] to initiate multiple VM instances using different OS onto the same physical box. Additionally, we will read multiple data sets into the memory of our VM. Without losing generality, for OS fingerprinting, we call the operating system of the target VM as S_t and those of the attacker's VMs as S_{a1}, S_{a2}, $\cdots$, S_{an}. The goal is to determine whether or not S_{ai} and S_t are of the same type. Without losing generality, for Data set fingerprinting, we assume that the target VM is using the data set D_t for analysis and computation, and the data sets we open are D_{a1}, D_{a2}, $\cdots$, D_{an}. The intent is to determine whether or not D_{ai} and D_t contain many identical pages. If so, we will conclude that the target VM is actually using D_{ai} for analysis, or running the same OS as S_{ai}. To achieve these goals, we will let the memory de-duplication mechanisms identify and merge those identical pages. Once this procedure is accomplished, we will introduce reading and writing operations to the memory pages that are unique for each different OS type or data set, respectfully. Since the hypervisor handles the operations differently for those de-duplicated pages and the pages with their own copies [90], we can measure the accumulated differences in the memory access delay to figure out whether or not S_{ai} and S_t are of the same type, or if D_{ai} and D_t are the same file. During this procedure, the attacker's VMs do not need to directly interact with the target VM.

We design several mechanisms to defend against OS Fingerprinting by the hypervisor and the guest OS respectively. At the virtual machine level, the guest OS can load the unique memory pages belonging to other OS into its memory to obfuscate the fingerprinting procedures. The hypervisor can monitor the behaviors of different VMs and avoid the de-duplication of any memory pages belonging to the OS image files. More details of the defense mechanisms will be discussed in Section 5.

The advantages of our approaches are as follows. First and most importantly, it is a non-interactive fingerprinting procedure since during the attack we only conduct operations on our own VM instances. This non-interactive property will prevent the target VM from detecting the identification operations. Second, our experiments show that many scientific data sets, even when they are from the same broad field, contain a large number of unique memory pages. Reading/Writing operations to these pages can generate a measurable difference in access delay. Third, we have analyzed the memory footprint of many different OS types and identified the memory pages that are unique to each type. Our preliminary results show that the number of unique pages is large enough to generate a measurable difference in access delay. Last but not least, both our experiment results on ParaView and our experiment results on VMware ESXi with both Windows and Linux systems show that our respective attacks are practical. Since our OS fingerprinting approach does not conflict with the interactive OS fingerprinting mechanisms, they can work together to improve the detection accuracy.

The remainder of the chapter is organized as follows. In section 2, we describe the details of the OS fingerprinting approach and our data set identification approach. We discuss the generation of OS and data signature files and the procedures to measure the accumulated differences in access delay. In section 3, we present the implementation of the attacks and the experimental results when VMware ESXi hypervisor is used. During section 4, we demonstrate our guest OS fingerprinting defense strategy with performance penalty and defense effectiveness results. Section 5 discusses the problems such as VM co-residence detection and other methods for the prevention of the attacks which were not experimented with. Section 6 concludes this chapter.

3.2 Attacks through Memory Deduplication

System Assumptions and Background

In the investigated scenario, we assume that an attacker can initiate VM instances in the same cloud infrastructure as the target VM. We also assume that through the co-residence detection mechanisms discussed in Section 5 we can determine whether or not the target system is running as a guest on the same physical box as the attacker's VMs. Since the target VM could have very sophisticated Intrusion Detection/Prevention Systems in place, it can detect any OS fingerprinting attempts through network interactions. Under this condition, the attacker wants to learn what OS version the target is, without any network scans, in order to exploit known vulnerabilities and conduct a direct one-hit attack before the IDS/IPS can respond. In a data set fingerprinting attack, we assume that the attacker can submit queries to the target VM to initiate data access for analysis and computation. The attacker also holds some data sets and it wants to determine whether or not the target VM is using one of these data sets to resolve the queries.

We assume the attacker has root control over the VM instances that it initiates. We also assume the attacker's VMs have large enough memory (such as 512MB) to avoid very frequent page swapping. We do not assume the attacker can decide how many CPU cycles it is allowed to use, nor do we assume the attacker can decide how much physical RAM its VMs are allowed to consume. These are reasonable assumptions based on current industry practice. Without losing generality, we assume that the host uses $4KB$ memory pages.

Since our experiments use VMware ESXi as the hypervisor, below we briefly describe its memory de-duplication operations. A comprehensive description can be found at [103]. To avoid unnecessary delay during page loading, whenever a new memory page is read from the hard disk, ESXi will allocate a new physical page for it. Later, ESXi will use idle CPU cycles to locate the identical memory pages in physical RAM, and remove duplicates by leaving pointers for each VM to access the same memory block. Hash results of the memory page contents are used as index values to locate identical pages. To avoid false de-duplication caused by hash collisions, a byte-by-byte comparison between the pages will be conducted. While the reading operations to the de-duplicated pages will access the same copy, copy-on-write is used to prevent one VM from changing another VM's memory pages. Specifically, on writing operations a new page will first be allocated and copied. This procedure will incur extra overhead compared to writing to not-shared pages, which will lead to a measurable delay when a large number of shared pages are allocated and copied.

ESXi uses three system wide parameters to adjust how it looks for de-duplicated pages. These are “ShareScanTime”, “ShareScanGHz”, and “ShareRateMax”. ShareScanTime specifies how much time the administrator would like ESXi to scan an entire VM’s memory pages for duplicates. ShareScanGHz specifies the maximum number of pages to scan in physical RAM per second. ShareRateMax specifies how quickly the pages should be scanned per-virtual machine. VMware sets these parameters to some default values to offer the least amount of overhead, while still finding identical pages fairly quickly. According to VMware, the algorithm also scans faster if it determines that there is a high likelihood of finding duplicate pages based on the previous scans, and vice-versa.

ParaView [54] is an open-source, multi-platform data analysis and visualization software package. ParaView allows users to generate visualizations to analyze their data using qualitative and quantitative techniques. ParaView is used by the organizations such as Army Research Laboratory, Sandia and Los Alamos National Laboratories, and NASA.

Generation of Memory Fingerprints

The fingerprints created in the following fashion give us a real world representation to what can be found in the wild. The off-line memory dump and analysis described in the following two sections is also much faster, easier, and only slightly less accurate than calculating what the similar memory pages would be based on the OS’s documented loading behavior on essential OS files, and the documented loading behavior of the data analysis software and the data files themselves, respectively. Based on these considerations, we believe that an attacker would most likely build memory signatures in the same fashion as we have for our approach.

Generation of OS Signatures

The first step to turn the proposed approach into a practical attack is to identify the memory pages that are unique to each OS type. To accomplish this task, we load the OS image files into a VM instance and then conduct a memory dump. The dump files are then cut into 4*KB* pages. We adopt the mechanism in [43] and use hash results of the memory contents as indexes to locate the identical pages. For each OS type and version, we analyze the memory dump files from different installation sources so that we can remove the impacts of the factors including different hardware drivers, different product keys, and different product IDs. Once we have categorized the memory pages by OS types and versions, we can find out which memory pages are unique to each OS version, but present in all copies of that OS version. These memory pages will hereafter be referred to as *OS signatures*. Please note that not all memory pages can be used as the signatures. For example, the memory dump of Windows XP SP3 contains 59,238 copies of all-zero memory pages. These pages, however, cannot be used for OS fingerprinting since they are not unique to any specific OS type.

During the signature generation procedures, we conduct cross-comparison among only the memory pages of different OS types. Therefore, it is possible that some of the signature pages will appear in the memory images of other software applications or data files. This may affect the OS fingerprinting accuracy since the attacker would not be able to identify the sources of these memory pages. Fortunately, because of the large number of diverse memory pages ($2^{4096\times 8}$ if every bit has the same probability to be 0 or 1), the impacts on fingerprinting accuracy will be very limited.

Generation of Data File Signatures

Considering the size of the data sets for scientific and military applications, their memory footprint often contains many pages. Some of these pages, however, cannot be used for data set identification since they are not unique for any specific file. For example, the header of the data files often follows a pre-defined format and contains information such as the dimension of the data, the number of records in the file, and the size of each data record. At the same time, the data files may also have some identical memory pages with the operating system or user applications. To determine whether or not a data file has enough unique memory pages, we propose to conduct off-line memory dumps of computers after they have loaded the file. The dump file is then cut into $4KB$ pages and compared to the memory pages of different operating systems and user applications. We adopt the mechanism in [43] and use hash results of the memory contents as indexes to locate the identical pages. Once we have categorized the memory pages of the data sets, we can find out which memory pages are unique to each data file. These coalesced memory pages will hereafter be referred to as *data file signatures*.

Fingerprinting Procedures

As we describe in Section 1, VMware ESXi uses different methods to handle the writing operations to the de-duplicated pages and pages with their own copies. For the pages with their own copies, the writing operation can be conducted immediately. For the de-duplicated pages, a new copy must be created first. This memory allocation and copy procedure will introduce extra processing delay. Our OS fingerprinting procedure, therefore, is to detect the accumulated difference in access delay caused by the de-duplication between our OS signatures and the target VM. To achieve the goal, we adopt the following schemes.

First, since we want to measure the extra processing delay caused by the writing operations to the de-duplicated pages of the data sets or operating systems, we need to control at what time and to which pages such writing operations will be initiated. Fortunately, for many data analysis and visualization applications such as ParaView, the data sets are treated as read-only raw-data inputs. Likewise, since many OS files are read-only, we plan to construct a different signature file for each OS and data set by chaining its unique memory pages together. Therefore, by loading the corresponding signature file into memory and writing to it, we can force the hypervisor to create a new copy for any de-duplicated pages that uniquely belong to that OS or data set.

Second, since the pages that have not been accessed recently will be swapped out by the hypervisor, we need to distinguish the delay of hard disk reading from that of copy-on-write. To accomplish this task, we plan to conduct a reading operation to the OS or data set signature file right before the writing operation. If a page is in the memory, this reading operation can be accomplished immediately and it will not change the de-duplication status of the page. However, if a page has been swapped out, this reading operation will force the hypervisor to execute a hard disk access. Since VMware ESXi will allocate a new memory page for the newly read contents, the next step of writing can be accomplished immediately and will not provide us useful information about the fingerprinting. In this way, we can distinguish between the two types of access delay.

With these basic components established, our OS fingerprinting procedure is illustrated in Figure 9. Although here we use the narrative description “short” and “long”, the experiments in Section 3 will help us to determine quantitative thresholds for OS and data set fingerprinting in real systems.

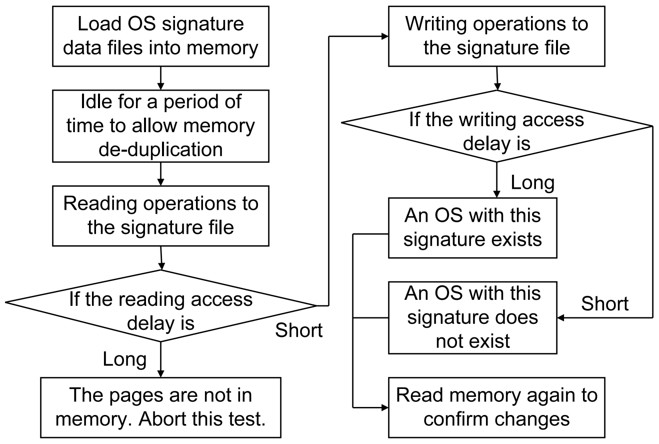

Load OS signature data files into memory
Idle for a period of time to allow memory de-duplication
Reading operations to the signature file
If the reading access delay is
Long
Short
The pages are not in memory. Abort this test.
Writing operations to the signature file
If the writing access delay is
Long
Short
An OS with this signature exists
An OS with this signature does not exist
Read memory again to confirm changes

Figure 9: The proposed fingerprinting procedure.

When we confirm that our VM instance and the target VM are located on the same physical box through co-residence detection, we will read the signature files into memory. These files will then be left alone for a period of time to allow memory de-duplication algorithms to locate and merge the identical pages. Once the de-duplication procedure is accomplished, we will conduct a reading operation on the signature files. The purpose of this operation is to determine whether or not the files have been swapped out to hard disk. If the reading access delay matches the hard disk loading time, we will abort the fingerprinting procedure since the newly loaded pages all have their own copies. Otherwise, we will conduct a writing operation to the signature files. Since we already know that these pages are in memory, based on the delay of the writing operation, we can determine whether or not they experience the copy-on-write procedures. If so, we know that a VM instance matching this signature file exists in the physical box.

3.3 Implementation and Experimental Results

Although the basic idea of the proposed approach is straightforward, many issues need to be solved before we can turn the idea into a practical attack. For example, we need to examine the size of the OS and data set signature files to make sure that the accumulated delay is actually measurable. At the same time, we need to examine the timekeeping schemes in hypervisors so that we can measure the delay accurately. In this Section, we present the details of our implementation of the attack and the experiment results.

OS Signature Experiment Environment Setup

Our VMware ESXi server is running on a PC with a dual core 2.4GHz Xeon CPU, 4GB RAM, and SATA hard drives. To simplify the experiment setup and examine the practicability of the attack, during each trial there are only the target VM and our attacking VM instance running on ESXi. OS fingerprinting in more complicated scenarios will be studied in future work.

In order to build the OS signatures, we have generated and examined the memory dump files of different operating systems to find memory pages that would be unique for a specific OS version. We sampled four types of Linux/Unix and nine types of Windows to cross examine their memory pages. The number of unique pages of each OS type/version is summarized in Table 2. From the table, we find that the size of the signature files ranges from several thousands to tens of thousands of pages. The experiment results presented later will show that the accumulated difference in access delay to these pages can be easily detected.

Another issue that we are facing is the accuracy of time measurement. Traditionally a computer provides three schemes to measure the length of a time duration: time of the day, CPU cycle counter, and APIC (advanced programmable interrupt controller) timer. The first method provides the measurement granularity of seconds which is too coarse for our application. The second method will be a good candidate for time measurement if the attacker's OS completely owns the hardware platform. In a VM-based system, however, it cannot accurately measure the time duration. For example, if a page fault happens during our reading operation, the hypervisor may pause the CPU cycle counter while it fetches the memory page. Therefore, the delay caused by hard disk reading will not be measured. Based on these observations, we choose to use the timestamp service provided by masm32 in: \masm32\lib\winmm.lib to access the APIC timer. Specifically, we use the timeGetTime directive because it provides a 1 millisecond resolution. [64] According to [102], VMware has fully emulated the local APIC timer to provide accurate time readings, so the page fault handler built into ESXi to handle de-duplication will not pause the virtual local APIC timer.

Table 2: Size of OS signature files in 4*KB* pages

OS Type	Fedora		Ubuntu	
	7	8	10	11
# of unique pages	4572	5274	8179	5547

OS Type	Windows								
	XP SP2	XP SP3	2003	Vista 32 bit	Vista 64 bit	Win 7 32 bit	Win 7 64 bit	2008 32 bit	2008 64 bit
# unique	3325	3582	3695	5503	20053	9753	23977	5638	16240

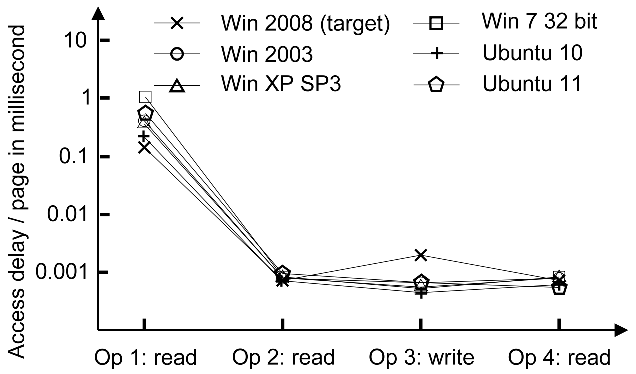
Three reasons make us choose Windows 95 as the OS of the attacker's VM instance. First, we want an operating system with a very small memory footprint size so that its contents will not pollute the de-duplication results. Since Windows 95 can easily run on a platform with only 64*MB* RAM, it achieves a good balance between the size and the supported functions. Second, there are not many systems still running Windows 95 so there is a very low probability that our VM instance and the target VM are running the same OS. Last but not least, we have extended experiences in working with the portable executable (PE) file format and Windows 95 is one of the earliest systems supporting this format. To further reduce extra delay caused by high level programming languages, we implemented the memory access and time measurement functions in assembly language.

As illustrated in Figure 9, we have divided the OS fingerprinting procedure into four groups of operations. Operation group one will read the first 32 bits of every memory page of the OS signature file and store each result into the EAX register. Our program then sleeps the processor for a duration of several hours to allow de-duplication to occur. After that, it will perform operation groups two through four immediately after each other. Group two does the same operation as group one. Group three writes junk data into the first 32 bits of every memory page for each OS signature. Finally, group four reads back the memory pages to confirm the changes.

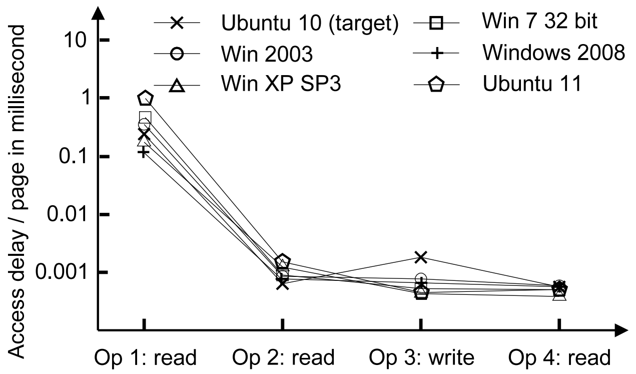
OS Signature Experiment Results

We conduct six groups of experiments to evaluate the OS fingerprinting capability of the proposed approach under different levels of computation and memory access workload. The target OS that we try to identify includes Windows 2008 Server 32 bit, Windows 7 32 bit, and Ubuntu 10. In the attacker's VM instance, we load the OS signature files of Windows 2008 Server 32 bit, Windows 2003 Server, Windows XP Service Pack 3, Windows 7 32 bit, Ubuntu 10, and Ubuntu 11. We choose these signatures since they are the favorites of IT staff in our day-to-day lives and would have a high probability of a hit in the real world. The idle periods for memory de-duplication are usually four hours. Our results are shown in Figures 10 through 14. Given the fact that the delays span across multiple degrees of magnitude, we use log-scale Y-axis. Seeing as the signature files of different OS types have different sizes, we illustrate the average reading and writing time per memory page in the figures. For a signature file that contains thousands of memory pages, the accumulated difference in access delay will be several to tens of milliseconds, which can be easily measured by the proposed approach. Each node in the figures is the average value of five experiment runs with the same configuration. All time delays are measured in milliseconds. To help readers to better understand the experiment results, the average access delay to the signature file of the target OS will always be represented by “x”.

In the first experiment, we set up a baseline test case. Here one instance of Windows 2008 is initiated in the host as the target OS. To reduce the impacts of page swapping on the approach and accelerate the de-duplication calculation procedure, the target VM instance does not activate any other applications. The attacker’s VM runs Windows 95 and loads the signature files of Windows 2008, 2003, XP SP3, Windows 7 32bit, Ubuntu 10, and Ubuntu 11 into its memory. As shown in Figure 10.(a), the delay of the first reading operations is relatively long since the pages have to be read from the hard disk. After that, the target VM instance and the attacker’s VM are left idle for four hours to give the de-duplication algorithms enough time to scan the memory. Given that each VM has enough memory to hold the OS files, we do not expect a lot of page swapping to happen. This is confirmed by the very short access delay of the second group of reading operations. The access delay of the writing operations, however, demonstrates the difference among the signature files. Here the access delay to the signature file of Windows 2008 is about three times longer than those of other OS types because of the copy-on-write operations. Our approach can successfully identify the type of the target OS in this baseline setup. The second experiment has the same configuration as experiment one except that the target VM is running Ubuntu 10. As the results shown in Figure 10.(b), the long delay will allow us to easily identify the target OS.



(a) when Windows 2008 is the target OS



(b) when Ubuntu 10 is the target OS

Figure 10: OS fingerprinting results when CPU and memory demands are low.

In the third experiment, we want to investigate the impacts of computation workload in the target VM on the fingerprinting accuracy. Here the basic setup is the same as the first experiment. The only difference is that we use Windows 7 32bit as the target OS. To introduce medium-level computation workload on the target VM, we run DES encryption and RSA encryption algorithms on the VM. The results are shown in Figure 11. Here the second group of reading operations become slower. This could be caused by context switch between the VMs since the target VM is running some applications. The writing delay to the signature files of the target OS is still much longer than those of other OS types (4 to 8 times longer). From this figure, we find that our OS fingerprinting approach will work properly when the computation workload on the target VM is not too heavy.

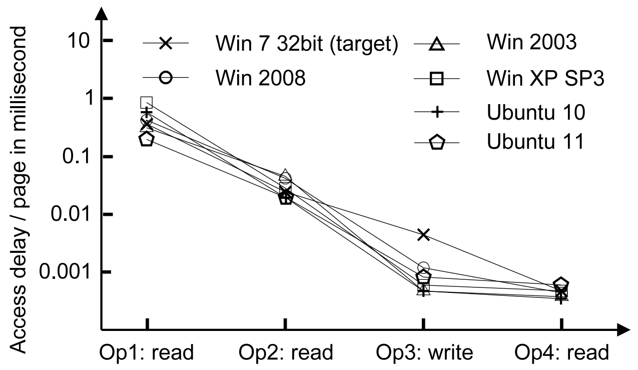


Figure 11: OS fingerprinting under medium computation workload.

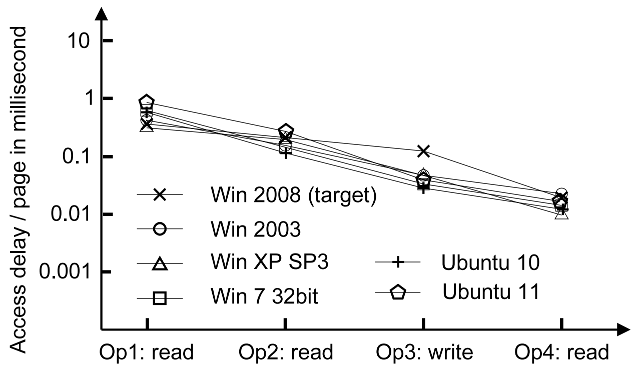


Figure 12: OS fingerprinting under medium level memory demand.

In the fourth experiment, we want to investigate the impacts of memory demands on the target VM on the fingerprinting accuracy. Again we choose Windows 2008 as the OS of the target VM. We run a memory testing software "QA+Win32" [35] to introduce medium-level memory demand on the target VM. The results are shown in Figure 12. Here the second group of reading operations become even slower because of the page swapping. The write delay to the signature files of the target OS is still 2 to 3 times longer than those of other OS types. This figure shows that our approach can work properly under medium level memory demand on the target VM.

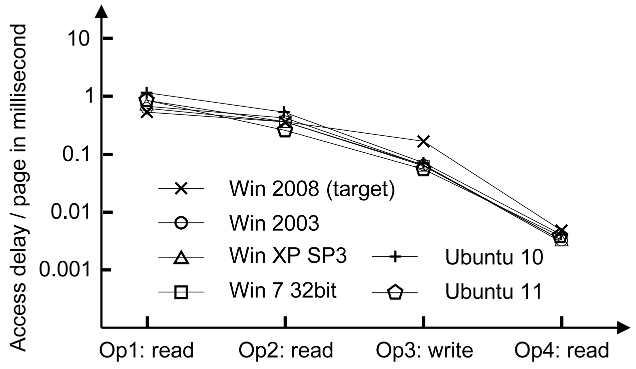


Figure 13: OS fingerprinting under medium workload and memory demand.

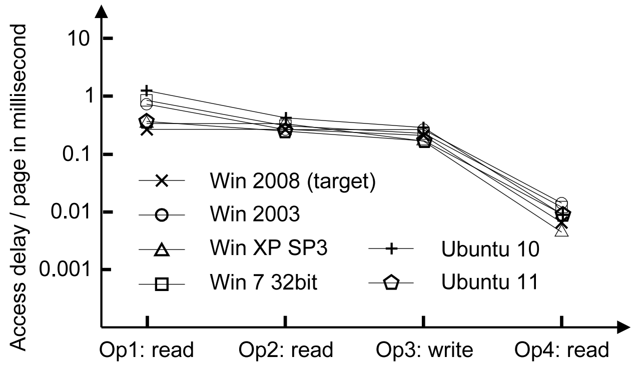


Figure 14: OS fingerprinting under extreme workload and memory demand.

In the fifth experiment, we have medium-level computation workload and memory demand on the target VM. The results are shown in Figure 13. The OS type of the target VM can still be identified based on the slow speed of the writing operations.

In the last group of experiments, we want to investigate the OS fingerprinting accuracy of our approach under very heavy computation workload and memory demand. We use Windows 2008 as the OS of the target VM. The target VM is running Prime95 [106], a CPU and RAM stress test software package. Very frequent memory page swapping is expected. The results are shown in Figure 14. We can see that the writing delay to the signature files of different OS types cannot be distinguished from each other. This experiment shows that memory de-duplication based OS fingerprinting will not work properly under extremely heavy computation workload and memory demand. This is reasonable since under this condition, the hypervisor would not have enough computation power or a relatively stable memory image to identify and maintain the de-duplicated pages.

Data Set Experiment Environment Setup

Our VMWare ESXi server is running on a PC with a dual core 2.4GHz Xeon CPU, 4GB RAM, and SATA hard drives. We have chosen two Windows Operating Systems as the guest OSes for the target VM: Windows 7 32-bit, and Windows XP SP3. The attackers VM is Windows 95 because it has a smaller memory footprint than modern Windows Operating Systems. Here, having a small memory footprint prevents our attackers' VM from requiring excessive amounts of memory for loading memory page samples. We have chosen five biological data sets available from 3D-IRCADb [34] to serve as the data files for fingerprinting and identification. A screenshot of an example data set is available in Figure 15. We choose the five data sets available that have a signature of over 3,000 memory pages. Their basic information is shown in Table 3.

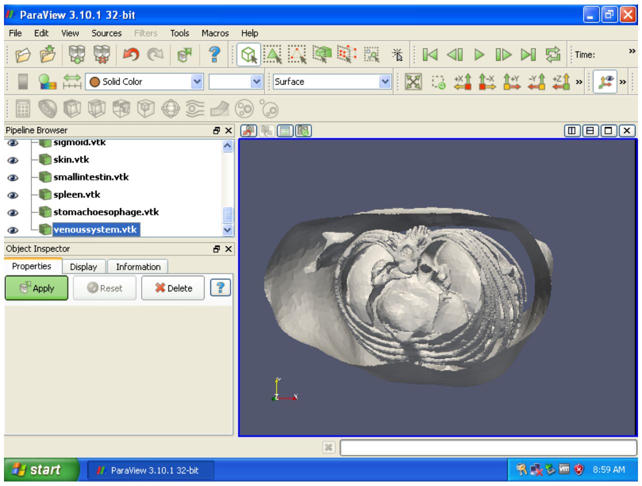


Figure 15: Screenshot of 3D-IRCADb2.2, as used in experimentation.

Table 3: Selected data sets.

file & size	content	source	signature size
3D-IRCADb1.1 (6.9MB)	liver tumor structure	3D-IRCADb database	3524 pages
3D-IRCADb1.3 (13.4MB)	liver tumor structure	3D-IRCADb database	7836 pages
3D-IRCADb1.5 (7.1MB)	liver tumor structure	3D-IRCADb database	3640 pages
3D-IRCADb1.6 (7.2MB)	liver tumor structure	3D-IRCADb database	3661 pages
3D-IRCADb2.2 (13.1MB)	Chest/Abdomen 3D CT-scan	3D-IRCADb database	7435 pages

In order to build our data set signatures, we have generated and examined the memory dump files under different scenarios. Specifically, we need to examine the memory page contents under two conditions: (1) ParaView is initiated but no data file is opened; and (2) ParaView has opened the data file and generated the visualization. We experiment with different types of guest OS to investigate their impacts on the size of the signatures. The number of unique pages of each data set is summarized in Table 3. Please note that in this table, we have cross-compared all the memory pages of the two guest operating systems, the memory pages of ParaView opened under different guest OS, and the pages of the data sets. From the table, we find that the signature files usually have the size of several thousands pages. The later experiment results will show that the accumulated delay of accessing these pages can be easily detected. At the same time, we find that when ParaView opens the same data file in different guest OSes, the memory dumps have many duplicate pages. This shows that ParaView is almost OS-independent. This property will definitely promote its wide adoption. However, it will also provide convenience to attackers in data set identification since the malicious parties do not need to first figure out the guest OS of the target virtual machine.

Another issue that we are facing is the accuracy of time measurement. Traditionally an operating system provides three methods to measure the length of a time duration: time of the day, CPU cycle counter, and APIC timer. The first method provides the granularity of seconds and it is too coarse for our application. The second method will be a good candidate for time measurement if the OS completely owns the hardware platform. In a VM-based system, however, it cannot accurately measure the time duration. For example, if a page fault happens during our reading operation, the hypervisor will pause the CPU cycle counter and switch to another VM. Therefore, the delay caused by hard disk reading will not be measured. Based on these observations, we choose to use the timestamp service provided by masm32 in winmm.lib to access the APIC timer. Specifically, we use the timeGetTime directive because it provides a 1 millisecond resolution. According to [102], VMWare has fully emulated the local APIC timer to provide accurate time readings, so the page fault handler built into ESXi to handle de-duplication will not pause the virtual local APIC timer. To further reduce extra delay caused by high level programming languages such as Java, we implemented the memory access and time measurement functions in assembly.

As illustrated in Figure 9, we have divided our program's functions into four groups of operations. The operation group one is to immediately read the first 32 bits of each data set signature's memory page and store the result of each read operation in the accumulator (EAX). Our program then sleeps the processor for a sufficient amount of time to allow de-duplication to occur. Our program then performs operation groups two through four immediately after each other. Group two does the same operation as group one, but reads the signatures after the sleep period. Group three writes junk data to the first 32 bits of every memory page for each data set signature. Group four then reads back the memory pages to confirm the changes.

Data Set Experiment Results

We conduct four groups of experiments to evaluate the data set identification capability of the proposed approach under different levels of computation and memory access workload. We have two virtual machines running on the physical box. The operating system of the target virtual machine is Windows XP SP3. It has the latest version of ParaView installed. The attacker's VM uses Windows 95 as the operating system. The target VM has 512 *MB* memory and the attacker's VM has 256 *MB* memory. Because we have constructed the data set signature files in the binary format, we do not need to install ParaView on the attacker's VM. This is preferred, because writing junk data to memory pages in actual use by ParaView may cause the program to crash, which may be a detectable event on the hypervisor. We assume that the attacker has a rough idea of what sample data sets might be used by the target VM. In our experiments, since we have direct access to the target VM, we will activate ParaView to operate on a single selected data set at a time. Given that the attackers do not know which data sets the target VM is using, we will load the signature files of all data sets into memory and conduct the reading/writing operations on each of them.

Our results are shown in Figures 16 through 19. In addition to our 4 groups of operations, we first show in each figure the hard disk delay required to load each signature from disk. As the access delays span across multiple degrees of magnitude, we use log-scale Y-axis. Because the signature files of different data sets have different sizes, we illustrate the average reading and writing time per memory page in the figures. Each node in the figures are the average value of five experiment runs with the same configuration. All time delays are measured in milliseconds. To help readers better understand the identification results, the page access delay of the data set read by ParaView on the target VM is always represented by “x”.

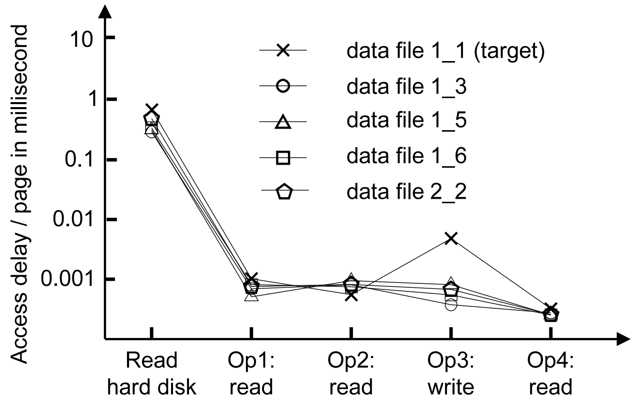


Figure 16: Results with 3D-IRCADb1.1 under idle level computation workload.

In the first experiment, we set up a baseline test case. Here the ParaView software on the target VM reads the 3D-IRCADb1.1 data set and generates the visualization. We choose this data set as the test case since it contains the smallest number of signature pages that still fit our 3000 page count lower limit. As shown in Figure 16, the hard disk delay is long. Next, the first read operation is short because the pages have just been freshly loaded into memory. After that, the target VM and the attacker’s VM are left idle to give the de-duplication algorithms enough time to scan the memory. Since each VM has enough memory to store the signature files, we do not expect a lot of page swapping to happen. This is confirmed by the very short access delay of the second group of reading operations.

The access delay of the writing operations, however, demonstrates the difference among the signature files. Here the delay of the signature file of the 3D-IRCADb1.1 data set is about twelve times longer than those of other data sets because of the copy-on-write operations. Our approach can successfully identify the data set in use in this baseline setup.

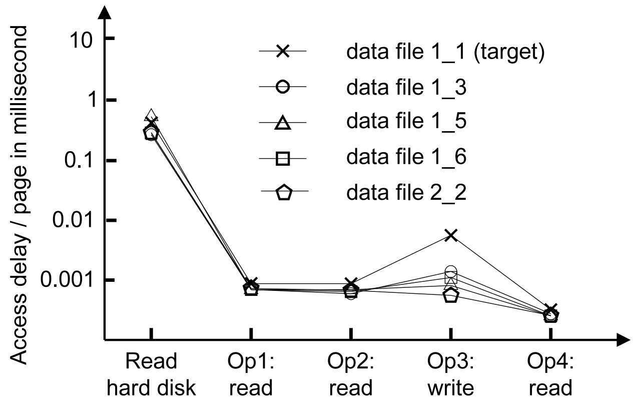


Figure 17: Results with 3D-IRCADb1.1 under moderate computation workload.

For the second experiment we continued with the 3D-IRCADb1.1 data set. This time, we also startup a Windows 7 VM and allow it to run along side the target and attacker VMs. We repeatedly ran the System File Checker (SFC) included with Windows 7 to simulate a normal moderate computer load by a third party Virtual Machine running on the hypervisor. The physical machine maintained fluctuating demands on the CPU, memory, and hard disk through the entire test. The results are shown in Figure 17. From the figure, we find that the reading and writing delays are very similar to Figure 16. The measured writing delay is still much longer than those of other data sets (4 to 6 times longer). From this figure, we find that the proposed approach will work properly when the signature file is as small as three thousand pages under normal real-world operating conditions.

In the third experiment, we want to assess what the impacts are of using a data set with a larger signature, so we used the 3D-IRCADb1.3 data set. To introduce the impacts of

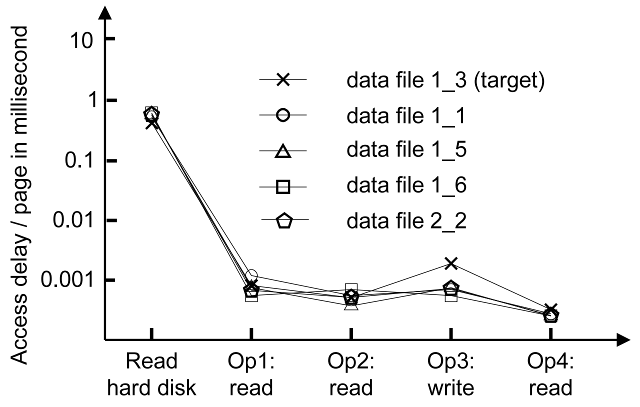


Figure 18: Results with 3D-IRCADb1.3 under moderate computation workload.

normal CPU, memory and disk usage operations, we again run the SFC included with Windows 7 in a loop during the experiment. As we describe above, the ParaView software on the target VM reads the 3D-IRCADb1.3 data set. After the visualization is generated, we leave the ParaView application alone to avoid extra computation and memory access overhead caused by the software. The results are shown in Figure 18. The write delay measured is still 2 times longer than those of other data sets. This figure shows that our approach can work properly with larger signature data sets under normal operating conditions.

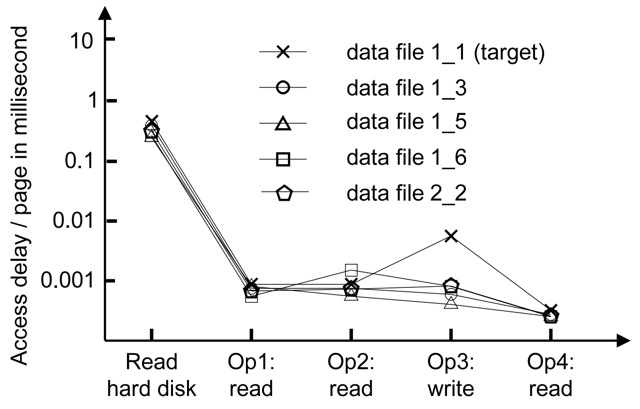


Figure 19: 3D-IRCADb1.1 with high level workload and memory demand.

In the fourth experiment, we write a script to continually change the viewing angle on the target VM. The view point rotated between each viewing axis and back again within one second, causing the virtual CPU to peak out at 100 percent usage, with each physical core maintaining about 50 percent usage. We use the 3D-IRCADb1.1 data set to see if a smaller signature will have a noticeable access delay. The results are shown in Figure 19. The identification of the data set in use in the target VM can still be identified with a 10 times longer writing delay.

3.4 Guest Based Prevention of Memory Deduplication Fingerprinting

As the analysis in Section 2 shows, the differences in access delay will allow attackers to detect only the existence of specific pages in the main memory. However, they cannot identify to which applications or data files these pages belong. Therefore, a VM can obfuscate the attack by intentionally loading the signature files of other OS types into its memory. Since most signature files contain only a few thousand unique memory pages, we can easily fit multiple signatures into the main memory of a VM.

Experiment Environment Setup and Results

We set up an experiment environment identical to Section 3.1. We must first determine a proper signature read time frequency for the defense program to use, in order to prevent the target OS from swapping out signature files, subsequently preventing de-duplication from masking the target's signature. In order to do this, we used three different read frequencies: 2000 seconds, 200 seconds, and 20 seconds. Our attacker again used Windows 95 as the attacking OS, and our target was chosen as Windows 7 because it is the most frequently used OS at this time. On the target, we introduced a moderately high memory usage by opening different retail web pages in Internet Explorer until there was no free memory and the disk cache was down to 300MB. We then refreshed each web page in serial 10 second loops. We noticed during our experiment that the CPU usage chart became highly unstable, but should not interfere much with this experiment. Our attack results are shown in Figure 20.

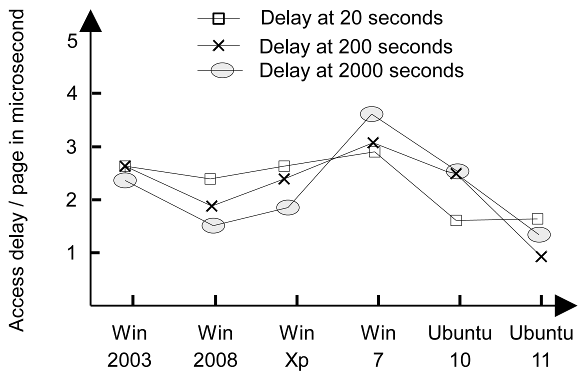


Figure 20: Defense results under different orders of magnitude.

These results show an inverse correlation between signature read frequency and attacker detection accuracy. This is the be expected because the target OS will have a better chance of swapping signature pages out with longer signature read wait times, thus not de-duplicating as many signature pages and increasing the attacker's accuracy. In order to avoid this problem, our defense experiments and performance analysis will be done with a 2 second read wait time.

For our defense vs. no defense comparison experiments, our attacker again used Windows 95 as the attacking OS, and our target was Windows XP Service Pack 3. We chose Windows XP because it is easier to more precisely control cpu and memory variables to alleviate the unstable cpu usage seen in Windows 7. We conduct two groups of experiments to evaluate the OS fingerprinting defense capability of the proposed approach under different levels of computation and memory access workload. In the attacker's VM instance, we load the OS signature files of Windows 2008 Server 32 bit, Windows 2003 Server, Windows XP Service Pack 3, Windows 7 32 bit, Ubuntu 10, and Ubuntu 11. The attacker wrote these signature files into their virtual machine's memory and waited five hours before trying to write to these pages. The target wrote these same signature files into their memory and read them every two seconds to make sure they were not swapped out by either the guest or the hypervisor. An experiment was not conducted for both high CPU and High Memory demand because, as shown in Figure 14, the attack does not work under these conditions anyways.

In experiment group one, the memory signature write times were taken by the attacker with both no defense and the defense program running in the target VM. The target VM was left idle, other than the defense program running. The results are shown in Figure 21. As is shown, before the defense, the attacker could successfully identify the operating system version running on the target VM. However, with the defense program running, the attacker is now no longer able to successfully detect the target VM's Operating System.

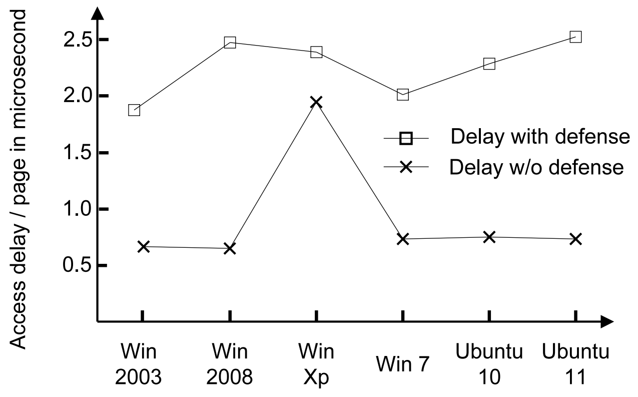


Figure 21: Page write results with and without defense under idle workload.

In experiment group two, the memory signature write times were taken by the attacker with both no defense and the defense program running in the target VM. The target VM was running Prime95, instructed to use very little ram or disk. The CPU in the virtual machine stayed very near 100 percent usage the entire wait time. The results are presented in Figure 22. Here, we can see that the memory write time for XP decreased without the defense program running from Figure 21 to Figure 22. We believe this was due to a slight decrease in the efficiency of the de-duplication engine in the hypervisor due to the large CPU workload. With the defense program running, the write times for all signatures went up as expected, with a slight dip in the middle, again, most likely due to a decrease in the de-duplication efficiency. Despite this decrease, the defense program does successfully mask the correct operating system signature.

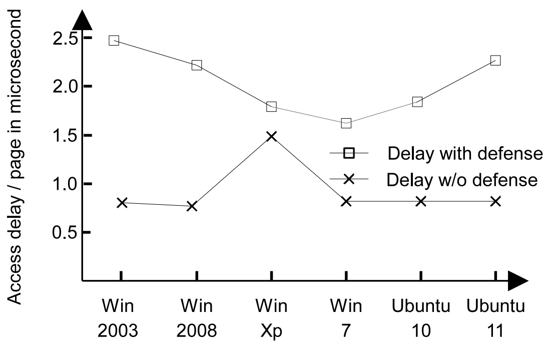


Figure 22: Write results with/without defense; high CPU/low memory usage.

Performance analysis of VM with defense program running.

In order to assess the performance impact of our defense program, we ran our defense program on a Microsoft SQL server configured Windows 7 guest VM and tested the memory performance when assigned 1.5 GB of memory. Windows 7 was chosen because it has the highest idle memory demands of all the other OS signatures we made. To evaluate a non-idle SQL server, we ran SQL stress tests on the server at the same time. These stress tests simulated ten users submitting typical queries such as ”SELECT TOP 1024,” ”INSERT” and ”DELETE.” We conducted our test with first without any defenses, second with our defense programming running and no antivirus, and third without our defense program and Avast! version 8.0.1483. We included the results of the SQL performance test with antivirus running to compare with a known user acceptable performance penalty. Our results are shown in Table 4.

Table 4: Averages of SQL Memory Performance Measurements.

	Before Defense or Avast!	After Defense	After Avast!
Cache Faults/sec	0	0.2105	1.6549
Page Faults / sec	63.0820	66.4911	102.6774
Page Reads/sec	0.0294	0.01754	0.0909
Pool Paged Bytes	85449089	84826606	102458806
% Committed Bytes In Use	79.6416	92.1890	82.01443

As is shown from these results, there is a slight performance penalty, but all the values are negligible except for "Percent Committed Bytes In Use." This value simply represents the percentage of memory in use by all processes. This value goes up a reasonable amount when containing six OS signatures, so we feel it proves to be of an reasonable performance impact in comparison to the security offered. An attack on an operating system that is successful without any detection is the worst nightmare of a security professional. Our defense program provides less of a performance impact than many signature based antivirus products, which have become a staple in the defense arsenal for any computer system. The programming of our defense program in assembly made it have no noticeable impact on the processor, and only a slight penalty to the OS memory management. We feel defense programs written in this way could easily become standard practice for security conscious virtual machine operators (such as mobile cloud users).

3.5 Discussion

The problem of VM instance co-residence detection

Although the experimental results in Section 4 are very encouraging, one problem is left unsolved: how can the attacker put the malicious VM instance onto the same physical box as the target and determine their co-residence. There exist two adversarial strategies to place attacker's VM onto the same physical box as the target. The first strategy is brute-forcing placement. In this mechanism, the attacker will launch numerous instances over a period of time and conduct co-residence test discussed below. Previous research [81] shows that this simple approach has about 10% probability to successfully put at least one VM of the attacker onto the same physical box as the target. In the second attack strategy, the attacker can attempt to use the strong sequential and parallel placement locality of VM instances that has been shown in third party clouds such as Amazon EC2. [81] With this property, if attackers launch VM instances relatively soon after the launch of the target, they have a better chance to achieve co-existence.

We understand that different VM management systems have different mapping policies among the virtual machines and physical boxes. For example, some systems use static mapping between the two groups. Under this condition, we can use the information such as the Dom0 IP addresses and internal IP addresses in the cloud to determine whether or not two instances are on the same physical box. Such management policy will also allow us to launch a new instance immediately after the termination of our previous instance so that the new one will take the slot of the terminated one.

The dynamic mapping between the VMs and physical boxes may even help attackers on their fingerprinting procedures. For example, to maximize the benefits of memory de-duplication, the hypervisors may move all instances with the same OS type onto a single physical box. Under this condition, the attacker only needs to use the co-residence detection schemes to determine whether or not its VM is on the same physical box as the target.

Attackers can use two groups of mechanisms to verify co-residence of their malicious VM and the target. In the first group the attacker can examine the similarity of their Dom0 IP addresses and internal IP addresses in the cloud since many third party cloud management systems use static mapping between the addresses of VMs and the physical boxes. In the second group attackers can investigate load-based co-residence detection schemes through side-channels. [81, 76] The basic idea is to induce different levels of computation and data access loads onto the target and measure the operation delay of attacker's VM instance. If the two sequences of events match very well, the two VMs have a good chance to be located on the same physical box.

Porting the attack to other hypervisors

Although we implement and evaluate the proposed attack using only the VMware ESXi hypervisor, the technique can be easily ported to other hypervisors that support memory de-duplication. For example, researchers [90] have examined information leakage caused by memory de-duplication in Linux KSM (Kernel Samepage Merging). In their experiments, attackers construct memory pages that have the same contents as some specific applications. They will then measure the write access time to these pages to determine whether or not the applications are initiated in the target VM. In [43], memory harvesting using de-duplication is implemented in Xen. Therefore, the similar attack can be conducted in this environment with minor changes.

Other Prevention Methods of de-duplication based fingerprinting

Prevention of OS fingerprinting at the hypervisor level

Special mechanisms can also be designed to defend against the de-duplication based OS fingerprinting attacks at the hypervisor. We propose two mechanisms which can be adopted by the hypervisor to defend against the studied attacks. First, the hypervisor can label the memory pages so that the de-duplication operations can be conducted on only the pages of data files but not OS images. The disadvantages of this mechanism include the required modification to existing hypervisors and the reduced memory usage efficiency. In the second mechanism, physical isolation among the VMs will be enforced so that only mutually trusted VMs will be positioned in the same physical box. For example, Amazon introduces a new service with physically isolated, tenant-specific hardware so that NASA will join its cloud infrastructure. [89] In [112], researchers have designed a scheme to help end users to verify the physical isolation among VMs.

Prevention of Data Set Fingerprinting

Since the proposed data set identification mechanism uses the access delay to the memory pages to identify their contents, the defense mechanisms need to hide the difference. Two approaches can be used to achieve the goal. In the first approach, we can combine the unique pages of different data sets to construct a data file. When a VM instance tries to defend against the fingerprinting attack through memory de-duplication, it can periodically read the data file into its memory. In this way, the instance will demonstrate the signatures of multiple data sets and raise the difficulty level of identification. Please note that this approach can only defer the attack but not totally disable it since the real data set in use will still be in memory.

In the second approach, we can change the organization of the data records in the memory when they are loaded from the hard disk. In this mechanism, we can adopt different indexing schemes to organize the data records in the main memory. Therefore, even when the same data file is read at different VMs, the memory pages will still have different contents. The flexibility in data organization will not introduce too much overhead in information processing and analysis since many software packages such as MS SQL server are designed to support multiple logical equivalent plans for query processing. [77]

3.6 Conclusion

In this chapter we propose a new OS and data set fingerprinting mechanism for VM instances on hypervisors that enable the memory de-duplication functionality. Memory de-duplication will make it so mobile mashups who use virtual machines to better secure the end users can run several of the mobile virtual machines on a single host. However, this strategy must be properly secured. The analysis shows that the reading and writing delay of the memory pages will demonstrate a measurable difference when they do not have their own copies in the memory. Experimental results on multiple OS types show that each of these OS contains a large number of unique pages that can be used as its signature. Additionally, experimental results on multiple biological data sets show that each of these sets contains a large number of unique pages that can be used as its signature. We can use the co-residence detection schemes to launch VM instances onto the same physical box as the target and determine its OS type or the data set in use. Different from previous approaches that need interaction with the target, our approach is more difficult to detect by IDS or network traffic monitor. Our defense program proves that a virtual machine can defend itself from deduplication based fingerprinting.

CHAPTER 4: PRESERVING DATA QUERY PRIVACY IN MOBILE MASHUPS

4.1 Introduction

With the proliferation of smart wireless devices such as Android and Apple phones, numerous new widgets based on the widely adopted Web 2.0 standard are developed. Among these applications, many are built upon the mashup technique. A mashup application is a system which combines the local contents with the information from other web providers, such as Google, Ebay, and Craigslist into an integrated information presentation platform. The published web service interfaces such as Google Maps, Yahoo! Flickr, and Amazon Web Services greatly simplify the creation of the mashups by hiding their internal complexity.

While the mashup technique enables the development of new and convenient applications, it also raises security concerns. Previous research has been focusing on the prevention of information leakage among data/service providers in the same mashup pages. [29, 92, 98, 108] In this chapter, we investigate the security problem from a new perspective. As an example, a health care service delivering company develops a mobile mashup: a patient needs to input only an address and a disease. The mashup will search in the company's database to locate all doctors specialized in this disease within 10 miles of the input address. It will then retrieve information from two other providers to label these doctors on a map and show their patient reviews. The company knows that the privacy of health information is critical. Therefore, it encrypts the query results from the database. Unfortunately, the map and doctor review companies transmit information in plain text. Now if an attacker *Eve* monitors the returned data of this mashup to *Alice*'s cell phone, she can easily derive the disease that Alice might have. This kind of information leakage is beyond the control of the developer of this mashup since she/he cannot determine the interface design of the data providers.

In parallel to the development of mobile mashups is the design and deployment of mobile cloud computing. [33, 46] Mobile cloud provides transparent services to portable devices such as smart phones in order to resolve the discrepancy between the limited resources of such devices and the demands of innovative applications. While the security of mobile cloud has attracted interests from researchers [45, 111], using it to provide security services for other application environments deserves more efforts. At least three reasons make us believe that mobile cloud is an appropriate solution to privacy preservation in mashups. First, with cloud based storage providers offering to keep private information available despite company technology failures and enable fast storage retrieval from anywhere in the world with worldwide distributed storage, mobile users will easily be able to supplement their small data storage capabilities with cloud based storage. Thus preventing a lost smartphone from providing sensitive information to third parties. Second, mobile cloud provides transparent services to end users so that we can hide the sources of aggregated information. Third, mobile cloud allows components of the mashup software to move freely between end users and cloud infrastructure. In this way, it becomes more difficult for attackers to track the information flow and/or gather private information.

Although the client-side mashup technique has dominated mashups for desktop computers, recent research [2] shows that for mobile devices client-side, server-side, and even hybrid mashups are all popular. Therefore, in this chapter we propose to design two mechanisms for privacy preservation in data acquirement for client-side and server-side mashups respectively. For server-side mashups, the server can dynamically create virtual machines in the mobile cloud to work as proxies to handle information collection and integration. The nondeterministic location of a virtual machine makes it extremely difficult for attackers to eavesdrop on the communication contents. For client-side mashups, we propose to use the techniques described in [22, 25] to decompose the application into multiple components and move the data collection and aggregation part into the cloud. The overhead and safety of both approaches will also be studied.

The contributions of the chapter can be summarized as follows. First, we explore using mobile cloud to improve information privacy in mobile mashups. Second, we have designed different mechanisms for both client-side and server-side approaches so that they can be integrated with various widgets. Last but not least, we study both security and performance of the proposed approaches to evaluate their potential for wide deployment.

The remainder of this chapter is organized as follows. In Section 2 we will discuss the related work. In Section 3 we will present the details of the proposed approaches and their application environments. Section 4 will investigate the performance and security of the approaches. Finally, Section 5 will conclude this chapter.

4.2 Related Work

As summarized in [30], previous research on security in mashup applications focuses on preventing information leakage among multiple sub-frames belonging to different information sources. The approaches can be classified into three groups. In the first group, researchers define the “object-capability” languages in which access control, message passing, and object reference are all tightly managed. For example, Caja [67] implements a subset of javascript to protect third party contents in web applications. The second group use the concept of sandboxing. They first encapsulate information from each source with a subspace. Specially designed communication channels are then implemented among these components. Subspace [48] and WebJail [98] are examples of the approaches. In the third group, a complete set of communication protocols are implemented so that all sub-frames must communicate with each other through the protocols. OMOS [108, 109] and SMash [29] are examples of the approaches.

Using proxy-based approaches to preserve privacy in networks has been investigated in different environments. For example, both Crowds [78] and Hordes [86] use proxies to defend against traffic analysis attacks in Internet. Similar technique has been adopted to achieve anonymity in HTTP requests [38], peer-to-peer networks [85], location based services [12], and authorization and accounting [69], to name a few. The techniques of cloud and virtualization make the dynamic establishment and maintenance of a proxy very easy.

Decomposing a web application into multiple components and moving some of them into cloud can serve different purposes of mobile computing. The early models such as Click modular router [55], Dryad [47], and IBM SPL [44] all enable an application to be composed by connecting modules. Both CloneCloud [22] and MAUI [25] focus on reducing energy consumption by moving some components of an application into the cloud. In [60], the authors propose a composition approach to rich mobile application development to promote modular, flexible and configurable widgets, and reuse of independent software components.

In [3] the researchers investigate how to improve the application and session layers of mashup-to-browser communications so that they can enable data sharing between providers while still preserving the user privacy. This research does not investigate safety of the user-to-mashup-to-data provider communication links, nor does it acknowledge a possible correlation between these communication pathways that an attacker could take advantage of.

In [4] the authors focus on securing the link between end users and mashup content providers. The approach can also defend against cross site scripting and cross site request forgery attacks. In [11] the authors investigate solutions to preventing a malicious widget used by a mashup interface from gathering information about mobile users. They also investigate malicious widgets communicating with the network in a manner that appears as though the user's web browser itself is using the communication link. Our proposed work focuses more on network traffic vulnerabilities presented by mashup providers inadvertently turning themselves into "web traffic funnels", and thus, becoming a very enticing target for malicious parties.

4.3 The Proposed Approaches

In this section, we present the details of the proposed approaches. We will first elaborate on the system assumptions and the attacker model. We will then discuss the privacy preservation procedures for server-side and client-side mashups respectively.

System Assumptions and Attacker Model

Figure 23 illustrates the application scenarios that we need to protect. In this environment, end users depend on mashups through mobile cloud to get the needed data and services. Some of the providers may reside in the cloud, while others are not. The figure shows both server-side and client-side mashup applications. The mashup engine is in charge of the collection and integration of returned data. We assume that end users have enough computation power to support secure encryption algorithms so that they can protect confidentiality of the traffic. However, neither the end users nor the server has any control over the communication standard with the third party providers. They must follow the exposed interfaces of the data providers. Therefore, it is possible that a data provider chooses to return data in plain text and any eavesdropper will be able to understand the contents.

We assume the attacker model described in [62] and make some updates to it. Here an attacker will be able to eavesdrop on the incoming and outgoing traffic of the mobile end users. If a mashup server is used and it has a fixed position and static IP address in the infrastructure, the attacker will be able to eavesdrop on the node as well. The attacker does not have the computation power to compromise the secure encryption algorithms and recover the contents. In theory, for a dynamically created virtual machine in the mobile cloud, the attacker could trace and locate the node, launch a co-residence attack [81, 112], and steal information from the VM through side-channel attacks. [73] However, the analysis in subsequent subsections will show that the difficulty level and overhead of such attacks are very high. Therefore, we assume that the attackers cannot eavesdrop on a dynamically created virtual machine in the mobile cloud even if its IP address is known.

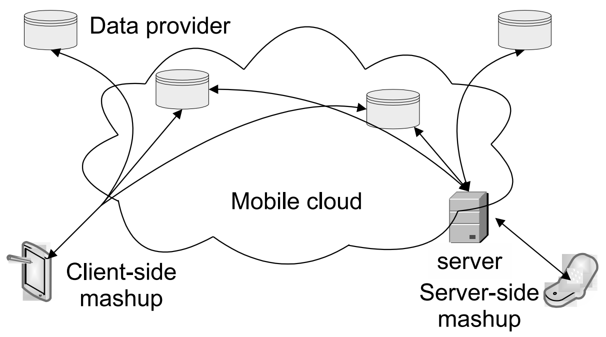


Figure 23: Mashup scenarios in mobile cloud.

k-anonymity [91] is a popular technique for hiding the target in a crowd in order to preserve privacy. This technique, however, is not efficient in the investigated scenarios. If the chosen value of *k* is too small, the privacy of end users will not be properly protected. On the contrary, if *k* is too large, too much overhead will be introduced. For client-side mashups, a mobile device has to consume valuable power resources to send out $k-1$ fake requests. Since some providers will charge the mashup engine for every submitted request, this approach could also be costly for server-side mashups. Based on this analysis, we find that a new mechanism must be designed for privacy preservation in this application.

Privacy Preservation for Server-side Mashups

As described in section 3.1, an attacker can eavesdrop on the network traffic of the mashup server if it has a fixed position and static IP address. However, it becomes much more difficult to locate and monitor a dynamically created virtual machine in the mobile cloud. In this part, we will design a mechanism based on this observation.

Figure 24 illustrates the proposed mechanism to protect information privacy in server-side mashups. Here instead of letting the mashup server directly connect to the data and service providers, we request it to initiate multiple virtual machines in the cloud to serve as proxies. Since the VMs are created by the server, it can determine the communication

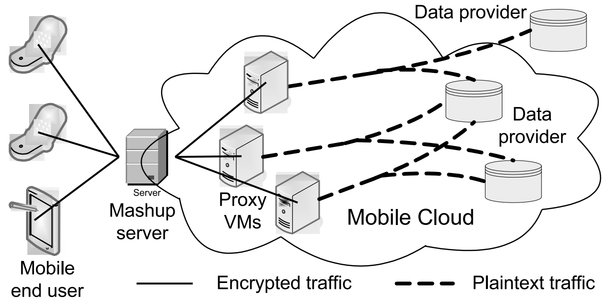


Figure 24: Privacy preservation for server-side mashups.

methods with the VMs. Therefore, secure encryption algorithms can be used to protect network traffic on this segment. Note that traffic between the proxies and data providers could still be transmitted as plain text.

Through monitoring the outgoing traffic from the server, attackers can figure out the IP addresses of the proxies. However, eavesdropping on the proxies will be a very difficult job that demands a lot of resources from the attackers. Experiments in [81] show that the attackers need to launch 40 to 80 virtual machines to achieve co-residence with one of the proxies if the mobile cloud provider does not adopt any VM placement policies to prevent cartography. This number could be even larger if the owner of the cloud randomly places VMs in the infrastructure.

Even after putting a malicious VM onto the same physical box as a proxy, the attacker still faces the challenge to derive out the network traffic of the proxy through side channel attacks. Please note that this is different from the cross VM private key extraction attack [113] in the following aspects. First and most importantly, in [113] the attacker must be able to remotely and repeatedly activate the encryption algorithm in the target VM and put the secret key into cache. In our scenario, the mashup procedure is initiated by the end user as a one-time execution so the attacker has no control over its happening. Second, in [113] the attacker knows exactly where the key will reside in the cache through code analysis. In our scenario, the returned information for the mashup could be put at any place in the cache. Last but not least, in [113] the attacker must be able to regain control of the CPU sufficiently frequently to conduct side-channel observations through the trigger of Inter-Processor Interrupts (IPI). Such functionality may not be enabled in the mobile cloud environment.

The mashup server can create multiple VM proxies to further increase the difficulty for the attackers to trace and intercept network traffic. Here we propose two dispatching mechanisms for the server to place requests from end users to different proxies. In the "vertical" dispatching scheme, every request from a mobile user will be treated as a separate transaction and assigned to a single proxy. Round-robin or least workload based placement can be used to maintain balance among the proxies. The advantage of this approach is that when the server determines to terminate a proxy, we can stop assigning new requests to it. Therefore, the switch to a new proxy can be accomplished smoothly.

In the "horizontal" dispatching scheme, each proxy will be in charge of handling all interaction with one or a few data providers. In this way, every user request needs the collaboration of multiple proxies and only the server itself can integrate the returned data. The advantage of this approach is that when an attacker intercepts the network traffic to a proxy, it can get access to only a part of the mashup result. At the same time, since there could be many returned messages from the same provider, the attacker will not be able to identify the request that it is interested in. The disadvantage, however, is that a malfunctioned proxy may impact the processing of a large number of user requests. The designer of a mashup application needs to choose the dispatching mechanism that fits her/his needs the best.

Privacy Preservation for Client-side Mashups

In theory, we can adopt a similar technique to protect the privacy of information in client-side mashups. Here each mobile device can initiate a virtual machine in the mobile cloud and use it as a proxy to handle information integration. However, products such as Mobile Virtualization Platform (MVP) by VMware cannot yet provide full control of a virtual environment through an app on a thin client such as a smart phone. Therefore, we need to design some new mechanisms to protect client-side mashups.

Although the technique to control multiple VM proxies through a thin mobile client has not become mature, partitioning the execution of a mobile application and outsourcing some operations into the cloud has become a practical solution. [22, 25] In this chapter, we propose to build an approach upon CloneCloud. [22] The basic idea is shown in Figure 25.

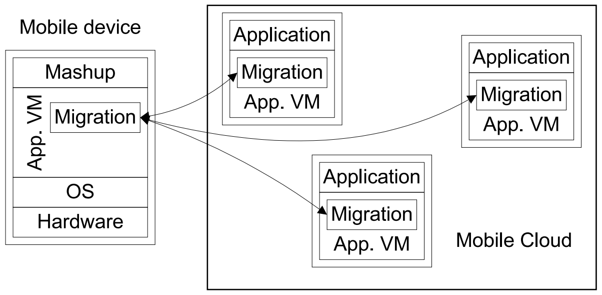


Figure 25: CloneCloud based privacy preservation.

In this approach, the mobile device will host an application level virtual machine such as JVM, Microsoft .NET, or the DalvikVM in Android to execute the mashup application. The VM will contain a partitioning and migration management component. When the mashup application is ready to pull information from providers, the partitioner will suspend the execution on the mobile device, and migrate the thread to a device clone that is hosted in the cloud. The migrated thread will execute on the clone to acquire and integrate information from different providers. Eventually, the thread will return to the mobile device and merge the remotely created states into the original process.

Different from the mechanism described in Section 3.2, this approach does not depend on a static server. On the contrary, the VM clone is a service provided by the mobile cloud and it can be hosted by any physical box in the cloud. The mobile device can execute the mashup in an application level VM until the information acquirement phase. It will then package the states of the VM and port it to its clone in the cloud. Since a mobile user can randomly choose the placement of its clone, it is very difficult for an attacker to intercept the mashup traffic unless it can eavesdrop on the whole cloud simultaneously. Since the thread migration traffic is encrypted, the attacker will not learn anything by monitoring the inbound and outbound traffic of the mobile user.

Most functionality needed for this approach can be found in the implementation of CloneCloud. We can choose the start position of thread migration in the mashup application so that all communication with the data providers is conducted by the clone VM. The functionalities of suspension, porting, resume, and merge have been included in the DalvikVM of Android. Since the mobile end users can randomly choose machines in the cloud to host their clones, we do not expect unbalanced workload to occur frequently. A machine in the cloud can reject the migration request if it does not have enough resources to host another application level VM.

4.4 Experiment Results and Evaluation

In this Section, we will present the evaluation efforts and the experiment results. We will focus on the changes in computation and communication overhead in both mobile devices and servers. We will also study power consumption of the proposed approaches.

Server-side Mashups

The proxy based approach has no impact on the computation or communication overhead at the mobile devices since all operations are conducted by the server. The end users, however, could expect a longer delay that is introduced by the extra communication segment. The extra overhead at the server comes from the following aspects. First, the server needs to initiate, manage, and terminate the proxy VMs in the cloud. Second, to prevent attackers from eavesdropping on the server, it has to encrypt the communication traffic with the proxies. When the server handles many requests simultaneously, the increased CPU usage could impact the system performance. Last, depending on the adopted dispatching algorithm for proxies, the server may need to integrate returned data from different providers to generate the mashup results.

To assess the overhead in real networks, we setup an evaluation environment. The mashup server is a Ubuntu virtual machine with 1GB of RAM and an allotment of two processors from a Dell Optiplex 980. We choose the open source mashup WSO2 as the application so that we can easily change its configuration. The mashup server's Internet connection was 1.47Mbps down and 490Kbps up. The proxies were running on remote servers with VMWare ESXi as the hypervisor. Each proxy was assigned one processor with 512 MB of RAM, using an Internet connection with 9.59Mbps down and 1.34Mbps up. All connections for the mashup server to data providers on the Internet were routed through the remote proxies either encrypted with Blowfish (128 bit key size) or in plain text, both by point-to-point links in OpenVPN. Experiments show that a 19-hop path exists between our server and the proxies. Our evaluation focuses on the increase in delay that is introduced by the communication segment between the mashup server and the proxies.

Table 5 shows the transmission and processing delay that is measured at the end user. It represents the time duration between the sending of a request and the return of corresponding mashup results. The decryption delay at the end user is not included since it is not impacted by our approach. The average amount of returned traffic from data providers is approximately 700KB. To better differentiate the delays caused by communication and computation overhead, we conduct three groups of experiments. In the first group, the mashup server will directly connect to the data providers and retrieve information. This is the baseline case. In the second group, the mashup server will retrieve data with the help of proxies that are hosted in the cloud. However, the data traffic between the proxies and mashup server is not encrypted. In the last group, the proxies will encrypt the returned data and then deliver it to the mashup server. 250 experiments for each group are conducted at different time in a weekday.

Table 5: Delay of proxy-based server-side mashup.

	Measured delay (ms)			95% Confidence	
	maximum	minimum	average	min	max
baseline case	820	274	292.5	292.21	292.96
proxy w/o encryption	1890	519	613	612.22	614.08
proxy with encryption	1954	523	615	614.54	616.32

From the table we find that the proposed approach roughly doubles the waiting time of the end user. This is mainly caused by the long path between the mashup server and proxies.

Comparing the second and third groups of experiments, we find that the communication delay between the mashup server and proxies dominates the increase in response time. The encryption/decryption of the returned data does not impact the delay to a large extent thanks to the computation resources available to the servers. This is very different from the results in the next section when a mobile device has to conduct such operations. We also notice that the values of the average delay and minimum delay are not far from each other, which means that most end users will experience a relatively small increase in waiting time when the proposed approach is adopted.

Client-side Mashups

For client-side mashups, most overhead introduced by the proposed approach will be put upon mobile devices. Therefore, we need to carefully evaluate its impacts on these thin clients before this approach can be widely deployed. Since modern mobile devices are usually equipped with extension storage slots such as microSD cards, we do not expect the increase in storage overhead to cause a problem. In the following discussion, we will focus on the increases in communication overhead and delay, computation overhead during the migration of the application level VM, and power consumption.

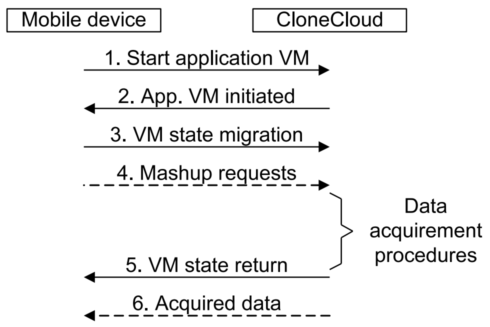


Figure 26: CloneCloud based data acquirement procedure.

Figure 26 illustrates the data acquirement procedure through clonecloud. The steps with solid underlines (steps 1, 2, 3, and 5) are new operations that will introduce extra overhead. Below we evaluate the overhead from different aspects.

The majority of the communication overhead comes from the migration and return of the application level VM states (steps 3 and 5). For our experiment, we obtain a Samsung Captivate smartphone and flash it with Cyanogenmod 10. We then setup a dual core computer with an AndroVM virtual machine. We ran two client mashup programs, Landmark Manager and EarthAlbum. Landmark Manager is a program which shows the user a Google map satellite image of a city, overlaid with frequently visited landmarks in that city. It pulls its data from a number of sources such as google.com, googlehosted.com, ggpht.com, skyhookwireless.com, amazonaws.com, gms-world.net and doubleclick.net. The data collected from these sources is approximately 700KB. EarthAlbum is a program that displays a Google map satellite image of the globe, and permits the user to tap a location. When a location is tapped, flickr.com images are displayed of the area. In our experiment environment, we used a WIFI connection. Using this WIFI connection, the phone was rated at 371KB/s download and 522 KB/s upload. Our communication and computation overhead results are shown in Table 6.

To defend against eavesdropping attacks from malicious nodes, data traffic between the mobile device and its clone must be encrypted, so encryption and decryption of upload and download results are factored in with the rendering time. As can be seen in Table 6, performance either increases or goes down only slightly. We expect users to see approximately the same or better performance, based on how much processing power the mashup application requires vs. the size of the mashed up result.

Table 6: Measured Delays with and without CloneCloud expectation.

	Up	AndroVM	Down	Render	CloneCloud	Autonomously
Landmark Manager	6.7 s	25.5 s	13.3 s	7 s	52.5 s	82.6 s
Earth Album	0.62 s	6.09 s	0.81 s	1 s	8.52 s	7.89 s

Table 7: Measured Power Consumption with and without CloneCloud expectation.

	Up	Down	Render/Encrypt	CloneCloud	Autonomously
Landmark Manager	0.0298%	0.0418%	0.25523%	0.33%	0.56%
Earth Album	0.0027%	0.0026%	0.1218%	0.13%	0.15%

Since the most valuable resource for mobile devices is energy, we must carefully assess the power consumption of the proposed approach. The increases in power consumption come from two aspects: (1) the transmission and receiving of the VM migration states, and (2) encryption/decryption of the acquired data. Table 7 shows our results. From this table, it can be see that in both cases, the power consumption is expected to go down when using CloneCloud as opposed to the phone operating autonomously.

4.5 Conclusion

In this chapter, we study the problem of privacy preservation for mobile mashups. Since end users do not have control over the communication protocols with the data providers, new mechanisms must be designed to defend against eavesdropping attacks. For server-side mashups in mobile clouds, we propose to use a proxy based approach to protect confidentiality of the communication between the mashup server and data providers. Experiments in cloud environments show that except for the lengthened response time, other aspects of the application performance are not impacted. For client-side mashups, we use live migration of application level virtual machines into mobile clouds to hide the data acquirement and aggregation procedures from eavesdroppers. We investigate the computation, communication, and power consumption overhead of the approach.

An immediate extension to this chapter is to explore the integration of the two approaches so that application developers and end users do not have to explicitly distinguish server-side mashups from client-side applications. A uniform approach would further reduce the difficulty of its wide deployment and adoption.

CHAPTER 5: CONCLUSION

In this dissertation, we solve the most important security problems, at this time, with cloud computing. More specifically the problems most at risk in cloud computing as is needed to facilitate mobile devices in the very near future. Mobile cloud computing promises to supplement the limited computing and power available to mobile devices to enable them to deliver nearly the same computing resources as desktop computers attached to local organizational networks, but from anywhere and with less down time. This always globally available aspect introduces new security problems that must be addressed. We have focused on:

1. Flexible cryptography-based access control of outsourced data.
2. Methods for securing virtual machines from OS and Data Fingerprinting.
3. Securing mobile-to-cloud-to-provider communications.

Flexible cryptography-based access control of outsourced data is delivered by using hierarchical-based key management in owner-write-users-read applications. We propose to encrypt every data block with a different key so that flexible cryptography-based access control can be achieved. Through the adoption of the key derivation method, the owner needs to maintain only a few secrets. We propose to store the metadata with the blocks so that multiple key derivation hierarchies can be established for different access patterns to reduce the processing overhead of the data access requests. Analysis shows that the key derivation procedure based on hash functions will introduce very limited overhead. We propose to use over-encryption and/or lazy revocation to prevent revoked users from getting access to updated data blocks. A new key management mechanism has yet to be designed to satisfy many-write-many-read applications. Recognizability of current access patterns in specific applications for optimized key management must also be designed and tested.

We have both proposed a new memory deduplication based OS and data set fingerprinting attack technique and delivered methods for securing virtual machines from this attack. By loading samplings of OS signatures into memory, we have proven it possible to thwart memory deduplication based OS fingerprinting in publicly available cloud computing environments with minimum performance impact through experimentation on VMware ESXi. The process of realigning the memory page content of data sets per application launch was proposed which, once enabled on memory consuming data set applications, should easily prevent data set fingerprinting without any degradation in performance. This defense could be further extended through enabling the hypervisor to be aware of this attack, and enabling it to inject random memory delays in known OS signatures, or alert the operator as to the attempt to deduplicate multiple OS signatures within a single VM.

Finally, we have explored securing mobile-to-cloud-to-provider communications. We have explored different mechanisms based on either server-side or client-side mashups. For server-side mashups, we proposed separating the roles of "mashing up the data" and gathering the data from external providers onto geographically separated virtual machines and presented the performance penalties from this solution through experimentation with virtual machines on separate ISPs. For client-side mashups, we proposed live migration of the mashup application to mobile cloud based virtual machines for the purposes of data gathering and computation. We also investigate the computation, communication, and power consumption overhead of this approach. An immediate extension to this research is to explore the integration of the two approaches so that application developers and end users do not have to explicitly distinguish server-side mashups from client-side applications. A uniform approach would further reduce the difficulty of its wide deployment and adoption.

Cloud computing, and most especially mobile cloud computing, are technologies that are still evolving to fill the “niches” in industry where they can best solve problems or make computing resources more efficient. This primarily industry driven technology has yet to fulfill its full potential. Every step of the way, new security problems will arise that must be addressed. After my graduation, the research on how to better optimize defenses to the primary security problems addressed in this paper will continue. Both myself and others at UNC Charlotte will continue developing cloud computing security solutions that better address these problems from which many of the existing and ever changing cloud computing providers and end users can use to provide a safer cloud computing experience for all.

REFERENCES

[1] Adya, A., Bolosky, W. J., Castro, M., Cermak, G., Chaiken, R., Douceur, J. R., Howell, J., Lorch, J. R., Theimer, M., and Wattenhofer, R. P. Farsite: federated, available, and reliable storage for an incompletely trusted environment. SIGOPS Oper. Syst. Rev. 36, SI (2002), 1–14.

[2] Agarwal, V., Goyal, S., Mittal, S., and Mukherjea, S. A middleware framework for mashing device and telecom features with the web. Tech. rep., IBM Research, RI 10009, 2010.

[3] Alam, M., Zhang, X., Nauman, M., Khan, S., and Alam, Q. Mauth: A fine-grained and user-centric permission delegation framework for multi-mashup web services. In Proceedings of the World Congress on Services (2010), pp. 56–63.

[4] Ali, S., Khusro, S., and Rauf, A. A cryptography-based approach to web mashup security. In International Conference on Computer Networks and Information Technology (ICCNIT) (2011), pp. 53–57.

[5] Alliance, O. A. Ajax and mashup security. http://www.openajax.org/whitepapers/Ajax and Mashup Security.php, 2013.

[6] Alvarez, G., Borowsky, E., Go, S., Romer, T., Becker-szendy, R., Golding, R., Merchant, A., Spasojevic, M., Veitch, A., and Wilkes, J. Minerva: an automated resource provisioning tool for large-scale storage systems. ACM Transactions on Computer Systems 19 (2001), 483–518.

[7] Arcangeli, A., Eidus, I., and Wright, C. Increasing memory density by using ksm. In Linux Symposium (2009), pp. 19–28.

[8] Atallah, M. J., Blanton, M., Fazio, N., and Frikken, K. B. Dynamic and efficient key management for access hierarchies. ACM Trans. Inf. Syst. Secur. 12, 3 (2009), 1–43.

[9] Auffret, P. Sinfp, unification of active and passive operating system fingerprinting. Jour. Comp. Virology 6, 3 (2010), 197–205.

[10] Aviram, A., Hu, S., Ford, B., and Gummadi, R. Determinating timing channels in compute clouds. In Proceedings of ACM workshop on Cloud computing security workshop (2010), pp. 103–108.

[11] Batard, F., Boudaoud, K., and Riveill, M. A middleware for securing mobile mashups. In Proceedings of international conference companion on World wide web (2011), pp. 9–10.

[12] Beresford, A., and Stajano, F. Location privacy in pervasive computing. Pervasive Computing, IEEE 2, 1 (2003), 46–55.

[13] Blaze, M. Key management in an encrypting file system. In Proceedings of the USENIX Summer Technical Conference (1994), pp. 27–35.

[14] Bradshaw, D., Folco, G., Cattaneo, G., and Kolding, M. Quantitative estimates of the demand for cloud computing in europe and the likely barriers to up-take, 2012.

[15] Caldwell, T. Mobile cloud security: How to address the issues, 2011.

[16] Caprara, A., Fischetti, M., and Toth, P. Algorithms for the set covering problem. Annals of Operations Research 98 (1998), 353–371.

[17] Catteddu, D., and Hogben, G. Cloud computing - benefits, risks and recommendations for information security. Information Security 51273, 9 (2009), 1–125.

[18] Cedilnik, A., Geveci, B., Moreland, K., Ahrens, J., and Favre, J. Remote large data visualization in the paraview framework. In Eurographics Parallel Graphics and Visualization (2006), pp. 162–170.

[19] Chen, T., Chung, Y., and Tian, C. A novel key management scheme for dynamic access control in a user hierarchy. In IEEE Annual International Computer Software and Applications Conference (2004), pp. 396–401.

[20] Chien, H., and Jan, J. New hierarchical assignment without public key cryptography. Computers & Security 22, 6 (2003), 523–526.

[21] Chow, A., Coates, W., and Hopkins, D. A configurable asynchronous pseudorandom bit sequence generator. In IEEE International Symposium on Asynchronous Circuits and Systems (2007), pp. 143–152.

[22] Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. Clonecloud: elastic execution between mobile device and cloud. In Proceedings of the sixth conference on Computer systems (2011), pp. 301–314.

[23] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing. http://www.cloudsecurityalliance.org/, April 2009.

[24] Cox, P. A. Privacy build a more secure, mobile cloud environment. http://www.ibm.com/developerworks/cloud/library/cl-mobilecloudsecurity/, 2011.

[25] Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. Maui: making smartphones last longer with code offload. In Proceedings of the international conference on Mobile systems, applications, and services (2010), pp. 49–62.

[26] Damiani, E., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P. An Experimental Evaluation of Multi-Key Strategies for Data Outsourcing, IFIP International Federation for Information Processing, Volume 232, New Approaches for Security, Privacy and Trust in Complex Environments. Springer, 2007, pp. 385–396.

[27] Damiani, E., di Vimercati, S. D. C., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P. Key management for multi-user encrypted databases. In Proceedings of the ACM workshop on Storage security and survivability (2005), pp. 74–83.

[28] D'Atri, A., and Ricci, F. L. Interpretation of statistical queries to relational databases. In Proceedings of the international conference on Statistical and Scientific Database Management (1988), pp. 246–258.

[29] De Keukelaere, F., Bhola, S., Steiner, M., Chari, S., and Yoshihama, S. Smash: secure component model for cross-domain mashups on unmodified browsers. In WWW (2008), pp. 535–544.

[30] Decat, M., Ryck, P. D., Desmet, L., Piessens, F., and Joosen, W. Towards building secure web mashups. In OWASP AppSec Research (2010).

[31] di Vimercati, S. D. C., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P. A data outsourcing architecture combining cryptography and access control. In Proceedings of the ACM workshop on Computer security architecture (2007), pp. 63–69.

[32] di Vimercati, S. D. C., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P. Over-encryption: management of access control evolution on outsourced data. In Proceedings of the international conference on Very large data bases (2007), pp. 123–134.

[33] Dinh, H. T., Lee, C., Niyato, D., and Wang, P. A survey of mobile cloud computing: architecture, applications, and approaches. Wireless Communications and Mobile Computing (2011).

[34] European Institute of Tele-Surgery - Institut de Recherche Contre les Cancers de L'Appareil Digestif. Ircad/eits laparoscopic center. http://www.ircad.fr/softwares/3Dircadb/3Dircadb.php, 2011.

[35] Eurosoft. Qa+win32 - diagnostic software. http://www.eurosoft-uk.com/qawin32.html, 2010.

[36] Farber, D. Amazon the new geek chic: Data centers, 2012.

[37] Fyodor. Remote os detection via tcp/ip stack fingerprinting. Tech. rep., 1999.

[38] Gabber, E., Gibbons, P. B., Kristol, D. M., Matias, Y., and Mayer, A. Consistent, yet anonymous, web access with lpwa. Commun. ACM 42, 2 (1999), 42–47.

[39] Gellman, R. Privacy in the clouds: Risks to privacy and confidentiality from cloud computing, 2009.

[40] Goh, E., Shacham, H., Modadugu, N., and Boneh, D. Sirius: Securing remote untrusted storage. In Proceedings of the Internet Society (ISOC) Network and Distributed Systems Security (NDSS) Symposium (2003), pp. 131–145.

[41] Goodrich, M. T., Papamanthou, C., Tamassia, R., and Triandopoulos, N. Athos: Efficient authentication of outsourced file systems. In Proceedings of the international conference on Information Security (2008), pp. 80–96.

[42] Greenwald, L. G., and Thomas, T. J. Toward undetected operating system fingerprinting. In Proceedings of the first USENIX workshop on Offensive Technologies (2007), pp. 6:1–6:10.

[43] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A., Varghese, G., Voelker, G., and Vahdat, A. Difference engine: harnessing memory redundancy in virtual machines. Commun. ACM 53, 10 (2010), 85–93.

[44] Hirzel, M., Andrade, H., Gedik, B., Kumar, V., Losa, G., Nasgaard, M. H., Soule, R., and Wu, K.-L. Spl stream processing language specification. Tech. rep., Tech. Rep RC24897 (W0907-066), IBM Research, 2009.

[45] Huang, D., Zhang, X., Kang, M., and Luo, J. Mobicloud: Building secure cloud framework for mobile computing and communication. In IEEE International Symposium on Service Oriented System Engineering (SOSE) (2010), pp. 27–34.

[46] Huerta-Canepa, G., and Lee, D. A virtual cloud computing provider for mobile devices. In Proceedings of the ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond (2010), pp. 6:1–6:5.

[47] Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. Dryad: distributed data-parallel programs from sequential building blocks. In ACM European Conference on Computer Systems (2007), pp. 59–72.

[48] Jackson, C., and Wang, H. J. Subspace: secure cross-domain communication for web mashups. In WWW (2007), pp. 611–620.

[49] Jansen, W., and Grance, T. Guidelines on security and privacy in public cloud computing, 2011.

[50] Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., and Fu, K. Plutus: Scalable secure file sharing on untrusted storage. In Proceedings of the USENIX Conference on File and Storage Technologies (2003), pp. 29–42.

[51] Karp, R. Reducibility among combinatorial problems. In Complexity of Computer Computations, R. Miller and J. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.

[52] Kaufman, W. Cloud computing saves health care industry time and money. http://www.npr.org/blogs/alltechconsidered/2012/10/01/162080613/cloud-computing-saves-health-care-industry-time-and-money, 2012.

[53] Kher, V., and Kim, Y. Securing distributed storage: challenges, techniques, and systems. In Proceedings of the ACM workshop on Storage security and survivability (2005), pp. 9–25.

[54] Kitware. Paraview. http://paraview.org/, 2011.

[55] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. The click modular router. ACM Trans. Comput. Syst. 18, 3 (2000), 263–297.

[56] Koschmider, A., Torres, V., and Pelechano, V. Elucidating the Mashup Hype: Definition, Challenges, Methodical Guide and Tools for Mashups. In 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (2009).

[57] Li, J., Krohn, M., Mazières, D., and Shasha, D. Secure untrusted data repository (sundr). In Proceedings of the conference on Symposium on Opearting Systems Design & Implementation (2004), pp. 121–136.

[58] Li, P., Li, Z., Halang, W. A., and Chen, G. A multiple pseudorandom-bit generator based on a spatiotemporal chaotic map. Physics Letters A 349, 6 (2006), 467–473.

[59] Lin, C. Hierarchical key assignment without public-key cryptography. Computers & Security 20, 7 (2001), 612–619.

[60] March, V., Gu, Y., Leonardi, E., Goh, G., Kirchberg, M., and Lee, B. S. μcloud: Towards a new paradigm of rich mobile applications. Tech. rep., HP Laboratories, HPL-2011-55R1, 2011.

[61] McCullough, B., and McKitrick, R. Check the numbers: The case for due diligence in policy formation. the Fraser Institute, February, pp.2–43, 2009.

[62] Mehta, K., Liu, D., and Wright, M. Location privacy in sensor networks against a global eavesdropper. In Proceedings of the IEEE International Conference on Network Protocols (ICNP) (2007).

[63] Mell, P., and Grance, T. The nist definition of cloud computing, 2012.

[64] MICROSOFT DEVELOPER NETWORK. timegettime. http://msdn.microsoft.com/en-us/library/ms713418(VS.85).aspx, 2010.

[65] Miled, Z., Liu, Y., Powers, D., Bukhres, O., Bem, M., Jones, R., and Oppelt, R. An efficient implementation of a drug candidate database. J. Chem. Inf. Comput. Sci. 43, 1 (2003), 25–35.

[66] Miller, E., Long, D., Freeman, W., and Reed, B. Strong security for distributed file systems. In IEEE International Conference on Performance, Computing, and Communications (2001), pp. 34–40.

[67] Miller, M. S., Samuel, M., Laurie, B., Awad, I., and Stay, M. Caja: Safe active content in sanitized javascript, June 2008.

[68] Mudge, J. C. Cloud computing: Opportunities and challenges for australia. Tech. rep., The Australian Academy of Technological Sciences and Engineering (ATSE), 2010.

[69] Neuman, B. C. Proxy-based authorization and accounting for distributed systems. In ICDCS (1993), pp. 283–291.

[70] Okamura, K., and Oyama, Y. Load-based covert channels between xen virtual machines. In Proceedings of the ACM Symposium on Applied Computing (2010), pp. 173–180.

[71] Ort, E., Brydon, S., and Basler, M. Mashup styles, part 1: Server-side mashups. Tech. rep., Oracle, 2007.

[72] Owens, R., and Wang, W. Fingerprinting large data sets through memory deduplication technique in virtual machines. In MILITARY COMMUNICATIONS CONFERENCE, 2011 - MILCOM 2011 (2011), pp. 1363–1368.

[73] Owens, R., and Wang, W. Non-interactive os fingerprinting through memory deduplication technique in virtual machines. In IEEE International Performance Computing and Communications Conference (2011).

[74] Plummer, D. The business landscape of cloud computing. Tech. rep., Gartner, 2012.

[75] Preneel, B., Rompay, B. V., Ors, S. B. ., Biryukov, A., Granboulan, L., Dottax, E., Dichtl, M., Schafheutle, M., Serf, P., Pyka, S., Biham, E., Barkan, E., Dunkelman, O., Stolin, J., Ciet, M., Quisquater, J.-J., Sica, F., H.Raddum, and Parker, M. Performance of optimized implementations of the nessie primitives. Deliverable 21 from the NESSIE IST FP5 project, 2003.

[76] Raj, H., Nathuji, R., Singh, A., and England, P. Resource management for isolation enhanced cloud services. In ACM Cloud Computing Security Workshop (2009), pp. 77–84.

[77] Rankins, R., Jensen, P., Bertucci, P., Gallelli, C., and Silverstein, A. Microsoft SQL Server 2000 Unleashed. Sams, 2001.

[78] Reiter, M. K., and Rubin, A. D. Crowds: anonymity for web transactions. ACM Trans. Inf. Syst. Secur. 1, 1 (1998), 66–92.

[79] Rhodes, P. J. Granite: a scientific database model and implementation. PhD thesis, University of New Hampshire, 2004. Adviser-Bergeron, R. Daniel and Adviser-Sparr, Ted M.

[80] Richardson, D., Gribble, S., and Kohno, T. The limits of automatic os fingerprint generation. In ACM workshop on Artificial intelligence and security (AISec) (2010), pp. 24–34.

[81] Ristenpart, T., Tromer, E., Shacham, H., and Savage, S. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In ACM CCS (2009), pp. 199–212.

[82] Robert J. Jenkins, J. Isaac. In Third International Workshop on Fast Software Encryption (1996), pp. 41–49.

[83] Rotenberg, M., Verdi, J., and Sen, A. In the matter of google, inc. and cloud computing services complaint and request for injunction, request for investigation and for other relief. http://epic.org/privacy/cloudcomputing/google/ftc031709.pdf, 2009.

[84] Sakamoto, H. Data grid deployment for high energy physics in japan. Computer Physics Communications 177, 1-2 (July 2007), 239–242.

[85] Scarlata, V., Levine, B. N., and Shields, C. Responder anonymity and anonymous peer-to-peer file sharing. In International Conference on Network Protocols (ICNP) (2001), pp. 272–280.

[86] Shields, C., and Levine, B. N. A protocol for anonymous communication over the internet. In ACM CCS (2000), pp. 33–42.

[87] Shimrat, O. Cloud computing and healthcare. Tech. rep., Healthcare Information and Management Systems, 2009.

[88] Singh, A., and Liu, L. Sharoes: A data sharing platform for outsourced enterprise storage environments. In Proceedings of the IEEE International Conference on Data Engineering (2008), pp. 993–1002.

[89] Stone, B., and Vance, A. Companies slowly join cloud computing. New York Times, 18 April, 2010.

[90] Suzaki, K., Iijima, K., Yagi, T., and Artho, C. Memory deduplication as a threat to the guest os. In Proceedings of the Fourth European Workshop on System Security (2011), pp. 1:1–1:6.

[91] Sweeney, L. k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10, 5 (2002), 557–570.

[92] Taivalsaari, A., and Mikkonen, T. Mashups and modularity: Towards secure and reusable web applications. In Automated Software Engineering Workshops (2008), pp. 25–33.

[93] Taleck, G. Ambiguity resolution via passive os fingerprinting. In RAID (2003), pp. 192–206.

[94] Taylor, S., Young, A., Kumar, N., and Macaulay, J. A marriage made in heaven: Mobile devices meet the mobile cloud. Tech. rep., CISCO, 2011.

[95] Taylor, S., Young, A., Kumar, N., and Macaulay, J. Mobile consumers reach for the clouds cisco ibsg research uncovers opportunities for sps to prosper in mobile cloud market. Tech. rep., CISCO, 2011.

[96] Taylor, S., Young, A., Kumar, N., and Macaulay, J. Taking care of business in the mobile cloud. Tech. rep., CISCO, 2011.

[97] Umetani, S., and Yagiura, M. Relaxation heuristics for the set covering problem. Journal of the Operations Research 50 (2007).

[98] Van Acker, S., De Ryck, P., Desmet, L., Piessens, F., and Joosen, W. Webjail: least-privilege integration of third-party components in web mashups. In Annual Computer Security Applications Conference (ACSAC) (2011), pp. 307–316.

[99] Verma, A., Ahuja, P., and Neogi, A. Power-aware dynamic placement of hpc applications. In Proceedings of the annual international conference on Supercomputing (2008), pp. 175–184.

[100] Viana, W., Filho, B., and Castro, R. M. C. Pearl: a performance evaluaator of cryptographic algorithms for mobile devices. In The First International Workshop on Mobility Aware Technologies and Applications (MATA) (2004).

[101] VMWare. Esxi configuration guide. VMware vSphere 4.1 Documentation, 2010.

[102] VMWare. Timekeeping in vmware virtual machines. http://www.vmware.com/files/pdf/Timekeeping-In-VirtualMachines.pdf, 2010.

[103] VMWare. Understanding memory resource management in vmware esx 4.1. VMware vSphere 4.1 Documentation, 2010.

[104] Voorsluys, W., Broberg, J., and Buyya, R. Introduction to cloud computing. In Cloud Computing: Principles and Paradigms (2011), pp. 1–44.

[105] Wang, W., Li, Z., Owens, R., and Bhargava, B. Secure and efficient access to outsourced data. In Proceedings of the 2009 ACM workshop on Cloud computing security (2009), CCSW '09, pp. 55–66.

[106] Woltman, G. Prime95. a component of Great Internet Mersenne Prime Search (GIMPS), http://www.mersenne.org/, 2009.

[107] Xie, M., Wang, H., Yin, J., and Meng, X. Integrity auditing of outsourced data. In Proceedings of the international conference on Very large data bases (2007), pp. 782–793.

[108] Zarandioon, S., Yao, D., and Ganapathy, V. Omos: A framework for secure communication in mashup applications. In Annual Computer Security Applications Conference (ACSAC) (2008), pp. 355–364.

[109] Zarandioon, S., Yao, D., and Ganapathy, V. Privacy-aware identity management for client-side mashup applications. In ACM workshop on Digital identity management (2009), pp. 21–30.

[110] Zhang, X., Huo, Z., Ma, J., and Meng, D. Exploiting data deduplication to accelerate live virtual machine migration. In IEEE International Conference on Cluster Computing (2010), pp. 88–96.

[111] Zhang, X., Schiffman, J., Gibbs, S., Kunjithapatham, A., and Jeong, S. Securing elastic applications on mobile devices for cloud computing. In ACM CCSW (2009), pp. 127–134.

[112] Zhang, Y., Juels, A., Oprea, A., and Reiter, M. Homealone: Co-residency detection in the cloud via side-channel analysis. In IEEE Symposium on Security and Privacy (SP) (2011), pp. 313–328.

[113] Zhang, Y., Juels, A., Reiter, M. K., and Ristenpart, T. Cross-vm side channels and their use to extract private keys. In ACM CCS (2012), pp. 305–316.

[114] Zhong, S. A practical key management scheme for access control in a user hierarchy. Computers & Security 21, 8 (2002), 750–759.